User Manual

for MPC574XG CAN Driver

Document Number: UM35CANASR4.2 Rev0002 RTM 1.0.0
Rev. 1.0

Contents

Section number **Title** **Page**

Chapter 1 Revision History

Chapter 2 Introduction

Chapter 3 Driver

Section number Title Page

Section number Title Page

Section number Title Page

Section number Title Page

Section number **Title** **Page**

Chapter 4
The Configuration of Can Bit Timing

Chapter 5
Interrupts Implementation

Chapter 1
Revision History

Table 1-1. Revision History

Revision	Date	Author	Description
1.0	17/02/2017	Hieu Tran	First version for MPC574XG ASR 4.2 RTM 1.0.0 release

Chapter 2
Introduction

This User Manual describes NXP Semiconductor AUTOSAR Controller Area Network (CAN) for MPC5748G.

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 intented to be used with the following microcontroller devices of NXP Semiconductor .

Table 2-1. MPC574XG Derivatives

NXP Semiconductor	MPC5748G_LQFP176, MPC5748G_MAPBGA256, MPC5748G_MAPBGA324, MPC5747G_LQFP176, MPC5747G_MAPBGA256, MPC5747G_MAPBGA324, MPC5746G_LQFP176, MPC5746G_MAPBGA256, MPC5746G_MAPBGA324, MPC5748C_LQFP176, MPC5748C_MAPBGA256, MPC5748C_MAPBGA324, MPC5747C_LQFP176, MPC5747C_MAPBGA256, MPC5747C_MAPBGA324, MPC5746C_LQFP176, MPC5746C_MAPBGA256, MPC5746C_MAPBGA324, MPC5746C_MAPBGA100, MPC5745C_LQFP176, MPC5745C_MAPBGA256, MPC5745C_MAPBGA100, MPC5744C_LQFP176, MPC5744C_MAPBGA256,

Table 2-1. MPC574XG Derivatives

	MPC5744C_MAPBGA100, MPC5746B_LQFP176, MPC5746B_MAPBGA256, MPC5746B_MAPBGA100, MPC5744B_LQFP176, MPC5744B_MAPBGA256, MPC5744B_MAPBGA100, MPC5745B_LQFP176, MPC5745B_MAPBGA256, MPC5745B_MAPBGA100

All of the above microcontroller devices are collectively named as MPC574XG .

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
N/A	Not Applicable
MCU	Micro Controller Unit

2.5 Reference List

Table 2-3. Reference List

#	Title	Version
1	AUTOSAR 4.2 Rev0002CAN Driver Software Specification Document.	V4.0.0
2	MPC5748G Reference Manual	Rev. 5, 12/2016
3	MPC5746C Reference Manual	Rev. 4, 12/2016
4	MPC5748G_1N81M_Rev.2 (official document) (1N81M)	Jun-16
5	MPC5748G_1N81M_0N78S_Comparison_Summary_v2_0 (internal document) (1N81M, 0N78S)	31.10.2016
6	MPC5746C_1N06M_Rev.4 (official document) (1N06M)	Jul-16
7	MPC5746C_cut1.1_cut2.0_cut2.1_comparison_v0 (internal document) (1N06M, 0N84S, 1N84S)	14-Sep-16
8	C3M_cut2.1_new_errata_20170113 (internal document) (1N84S)	13-Jan-17

Chapter 3
Driver

3.1 Requirements

Requirements for this driver are detailed in the AUTOSAR 4.2 Rev0002CAN Driver Software Specification document (See Table Reference List).

3.2 Driver Design Sumary

The MPC574XG contains up to 8 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:

- 96 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/none-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 Calling Driver APIs

1. void Can_GetVersionInfo (Std_VersionInfoType* versioninfo)

This function returns the version information of this module.

versioninfo - Pointer where to store the version information of this module.

```
typedef struct {
    uint16 vendorID;
    uint16 moduleID;
    uint8 sw_major_version;
    uint8 sw_minor_version;
    uint8 sw_patch_version;
} Std_VersionInfoType;
```

Sample code:

```
/* main_app.c */
....................
Std_VersionInfoType version;
uint16  vendor_id;

Can_Init( Can_Cfg1);
#if (CAN_VERSION_INFO_API == STD_ON)
    Can_GetVersionInfo( &version);
    vendor_id = version.vendorID;
#endif /* (CAN_VERSION_INFO_API == STD_ON) */
....................
```

Note

This API can be used only if CAN_VERSION_INFO_API is set to STD_ON.

2. void Can_Init (const Can_ConfigType* Config)

This function initializes the module.

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

Can_Cfg.c file will contains the structure type:

```
....................
CONST(CanStatic_ConfigType, CAN_CONST) CanStatic_ConfigSet = {
.....
};
....................
```

Can_PbCfg.h file will contain the extern declaration of "Can_ConfigType" structure.

```
....................
 #define CAN_INIT_CONFIG_PB_DEFINES_VS_0 \
    extern CONST(Can_ConfigType, CAN_CONST) CanConfigSet_0_VS_0;
....................
```

Can_PBcfg.c file will contains the structure type:

```
....................
CONST(Can_ConfigType, CAN_CONST) CanConfigSet_0_VS_0 = {
    .....
};
....................
```

Can.h export driver configuration:

```
#ifdef CAN_PRECOMPILE_SUPPORT
    /* Export Driver configuration */
    CAN_INIT_CONFIG_PC_DEFINES
#else
    /* Export Driver configuration */
    CAN_INIT_CONFIG_PB_DEFINES
#endif
```

Sample code:

```
/* main_app.c */
....................
#ifdef CAN_PRECOMPILE_SUPPORT
    CONST(Can_ConfigType, CAN_CONST)    *Can_Cfg1 = &CanConfigSet_0_VS_0;
#else
    CONST(Can_ConfigType, CAN_CONST)    *Can_Cfg1 = &CanConfigSet_0_VS_0;
    CONST(Can_ConfigType, CAN_CONST)    *Can_Cfg2 = &CanConfigSet_0_VS_1;
    CONST(Can_ConfigType, CAN_CONST)    *Can_Cfg3 = &CanConfigSet_0_VS_2;
#endif
```

```
Can_Init( Can_Cfg1);
....................
```

3. Std_ReturnType Can_ChangeBaudrate(uint8 Controller, const uint16 Baudrate)

This function change the baudrate for the configured controller.

Controller - Can controller to be initialized - based on configuration order list (CanControllerId).

Baudrate - Baudrate for which the controller shall be initialised.

Can_Cfg.h contains the controller indexes:

```
....................
#define CanA0 0U  /* Default configuration for FlexCAN_A */
....................
```

Note

This API can be used only if CAN_CHANGE_BAUDRATE_SUPPORT is set to STD_ON.

4. Std_ReturnType Can_CheckBaudrate(uint8 Controller, const uint16 Baudrate)

This function check the controller's configured baudrate.

Controller - Can controller to be initialized - based on configuration order list (CanControllerId).CAN Controller to check for the support of a certain baudrate

Baudrate - Baudrate to check in kbps

Note

This API can be used only if CAN_CHANGE_BAUDRATE_SUPPORT is set to STD_ON.

5. Can_ReturnType Can_SetControllerMode (uint8 Controller, Can_StateTransitionType Transition)

This function performs software triggered state transitions of the CAN controller State machine.

Controller - CAN controller for which the status shall be changed

Transition - CAN state transition: CAN_STOP / CAN_T_START / CAN_T_SLEEP / CAN_T_WAKEUP

Sample code:

```
/* main_app.c */
....................
Can_ReturnType ret_val = CAN_NOT_OK;

Can_Init( Can_Cfg1);
ret_val = Can_SetControllerMode( CanA0, CAN_T_START);
....................
```

6. void Can_DisableControllerInterrupts (uint8 Controller)

This function disables all interrupts for this CAN controller.

Controller - CAN controller for which interrupts shall be disabled.

Sample code:

```
/* main_app.c */
....................
Can_Init( Can_Cfg1);
Can_DisableControllerInterrupts( CanA0);
....................
```

7. void Can_EnableControllerInterrupts (uint8 Controller)

This function enables all allowed interrupts.

Controller - CAN controller for which interrupts shall be re-enabled.

```
/* main_app.c */
....................
Can_Init( Can_Cfg1);
Can_EnableControllerInterrupts( CanA0);
....................
```

8. Can_ReturnType Can_Write (Can_HwHandleType Hth, const Can_PduType* PduInfo)

This function transmit a message on the bus.

Hth - information which HW-transmit handle shall be used for transmit. Implicitly this is also the information about the controller to use because the Hth numbers are unique inside one hardware unit.

PduInfo - Pointer to SDU user memory, DLC and Identifier.

Can_Cfg.h will contains the MB indexes:

```
#define CTRL0_MB0 0U /* RECEIVE object of Can Controller ID = 0 */
#define CTRL0_MB1 1U /* RECEIVE object of Can Controller ID = 0 */
#define CTRL0_MB2 2U /* RECEIVE object of Can Controller ID = 0 */
#define CTRL0_MB3 3U /* TRANSMIT object of Can Controller ID = 0 */
```

```
#define CTRL0_MB4 4U /* TRANSMIT object of Can Controller ID = 0 */
#define CTRL0_MB5 5U /* TRANSMIT object of Can Controller ID = 0 */
#define CTRL1_MB0 6U /* RECEIVE object of Can Controller ID = 1 */
#define CTRL1_MB1 7U /* TRANSMIT object of Can Controller ID = 1 */
```

Sample code:

```
/* main_app.c */
....................
Can_ReturnType ret_val = CAN_NOT_OK;
Can_PduType    CanMessage;
VAR(uint8, CAN_VAR) data[8U] = {0x77, 0x77, 0x77, 0x77, 0x77, 0x77, 0x77, 0x77};

Can_Init( Can_Cfg1);
CanMessage.length = 8U;
CanMessage.sdu = data;
CanMessage.id = 0x1U;

ret_val = Can_Write( CTRL0_MB3, &CanMessage);
....................
```

9. void Can_MainFunction_Write ()

This function performs the polling of TX confirmation and TX cancellation confirmation when CAN_TX_PROCESSING is set to POLLING.

Notice that:

```
/* Can.h */
#if (CAN_TXPOLL_SUPPORTED == STD_ON)
    extern FUNC (void, CAN_CODE) Can_MainFunction_Write( void );
#else
    #define Can_MainFunction_Write()
#endif
```

Sample code:

```
/* main_app.c */
....................
ret_val = Can_Write( CTRL0_MB3, &CanMessage);
Can_MainFunction_Write();
....................
```

Also, if different polling periods are configured, another polling function is enabled "Can_MainFunction_Write_0()". This function will poll only one Hardware Object, specified by CanMainFunctionWritePeriodRef

```
/* Can.h */
#ifdef CAN_MAINFUNCTION_PERIOD_WRITE_0
    extern FUNC (void, CAN_CODE) Can_MainFunction_Write_0( void);
#endif /* CAN_MAINFUNCTION_PERIOD_WRITE_0 */
```

Note

This API can be used only if CAN_TXPOLL_SUPPORTED is set to STD_ON.

10. void Can_MainFunction_Read ()

This function performs the polling of RX indications when CAN_RX_PROCESSING is set to POLLING.

Notice that:

```
/* Can.h */
#if (CAN_RXPOLL_SUPPORTED == STD_ON)
    extern FUNC (void, CAN_CODE) Can_MainFunction_Read( void );
#else
    #define Can_MainFunction_Read()
#endif
```

Also, if different polling periods are configured, another polling function is enabled "Can_MainFunction_Read_0()". This function will poll only one Hardware Object, specified by CanMainFunctionReadPeriodRef

```
/* Can.h */
#ifdef CAN_MAINFUNCTION_PERIOD_READ_0
    extern FUNC (void, CAN_CODE) Can_MainFunction_Read_0( void);
#endif /* CAN_MAINFUNCTION_PERIOD_READ_0 */
```

Note

This API can be used only if CAN_RXPOLL_SUPPORTED is set to STD_ON.

11. void Can_MainFunction_BusOff ()

This function performs the polling of bus-off events that are configured statically as 'to be polled'.

Notice that:

```
/* Can.h */
#if (CAN_BUSOFFPOLL_SUPPORTED == STD_ON)
    extern FUNC (void, CAN_CODE) Can_MainFunction_BusOff( void );
#else
    #define Can_MainFunction_BusOff()
#endif
```

Note

This API can be used only if CAN_BUSOFFPOLL_SUPPORTED is set to STD_ON.

12. void Can_MainFunction_Wakeup ()

This function performs the polling of wake-up events that are configured statically as 'to be polled'.

Notice that:

```
/* Can.h */
#if (CAN_WAKEUPPOLL_SUPPORTED == STD_ON)
    extern FUNC (void, CAN_CODE) Can_MainFunction_Wakeup( void );
#else
    #define Can_MainFunction_Wakeup()
#endif
```

Note

This API can be used only if CAN_WAKEUPPOLL_SUPPORTED and CAN_WAKEUP_SUPPORT are set to STD_ON.

13. void Can_MainFunction_Mode ()

This function performs the polling of CAN controller mode transitions..

Notice that:

```
 /* Can.h */
extern FUNC (void, CAN_CODE) Can_MainFunction_Mode( void);
```

14. void Can_CheckWakeup (uint8 controller)

This function checks if a wakeup has occurred for the given controller.

Sample code:

```
/* main_app.c */
....................
Std_ReturnType std_ret_val = E_NOT_OK;

Can_Init( Can_Cfg1);
.....
std_ret_val = Can_CheckWakeup( CanA0);
....................
```

Note

This API can be used only if CAN_WAKEUP_SUPPORT is set to STD_ON.

15. void Can_AbortMb (Can_HwHandleType Hth)

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

Sample code:

```
/* main_app.c */
....................
Can_PduType    CanMessage;
Can_ReturnType ret_val = CAN_NOT_OK;

Can_Init( Can_Cfg1);
ret_val = Can_SetControllerMode( CanA0, CAN_T_START);
ret_val = Can_Write( CTRL0_MB3, &CanMessage);
Can_AbortMb ( Hth);
....................
```

Note

This API can be used only if CAN_API_ENABLE_ABORT_MB is set to STD_ON.

Note

When Tx processing is due using Polling mode, any call of Can_AbortMb shall be followed by the call of Can_MainFunctionWrite in order to finalize the abort procedure.

16. Std_ReturnType Can_SetClockMode(uint8 can_controller, Can_ClockMode can_clk_mode)

This function is configuring Can controller to run on the same baudrate, but having a different MCU source clock..

Sample code:

```
       /* main_app.c */
Std_ReturnType   can_return = E_NOT_OK;
Can_ReturnType  ret_val = CAN_NOT_OK;
       ....................
       Can_Init( Can_Cfg1);
can_return = Can_SetClockMode( controller, CAN_NORMAL);
.............
can_return = Can_SetClockMode( controller, CAN_ALTERNATE);
ret_val = Can_SetControllerMode( controller, CAN_T_START);
       ....................
```

Note

This API can be used only if CAN_DUAL_CLOCK_MODE is set to STD_ON.

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 somethimg 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
CAN011	N/S	The CAN driver directly uses the buffer of the upper layer. It is the responsibility of the upper layer to keep the buffer consistent. The function CanIf_Transmit copies the L-SDU either directly in the CAN Hardware or buffers it if CAN Hardware transmit resources are presently not available. The L-SDU source buffer (provided by source layer, e.g. COM) can be written with the next value as soon as CanIf_Transmit returns. The source layer (i.e. COM) is responsible that the L-SDU buffer is not overwritten until the function CanIf_Transmit returned. SDU must be protected by the layer that calls CanIf_Transmit.	Can module is not responsible for keeping the buffers consistent.Upper layer should provide it.
CAN323_Conf	N/I	CAN_HANDLE_TYPE Specifies the type (Full-CAN or Basic-CAN) of a hardware object.	Controller doesn't provide any bit field to differentiate BASIC-CAN and FULL-CAN.
CAN242	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 µC and CAN Hardware Unit has only impact on implementation and not on the API.	No off-chip controller is used.
CAN110	N/S	There is no requirement regarding the execution order of the CAN main processing functions.	Application Code Requirement.
CAN240	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.
CAN077	N/S	For CAN Hardware Units of different type, different Can modules shall be implemented.	Current platforms have only one type of hardware unit.
CAN397	N/S	The Can module shall include the header file Os.h file.	No services of other drivers are used by Can driver.
CAN389_Conf	N/S	Specifies the CAN controller base address.	The possibility for user to input the base address of the controller is not supported

Table 3-2. CAN Deviations Table

Requirement	Status	Description	Notes
			by the Can Driver. The base address is given in the Base module.

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 Function Reference

Functions of all functions supported by the driver are as per AUTOSAR CAN Driver software specification Version 4.2 Rev0002 .

3.5.1 Function Can_Init

Initialize the CAN driver. SID = 0x00.

Prototype: `void Can_Init(const Can_ConfigType *Config);`

Table 3-3. 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.

3.5.2 Function Can_GetVersionInfo

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

Prototype: `void Can_GetVersionInfo(Std_VersionInfoType *versioninfo);`

Table 3-4. 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.

3.5.3 Function Can_SetControllerMode

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

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

Table 3-5. 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 3-6. 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 enebaled.

3.5.4 Function Can_DisableControllerInterrupts

Disable INTs. SID = 0x04.

Prototype: `void Can_DisableControllerInterrupts(uint8 Controller);`

Table 3-7. 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 initalzied 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

3.5.5 Function Can_EnableControllerInterrupts

Enable INTs. SID = 0x05.

Prototype: `void Can_EnableControllerInterrupts(uint8 Controller);`

Table 3-8. 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 initalzied 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.

3.5.6 Function Can_Write

Transmit information on CAN bus. SID = 0x06.

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

Table 3-9. 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 3-10. Can_Write Returns

Value	Description
CAN_OK	Write command has been accepted.
CAN_NOT_OK	Development error occured.
CAN_BUSY	No of TX hardware buffer available or preemtive 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.

3.5.7 Function Can_CheckWakeup

Process check of WakeUp condition. SID = 0x0B.

Prototype: `Std_ReturnType Can_CheckWakeup(uint8 controller);`

Table 3-11. Can_CheckWakeup Arguments

Type	Name	Direction	Description
`uint8`	controller	**input**	Can controller ID - based on configuration order list (CanControllerId).

Return: Std_ReturnType Result of the wakeup verification.

Table 3-12. Can_CheckWakeup Returns

Value	Description
E_OK	Wakeup was detected for the given controller.
E_NOT_OK	No wakeup was detected for the given controller.

This service shall evaluate the WakeupSource parameter to get the information, which dedicate wakeup source needs to be checked, either a CAN transceiver or controller device. This routine is called by:

- CanIf or an upper layer according to Autosar requirements.

pre: Driver must be initialized.

post: Return the Wakeup event occurence.

Note

> The CAN Controller of this platform doesn't support wakeup, the appearance of this function is as required by AUTOSAR

3.5.8 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.

3.5.9 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 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.

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

- CanIf or an upper layer according to Autosar requirements.

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.

3.5.10 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 RX indications when CAN_RX_PROCESSING is set to POLLING. This routine is called by:

- CanIf or an upper layer according to Autosar requirements.

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

pre: Driver must be initialized.

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

- CanIf or an upper layer according to Autosar requirements.

pre: Driver must be initialized.

post: Handle the Busoff event.

3.5.11 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.

3.5.12 Function Can_AbortMb

Process a message buffer abort.

Prototype: `void Can_AbortMb(Can_HwHandleType Hth);`

Table 3-13. 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.

3.5.13 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 3-14. 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 3-15. 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.

3.5.14 Function Can_ChangeBaudrate

Initialize the CAN controllers. SID = 0x0d.

Prototype: `Std_ReturnType Can_ChangeBaudrate( uint8 Controller, const uint16 Baudrate )`

Table 3-16. Can_ChangeBaudrate Arguments

Type	Name	Direction	Description
`uint8`	Controller	**input**	CAN Controller, whose baudrate shall be changed
const uint16	Baudrate	**input**	Requested baudrate in kbps

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-initalization 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.

3.5.15 Function Can_CheckBaudrate

Initialize the CAN controllers. SID = 0x0e.

Prototype: Std_ReturnType Can_CheckBaudrate(uint8 Controller, const uint16 Baudrate)

Table 3-17. Can_CheckBaudrate Arguments

Type	Name	Direction	Description
`uint8`	Controller	**input**	CAN Controller to check for the support of a certain baudrate
const uint16	Baudrate	**input**	Baudrate to check in kbps

Return: Std_ReturnType

The service Can_CheckBaudrate(Controller, Baudrate) shall be called by CanIf_CheckBaudrate() for the requested CAN controller.

- CanIf or an upper layer according to Autosar requirements.

Note

If Can supports changing of the baudrate and thus this service, shall be configurable via CAN_CHANGE_BAUDRATE_API.

3.5.16 Function Can_SetBaudrate

Initialize the CAN controllers. SID = 0x0f.

Prototype: `Std_ReturnType Can_SetBaudrate( uint8 Controller, uint16 BaudRateConfigID )`

Table 3-18. 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-initalization 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.

3.5.17 Function Can_SetIcomConfiguration

This function is API which support the Pretended Networking feature. After this function is called, it will configure for controller with information in the configurationIDs (corresponding with ID valid).

Prototype: `Std_ReturnType Can_SetIcomConfiguration( uint8 Controller, IcomConfigIdType ConfigurationId);`

Table 3-19. Can_SetIcomConfiguration Arguments

Type	Name	Direction	Description
`uint8`	Controller	**input**	CAN controller for which the status shall be changed
`IcomConfigIdType`	ConfigurationId	**input**	Requested Configuration.

Return: Std_ReturnType Result of the change request.

Table 3-20. Can_SetIcomConfiguration Returns

Value	Description
E_OK	CAN driver succeeded in setting a configuration with a valid Configuration id.
E_NOT_OK	CAN driver failed to set a configuration with a valid Configuration id.

3.6 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).

3.7 Structs Reference

Data structures supported by the driver are as per AUTOSAR CAN Driver software specification Version 4.2 Rev0002 .

3.7.1 Structure Can_ConfigType and CanStatic_ConfigType

Top Level structure containing all Driver configuration.

A pointer to this structure is transmitted to `Can_Init()` to initalize 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_ControlerDescriptorType *ControlerDescriptors,
            Can_RxFiFoTableIdConfigType *pRxFiFoTableIdConfig,

            const Can_RxFilterTableType pRxFilterTableConfig,
            const Can_HwHandleType uCanFirstHTHIndex,
            const uint8 u8ControllerIdMapping[],
            const Can_ObjType eObjectTypeMapping[]
        } Can_ConfigType;
```

Table 3-21. 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.
ControlerDescriptors	Pointer to the first CAN Controller description.
pRxFiFoTableIdConfig	Pointer to the Table IDs for the RxFifo.
pRxFilterTableConfig	The Rx Filter Table.
uCanFirstHTHIndex	The index of the first HTH configured.
u8ControllerIdMapping[]	An array stores Controller ID mapping.
eObjectTypeMapping[]	An array stores Can Object Type mapping.

```
typedef struct
    {
        uint8 u8ControllersConfigured,
        uint32 CanStatic_ControlerDescriptorType,
        uint32 u32CanMaxObjectId,
        uint8 u8NumCanIcomConfigs,
        Can_IcomConfigsType pCanIcomConfigs
    } CanStatic_ConfigType;
```

Table 3-22. Structure CanStatic_ConfigType member description

Member	Description
u8ControllersConfigured	Number of Can Controllers configured in Tresos plugin.
StaticControlerDescriptors	Pointer to the first FlexCAN Controller description.
u32CanMaxObjectId	Maximum Hardware Object IDs configured.
u8NumCanIcomConfigs	Number of Can ICOM configured.
pCanIcomConfigs	Pointer to the first ICOM configuration.

3.7.2 Structure Can_ControlerDescriptorType and CanStatic_ControlerDescriptorType

Structures 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

```
typedef struct
    {
        const uint8 u8MaxMBCount,
        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 u32MBBlockSize,
        const uint32 u32Options,
    } Can_ControlerDescriptorType;
```

Table 3-23. Structure Can_ControlerDescriptorType member description

Member	Description
u8MaxMBCount	Maximum number of MB.
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.
u32MBBlockSize	This parameter is used to configure for three MBDSR fields in CAN_FDCTRL register.
u32Options	BusOff Sw Recovery, RXFifo En, IDAM Type,.

```
typedef struct
    {
        const uint8 u8ControllerOffset,
        const CanStatic_ControllerBaudrateConfigType *pStaticControllerBaudrateConfigsPtr,
        const Can_PCallBackType Can_ErrorNotification,
        const uint8 u8NumberOfMB,
        const boolean bPnSupported,
        const uint32 u32Options,
    } CanStatic_ControlerDescriptorType;
```

Table 3-24. Structure Can_ControlerDescriptorType 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.
bPnSupported	This is used to determine whether the Pretended Networking mode is supported or not.
u32Options	Event Trigger Mode TxProcessing/RxProcessing/BusoffProcessing/ WakeuProcessing: Polling vs Interrupt mode.

3.7.3 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 3-25. Structure Can_ControllerBaudrateConfigType member description

Member	Description
u32ControlRegister	Content of the Control Register (CTRL) fields: PRESDIV, 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: PRESDIV, 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 3-26. Structure CanStatic_ControllerBaudrateConfigType member description

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

3.7.4 Structure Can_ControllerStatusType

Records the status of a CAN Controller during runtime.

Note

This structure is not configured by Tresos.

Declaration

```
typedef struct
        {
            uint32 u32TxGuard[3],
            uint32 u32MBInterruptMask[3],
            PduIdType u32TxPduId[96],
            sint8 s8IntDisableLevel,
            Can_HwHandleType u32CancelMBIndex,
            uint8 u8FirstTxMBIndex,
            uint8 eInterruptMode,
            Can_ControllerStateType ControllerState,
            uint32 u16MBMapping[CAN_MAXMB_SUPPORTED],
            Can_ClockModeType CanClockMode,
            uint8 u8CurrentBaudRateIndex,
            uint32 u32TxCancelFlag
        } Can_ControllerStatusType;
```

Table 3-27. Structure Can_ControllerStatusType member description

Member	Description
u32TxGuard	Guard bits for EXCLUSIVE ACCESS to Tx MBs.
u32MBInterruptMask	Pre-calculated MB INT masks (used in Can_EnableControllerInterrupts).
u32TxPduId	Storage space for PDU_ID (supplied in call to Can_Write and needed after Tx in CanIf_TxConfirmation).
s8IntDisableLevel	Storage space for Can_DisableControllerInterrupts nesting level.
u32CancelMBIndex	Index of MB buffer being cancelled.
u8FirstTxMBIndex	Index of the first MB used for Tx for a specific controller. This value is relative to 0 (which is first MB).
eInterruptMode	Global interrupt autorization state.
ControllerState	FlexCAN controller power state.
u16MBMapping	Map for every MB the HOH assigned according to configuration.
CanClockMode	Define the clock mode (normal clock mode or alternate clock mode) is used in the dual clock mode.
u8CurrentBaudRateIndex	Current controller baudrate.
u32TxCancelFlag	Guard bits for EXCLUSIVE ACCESS to Tx MBs.

3.7.5 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 3-28. 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).

3.7.6 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 u32CanMainFuncRWPeriodRef,
        const uint16 u16MBOffsetAddr,
        const uint8 u8MBPayloadLength,
        const uint8 u8HWMBIndex,
        const boolean CanTriggerTransmitEnable
    } Can_MBConfigObjectType;
```

Table 3-29. Structure Can_MBConfigObjectType member description

Member	Description
uIdMaskIndex	Index into array of Can_FilterMaskType values (uint8/uint16), Current MB and the coresponding 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 Obiect ID.
u8FdPaddingValue	Padding value for MBs with the number of data bytes > 8. This parameter is used in FD mode
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.

3.7.7 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 3-30. 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).

Table continues on the next page...

Table 3-30. Structure Can_PduType member description (continued)

Member	Description
sdu	CAN L-SDU = Link Layer Service Data Unit. Data that is transported inside the L-PDU.

3.7.8 Structure Can_RxFiFoTableIdConfigType

Rx Fifo Table IDs and Filter Masks.

Declaration

```
typedef struct
    {
        const uint32 u32TableId,
        const uint32 u32TableFilterMask
    } Can_RxFiFoTableIdConfigType;
```

Table 3-31. Structure Can_RxFiFoTableIdConfigType member description

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

3.7.9 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 3-32. 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.

3.7.10 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 3-33. 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.

3.7.11 Structure Can_IcomRxMessageSignalType

This container contains the configuration parameters for the wakeup causes for matching signals.

Declaration

```
typedef struct
        {
            const uint64 CanIcomSignalMask,
            const Can_IcomSignalOperationType CanIcomSignalOperation,
            const uint64 CanIcomSignalValue,
            const uint8 DLCLowValue,
            const uint8 DLCHighValue,
            const uint32 CanIcomSignalRef,
        } Can_IcomRxMessageSignalType;
```

Table 3-34. Structure Can_IcomRxMessageSignalType member description

Member	Description
CanIcomSignalMask	This parameter shall be used to mask a signal in the payload of a CAN message.
CanIcomSignalOperation	This parameter defines the operation, which shall be used to verify the signal value creates a wakeup condition.
CanIcomSignalValue	This parameter shall be used to define a signal value which shall be compared (CanIcomSignalOperation) with the masked CanIcomSignalMask value of the received signal (CanIcomSignalRef).
DLCLowValue	Records the lower limit for the number of data bytes considered valid for payload comparison. It is used as part of payload reception filter (This parameter is not defined by AUTOSAR requirement).
DLCHighValue	Records the upper limit for the number of data bytes considered valid for payload comparison. It is used as part of payload reception filter (This parameter is not defined by AUTOSAR requirement).
CanIcomSignalRef	This parameter defines a reference to the signal which shall be checked additional to the message id (CanIcomMessageId).This reference is used for documentation to define which ComSignal originates this filter setting. All signals being referred by this reference shall point to the same PDU.

3.7.12 Structure Can_IcomRxMessageConfigsType

This container contains the configuration parameters for the wakeup causes for matching received messages.

Declaration

```
typedef struct
    {
        const uint32 CanIcomCounterValue,
        const uint32 CanIcomMessageId,
        const uint32 CanIcomMessageIdMask,
        const uint32 CanIcomMissingMessageTimerValue,
        const Can_IcomIdOperationType CanIcomIdOperation,
        const uint8 u8NumCanIcomRxMessageSignal,
        const Can_IcomRxMessageSignalType *pCanIcomRxMessageSignalConfigs,
    } Can_IcomRxMessageConfigsType;
```

Table 3-35. Structure Can_IcomRxMessageConfigsType member description

Member	Description
CanIcomCounterValue	This parameter defines that the MCU shall wake if the message with the ID is received n times on the communication channel.
CanIcomMessageId	This parameter defines the message ID the wakeup causes of this CanIcomRxMessage are configured for. In addition a mask (CanIcomMessageIdMask) can be defined, in that case it is possible to define a range of rx messages, which can create a wakeup condition.
CanIcomMessageIdMask	Describes a mask for filtering of CAN identifiers. The CAN identifiers of incoming messages are masked with this CanIcomMessageIdMask. If the masked identifier matches the masked value of CanIcomMessageId, it can create a wakeup condition for this CanIcomRxMessage. Bits holding a 0 mean don't care, i.e. do not compare the message's identifier in the respective bit position. The mask shall be build by filling with leading 0.
CanIcomMissingMessageTimerValue	This parameter defines that the MCU shall wake if the message with the ID is not received for a specific time in s on the communication channel.
CanIcomIdOperation	Records ID filter type in the Pretended Networking mode.
u8NumCanIcomRxMessageSignal	Records the number of the configured CanIcomRxMessageSignal.
pCanIcomRxMessageSignalConfigs	Points to the CanIcomRxMessageSignalConfigs.

3.7.13 Structure Can_IcomConfigsType

This container contains the configuration parameters of the ICOM Configuration.

Declaration

```
typedef struct
    {
        const uint8 u8CanIcomConfigId,
        const boolean CanIcomWakeOnBusOff,
        const uint8 u8NumberCanIcomRxMessage,
        const Can_IcomRxMessageConfigsType pCanIcomRxMessageConfigs,
    } Can_IcomConfigsType;
```

Table 3-36. Structure Can_IcomConfigsType member description

Member	Description
u8CanIcomConfigId	This parameter identifies the ID of the ICOM configuration.
CanIcomWakeOnBusOff	This parameter defines that the MCU shall wake if the bus off is detected or not.
u8NumberCanIcomRxMessage	Records the number of the configured wake up Rx messages.
pCanIcomRxMessageConfigs	Pointer points to the container which records the information of the configured wake up Rx messages.

3.8 Define Reference

Constants supported by the driver are as per AUTOSAR CAN Driver software specification Version 4.2 Rev0002 .

3.8.1 Define CAN_API_ENABLE_ABORT_MB

Support for Special MB Abort API.

Definition:`#define CAN_API_ENABLE_ABORT_MB (STD_ON)`

3.8.2 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)`

3.8.3 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)`

3.8.4 Define CAN_CHANGE_BAUDRATE_API

This macro switches the Can_ChangeBaudrate API and Can_CheckBaudRate API ON or OFF.

Definition:`#define CAN_CHANGE_BAUDRATE_API (STD_ON)`

3.8.5 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)`

3.8.6 Define CAN_DEV_ERROR_DETECT

Switches the Development Error Detection and Notification ON or OFF.

Definition:`#define CAN_DEV_ERROR_DETECT (STD_ON)`

3.8.7 Define CAN_DUAL_CLOCK_MODE

Enable Non-Autosar API for Dual-Clock support.

Definition:`#define CAN_DUAL_CLOCK_MODE (STD_OFF)`

3.8.8 Define CAN_E_DEFAULT

Development errors.

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

Detail:

```
This feature is reserved for future
```

3.8.9 Define CAN_E_DATALOST

Development errors.

Definition:`#define CAN_E_DATALOST (uint8)0x07U`

Violates: MISRA 2004 Required Rule 16.10, Don't check return type of Det_ReportError 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_DATALOST in case of OVERWRITE or OVERRUN event detection.
```

3.8.10 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.
```

3.8.11 Define CAN_E_PARAM_DLC

Development errors.

Definition:`#define CAN_E_PARAM_DLC (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_DLC if the length is more than 8
byte.
```

Suggestion:

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

3.8.12 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.
```

3.8.13 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.
```

3.8.14 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.
```

3.8.15 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
```

3.8.16 Define CAN_E_PARAM_BAUDRATE

Development errors.

Definition: `#define CAN_E_PARAM_BAUDRATE (uint8)0x09U`

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.
```

3.8.17 Define CAN_ERROR_NOTIFICATION_ENABLE

Error notification enabled/disabled.

Definition: `#define CAN_ERROR_NOTIFICATION_ENABLE (STD_ON)`

3.8.18 Define CAN_EXTENDEDID

Extended identifiers.

Definition: `#define CAN_EXTENDEDID (STD_ON)`

3.8.19 Define CAN_HW_TRANSMIT_CANCELLATION

Support for Transmision Cancellation.

Definition: `#define CAN_HW_TRANSMIT_CANCELLATION (STD_ON)`

3.8.20 Define CAN_ISROPTCODESIZE

Optimization of interrupt service code for size.

Definition:`#define CAN_ISROPTCODESIZE (STD_OFF)`

3.8.21 Define CAN_MAINFUNCTION_MODE_PERIOD

Periods for cyclic call of Main function Mode.

Definition:`#define CAN_MAINFUNCTION_MODE_PERIOD 0U`

3.8.22 Define CAN_MAINFUNCTION_PERIOD_BUSOFF

Periods for cyclic call of Main function.

Definition:`#define CAN_MAINFUNCTION_PERIOD_BUSOFF 0U`

3.8.23 Define CAN_MAINFUNCTION_PERIOD_READ

Periods for cyclic call of Main function.

Definition:`#define CAN_MAINFUNCTION_PERIOD_READ 1U`

3.8.24 Define CAN_MAINFUNCTION_PERIOD_WRITE

Periods for cyclic call of Main function Write.

Definition:`#define CAN_MAINFUNCTION_PERIOD_WRITE 3U`

3.8.25 Define CAN_MAXCTRL_SUPPORTED

Maximum possible controllers per specific derivative.

Definition:`#define CAN_MAXCTRL_SUPPORTED 2U`

3.8.26 Define CAN_MAXMB_SUPPORTED

Maximum possible Message Buffers per controller specific to this platform.

Definition:`#define CAN_MAXMB_SUPPORTED 32U`

3.8.27 Define CAN_MBCOUNTEXTENSION

Extended number of can hardware objects.

Definition:`#define CAN_MBCOUNTEXTENSION (STD_ON)`

3.8.28 Define CAN_MIX_MB_SUPPORT

Platform support mix of controllers with 64 and 32 MBs.

Definition:`#define CAN_MIX_MB_SUPPORT (STD_OFF)`

3.8.29 Define CAN_MULTIPLEXED_TRANSMISSION

Support for Multiplexed Transmision.

Definition:`#define CAN_MULTIPLEXED_TRANSMISSION (STD_ON)`

3.8.30 Define CAN_PRECOMPILE_SUPPORT

Precompile Support On.

Definition:`#define CAN_PRECOMPILE_SUPPORT`

3.8.31 Define CAN_RXFIFO_ENABLE

Support for Rx Fifo.

Definition:`#define CAN_RXFIFO_ENABLE (STD_ON)`

3.8.32 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)`

3.8.33 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)`

3.8.34 Define CAN_SID_ABORT_MB

Service ID (APIs) for Det reporting.

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

3.8.35 Define CAN_SID_CBK_CHECK_WAKEUP

Service ID (APIs) for Det reporting.

Definition:`#define CAN_SID_CBK_CHECK_WAKEUP (uint8)0x0BU`

3.8.36 Define CAN_SID_DISABLE_CONTROLLER_INTERRUPTS

Service ID (APIs) for Det reporting.

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

3.8.37 Define CAN_SID_ENABLE_CONTROLLER_INTERRUPTS

Service ID (APIs) for Det reporting.

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

3.8.38 Define CAN_SID_GET_VERSION_INFO

Service ID (APIs) for Det reporting.

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

3.8.39 Define CAN_SID_INIT

Service ID (APIs) for Det reporting.

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

3.8.40 Define CAN_SID_MAIN_FUNCTION_BUS_OFF

Service ID (APIs) for Det reporting.

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

3.8.41 Define CAN_SID_MAIN_FUNCTION_MODE

Service ID (APIs) for Det reporting.

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

3.8.42 Define CAN_SID_MAIN_FUNCTION_READ

Service ID (APIs) for Det reporting.

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

3.8.43 Define CAN_SID_MAIN_FUNCTION_WAKEUP

Service ID (APIs) for Det reporting.

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

3.8.44 Define CAN_SID_MAIN_FUNCTION_WRITE

Service ID (APIs) for Det reporting.

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

3.8.45 Define CAN_SID_SET_CONTROLLER_MODE

Service ID (APIs) for Det reporting.

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

3.8.46 Define CAN_SID_SETCLOCKMODE

Service ID (APIs) for Det reporting.

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

3.8.47 Define CAN_SID_WRITE

Service ID (APIs) for Det reporting.

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

3.8.48 Define CAN_SID_CHANGE_BAUDRATE

Service ID (APIs) for Det reporting.

Definition:`#define CAN_SID_CHANGE_BAUDRATE (uint8)0x0DU`

3.8.49 Define CAN_SID_CHECK_BAUDRATE

Service ID (APIs) for Det reporting.

Definition:`#define CAN_SID_CHECK_BAUDRATE (uint8)0x0EU`

3.8.50 Define CAN_SID_SET_BAUDRATE

Service ID (APIs) for Det reporting.

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

3.8.51 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)`

3.8.52 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`

3.8.53 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)`

3.8.54 Define CAN_VERSION_INFO_API

Support for version info API.

Definition:`#define CAN_VERSION_INFO_API (STD_ON)`

3.8.55 Define CAN_ENABLE_USER_MODE_SUPPORT

This parameter is enabled only in order to support the write access to some registers are protected in user mode. It may be STD_ON or STD_OFF. The user mode will be supported and used if it is STD_ON

Definition:`#define CAN_ENABLE_USER_MODE_SUPPORT (STD_ON)`

3.9 Enum Reference

Enumeration of all constants supported by the driver are as per AUTOSAR CAN Driver software specification Version 4.2 Rev0002 .

3.9.1 Enumeration Can_ClockModeType

CAN source clock selection used in Can_SetClockMode Non-Autosar API.

Table 3-37. Enumeration Can_ClockModeType Values

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

3.9.2 Enumeration Can_ControllerStateType

States that defines the controllers.

Table 3-38. Enumeration Can_ControllerStateType Values

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

3.9.3 Enumeration Can_ObjType

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

Table 3-39. Enumeration Can_ObjType Values

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

3.9.4 Enumeration Can_ReturnType

CAN Return Types from Functions.

Table 3-40. Enumeration Can_ReturnType Values

Value	Description
CAN_OK = 0U	Operation was ok executed.

Table continues on the next page...

Table 3-40. Enumeration Can_ReturnType Values (continued)

Value	Description
CAN_NOT_OK	Operation was not ok executed.
CAN_BUSY	Operation was rejected because of busy state.

3.9.5 Enumeration Can_StateTransitionType

State transitions that are used by the function CAN_SetControllerMode().

Table 3-41. Enumeration Can_StateTransitionType Values

Value	Description
CAN_T_STOP = 0U	CANIF_CS_STARTED -> CANIF_CS_STOPPED.
CAN_T_START	CANIF_CS_STOPPED -> CANIF_CS_STARTED.
CAN_T_SLEEP	CANIF_CS_STOPPED -> CANIF_CS_SLEEP.
CAN_T_WAKEUP	CANIF_CS_SLEEP -> CANIF_CS_STOPPED.

3.9.6 Enumeration Can_StatusType

CAN Driver status used for checking and preventing double driver intialization. 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 3-42. Enumeration Can_StatusType Values

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

3.9.7 Enumeration CanIdType

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

Table 3-43. Enumeration CanIdType Values

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

3.9.8 Enumeration Can_FdType

Can_FdType.

Used for value received by Tressos interface configuration.

Table 3-44. 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.

3.9.9 Enumeration Can_IcomSignalOperationType

Can_IcomSignalOperationType.

This parameter defines the operation, which shall be used to verify the signal value creates a wakeup condition.

Table 3-45. Enumeration Can_IcomSignalOperationType Values

Value	Description
AND = 0U	The received signal value masked by CanIcomSignalMask has at least one bit set in common with CanIcomSignalValue (binary AND). (Note MPC574XG don't support the "AND" filter type)
EQUAL	The received signal value masked by CanIcomSignalMask is equal to CanIcomSignalValue.

Table continues on the next page...

Table 3-45. Enumeration Can_IcomSignalOperationType Values (continued)

Value	Description
GREATER	The received signal value masked by CanIcomSignalMask is strictly greater than CanIcomSignalValue. Values are interpreted as unsigned integers.
SMALLER	The received signal value masked by CanIcomSignalMask is strictly smaller than CanIcomSignalValue. Values are interpreted as unsigned integers.
XOR	The received signal value masked by CanIcomSignalMask then XORed to CanIcomSignalValue is not null.

3.9.10 Enumeration Can_IcomIdOperationType

Can_IcomIdOperationType.

The ID filter type in the Pretended Networking mode.

Table 3-46. Enumeration Can_IcomIdOperationType Values

Value	Description
EXACTLY	A match with the exact ID value.
GREATER_MINNUM	A match with the minimum range of ID.
SMALLER_MAXNUM	A match with the maximum range of ID.
INSIDE_RANGE	A match inside a range of IDs.

3.10 Types Reference

Types supported by the driver are as per AUTOSAR CAN Driver software specification Version 4.2 Rev0002 .

3.10.1 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.

3.10.2 Typedef Can_IdPtrType

Type: const `uint32` *const

Type for storing pointer to the Identifier Lenght Type.

- used by "Can_ConfigType" structure (pointer to the FilterMasks).

3.10.3 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.

3.10.4 Typedef Can_PCallBackType

Type for pointer to function.

Type: void(*

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

3.10.5 Typedef Can_PtrControlerDescriptorType

Type: const `Can_ControlerDescriptorType` *

3.10.6 Typedef CanStatic_PtrControlerDescriptorType

Type: const `CanStatic_ControlerDescriptorType` *

3.10.7 Typedef Can_PtrMBConfigContainerType

Type: const `Can_MBConfigContainerType` *

3.11 Configuration Parameters

As per the AUTOSAR specification the driver has two types of configurations parameters:**Pre-Compile** parameters and **Post-Build** parameters.

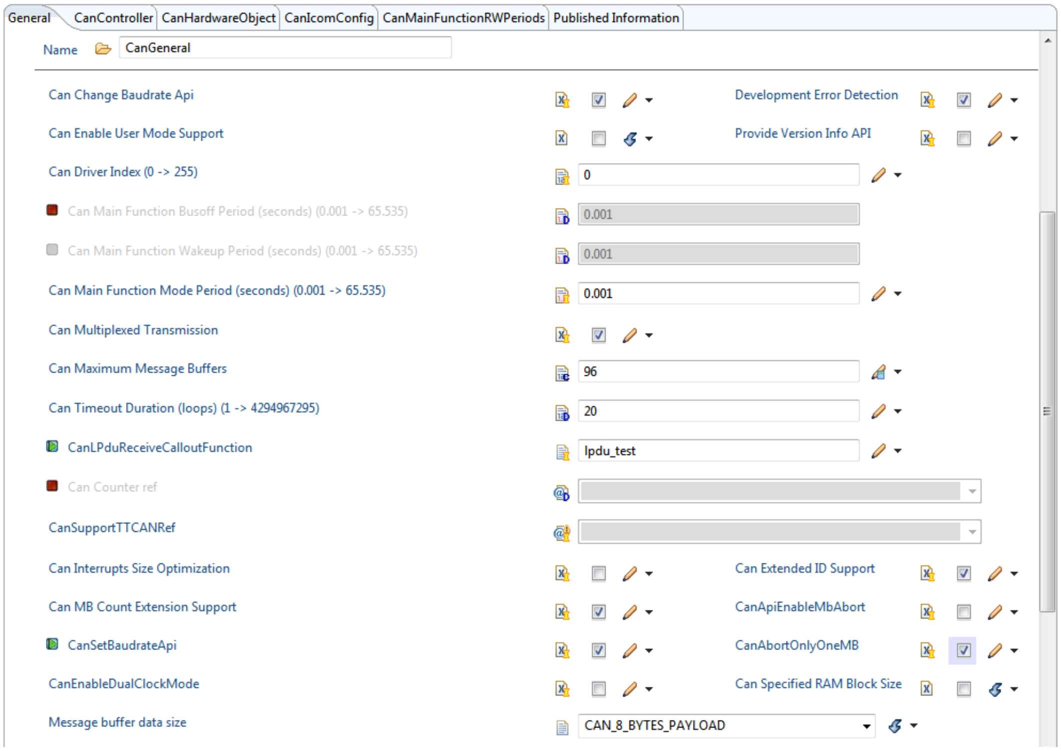
The files to be used for different configuration types are listed below:

1. **Variant PC**: Can_Cfg.c, Can_PBcfg_VS.c, Can_Cfg.h
2. **Variant PB**: Can_Cfg.c, Can_PBcfg_VS.c, Can_Cfg.h
3. **Variant LT**: Not Applicable

The section for **Can_PBcfg_VS.c and Can_Cfg.c** file are needed in linker file to place the post build configuration in desired location.

3.11.1 Can-General Parameters

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



Name: CanGeneral

Can Change Baudrate Api ☑
Development Error Detection ☑
Can Enable User Mode Support □
Provide Version Info API □
Can Driver Index (0 -> 255): 0
Can Main Function Busoff Period (seconds) (0.001 -> 65.535): 0.001
Can Main Function Wakeup Period (seconds) (0.001 -> 65.535): 0.001
Can Main Function Mode Period (seconds) (0.001 -> 65.535): 0.001
Can Multiplexed Transmission ☑
Can Maximum Message Buffers: 96
Can Timeout Duration (loops) (1 -> 4294967295): 20
CanLPduReceiveCalloutFunction: lpdu_test
Can Counter ref:
CanSupportTTCANRef:
Can Interrupts Size Optimization □
Can Extended ID Support ☑
Can MB Count Extension Support ☑
CanApiEnableMbAbort □
CanSetBaudrateApi ☑
CanAbortOnlyOneMB ☑
CanEnableDualClockMode □
Can Specified RAM Block Size □
Message buffer data size: CAN_8_BYTES_PAYLOAD

Figure 3-1. Can General Parameters

3.11.1.1 IMPLEMENTATION_CONFIG_VARIANT

Table 3-47. IMPLEMENTATION_CONFIG_VARIANT

Description	Defines whether pre-compile version is used. Using this option with VariantPostBuild value, Tresos can generate many CanConfigSet variants.
Class	Implementation Specific Parameter
Range	VariantPreCompile, VariantPostBuild
Default	VariantPreCompile
Source File	Can_Cfg.h
Source Representation	#define CAN_PRECOMPILE_SUPPORT
Autosar 4.0 Requirement	NA

Note

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

3.11.1.2 CanDevErrorDetection

Table 3-48. CanDevErrorDetection

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 4.0 Requirement	CAN064_Conf

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).

3.11.1.3 CanChangeBaudrateApi

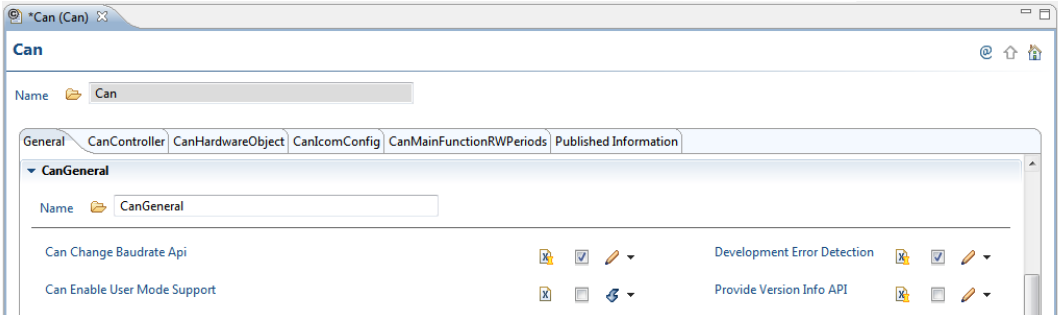
Table 3-49. CanChangeBaudrateApi

Description	Defines whether baudrate information reporting should be included at compile time (STD_ON) or excluded (STD_OFF).
Class	Autosar Parameter
Range	True, False
Default	True

Table continues on the next page...

Table 3-49. CanChangeBaudrateApi (continued)

Source File	Can_Cfg.h
Source Representation	#define CAN_CHANGE_BAUDRATE_API STD_OFF / STD_ON
Autosar 4.0 Requirement	CAN436_Conf



*Can (Can)

Can

Name: Can

CanGeneral

Name: CanGeneral

Can Change Baudrate Api ☑

Development Error Detection ☑

Can Enable User Mode Support □

Provide Version Info API □

Figure 3-2. can_changebaudrate

3.11.1.4 CanVersionInfoApi

Table 3-50. CanVersionInfoApi

Description	Defines whether version information reporting should be included at compile time (STD_ON) or excluded (STD_OFF).
Class	Autosar Parameter
Range	True, False
Default	True
Source File	Can_Cfg.h
Source Representation	#define CAN_VERSION_INFO_API STD_OFF / STD_ON
Autosar 4.0 Requirement	CAN106_Conf

Note

Setting this define to STD_OFF will decrease the code size, but not support for Can_GetVersionInfo API will be available.

3.11.1.5 CanEnableUserModeSupport

Table 3-51. CanEnableUserModeSupport

Description	When this parameter is enabled, the CAN module will adapt to run from User Mode, with the following measures: (if applicable) a) configuring REG_PROT for the Can Controllers so that the registers under protection can be accessed from user mode by setting UAA bit in REG_PROT_GCR to 1 (if applicable) b) using 'call trusted function' stubs for all internal function calls that access registers requiring supervisor mode.

Table continues on the next page...

Table 3-51. CanEnableUserModeSupport (continued)

	(if applicable) c) other module specific measures for more information, please see chapter 5.7 User Mode Support in IM.
Class	Implementation Specific Parameter
Range	True, False
Default	False
Source File	NA
Source Representation	NA
Autosar 4.2 Requirement	NA

3.11.1.6 CanIndex

Table 3-52. CanIndex

Description	Specifies the Instance ID of this module instance. If only one instance is present it shall have the ID 0.
Class	Autosar Parameter
Range	Integer
Default	0
Source File	Can_Cfg.h
Source Representation	#define CAN_INSTANCE 0
Autosar 4.0 Requirement	CAN320_Conf
NOTE	*This parameter is transmitted as the second parameter to the Det_Reporterror function.*

Note

This parameter is transmitted as the second parameter to the Det_Reporterror function.

3.11.1.7 CanMainFunctionBusOffPeriod

Table 3-53. CanMainFunctionBusOffPeriod

Description	Describes the period for cyclic call to Can_MainFunction_Busoff (in seconds).
Class	Autosar Parameter
Range	0.001 .. 65.535
Default	0.001
Source File	Can_Cfg.h
Source Representation	#define CAN_MAINFUNCTION_PERIOD_BUSOFF 0.001U
Autosar 4.0 Requirement	CAN355_Conf

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.

3.11.1.8 CanMainFunctionModePeriod

Table 3-54. CanMainFunctionModePeriod

Description	Describes the period for cyclic call to Can_MainFunction_Mode (in seconds).
Class	Autosar Parameter
Range	0.001 .. 65.535
Default	0.001
Source File	Can_Cfg.h
Source Representation	#define CAN_MAINFUNCTION_MODE_PERIOD 0.001U
Autosar 4.0 Requirement	CAN376_Conf

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.

3.11.1.9 CanIdenticalIdCancellation

Table 3-55. CanIdenticalIdCancellation

Description	Specifies if identical ID cancellation shall be supported ON or OFF.
Class	Autosar Parameter
Range	True, False
Default	True
Source File	Can_Cfg.h
Source Representation	#define CAN_IDENTICAL_ID_CANCELLATION STD_ON
Autosar 4.0 Requirement	CAN378_Conf

Note

Setting this control to false, the Can Module shall not initiate a cancellation, when the hardware transmit object assigned by a HTH is busy, an L-PDU with identical priority is requested to be transmitted.

3.11.1.10 CanMultiplexedTransmission

Table 3-56. CanMultiplexedTransmission

Description	Defines whether support for multiplex transmission should be included at compile time (STD_ON) or excluded (STD_OFF).
Class	Autosar Parameter
Range	True, False
Default	True
Source File	Can_Cfg.h
Source Representation	#define CAN_MULTIPLEXED_TRANSMISSION STD_OFF / STD_ON
Autosar 4.0 Requirement	CAN095_Conf

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.

3.11.1.11 CanHardwareCancellation

Table 3-57. CanHardwareCancellation

Description	Specifies if hardware cancellation shall be supported ON or OFF.
Class	Autosar Parameter
Range	True, False
Default	True
Source File	Can_Cfg.h
Source Representation	#define CAN_HW_TRANSMIT_CANCELLATION STD_OFF / STD_ON
Autosar 4.0 Requirement	CAN069_Conf

Note

Setting this control to false will generate code that is reduced, but no support for MB cancellation is available.

3.11.1.12 CanMaxMessageBuffers

Table 3-58. CanMaxMessageBuffers

Description	This parameter describes the maximum Message Buffers (MBs) of whole active CAN controllers.
Class	Implementation Specific Parameter

Table continues on the next page...

Table 3-58. CanMaxMessageBuffers (continued)

Range	Integer
Default	96
Source File	Can_Cfg.h
Source Representation	#define CAN_MAXMB_CONFIGURED 96U
Autosar 4.0 Requirement	NA

Note

With a CAN controller, we have the maximum MBs which equals the sum of the MBs for RxFifo (if RxFifo enabled), HRHs (decrease 1 if RxFifo enabled), HTHs and the additional HTHs that are used for the multi transmission feature (if CanMultiplexedTransmission enabled). The value of CanMaxMessageBuffers must be equal or greater than that value .

3.11.1.13 CanTimeoutDurationFactor

Table 3-59. CanTimeoutDurationFactor

Description	Specifies the maximum number of loops for blocking function until a timeout is raised in short term wait loops.
Class	Autosar Parameter
Range	Integer
Default	2000
Source File	Can_Cfg.h
Source Representation	#define CAN_TIMEOUT_DURATION 20U
Autosar 4.0 Requirement	CAN113_Conf

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.

3.11.1.14 CanLPduReceiveCalloutFunction

Table 3-60. CanLPduReceiveCalloutFunction

Description	Specifies the name of the callout function.
Class	Autosar Parameter
Range	NA
Default	NULL_PTR
Source File	Can_Cfg.h
Source Representation	extern FUNC(boolean, COM_APPL_CODE) 'LPduCalloutFunction_name'(uint8 Hrh,Can_IdType CanId,uint8 CanDlc,const uint8 *CanSduPtr);
Autosar 4.0 Requirement	CAN434_Conf

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

3.11.1.15 CanCodeSizeOptimization

Table 3-61. CanCodeSizeOptimization

Description	Enables optimization of interrupt service routines for code size.
Class	Implementation Specific Parameter
Range	True, False
Default	True
Source File	Can_Cfg.h
Source Representation	#define CAN_ISROPTCODESIZE STD_OFF / STD_ON
Autosar 4.0 Requirement	NA

Note

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

3.11.1.16 CanExtendedIdSupport

Table 3-62. CanExtendedIdSupport

Description	Enables support of Extended/Mixed mode.
Class	Implementation Specific Parameter
Range	True, False
Default	True
Source File	Can_Cfg.h
Source Representation	#define CAN_EXTENDEDID STD_OFF / STD_ON
Autosar 4.0 Requirement	NA

Note

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

3.11.1.17 CanMBCountExtensionSupport

Table 3-63. CanMBCountExtensionSupport

Description	Enables support of more than 255 Can Hardware Objects.
Class	Implementation Specific Parameter
Range	True, False
Default	False
Source File	Can_Cfg.h
Source Representation	#define CAN_MBCOUNTEXTENSION STD_OFF / STD_ON
Autosar 4.0 Requirement	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.

3.11.1.18 CanApiEnableMbAbort

Table 3-64. CanApiEnableMbAbort

Description	Enables an additional API, to write an ABORT code (b1001) to the MBCB filed of the MB to abort a message transmission.
Class	Implementation Specific Parameter
Range	True, False
Default	False
Source File	Can_Cfg.h
Source Representation	#define CAN_API_ENABLE_ABORT_MB STD_OFF / STD_ON
Autosar 4.0 Requirement	NA

Note

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

3.11.1.19 CanEnableDualClockMode

Table 3-65. CanEnableDualClockMode

Description	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.
Class	Implementation Specific Parameter
Range	True, False
Default	False
Source File	Can_Cfg.h
Source Representation	#define CAN_DUAL_CLOCK_MODE STD_OFF / STD_ON
Autosar 4.0 Requirement	NA

Note

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

3.11.1.20 CanSpecifiedRAMBlockSize

Table 3-66. CanSpecifiedRAMBlockSize

Description	This parameter is used to enable the feature which separately configure Message Buffer Data Size for each RAM block.
Class	Implementation Specific Parameter
Range	True, False

Table continues on the next page...

Table 3-66. CanSpecifiedRAMBlockSize (continued)

Default	False
Source File	NA
Source Representation	NA
Autosar 4.2 Requirement	NA

3.11.1.21 CanCounterRef

Table 3-67. CanCounterRef

Description	Contains a reference to the counter.
Class	Autosar Parameter
Range	NA
Default	Reference to [OsCounter]
Source File	NA
Source Representation	NA
Autosar 4.0 Requirement	CAN431_Conf

Note

This parameter contains a reference to the counter, which is used by the CAN driver. The Can Driver does not support this requirement.

3.11.1.22 CanSupportTTCANRef

Table 3-68. CanSupportTTCANRef

Description	Refers to CanIfSupportTTCAN parameter in the CAN Interface Module configuration.
Class	Autosar Parameter
Range	NA
Default	NA
Source File	NA
Source Representation	NA
Autosar 4.0 Requirement	CAN430_Conf

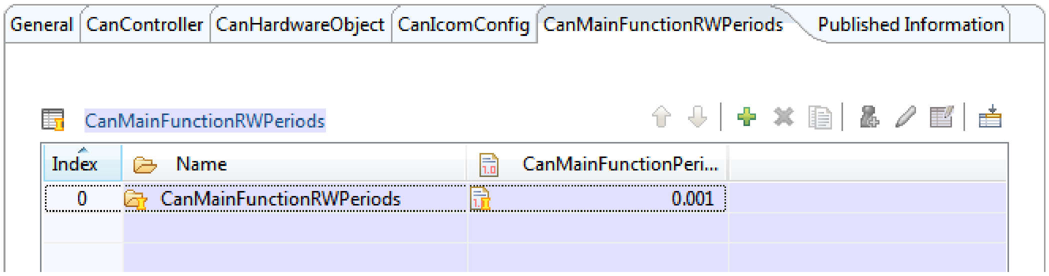
Note

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

3.11.1.23 CanMainFunctionRWPeriods

Table 3-69. CanMainFunctionRWPeriods

Container Name	CanMainFunctionRWPeriods{CAN_MAIN_FUNCTION_RWPERIODS}
Description	Reference to CAN Controller to which the HOH is associated to.
Class	Autosar Parameter
Autosar 4.0 Requirement	CAN437_Conf



Index	Name	CanMainFunctionPeri...
0	CanMainFunctionRWPeriods	0.001

Figure 3-3. 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.

3.11.1.23.1 CanMainFunctionPeriod

Table 3-70. CanMainFunctionPeriod

Description	This parameter describes the period for cyclic call to Can_MainFunction_Read or Can_MainFunction_Write depending on the referring item.
Class	Autosar Parameter
Range	0.001 .. 65.535
Default	NA
Source File	Can_Cfg.h
Source Representation	#define CAN_MAINFUNCTION_PERIOD_READ 0.001U #define CAN_MAINFUNCTION_PERIOD_WRITE 0.001U
Autosar 4.2 Requirement	ECUC_Can_00484

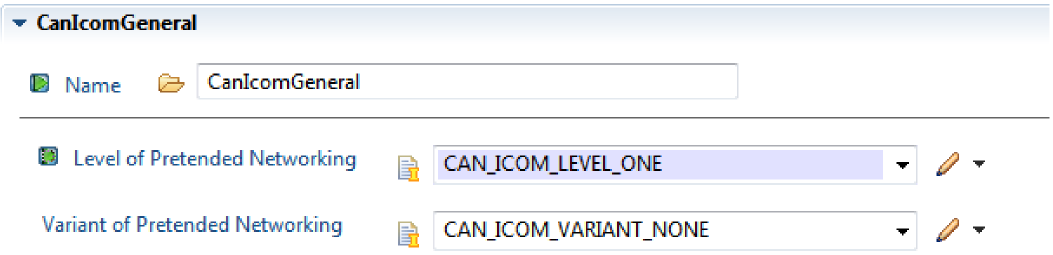
Note

Different poll-cycles will be configurable if more than one CanMainFunctionPeriod is configured. In this case multiple Can_MainFunction_Read() or Can_MainFunction_Write() will be provided by the CAN Driver module.

3.11.1.24 CanIcomGeneral

Table 3-71. CanIcomGeneral

Container Name	CanIcomGeneral
Description	This container contains the general configuration parameters of the ICOM Configuration.
Class	Implementation Specific Container
Autosar 4.2 Requirement	NA



CanIcomGeneral

Name: CanIcomGeneral

Level of Pretended Networking: CAN_ICOM_LEVEL_ONE

Variant of Pretended Networking: CAN_ICOM_VARIANT_NONE

Figure 3-4. CanIcomGeneral

3.11.1.24.1 CanIcomLevel

Table 3-72. CanIcomLevel

Description	Defines the level of Pretended Networking.This parameter is reserved for future implementations (Pretended Networking level 2).
Class	Autosar Parameter
Range	CAN_ICOM_LEVEL_ONE
Default	CAN_ICOM_LEVEL_ONE
Source File	NA
Source Representation	NA
Autosar 4.2 Requirement	ECUC_Can_00445

3.11.1.24.2 CanIcomVariant

Table 3-73. CanIcomVariant

Description	Defines the variant, which is supported by this CanController.

Table continues on the next page...

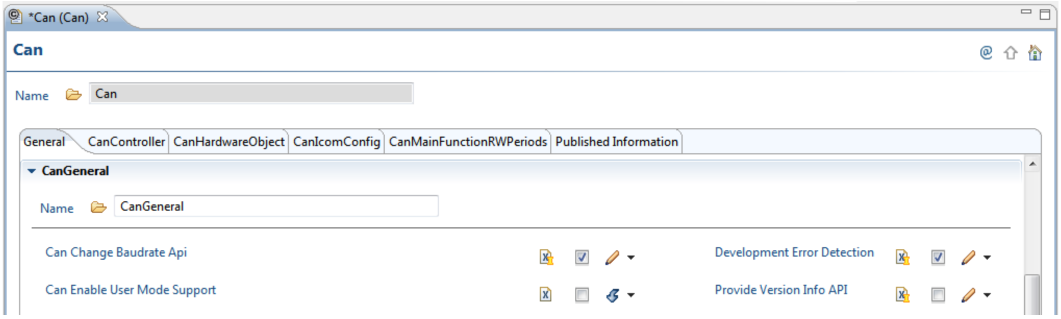
Table 3-73. CanIcomVariant (continued)

Class	Autosar Parameter
Range	CAN_ICOM_VARIANT_HW, CAN_ICOM_VARIANT_NONE, CAN_ICOM_VARIANT_SW
Default	CAN_ICOM_VARIANT_NONE
Source File	NA
Source Representation	NA
Autosar 4.2 Requirement	ECUC_Can_00446

3.11.1.25 CanChangeBaudrateApi

Table 3-74. CanChangeBaudrateApi

Description	Defines whether baudrate information reporting should be included at compile time (STD_ON) or excluded (STD_OFF).
Class	Autosar Parameter
Range	True, False
Default	True
Source File	Can_Cfg.h
Source Representation	#define CAN_CHANGE_BAUDRATE_API STD_OFF / STD_ON
Autosar 4.0 Requirement	CAN436_Conf



Name: Can

Name: CanGeneral

Can Change Baudrate Api ☑

Development Error Detection ☑

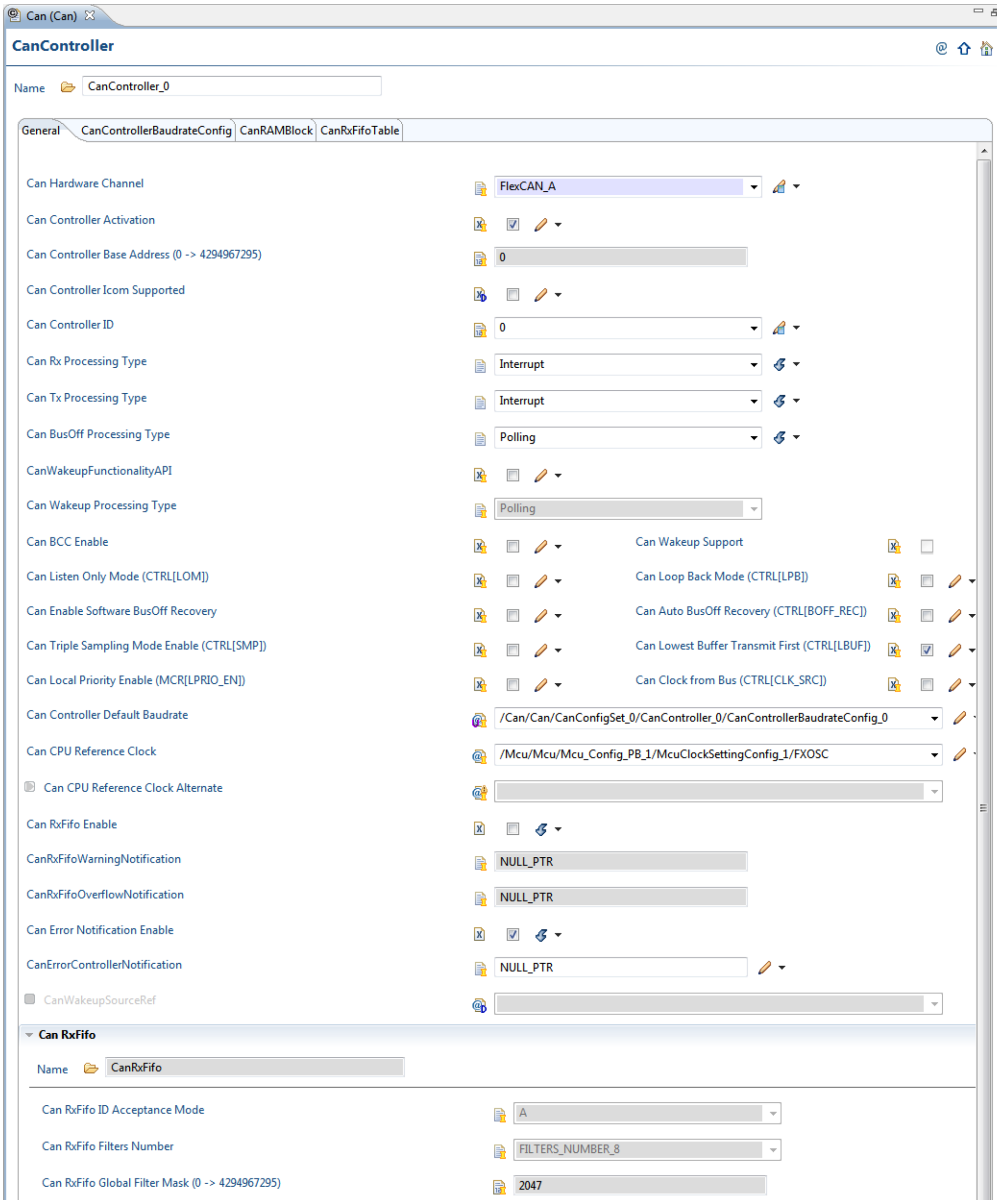
Can Enable User Mode Support □

Provide Version Info API □

Figure 3-5. can_changebaudrate

3.11.2 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.



Can (Can)

CanController

Name: CanController_0

Can Hardware Channel: FlexCAN_A

Can Controller Activation: ☑

Can Controller Base Address (0 -> 4294967295): 0

Can Controller Icom Supported: □

Can Controller ID: 0

Can Rx Processing Type: Interrupt

Can Tx Processing Type: Interrupt

Can BusOff Processing Type: Polling

CanWakeupFunctionalityAPI: □

Can Wakeup Processing Type: Polling

Can BCC Enable: □

Can Wakeup Support: □

Can Listen Only Mode (CTRL[LOM]): □

Can Loop Back Mode (CTRL[LPB]): □

Can Enable Software BusOff Recovery: □

Can Auto BusOff Recovery (CTRL[BOFF_REC]): □

Can Triple Sampling Mode Enable (CTRL[SMP]): □

Can Lowest Buffer Transmit First (CTRL[LBUF]): ☑

Can Local Priority Enable (MCR[LPRIO_EN]): □

Can Clock from Bus (CTRL[CLK_SRC]): □

Can Controller Default Baudrate: /Can/Can/CanConfigSet_0/CanController_0/CanControllerBaudrateConfig_0

Can CPU Reference Clock: /Mcu/Mcu/Mcu_Config_PB_1/McuClockSettingConfig_1/FXOSC

Can CPU Reference Clock Alternate:

Can RxFifo Enable: □

CanRxFifoWarningNotification: NULL_PTR

CanRxFifoOverflowNotification: NULL_PTR

Can Error Notification Enable: ☑

CanErrorControllerNotification: NULL_PTR

CanWakeupSourceRef:

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 3-6. Can Controller Parameters

3.11.2.1 CanHwChannel

Table 3-75. 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 3-75. CanHwChannel (continued)

Source File	Can_Cfg.c, Can_PBcfg.c

Source Representation	CONST(CanStatic_ControlerDescriptorType, CAN_CONST) StaticControlerDescriptors_PC[CAN_MAXCONTROLLERCOUNT] = {
	....
	/* Can Controller Offset on chip */
	FLEXCAN_A_OFFSET,
	....
	}
	CONST(CanStatic_ControlerDescriptorType, CAN_CONST) StaticControlerDescriptors[index]_PB[CAN_MAXCONTROLLERCOUNT_[index]] = {
	....
	/* Can Controller Offset on chip */
	FLEXCAN_A_OFFSET,
	....
	}

Autosar 4.0 Requirement	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.

3.11.2.2 CanControllerActivation

Table 3-76. CanControllerActivation

Description	Defines if a CAN controller is used in the configuration.
Class	Autosar Parameter
Range	True , False
Default	True
Source File	Can_Cfg.c, Can_PBcfg.c

Source Representation	CONST(CanStatic_ControlerDescriptorType, CAN_CONST) StaticControlerDescriptors_PC[CAN_MAXCONTROLLERCOUNT] = {
	....
	/* ControllerState */
	CONTRL_ENABLED,
	....
	}

Table continues on the next page...

Table 3-76. CanControllerActivation (continued)

	CONST(CanStatic_ControlerDescriptorType, CAN_CONST) StaticControlerDescriptors[index]_PB[CAN_MAXCONTROLLERCOUNT_[index]] = {
	....
	/* ControllerState */
	CONTRL_ENABLED,
	....
	}

Autosar 4.0 Requirement	CAN315_Conf

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).

3.11.2.3 CanControllerBaseAddress

Table 3-77. CanControllerBaseAddress

Description	Specifies the CAN controller base address.
Class	Autosar Parameter
Range	0 .. 4294967295
Default	0
Source File	NA
Source Representation	NA
Autosar 4.0 Requirement	CAN382_Conf

Note

This requirement is not supported by the Can Driver.

3.11.2.4 CanControllerIcomSupported

Table 3-78. CanControllerIcomSupported

Description	Define if the current controller support Pretended Networking (Icom).
Class	Implementation Specific Parameter
Range	NA
Default	/Can/CanConfigSet/CanController/CanController_0/CanControllerIcomSupported
Source File	Can_Cfg.c
Source Representation	NA
Autosar 4.2 Requirement	

3.11.2.5 CanControllerId

Table 3-79. CanControllerId

Description	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.
Class	Autosar Parameter
Range	0 .. 255
Default	0
Source File	Can_Cfg.h
Source Representation	#define CanController_0 0U /* Default configuration for FlexCAN_A */
Autosar 4.0 Requirement	CAN316_Conf

Note

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

3.11.2.6 CanRxProcessing

Table 3-80. CanRxProcessing

Description	Specifies if RX events are polled inside Can_MainFunction_Read or cause an interrupt.
Class	Autosar Parameter
Range	Polling , Interrupt
Default	Polling
Source File	Can_Cfg.c, Can_PBcfg.c

Source Representation	CONST(CanStatic_ControlerDescriptorType, CAN_CONST) StaticControlerDescriptors_PC[CAN_MAXCONTROLLERCOUNT_[index]] = {
	....
	/* ===== Controller Options ===== */
	/* RxPoll Enabled */
	CAN_CONTROLLERCONFIG_RXPOL_EN_U32
	....
	}
	CONST(CanStatic_ControlerDescriptorType, CAN_CONST) StaticControlerDescriptors[index]_PB[CAN_MAXCONTROLLERCOUNT_[index]] = {
	....
	/* ===== Controller Options ===== */
	/* RxPoll Enabled */

Table continues on the next page...

Table 3-80. CanRxProcessing (continued)

	CAN_CONTROLLERCONFIG_RXPOL_EN_U32 .... }
Autosar 4.0 Requirement	CAN317_Conf

Note

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

3.11.2.7 CanTxProcessing

Table 3-81. CanTxProcessing

Description	Specifies if TX events are polled inside Can_MainFunction_Write or cause an interrupt.
Class	Autosar Parameter
Range	Polling , Interrupt
Default	Polling
Source File	Can_Cfg.c, Can_PBcfg.c

Source Representation	CONST(CanStatic_ControlerDescriptorType, CAN_CONST) StaticControlerDescriptors_PC[CAN_MAXCONTROLLERCOUNT] = {
	....
	/* ===== Controller Options ===== */
	/* TxPoll Enabled */
	CAN_CONTROLLERCONFIG_TXPOL_EN_U32 |
	....
	}
	CONST(CanStatic_ControlerDescriptorType, CAN_CONST) StaticControlerDescriptors[index]_PB[CAN_MAXCONTROLLERCOUNT_[index]] = {
	....
	/* ===== Controller Options ===== */
	/* TxPoll Enabled */
	CAN_CONTROLLERCONFIG_TXPOL_EN_U32 |
	....
	}

Autosar 4.0 Requirement	CAN318_Conf

Note

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

3.11.2.8 CanBusOffProcessing

Table 3-82. CanBusOffProcessing

Description	Specifies if bus-off events are polled inside Can_Main_Function_BusOff or cause an interrupt.
Class	Autosar Parameter
Range	Polling , Interrupt
Default	Polling
Source File	Can_Cfg.c, Can_PBcfg.c

Source Representation	CONST(CanStatic_ControlerDescriptorType, CAN_CONST) StaticControlerDescriptors_PC[CAN_MAXCONTROLLERCOUNT_[index]] = {
	....
	/* ===== Controller Options ===== */
	/* BusOffPoll Enabled */
	CAN_CONTROLLERCONFIG_BOPOL_EN_U32 |
	....
	}
	CONST(CanStatic_ControlerDescriptorType, CAN_CONST) StaticControlerDescriptors[index]_PB[CAN_MAXCONTROLLERCOUNT_[index]] = {
	....
	/* ===== Controller Options ===== */
	/* BusOffPoll Enabled */
	CAN_CONTROLLERCONFIG_BOPOL_EN_U32 |
	....
	}

Autosar 4.0 Requirement	CAN314_Conf

Note

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

3.11.2.9 CanListenOnlyMode

Table 3-83. CanListenOnlyMode

Description	Enables the Listen only mode.
Class	Implementation Specific Parameter
Range	True, False
Default	False
Source File	Can_Cfg.c, Can_PBcfg.c

Table continues on the next page...

Table 3-83. CanListenOnlyMode (continued)

Source Representation	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 */
	....
	}

Autosar 4.0 Requirement	NA

Note

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

3.11.2.10 CanLoopBackMode

Table 3-84. CanLoopBackMode

Description	Enables the Loop Back Mode.
Class	Implementation Specific Parameter
Range	True, False
Default	False
Source File	Can_Cfg.c, Can_PBcfg.c

Source Representation	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 ===== */
	....

Table continues on the next page...

Table 3-84. CanLoopBackMode (continued)

	/* CTRL[LPB] - Loop-back mode */ .... }
Autosar 4.0 Requirement	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").

3.11.2.11 CanSoftwareBusOffRecovery

Table 3-85. CanSoftwareBusOffRecovery

Description	Enables Automatic Bus Recovery Off.
Class	Implementation Specific Parameter
Range	True, False
Default	False
Source File	Can_Cfg.c, Can_PBcfg.c

Source Representation	CONST(Can_ControlerDescriptorType, CAN_CONST) ControlerDescriptors_PC[CAN_MAXCONTROLLERCOUNT] = {
	/* ===== Controller Options ===== */
	/* Software BusOff Recovery */
	CAN_CONTROLLERCONFIG_BUSOFFSWREC_U32 |
	....
	}
	CONST(Can_ControlerDescriptorType, CAN_CONST) ControlerDescriptors[index]_PB[CAN_MAXCONTROLLERCOUNT_[index]] = {
	/* ===== Controller Options ===== */
	/* Software BusOff Recovery */
	CAN_CONTROLLERCONFIG_BUSOFFSWREC_U32 |
	....
	}

Autosar 4.0 Requirement	NA

Note

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

3.11.2.12 CanAutoBusOffRecovery

Table 3-86. CanAutoBusOffRecovery

Description	Enables Automatic Bus Recovery Off.
Class	Implementation Specific Parameter
Range	True, False
Default	False
Source File	Can_Cfg.c, Can_PBcfg.c

Source Representation	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 */
	....
	}

Autosar 4.0 Requirement	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.

3.11.2.13 CanTripleSamplingEnable

Table 3-87. CanTripleSamplingEnable

Description	Enables acquisition of 3 samples and majority voting for the value of received bit.
Class	Implementation Specific Parameter
Range	True, False
Default	False

Table continues on the next page...

Table 3-87. CanTripleSamplingEnable (continued)

Source File	Can_Cfg.c, Can_PBcfg.c

Source Representation	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 */
	....
	}

Autosar 4.0 Requirement	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.

3.11.2.14 CanLowestBuffTransmitFirst

Table 3-88. CanLowestBuffTransmitFirst

Description	This parameter defines the ordering mechanism for MB transmission.
Class	Implementation Specific Parameter
Range	True, False
Default	False
Source File	Can_Cfg.c, Can_PBcfg.c

Source Representation	CONST(CanStatic_ControllerBaudrateConfigType, CAN_CONST) StaticControllerBaudrateCfgSet_PC[CAN_MAXCONTROLLERCOUNT] = {
	/* ===== Control Register - CTRL ===== */
	....
	/* CTRL[LBUF] - Lowest Buffer Transmitted First */
	....
	}

Table continues on the next page...

Table 3-88. CanLowestBuffTransmitFirst (continued)

	CONST(CanStatic_ControllerBaudrateConfigType, CAN_CONST) StaticControllerBaudrateCfgSet[index]_PB[CAN_MAXCONTROLLERCOUNT_[index]] = {
	/* ===== Control Register - CTRL ===== */
	....
	/* CTRL[LBUF] - Lowest Buffer Transmitted First */
	....
	}

Autosar 4.0 Requirement	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.

3.11.2.15 CanLocalPriorityEn

Table 3-89. CanLocalPriorityEn

Description	This field is used when MCR[LPRIO_EN] is set and makes sense only for Tx MBs.
Class	Implementation Specific Parameter
Range	True, False
Default	False
Source File	Can_Cfg.c, Can_PBcfg.c

Source Representation	CONST(Can_ControlerDescriptorType, CAN_CONST) ControlerDescriptors_PC[CAN_MAXCONTROLLERCOUNT_0] = {
	/* ===== Controller Options ===== */
	....
	/* Local Priority Feature */
	CAN_CONTROLLERCONFIG_LPRIO_EN_U32 |
	}

Autosar 4.0 Requirement	NA

Note

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

3.11.2.16 CanWarningEnable

Table 3-90. CanWarningEnable

Description	This parameter defines if warning interrupt is enabled for Rx and Tx.
Class	Implementation Specific Parameter
Range	True, False
Default	False
Source File	Can_Cfg.c, Can_PBCfg.c
Source Representation	CONST(Can_ControlerDescriptorType, CAN_CONST) ControllerConfigs0_PB[CAN_MAXCONTROLLERCOUNT] = { …….. / * Warning Interrupt Enable Feature */ CAN_CONTROLLERCONFIG_WRNINT_EN | ………..
Autosar 4.0 Requirement	NA

3.11.2.17 CanClockFromBus

Table 3-91. CanClockFromBus

Description	Switches the source clock for the module to the system bus (rather than crystal).
Class	Implementation Specific Parameter
Range	True, False
Default	False
Source File	Can_Cfg.c, Can_PBcfg.c

Source Representation	CONST(Can_ControllerBaudrateConfigType, CAN_CONST) ControllerBaudrateConfigs_PC[CAN_MAXCONTROLLERCOUNT] = {
	/* ===== Control Register - CTRL ===== */
	....
	/* CTRL[CLK_SRC] - Clock source */
	....
	}

Autosar 4.0 Requirement	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.

3.11.2.18 CanCpuClockRef

Table 3-92. CanCpuClockRef

Description	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.
Class	Autosar Parameter
Range	NA
Default	/Mcu/Mcu/McuModuleConfiguratio_0/McuClockSettingConfig/McuClockReferencePoin_0
Source File	Can_Cfg.c, Can_PBcfg.c
Source Representation	NA
Autosar 4.0 Requirement	CAN313_Conf

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. It is user responsability to synchronize the value selected in this field with “CanClockFromBus” value

3.11.2.19 CanControllerRXFifoEnable

Table 3-93. CanControllerRXFifoEnable

Description	Defines if RX FIFO feature of CAN controller is used in the configuration.
Class	Implementation Specific Parameter
Range	True, False
Default	False
Source File	Can_Cfg.h
Source Representation	#define CAN_A_FIFO_EN STD_ON / STD_OFF
Autosar 4.0 Requirement	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.

3.11.2.20 CanRxFifoWarningNotification

Table 3-94. CanRxFifoWarningNotification

Description	Defines the handler for Rx Fifo warning.
Class	Implementation Specific Parameter
Range	String
Default	NULL_PTR
Source File	Can_Cfg.c, Can_PBcfg.c
Source Representation	extern FUNC(void, CAN_CODE) handler_name (void);
Autosar 4.0 Requirement	NA

3.11.2.21 CanRxFifoOverflowNotification

Table 3-95. CanRxFifoOverflowNotification

Description	Defines the handler for Rx Fifo overflow.
Class	Implementation Specific Parameter
Range	String
Default	NULL_PTR
Source File	Can_Cfg.c, Can_PBcfg.c
Source Representation	extern FUNC(void, CAN_CODE) handler_name (void);
Autosar 4.0 Requirement	NA

3.11.2.22 CanErrorControllerNotification

Table 3-96. CanErrorControllerNotification

Description	Defines the handler for error controller.
Class	Implementation Specific Parameter
Range	String
Default	NULL_PTR
Source File	Can_Cfg.c, Can_PBcfg.c
Source Representation	extern FUNC(void, CAN_CODE) handler_name (void);
Autosar 4.0 Requirement	NA

3.11.2.23 CanCpuClockRef_Alternate

Table 3-97. CanCpuClockRef_Alternate

Description	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.
Class	Implementation Specific Parameter
Range	NA
Default	/Mcu/Mcu/McuModuleConfiguratio_0/McuClockSettingConfig/McuClockReferencePoin_0
Source File	Can_Cfg.c, Can_PBcfg.c
Source Representation	NA
Autosar 4.0 Requirement	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 responsability to synchronize the value selected in this field with “CanClockFromBus” value

3.11.2.24 CanBccSupport

Table 3-98. CanBccSupport

Description	Defines if Backwards Compatibility Configuration (BCC) feature of CAN controller is used in the configuration.
Class	Implementation Specific Parameter
Range	True, False
Default	False
Source File	Can_Cfg.h
Source Representation	#define CAN_BCC_SUPPORT_ENABLE STD_ON / STD_OFF
Autosar 4.0 Requirement	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

*/

3.11.2.25 CanErrorControllerNotifEn

Table 3-99. CanErrorControllerNotifEn

Description	Enables/Disables the Error Controller Notification. If Disabled, no error interrupt or notification shall take place.
Class	Implementation Specific Parameter
Range	True, False
Default	True
Source File	Can_Cfg.h
Source Representation	#define CAN_ERROR_NOTIFICATION_ENABLE STD_ON / STD_OFF
Autosar 4.0 Requirement	NA

Note

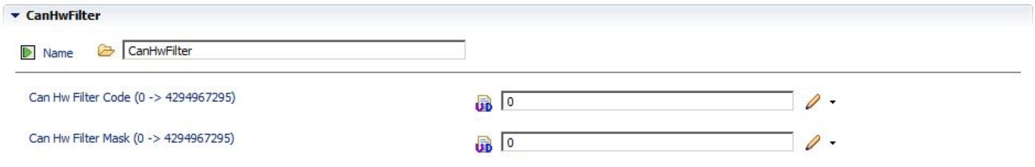
Not AutoSar Required.

3.11.2.26 CanControllerDefaultBaudrate

Table 3-100. CanControllerDefaultBaudrate

Description	Reference to baudrate configuration container configured for the Can Controller.
Class	Implementation Specific Parameter
Range	NA
Default	/Can/Can/CanConfigSet_0/CanController_0/CanControllerBaudrateConfig_0
Source File	Can_Cfg.c, Can_PBcfg.c
Source Representation	NA
Autosar 4.0 Requirement	CAN435_Conf

3.11.3 CanHwFilter



CanHwFilter

Name CanHwFilter

Can Hw Filter Code (0 -> 4294967295) 0

Can Hw Filter Mask (0 -> 4294967295) 0

Figure 3-7. CanHwFilter

3.11.3.1 CanHwFilterMask

Table 3-101. CanHwFilterMask

Description	Reference to the filter mask that is used for hardware filtering together with the CAN_ID_VALUE
Class	Autosar Parameter
Range	0 - 4294967296
Default	2047
Source File	Can_Cfg.c, Can_PBCfg.c

Source Representation	CONST(Can_IdType, CAN_CONST) Can_FilterMasks0_PB[CAN_MAXFILTERCOUNT_0] =
	{
	....
	/* FilterMasks_PC[0], "CanFilterMask_0" */
	0x7ffU,
	....
	}

Autosar 4.0 Requirement	CAN066_Conf

Note

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

3.11.3.2 CanHwFilterCode

Table 3-102. CanHwFilterCode

Description	Reference to the filter mask that is used for hardware filtering together with the CAN_ID_VALUE
Class	Autosar Parameter
Range	0 - 4294967296
Default	2047
Source File	Can_Cfg.c, Can_PBCfg.c

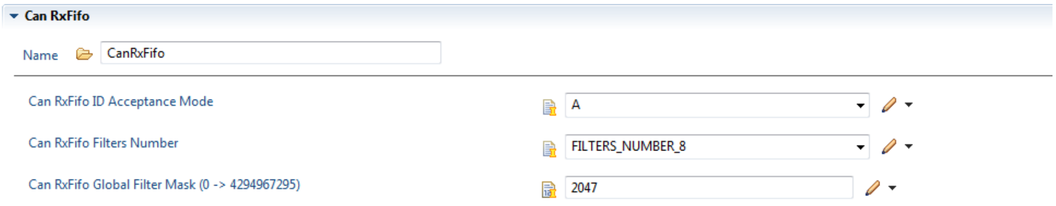
Source Representation	CONST(Can_IdType, CAN_CONST) Can_FilterMasks0_PB[CAN_MAXFILTERCOUNT_0] =
	{
	....
	/* FilterMasks_PC[0], "CanFilterMask_0" */
	0x7ffU,
	....
	}

Autosar 4.0 Requirement	CAN066_Conf

Note

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

3.11.4 Can RxFifo



Can RxFifo

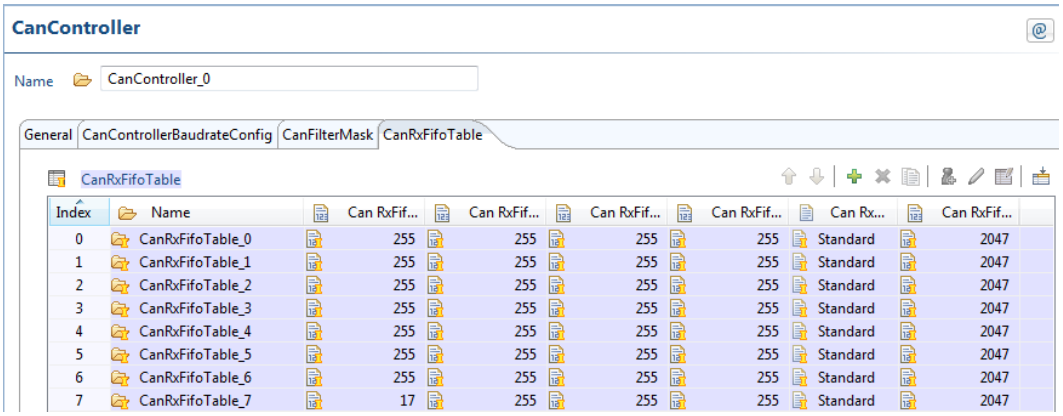
Name: CanRxFifo

Can RxFifo ID Acceptance Mode: A

Can RxFifo Filters Number: FILTERS_NUMBER_8

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

Figure 3-8. Can RxFifo Filters Number and Can RxFifo Global Filter Mask



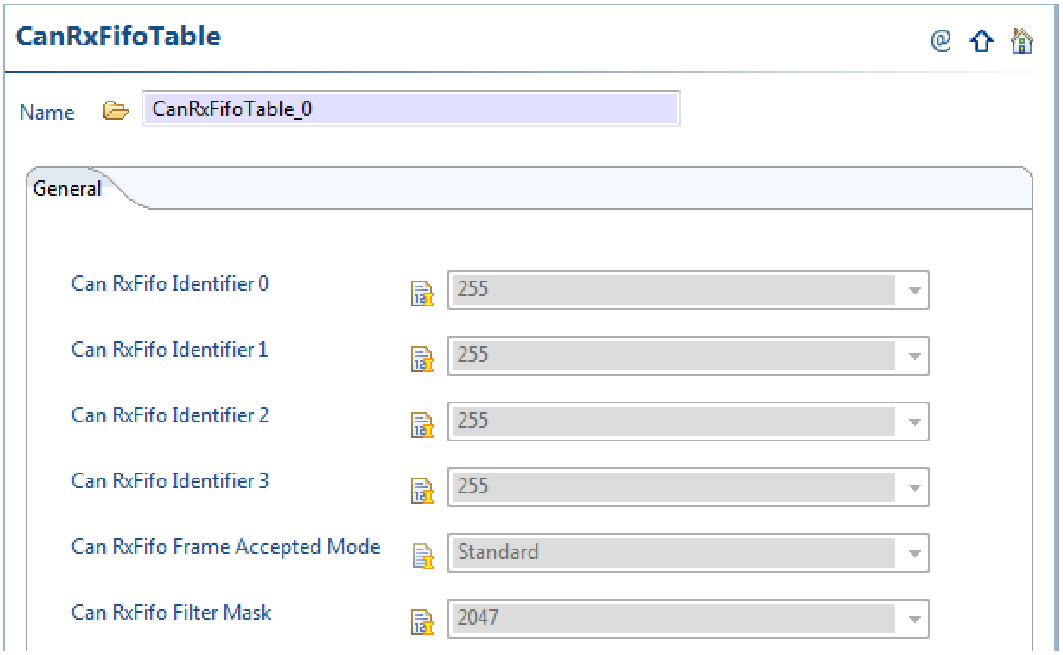
CanController

Name: CanController_0

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 3-9. Can RxFifo



CanRxFifoTable

Name: CanRxFifoTable_0

General

Can RxFifo Identifier 0: 255

Can RxFifo Identifier 1: 255

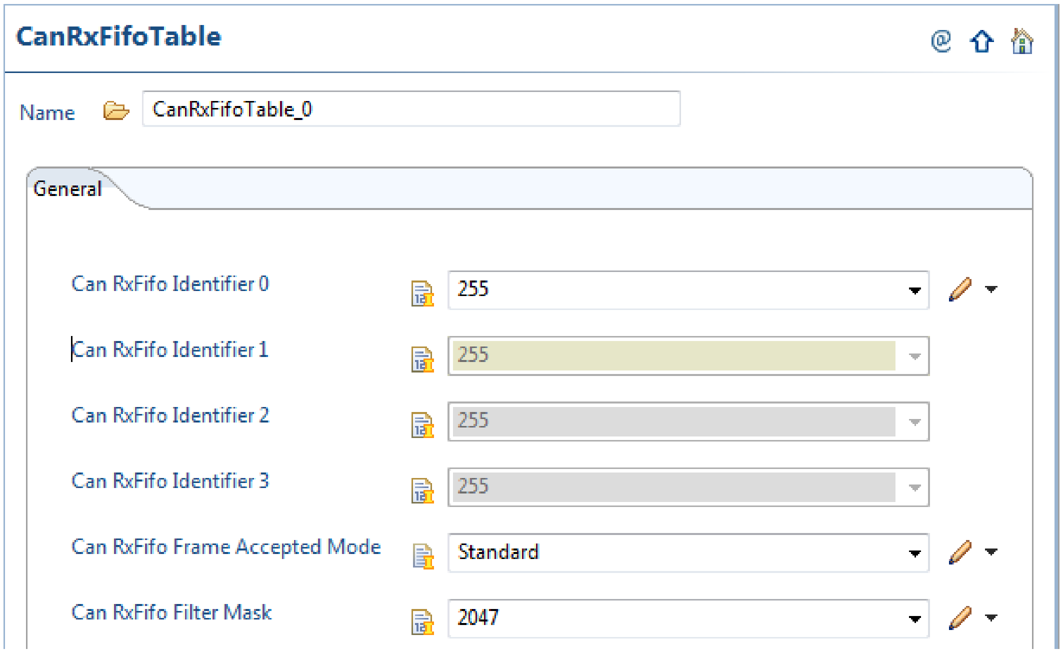
Can RxFifo Identifier 2: 255

Can RxFifo Identifier 3: 255

Can RxFifo Frame Accepted Mode: Standard

Can RxFifo Filter Mask: 2047

Figure 3-10. Can CanRxFifoTable - CanControllerRxFifoEnable = FALSE



CanRxFifoTable

Name: CanRxFifoTable_0

General

Can RxFifo Identifier 0: 255

Can RxFifo Identifier 1: 255

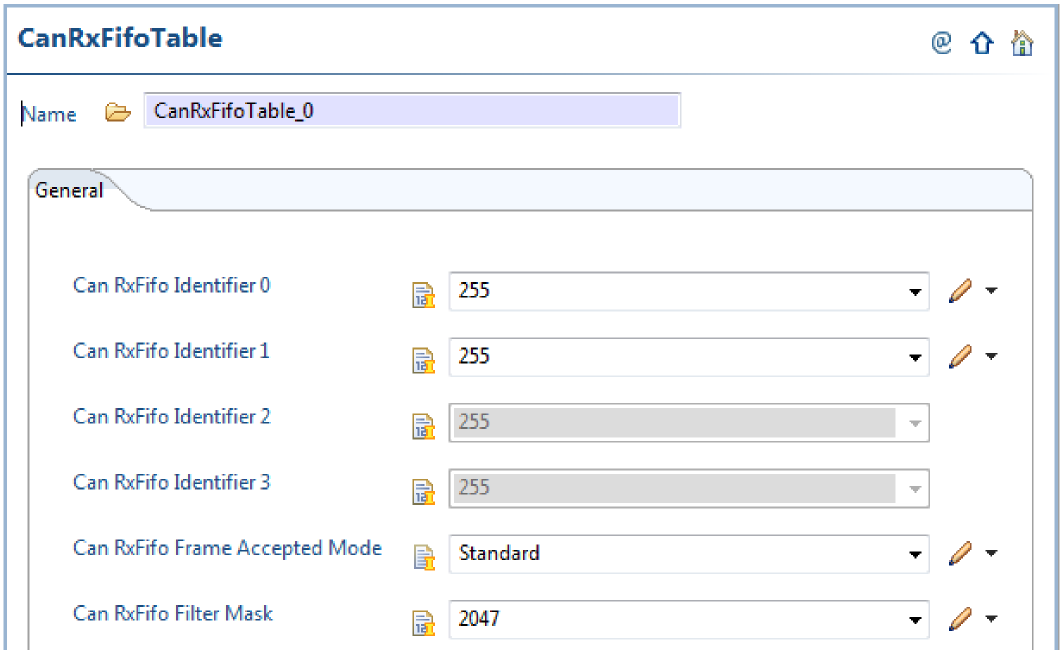
Can RxFifo Identifier 2: 255

Can RxFifo Identifier 3: 255

Can RxFifo Frame Accepted Mode: Standard

Can RxFifo Filter Mask: 2047

Figure 3-11. Can CanRxFifoTable - CanControllerRxFifoEnable = TRUE and CanControllerIDAcceptanceMode = A



CanRxFifoTable

Name: CanRxFifoTable_0

General

Can RxFifo Identifier 0: 255

Can RxFifo Identifier 1: 255

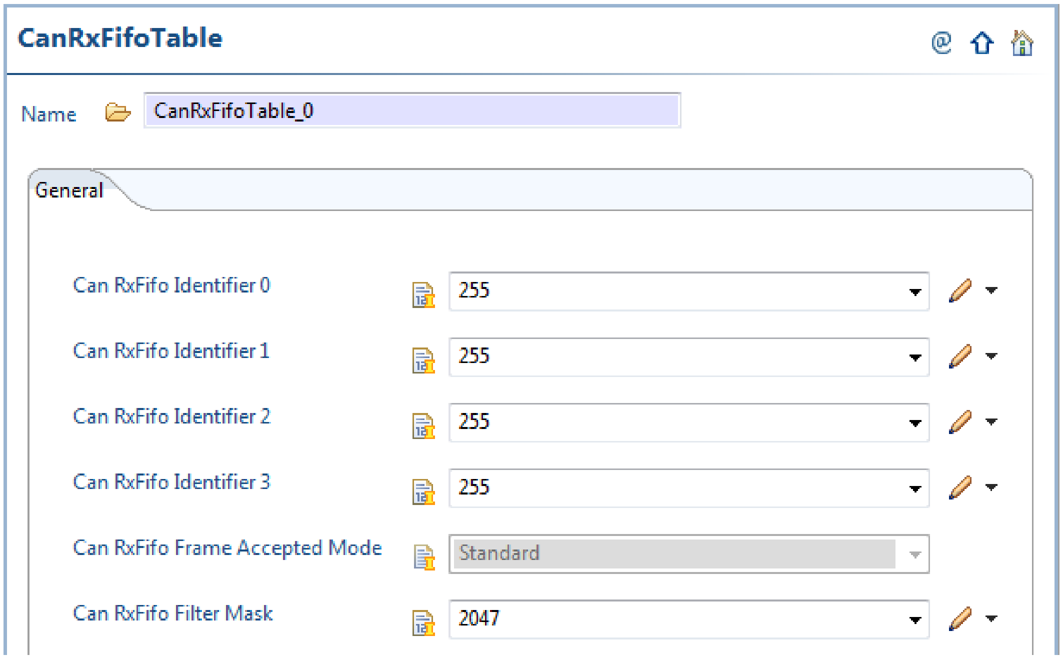
Can RxFifo Identifier 2: 255

Can RxFifo Identifier 3: 255

Can RxFifo Frame Accepted Mode: Standard

Can RxFifo Filter Mask: 2047

Figure 3-12. Can CanRxFifoTable - CanControllerRxFifoEnable = TRUE and CanControllerIDAcceptanceMode = B



CanRxFifoTable

Name: CanRxFifoTable_0

General

Can RxFifo Identifier 0: 255

Can RxFifo Identifier 1: 255

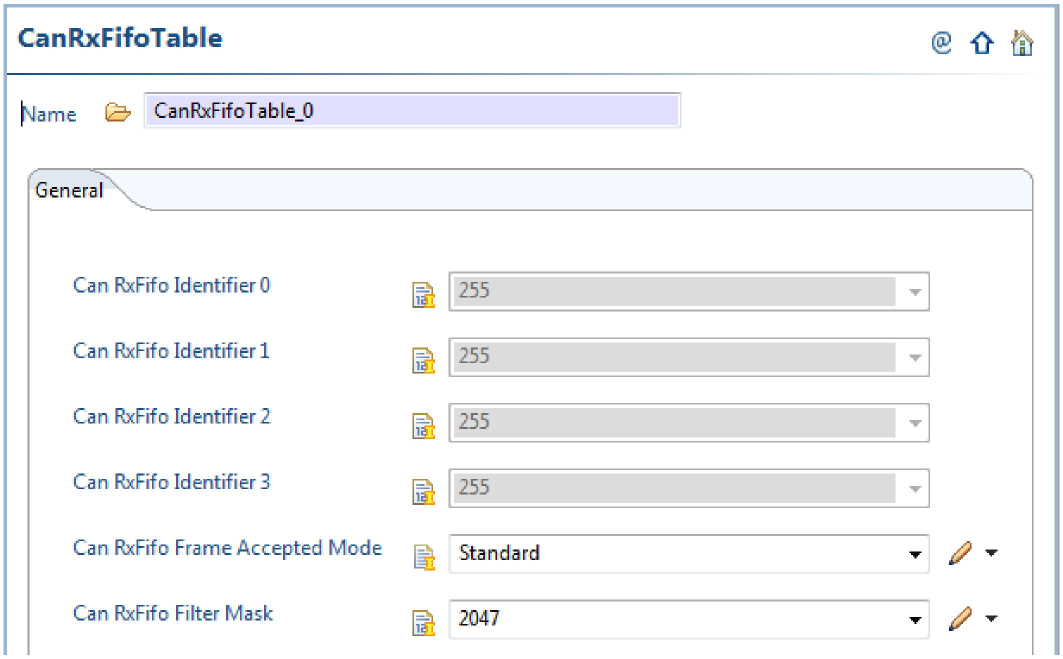
Can RxFifo Identifier 2: 255

Can RxFifo Identifier 3: 255

Can RxFifo Frame Accepted Mode: Standard

Can RxFifo Filter Mask: 2047

Figure 3-13. 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 3-14. Can CanRxFifoTable - CanControllerRxFifoEnable = TRUE and CanControllerIDAcceptanceMode = D

3.11.4.1 CanControllerIDAcceptanceMode

Table 3-103. 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	CONST(Can_ControlerDescriptorType, CAN_CONST) ControlerDescriptors_PC[CAN_MAXCONTROLLERCOUNT_0] = {
	....
	/* ===== Controller Options ===== */
	/* ID Acceptance Mode A */
	CAN_CONTROLLERCONFIG_IDAM_A_U32 |
	}

AUTOSAR 4.0 Requirement	NA

3.11.4.2 CanIDValue0

Table 3-104. CanIDValue0

Description	Specifies an ID to be used as acceptance criteria for the ID Table0. Value for this parameter should be entered based on the CanControllerIDAcceptanceMode selected as explained below: 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.
Class	Implementation Specific Parameter
Range	0 - 2047 for Format A and B– Standard CanTableIDType typ 0 – 536870911 for Format A – Extended CanTableIDType type 0 – 1683 for Format B – Extended CanTableIDType type 0 – 255 for Format C
Default	0xF0
Source File	Can_Cfg.c, Can_PBCfg.c

Source Representation	Format C
	CONST(Can_RxFiFoTableIdConfigType, CAN_CONST) RxFifoTableID_PCConfig[CAN_MAXTABLEID_0] = {
	{0x**11**121314, /* CanRxFifoTable_0 of formatC for FlexCAN_A */
	0xffffffff }}
	Format B
	CONST(Can_RxFiFoTableIdConfigType, CAN_CONST) RxFifoTableID_PBConfig [CAN_MAXTABLEID]= {
	{ {0x**1ff8**07f8, /* CanRxFifoTable_0 of type Standard and formatB for FlexCAN_A */
	0xfff8fff8 }}
	Format A
	CONST(Can_RxFiFoTableIdConfigType, CAN_CONST) RxFifoTableID_PBConfig [CAN_MAXTABLEID]= {
	{0x**4001fffe**, /* CanRxFifoTable_3 of type Extended and formatA for FlexCAN_A */
	0xfffffffe }}

Autosar 4.0 Requirement	NA

Note

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

3.11.4.3 CanIDValue1

Table 3-105. CanIDValue1

Description	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 Format C - One 8 most significant bit value of Standard or Extended ID.
Class	Implementation Specific Parameter
Range	0 - 2047 for Format B– Standard CanTableIDType type 0 – 1683 for Format B – Extended CanTableIDType type 0 – 255 for Format C
Default	0xF0
Source File	Can_Cfg.c, Can_PBCfg.c

Source Representation	Format C
	CONST(Can_RxFiFoTableIdConfigType, CAN_CONST) RxFifoTableID_PCConfig[CAN_MAXTABLEID_0] = {
	{0x11**12**1314, /* CanRxFifoTable_0 of formatC for FlexCAN_A */
	0xffffffff }}
	Format B
	CONST(Can_RxFiFoTableIdConfigType, CAN_CONST) RxFifoTableID_PBConfig [CAN_MAXTABLEID]= {
	{ {0x1ff8**07f8**, /* CanRxFifoTable_0 of type Standard and formatB for FlexCAN_A */
	0xfff8fff8 }}

Autosar 4.0 Requirement	NA

Note

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

3.11.4.4 CanIDValue2

Table 3-106. CanIDValue2

Description	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.
Class	Implementation Specific Parameter
Range	0 - 255
Default	0xF0

Table continues on the next page...

Table 3-106. CanIDValue2 (continued)

Source File	Can_Cfg.c, Can_PBCfg.c

Source Representation	Format C
	CONST(Can_RxFiFoTableIdConfigType, CAN_CONST) RxFifoTableID_PBConfig [CAN_MAXTABLEID]= {
	{0x1112**13**14, /* CanRxFifoTable_0 of formatC for FlexCAN_A */
	0xffffffff }}

Autosar 4.0 Requirement	NA

Note

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

3.11.4.5 CanIDValue3

Table 3-107. CanIDValue3

Description	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.
Class	Implementation Specific Parameter
Range	0 - 255
Default	0xF0
Source File	Can_Cfg.c, Can_PBCfg.c

Source Representation	Format C
	CONST(Can_RxFiFoTableIdConfigType, CAN_CONST) RxFifoTableID_PBConfig[CAN_MAXTABLEID]= {
	{0x111213**14**, /* CanRxFifoTable_0 of formatC for FlexCAN_A */
	0xffffffff } }

Autosar 4.0 Requirement	NA

Note

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

3.11.4.6 CanMBFilterMaskValue

Table 3-108. CanMBFilterMaskValue

Description	Specifies filter mask value to be used as acceptance criteria for the ID Table.

Table continues on the next page...

Table 3-108. CanMBFilterMaskValue (continued)

	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).
Class	Implementation Specific Parameter
Range	0 – 2047 for Format A Standard CanTableIDType 0 - 536870911 Format A Extended CanTableIDType 0 – 268435455 for Format B Extended CanTableIDType 0 - 4194303 for Format B Standard CanTableIDType 0 – 4294967295 for Format C
Default	2047
Source File	Can_Cfg.c, Can_PBCfg.c

Source Representation	CONST(Can_RxFiFoTableIdConfigType, CAN_CONST) RxFifoTableID_PBConfig[CAN_MAXTABLEID]= {
	{0x11121314, /* CanRxFifoTable_0 of formatC for FlexCAN_A */,
	0xFFFFFFFF } }

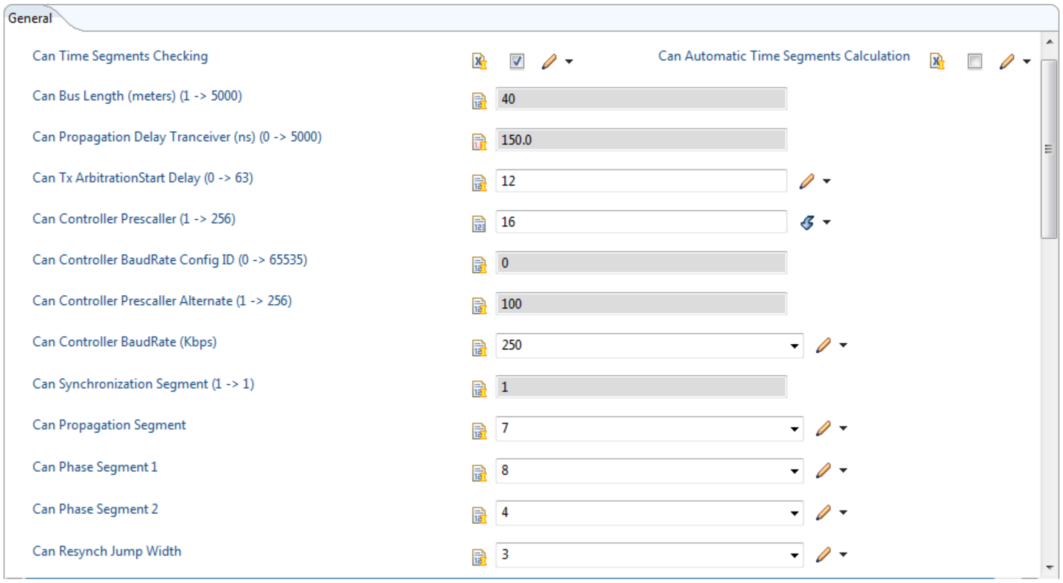
Autosar 4.0 Requirement	NA

3.11.4.7 CanTableIDType

Table 3-109. CanTableIDType

Description	Specifies whether extended or standard frames are accepted into the FIFO.
Class	Implementation Specific Parameter
Range	Standard, Extended
Default	Standard
Source File	Can_Cfg.c, Can_PBCfg.c
Source Representation	NA
Autosar 4.0 Requirement	NA

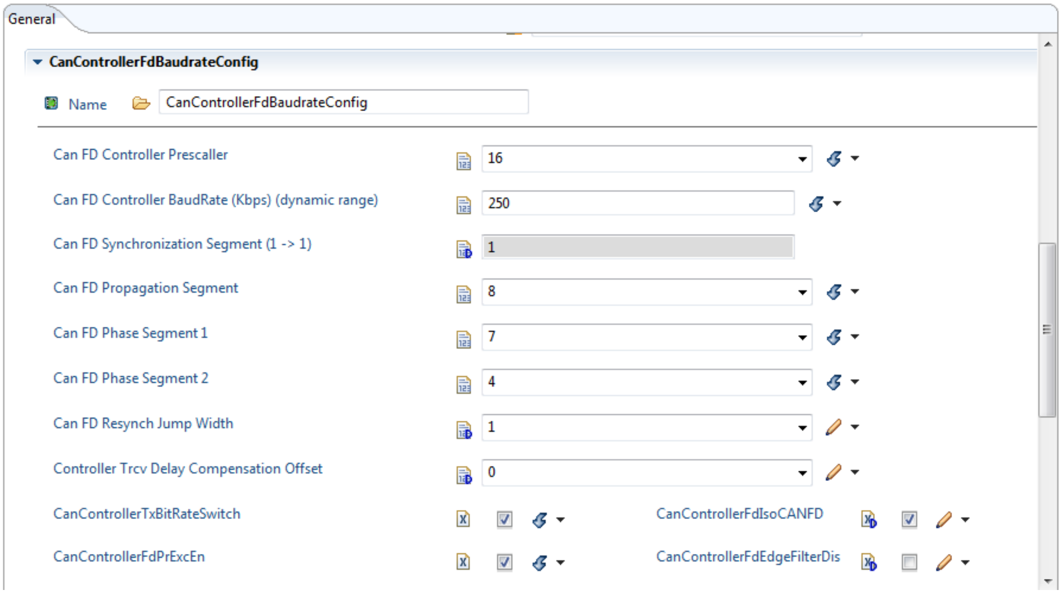
3.11.5 CanControllerBaudrateConfig



General

Can Time Segments Checking	☑
Can Automatic Time Segments Calculation	□
Can Bus Length (meters) (1 -> 5000)	40
Can Propagation Delay Tranceiver (ns) (0 -> 5000)	150.0
Can Tx ArbitrationStart Delay (0 -> 63)	12
Can Controller Prescaller (1 -> 256)	16
Can Controller BaudRate Config ID (0 -> 65535)	0
Can Controller Prescaller Alternate (1 -> 256)	100
Can Controller BaudRate (Kbps)	250
Can Synchronization Segment (1 -> 1)	1
Can Propagation Segment	7
Can Phase Segment 1	8
Can Phase Segment 2	4
Can Resynch Jump Width	3

Figure 3-15. CanControllerBaudrateConfig



General

CanControllerFdBaudrateConfig

Name: CanControllerFdBaudrateConfig

Can FD Controller Prescaller	16
Can FD Controller BaudRate (Kbps) (dynamic range)	250
Can FD Synchronization Segment (1 -> 1)	1
Can FD Propagation Segment	8
Can FD Phase Segment 1	7
Can FD Phase Segment 2	4
Can FD Resynch Jump Width	1
Controller Trcv Delay Compensation Offset	0
CanControllerTxBitRateSwitch	☑
CanControllerFdIsoCANFD	☑
CanControllerFdPrExcEn	☑
CanControllerFdEdgeFilterDis	□

Figure 3-16. 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.

3.11.5.1 CanControllerCheckCanStandard

Table 3-110. CanControllerCheckCanStandard

Description	If enabled, Can Plugin checks that CanControllerPropSeg, CanControllerSeg1, CanControllerSeg2 and CanSyncJumpWidth settings match the CAN Standard Compliant Bit Time Segment Settings.
Class	Implementation Specific Parameter.
Range	True, False
Default	True
Source File	NA
Source Representation	NA
Autosar 4.0 Requirement	NA

Note

This parameter is used in the Can_Cfg.c and Can_PBcfg.c files validating time segments values if it is enabled.

3.11.5.2 CanAdvancedSetting

Table 3-111. CanAdvancedSetting

Description	If True initiates the derivation of the Can bit timing values from the CanControllerBaudRate parameter and source clock value.
Class	Implementation Specific Parameter
Range	True, False
Default	False
Source File	Can_Cfg.c, Can_PBcfg.c
Source Representation	NA
Autosar 4.0 Requirement	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".

3.11.5.3 CanBusLength

Table 3-112. CanBusLength

Description	Specifies the Can bus length in meters. This parameter is used for PropSeg calculation when “CanAdvancedSetting” is set to true.
Class	Implementation Specific Parameter
Range	1 – 5000
Default	40
Source File	Can_Cfg.c, Can_PBCfg.c
Source Representation	NA
Autosar 4.0 Requirement	NA

Note

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

3.11.5.4 CanPropDelayTranceiver

Table 3-113. PropagationDelayOfTranceiver

Description	Specifies the propagation delay in ns for the Can transceiver. This parameter is used for PropSeg calculation when “CanAdvancedSetting” is set to true.
Class	Implementation Specific Parameter
Range	1 – 5000
Default	150
Source File	Can_Cfg.c, Can_PBCfg.c
Source Representation	NA
Autosar 4.0 Requirement	NA

Note

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

3.11.5.5 CanControllerPrescaller

Table 3-114. CanControllerPrescaller

Description	Specifies the prescaller for the controller. The calculation of the resulting CanControllerTimeQuanta value depending on module clocking and prescaller shall be done offline. Prescaler = FreqCanClk / FreqTq; FreqTq = 1 / CanControllerTimeQuanta .
Class	Implementation Specific Parameter
Range	1 - 256

Table continues on the next page...

Table 3-114. CanControllerPrescaller (continued)

Default	100
Source File	Can_Cfg.c, Can_PBCfg.c
Source Representation	NA
Autosar 4.0 Requirement	NA

3.11.5.6 CanControllerPrescaller_Alt

Table 3-115. CanControllerPrescaller_Alt

Description	Specifies the alternate prescaller for the controller .The calculation of the resulting CanControllerTimeQuanta_Alternate value depending on module clocking and prescaller shall be done offline.Prescaler = FreqCanClk / FreqTq; FreqTq = 1 / CanControllerTimeQuanta .
Class	Implementation Specific Parameter
Range	1 - 256
Default	100
Source File	Can_Cfg.c, Can_PBCfg.c
Source Representation	NA
Autosar 4.0 Requirement	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.

3.11.5.7 CanControllerBaudRate

Table 3-116. CanControllerBaudRate

Description	Specifies the buadrate of the controller in kbps.
Class	Autosar Parameter
Range	0 - 1000
Default	20
Source File	Can_Cfg.c, Can_PBCfg.c
Source Representation	NA
Autosar 4.0 Requirement	CAN005_Conf

3.11.5.8 CanControllerPropSeg

Table 3-117. CanControllerPropSeg

Description	Propogation delay in time quanta.
Class	Autosar Parameter
Range	1 - 8
Default	5
Source File	Can_Cfg.c, Can_PBCfg.c

Source Representation	CONST(Can_ControllerBaudrateConfigType, CAN_CONST) ControllerBaudrateConfigs_PC[CAN_MAXCONTROLLERCOUNT] = {
	....
	/* ===== Control Register - CTRL ===== */
	....
	/* CTRL[PROPSEG] - Propagation segment */
	5U ,
	....
	}

Autosar 4.0 Requirement	CAN073_Conf

Note

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

3.11.5.9 CanControllerSeg1

Table 3-118. CanControllerSeg1

Description	Specifies Phase Segment 1
Class	Autosar Parameter
Range	1 - 8
Default	5
Source File	Can_Cfg.c, Can_PBCfg.c

Source Representation	CONST(Can_ControllerBaudrateConfigType, CAN_CONST) ControllerBaudrateConfigs_PC[CAN_MAXCONTROLLERCOUNT] = {
	....
	/* ===== Control Register - CTRL ===== */
	....
	/* CTRL[PSEG1] - Segment 1 */
	5U ,
	....
	}

Autosar 4.0 Requirement	CAN074_Conf

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.

3.11.5.10 CanControllerSeg2

Table 3-119. CanControllerSeg2

Description	Specifies Phase Segment 2
Class	Autosar Parameter
Range	2 - 8
Default	6
Source File	Can_Cfg.c, Can_PBCfg.c

Source Representation	CONST(Can_ControllerBaudrateConfigType, CAN_CONST) ControllerBaudrateConfigs_PC[CAN_MAXCONTROLLERCOUNT] = {
	....
	/* ===== Control Register - CTRL ===== */
	....
	/* CTRL[PSEG2] - Segment 2 */
	6U ,
	....
	}

Autosar 4.0 Requirement	CAN075_Conf

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.

3.11.5.11 CanSyncJumpWidth

Table 3-120. CanSyncJumpWidth

Description	Specifies Synchronization Jump Width.
Class	Implementation Specific Parameter
Range	1 - 4
Default	0
Source File	Can_Cfg.c, Can_PBCfg.c
Source Representation	CONST(Can_ControllerBaudrateConfigType, CAN_CONST) ControllerBaudrateConfigs_PC[CAN_MAXCONTROLLERCOUNT] = {

Table continues on the next page...

Table 3-120. CanSyncJumpWidth (continued)

	.... /* ===== Control Register - CTRL ===== */ .... /* CTRL[RJW] - Resynchronization Jump Width */ .... }
Autosar 4.0 Requirement	CAN383_Conf

Note

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

3.11.5.12 CanControllerFdBaudrateConfig

Table 3-121. CanControllerFdBaudrateConfig

Description	Enable or disable config for CAN_FD and also FD mode
Class	Implementation Specific Parameter
Range	True, False
Default	False
Source File	Can_Cfg.c, Can_PBCfg.c

Source Representation	static CONST(Can_ControllerBaudrateConfigType, CAN_CONST) ControllerBaudrateConfigs_PC_0[CAN_CFGSET0_MAX_BAUDRATE_FC_A_0] = {
	#if (CAN_FD_MODE_ENABLE == STD_ON){
	/*true;*/
	....
	....
	}

Autosar 4.0 Requirement	NA

Note

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

3.11.5.13 CanControllerFdBaudRate

Table 3-122. CanControllerFdBaudRate

Description	Specifies the data segment baud rate of the controller in kbps.
Class	Autosar Parameter

Table continues on the next page...

Table 3-122. CanControllerFdBaudRate (continued)

Range	0 - 8000
Default	0
Source File	Can_Cfg.c, Can_PBCfg.c

Source Representation	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 */
	....
	}

Autosar 4.0 Requirement	NA

3.11.5.14 CanControllerPropSeg

Table 3-123. CanControllerPropSeg

Description	Specifies propagation delay in time quantas.
Class	Autosar Parameter
Range	0 - 255
Default	0
Source File	Can_Cfg.c, Can_PBCfg.c

Source Representation	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 << (IPV_FlexCAN_FD_PROPSEG_OFFSET))
	....
	}

Autosar 4.0 Requirement	NA

Note

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

3.11.5.15 CanControllerFDPrescaller

Table 3-124. CanControllerFDPrescaller

Description	Specifies the prescaller for the controller in FD mode.The calculation of the resulting CanControllerTimeQuanta value depending on module clocking and prescaller shall be done offline. Prescaler = FreqCanClk / FreqTq; FreqTq = 1 / CanControllerTimeQuanta .
Class	Implementation Specific Parameter
Range	1 - 256
Default	100
Source File	Can_Cfg.c, Can_PBCfg.c

Source Representation	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 << IPV_FlexCAN_FD_PROPSEG_OFFSET) |
	....
	}

Autosar 4.0 Requirement	NA

3.11.5.16 CanControllerSeg1

Table 3-125. CanControllerSeg1

Description	Specifies phase segment 1 in time quantas.
Class	Autosar Parameter
Range	0 - 255
Default	0
Source File	Can_Cfg.c, Can_PBCfg.c

Source Representation	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 << IPV_FlexCAN_FD_PSEG1_OFFSET) |
	....
	}

Autosar 4.0 Requirement	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.

3.11.5.17 CanControllerSeg2

Table 3-126. CanControllerSeg2

Description	Specifies phase segment 2 in time quantas.
Class	Autosar Parameter
Range	0 - 255
Default	0
Source File	Can_Cfg.c, Can_PBCfg.c

Source Representation	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<<IPV_FlexCAN_FD_PSEG2_OFFSET) |
	....
	}

Autosar 4.0 Requirement	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.

3.11.5.18 CanControllerSyncJumpWidth

Table 3-127. CanControllerSyncJumpWidth

Description	Specifies the synchronization jump width for the controller in time quantas.
Class	Implementation Specific Parameter
Range	0 - 255
Default	0
Source File	Can_Cfg.c, Can_PBCfg.c
Source Representation	static CONST(Can_ControllerBaudrateConfigType, CAN_CONST) ControllerBaudrateConfigs_PC_0[CAN_CFGSET0_MAX_BAUDRATE_FC_A_0] = { ....

Table continues on the next page...

Table 3-127. CanControllerSyncJumpWidth (continued)

	#if (CAN_FD_MODE_ENABLE == STD_ON){
	....
	(uint32)(0U<<IPV_FlexCAN_FD_SJW_OFFSET) , /*Sync jump width*/
	....
	}

Autosar 4.0 Requirement	NA

Note

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

3.11.5.19 CanControllerTrcvDelayCompensationOffset

Table 3-128. CanControllerTrcvDelayCompensationOffset

Description	Specifies the Transceiver Delay Compensation Offset in ns. If not specified Transceiver Delay Compensation is disabled.
Class	Implementation Specific Parameter
Range	0 - 400
Default	0
Source File	Can_Cfg.c, Can_PBCfg.c

Source Representation	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*/
	....
	}

Autosar 4.0 Requirement	Define in ASR 4.2.1 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.

3.11.5.20 CanControllerTxBitRateSwitch

Table 3-129. CanControllerTxBitRateSwitch

Description	CanControllerTxBitRateSwitch it is used to enable a feature, which can switch baudrate transmition 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.
Class	Implementation Specific Parameter.
Range	True, False
Default	True
Source File	NA

Source Representation	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<<IPV_FlexCAN_FD_BRS_OFFSET) /*false
	....
	}

Autosar 4.0 Requirement	NA

Note

This parameter is used in the Can_Cfg.c and Can_PBcfg.c files validating time segments values if it is enabled.

3.11.5.21 CanControllerCbtEnable

Table 3-130. CanControllerCbtEnable

Description	CanControllerTxBitRateSwitch it is used to enable FD data baudrate (which has a different value comparing with the nominal baudrate).
Class	Implementation Specific Parameter.
Range	True, False
Default	True
Source File	NA

Source Representation	static CONST(Can_ControllerBaudrateConfigType, CAN_CONST) ControllerBaudrateConfigs_PC_0[CAN_CFGSET0_MAX_BAUDRATE_FC_A_0] = {
	....
	#if (CAN_CBT_ENABLE == STD_ON){
	....
	(uint32)(1U <<FLEXCAN_CBT_OFFSET),
	....
	}

Table continues on the next page...

Table 3-130. CanControllerCbtEnable (continued)

Autosar 4.0 Requirement	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 fiels. 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.

3.11.5.22 CanControllerBaudRate

Table 3-131. CanControllerBaudRate

Description	Specifies the buadrate of the controller in kbps.
Class	Implementation Specific Parameter.
Range	0 - 16000
Default	0
Source File	Can_Cfg.c, Can_PBCfg.c

Source Representation	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 */
	....
	}

Autosar 4.0 Requirement	NA

3.11.5.23 CanControllerCbtPropSeg

Table 3-132. CanControllerCbtPropSeg

Description	Specifies propagation delay in time quantas.
Class	Autosar Parameter
Range	1 - 64
Default	5
Source File	Can_Cfg.c, Can_PBCfg.c

Table continues on the next page...

Table 3-132. CanControllerCbtPropSeg (continued)

Source Representation	static CONST(Can_ControllerBaudrateConfigType, CAN_CONST) ControllerBaudrateConfigs_PC_0[CAN_CFGSET0_MAX_BAUDRATE_FC_A_0] = {
	....
	#if (CAN_CBT_ENABLE == STD_ON){
	....
	(uint32)(0U << FLEXCAN_CBT_PROPSEG_OFFSET) |
	....
	}

Autosar 4.0 Requirement	CAN073_Conf

Note

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

3.11.5.24 CanControllerCbtPrescaller

Table 3-133. CanControllerCbtPrescaller

Description	The calculation of the resulting CanControllerTimeQuanta value depending on module clocking and prescaller shall be done offline. Prescaler = FreqCanClk / FreqTq; FreqTq = 1 / CanControllerTimeQuanta .
Class	Implementation Specific Parameter
Range	1 - 1023
Default	100
Source File	Can_Cfg.c, Can_PBCfg.c

Source Representation	static CONST(Can_ControllerBaudrateConfigType, CAN_CONST) ControllerBaudrateConfigs_PC_0[CAN_CFGSET0_MAX_BAUDRATE_FC_A_0] = {
	....
	#if (CAN_CBT_ENABLE == STD_ON){
	....
	(uint32)(99U << FLEXCAN_FD_PRESDIV_CBT_OFFSET).|,
	....
	}

Autosar 4.0 Requirement	NA

3.11.5.25 CanControllerCbtSeg1

Table 3-134. CanControllerCbtSeg1

Description	Specifies phase segment 1 in time quantas.
Class	Autosar Parameter
Range	1 - 32
Default	0
Source File	Can_Cfg.c, Can_PBCfg.c

Source Representation	static CONST(Can_ControllerBaudrateConfigType, CAN_CONST) ControllerBaudrateConfigs_PC_0[CAN_CFGSET0_MAX_BAUDRATE_FC_A_0] = {
	....
	#if (CAN_CBT_ENABLE == STD_ON){
	....
	(uint32)(0U <<FLEXCAN_CBT_PSEG1_OFFSET) |
	....
	}

Autosar 4.0 Requirement	CAN074_Conf

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.

3.11.5.26 CanControllerCbtSeg2

Table 3-135. CanControllerCbtSeg2

Description	Specifies phase segment 2 in time quantas.
Class	Autosar Parameter
Range	1 - 32
Default	0
Source File	Can_Cfg.c, Can_PBCfg.c

Source Representation	static CONST(Can_ControllerBaudrateConfigType, CAN_CONST) ControllerBaudrateConfigs_PC_0[CAN_CFGSET0_MAX_BAUDRATE_FC_A_0] = {
	....
	#if (CAN_CBT_ENABLE == STD_ON){
	....
	(uint32)(0U<<FLEXCAN_CBT_PSEG2_OFFSET) | /*Time segment 2 */
	....
	}

Autosar 4.0 Requirement	CAN075_Conf

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.

3.11.5.27 CanControllerSyncJumpWidthCbt

Table 3-136. CanControllerSyncJumpWidthCbt

Description	Specifies the synchronization jump width for the controller in time quantas.
Class	Implementation Specific Parameter
Range	1 - 16
Default	0
Source File	Can_Cfg.c, Can_PBCfg.c

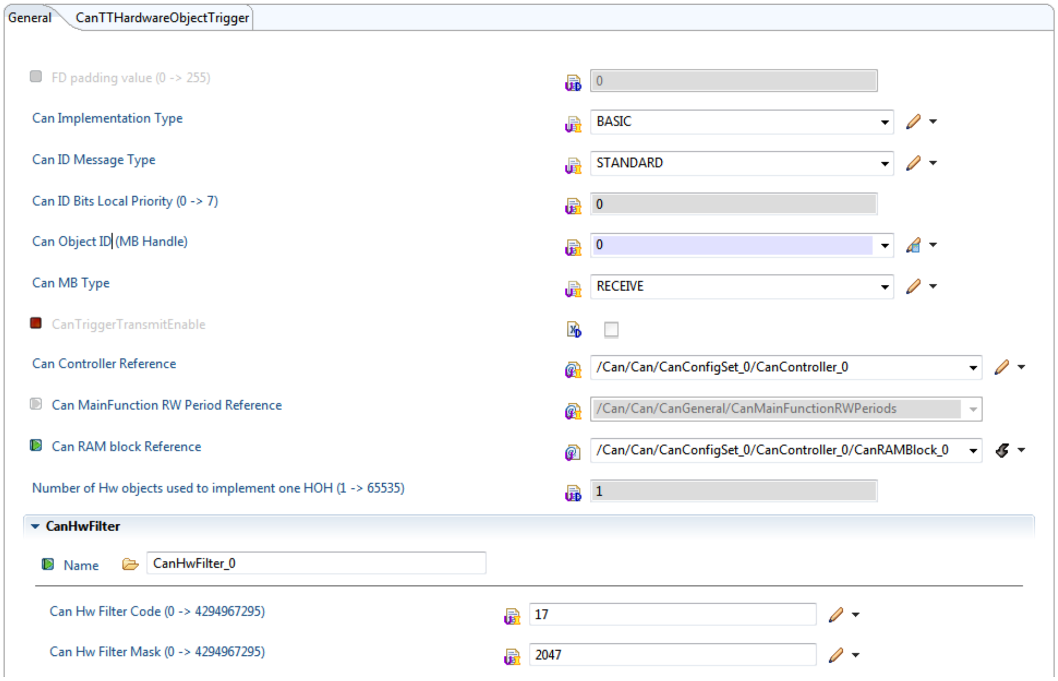
Source Representation	static CONST(Can_ControllerBaudrateConfigType, CAN_CONST) ControllerBaudrateConfigs_PC_0[CAN_CFGSET0_MAX_BAUDRATE_FC_A_0] = {
	....
	#if (CAN_CBT_ENABLE == STD_ON){
	....
	(uint32)(0U<<FLEXCAN_CBT_SJW_OFFSET) /*Sync jump width*/
	....
	}

Autosar 4.0 Requirement	NA

Note

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

3.11.6 Can HardwareObject



FD padding value (0 -> 255)	0
Can Implementation Type	BASIC
Can ID Message Type	STANDARD
Can ID Bits Local Priority (0 -> 7)	0
Can Object ID (MB Handle)	0
Can MB Type	RECEIVE
CanTriggerTransmitEnable	□
Can Controller Reference	/Can/Can/CanConfigSet_0/CanController_0
Can MainFunction RW Period Reference	/Can/Can/CanGeneral/CanMainFunctionRWPeriods
Can RAM block Reference	/Can/Can/CanConfigSet_0/CanController_0/CanRAMBlock_0
Number of Hw objects used to implement one HOH (1 -> 65535)	1

CanHwFilter

Name: CanHwFilter_0

Can Hw Filter Code (0 -> 4294967295)	17
Can Hw Filter Mask (0 -> 4294967295)	2047

Figure 3-17. 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 56.

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 MPC574XG, Flexcan A has 96 MBs. If you using this controller with FIFO, they can not configure up to 96 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 MPC574XG, FlexCan A has 96 MBs. However, if using FD with 64 bytes of data, you can only configure 7 HOHs per ram block.

3.11.6.1 CanFdPaddingValue

Table 3-137. CanFdPaddingValue

Description	This value it is the padding value when FD it is used.
Class	Implementation Specific Parameter
Range	0 - 255
Default	0
Source File	Can_Cfg.c,Can_PBcfg.c

Source Representation	static CONST(Can_MBConfigObjectType, CAN_CONST) MessageBufferConfigs0_PB[CAN_MAXMBCOUNT_0] = {
	....
	#if (CAN_FD_MODE_ENABLE == STD_ON){
	....
	(uint8)0x0, /**< @brief Padding value for MB > 8 bytes */
	....
	}

Autosar 4.0 Requirement	Define in ASR 4.2.1 ECUC_CAN_00485CAN326_Conf

Note

Holds the handle ID of HRH or HTH.

3.11.6.2 CanHandleType

Table 3-138. CanHandleType

Description	Specifies the type (Full-Can or Basic-Can) of the hardware object.
Class	Autosar Parameter
Range	BASIC, FULL
Default	BASIC
Source File	Can_Cfg.c, Can_PBCfg.c
Source Representation	CONST(Can_MBConfigObjectType, CAN_CONST) MessageBufferConfigs_PC[CAN_MAXMBCOUNT_0] = { /* MessageBufferConfigs_PB[0], "CanA0_RX0" * /* uIdMaskIndex */ 0U, /* ControllerId - based on the order from CanController list */ 0U, /* ID type: EXTENDED, STANDARD, MIXED */ STANDARD,

Table continues on the next page...

Table 3-138. CanHandleType (continued)

	/* Receive/Transmit MB configuration */ RECEIVE, /* MessageId */ 0x1U, /* Local priority bits used for arbitration */ 0U,
Autosar 4.0 Requirement	CAN323_Conf

3.11.6.3 CanIdType

Table 3-139. CanIdType

Description	Specifies whether the IdValue is of type: standard identifier, extended identifier, mixed mode.
Class	Autosar Parameter
Range	Standard, Mixed, Extended
Default	Standard
Source File	Can_Cfg.c, Can_PBCfg.c
Source Representation	CONST(Can_MBConfigObjectType, 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,
Autosar 4.0 Requirement	CAN065_Conf

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).

3.11.6.4 CanIdValue

Table 3-140. CanIdValue

Description	Specifies (together with the filter mask) the identifiers that pass the hardware filter for of RX objects.
Class	Autosar Parameter
Range	0 .. 4294967295
Default	0
Source File	Can_Cfg.c, Can_PBCfg.c
Source Representation	CONSTCONST(Can_MBConfigObjectType, 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,
Autosar 4.0 Requirement	CAN325_Conf

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.

3.11.6.5 CanMBPrio

Table 3-141. CanMBPrio

Description	This field is used when MCR[LPRIO_EN] is set and makes sense only for Tx MBs.
Class	Implementation Specific Parameter

Table continues on the next page...

Table 3-141. CanMBPrio (continued)

Range	0 - 7
Default	0
Source File	Can_Cfg.c, Can_PBCfg.c
Source Representation	CONST(Can_MBConfigObjectType, 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**,
Autosar 4.0 Requirement	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.

3.11.6.6 CanObjectId

Table 3-142. CanObjectId

Description	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
Class	Autosar Parameter
Range	0 .. 65535
Default	0
Source File	Can_Cfg.h
Source Representation	#define CanA0_RX0 0U /* RECEIVE object */
Autosar 4.0 Requirement	CAN326_Conf

Note

Holds the handle ID of HRH or HTH.

3.11.6.7 CanObjectType

Table 3-143. CanObjectType

Description	Specifies if the HardwareObject is used as Transmit or as Receive object
Class	Autosar Parameter
Range	Transmit, Receive
Default	Receive
Source File	Can_Cfg.c, Can_PBCfg.c
Source Representation	CONST(Can_MBConfigObjectType, 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,
Autosar 4.0 Requirement	CAN327_Conf

Note

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

3.11.6.8 CanControllerRef

Table 3-144. CanControllerRef

Description	This associates the hardware object to the CAN controller that uses this hardware object.
Class	Autosar Parameter
Range	Integer
Default	0

Table continues on the next page...

Table 3-144. CanControllerRef (continued)

Source File	Can_Cfg.c, Can_PBCfg.c
Source Representation	CONST(Can_MBConfigObjectType, 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,
Autosar 4.0 Requirement	CAN322_Conf

Note

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

3.11.6.9 CanRAMBlockRef

Table 3-145. CanRAMBlockRef

Description	Reference to RAM block which the HOH is associated to in the CAN FD mode.
Class	Autosar Parameter
Range	NA
Default	NA
Container Name	CanRAMBlock
Autosar 4.0 Requirement	NA

3.11.6.10 CanMainFunctionRWPeriodRef

Table 3-146. CanMainFunctionRWPeriodRef

Description	Reference to CAN Controller to which the HOH is associated to.
Class	Autosar Parameter

Table continues on the next page...

Table 3-146. CanMainFunctionRWPeriodRef (continued)

Range	NA
Default	NA
Container Name	CanMainFunctionRWPeriods{CAN_MAIN_FUNCTION_RWPERIODS}
Autosar 4.0 Requirement	CAN437_Conf

Note

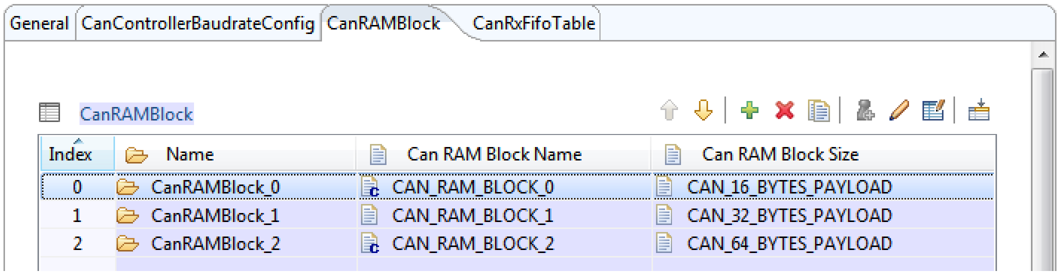
This parameter describes the period for cyclic call to Can_MainFunction_Write and Can_MainFunction_Read . Unit is seconds. Different poll-cycles will be configurable if more than one CanMainFunctionWritePeriod or Can_MainFunction_ReadPeriod are configured. In this case multiple Can_MainFunction_Write()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

3.11.7 CanRAMBlock

When CAN FD is enabled, the FlexCAN RAM can be partitioned in the three blocks of 512 bytes. Each block can accommodate a number of Message Buffers which depends on the configuration provided by the CanRAMBlockSizeValue parameters in the container CanRAMBlock.

Table 3-147. CanRAMBlock

Container Name	CanRAMBlock
Description	This container contains the configuration (parameters) of RAM Message Buffer Data Size.
Class	NA
Autosar 4.2 Requirement	NA



Index	Name	Can RAM Block Name	Can RAM Block Size
0	CanRAMBlock_0	CAN_RAM_BLOCK_0	CAN_16_BYTES_PAYLOAD
1	CanRAMBlock_1	CAN_RAM_BLOCK_1	CAN_32_BYTES_PAYLOAD
2	CanRAMBlock_2	CAN_RAM_BLOCK_2	CAN_64_BYTES_PAYLOAD

Figure 3-18. CanRAMBlock

3.11.7.1 CanRAMBlockName

Table 3-148. CanRAMBlockName

Description	Sets the index of the configured RAM block.
Class	NA
Range	CAN_RAM_BLOCK_0, CAN_RAM_BLOCK_1, CAN_RAM_BLOCK_2
Default	NA
Source File	NA
Source Representation	NA
Autosar 4.2 Requirement	NA

3.11.7.2 CanRAMBlockSizeValue

Table 3-149. CanRAMBlockSizeValue

Description	Specify Message Buffer Data Size for each RAM block.
Class	NA
Range	CAN_8_BYTES_PAYLOAD, CAN_16_BYTES_PAYLOAD, CAN_32_BYTES_PAYLOAD, CAN_64_BYTES_PAYLOAD
Default	CAN_8_BYTES_PAYLOAD
Source File	Can_PBcfg.c

Source Representation	`static`
	`constCan_ControlerDescriptorTypeControlerDescriptors0_PB[CAN_MAXCONTROLLERCOUNT_0] =`
	`{`
	`    /* ControlerDescriptor of CanController_0*/`
	`    {`
	`        ....................`
	`        #if (CAN_FD_MODE_ENABLE == STD_ON)`
	`        /* Message Buffer Data Size for three RAM regions */`
	`        (uint32)(`
	`                  ((uint32)CAN_16_BYTES_PAYLOAD_U32 <<(uint32)FLEXCAN_MBDSR0_OFFSET_U8) |`
	`                  ((uint32)CAN_32_BYTES_PAYLOAD_U32 <<(uint32)FLEXCAN_MBDSR1_OFFSET_U8) |`
	`                  ((uint32)CAN_64_BYTES_PAYLOAD_U32 <<(uint32)FLEXCAN_MBDSR2_OFFSET_U8)`
	`                ),`
	`        #endif`
	`        ....................`
	`    },`
	`    ....................`
	`};`

Autosar 4.2 Requirement	NA

3.11.8 CanIcom

Table 3-150. CanIcom

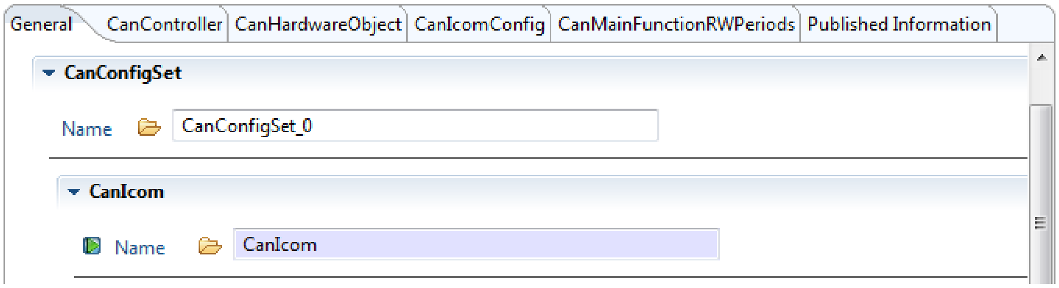
Container Name	CanIcom
Description	This container contains the parameters for configuring pretended networking.
Class	Autosar container
Autosar 4.2 Requirement	ECUC_Can_00440

Table 3-151. CanIcom Included Containers

Included Containers

Table 3-151. CanIcom Included Containers

Container Name	**Description**
CanIcomConfig	This container contains the parameters for configuring pretended networking.



Name: CanConfigSet_0

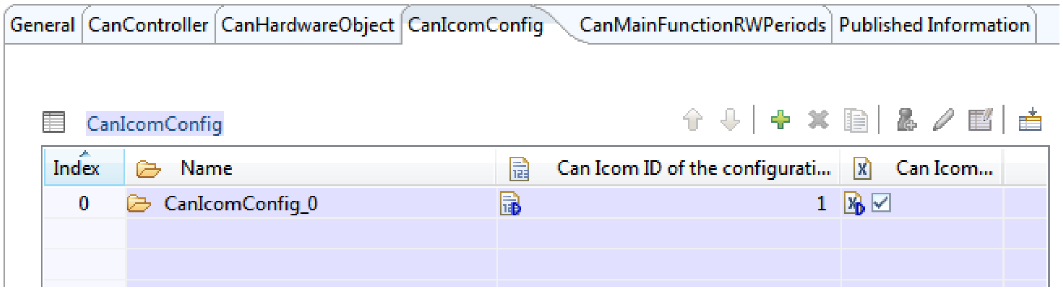
Name: CanIcom

Figure 3-19. CanIcom

3.11.9 CanIcomConfig

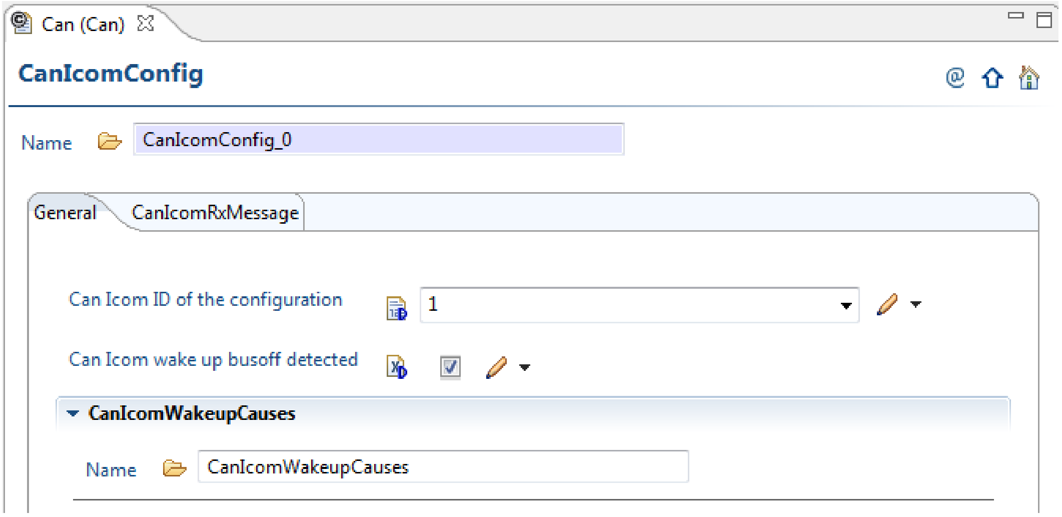
Table 3-152. CanIcomConfig

Container Name	CanIcomConfig
Description	This container contains the general configuration parameters of the ICOM Configuration.
Class	Autosar container
Autosar 4.2 Requirement	ECUC_Can_00459



Index	Name	Can Icom ID of the configurati...	Can Icom...
0	CanIcomConfig_0	1	☑

Figure 3-20. CanIcomConfig



Can (Can)

CanIcomConfig

Name CanIcomConfig_0

Can Icom ID of the configuration 1

Can Icom wake up busoff detected ☑

CanIcomWakeupCauses

Name CanIcomWakeupCauses

Figure 3-21. CanIcomConfig Parameters

3.11.9.1 CanIcomConfigId

Table 3-153. CanIcomConfigId

Description	This parameter identifies the ID of the ICOM configuration.
Class	Autosar Parameter
Range	1 .. 255
Default	NA
Source File	Can_Cfg.c

Source Representation	`static const Can_IcomConfigsType Can_PnConfig[1] =`
	`{`
	`    {`
	`        /*value for node CanIcomConfigId */`
	`        (uint8)1U,`
	`        /*value for node CanIcomWakeOnBusOff*/`
	`        (boolean)TRUE,`
	`        /*u8NumberCanIcomRxMessage */`
	`        (uint8)1U,`
	`        /*pCanIcomRxMessageConfigs*/`
	`        &Can_PnConfig_0_Rx[0U]`
	`    }`
	`};`

Table continues on the next page...

Table 3-153. CanIcomConfigId (continued)

Autosar 4.2 Requirement	ECUC_Can_00441

3.11.9.2 CanIcomWakeOnBusOff

Table 3-154. CanIcomWakeOnBusOff

<table>
<tr><td><b>Description</b></td><td>This parameter defines that the MCU shall wake if the bus off is detected or not.</td></tr>
<tr><td><b>Class</b></td><td>Autosar Parameter</td></tr>
<tr><td><b>Range</b></td><td>True, False</td></tr>
<tr><td><b>Default</b></td><td>True</td></tr>
<tr><td><b>Source File</b></td><td>Can_Cfg.c</td></tr>
<tr><td><b>Source Representation</b></td><td><pre>
static const Can_IcomConfigsType Can_PnConfig[1] =
{
    {
        /*value for node CanIcomConfigId */
        (uint8)1U,
        /*value for node CanIcomWakeOnBusOff*/
        (boolean)TRUE,
        /*u8NumberCanIcomRxMessage */
        (uint8)1U,
        /*pCanIcomRxMessageConfigs*/
        &Can_PnConfig_0_Rx[0U]
    }
};
</pre></td></tr>
<tr><td><b>Autosar 4.2 Requirement</b></td><td>ECUC_Can_00442</td></tr>
</table>

3.11.9.3 CanIcomWakeupCauses

Table 3-155. CanIcomWakeupCauses

Description	This container contains the configuration parameters of the wakeup causes to leave the power saving mode..
Class	Autosar container
Source File	NA
Source Representation	NA
Autosar 4.2 Requirement	ECUC_Can_00443

Table 3-156. CanIcomWakeupCauses Included Containers

Included Containers

Table 3-156. CanIcomWakeupCauses Included Containers

Container Name	**Description**

Table continues on the next page...

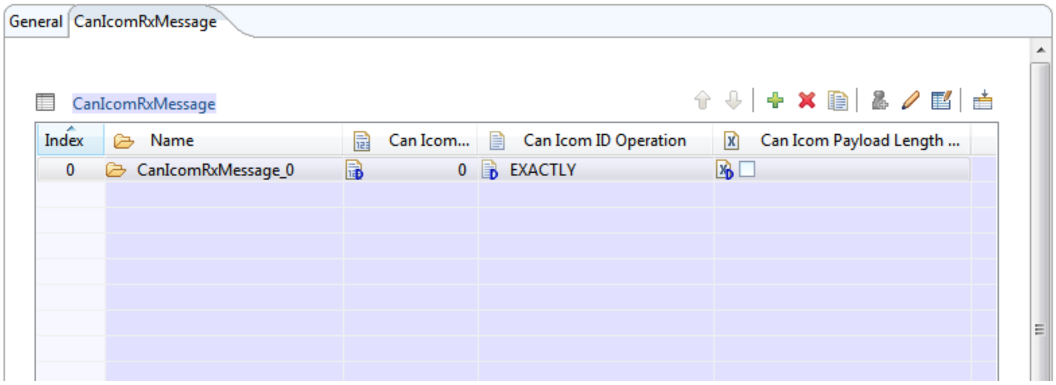
Table 3-156. CanIcomWakeupCauses Included Containers (continued)

CanIcomRxMessage	This container contains the configuration parameters for the wakeup causes for matching received messages. It has to be configured as often as received messages are defined as wakeup cause.constraint: For all CanIcomRxMessage instances the Message IDs which are defined in CanIcomMessageId and in CanIcomRxMessageIdMask shall not overlap.

3.11.9.4 CanIcomRxMessage

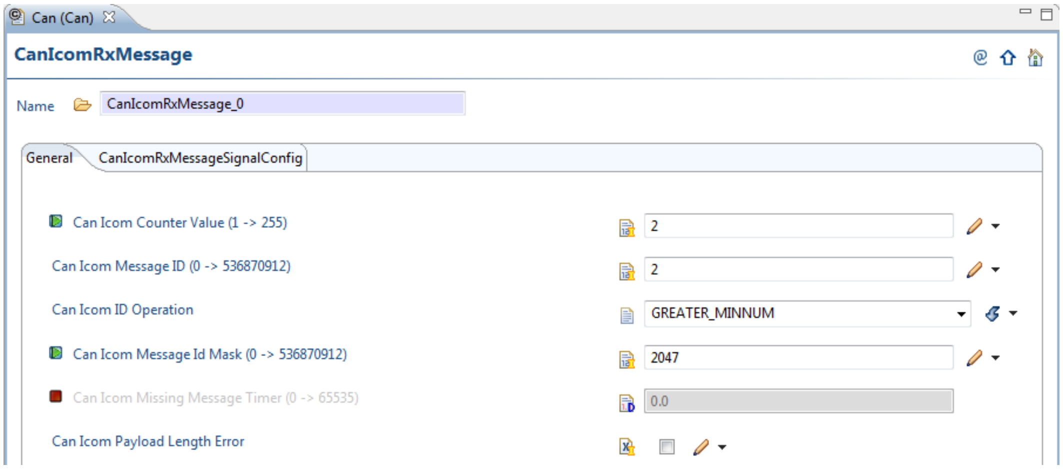
Table 3-157. CanIcomRxMessage

Container Name	CanIcomRxMessage
Description	This container contains the configuration parameters for the wakeup causes for matching received messages. It has to be configured as often as received messages are defined as wakeup cause. constraint: For all CanIcomRxMessage instances the Message IDs which are defined in CanIcomMessageId and in CanIcomRxMessageIdMask shall not overlap.
Class	Autosar container
Autosar 4.2 Requirement	ECUC_Can_00447



Index	Name	Can Icom...	Can Icom ID Operation	Can Icom Payload Length ...
0	CanIcomRxMessage_0	0	EXACTLY	☐

Figure 3-22. CanIcomRxMessage



Name CanIcomRxMessage_0

Can Icom Counter Value (1 -> 255)	2
Can Icom Message ID (0 -> 536870912)	2
Can Icom ID Operation	GREATER_MINNUM
Can Icom Message Id Mask (0 -> 536870912)	2047
Can Icom Missing Message Timer (0 -> 65535)	0.0
Can Icom Payload Length Error	☐

Figure 3-23. CanIcomRxMessage Parameters

3.11.9.4.1 CanIcomCounterValue

Table 3-158. CanIcomCounterValue

Description	This parameter defines that the MCU shall wake if the message with the ID is received n times on the communication channel.
Class	Autosar Parameter
Range	1 .. 65536
Default	NA
Source File	Can_Cfg.h, Can_Cfg.c

Source Representation	**In Can_Cfg.h**
	`#define CAN_PRETENDED_COUNT_MESSAGE (STD_ON)`
	In Can_Cfg.c
	`static constCan_IcomRxMessageConfigsType Can_PnConfig_0_Rx[1]=`
	`{`
	`    {`
	`        `**`/*value for node CanIcomCounterValue */`**
	`        `**`2U,`**
	`        /*value for node CanIcomMessageId*/`
	`        2U,`
	`        /*value for node CanIcomMessageIdMask*/`
	`        2047U,`
	`        /*value for node CanIcomMissingMessageTimerValue*/`
	`        0U ,`
	`        /* Type of ID filtering */`
	`        GREATER_MINNUM,`
	`        /*total number of CanIcomRxSignalMessage defined structures`
	`- u8NumCanIcomRxMessageSignal*/`
	`        (uint8)1U,`
	`        /*pCanIcomRxMessageSignalConfigs*/`
	`        &Can_PnConfig_0_Rx0_Signal[0U]`
	`    }`
	`};`

Autosar 4.2 Requirement	ECUC_Can_00448

3.11.9.4.2 CanIcomIdOperation

Table 3-159. CanIcomIdOperation

Description	This is a non-autosar parameter. It is generated in order support for selection the ID filter type.
Class	NA
Range	EXACTLY, SMALLER_MAXNUM, GREATER_MINNUM, INSIDE_RANGE
Default	EXACTLY
Source File	Can_Cfg.c

Source Representation	`static constCan_IcomRxMessageConfigsType Can_PnConfig_0_Rx[1]=`
	`{`
	`    {`
	`        /*value for node CanIcomCounterValue */`
	`        2U,`
	`        /*value for node CanIcomMessageId*/`
	`        2U,`

Table continues on the next page...

Table 3-159. CanIcomIdOperation (continued)

<table>
<tr><td></td><td>

```
        /*value for node CanIcomMessageIdMask*/
        2047U,
        /*value for node CanIcomMissingMessageTimerValue*/
        0U ,
        /* Type of ID filtering */
        GREATER_MINNUM,
        /*total number of CanIcomRxSignalMessage defined structures
- u8NumCanIcomRxMessageSignal*/
        (uint8)1U,
        /*pCanIcomRxMessageSignalConfigs*/
        &Can_PnConfig_0_Rx0_Signal[0U]
    }
};
```

</td></tr>
<tr><td><b>Autosar 4.2 Requirement</b></td><td>NA</td></tr>
</table>

3.11.9.4.3 CanIcomMessageId

Table 3-160. CanIcomMessageId

<table>
<tr><td><b>Description</b></td><td>This parameter defines the message ID the wakeup causes of this CanIcomRxMessage are configured for. In addition a mask (CanIcomMessageIdMask) can be defined, in that case it is possible to define a range of rx messages, which can create a wakeup condition.</td></tr>
<tr><td><b>Class</b></td><td>Autosar Parameter</td></tr>
<tr><td><b>Range</b></td><td>0 .. 536870912</td></tr>
<tr><td><b>Default</b></td><td>NA</td></tr>
<tr><td><b>Source File</b></td><td>Can_Cfg.c</td></tr>
<tr><td><b>Source Representation</b></td><td>

```
static constCan_IcomRxMessageConfigsType Can_PnConfig_0_Rx[1]=
{
    {
        /*value for node CanIcomCounterValue */
        2U,
        /*value for node CanIcomMessageId*/
        2U,
        /*value for node CanIcomMessageIdMask*/
        2047U,
        /*value for node CanIcomMissingMessageTimerValue*/
        0U ,
        /* Type of ID filtering */
        GREATER_MINNUM,
        /*total number of CanIcomRxSignalMessage defined structures
- u8NumCanIcomRxMessageSignal*/
        (uint8)1U,
        /*pCanIcomRxMessageSignalConfigs*/
        &Can_PnConfig_0_Rx0_Signal[0U]
    }
};
```

</td></tr>
<tr><td><b>Autosar 4.2 Requirement</b></td><td>ECUC_Can_00449</td></tr>
</table>

3.11.9.4.4 CanIcomMessageIdMask

Table 3-161. CanIcomMessageIdMask

Description	Describes a mask for filtering of CAN identifiers. The CAN identifiers of incoming messages are masked with this CanIcomMessageIdMask. If the masked identifier matches the masked value of CanIcomMessageId, it can create a wakeup condition for this CanIcomRxMessage. Bits holding a 0 mean don't care, i.e. do not compare the message's identifier in the respective bit position. The mask shall be build by filling with leading 0.
Class	Autosar Parameter
Range	0 .. 536870912
Default	NA
Source File	Can_Cfg.c

Source Representation	static constCan_IcomRxMessageConfigsType Can_PnConfig_0_Rx[1]=
	{
	    {
	        /*value for node CanIcomCounterValue */
	        2U,
	        /*value for node CanIcomMessageId*/
	        2U,
	        /*value for node CanIcomMessageIdMask*/
	        2047U,
	        /*value for node CanIcomMissingMessageTimerValue*/
	        0U ,
	        /* Type of ID filtering */
	        GREATER_MINNUM,
	        /*total number of CanIcomRxSignalMessage defined structures
	- u8NumCanIcomRxMessageSignal*/
	        (uint8)1U,
	        /*pCanIcomRxMessageSignalConfigs*/
	        &Can_PnConfig_0_Rx0_Signal[0U]
	    }
	};

Autosar 4.2 Requirement	ECUC_Can_00465

3.11.9.4.5 CanIcomMissingMessageTimerValue

Table 3-162. CanIcomMissingMessageTimerValue

Description	This parameter defines that the MCU shall wake if the message with the ID is not received for a specific time in s on the communication channel.
Class	Autosar Parameter
Range	-INF .. INF
Default	NA
Source File	Can_Cfg.h, Can_Cfg.c
Source Representation	**In Can_Cfg.h** #define CAN_PRETENDED_TIMEOUT_CHECK (STD_OFF) **In Can_Cfg.c** static constCan_IcomRxMessageConfigsType Can_PnConfig_0_Rx[1]= { { /*value for node CanIcomCounterValue */

Table continues on the next page...

Table 3-162. CanIcomMissingMessageTimerValue (continued)

<table>
<tr><td></td><td><pre>
        2U,
        /*value for node CanIcomMessageId*/
        2U,
        /*value for node CanIcomMessageIdMask*/
        2047U,
        /*value for node CanIcomMissingMessageTimerValue*/
        0U,
        /* Type of ID filtering */
        GREATER_MINNUM,
        /*total number of CanIcomRxSignalMessage defined structures
- u8NumCanIcomRxMessageSignal*/
        (uint8)1U,
        /*pCanIcomRxMessageSignalConfigs*/
        &Can_PnConfig_0_Rx0_Signal[0U]
    }
};
</pre></td></tr>
<tr><td><b>Autosar 4.2 Requirement</b></td><td>ECUC_Can_00450</td></tr>
</table>

3.11.9.4.6 CanIcomPayloadLengthError

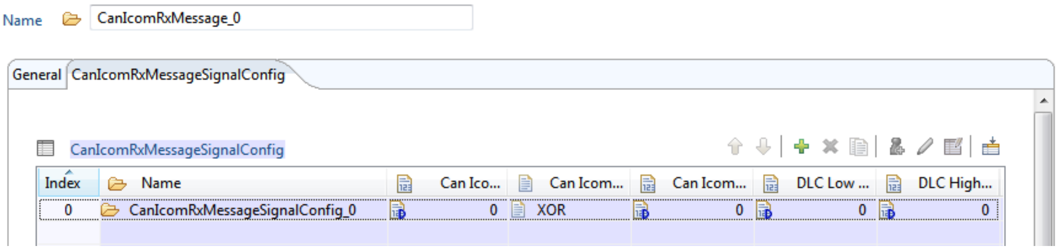
Table 3-163. CanIcomPayloadLengthError

Description	This parameter defines that the MCU shall wake if a payload error occurs.
Class	Autosar Parameter
Range	False
Default	False
Source File	NA
Source Representation	NA
Autosar 4.2 Requirement	ECUC_Can_00451

3.11.9.4.7 CanIcomRxMessageSignalConfig

Table 3-164. CanIcomRxMessageSignalConfig

Description	This container contains the configuration parameters for the wakeup causes for matching signals.It has to be configured as often as a signal is defined as wakeup cause. If at least one Signal conditions defined in a CanIcomRxMessageSignalConfig evaluates to true or if no CanIcomRxMessageSignalConfig are defined, the whole wakeup condition is considered to be true. All instances of this container refer to the same frame/pdu (see CanIcomMessageId).
Class	Autosar container
Autosar 4.2 Requirement	ECUC_Can_00452

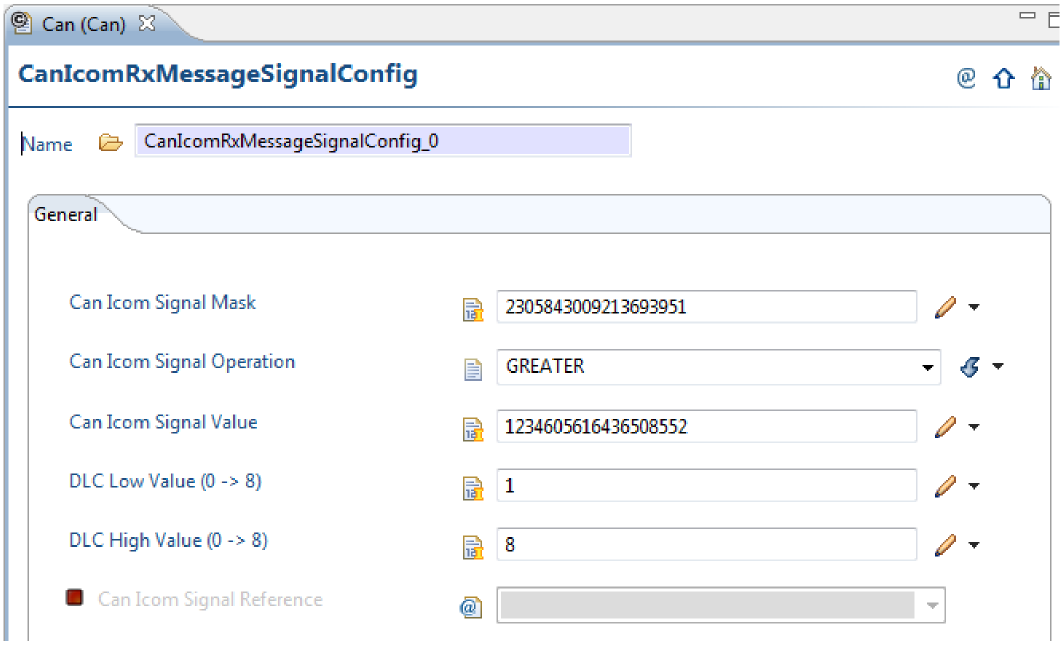


Name CanIcomRxMessage_0

CanIcomRxMessageSignalConfig

Index	Name	Can Ico...	Can Icom...	Can Icom...	DLC Low ...	DLC High...
0	CanIcomRxMessageSignalConfig_0	0	XOR	0	0	0

Figure 3-24. CanIcomRxMessageSignalConfig Elements



Name CanIcomRxMessageSignalConfig_0

Can Icom Signal Mask 2305843009213693951

Can Icom Signal Operation GREATER

Can Icom Signal Value 1234605616436508552

DLC Low Value (0 -> 8) 1

DLC High Value (0 -> 8) 8

Can Icom Signal Reference

Figure 3-25. CanIcomRxMessageSignalConfig Parameters

3.11.9.4.7.1 CanIcomSignalMask

Table 3-165. CanIcomSignalMask

Description	This parameter shall be used to mask a signal in the payload of a CAN message. The mask is binary AND with the signal payload. The result will be used in combination of the operations defined in CanIcomSignalOperation with the CanIcomSignalValue.
Class	Autosar Parameter
Range	0 .. 18446744073709551615
Default	NA
Source File	Can_Cfg.c

Source Representation	`static constCan_IcomRxMessageSignalType Can_PnConfig_0_Rx0_Signal[1] =`
	`{`
	`    {`
	`        /* CanIcomSignalMask */`
	`        (uint64)2305843009213693951U,`
	`        /*CanIcomSignalOperation */`
	`        GREATER,`
	`        /*CanIcomSignalValue */`
	`        (uint64)1234605616436508552U,`

Table continues on the next page...

Table 3-165. CanIcomSignalMask (continued)

	/*DLCLowValue */
	1U,
	/*DLCHighValue */
	8U,
	/*CanIcomSignalRef */
	0U
	}
	};

Autosar 4.2 Requirement	ECUC_Can_00487

3.11.9.4.7.2 CanIcomSignalOperation

Table 3-166. CanIcomSignalOperation

Description	This parameter defines the operation, which shall be used to verify the signal value creates a wakeup condition.
Class	Autosar Parameter
Range	AND, EQUAL, GREATER, SMALLER, XOR
Default	NA
Source File	Can_Cfg.c

Source Representation	static constCan_IcomRxMessageSignalType Can_PnConfig_0_Rx0_Signal[1] =
	{
	{
	/* CanIcomSignalMask */
	(uint64)2305843009213693951U,
	/*CanIcomSignalOperation */
	GREATER,
	/*CanIcomSignalValue */
	(uint64)1234605616436508552U,
	/*DLCLowValue */
	1U,
	/*DLCHighValue */
	8U,
	/*CanIcomSignalRef */
	0U
	}
	};

Autosar Requirement	ECUC_Can_00462

3.11.9.4.7.3 CanIcomSignalValue

Table 3-167. CanIcomSignalValue

Description	This parameter shall be used to define a signal value which shall be compared (CanIcomSignalOperation) with the masked CanIcomSignalMask value of the received signal (CanIcomSignalRef).
Class	Autosar Parameter
Range	0 .. 18446744073709551615
Default	NA

Table continues on the next page...

Table 3-167. CanIcomSignalValue (continued)

Source File	Can_Cfg.h, Can_Cfg.c

Source Representation	**In Can_Cfg.h**
	#define CAN_PRETENDED_SIGNAL_VALUE (STD_ON)
	In Can_Cfg.c
	static constCan_IcomRxMessageSignalType Can_PnConfig_0_Rx0_Signal[1]
	=
	{
	    {
	        /* CanIcomSignalMask */
	        (uint64)2305843009213693951U,
	        /*CanIcomSignalOperation */
	        GREATER,
	        /*CanIcomSignalValue */
	        (uint64)12346056164365O8552U,
	        /*DLCLowValue */
	        1U,
	        /*DLCHighValue */
	        8U,
	        /*CanIcomSignalRef */
	        0U
	    }
	};

Autosar 4.2 Requirement	ECUC_Can_00488

3.11.9.4.7.4 DLCLowValue

Table 3-168. DLCLowValue

Description	Sets the lower limit for the number of data bytes considered valid for payload comparison. It is used as part of payload reception filter.
Class	NA
Range	0 .. 8
Default	0
Source File	Can_Cfg.c

Source Representation	static constCan_IcomRxMessageSignalType Can_PnConfig_0_Rx0_Signal[1]
	=
	{
	    {
	        /* CanIcomSignalMask */
	        (uint64)2305843009213693951U,
	        /*CanIcomSignalOperation */
	        GREATER,
	        /*CanIcomSignalValue */
	        (uint64)12346056164365O8552U,
	        /* DLCLowValue */
	        1U,
	        /* DLCHighValue */
	        8U,
	        /*CanIcomSignalRef */
	        0U
	    }
	};

Table continues on the next page...

Table 3-168. DLCLowValue (continued)

Autosar 4.2 Requirement	NA

3.11.9.4.7.5 DLCHighValue

Table 3-169. DLCHighValue

Description	Sets the upper limit for the number of data bytes considered valid for payload comparison. It is used as part of payload reception filter.
Class	NA
Range	0 .. 8
Default	0
Source File	Can_Cfg.c

Source Representation	static constCan_IcomRxMessageSignalType Can_PnConfig_0_Rx0_Signal[1]
	=
	{
	    {
	        /* CanIcomSignalMask */
	        (uint64)2305843009213693951U,
	        /*CanIcomSignalOperation */
	        GREATER,
	        /*CanIcomSignalValue */
	        (uint64)1234605616436508552U,
	        /* DLCLowValue */
	        1U,
	        /* DLCHighValue */
	        8U,
	        /*CanIcomSignalRef */
	        0U
	    }
	};

Autosar 4.2 Requirement	NA

3.11.9.4.7.6 CanIcomSignalRef

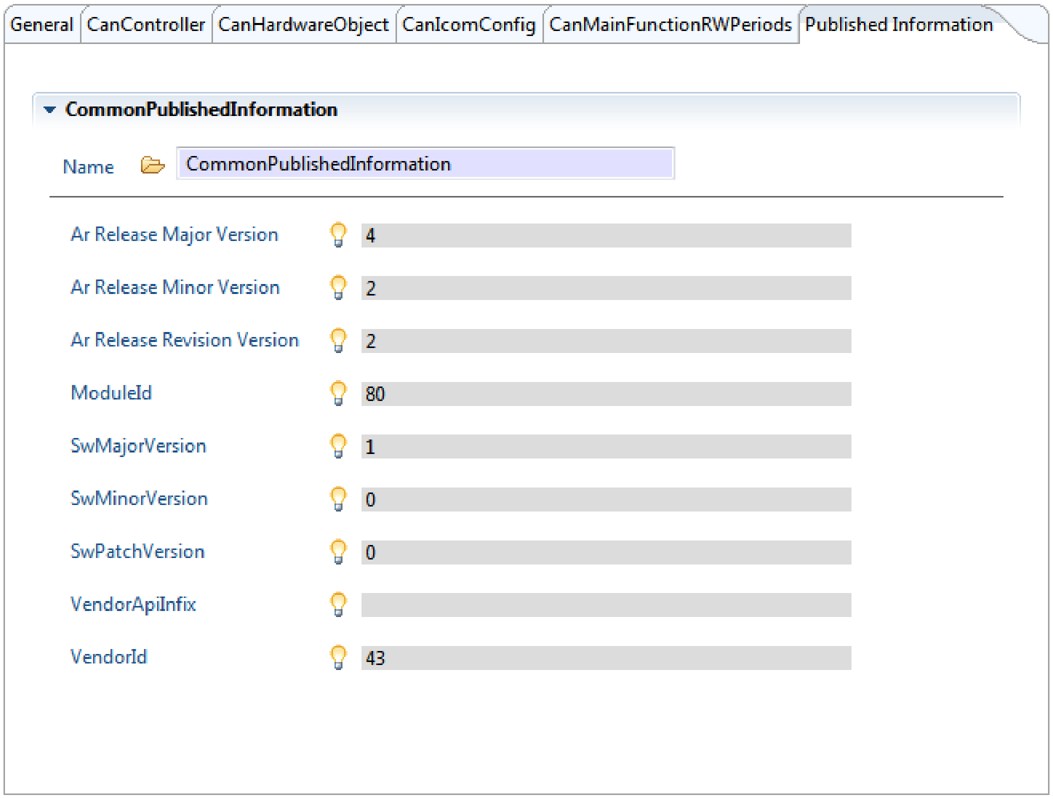
Table 3-170. CanIcomSignalRef

Description	This parameter defines a reference to the signal which shall be checked additional to the message id (CanIcomMessageId). This reference is used for documentation to define which ComSignal originates this filter setting. All signals being referred by this reference shall point to the same PDU.
Class	Autosar Parameter
Default	NA
Source File	NA
Source Representation	NA
Autosar 4.2 Requirement	ECUC_Can_00456

Note

scope: ECU dependency: The signal referenced by CanIcomSignalRef shall be included in a ComIPdu which matches with the current CAN Controller and the CAN Identifier (CanIcomMessageId) configured for this CanIcomRxMessage.

3.11.10 Can Common Published Information



CommonPublishedInformation

Name CommonPublishedInformation

Ar Release Major Version 4

Ar Release Minor Version 2

Ar Release Revision Version 2

ModuleId 80

SwMajorVersion 1

SwMinorVersion 0

SwPatchVersion 0

VendorApiInfix

VendorId 43

Figure 3-26. Can Common Published Information

3.11.10.1 ArReleaseMajorVersion

Table 3-171. ArReleaseMajorVersion

Description	Major version number of AUTOSAR specification on which the appropriate implementation is based on.
Class	Implementation Specific Parameter
Range	Integer
Default	M4_SRC_AR_SPEC_VERSION_MAJOR
Source File	All
Source Representation	ARVersion **M4_SRC_AR_SPEC_VERSION_MAJOR**.M4_SRC_AR_SPEC_VERSION_MINOR.M4_SRC_AR_SPEC_VERSION_PATCH
Autosar 4.0 Requirement	NA

3.11.10.2 ArReleaseMinorVersion

Table 3-172. ArReleaseMinorVersion

Description	Minor version number of AUTOSAR specification on which the appropriate implementation is based on.
Class	Implementation Specific Parameter
Range	Integer
Default	M4_SRC_AR_SPEC_VERSION_MINOR
Source File	All
Source Representation	ARVersion M4_SRC_AR_SPEC_VERSION_MAJOR.**M4_SRC_AR_SPEC_VERSION_MINOR**.M4_SRC_AR_SPEC_VERSION_PATCH
Autosar 4.0 Requirement	NA

3.11.10.3 ArReleaseRevisionVersion

Table 3-173. ArReleaseRevisionVersion

Description	Revision level version number of AUTOSAR specification on which the appropriate implementation is based on.
Class	Implementation Specific Parameter
Range	Integer
Default	M4_SRC_AR_SPEC_VERSION_PATCH
Source File	All
Source Representation	ARVersion M4_SRC_AR_SPEC_VERSION_MAJOR.M4_SRC_AR_SPEC_VERSION_MINOR.**M4_SRC_AR_SPEC_VERSION_PATCH**
Autosar 4.0 Requirement	NA

3.11.10.4 ModuleId

Table 3-174. ModuleId

Description	Module ID of this module from Module List.
Class	Implementation Specific Parameter
Range	Integer
Default	M4_XDM_AR_MODULE_ID
Source File	NA
Source Representation	NA
Autosar 4.0 Requirement	NA

3.11.10.5 SwMajorVersion

Table 3-175. SwMajorVersion

Description	Major version number of the vendor specific implementation of the module. The numbering is vendor specific.
Class	Implementation Specific Parameter
Range	Integer
Default	M4_SRC_SW_VERSION_MAJOR
Source File	All
Source Representation	@version **M4_SRC_SW_VERSION_MAJOR**.M4_SRC_SW_VERSION_MINOR.M4_SRC_SW_VERSION_PATCH
Autosar 4.0 Requirement	NA

3.11.10.6 SwMinorVersion

Table 3-176. SwMinorVersion

Description	Minor version number of the vendor specific implementation of the module. The numbering is vendor specific.
Class	Implementation Specific Parameter
Range	Integer
Default	M4_SRC_SW_VERSION_MINOR
Source File	All
Source Representation	@version M4_SRC_SW_VERSION_MAJOR.**M4_SRC_SW_VERSION_MINOR**.M4_SRC_SW_VERSION_PATCH
Autosar 4.0 Requirement	NA

3.11.10.7 SwPatchVersion

Table 3-177. SwPatchVersion

Description	Patch level version number of the vendor specific implementation of the module. The numbering is vendor specific.
Class	Implementation Specific Parameter
Range	Integer
Default	M4_SRC_SW_VERSION_PATCH
Source File	All

Table continues on the next page...

Table 3-177. SwPatchVersion (continued)

Source Representation	@version M4_SRC_SW_VERSION_MAJOR.M4_SRC_SW_VERSION_MINOR.**M4_SRC_SW_VERSION_PATCH**
Autosar 4.0 Requirement	NA

3.11.10.8 VendorApiInfix

Table 3-178. VendorApiInfix

Description	Vendor API.
Class	Implementation Specific Parameter
Range	Integer
Default	NA
Source File	NA
Source Representation	NA
Autosar 4.0 Requirement	NA

3.11.10.9 VendorId

Table 3-179. VendorId

Description	Vendor ID of the dedicated implementation of this module according to the AUTOSAR vendor list.
Class	Implementation Specific Parameter
Range	Integer
Default	M4_SRC_AR_MODULE_VENDOR_ID
Source File	All
Source Representation	#define CAN_VENDOR_ID_C **M4_SRC_AR_MODULE_VENDOR_ID**
Autosar 4.0 Requirement	NA

3.12 Configuration Exporting

Can_Cfg.h file contains the declaration of configuration structures:

```
#define CAN_CONF_PB \
    extern CONST(Can_ConfigType, CAN_CONST) CanConfigSet_0_VS_0; \
    extern CONST(Can_ConfigType, CAN_CONST) CanConfigSet_0_VS_1; \
    extern CONST(Can_ConfigType, CAN_CONST) CanConfigSet_0_VS_2;
```

Then Can.h file will export this configuration structures, if PostBuild support is enabled:

```
#if (CAN_PRECOMPILE_SUPPORT == STD_OFF)
    CAN_CONF_PB
#endif /* (CAN_PRECOMPILE_SUPPORT == STD_OFF) */
```

Can.h file always exports the below structure:

```
extern CONST(CanStatic_ConfigType, CAN_CONST) CanStatic_ConfigSet;
```

3.13 Rx Fifo

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 behavior when Rx Fifo in 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+(CanRxFifoFiltersNumber div 4). As a result of this dependency the total number of hardware objects per controller is reduced to:96 -(6+CanRxFifoFiltersNumber div 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 parameter CanIdValue 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.

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

3.14 Driver Usage and Configuration Tips

1. A CAN Hardware Unit consists of one or multiple CAN controllers of the same type. Can_MainFunction_Write(), Can_MainFunction_Read(), Can_MainFunction_BusOff() and Can_MainFunction_Wakeup() APIs are defined if at least one Can controller from the Hardware Unit is configured to Polling mode for write, read, busoff and wakeup operation - else are empty functions. Refer to CAN178, CAN180, CAN183 and CAN185.

2. Can_CheckWakeup() and Can_MainFunction_Wakeup() APIs are not defined if Wakeup support is disabled.

3. Can_AbortMb() API (Non Autosar) is defined if this feature is enabled from the Tresos plugin.

4. Can_SetClockMode() API (Non Autosar) is defined if this feature is enabled from the Tresos plugin.

5. The CAN Harwdare Unit can be initialized using Can_Init() API.

6. A single CAN controller can be initalized using Can_InitController() API. The condition is that controller should be in STOP state (not participating to bus communication) while it is initialized.

7. Every CAN controller initialization is preceded by a software reset. See Can_IPW_ResetController() routine from low level driver.

8. Can_InitController() API should configure the MCR, CTRL, RXIMR registers, RxFifo block structure (if enabled), Message Buffers (for Rx MB is configured also every RXIMR register - used for message filtering).

9. Can_IPW_SetControllerMode() API can use transitions to CAN_T_SLEEP and CAN_T_WAKEUP only if Wakeup feature is enabled or supported by the platform.

10. Interrupts can be enabled calling Can_EnableControllerInterrupts() only if Can_DisableControllerInterrupts() was callled prior. Refer to CAN208.

11. Multiplex transmission (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 meens that several Hardware Objects can have the same HWObjectID. This feature can be used only if it's enabled from the Tresos Plugin. Refer to CAN277.

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

13. For Rx MBs the MIXED message buffer type is handled as EXTENDED type. Based on the MB type the RXIMR register is configured according: for STANDARD type the value is left shift with 18 bits.

14. In order to implement fault injection tests using FlexCAN hardware support, the user shall configure the tested controller to use Loop Back mode.

15. The call of the non-autosar function Can_AbortMb shall be follwed by the call of Can_MainFunction_Write when the pooling mode transmission is configured.

Chapter 4
The Configuration of Can Bit Timing

4.1 Clock Source Description

The FlexCAN module supports a variety of means to setup bit timing parameters that are required by the CAN protocol. The Control Register has various fields used to control bit timing parameters: PRESDIV, PROPSEG, PSEG1, PSEG2 and RJW.

The PRESDIV field (CTRL[PRESDIV]) controls a prescaler that generates the Serial Clock (Sclock), whose period defines the ‘time quantum’ used to compose the CAN waveform. A time quantum is the atomic unit of time handled by the CAN engine.

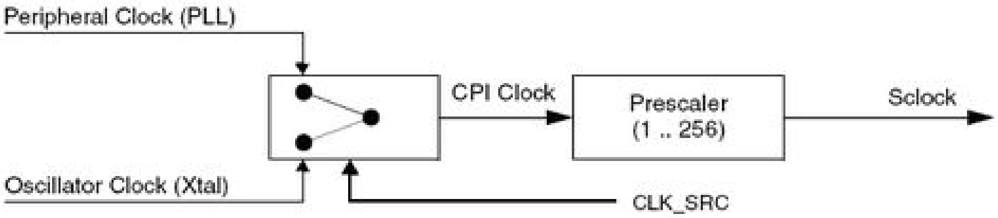


Figure 4-1. Can Engine Clock Scheme

A bit time is subdivided into three segments:

SYNC_SEG: This segment has a fixed length of one time quantum. Signal edges are expected to happen within this section.

Time Segment 1: This segment includes the Propagation Segment and the Phase Segment 1 of the CAN standard. It can be programmed by setting the PROPSEG and the PSEG1 fields of the CTRL Register so that their sum (plus 2) is in the range of 4 to 16 time quanta.

Time Segment 2: This segment represents the Phase Segment 2 of the CAN standard. It can be programmed by setting the PSEG2 field of the CTRL Register (plus 1) to be 2 to 8 time quanta long.

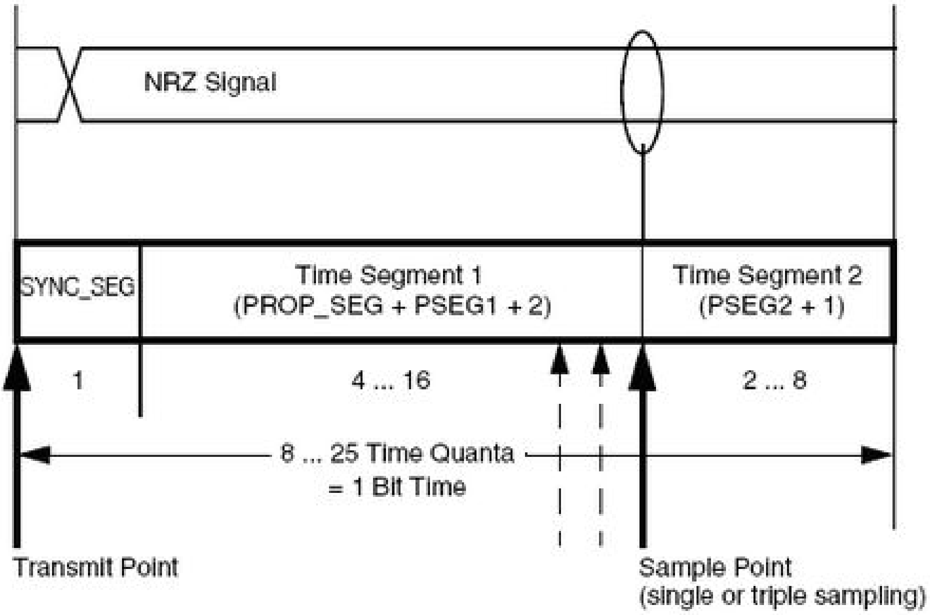
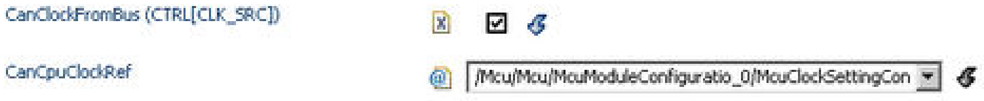


Figure 4-2. Can Time Segments

Understanding the Clock Selection for Can

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

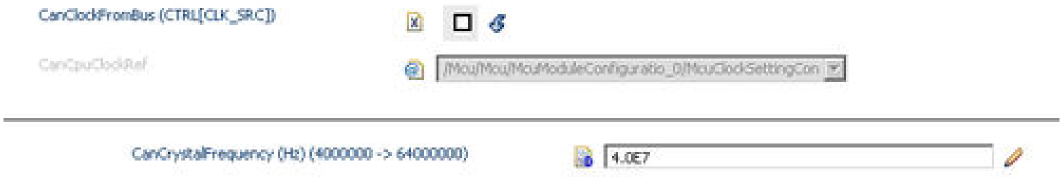


CanClockFromBus (CTRL[CLK_SRC]) ☑

CanCpuClockRef /Mcu/Mcu/McuModuleConfiguratio_0/McuClockSettingCon

Figure 4-3. Can Clock Selection: bus

If “CanClockFromBus” is checked, the clock used by the hardware is extracted from external oscillator. If “CanClockFromBus” is unchecked the clock value is extracted from peripheral clock. It is user responsability to check and to select the wright clock source for his application.



CanClockFromBus (CTRL[CLK_SRC]) ☐

CanCpuClockRef /Mcu/Mcu/McuModuleConfiguratio_0/McuClockSettingCon

CanCrystalFrequency (Hz) (4000000 -> 64000000) 4.0E7

Figure 4-4. Can Clock Selection: osc_clock

4.2 Time Segments Calculation

Basic-Can vs. Full-Can

The terms **Basic CAN** and **Full CAN** must not be confused with the terms Standard CAN - also known as Base Frame Format (11 bit identifier, Version 2.0A data format) and Extended CAN - also known as Extended Frame Format (29 bit identifier, or Version 2.0B data format). Suitably configured, each implementation (Basic or Full CAN) can handle both Base and Extended data formats.

NOTE: Can Driver doesn’t make any difference between Full and Basic Hardware Object Type. This is implemented in the plugin configuration just because it’s a requirement of AUTOSAR (CAN323_Conf)

Time Segments Calculation

Every CanController container of Tresos configuration tool has a parameter named “CanClockFromBus” which selects the clock source to the CanController to be either the peripheral clock (from MCU configuration if “CanClockFromBus” is true) or the crystal oscillator clock. Based on “CanClockFromBus” parameter, we have the frequency of CanController named **CanClockFrequency** (in Hertz).

Step.1

First step is to input the Prescaler (CTRL[PRESDIV]). Under the "Can Controller BaudRate" container, the Can Controller Prescaller must be completed. From this unit, we can calculate the **TimeQuantum** as follow:

TimeQuantum = Prescaler / CanClockFrequency (seconds)

Note

Valid interval for Prescaler is between 1 and 256.

Step.2

Second step is to calculate the number of **TimeQuantum** per Can bit (**No. of CanTimeQuantas**) based on the **TimeQuantum** parameter and "CanControllerBaudRate" parameter. Where, **TimeQuantum** is calculated as above and "CanControllerBaudRate" parameter is configured from the "CanControllerBaudRate" container of Tresos Can Plugin.

No. of CanTimeQuantas = (1 / CancontrollerBaudRate) / TimeQuantum (const = (1 / hertz) / seconds)

Note

Valid interval for No of CanTimeQuantas is between 8 and 25.

Step.3

Third step is to check the compatibility for parameters. The sum of SyncSeg, Propagation Segment, Phase Segment 1 and Phase Segment 2 must be equal with **No. of CanTimeQuantas** resulted as above. The SyncSeg has a fixed value of 1. Propagation Segment, Phase Segment 1 and Phase Segment 2 can be configured by using "CanControllerPropSeg", "CanControllerSeg1" and "CanControllerSeg2" parameters in Tresos Can Plugin, respectively. Thus, the below equation must be satisfied:

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

where,

CanControllerPropSeg = PROP_SEG + 1

CanControllerSeg1 = PSEG1 + 1

CanControllerSeg2 = PSEG2 + 1

(see PROP_SEG , PSEG1 and PSEG2 in Figure 4-2).

4.3 CAN Bit Timing

Selecting bit timing parameters that work well on the bench may not equate with the situation that the product moves into the real environment (maximum wiring length, worst case configuration, oscillator tolerance, etc) and we can find that bit timing parameters are inadequate.

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

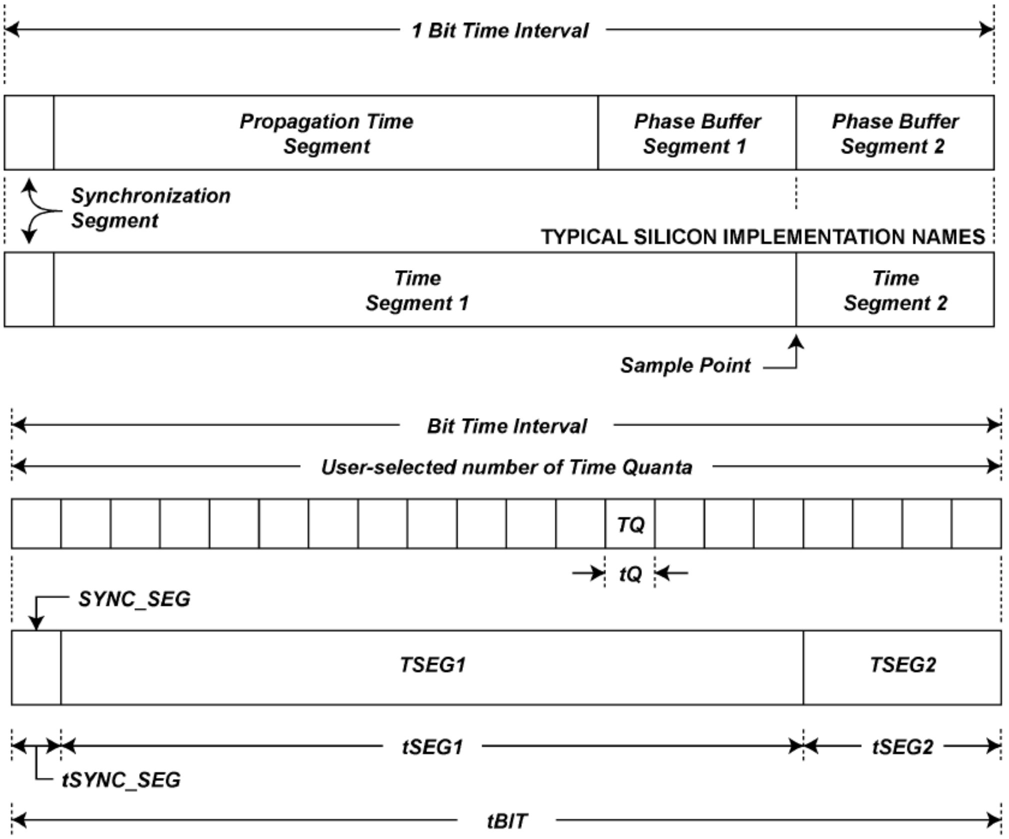


Figure 4-5. Can Bit Timing

The most important part of CAN bit timing is the Time Quantum (TQ). The time duration of the time quantum is derived from the CAN controller clock oscillator and the adjustable clock divider (prescaler). For Autosar CAN requirements we have to define Clock source and Time Quantum and calculate indirectly the prescaler.

The Synchronization Segment (SYNC_SEG) time interval is used to synchronize all the nodes across the network. SYNC_SEG time interval has a fixed period of one Time Quantum.

Time Segment 1 (TSEG1) is the time interval used to compensate for both positive phase errors in synchronization between nodes on the network and propagation delay between network nodes.

Time Segment 2 (TSEG2) is the time interval used to compensate for negative phase errors in synchronization between nodes.

Re-synchronization Jump Width (SJW) is not directly a segment of the bit time, but is used to dynamically adjust TSEG1 and TSEG2. SJW is the maximum amount of time by which TSEG1 may be lengthened or TSEG2 shortened to compensate for synchronization differences between nodes on the CAN network.

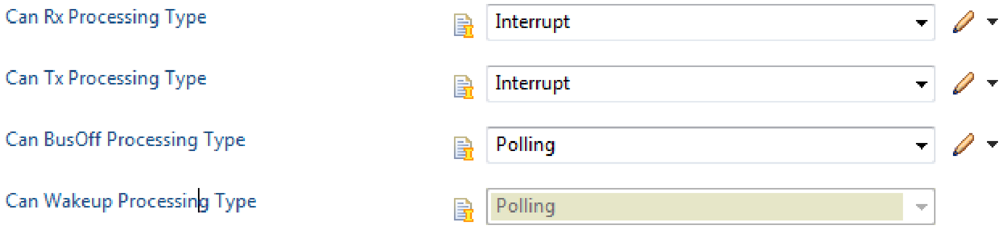
Bit Timing Example

```
Clock Source= 20MHz
Prescaler = 2
Bit Rate = 500K bits per second
From these input data will result:
    Time Bit = 2 microseconds ( 1/ 500Kbps)
    CAN clock source = 10 MHz (Clock Source / Prescaler)
    Selecting TQ = 100 nsec -> No. TQs per bit = 20 ( 2 usec / 100 nsec)
Bit Time Segments Calculation:

    SYNC_SEG = 1 (according to CAN ISO standard)

    TSEG1 = 15

    TSEG2 = 4

Sample Point Calculation:

    SP = (1 + TSEG1) / (1 + TSEG1 + TSEG2) = 0.8 -> 80%
```

Chapter 5
Interrupts Implementation

5.1 General Aspects

Autosar specifications permit to configure Can controller in interrupt mode for the following events: Rx Processing, Tx Processing, BusOff Processing, Wakeup Processing.



Can Rx Processing Type	Interrupt
Can Tx Processing Type	Interrupt
Can BusOff Processing Type	Polling
Can Wakeup Processing Type	Polling

Figure 5-1. Can Interrupts Selection

Interrupts implementation and mapping is done according to MPC574XG Reference Manual.

Note

> The interrupt service routines are not coded as re-entrant and may only be preempted by code which does not call any of the driver functions.

5.2 Interrupt Handlers

The interrupt handlers for all controllers follow the same naming convention: Can_Isr_X, where X could be A, B, C, D, etc. based on the number of CAN controllers included on on the microcontroller. Each controller has a single interrupt handler which is triggered for any interrupt event generated by the controller. The interrupt handler shall check all the interrupt status flags of the controller in order to identify the interrupt source and treat it properly.

How to Reach Us:

Home Page:
nxp.com

Web Support:
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.

Document Number UM35CANASR4.2 Rev0002 RTM
1.0.0
Revision 1.0