MCAL User Manual for Can_17_McmCan

32-bit TriCoreTM AURIXTM TC3xx microcontroller

About this document

Scope and purpose

This User Manual is intended to enable users to integrate the Microcontroller Abstraction Layer (MCAL) software for the TriCoreTM AURIXTM family of 32-bit microcontrollers.

This document describes responsibilities of integrator in-charge of integrating MCAL software with the basic software (BSW) stack. This document also provides detailed information on safety, configuration and functions along with examples of usage of significant features.

Note: Detailed information about package installation, safety and other generic information that are common across all modules are provided in MCAL User Manual General.

Intended audience

This document is intended for anyone using the Can_17_McmCan module of the TC3xx MCAL software.

Document conventions

Table 1 Conventions

Convention	Explanation
Bold	Emphasizes heading levels, column headings, table and figure captions, screen names, windows, dialog boxes, menus, sub-menus
Italics	Denotes variable(s) and reference(s)
`Courier` `New`	Denotes APIs, functions, interrupt handlers, events, data types, error handlers, file/folder names, directories, command line inputs, code snippets
>	Indicates that a cascading sub-menu opens when you select a menu item
[cover parentID=<alpha numeric value>]	Used for traceability completeness. Reader should ignore these.

Reference documents

This User Manual should be read in conjunction with the following documents:

- AURIXTM TC3xx User Manual User Manual General
- Specification of CAN Driver, AUTOSAR_SWS_CAN_Driver, AUTOSAR Release 4.2.2
- Specification of CAN Driver, AUTOSAR_SWS_CAN_Driver, AUTOSAR Release 4.4.0

Table of contents

1 Can_17_McmCan driver

1.1 User information

1.1.1 Description

The CAN driver is responsible for providing standard CAN communication services specified by AUTOSAR 4.2.2 and 4.4.0. The M_CAN unit is the underlying CAN hardware unit, which consists of nodes (called as controllers in AUTOSAR) sharing the message RAM (called as hardware objects in AUTOSAR). The CAN driver provides services for:

- Initialization of CAN controllers to control the behavior and state of the CAN controllers
- Setting and modifying the baud-rate configuration of the CAN controller
- CAN and CAN FD frame transmission and reception is supported
- Successful frame transmission notification, reception of dedicated and FIFO messages and bus-off event notification in the polling and interrupt modes
- Data reception using the receive FIFO functionality
- Pretended networking mode handling
- Multiple read/write period functionality support
- Multiplexed transmission using Tx queue
- Individual interrupt lines are routed for the handling of the following events of each CAN node:

- Bus-off event handling - Transmit event handling - Dedicated message receive event handling - Receive FIFO 0 and FIFO 1 watermark and FIFO full event handling.

- Mixed mode handling for Rx and Tx processing

The CAN driver is delivered as a Post-Build variant. Therefore, the driver supports configuration parameters with pre-compile and post-build configuration classes. The APIs provided by the CAN driver are multicore capable, which means that they may be invoked from several cores simultaneously. The availability of the APIs, configuration parameters and the error handling are dependent on the AUTOSAR version being used.

1.1.2 Hardware-software mapping

This section describes the system view of the CAN driver and peripherals administered by it.

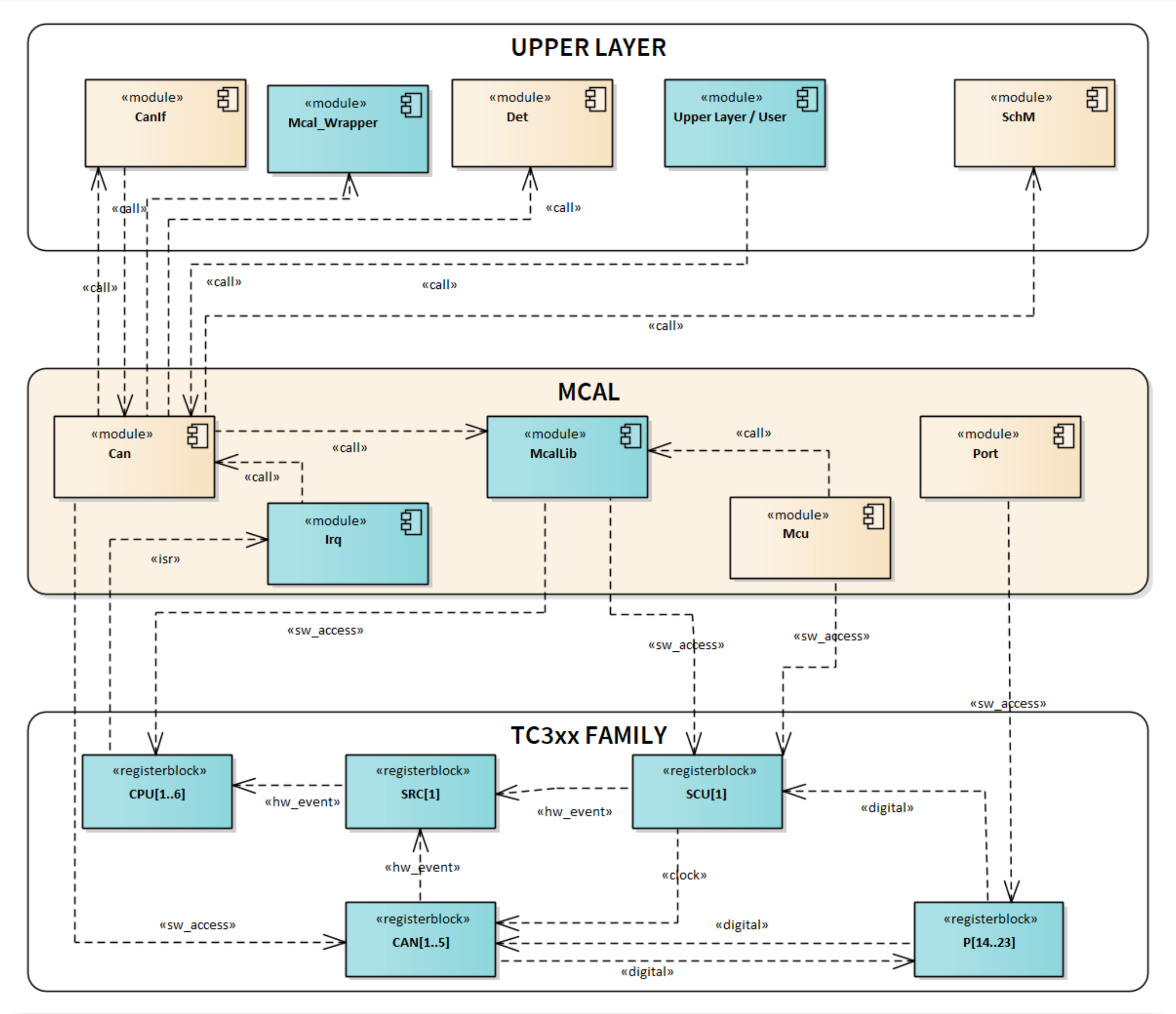

Figure 1 Mapping of hardware-software interfaces

1.1.2.1 M_CAN: primary hardware peripheral

Hardware functional features

The CAN driver uses the M_CAN to communicate according to the ISO 11898-1. In addition, the M_CAN supports communication according to the CAN FD protocol specification 1.0. The key hardware functional features used by the driver are:

- All the CAN controllers and message RAM available in the M_CAN module are used to implement the CAN driver
- CAN FD with up to 64 data bytes supported
- Up to 64 dedicated receive buffers
- Up to 32 dedicated transmit buffers
- Two configurable receive FIFOs

- Configurable transmit queue
- Four Individual interrupts are configured per controller. These are dedicated Rx, Rx FIFOs, Tx and bus-off events

The unsupported features of the M_CAN are:

- Event-synchronized time-triggered communication
- CAN error logging
- High priority messages

Users of the hardware

The CAN driver exclusively utilizes the M_CAN module.

Hardware diagnostic features

The SMU alarms configured for the M_CAN are not monitored by the CAN driver.

Hardware events

The CAN driver uses the following hardware events from the M_CAN IP:

- Successful transmission of a CAN / CAN FD frame is notified by flag (relevant bit in the IR register) as well as interrupt. The CAN driver uses the TxEvent FIFO new entry to handle notifications to upper layer
- Successful reception of a CAN / CAN FD frame is notified by flag (relevant bit in the IR register) as well as interrupt. The CAN driver uses the receive interrupt raised
- Bus-Off event is notified by flag (relevant bit in the IR register) as well as interrupt. The CAN driver uses the bus-off interrupt which is raised
- Both Rx FIFO0 watermark reached, RxFIFO0 Full, RxFIFO 1 watermark reached event, RxFIFO1 Full events are routed to same ISR. All the listed flags are handled in the CAN driver to process the received data through FIFO

1.1.3 File structure

1.1.3.1 C file structure

This section provides details of the C files of the CAN driver.

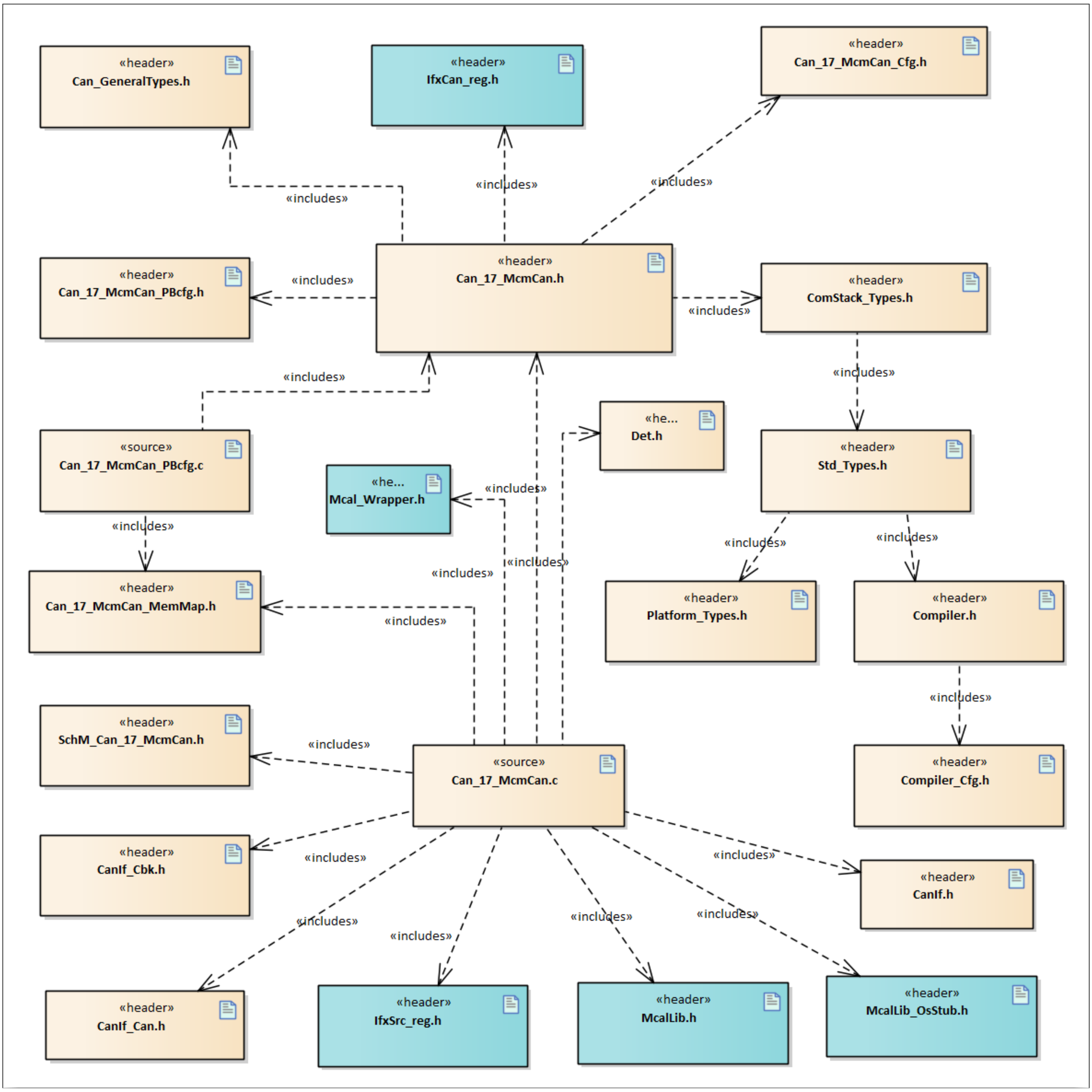

Figure 2 Can_C_File_Structure-1.png

Table 2 C file structure

File name	Description
`CanIf.h`	Header file containing the exported interfaces of CanIf
`CanIf_Can.h`	Header file containing declarations of the CanIf callbacks *Note: This file is available only for AUTOSAR version 4.4.0*
`CanIf_Cbk.h`	Header file containing declarations of the CanIf callbacks. *Note: This file is available only for AUTOSAR version 4.2.2*
`Can_17_McmCan.c`	Implementation of the CAN driver functionality

(table continues...)

Table 2 (continued) C file structure

File name	Description
`Can_17_McmCan.h`	Export of the CAN driver functionality
`Can_17_McmCan_Cfg.h`	The configuration data of the CAN driver is declared here *Note: All pre-compile time configuration parameters shall be defined as pre-processor directives (#define)*
`Can_17_McmCan_MemMap.h`	Mapping of code and data (variables, constant variables) to specific memory sections
`Can_17_McmCan_PBcfg.c`	Post Build configuration data of the CAN driver is defined here
`Can_17_McmCan_PBcfg.h`	Header file (generated) containing declaration of the post-build configuration data structures
`Can_GeneralTypes.h`	Contains all types and constants that are shared among the AUTOSAR CAN modules Can, CanIf and CanTrcv
`ComStack_Types.h`	Type Definition for Com stack
`Compiler.h`	Provides abstraction from compiler-specific keywords
`Compiler_Cfg.h`	Configuration header file for compiler abstraction
`Det.h`	Provides the exported interfaces of Development Error Tracer
`IfxCan_reg.h`	SFR header file for CAN
`IfxSrc_reg.h`	SFR header file for Interrupt Controller
`McalLib.h`	Static header file defining prototypes of data structure and APIs exported by the MCALLIB.
`McalLib_OsStub.h`	McalLib_OsStub.h provides macros to support user mode of Tricore. This shall be included by other drivers to call OS APIs.
`Mcal_Wrapper.h`	Provides the exported interfaces for Production Error and Runtime Development Errors. Implemented by default to include functions of Dem.h and Det.h files. This file can be modified by the user but function prototype is not user modifiable.
`Platform_Types.h`	Platform-specific type declaration file as defined by AUTOSAR
`SchM_Can_17_McmCan.h`	Functions to enable/disable interrupts are declared here.
`Std_Types.h`	Standard type declaration file as defined by AUTOSAR. It is independent of compiler or platform.

1.1.3.2 Code generator plugin files

This section provides details of the code generator plugin files of the CAN driver.

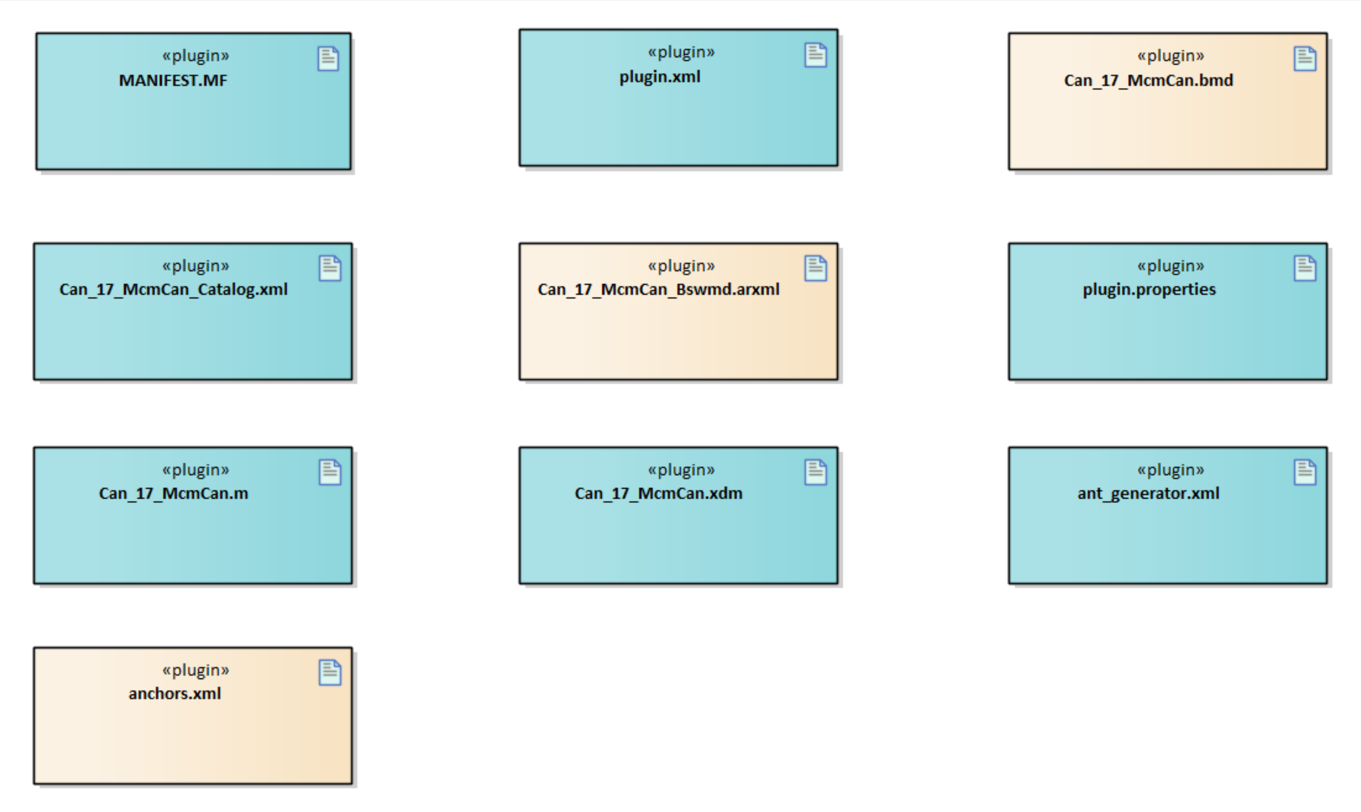

Figure 3 Can_Code_Generator_Plugin_Files-1.png

Table 3 Code generator plugin files

File name	Description
`Can_17_McmCan.bmd`	AUTOSAR format XML data model schema file for the CAN driver
`Can_17_McmCan.m`	Code template macro file for the CAN driver
`Can_17_McmCan.xdm`	Tresos format XML data model schema file
`Can_17_McmCan_Bswmd.arxml`	AUTOSAR format module description file
`Can_17_McmCan_Catalog.xml`	AUTOSAR format catalog file
`MANIFEST.MF`	Tresos plugin support file containing the metadata for the CAN driver
`anchors.xml`	AUTOSAR format module description file
`ant_generator.xml`	Tresos support file to generate and rename multiple post-build configuration when using variation point
`plugin.properties`	Tresos plugin support file for the CAN driver
`plugin.xml`	Tresos plugin support file for the CAN driver

1.1.4 Integration hints

This section lists the key points, that an integrator or user of the CAN driver must consider.

1.1.4.1 Integration with AUTOSAR stack

This section lists the modules, which are not part of MCAL, but required to integrate the CAN driver.

- **EcuM:**

  The ECU Manager module is a part of the AUTOSAR stack that manages common aspects of ECU. Specifically, in the context of MCAL, EcuM is used for initialization and de-initialization of the software drivers. The EcuM module provided in the MCAL package is a stub code and needs to be replaced with a complete EcuM module during the integration phase.

- **CAN Interface (CanIf):**

  The CanIf module is a part of the AUTOSAR stack that provides upper layers a hardware independent interface to the CAN communication system comprising multiple CAN controllers.

  The `CanIf_Cbk.c` and `CanIf_Cbk.h` files are provided as stub code and needs to be replaced with complete CanIf module during integration phase. The CAN driver uses the APIs of CanIf to provide notifications as listed.

  `CanIf_ControllerModeIndication()`: Notification for a successful state transition that was triggered for a controller

  `CanIf_TxConfirmation()`: Notification for a successfully processed transmission of a CAN Tx pdu.

  `CanIf_RxIndication()`: Notification for a successful reception of a received CAN Rx l-pdu to the CanIf after passing all filters and validation checks.

  `CanIf_ControllerBusOff()`: Notification for a Controller BusOff event referring to the corresponding CAN Controller.

  `CanIf_CurrentIcomConfiguration()`: Notification to inform about the change of the Icom configuration of a CAN controller.

  `CanIf_TriggerTransmit()`: Within this API, the CanIf shall check whether the available data fits into the buffer size reported by PduInfoPtr->SduLength. If it fits, it shall copy its data into the buffer provided by PduInfoPtr.

- **Memory mapping:**

  Memory mapping is a concept from AUTOSAR that allows relocation of text, variables, constants and configuration data to user specific memory regions. To achieve this, all the re-locatable elements of the driver are encapsulated in different memory-section macros. These macros are defined in the `Can_17_McmCan_MemMap.h`.

  The file `Can_17_McmCan_MemMap.h` file is provided in the MCAL package as a stub code. The integrator must place appropriate compiler pragmas within the memory-section macros. The pragmas ensure that the elements are re-located to the correct memory region. A sample implementation listing the memory-section macros is shown as follows.

```
/***** GLOBAL RAM DATA -- NON CLEARED LMU *****/
#if defined CAN_17_MCMCAN_START_SEC_VAR_CLEARED_QM_GLOBAL_32
 /******User pragmas here for non- cached LMU*******/
 #undef CAN_17_MCMCAN_START_SEC_VAR_CLEARED_QM_GLOBAL_32
 #undef MEMMAP_ERROR
#elif defined CAN_17_MCMCAN_STOP_SEC_VAR_CLEARED_QM_GLOBAL_32
 /******User pragmas here for non- cached LMU*******/
 #undef CAN_17_MCMCAN_STOP_SEC_VAR_CLEARED_QM_GLOBAL_32
 #undef MEMMAP_ERROR

/***** CORE[x] CONFIG DATA --PF[x] *****/
#elif defined CAN_17_MCMCAN_START_SEC_CONFIG_DATA_QM_CORE0_UNSPECIFIED
 /******User pragmas here for PF[x] *******/
 #undef CAN_17_MCMCAN_START_SEC_CONFIG_DATA_QM_CORE0_UNSPECIFIED
 #undef MEMMAP_ERROR

#elif defined CAN_17_MCMCAN_STOP_SEC_CONFIG_DATA_QM_CORE0_UNSPECIFIED
 /******User pragmas here for PF[x] *******/
 #undef CAN_17_MCMCAN_STOP_SEC_CONFIG_DATA_QM_CORE0_UNSPECIFIED
 #undef MEMMAP_ERROR

/***** CODE -- PF[x] *****/
#elif defined CAN_17_MCMCAN_START_SEC_CODE_QM_GLOBAL
 /******User pragmas here for PF[x] *******/
 #undef CAN_17_MCMCAN_START_SEC_CODE_QM_GLOBAL
 #undef MEMMAP_ERROR

#elif defined CAN_17_MCMCAN_STOP_SEC_CODE_QM_GLOBAL
 /******User pragmas here for PF[x] *******/
 #undef CAN_17_MCMCAN_STOP_SEC_CODE_QM_GLOBAL
 #undef MEMMAP_ERROR
#endif

#if defined MEMMAP_ERROR
#error "Can_17_McmCan_MemMap.h, wrong pragma command"
#endif
```

- **DET:**

  The DET module is a part of the AUTOSAR stack that handles all the development errors reported by the BSW modules. The CAN driver reports all the development errors to the DET module through the `Det_ReportError()` API. The user of the CAN driver must process all the errors reported to the DET module through the API `Det_ReportError()`.

  The files `Det.h` and `Det.c` are provided in the MCAL package as a stub code and needs to be replaced with a complete DET module during the integration phase.

- **Mcal_Wrapper:**

  This Driver performs reporting of the Production and Runtime errors. The handling of the reported errors shall be done by the user. The `Mcal_Wrapper_Det_ReportRuntimeError()` API, `Mcal_Wrapper_Dem_SetEventStatus()` API and `Mcal_Wrapper_Dem_ReportErrorStatus()` API are provided in the `Mcal_Wrapper.c` and `Mcal_Wrapper.h` files as a stub code, and can be updated by the integrator to handle the reported errors. The files `Mcal_Wrapper.c` and `Mcal_Wrapper.h` are user modifiable but the function prototypes are not user modifiable and by default the Mcal_Wrapper function shall call AUTOSAR DEM and DET Modules.

The user of the CAN driver shall process all the Runtime errors reported to the Mcal_Wrapper module. The interface used for reporting Runtime error in both AUTOSAR is `Mcal_Wrapper_Det_ReportRuntimeError()` API. The `Mcal_Wrapper.c` and `Mcal_Wrapper.h` files are provided in the MCAL package as a stub code and can be replaced with a user specific production and Runtime error handling module/s during the integration phase. The production error handling module is not required for integrating the CAN driver.

- **SchM:**

  The SchM module is a part of the RTE that manages the Basic Software Scheduler. The CAN driver uses the exclusive areas defined in `SchM_Can_17_McmCan.h` to protect the SFRs and variables from concurrent accesses from different threads. The SchMs identified for the CAN driver are:

  CanWrMO

  IcomMsgCntrVal

  The `SchM_Can_17_McmCan.h` and `SchM_Can_17_McmCan.c` files are provided in the MCAL package as an example code and needs to updated by the integrator. The user must implement the SchM functions defined by the CAN driver as suspend / resume of interrupts for the CPU on which the API is invoked. A sample implementation of the SchM functions is depicted below:

```
/**** Sample implementation of SchM_Can_17_McmCan.c ****/
void SchM_Enter_Can_17_McmCan_CanWrMO()
{
 /* Start critical section */
 SuspendAllInterrupts(); /* Suspend CPU core interrupts */
}

void SchM_Exit_Can_17_McmCan_CanWrMO()
{
 /* End of critical section */
 ResumeAllInterrupts(); /* Resume CPU core interrupt */
}

void SchM_Enter_Can_17_McmCan_IcomMsgCntrVal()
{
 /* Start critical section */
 SuspendAllInterrupts(); /* Suspend CPU core interrupts */
}

void SchM_Exit_Can_17_McmCan_IcomMsgCntrVal()
{
 /* End of critical section */
 ResumeAllInterrupts(); /* Resume CPU core interrupt */
}
```

- **Safety error:**

  The CAN driver does not report any safety errors.

- **Notifications and call-backs:**

  The CAN driver does not implement any notifications. However, it does report transmit confirmation, mode change indication, bus-off and wake up identification, pretended network activation or de-activation completion and successful reception through the CanIf module call backs.

- **Callout:**

  The CAN driver provides the prototype of the LPDU callout function. The callout name and implementation is defined by the application using the parameter `CanLPduReceiveCalloutFunction`.

- **Operating System:**

  OS or application must ensure correct type of service and interrupt priority is configured in the SR register. Enabling and disabling of the interrupts must also be managed by the OS or application.

  The operating system files provided by MCAL package is only an example code and must be updated by the integrator with actual OS files for desired function.

1.1.4.2 Multicore and Resource Manager

The CAN driver supports execution of its APIs simultaneously from all CPU cores. The user should allocate resources of CAN to CPU cores at pre-compile time using the Resource Manager module. The following are the key points to be considered with respect to multicore in the driver:

- CAN controllers of the CAN driver can be allocated to CPU cores at pre-compile time.
- CAN controllers that are not allocated to a CPU core shall be by default allocated to the master core.
- Initialization of the CAN controller must start with the master core initialization only after the successful initialization of the master core should there be a trigger for a slave core initialization. CAN driver of the slave cores can be initialized simultaneously.
- De-initialization of the CAN driver for different slave cores can be initiated simultaneously. The master core de-initialization of the CAN driver should be carried out only after the de-initialization of the CAN driver in all the slave cores.
- DETs will be raised in case APIs are invoked with mismatch of CPU core and controller IDs or hardware object IDs.
- Interrupts raised by a hardware group must be serviced by the CPU core to which the hardware group has been allocated to.
- Locating constants, variables and configuration data to correct memory space should be done by the user. Memory sections are marked GLOBAL(common to all cores) and CORE[x] (specific to a CPU core). The following should be considered by the user to ensure better performance of the driver:

  Code section:

  The executable code of CAN driver is placed under single MemMap section. It can be relocated to any PFlash region.

  Data section:

  The RAM variable memory sections marked as specific to a core should be relocated to the DSPR/DLMU of the same core. The sections marked as global should be relocated to the non-cached LMU region. In devices with no LMU, non-cached DSPR can be used.

  Configuration data and constants:

  The configuration data sections marked as specific to a core should be re-located to the PFlash of the same core. The sections marked as global should be relocated to the PFlash of the master core.

  Note: Relocating of code, data or constants to a distant memory region would impact execution timings.

  Note: If the driver operates from a single (master) core, all the sections may be relocated to the PFlash/DSPR/DLMU of the same CPU core.</I>

1.1.4.3 MCU support

The CAN driver is dependent on the MCU driver for clock configuration. The initialization of CAN driver must be started only after the completion of MCU initialization. The following must be considered while configuring the MCU driver in EB Tresos:

- McuMCanClockSourceSelection - Used to select the different clock source.
- McuMCanFrequency - To be set if the McuMCanClockSourceSelection is MCAN_CLOCK_SOURCE_MCANI_SEL1.
- McuMainOscillatorFrequency - To be set if the McuMCanClockSourceSelection is MCAN_CLOCK_SOURCE_OSC_SEL2.

1.1.4.4 Port support

The PORT driver configures the port pins of the entire microcontroller. The user must configure port pins used by the CAN driver through the PORT configuration and initialize the port pins prior to invoking of CAN initialization.

The TxD and RxD pins (corresponding to the Rx Pin selection made in CAN driver) of the different CAN controllers must be configured with respective direction and configuration in the PORT driver.

1.1.4.5 DMA support

The CAN driver does not use any services provided by the DMA driver.

1.1.4.6 Interrupt connections

The interrupt connections of the CAN driver are described in this section.

Table 4 Handling CAN interrupt lines:

Controller	Signal	Service type	Function to be called
Controller 0	`CAN0SR0_ISR`	Service on CAN data Reception through dedicated buffer.	`Can_17_McmCan_IsrReceiveHandler(CAN_17_MCMCAN_HWMCMKERNEL0_ID,CAN_17_MCMCAN_HWMCMCONTROLLER0_ID);`
Controller 0	`CAN0SR1_ISR`	Service for the transmission completion new entry event on TX FIFO Event.	`Can_17_McmCan_IsrTransmitHandler(CAN_17_MCMCAN_HWMCMKERNEL0_ID,CAN_17_MCMCAN_HWMCMCONTROLLER0_ID);`
Controller 0	`CAN0SR2_ISR`	Service on CAN controller within Bus Off mode.	`Can_17_McmCan_IsrBusOffHandler(CAN_17_MCMCAN_HWMCMKERNEL0_ID,CAN_17_MCMCAN_HWMCMCONTROLLER0_ID);`
Controller 0	`CAN0SR3_ISR`	Service on CAN receive FIFO water mark level or FIFO full level reached on Rx FIFO0 or Rx FIFO1.	`Can_17_McmCan_IsrRxFIFOHandler(CAN_17_MCMCAN_HWMCMKERNEL0_ID,CAN_17_MCMCAN_HWMCMCONTROLLER0_ID);`

(table continues...)

Table 4 (continued) Handling CAN interrupt lines:

Controller 1	CAN0SR4_ISR	Service on CAN data Reception through dedicated buffer.	Can_17_McmCan_IsrReceiveHandler(CAN_17_MCMCAN_HWMCMKERNEL0_ID,CAN_17_MCMCAN_HWMCMCONTROLLER1_ID);
Controller 1	CAN0SR5_ISR	Service for the transmission completion new entry event on TX FIFO Event.	Can_17_McmCan_IsrTransmitHandler(CAN_17_MCMCAN_HWMCMKERNEL0_ID,CAN_17_MCMCAN_HWMCMCONTROLLER1_ID);
Controller 1	CAN0SR6_ISR	Service on CAN controller within Bus Off mode.	Can_17_McmCan_IsrBusOffHandler(CAN_17_MCMCAN_HWMCMKERNEL0_ID,CAN_17_MCMCAN_HWMCMCONTROLLER1_ID);
Controller 1	CAN0SR7_ISR	Service on CAN receive FIFO water mark level or FIFO full level reached on Rx FIFO0 or Rx FIFO1.	Can_17_McmCan_IsrRxFIFOHandler(CAN_17_MCMCAN_HWMCMKERNEL0_ID,CAN_17_MCMCAN_HWMCMCONTROLLER1_ID);
Controller 2	CAN0SR8_ISR	Service on CAN data Reception through dedicated buffer.	Can_17_McmCan_IsrReceiveHandler(CAN_17_MCMCAN_HWMCMKERNEL0_ID,CAN_17_MCMCAN_HWMCMCONTROLLER2_ID);
Controller 2	CAN0SR9_ISR	Service for the transmission completion new entry event on TX FIFO Event.	Can_17_McmCan_IsrTransmitHandler(CAN_17_MCMCAN_HWMCMKERNEL0_ID,CAN_17_MCMCAN_HWMCMCONTROLLER2_ID);
Controller 2	CAN0SR10_ISR	Service on CAN controller within Bus Off mode.	Can_17_McmCan_IsrBusOffHandler(CAN_17_MCMCAN_HWMCMKERNEL0_ID,CAN_17_MCMCAN_HWMCMCONTROLLER2_ID);
Controller 2	CAN0SR11_ISR	Service on CAN receive FIFO water mark level or FIFO full level reached on Rx FIFO0 or Rx FIFO1.	Can_17_McmCan_IsrRxFIFOHandler(CAN_17_MCMCAN_HWMCMKERNEL0_ID,CAN_17_MCMCAN_HWMCMCONTROLLER2_ID);
Controller 3	CAN0SR12_ISR	Service on CAN data Reception through dedicated buffer.	Can_17_McmCan_IsrReceiveHandler(CAN_17_MCMCAN_HWMCMKERNEL0_ID,CAN_17_MCMCAN_HWMCMCONTROLLER3_ID);
Controller 3	CAN0SR13_ISR	Service for the transmission completion new entry event on TX FIFO Event.	Can_17_McmCan_IsrTransmitHandler(CAN_17_MCMCAN_HWMCMKERNEL0_ID,CAN_17_MCMCAN_HWMCMCONTROLLER3_ID);

(table continues...)

Table 4 (continued) Handling CAN interrupt lines:

Controller 3	`CAN0SR14_ISR`	Service on CAN controller within Bus Off mode.	`Can_17_McmCan_IsrBusOffHandler(CAN_17_MCMCAN_HWMCMKERNEL0_ID,CAN_17_MCMCAN_HWMCMCONTROLLER3_ID);`
Controller 3	`CAN0SR15_ISR`	Service on CAN receive FIFO water mark level or FIFO full level reached on Rx FIFO0 or Rx FIFO1.	`Can_17_McmCan_IsrRxFIFOHandler(CAN_17_MCMCAN_HWMCMKERNEL0_ID,CAN_17_MCMCAN_HWMCMCONTROLLER3_ID);`
Controller 4	`CAN1SR0_ISR`	Service on CAN data Reception through dedicated buffer.	`Can_17_McmCan_IsrReceiveHandler(CAN_17_MCMCAN_HWMCMKERNEL1_ID,CAN_17_MCMCAN_HWMCMCONTROLLER0_ID);`
Controller 4	`CAN1SR1_ISR`	Service for the transmission completion new entry event on TX FIFO Event.	`Can_17_McmCan_IsrTransmitHandler(CAN_17_MCMCAN_HWMCMKERNEL1_ID,CAN_17_MCMCAN_HWMCMCONTROLLER0_ID);`
Controller 4	`CAN1SR2_ISR`	Service on CAN controller within Bus Off mode.	`Can_17_McmCan_IsrBusOffHandler(CAN_17_MCMCAN_HWMCMKERNEL1_ID,CAN_17_MCMCAN_HWMCMCONTROLLER0_ID);`
Controller 4	`CAN1SR3_ISR`	Service on CAN receive FIFO water mark level or FIFO full level reached on Rx FIFO0 or Rx FIFO1.	`Can_17_McmCan_IsrRxFIFOHandler(CAN_17_MCMCAN_HWMCMKERNEL1_ID,CAN_17_MCMCAN_HWMCMCONTROLLER0_ID);`
Controller 5	`CAN1SR4_ISR`	Service on CAN data Reception through dedicated buffer.	`Can_17_McmCan_IsrReceiveHandler(CAN_17_MCMCAN_HWMCMKERNEL1_ID,CAN_17_MCMCAN_HWMCMCONTROLLER1_ID);`
Controller 5	`CAN1SR5_ISR`	Service for the transmission completion new entry event on TX FIFO Event.	`Can_17_McmCan_IsrTransmitHandler(CAN_17_MCMCAN_HWMCMKERNEL1_ID,CAN_17_MCMCAN_HWMCMCONTROLLER1_ID);`
Controller 5	`CAN1SR6_ISR`	Service on CAN controller within Bus Off mode.	`Can_17_McmCan_IsrBusOffHandler(CAN_17_MCMCAN_HWMCMKERNEL1_ID,CAN_17_MCMCAN_HWMCMCONTROLLER1_ID);`
Controller 5	`CAN1SR7_ISR`	Service on CAN receive FIFO water mark level or FIFO full level reached on Rx FIFO0 or Rx FIFO1.	`Can_17_McmCan_IsrRxFIFOHandler(CAN_17_MCMCAN_HWMCMKERNEL1_ID,CAN_17_MCMCAN_HWMCMCONTROLLER1_ID);`

(table continues...)

Table 4 (continued) Handling CAN interrupt lines:

Controller 6	`CAN1SR8_ISR`	Service on CAN data Reception through dedicated buffer.	`Can_17_McmCan_IsrReceiveHandler(CAN_17_MCMCAN_HWMCMKERNEL1_ID,CAN_17_MCMCAN_HWMCMCONTROLLER2_ID);`
Controller 6	`CAN1SR9_ISR`	Service for the transmission completion new entry event on TX FIFO Event.	`Can_17_McmCan_IsrTransmitHandler(CAN_17_MCMCAN_HWMCMKERNEL1_ID,CAN_17_MCMCAN_HWMCMCONTROLLER2_ID);`
Controller 6	`CAN1SR10_ISR`	Service on CAN controller within Bus Off mode.	`Can_17_McmCan_IsrBusOffHandler(CAN_17_MCMCAN_HWMCMKERNEL1_ID,CAN_17_MCMCAN_HWMCMCONTROLLER2_ID);`
Controller 6	`CAN1SR11_ISR`	Service on CAN receive FIFO water mark level or FIFO full level reached on Rx FIFO0 or Rx FIFO1.	`Can_17_McmCan_IsrRxFIFOHandler(CAN_17_MCMCAN_HWMCMKERNEL1_ID,CAN_17_MCMCAN_HWMCMCONTROLLER2_ID);`
Controller 7	`CAN1SR12_ISR`	Service on CAN data Reception through dedicated buffer.	`Can_17_McmCan_IsrReceiveHandler(CAN_17_MCMCAN_HWMCMKERNEL1_ID,CAN_17_MCMCAN_HWMCMCONTROLLER3_ID);`
Controller 7	`CAN1SR13_ISR`	Service for the transmission completion new entry event on TX FIFO Event.	`Can_17_McmCan_IsrTransmitHandler(CAN_17_MCMCAN_HWMCMKERNEL1_ID,CAN_17_MCMCAN_HWMCMCONTROLLER3_ID);`
Controller 7	`CAN1SR14_ISR`	Service on CAN controller within Bus Off mode.	`Can_17_McmCan_IsrBusOffHandler(CAN_17_MCMCAN_HWMCMKERNEL1_ID,CAN_17_MCMCAN_HWMCMCONTROLLER3_ID);`
Controller 7	`CAN1SR15_ISR`	Service on CAN receive FIFO water mark level or FIFO full level reached on Rx FIFO0 or Rx FIFO1.	`Can_17_McmCan_IsrRxFIFOHandler(CAN_17_MCMCAN_HWMCMKERNEL1_ID,CAN_17_MCMCAN_HWMCMCONTROLLER3_ID);`
Controller 8	`CAN2SR0_ISR`	Service on CAN data Reception through dedicated buffer.	`Can_17_McmCan_IsrReceiveHandler(CAN_17_MCMCAN_HWMCMKERNEL2_ID,CAN_17_MCMCAN_HWMCMCONTROLLER0_ID);`
Controller 8	`CAN2SR1_ISR`	Service for the transmission completion new entry event on TX FIFO Event.	`Can_17_McmCan_IsrTransmitHandler(CAN_17_MCMCAN_HWMCMKERNEL2_ID,CAN_17_MCMCAN_HWMCMCONTROLLER0_ID);`

(table continues...)

Table 4 (continued) Handling CAN interrupt lines:

Controller 8	`CAN2SR2_ISR`	Service on CAN controller within Bus Off mode.	`Can_17_McmCan_IsrBusOffHandler(CAN_17_MCMCAN_HWMCMKERNEL2_ID,CAN_17_MCMCAN_HWMCMCONTROLLER0_ID);`
Controller 8	`CAN2SR3_ISR`	Service on CAN receive FIFO water mark level or FIFO full level reached on Rx FIFO0 or Rx FIFO1.	`Can_17_McmCan_IsrRxFIFOHandler(CAN_17_MCMCAN_HWMCMKERNEL2_ID,CAN_17_MCMCAN_HWMCMCONTROLLER0_ID);`
Controller 9	`CAN2SR4_ISR`	Service on CAN data Reception through dedicated buffer.	`Can_17_McmCan_IsrReceiveHandler(CAN_17_MCMCAN_HWMCMKERNEL2_ID,CAN_17_MCMCAN_HWMCMCONTROLLER1_ID);`
Controller 9	`CAN2SR5_ISR`	Service for the transmission completion new entry event on TX FIFO Event.	`Can_17_McmCan_IsrTransmitHandler(CAN_17_MCMCAN_HWMCMKERNEL2_ID,CAN_17_MCMCAN_HWMCMCONTROLLER01_ID);`
Controller 9	`CAN2SR6_ISR`	Service on CAN controller within Bus Off mode.	`Can_17_McmCan_IsrBusOffHandler(CAN_17_MCMCAN_HWMCMKERNEL2_ID,CAN_17_MCMCAN_HWMCMCONTROLLER01_ID);`
Controller 9	`CAN2SR7_ISR`	Service on CAN receive FIFO water mark level or FIFO full level reached on Rx FIFO0 or Rx FIFO1.	`Can_17_McmCan_IsrRxFIFOHandler(CAN_17_MCMCAN_HWMCMKERNEL2_ID,CAN_17_MCMCAN_HWMCMCONTROLLER01_ID);`
Controller 10	`CAN2SR8_ISR`	Service on CAN data Reception through dedicated buffer.	`Can_17_McmCan_IsrReceiveHandler(CAN_17_MCMCAN_HWMCMKERNEL2_ID,CAN_17_MCMCAN_HWMCMCONTROLLER2_ID);`
Controller 10	`CAN2SR9_ISR`	Service for the transmission completion new entry event on TX FIFO Event.	`Can_17_McmCan_IsrTransmitHandler(CAN_17_MCMCAN_HWMCMKERNEL2_ID,CAN_17_MCMCAN_HWMCMCONTROLLER2_ID);`
Controller 10	`CAN2SR10_ISR`	Service on CAN controller within Bus Off mode.	`Can_17_McmCan_IsrBusOffHandler(CAN_17_MCMCAN_HWMCMKERNEL2_ID,CAN_17_MCMCAN_HWMCMCONTROLLER2_ID);`
Controller 10	`CAN2SR11_ISR`	Service on CAN receive FIFO water mark level or FIFO full level reached on Rx FIFO0 or Rx FIFO1.	`Can_17_McmCan_IsrRxFIFOHandler(CAN_17_MCMCAN_HWMCMKERNEL2_ID,CAN_17_MCMCAN_HWMCMCONTROLLER2_ID);`

(table continues...)

Table 4 (continued) Handling CAN interrupt lines:

Controller 11	`CAN2SR12_ISR`	Service on CAN data Reception through dedicated buffer.	`Can_17_McmCan_IsrReceiveHandler(CAN_17_MCMCAN_HWMCMKERNEL2_ID,CAN_17_MCMCAN_HWMCMCONTROLLER3_ID);`
Controller 11	`CAN2SR13_ISR`	Service for the transmission completion new entry event on TX FIFO Event.	`Can_17_McmCan_IsrTransmitHandler(CAN_17_MCMCAN_HWMCMKERNEL2_ID,CAN_17_MCMCAN_HWMCMCONTROLLER3_ID);`
Controller 11	`CAN2SR14_ISR`	Service on CAN controller within Bus Off mode.	`Can_17_McmCan_IsrBusOffHandler(CAN_17_MCMCAN_HWMCMKERNEL2_ID,CAN_17_MCMCAN_HWMCMCONTROLLER3_ID);`
Controller 11	`CAN2SR15_ISR`	Service on CAN receive FIFO water mark level or FIFO full level reached on Rx FIFO0 or Rx FIFO1.	`Can_17_McmCan_IsrRxFIFOHandler(CAN_17_MCMCAN_HWMCMKERNEL2_ID,CAN_17_MCMCAN_HWMCMCONTROLLER3_ID);`
Controller 12	`CAN3SR0_ISR`	Service on CAN data Reception through dedicated buffer.	`Can_17_McmCan_IsrReceiveHandler(CAN_17_MCMCAN_HWMCMKERNEL3_ID,CAN_17_MCMCAN_HWMCMCONTROLLER0_ID);`
Controller 12	`CAN3SR1_ISR`	Service for the transmission completion new entry event on TX FIFO Event.	`Can_17_McmCan_IsrTransmitHandler(CAN_17_MCMCAN_HWMCMKERNEL3_ID,CAN_17_MCMCAN_HWMCMCONTROLLER0_ID);`
Controller 12	`CAN3SR2_ISR`	Service on CAN controller within Bus Off mode.	`Can_17_McmCan_IsrBusOffHandler(CAN_17_MCMCAN_HWMCMKERNEL3_ID,CAN_17_MCMCAN_HWMCMCONTROLLER0_ID);`
Controller 12	`CAN3SR3_ISR`	Service on CAN receive FIFO water mark level or FIFO full level reached on Rx FIFO0 or Rx FIFO1.	`Can_17_McmCan_IsrRxFIFOHandler(CAN_17_MCMCAN_HWMCMKERNEL3_ID,CAN_17_MCMCAN_HWMCMCONTROLLER0_ID);`
Controller 13	`CAN3SR4_ISR`	Service on CAN data Reception through dedicated buffer.	`Can_17_McmCan_IsrReceiveHandler(CAN_17_MCMCAN_HWMCMKERNEL3_ID,CAN_17_MCMCAN_HWMCMCONTROLLER1_ID);`
Controller 13	`CAN3SR5_ISR`	Service for the transmission completion new entry event on TX FIFO Event.	`Can_17_McmCan_IsrTransmitHandler(CAN_17_MCMCAN_HWMCMKERNEL3_ID,CAN_17_MCMCAN_HWMCMCONTROLLER1_ID);`

(table continues...)

Table 4 (continued) Handling CAN interrupt lines:

Controller 13	CAN3SR6_ISR	Service on CAN controller within Bus Off mode.	Can_17_McmCan_IsrBusoffHandler(CAN_17_MCMCAN_HWMCMKERNEL3_ID,CAN_17_MCMCAN_HWMCMCONTROLLER1_ID);
Controller 13	CAN3SR7_ISR	Service on CAN receive FIFO water mark level or FIFO full level reached on Rx FIFO0 or Rx FIFO1.	Can_17_McmCan_IsrRxFIFOHandler(CAN_17_MCMCAN_HWMCMKERNEL3_ID,CAN_17_MCMCAN_HWMCMCONTROLLER1_ID);
Controller 14	CAN3SR8_ISR	Service on CAN data Reception through dedicated buffer.	Can_17_McmCan_IsrReceiveHandler(CAN_17_MCMCAN_HWMCMKERNEL3_ID,CAN_17_MCMCAN_HWMCMCONTROLLER2_ID);
Controller 14	CAN3SR9_ISR	Service for the transmission completion new entry event on TX FIFO Event.	Can_17_McmCan_IsrTransmitHandler(CAN_17_MCMCAN_HWMCMKERNEL3_ID,CAN_17_MCMCAN_HWMCMCONTROLLER2_ID);
Controller 14	CAN3SR10_ISR	Service on CAN controller within Bus Off mode.	Can_17_McmCan_IsrBusoffHandler(CAN_17_MCMCAN_HWMCMKERNEL3_ID,CAN_17_MCMCAN_HWMCMCONTROLLER2_ID);
Controller 14	CAN3SR11_ISR	Service on CAN receive FIFO water mark level or FIFO full level reached on Rx FIFO0 or Rx FIFO1.	Can_17_McmCan_IsrRxFIFOHandler(CAN_17_MCMCAN_HWMCMKERNEL3_ID,CAN_17_MCMCAN_HWMCMCONTROLLER2_ID);
Controller 15	CAN3SR12_ISR	Service on CAN data Reception through dedicated buffer.	Can_17_McmCan_IsrReceiveHandler(CAN_17_MCMCAN_HWMCMKERNEL3_ID,CAN_17_MCMCAN_HWMCMCONTROLLER3_ID);
Controller 15	CAN3SR13_ISR	Service for the transmission completion new entry event on TX FIFO Event.	Can_17_McmCan_IsrTransmitHandler(CAN_17_MCMCAN_HWMCMKERNEL3_ID,CAN_17_MCMCAN_HWMCMCONTROLLER3_ID);
Controller 15	CAN3SR14_ISR	Service on CAN controller within Bus Off mode.	Can_17_McmCan_IsrBusoffHandler(CAN_17_MCMCAN_HWMCMKERNEL3_ID,CAN_17_MCMCAN_HWMCMCONTROLLER3_ID);
Controller 15	CAN3SR15_ISR	Service on CAN receive FIFO water mark level or FIFO full level reached on Rx FIFO0 or Rx FIFO1.	Can_17_McmCan_IsrRxFIFOHandler(CAN_17_MCMCAN_HWMCMKERNEL3_ID,CAN_17_MCMCAN_HWMCMCONTROLLER3_ID);

(table continues...)

Table 4 (continued) Handling CAN interrupt lines:

Controller 16	CAN4SR0_ISR	Service on CAN data Reception through dedicated buffer.	Can_17_McmCan_IsrReceiveHandler(CAN_17_MCMCAN_HWMCMKERNEL4_ID,CAN_17_MCMCAN_HWMCMCONTROLLER0_ID);
Controller 16	CAN4SR1_ISR	Service for the transmission completion new entry event on TX FIFO Event.	Can_17_McmCan_IsrTransmitHandler(CAN_17_MCMCAN_HWMCMKERNEL4_ID,CAN_17_MCMCAN_HWMCMCONTROLLER0_ID);
Controller 16	CAN4SR2_ISR	Service on CAN controller within Bus Off mode.	Can_17_McmCan_IsrBusoffHandler(CAN_17_MCMCAN_HWMCMKERNEL4_ID,CAN_17_MCMCAN_HWMCMCONTROLLER0_ID);
Controller 16	CAN4SR3_ISR	Service on CAN receive FIFO water mark level or FIFO full level reached on Rx FIFO0 or Rx FIFO1.	Can_17_McmCan_IsrRxFIFOHandler(CAN_17_MCMCAN_HWMCMKERNEL4_ID,CAN_17_MCMCAN_HWMCMCONTROLLER0_ID);
Controller 17	CAN4SR4_ISR	Service on CAN data Reception through dedicated buffer.	Can_17_McmCan_IsrReceiveHandler(CAN_17_MCMCAN_HWMCMKERNEL4_ID,CAN_17_MCMCAN_HWMCMCONTROLLER1_ID);
Controller 17	CAN4SR5_ISR	Service for the transmission completion new entry event on TX FIFO Event.	Can_17_McmCan_IsrTransmitHandler(CAN_17_MCMCAN_HWMCMKERNEL4_ID,CAN_17_MCMCAN_HWMCMCONTROLLER1_ID);
Controller 17	CAN4SR6_ISR	Service on CAN controller within Bus Off mode.	Can_17_McmCan_IsrBusoffHandler(CAN_17_MCMCAN_HWMCMKERNEL4_ID,CAN_17_MCMCAN_HWMCMCONTROLLER1_ID);
Controller 17	CAN4SR7_ISR	Service on CAN receive FIFO water mark level or FIFO full level reached on Rx FIFO0 or Rx FIFO1.	Can_17_McmCan_IsrRxFIFOHandler(CAN_17_MCMCAN_HWMCMKERNEL4_ID,CAN_17_MCMCAN_HWMCMCONTROLLER1_ID);
Controller 18	CAN4SR8_ISR	Service on CAN data Reception through dedicated buffer.	Can_17_McmCan_IsrReceiveHandler(CAN_17_MCMCAN_HWMCMKERNEL4_ID,CAN_17_MCMCAN_HWMCMCONTROLLER2_ID);
Controller 18	CAN4SR9_ISR	Service for the transmission completion new entry event on TX FIFO Event.	Can_17_McmCan_IsrTransmitHandler(CAN_17_MCMCAN_HWMCMKERNEL4_ID,CAN_17_MCMCAN_HWMCMCONTROLLER2_ID);

(table continues...)

Table 4 (continued) Handling CAN interrupt lines:

Controller 18	`CAN4SR10_ISR`	Service on CAN controller within Bus Off mode.	`Can_17_McmCan_IsrBusoffHandler(CAN_17_MCMCAN_HWMCMKERNEL4_ID,CAN_17_MCMCAN_HWMCMCONTROLLER2_ID);`
Controller 18	`CAN4SR11_ISR`	Service on CAN receive FIFO water mark level or FIFO full level reached on Rx FIFO0 or Rx FIFO1.	`Can_17_McmCan_IsrRxFIFOHandler(CAN_17_MCMCAN_HWMCMKERNEL4_ID,CAN_17_MCMCAN_HWMCMCONTROLLER2_ID);`
Controller 19	`CAN4SR12_ISR`	Service on CAN data Reception through dedicated buffer.	`Can_17_McmCan_IsrReceiveHandler(CAN_17_MCMCAN_HWMCMKERNEL4_ID,CAN_17_MCMCAN_HWMCMCONTROLLER3_ID);`
Controller 19	`CAN4SR13_ISR`	Service for the transmission completion new entry event on TX FIFO Event.	`Can_17_McmCan_IsrTransmitHandler(CAN_17_MCMCAN_HWMCMKERNEL4_ID,CAN_17_MCMCAN_HWMCMCONTROLLER3_ID);`
Controller 19	`CAN4SR14_ISR`	Service on CAN controller within Bus Off mode.	`Can_17_McmCan_IsrBusoffHandler(CAN_17_MCMCAN_HWMCMKERNEL4_ID,CAN_17_MCMCAN_HWMCMCONTROLLER3_ID);`
Controller 19	`CAN4SR15_ISR`	Service on CAN receive FIFO water mark level or FIFO full level reached on Rx FIFO0 or Rx FIFO1.	`Can_17_McmCan_IsrRxFIFOHandler(CAN_17_MCMCAN_HWMCMKERNEL4_ID,CAN_17_MCMCAN_HWMCMCONTROLLER3_ID);`

Invoking of interrupt handlers provided by the driver must be done by the user. A sample invocation for controller 0, dedicated Rx interrupt is shown as follows:

```
#include "Can_17_McmCan.h"
 ISR(CAN0SR0_ISR)
 {
 /* Enable Global Interrupts */
 ENABLE();

 /* Call CAN Rx Interrupt function for dedicated buffer */

Can_17_McmCan_IsrReceiveHandler(CAN_17_MCMCAN_HWMCMKERNEL0_ID,CAN_17_MCMCAN_HWMCMCONTROLLER0_ID)
;
 }
```

1.1.4.7 Example usage

Configuring the driver and related modules

The AUTOSAR configuration parameter CanControllerBaseAddress is used with a selection of address for the mapping of CAN controller in the hardware to the configured CAN controller Id.

Note: Kernel specific RAM allocation per controller is based on the CAN hardware objects allocated to that controller. In the case of a controller with FD baudrate configured the RAM allocated per hardware object is 4 times higher. Hence, in order to have an optimised RAM memory utilisation it is recommended that a controller with FD baudrate configured have only hardware objects that use CAN FD communication.

Configuring of CAN hardware object

The MCMCAN hardware is supported with Rx FIFO and dedicated Rx buffer for reception and Tx Queue and Tx dedicated buffer for the transmission operation. The user can select the hardware object buffer type while configuring hardware object handler (HOH).

The following rules are considered while configuring the buffer type selection for HRH and HTH:

- If the CanObjectType value is configured with RECEIVE and CanHwObjectCount value is equal to 1 then the buffer type of HRH is assigned as Rx dedicated buffer.
- If the CanObjectType value is configured with RECEIVE and CanHwObjectCount value is greater than 1 then the buffer type of HRH is assigned as Rx FIFO. First instance shall be considered with buffer type as Rx FIFO0 and if second one is available for the same controller, it is considered with buffer type as Rx FIFO1.
- The user can configure Rx dedicated buffer or Rx FIFO (0 and 1) or the combination of the two types of the receive operations in any order.
- The user can configure Tx dedicated buffer or Tx Queue or the combination of the two types of the transmit operations in any order.
- If the CanObjectType value is configured with TRANSMIT and CanHwObjectCount value is greater than 1 then the buffer type of HTH is assigned as Tx Queue. The CanMultiplexedTransmission check needs to be done to make CanHwObjectCount value greater than 1 for a TRANSMIT message.
- If the CanObjectType value is configured with TRANSMIT and CanHwObjectCount value is equal to 1 then the buffer type of HTH is assigned as Tx dedicated buffer.
- In case Rxprocessing or Txprocessing is configured as MIXED at least one of the hardware object should be configured to polling and another one should be configured to interrupt.

CanObjectId configuration rules

- CanObjectId shall be unique and shall start with 0 and continue without any gap
- HRHs (CanObjectId) belonging to a controller shall be grouped together.
- HTHs (CanObjectId) belonging to a controller shall be grouped together.

  Ensure HRHs of all controllers are grouped before the HTHs of all controllers, then the entire HRH id shall have lower CanObjectId than all HTH.

Initializing the CAN driver

```
/* Mcu Initialization */
 Mcu_Init(&Mcu_Config);
 Mcu_InitClock(0U);
 while(Mcu_GetPllStatus() != MCU_PLL_LOCKED);
 Mcu_DistributePllClock ();
 /* Port Initialization */
 Port_Init(&Port_Config);
 /* CAN Initialization */
 Can_17_McmCan_Init(&Can_17_McmCan_Config);
 /* Further APIs of CAN driver can be called now */
```

CAN controller mode change

After CAN initialization the following sequence may be followed.

```
/* Set the controller with state as START */
 Can_17_McmCan_SetControllerMode (Can_17_McmCanConf_CanController_CanController_0,CAN_T_START);
 Can_17_McmCan_SetControllerMode (Can_17_McmCanConf_CanController_CanController_1,CAN_T_START);
```

Disabling and enabling CAN controller interrupts

```
/* Disable the interruption by CAN event */
 Can_17_McmCan_DisableControllerInterrupts(Can_17_McmCanConf_CanController_CanController_0);
 /* Request Write operation */
 Can_17_McmCan_Write(10, &PduInfo_ExtId[0]) ;
 /* Enable the interruption by CAN event */
 Can_17_McmCan_EnableControllerInterrupts(Can_17_McmCanConf_CanController_CanController_0);
 /* Notification can be expected now */
```

Re-initializing CAN controller baudrate

```
/* Set the controller with state as STOP */
 Can_17_McmCan_SetControllerMode (Can_17_McmCanConf_CanController_CanController_0,CAN_T_STOP);
 Can_17_McmCan_SetControllerMode (Can_17_McmCanConf_CanController_CanController_1,CAN_T_STOP);
 /* Set the baudrate */
 #if (CAN_17_MCMCAN_SET_BAUDRATE_API == STD_ON)
 Can_17_McmCan_SetBaudrate(Can_17_McmCanConf_CanController_CanController_0, 0);
 #endif

 /* Set the baudrate */
 #if (CAN_17_MCMCAN_SET_BAUDRATE_API == STD_ON)
 Can_17_McmCan_SetBaudrate(Can_17_McmCanConf_CanController_CanController_1, 0);
 #endif
 /* Set the controller with state as START */
 Can_17_McmCan_SetControllerMode (Can_17_McmCanConf_CanController_CanController_0,CAN_T_START);
 Can_17_McmCan_SetControllerMode (Can_17_McmCanConf_CanController_CanController_1,CAN_T_START);
```

Activating and de-activating the pretended networking

```
/* Set the controller with state as START */
 Can_17_McmCan_SetControllerMode (Can_17_McmCanConf_CanController_CanController_0,CAN_T_START);
 Can_17_McmCan_SetControllerMode (Can_17_McmCanConf_CanController_CanController_1,CAN_T_START);
 /* Activate Pretended networking */
 #if (CAN_17_MCMCAN_PUBLIC_ICOM_SUPPORT == STD_ON)
 Can_17_McmCan_SetIcomConfiguration(Can_17_McmCanConf_CanController_CanController_0, 1);
 #endif
 /* Activate Pretended networking */
 #if (CAN_17_MCMCAN_PUBLIC_ICOM_SUPPORT == STD_ON)
 Can_17_McmCan_SetIcomConfiguration(Can_17_McmCanConf_CanController_CanController_1, 2);
 #endif

 /* Deactivate Pretended networking */
 #if (CAN_17_MCMCAN_PUBLIC_ICOM_SUPPORT == STD_ON)
 Can_17_McmCan_SetIcomConfiguration(Can_17_McmCanConf_CanController_CanController_0, 0);
 #endif
 /* Deactivate Pretended networking */
 #if (CAN_17_MCMCAN_PUBLIC_ICOM_SUPPORT == STD_ON)
 Can_17_McmCan_SetIcomConfiguration(Can_17_McmCanConf_CanController_CanController_1, 0);
 #endif
```

De-initializing the CAN driver

```
/* Mcu Initialization */
 Mcu_Init(&Mcu_Config);
 Mcu_InitClock(0U);
 while(Mcu_GetPllStatus() != MCU_PLL_LOCKED);
 Mcu_DistributePllClock ();
 /* Port Initialization */
 Port_Init(&Port_Config);
 /* CAN Initialization */
 Can_17_McmCan_Init(&Can_17_McmCan_Config);

 /* Set the controller with state as START */
 Can_17_McmCan_SetControllerMode (Can_17_McmCanConf_CanController_CanController_0,CAN_T_START);
 Can_17_McmCan_SetControllerMode (Can_17_McmCanConf_CanController_CanController_1,CAN_T_START);

 /* Data transmission by Controller 0 to 1 */
 Can_17_McmCan_Write(8, &PduInfo_1[0]) ;

 Can_17_McmCan_SetControllerMode (Can_17_McmCanConf_CanController_CanController_0,CAN_T_STOP);
 Can_17_McmCan_SetControllerMode (Can_17_McmCanConf_CanController_CanController_1,CAN_T_STOP);

 /* Call CAN de-Initialization function */
 Can_17_McmCan_DeInit();
```

Transmission and reception in polling Mode

```
/* Mcu Initialization */
 Mcu_Init(&Mcu_Config);
 Mcu_InitClock(0U);
 while(Mcu_GetPllStatus() != MCU_PLL_LOCKED);
 Mcu_DistributePllClock ();
 /* Port Initialization */
 Port_Init(&Port_Config);
 /* CAN Initialization */
 Can_17_McmCan_Init(&Can_17_McmCan_Config);

 /* Set the controller with state as START */
 Can_17_McmCan_SetControllerMode (Can_17_McmCanConf_CanController_CanController_0,CAN_T_START);
 Can_17_McmCan_SetControllerMode (Can_17_McmCanConf_CanController_CanController_1,CAN_T_START);

 /* Data transmission by Controller 0 to 1 */
 Can_17_McmCan_Write(8, &PduInfo_1[0]);

 /* In Scheduled function call poll for the reception of the message */
 /* Reception is polled for and shall raise Can If notification in controller 1 */
 Can_17_McmCan_MainFunction_Read_x();

 /* Transmission is polled for and shall raise a Can If notification in controller 0 */
 Can_17_McmCan_MainFunction_Write_x();
```

Possible values of CanControllerBaseAddress container

Table 5 Controller base address List

Sl. No.	Controller	Base address
1	Controller 0 (Node 00)	0xF0208100
2	Controller 1 (Node 01)	0xF0208500
3	Controller 2 (Node02)	0xF0208900
4	Controller 3 (Node03)	0xF0208D00
5	Controller 4 (Node10)	0xF0218100
6	Controller 5 (Node11)	0xF0218500
7	Controller 6 (Node12)	0xF0218900
8	Controller 7 (Node 13)	0xF0218D00
9	Controller 8 (Node20)	0xF0228100
10	Controller 9 (Node 21)	0xF0228500
11	Controller 10 (Node 22)	0xF0228900
12	Controller 11 (Node 23)	0xF0228D00
13	Controller 12 (Node 30)	0xF0238100
14	Controller 13 (Node 31)	0xF0238500

(table continues...)

Table 5 (continued) Controller base address List

Sl. No.	Controller	Base address
15	Controller 14 (Node 32)	0xF0238900
16	Controller 15 (Node 33)	0xF0238D00
17	Controller 16 (Node 40)	0xF0348100
18	Controller 17 (Node 41)	0xF0348500
19	Controller 18 (Node 42)	0xF0348900
20	Controller 19 (Node 43)	0xF0348D00

Above list contains the base address of the controller nodes supported, these address are to be updated by the integrator in the respective CanController configuration for mapping the controller to respective CAN hardware object (refer to CanControllerBaseAddress in the CanController container).

1.1.5 Key architectural considerations

1.1.5.1 CAN interrupt handling

Tx event handling

TEFN (Tx Event FIFO New Entry) is the only event enabled for handling the Tx event FIFO.

For every element added in the event FIFO, this event will be generated. TEFL(Tx Event FIFO Lost) and TEFF (Tx Event FIFO Full) events are cleared in the same handler. If TEFL is SET, this indicates that the Tx event is lost. In this case the CAN driver will raise an error as CAN_17_MCMCAN_E_DATALOST. The same error is raised during multi-period transmit in Can_17_McmCan_MainFunction_Write_x. This indicates that the bus is loaded, and hence, no sufficient time is provided to process the Tx notifications to upper layer.

Rx dedicated handling

DRX (Message stored to Dedicated Rx Buffer) is the event raised when one of the dedicated buffer is updated with the message from CAN bus. Corresponding independent dedicated buffer bits in NDAT1 and NDAT2 will be SET if message is copied.

Rx FIFO handling

RFxW (Rx FIFO x Watermark reached) and RFxF (Rx FIFO x Full) are the two events that are enabled for handling the Rx FIFO (x represents FIFO 0 or FIFO 1).

Watermark is used for CAN HW to trigger the interrupt when certain number of messages are received in FIFO. If RFxL is SET, this indicates that the FIFO message is lost for which CAN driver will raise the error as CAN_17_MCMCAN_E_DATALOST. Also RFxL is cleared by CAN driver. This indicates that the bus is loaded and no sufficient time is provided to process the received frames.

On watermark interrupt, handler processes maximum of configured FIFO elements. If messages are received while the Rx FIFO messages are being processed; and; if number of messages received is greater than the configured threshold level on exit of interrupt handler, watermark interrupt will not be triggered. Hence all messages will be processed only on FULL interrupt.

Bus Off handling

Bus off interrupt is enabled in order to indicate that the bus is faulty to notify upper layer for handling the erroneous bus. After a bus off occurred, further Can_Write requests should be only issued after a successful transition to STARTED state.

Note: TEFN, RF0W, RF1W and DRX bits have the RWH attribute. As there is a possibility of hardware updating the TEFN, RF0W, RF1W and DRX bits in background in same cycle when software is trying to clear this bit, IR remains updated due to hardware write. For this software has to clear the flag repeatedly to ensure that the intended flag *is cleared before processing the interrupt. So all the above listed flags are cleared in a loop with an exit condition for maximum of three retries. Note that retry mechanism is not implemented for Busoff interrupt since the bit setting and clearing from software cannot occur in same clock cycle.*

1.1.5.2 Multi-period Tx and Rx

- The multi-period Tx and Rx main function calls Can_17_McmCan_MainFunction_Read_(x) and Can_17_McmCan_MainFunction_Write_(x), where number of main function calls generated is based on the number of main function period entries configured in Can/CanGeneral/CanMainFunctionRWPeriods configured are generated with the suffix 'x' changing based on the index of Can/CanGeneral/CanMainFunctionRWPeriods/*[] configured
- The multi-period Tx and Rx will be available only when the number of Can/CanGeneral/CanMainFunctionRWPeriods configured is greater than one and at least one controller is configured with Rx (for Can_17_McmCan_MainFunction_Read_(x)) or Tx (for Can_17_McmCan_MainFunction_Write_(x)) in polling mode or in mixed mode
- The default function call Can_17_McmCan_MainFunction_Read will not be available when Can_17_McmCan_MainFunction_Read_(x) is generated
- The default function call Can_17_McmCan_MainFunction_Write will not be available when Can_17_McmCan_MainFunction_Write_(x) is generated
- If at least one controller is configured to have Tx as polling mode or mixed mode and CanMainFunctionPeriod is greater than one, then the function Can_17_McmCan_MainFunction_Write_(x) is generated for all values of 'x' available in the index of CanMainFunctionPeriod, even if no controller Tx operating is polling mode or mixed mode is referring to that CanMainFunctionPeriod index, this function generated for such CanMainFunctionPeriod is effectively an empty function with no actions performed
- If at least one controller is configured to have Rx as polling mode or mixed mode and CanMainFunctionPeriod is greater than one, then the function Can_17_McmCan_MainFunction_Read_(x) is generated for all values of 'x' available in the index of CanMainFunctionPeriod, even if no controller Rx operating is polling mode or mixed mode is referring to that CanMainFunctionPeriod index, this function generated for such CanMainFunctionPeriod is effectively an empty function with no actions performed
- The multi-period Tx and Rx functions generated for a CanMainFunctionPeriod will give out notifications only for hardware object handle events (Tx / Rx) associated to that CanMainFunctionPeriod

1.1.5.3 Mixed Mode Rx/Tx Processing

In case a controller's Rx/Tx processing is selected as MIXED, the hardware objects associated with that particular controller can be configured to be processed via INTERRUPT or POLLING method.

Mixed mode for Rx processing:

Rx FIFO0/1 or dedicated objects can be configured to INTERRUPT or POLLING, if the Rx processing is configured as MIXED.

In a controller configured as MIXED Rx processing, if some of the hardware objects are selected as INTERRUPT and some as POLLING, all dedicated hardware objects would still receive an interrupt. This is because the interrupt lines (DRX bit) are shared in case of dedicated. However, only those hardware objects configured as INTERRUPT would generate a notification. For the notification of hardware objects configured as POLLING Can_17_McmCan_MainFunction_Read should be called.

Mixed mode for Tx processing:

Tx queue or dedicated objects can be configured to INTERRUPT or POLLING for a controller configured in MIXED mode TX processing.

Even if some of the hardware objects are selected as INTERRUPT and some as POLLING, all hardware objects would still receive an interrupt. This is because, TEFN is common for all hardware objects in a controller. However, only those hardware objects configured as INTERRUPT would generate a notification. For the notification of hardware objects configured as POLLING Can_17_McmCan_MainFunction_Write should be called. Note that the Tx slots are released only after providing respective notifications.

1.1.5.4 L-PDU Callout

The AUTOSAR CAN module supports optional L-PDU callouts on every reception of L-PDU. Can_17_McmCan_PBcfg.c will contain the prototype of the L-PDU callout. Since the name of the callout function is provided through configuration, it cannot be in static file.

In case of ICom, the return value of the callout is not checked. This is because in case callout function returns false, RxIndication would not get triggered.

L-PDU Callout will only be invoked when the controller is in STARTED state and no callout will be provided when the device is in any other states.

1.2 Assumptions of Use (AoU)

There are no Assumptions of Use (AoU)s for the CAN driver.

1.3 Reference information

1.3.1 Configuration interfaces

Supported configuration variant: Post-Build

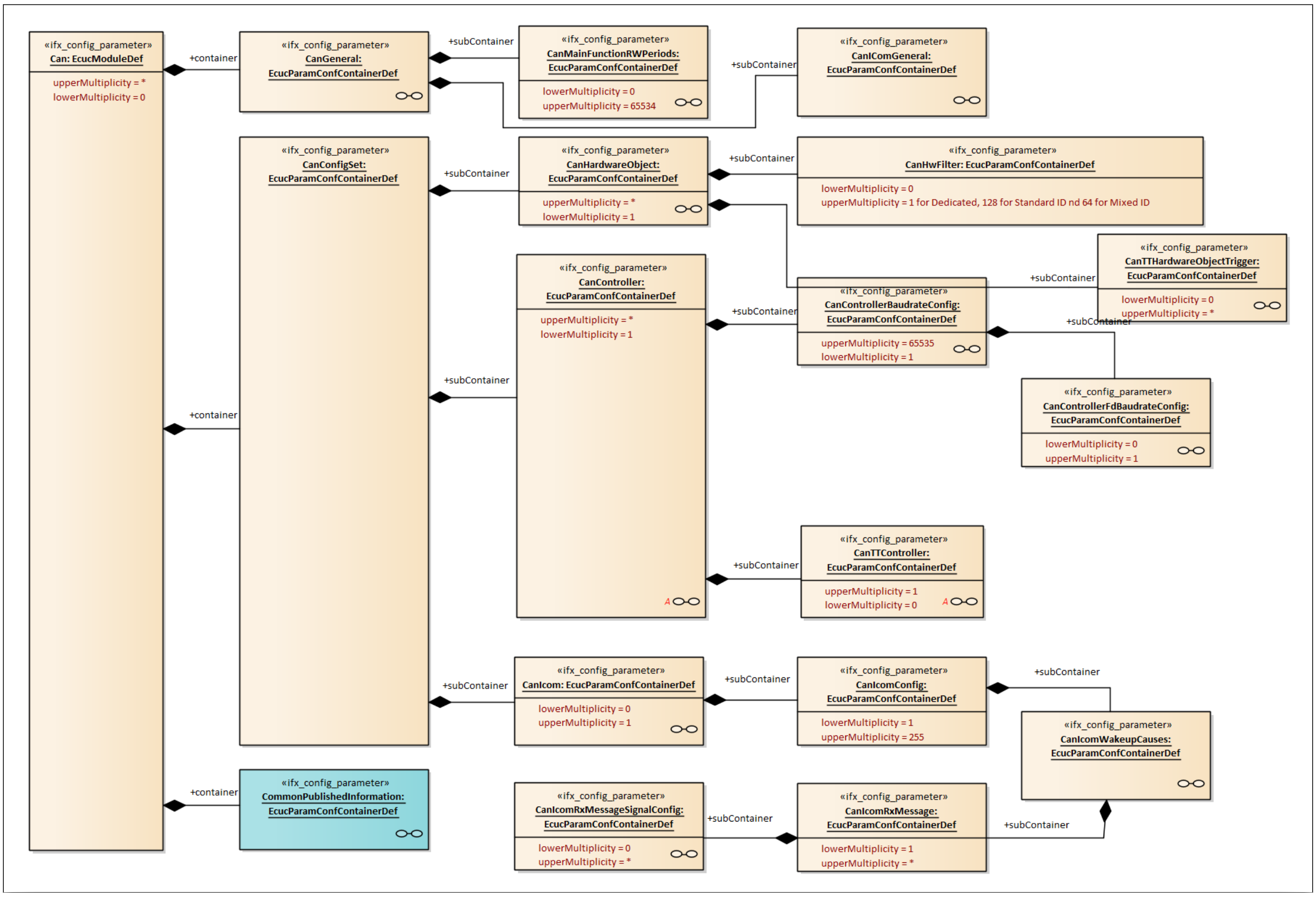

Figure 4 **Container hierarchy along with their configuration parameters**

1.3.1.1 Container: CanConfigSet

The container contains the configuration parameters and sub containers of the AUTOSAR CAN driver.

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

1.3.1.2 Container: CanController

The container contains the configuration parameters of the CAN controller(s).

The upper multiplicity of this container depends on the number of CAN controllers configured. This number cannot exceed the total number of CAN controllers present in a device.

Post-Build Variant Multiplicity: FALSE

Multiplicity Configuration Class: Pre-Compile

1.3.1.2.1 CanBusoffProcessing

Table 6 Specification for CanBusoffProcessing

Name	`CanBusoffProcessing`		
Description	Specifies the way bus off event on the controller is notified. Enables/disables the Can_17_McmCan_MainFunction_BusOff() API for handling bus-off events in the polling mode. It is applicable only when CanControllerActivation is set to TRUE. In case CanControllerActivation is set to FALSE, a configuration error when user tries to configure this parameter. The default value is set to INTERRUPT to set all the CAN driver configuration parameter default values to be interrupt compatible.		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	INTERRUPT: event is notified by the interrupt mechanism POLLING: event is notified when polled		
Default value	INTERRUPT		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	CanControllerActivation		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.2 CanControllerActivation

Table 7 Specification for CanControllerActivation

Name	`CanControllerActivation`		
Description	Defines if a CAN controller is used in the configuration. The default value is set to TRUE as a new controller added is automatically taken as activated.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	TRUE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-

(table continues...)

Table 7 (continued) Specification for CanControllerActivation

Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.3 CanControllerBaseAddress

Table 8 Specification for CanControllerBaseAddress

Name	`CanControllerBaseAddress`		
Description	The parameter specifies the CAN controller base address. It is applicable only when CanControllerActivation is set to TRUE. In case CanControllerActivation is set as FALSE user would receive a configuration error while trying to configure this parameter. The default value is set to the base address of CAN controller 0. The controller base address values for each controller is mentioned in the HW UM. In case the controller for the particular device is not present the configuration of the base address with respect to the particular controller will give a configuration error. The selection of address (not configurable by the user) is to be done for the mapping of CAN controller in the hardware to the configured CAN controller Id.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 4294967295		
Default value	4028662016		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	CanControllerActivation		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.4 CanControllerDefaultBaudrate

Table 9 Specification for CanControllerDefaultBaudrate

Name	`CanControllerDefaultBaudrate`

(table continues...)

Table 9 (continued) Specification for CanControllerDefaultBaudrate

Description	Reference to baudrate configuration container configured for the CAN controller. It is applicable only when CanControllerActivation is set to TRUE. In case CanControllerActivation is set to FALSE, a configuration error will be reported when the user tries to configure this parameter.		
Multiplicity	1..1	**Type**	EcucReferenceDef
Range	Reference to Node: CanControllerBaudrateConfig		
Default value	NULL		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	CanControllerActivation		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.5 CanControllerEcucPartitionRef

Table 10 Specification for CanControllerEcucPartitionRef

Name	`CanControllerEcucPartitionRef`		
Description	The parameter maps the CAN controller to zero or one ECUC partitions. The ECUC partition referenced is a subset of the ECUC partitions where the CAN driver is mapped to. There is no provision in the CAN driver to support ECUC partitions hence this parameter is not editable.		
Multiplicity	0..1	**Type**	EcucReferenceDef
Range	Reference to Node:		
Default value	NULL		
Post-build variant value	FALSE	**Post-build variant multiplicity**	FALSE
Value configuration class	Post-Build	**Multiplicity configuration class**	Post-Build
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar version 4.4.0.		

1.3.1.2.6 CanControllerId

Table 11 Specification for CanControllerId

Name	`CanControllerId`		
Description	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. The value 'n' depends on the number of CAN controllers supported by the hardware and is dependent on the device being used. It is applicable only when CanControllerActivation is set to TRUE. In case CanControllerActivation is set to FALSE, then a configuration error will be reported when the user tries to configure this parameter. The default value of CanControllerId is set to 0 representing the first index.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - n-1		
Default value	0		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	ECU
Dependency	CanControllerActivation		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.7 CanControllerLoopbackEnable

Table 12 Specification for CanControllerLoopbackEnable

Name	`CanControllerLoopbackEnable`		
Description	The parameter specifies whether the internal loop back mode is enabled or not for the controller This setting is applicable only when CanControllerActivation is set to TRUE. It is applicable only when CanControllerActivation is set to TRUE. In case CanControllerActivation is set to FALSE, a configuration error will be reported when the user tries to configure this parameter. By default, the optional interface APIs are disabled to minimize the executable code size.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-

(table continues...)

Table 12 (continued) Specification for CanControllerLoopbackEnable

Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	CanControllerActivation		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.8 CanCpuClockRef

Table 13 Specification for CanCpuClockRef

Name	`CanCpuClockRef`		
Description	Reference to the CPU clock configuration, which is set in the MCU driver configuration. It is applicable only when CanControllerActivation is set to TRUE. It also depends on the McuClockReferencePoint. CanCpuClockRef configuration parameter is made as non-editable as MCMCAN driver makes use of CPU peripheral bus clock for its clock, the CPU peripheral bus clock is referenced by the container CanPeripheralBusClockRef. The configuration parameter, even though not used, shall be present in the schema to maintain the AUTOSAR schema.		
Multiplicity	1..1	**Type**	EcucReferenceDef
Range	Reference to Node: McuClockReferencePoint		
Default value	NULL		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	McuClockReferencePoint		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.9 CanPeripheralBusClockRef

Table 14 Specification for CanPeripheralBusClockRef

Name	`CanPeripheralBusClockRef`
Description	Reference to the CPU peripheral bus clock configuration, which is set in the MCU driver configuration. It is applicable only when CanControllerActivation is set to TRUE. It also depends on the McuClockReferencePointConfig. The parameter is used instead of the CanCpuClockRef.

(table continues...)

Table 14 (continued) Specification for CanPeripheralBusClockRef

Multiplicity	1..1	**Type**	EcucReferenceDef
Range	Reference to Node: McuClockReferencePointConfig		
Default value	NULL		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	McuClockReferencePointConfig, CanControllerActivation		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.10 CanRxInputSelection

Table 15 Specification for CanRxInputSelection

Name	`CanRxInputSelection`		
Description	Provides alternative port pin selection for receive input line. It is applicable only when CanControllerActivation is set to TRUE and CanControllerLoopbackEnable is FALSE. In case this condition is not met, a configuration error will be reported when the user tries to configure this parameter. Default value: CANxx_RXDz: Receive input line CANxx_RXDz. Where 'z' will vary depending on device variant. The default value shall be set to CANxx_RXDA as it is the first Rx input selection available for all CAN controllers.		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	CANxx_RXDz: Receive input line CANxx_RXDz. Where, 'z' will vary depending on the device variant The default value is set to CANxx_RXDA		
Default value	CANxx_RXDz		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	CanControllerLoopbackEnable, CanControllerActivation		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.11 CanRxProcessing

Table 16 Specification for CanRxProcessing

Name	`CanRxProcessing`		
Description	Specifies the way reception event on the controller is notified. It is applicable only when CanControllerActivation is set to TRUE. In case CanControllerActivation is set to FALSE, a configuration error will be reported when the user tries to configure this parameter. The default value is set to INTERRUPT to set all the CAN driver configuration parameter default values to be interrupt compatible. *Note: In case Rxprocessing is configured as MIXED and if all hardware objects are configured to use polling or interrupt then a warning will be generated in configuration tool. Hence, in case the user wants to use only polling or only interrupt for the hardware objects associated with a certain controller, the user should select CanRxProcessing as POLLING or INTERRUPT respectively.*		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	INTERRUPT: event is notified by the interrupt mechanism MIXED: Mixed mode of operation i.e. event is notified when polled or through interrupt based on whether the hardware object uses polling. POLLING: event is notified when polled		
Default value	INTERRUPT		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	CanControllerActivation		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.12 CanTxProcessing

Table 17 Specification for CanTxProcessing

Name	`CanTxProcessing`

(table continues...)

Table 17 **(continued) Specification for CanTxProcessing**

Description	Specifies the way transmission event on the controller is notified. Enables/disables Can_17_McmCan_MainFunction_Write() API. It is applicable only if CanControllerActivation is set to TRUE. In case CanControllerActivation is set to FALSE, a configuration error will be reported when the user tries to configure this parameter. *Note: In case Txprocessing is configured as MIXED and if all hardware objects are configured to use polling or interrupt then a warning will be generated in configuration tool. Hence, in case the user wants to use only polling or only interrupt for the hardware objects associated with a certain controller, the user should select CanTxProcessing as POLLING or INTERRUPT respectively.*		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	INTERRUPT: event is notified by the interrupt mechanism MIXED: Mixed mode of operation i.e. event is notified when polled or through interrupt based on whether the hardware object uses polling. POLLING: event is notified when polled		
Default value	INTERRUPT		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	CanControllerActivation		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.13 CanWakeupFunctionalityAPI

Table 18 **Specification for CanWakeupFunctionalityAPI**

Name	`CanWakeupFunctionalityAPI`
Description	Adds/removes the Can_17_McmCan_CheckWakeup() service from the code True: Can_17_McmCan_CheckWakeup can be used False: Can_17_McmCan_CheckWakeup cannot be used It is applicable only when both CanControllerActivation and CanWakeupSupport are set to TRUE. In case these conditions are not met, a configuration error will be reported when the user tries to configure this parameter. The CanWakeupFunctionalityAPI configuration parameter is made non-editable as the CAN driver does not support wakeup over CAN bus. The configuration parameter even though not used shall be present in the schema to maintain the AUTOSAR schema.

(table continues...)

Table 18 (continued) Specification for CanWakeupFunctionalityAPI

Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	CanWakeupSupport, CanControllerActivation		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.14 CanWakeupProcessing

Table 19 Specification for CanWakeupProcessing

Name	`CanWakeupProcessing`		
Description	Specifies the way wake up event on the controller is notified. Enables/disables Can_17_McmCan_MainFunction_Wakeup() API for handling the wakeup events in the polling mode. It is applicable only when CanControllerActivation is set to TRUE. Wake up processing follows the Rx processing parameter configuration. The default value is set to INTERRUPT to set all the CAN driver configuration parameter default values to be interrupt compatible.		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	INTERRUPT: event is notified by the interrupt mechanism POLLING: event is notified when polled		
Default value	INTERRUPT		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	CanRxProcessing, CanControllerActivation		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.15 CanWakeupSourceRef

Table 20 Specification for CanWakeupSourceRef

Name	`CanWakeupSourceRef`		
Description	Contains a reference to the wakeup source for this controller as defined in the ECU State Manager. Implementation type: reference to EcuM_WakeupSourceType It is applicable only when both CanControllerActivation and CanWakeupSupport are set to TRUE. CanWakeupSourceRef configuration parameter is made as non-editable as MCMCAN driver does not support wakeup over CAN bus. The configuration parameter, even though not used, shall be present in the schema to maintain the AUTOSAR schema.		
Multiplicity	0..1	**Type**	EcucSymbolicNameReferenceDef
Range	Reference to Node: EcuMWakeupSource		
Default value	NULL		
Post-build variant value	FALSE	**Post-build variant multiplicity**	FALSE
Value configuration class	Pre-Compile	**Multiplicity configuration class**	Pre-Compile
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	EcuMWakeupSource, CanWakeupSupport, CanControllerActivation		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.16 CanWakeupSupport

Table 21 Specification for CanWakeupSupport

Name	`CanWakeupSupport`		
Description	Enable/disable the CAN driver support for wakeup over CAN bus. It is applicable only when CanControllerActivation is set to TRUE. By default, the optional interface APIs are disabled to minimize the executable code size. CanWakeupSupport configuration parameter is made non-editable as the MCMCAN driver does not support wakeup over CAN bus. The configuration parameter, even though not used, shall be present in the schema to maintain the AUTOSAR schema.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		

(table continues...)

Table 21 (continued) Specification for CanWakeupSupport

Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.3 Container: CanControllerBaudrateConfig

The container contains bit timing related configuration parameters of the CAN controller(s).

The multiplicity of the container is from 1 to 65535. The range is limited as the MCMCAN hardware supports only baud rates from 40 to 1000 kbps.
Post-Build Variant Multiplicity: FALSE
Multiplicity Configuration Class: Pre-Compile

1.3.1.3.1 CanControllerBaudRate

Table 22 Specification for CanControllerBaudRate

Name	`CanControllerBaudRate`		
Description	Specifies the baudrate of the controller (in Kbps). It is dependent on values of CanControllerActivation, CanPeripheralBusClockRef, CanControllerPropSeg, CanControllerSeg1, CanControllerSeg2 and CanControllerSyncJumpWidth. In case the user does not configure the mentioned parameters within their prescribed ranges, a generation error would be reported. The range is limited from 40 to 1000 kbps as the MCMCAN driver hardware supports only this range of baud rate accurately. The default value is set to 500 kbps, as it is the most commonly used baud rate.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	40 - 1000		
Default value	500		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	CanControllerSeg2, CanControllerSeg1, CanControllerPropSeg, CanControllerSyncJumpWidth, CanPeripheralBusClockRef, CanControllerActivation		

(table continues...)

Table 22 (continued) Specification for CanControllerBaudRate

Autosar Version	Applicable for Autosar version 4.2.2.

1.3.1.3.2 CanControllerBaudRate

Table 23 Specification for CanControllerBaudRate

Name	`CanControllerBaudRate`		
Description	Specifies the baudrate of the controller (in Kbps). It is dependent on values of CanControllerActivation, CanPeripheralBusClockRef, CanControllerPropSeg, CanControllerSeg1, CanControllerSeg2 and CanControllerSyncJumpWidth. In case the user does not configure the mentioned parameters within their prescribed ranges, a generation error would be reported. The range is limited from 40 to 1000 kbps as the MCMCAN driver hardware supports only this range of baud rate accurately. The default value is set to 500 kbps, as it is the most commonly used baud rate.		
Multiplicity	1..1	**Type**	EcucFloatParamDef
Range	40.0 - 1000.0		
Default value	500.0		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	CanControllerActivation, CanPeripheralBusClockRef, CanControllerSyncJumpWidth, CanControllerSeg2, CanControllerSeg1, CanControllerPropSeg		
Autosar Version	Applicable for Autosar version 4.4.0.		

1.3.1.3.3 CanControllerBaudRate

Table 24 Specification for CanControllerBaudRate

Name	`CanControllerBaudRate`		
Description	Specifies the baudrate of the controller (in Kbps). It is dependent on values of CanControllerActivation, CanPeripheralBusClockRef, CanControllerPropSeg, CanControllerSeg1, CanControllerSeg2 and CanControllerSyncJumpWidth. In case the user does not configure the mentioned parameters within their prescribed ranges, a generation error would be reported. The range is limited from 40 to 1000 kbps as the MCMCAN driver hardware supports only this range of baud rate accurately. The default value is set to 500 kbps, as it is the most commonly used baud rate.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef

(table continues...)

Table 24 (continued) Specification for CanControllerBaudRate

Range	40 - 1000		
Default value	500		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	CanControllerSeg2, CanControllerSeg1, CanControllerPropSeg, CanControllerSyncJumpWidth, CanPeripheralBusClockRef, CanControllerActivation		
Autosar Version	Applicable for Autosar version 4.2.2.		

1.3.1.3.4 CanControllerBaudRate

Table 25 Specification for CanControllerBaudRate

Name	`CanControllerBaudRate`		
Description	Specifies the baudrate of the controller (in Kbps). It is dependent on values of CanControllerActivation, CanPeripheralBusClockRef, CanControllerPropSeg, CanControllerSeg1, CanControllerSeg2 and CanControllerSyncJumpWidth. In case the user does not configure the mentioned parameters within their prescribed ranges, a generation error would be reported. The range is limited from 40 to 1000 kbps as the MCMCAN driver hardware supports only this range of baud rate accurately. The default value is set to 500 kbps, as it is the most commonly used baud rate.		
Multiplicity	1..1	**Type**	EcucFloatParamDef
Range	40.0 - 1000.0		
Default value	500.0		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	CanControllerActivation, CanPeripheralBusClockRef, CanControllerSyncJumpWidth, CanControllerSeg2, CanControllerSeg1, CanControllerPropSeg		
Autosar Version	Applicable for Autosar version 4.4.0.		

1.3.1.3.5 CanControllerBaudRateConfigID

Table 26 Specification for CanControllerBaudRateConfigID

Name	`CanControllerBaudRateConfigID`		
Description	Uniquely identifies a specific baud rate configuration. This ID is used by the SetBaudrate API. It is applicable only when both CanControllerActivation and CanSetBaudrateApi are set to TRUE. In case the mentioned conditions are not met, a configuration error will be reported when the user tries to configure this parameter. The default value is set to 0, as it is the start ID for the first configuration.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 65535		
Default value	0		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	ECU
Dependency	CanSetBaudrateApi, CanControllerActivation		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.3.6 CanControllerPropSeg

Table 27 Specification for CanControllerPropSeg

Name	`CanControllerPropSeg`		
Description	Specifies the propagation delay in time quanta. Configuration rule: - The sum of CanControllerPropSeg and CanControllerSeg1 should be within 2 (included) and 256 (included). - The sum of 1, CanControllerPropSeg, CanControllerSeg1 and CanControllerSeg2 should be within 4 (included) and 385 (included). The range is limited from 1 to 255 as the MCMCAN driver hardware supports this range of propagation segment value. The default value is set to 47 as the value is set to obtain the most commonly used baud rate of 500 kbps.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	1 - 255		
Default value	47		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-

(table continues...)

Table 27 (continued) Specification for CanControllerPropSeg

Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	CanControllerActivation, CanControllerSyncJumpWidth, CanControllerSeg2, CanControllerSeg1, CanControllerBaudRate, CanPeripheralBusClockRef		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.3.7 CanControllerSeg1

Table 28 Specification for CanControllerSeg1

Name	`CanControllerSeg1`		
Description	Specifies phase segment 1 in time quanta. Configuration rule: - The sum of CanControllerPropSeg and CanControllerSeg1 should be within 2 (included) and 256 (included). - The sum of 1, CanControllerPropSeg, CanControllerSeg1 and CanControllerSeg2 should be within 4 (included) and 385 (included). The default value is set to 16 as the value is set to obtain the most commonly used baud rate of 500 kbps.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	1 - 255		
Default value	16		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	CanControllerActivation, CanControllerSyncJumpWidth, CanControllerSeg2, CanControllerPropSeg, CanControllerBaudRate, CanPeripheralBusClockRef		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.3.8 CanControllerSeg2

Table 29 Specification for CanControllerSeg2

Name	`CanControllerSeg2`

(table continues...)

Table 29 **(continued) Specification for CanControllerSeg2**

Description	Specifies phase segment 2 in time quanta. Configuration rule: The sum of 1, CanControllerPropSeg, CanControllerSeg1 and CanControllerSeg2 should be within 4 (included) and 385 (included). The default value is set to 16 as the value is set to obtain the most commonly used baud rate of 500 kbps.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	2 - 128		
Default value	16		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	CanControllerActivation, CanControllerSyncJumpWidth, CanControllerSeg1, CanControllerBaudRate, CanControllerPropSeg, CanPeripheralBusClockRef		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.3.9 CanControllerSyncJumpWidth

Table 30 **Specification for CanControllerSyncJumpWidth**

Name	`CanControllerSyncJumpWidth`		
Description	The parameter specifies the synchronization jump width for the controller in time quanta. The default value is set to 4 as the value is set to obtain the most commonly used baud rate of 500 kbps. It is dependent on the parameters CanControllerSeg2, CanControllerSeg1, CanControllerPropSeg, CanControllerBaudRate,CanPeripheralBusClockRef, CanControllerActivation. In case the user does not configure the mentioned parameters within their prescribed ranges, a generation error would be reported.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	1 - 128		
Default value	4		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL

(table continues...)

Table 30 (continued) Specification for CanControllerSyncJumpWidth

Dependency	CanControllerActivation, CanPeripheralBusClockRef, CanControllerBaudRate, CanControllerPropSeg, CanControllerSeg1, CanControllerSeg2
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.1.4 Container: CanControllerFdBaudrateConfig

The container is optional and contains bit timing related configuration parameters of the CAN controller(s) for payload and CRC of a CAN FD frame. If this container exists the controller supports CAN FD frames. The lower multiplicity is 0 and upper multiplicity is 1 for this container.

Post-Build Variant Multiplicity: FALSE

Multiplicity Configuration Class: Pre-Compile

1.3.1.4.1 CanControllerFdBaudRate

Table 31 Specification for CanControllerFdBaudRate

Name	`CanControllerFdBaudRate`		
Description	Specifies the data segment baud rate of the controller (in kbps). It is dependent on the values of CanControllerActivation, CanPeripheralBusClockRef, CanControllerPropSeg, CanControllerSeg1, CanControllerSeg2 and CanControllerSyncJumpWidth. In case the user does not configure the mentioned parameters within their prescribed ranges, a generation error would be reported. The range is limited from 40 to 5000 kbps as the MCMCAN driver hardware supports only this range of baud rate accurately. The default value is set to 2500 kbps, as it is the most commonly used baud rate.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	40 - 5000		
Default value	2500		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	CanControllerSyncJumpWidth, CanControllerSeg2, CanControllerSeg1, CanControllerPropSeg, CanPeripheralBusClockRef, CanControllerActivation		
Autosar Version	Applicable for Autosar version 4.2.2.		

1.3.1.4.2 CanControllerFdBaudRate

Table 32 Specification for CanControllerFdBaudRate

Name	`CanControllerFdBaudRate`

(table continues...)

Table 32 **(continued) Specification for CanControllerFdBaudRate**

Description	Specifies the data segment baud rate of the controller (in kbps). It is dependent on the values of CanControllerActivation, CanPeripheralBusClockRef, CanControllerPropSeg, CanControllerSeg1, CanControllerSeg2 and CanControllerSyncJumpWidth. The range is limited from 40 to 5000 kbps as the MCMCAN driver hardware supports only this range of baud rate accurately. The default value is set to 2500 kbps, as it is the most commonly used baud rate.		
Multiplicity	1..1	**Type**	EcucFloatParamDef
Range	40.0 - 5000.0		
Default value	2500.0		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	CanControllerActivation, CanPeripheralBusClockRef, CanControllerSeg2, CanControllerSeg1, CanControllerSyncJumpWidth, CanControllerPropSeg		
Autosar Version	Applicable for Autosar version 4.4.0.		

1.3.1.4.3 CanControllerFdBaudRate

Table 33 **Specification for CanControllerFdBaudRate**

Name	`CanControllerFdBaudRate`		
Description	Specifies the data segment baud rate of the controller (in kbps). It is dependent on the values of CanControllerActivation, CanPeripheralBusClockRef, CanControllerPropSeg, CanControllerSeg1, CanControllerSeg2 and CanControllerSyncJumpWidth. In case the user does not configure the mentioned parameters within their prescribed ranges, a generation error would be reported. The range is limited from 40 to 5000 kbps as the MCMCAN driver hardware supports only this range of baud rate accurately. The default value is set to 2500 kbps, as it is the most commonly used baud rate.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	40 - 5000		
Default value	2500		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-

(table continues...)

Table 33 (continued) Specification for CanControllerFdBaudRate

Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	CanControllerSyncJumpWidth, CanControllerSeg2, CanControllerSeg1, CanControllerPropSeg, CanPeripheralBusClockRef, CanControllerActivation		
Autosar Version	Applicable for Autosar version 4.2.2.		

1.3.1.4.4 CanControllerFdBaudRate

Table 34 Specification for CanControllerFdBaudRate

Name	`CanControllerFdBaudRate`		
Description	Specifies the data segment baud rate of the controller (in kbps). It is dependent on the values of CanControllerActivation, CanPeripheralBusClockRef, CanControllerPropSeg, CanControllerSeg1, CanControllerSeg2 and CanControllerSyncJumpWidth. The range is limited from 40 to 5000 kbps as the MCMCAN driver hardware supports only this range of baud rate accurately. The default value is set to 2500 kbps, as it is the most commonly used baud rate.		
Multiplicity	1..1	**Type**	EcucFloatParamDef
Range	40.0 - 5000.0		
Default value	2500.0		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	CanControllerActivation, CanPeripheralBusClockRef, CanControllerSeg2, CanControllerSeg1, CanControllerSyncJumpWidth, CanControllerPropSeg		
Autosar Version	Applicable for Autosar version 4.4.0.		

1.3.1.4.5 CanControllerPropSeg

Table 35 Specification for CanControllerPropSeg

Name	`CanControllerPropSeg`

(table continues...)

Table 35 (continued) Specification for CanControllerPropSeg

Description	Specifies the propagation delay in time quanta. Configuration rule: - The sum of CanControllerPropSeg and CanControllerSeg1 should be within 1 (included) and 32 (included). - The sum of 1, CanControllerPropSeg, CanControllerSeg1 and CanControllerSeg2 should be within 4 (included) and 49 (included). The default value is set to 1 as the value is set to obtain the most commonly used baud rate of 2500 kbps.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 31		
Default value	1		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	CanControllerActivation, CanControllerSyncJumpWidth, CanControllerSeg2, CanControllerSeg1, CanControllerFdBaudRate, CanPeripheralBusClockRef		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.6 CanControllerSeg1

Table 36 Specification for CanControllerSeg1

Name	`CanControllerSeg1`		
Description	Specifies phase segment 1 in time quanta. Configuration rule: - The sum of CanControllerPropSeg and CanControllerSeg1 should be within 1 (included) and 32 (included). - The sum of 1, CanControllerPropSeg, CanControllerSeg1 and CanControllerSeg2 should be within 4 (included) and 49 (included). The default value is set to 2 as the value is set to obtain the most commonly used baud rate of 2500 kbps.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	1 - 32		
Default value	2		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-

(table continues...)

Table 36 **(continued) Specification for CanControllerSeg1**

Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	CanControllerActivation, CanControllerSyncJumpWidth, CanControllerSeg2, CanControllerPropSeg, CanControllerFdBaudRate, CanPeripheralBusClockRef		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.7 CanControllerSeg2

Table 37 **Specification for CanControllerSeg2**

Name	`CanControllerSeg2`		
Description	Specifies phase segment 2 in time quanta. Configuration rule: - The sum of 1, CanControllerPropSeg, CanControllerSeg1 and CanControllerSeg2 should be within 4 (included) and 49 (included). The default value is set to 1 as the value is set to obtain the most commonly used baud rate of 2500 kbps.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	1 - 16		
Default value	1		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	CanControllerActivation, CanControllerSyncJumpWidth, CanControllerSeg1, CanControllerPropSeg, CanControllerFdBaudRate, CanPeripheralBusClockRef		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.8 CanControllerSspOffset

Table 38 **Specification for CanControllerSspOffset**

Name	`CanControllerSspOffset`

(table continues...)

Table 38 **(continued) Specification for CanControllerSspOffset**

Description	The parameter specifies the Transmitter Delay Compensation Offset in as minimum time quanta (MTQ). Transmitter Delay Compensation Offset is used to adjust the position of the Secondary Sample Point (SSP), relative to the beginning of the received bit. If this parameter is configured, the Transmitter Delay Compensation is done by measurement of the CAN controller. If not specified, Transmitter Delay Compensation is disabled. By default the optional interface APIs are disabled to minimize the executable code size. Example: CAN Module clock frequency(McuMCanFrequency) = 40MHz - MTQ(Minimum Time Quanta) = 1/40 * 10^(-6) s = 0.025 us = 25ns CAN FD Baud Rate(CanControllerFdBaudRate) = 2MBit/s - FD BitTime = 1/(2 * 10^6) s/Bit = 0.5 * 10^(-6) = 500ns/Bit SSP offset in nano second = (SSP %) * FD BitTime = 0.80 * 500ns = 400 ns CanControllerSspOffset in MTQ = 400/25 = 16		
Multiplicity	0..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	0		
Post-build variant value	TRUE	**Post-build variant multiplicity**	TRUE
Value configuration class	Post-Build	**Multiplicity configuration class**	Post-Build
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	CanControllerFdBaudRate, CanControllerActivation, CanPeripheralBusClockRef		
Autosar Version	Applicable for Autosar version 4.4.0.		

1.3.1.4.9 CanControllerSyncJumpWidth

Table 39 **Specification for CanControllerSyncJumpWidth**

Name	`CanControllerSyncJumpWidth`		
Description	The parameter specifies the synchronization jump width for the controller in time quanta. The default value is set to 1 as the value is set to obtain the most commonly used baud rate of 2500 kbps. It is dependent on the parameters CanControllerSeg2, CanControllerSeg1, CanControllerPropSeg, CanControllerBaudRate, CanPeripheralBusClockRef, CanControllerActivation. In case the user does not configure the mentioned parameters within their prescribed ranges, a generation error would be reported.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	1 - 16		
Default value	1		

(table continues...)

Table 39 (continued) Specification for CanControllerSyncJumpWidth

Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	CanControllerActivation, CanControllerSeg1, CanControllerSeg2, CanControllerPropSeg, CanControllerFdBaudRate, CanPeripheralBusClockRef		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.10 CanControllerTrcvDelayCompensationOffset

Table 40 Specification for CanControllerTrcvDelayCompensationOffset

Name	`CanControllerTrcvDelayCompensationOffset`		
Description	The parameter specifies the transceiver delay compensation offset (in ns). This value needs to be provided in nano seconds and not in MTQ(Minimum Time Quantas). By default the optional interface APIs are disabled to minimize the executable code size. Example: CAN Module clock frequency(McuMCanFrequency) = 40MHz - MTQ(Minimum Time Quanta) = 1/40 * 10^(-6) s = 0.025 us = 25ns CAN FD Baud Rate(CanControllerFdBaudRate) = 2MBit/s - FD BitTime = 1/(2 * 10^6) s/Bit = 0.5 * 10^(-6) = 500ns/Bit CanControllerTrcvDelayCompensationOffset = (SSP %) * FD BitTime = 0.80 * 500ns = 400 ns The range of this parameter is deviated from the AUTOSAR value of 0-400 to 0-65535 (in ns) to accommodate larger values transceivers delay compensations required by different CAN FD baud rates.		
Multiplicity	0..1	**Type**	EcucIntegerParamDef
Range	0 - 65535		
Default value	0		
Post-build variant value	TRUE	**Post-build variant multiplicity**	TRUE
Value configuration class	Post-Build	**Multiplicity configuration class**	Post-Build
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	CanControllerFdBaudRate, CanControllerActivation, CanPeripheralBusClockRef		
Autosar Version	Applicable for Autosar version 4.2.2.		

1.3.1.4.11 CanControllerTxBitRateSwitch

Table 41 Specification for CanControllerTxBitRateSwitch

Name	`CanControllerTxBitRateSwitch`		
Description	Specifies if the bit rate switching shall be used for transmissions If FALSE, the CAN FD frames shall be sent without bit rate switching. The default value of the CanControllerTxBitRateSwitch configuration parameter is set to TRUE. CAN FD being used without bitrate switch enabled is a special case.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	TRUE		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	CanControllerActivation		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.5 Container: CanHwFilter

This container is only valid for HRHs and contains the configuration (parameters) of one hardware filter.

Multiplicity of container varies based on the type of object. In case the receive object type is dedicated, user can configure single filter which means message-id matching the value in CanHwFilterCode will only be received by hardware. CanHwFilterMask cannot be configured. In case of receive FIFO for which the CanHwObjectCount is greater than 1 multiple filter ranges can be defined and CanHwFilterMask will be enabled to define the mask to accept range of message-ids. In case only standard ids are configured, multiplicity is 128 elements, extended id, multiplicity is 64 and mixed is 64 elements. Note that if the CanIdType is configured as MIXED, a filter slot in standard ID and Extended ID will be utilized to support both 11-bit and 29-bit message-ids.

Post-Build Variant Multiplicity: FALSE

Multiplicity Configuration Class: Pre-Compile

1.3.1.5.1 CanHwFilterCode

Table 42 Specification for CanHwFilterCode

Name	`CanHwFilterCode`		
Description	Specifies (together with the filter mask) the identifiers range that passes the hardware filter. The referenced hardware object with CanObjectType as RECEIVE type. The default value is set to 2047 as this will match all the standard ID type 11-bit identifiers.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef

(table continues...)

Table 42 (continued) Specification for CanHwFilterCode

Range	0 - 4294967295		
Default value	2047		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	CanIdType, CanObjectType		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.5.2 CanHwFilterMask

Table 43 Specification for CanHwFilterMask

Name	`CanHwFilterMask`		
Description	Describes a mask for the hardware-based filtering of CAN identifiers. The CAN identifiers of the incoming messages are masked with the appropriate CanFilterMask bits holding a 0, which means do not care, that is, do not compare the message identifier in the respective bit position. A 29-bit or 11-bit mask should be created for CanIdType EXTENDED or STANDARD respectively. In case of CanIdType MIXED, a 29-bit mask is built for EXTENDED filter mask, and the 11 MSBs of this value should be taken as the STANDARD 11-bit mask. This is obtained by shifting the 11 MSBs right by 18 bits, and filling with leading 0. The default value is set to 2047 as this will mask all the standard ID type 11-bit identifiers. This is because for STANDARD ID it acts as an open filter, but for extended IDs it is not an open filter *Note: The CanHwFilterMask value shall be applicable only in the case of an Rx FIFO being used (i.e. CanHwObjectCount value greater than 1), in the case of dedicated Rx filter (i.e. CanHwObjectCount value equal to 1) range filtering is not applicable and will be set to non-editable.*		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 4294967295		
Default value	2047		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	CanHwObjectCount, CanObjectType, CanIdType		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.6 Container: CanIcom

This container contains the parameters for configuring pretended networking. The lower multiplicity of the container is 0 and upper multiplicity is 1.

Post-Build Variant Multiplicity: FALSE

Multiplicity Configuration Class: Pre-Compile

1.3.1.7 Container: CanIcomConfig

This container contains the configuration parameters of the ICOM configuration.

It is enabled only when CanPublicIcomSupport is enabled

The upper multiplicity of the CanIcomconfig configuration parameter is limited to 255 as this is the maximum ICOM configurations supported by the CAN driver.

Post-Build Variant Multiplicity: FALSE

Multiplicity Configuration Class: Pre-Compile

1.3.1.7.1 CanIcomConfigId

Table 44 Specification for CanIcomConfigId

Name	`CanIcomConfigId`		
Description	Identifies the ID of the ICOM configuration. The default value is set to 1 as it is the start value of the config ID value.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	1 - 255		
Default value	1		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	ECU
Dependency	CanPublicIcomSupport		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.7.2 CanIcomWakeOnBusOff

Table 45 Specification for CanIcomWakeOnBusOff

Name	`CanIcomWakeOnBusOff`
Description	Defines that the MCU should wake if the bus-off is detected or not. The default value is set to TRUE as bus-off error detection is commonly enabled in the communication systems.

(table continues...)

Table 45 (continued) Specification for CanIcomWakeOnBusOff

Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	TRUE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	ECU
Dependency	CanPublicIcomSupport		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.8 Container: CanICOMGeneral

This container contains the general configuration parameters of the ICOM configuration. the both the lower multiplicity of this container is 0 and upper multiplicity is 1.

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

1.3.1.8.1 CanIcomLevel

Table 46 Specification for CanIcomLevel

Name	`CanIcomLevel`		
Description	Defines the level of the pretended networking. The default value is set to CAN_ICOM_LEVEL_ONE as the CAN driver supports only this level of pretended networking. The CanIcomLevel configuration parameter is made non-editable as the CAN driver only supports one ICOM-level type.		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	CAN_ICOM_LEVEL_ONE: The first level of pretended networking is supported CAN_ICOM_LEVEL_TWO: The second level of pretended networking is supported		
Default value	CAN_ICOM_LEVEL_ONE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL

(table continues...)

Table 46 (continued) Specification for CanIcomLevel

Dependency	CanPublicIcomSupport
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.1.8.2 CanIcomVariant

Table 47 Specification for CanIcomVariant

Name	`CanIcomVariant`		
Description	Defines the variant, which is supported by this CanController. The default value is set to CAN_ICOM_VARIANT_SW as the CAN driver supports only software variant of ICOM. The CanIcomVariant configuration parameter is made non-editable as the CAN driver does not support variants of ICOM other than the default type mentioned.		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	CAN_ICOM_VARIANT_HW: Pretended networking is supported only by hardware CAN_ICOM_VARIANT_NONE: Pretended networking is not supported CAN_ICOM_VARIANT_SW: Pretended networking is supported only by software		
Default value	CAN_ICOM_VARIANT_SW		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	CanPublicIcomSupport		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.9 Container: CanIcomRxMessage

This container contains the configuration parameters for the wakeup causes for matching the 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 CanIcomMessageIdMask should not overlap.

Post-Build Variant Multiplicity: FALSE

Multiplicity Configuration Class: Pre-Compile

1.3.1.9.1 CanIcomCounterValue

Table 48 Specification for CanIcomCounterValue

Name	`CanIcomCounterValue`

(table continues...)

Table 48 (continued) Specification for CanIcomCounterValue

Description	Defines that the MCU should wake when the message with the ID is received 'n' times on the communication channel. The default value is set to 1 as this is the minimum value for this parameter.		
Multiplicity	0..1	**Type**	EcucIntegerParamDef
Range	1 - 65536		
Default value	1		
Post-build variant value	FALSE	**Post-build variant multiplicity**	FALSE
Value configuration class	Pre-Compile	**Multiplicity configuration class**	Pre-Compile
Origin	AUTOSAR_ECUC	**Scope**	ECU
Dependency	CanPublicIcomSupport		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.9.2 CanIcomMessageId

Table 49 Specification for CanIcomMessageId

Name	`CanIcomMessageId`		
Description	Defines the message ID for which 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. The default value is set to 1 as this is the minimum value for the configuration parameter.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 536870912		
Default value	0		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	ECU
Dependency	CanPublicIcomSupport		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.9.3 CanIcomMessageIdMask

Table 50 Specification for CanIcomMessageIdMask

Name	`CanIcomMessageIdMask`

(table continues...)

Table 50 (continued) Specification for CanIcomMessageIdMask

Description	Describes a mask for filtering the CAN identifiers. The CAN identifiers of incoming messages are masked with CanIcomMessageIdMask. If the masked identifier matches the masked value of CanIcomMessageId, it can create a wakeup condition for CanIcomRxMessage. Bits holding a 0 signifies do not care, that is, do not compare the message identifier in the respective bit position. The default value is set to 1 as this is the minimum value for the configuration parameter.		
Multiplicity	0..1	**Type**	EcucIntegerParamDef
Range	0 - 536870912		
Default value	0		
Post-build variant value	FALSE	**Post-build variant multiplicity**	FALSE
Value configuration class	Pre-Compile	**Multiplicity configuration class**	Pre-Compile
Origin	AUTOSAR_ECUC	**Scope**	ECU
Dependency	CanPublicIcomSupport		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.9.4 CanIcomMissingMessageTimerValue

Table 51 Specification for CanIcomMissingMessageTimerValue

Name	`CanIcomMissingMessageTimerValue`		
Description	Defines that the MCU should wake when the message with the configured ICOM Message ID is not received for a specific time in seconds on the communication channel. This parameter would be disabled for editing as MCMCAN does not support wakeup over CAN bus. The configuration parameter value range is limited from 1 ms to 65.535 s as per the timer ability of the micro-controller. The default value is set to 1 s to comply with common timer settings.		
Multiplicity	0..1	**Type**	EcucFloatParamDef
Range	0.000001 - 65.535		
Default value	1.0		
Post-build variant value	FALSE	**Post-build variant multiplicity**	FALSE
Value configuration class	Pre-Compile	**Multiplicity configuration class**	Pre-Compile
Origin	AUTOSAR_ECUC	**Scope**	ECU
Dependency	CanPublicIcomSupport		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.9.5 CanIcomPayloadLength

Table 52 Specification for CanIcomPayloadLength

Name	`CanIcomPayloadLength`		
Description	Defines the payload length that should be compared with the payload length of the received message. The MCU shall wake when the message with the selected ID is having a payload length mismatch.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	1 - 8		
Default value	1		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	CanPublicIcomSupport		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.9.6 CanIcomPayloadLengthError

Table 53 Specification for CanIcomPayloadLengthError

Name	`CanIcomPayloadLengthError`		
Description	Defines that the MCU should wake when a payload error occurs. If the received payload length does not match the configured payload length, this would act as a wake up condition. The default value is set to FALSE as the ICOM payload length error is a special feature of ICOM.		
Multiplicity	1..1	**Type**	EcucBooleanParamD ef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	ECU
Dependency	CanPublicIcomSupport		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.10 Container: CanIcomRxMessageSignalConfig

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

It has to be configured as often as a signal is defined as wakeup cause. When at least one Signal conditions defined in CanIcomRxMessageSignalConfig evaluates to TRUE or when no CanIcomRxMessageSignalConfig is defined, the whole wakeup condition is considered to be TRUE. All instances of this container refer to the same frame/PDU (see CanIcomMessageId). the lower multiplicity of the container is 0 and upper muliplicity is *.

Post-Build Variant Multiplicity: FALSE

Multiplicity Configuration Class: Pre-Compile

1.3.1.10.1 CanIcomSignalMask

Table 54 Specification for CanIcomSignalMask

Name	`CanIcomSignalMask`		
Description	This parameter should 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. The configuration parameter is non-editable as the mask value is taken from the CanIcomSignalMaskUpper32bits and CanIcomSignalMaskLower32bits container, the following split of the CanIcomSignalMask configuration parameter is due to the limitation of configuration tool to support the full range of this parameter.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 9223372036854775807		
Default value	0		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	CanIcomMessageIdMask, CanPublicIcomSupport		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.10.2 CanIcomSignalMaskLower32bits

Table 55 Specification for CanIcomSignalMaskLower32bits

Name	`CanIcomSignalMaskLower32bits`		
Description	Defines the lower 32 bit value of the parameter, which is used to mask a signal in the payload of a CAN message. The default value is set to the maximum range to accept all the messages.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 4294967295		
Default value	4294967295		

(table continues...)

Table 55 (continued) Specification for CanIcomSignalMaskLower32bits

Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	CanIcomMessageIdMask, CanPublicIcomSupport		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.10.3 CanIcomSignalMaskUpper32bits

Table 56 Specification for CanIcomSignalMaskUpper32bits

Name	`CanIcomSignalMaskUpper32bits`		
Description	Defines the upper32 bit value of parameter, which is used to mask a signal in the payload of a CAN message. The default value is set to the maximum range to accept all the messages.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 4294967295		
Default value	4294967295		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	CanIcomMessageIdMask, CanPublicIcomSupport		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.10.4 CanIcomSignalOperation

Table 57 Specification for CanIcomSignalOperation

Name	`CanIcomSignalOperation`		
Description	Defines the operation, which should be used to verify whether the signal value creates a wakeup condition or not. The default value is set to the most commonly used one-on-one message mapping of the EQUAL operation.		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef

(table continues...)

Table 57 (continued) Specification for CanIcomSignalOperation

Range	AND: The received signal value masked by CanIcomSignalMask has at least one bit set in common with CanIcomSignalValue (binary AND). EQUAL: the received signal value masked by CanIcomSignalMask is equal to CanIcomSignalValue. 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.		
Default value	EQUAL		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	ECU
Dependency	CanIcomMessageIdMask, CanPublicIcomSupport		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.10.5 CanIcomSignalRef

Table 58 Specification for CanIcomSignalRef

Name	`CanIcomSignalRef`		
Description	References to the COM layer signal that ICOM should use as a reference parameter. The CanIcomSignalRef configuration parameter is made non-editable as the McmCan driver does not support matching of the ICOM message with the messages in the upper layer. To comply with the AUTOSAR schema this configuration parameter is added but not used in the generator files.		
Multiplicity	0..1	**Type**	EcucReferenceDef
Range	Reference to Node:		
Default value	NULL		
Post-build variant value	FALSE	**Post-build variant multiplicity**	FALSE
Value configuration class	Pre-Compile	**Multiplicity configuration class**	Pre-Compile
Origin	AUTOSAR_ECUC	**Scope**	ECU
Dependency	CanIcomMessageIdMask		

(table continues...)

Table 58 (continued) Specification for CanIcomSignalRef

Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.1.10.6 CanIcomSignalValue

Table 59 Specification for CanIcomSignalValue

Name	`CanIcomSignalValue`		
Description	This parameter should be used to define a signal value, which shall be compared (CanIcomSignalOperation) with the masked CanIcomSignalMask value of the received signal (CanIcomSignalRef). The configuration parameter is non-editable as the value is taken from CanIcomSignalValueUpper32bits and CanIcomSignalValueLower32bits container, the following split of the CanIcomSignalValue configuration parameter is due to the limitation of configuration tool to support the full range of this parameter.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 9223372036854775807		
Default value	0		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	CanIcomMessageIdMask, CanPublicIcomSupport		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.10.7 CanIcomSignalValueLower32bits

Table 60 Specification for CanIcomSignalValueLower32bits

Name	`CanIcomSignalValueLower32bits`		
Description	Defines the lower 32 bit value of the parameter which is used to compare (CanIcomSignalOperation) with the masked CanIcomSignalMask value of the received signal (CanIcomSignalRef). The default value is set to the maximum range to accept all the messages.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 4294967295		
Default value	4294967295		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-

(table continues...)

Table 60 **(continued) Specification for CanIcomSignalValueLower32bits**

Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	CanIcomMessageIdMask, CanPublicIcomSupport		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.10.8 CanIcomSignalValueUpper32bits

Table 61 **Specification for CanIcomSignalValueUpper32bits**

Name	`CanIcomSignalValueUpper32bits`		
Description	Defines the upper32 bit value of the parameter which is used to compare (CanIcomSignalOperation) with the masked CanIcomSignalMask value of the received signal (CanIcomSignalRef). The default value is set to the maximum range to accept all the messages.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 4294967295		
Default value	4294967295		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	CanIcomMessageIdMask, CanPublicIcomSupport		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.11 Container: CanIcomWakeupCauses

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

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

1.3.1.12 Container: CanTTController

This container contains the configuration parameters of the TTCAN controller(s) (which are needed in addition to the configuration parameters of the CAN controller(s)) to support the TTCAN feature.

The TTCAN is not supported by the CAN driver and, therefore, the set of configurations related to TTCAN are made non-editable and not used in the generator files. The TTCAN-related configurations are kept for following the AUTOSAR schema.

Post-Build Variant Multiplicity: FALSE

Multiplicity Configuration Class: Pre-Compile

1.3.1.12.1 CanTTControllerApplWatchdogLimit

Table 62 Specification for CanTTControllerApplWatchdogLimit

Name	`CanTTControllerApplWatchdogLimit`		
Description	Defines the maximum time period (unit is 256 times NTU) after which the application has to serve the watchdog. The TTCAN is not supported by the CAN driver and, therefore, the set of configurations related to the TTCAN are made non-editable and not used in the generator files. The TTCAN related configurations are kept for following the AUTOSAR schema.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	0		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	ECU
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.12.2 CanTTControllerCycleCountMax

Table 63 Specification for CanTTControllerCycleCountMax

Name	`CanTTControllerCycleCountMax`		
Description	Defines the value for cycle_count_max. Allowed values: 0x00: 1 basic cycle 0x01: 2 basic cycles 0x03: 4 basic cycles 0x07: 8 basic cycles 0x0F: 16 basic cycles 0x1F: 32 basic cycles 0x3F: 64 basic cycles The TTCAN is not supported by the CAN driver module and, therefore, the set of configurations related to TTCAN are made non-editable and not used in the generator files. The TTCAN related configurations are kept for following the AUTOSAR schema.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 63		
Default value	0		

(table continues...)

Table 63 (continued) Specification for CanTTControllerCycleCountMax

Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	ECU
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.12.3 CanTTControllerEcucPartitionRef

Table 64 Specification for CanTTControllerEcucPartitionRef

Name	`CanTTControllerEcucPartitionRef`		
Description	Maps the Time triggered CAN controller to zero or one ECUC partitions. The ECUC partition referenced is a subset of the ECUC partitions where the CAN driver is mapped to.		
Multiplicity	0..1	**Type**	EcucSymbolicNameReferenceDef
Range	Reference to Node:		
Default value	NULL		
Post-build variant value	FALSE	**Post-build variant multiplicity**	FALSE
Value configuration class	Post-Build	**Multiplicity configuration class**	Post-Build
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar version 4.4.0.		

1.3.1.12.4 CanTTControllerExpectedTxTrigger

Table 65 Specification for CanTTControllerExpectedTxTrigger

Name	`CanTTControllerExpectedTxTrigger`		
Description	Defines the number of expected_tx_trigger. The TTCAN is not supported by the CAN driver and, therefore, the set of configurations related to TTCAN are made non editable and not used in the generator files. The TTCAN related configurations are kept for following the AUTOSAR schema.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	0		

(table continues...)

Table 65 (continued) Specification for CanTTControllerExpectedTxTrigger

Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	ECU
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.12.5 CanTTControllerExternalClockSynchronisation

Table 66 Specification for CanTTControllerExternalClockSynchronisation

Name	`CanTTControllerExternalClockSynchronisation`		
Description	Enables/disables the external clock synchronization. TRUE: external clock synchronization enabled. FALSE: external clock synchronization disabled. This parameter should only be configurable when the CanTTControllerLevel2 parameter is set to TRUE. The TTCAN is not supported by the CAN driver and, therefore, the set of configurations related to TTCAN are made non-editable and not used in the generator files. The TTCAN related configurations are kept for following the AUTOSAR schema.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	ECU
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.12.6 CanTTControllerGlobalTimeFiltering

Table 67 Specification for CanTTControllerGlobalTimeFiltering

Name	`CanTTControllerGlobalTimeFiltering`

(table continues...)

Table 67 (continued) Specification for CanTTControllerGlobalTimeFiltering

Description	Enables/disables the global time filtering. TRUE: global time filtering enabled. FALSE: global time filtering disabled. This parameter should only be configurable when the CanTTControllerLevel2 parameter is set to TRUE. The TTCAN is not supported by the CAN driver and, therefore, the set of configurations related to TTCAN are made non-editable and not used in the generator files. The TTCAN related configurations are kept for following the AUTOSAR schema.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	ECU
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.12.7 CanTTControllerInitialRefOffset

Table 68 Specification for CanTTControllerInitialRefOffset

Name	`CanTTControllerInitialRefOffset`		
Description	Defines the initial value for ref trigger offset. The TTCAN is not supported by the CAN driver and, therefore, the set of configurations related to TTCAN are made non-editable and not used in the generator files. The TTCAN related configurations are kept for following the AUTOSAR schema.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 127		
Default value	0		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	ECU
Dependency	-		

(table continues...)

Table 68 (continued) Specification for CanTTControllerInitialRefOffset

Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.1.12.8 CanTTControllerInterruptEnable

Table 69 Specification for CanTTControllerInterruptEnable

Name	`CanTTControllerInterruptEnable`		
Description	Enables/disables the respective interrupts. Bit position set to 1: enable respective interrupt. Bit position set to 0: disable respective interrupt. Bit position / interrupt source: 10: application watchdog. 9: watch trigger reached. 8: initialization watch trigger reached. 7: change of error level. 6: Tx overflow. 5: Tx underflow. 4: global time error. 3: gap. 2: start of cycle. 1: time discontinuity. 0: master state change. Bit position - 1: Time Discontinuity and - 4: Global Time Error should only be configurable when the CanTTControllerLevel2 parameter is set to TRUE. The TTCAN is not supported by the CAN driver and, therefore, the set of configurations related to TTCAN are made non-editable and not used in the generator files. The TTCAN related configurations are kept for following the AUTOSAR schema.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 1023		
Default value	0		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.12.9 CanTTControllerLevel2

Table 70 Specification for CanTTControllerLevel2

Name	`CanTTControllerLevel2`		
Description	Defines whether Level 2 or Level 1 is used. TRUE: Level 2 FALSE: Level 1 If the CanTTControllerLevel2 parameter is set to FALSE, all parameters with dependency to the CanTTControllerLevel2 parameter need not be configured. The TTCAN is not supported by the CAN driver and, therefore, the set of configurations related to TTCAN are made non-editable and not used in the generator files. The TTCAN related configurations are kept for following the AUTOSAR schema.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.12.10 CanTTControllerNTUConfig

Table 71 Specification for CanTTControllerNTUConfig

Name	`CanTTControllerNTUConfig`		
Description	Defines the config value for the NTU (network time unit). The value is expressed in microseconds. The value configured should be greater than 0. Together with the local oscillator period, the TUR (time unit ratio) can be derived from the NTU. This parameter should only be configurable when the CanTTControllerLevel2 parameter is set to TRUE. The TTCAN is not supported by the CAN driver and, therefore, the set of configurations related to TTCAN are made non-editable and not used in the generator files. The TTCAN related configurations are kept for following the AUTOSAR schema.		
Multiplicity	1..1	**Type**	EcucFloatParamDef
Range	0 - 100		
Default value	0		

(table continues...)

Table 71 (continued) Specification for CanTTControllerNTUConfig

Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	ECU
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.12.11 CanTTControllerOperationMode

Table 72 Specification for CanTTControllerOperationMode

Name	`CanTTControllerOperationMode`		
Description	Defines the operation mode. The TTCAN is not supported by the CAN driver and, therefore, the set of configurations related to TTCAN are made non-editable and not used in the generator files. The TTCAN related configurations are kept for following the AUTOSAR schema.		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	CAN_TT_EVENT_SYNC_TIME_TRIGGERED: synchronous time-triggered event mode CAN_TT_EVENT_TRIGGERED: event triggered mode CAN_TT_TIME_TRIGGERED: time triggered mode		
Default value	CAN_TT_TIME_TRIGGERED		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	ECU
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.12.12 CanTTControllerSyncDeviation

Table 73 Specification for CanTTControllerSyncDeviation

Name	`CanTTControllerSyncDeviation`

(table continues...)

Table 73 (continued) Specification for CanTTControllerSyncDeviation

Description	Defines the maximum synchronization deviation. Given as a percentage value of the NTU (network time unit). The value configured should be greater than 0. This parameter should only be configurable when the CanTTControllerLevel2 parameter is set to TRUE. The TTCAN is not supported by the CAN driver and, therefore, the set of configurations related to TTCAN are made non-editable and not used in the generator files. The TTCAN related configurations are kept for following the AUTOSAR schema.		
Multiplicity	1..1	**Type**	EcucFloatParamDef
Range	0 - 100		
Default value	0		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.12.13 CanTTControllerTURRestore

Table 74 Specification for CanTTControllerTURRestore

Name	`CanTTControllerTURRestore`		
Description	Enables/disables the TUR restore. Note that the value configured for the TUR can be derived from the value configured for the NTU and the local oscillator period. TRUE: TUR restore enabled FALSE: TUR restore disabled This parameter should only be configurable when the CanTTControllerLevel2 parameter is set to TRUE. The TTCAN is not supported by the CAN driver and, therefore, the set of configurations related to TTCAN are made non-editable and not used in the generator files. The TTCAN related configurations are kept for following the AUTOSAR schema.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		

(table continues...)

Table 74 (continued) Specification for CanTTControllerTURRestore

Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.12.14 CanTTControllerTimeMaster

Table 75 Specification for CanTTControllerTimeMaster

Name	`CanTTControllerTimeMaster`		
Description	Defines whether the controller acts as a potential time master. TRUE: potential time master. FALSE: time slave. The TTCAN is not supported by the CAN driver and, therefore, the set of configurations related to TTCAN are made non-editable and not used in the generator files. The TTCAN related configurations are kept for following the AUTOSAR schema.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	ECU
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.12.15 CanTTControllerTimeMasterPriority

Table 76 Specification for CanTTControllerTimeMasterPriority

Name	`CanTTControllerTimeMasterPriority`

(table continues...)

Table 76 (continued) Specification for CanTTControllerTimeMasterPriority

Description	Defines the time master priority. The TTCAN is not supported by the CAN driver and, therefore, the set of configurations related to TTCAN are made non-editable and not used in the generator files. The TTCAN related configurations are kept for following the AUTOSAR schema.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 7		
Default value	0		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	ECU
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.12.16 CanTTControllerTxEnableWindowLength

Table 77 Specification for CanTTControllerTxEnableWindowLength

Name	`CanTTControllerTxEnableWindowLength`		
Description	Length of the Tx enable window is expressed in CAN bit times. The CanTTControllerTxEnableWindowlength definition parameter is used such that: Length of enable window = CanTTControllerTxEnableWindowLength + 1 The TTCAN is not supported by the CAN driver and, therefore, the set of configurations related to TTCAN are made non-editable and not used in the generator files. The TTCAN related configurations are kept for following the AUTOSAR schema.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	1 - 16		
Default value	1		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	ECU
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.12.17 CanTTControllerWatchTriggerGapTimeMark

Table 78 Specification for CanTTControllerWatchTriggerGapTimeMark

Name	`CanTTControllerWatchTriggerGapTimeMark`		
Description	Defines the watch trigger time mark after a gap. The TTCAN is not supported by the CAN driver and, therefore, the set of configurations related to TTCAN are made non-editable and not used in the generator files. The TTCAN related configurations are kept for following the AUTOSAR schema.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 65535		
Default value	0		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.12.18 CanTTControllerWatchTriggerTimeMark

Table 79 Specification for CanTTControllerWatchTriggerTimeMark

Name	`CanTTControllerWatchTriggerTimeMark`		
Description	Defines the watch trigger time mark. The TTCAN is not supported by the CAN driver and, therefore, the set of configurations related to TTCAN are made non-editable and not used in the generator files. The TTCAN related configurations are kept for following the AUTOSAR schema.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 65535		
Default value	0		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.12.19 CanTTIRQProcessing

Table 80 Specification for CanTTIRQProcessing

Name	`CanTTIRQProcessing`		
Description	Enables/disables Can_MainFunction_BusOff() API for handling the bus-off events in the polling mode. The TTCAN is not supported by the CAN driver and, therefore, the set of configurations related to TTCAN are made non-editable and not used in the generator files. The TTCAN related configurations are kept for following the AUTOSAR schema.		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	INTERRUPT: event is notified by the interrupt mechanism POLLING: event is notified when polled		
Default value	INTERRUPT		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	ECU
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.13 Container: CanTTHardwareObjectTrigger

This container contains the configuration (parameters) of TTCAN triggers for hardware objects, which are additional to the configuration (parameters) of the CAN hardware objects.

The TTCAN is not supported by the CAN driver and, therefore, the set of configurations related to TTCAN are made non-editable and not used in the generator files. The TTCAN related configurations are kept for following the AUTOSAR schema.

Post-Build Variant Multiplicity: FALSE

Multiplicity Configuration Class: Pre-Compile

1.3.1.13.1 CanTTHardwareObjectBaseCycle

Table 81 Specification for CanTTHardwareObjectBaseCycle

Name	`CanTTHardwareObjectBaseCycle`		
Description	Defines the cycle_offset. CanTTHardwareObjectBaseCycle must be not greater than cycle_count_max. The TTCAN is not supported by the CAN driver and, therefore, the set of configurations related to TTCAN are made non-editable and not used in the generator files. The TTCAN related configurations are kept for following the AUTOSAR schema.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef

(table continues...)

Table 81 (continued) Specification for CanTTHardwareObjectBaseCycle

Range	0 - 63		
Default value	0		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	ECU
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.13.2 CanTTHardwareObjectCycleRepetition

Table 82 Specification for CanTTHardwareObjectCycleRepetition

Name	`CanTTHardwareObjectCycleRepetition`		
Description	Defines the repeat_factor. CanTTHardwareObjectCycleRepetition should be a power of two (2), greater than cycle_offset but not greater than cycle_count_max + 1. The TTCAN is not supported by the CAN driver and, therefore, the set of configurations related to TTCAN are made non-editable and not used in the generator files. The TTCAN related configurations are kept for following the AUTOSAR schema.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	1 - 64		
Default value	1		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	ECU
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.13.3 CanTTHardwareObjectTimeMark

Table 83 Specification for CanTTHardwareObjectTimeMark

Name	`CanTTHardwareObjectTimeMark`

(table continues...)

Table 83 (continued) Specification for CanTTHardwareObjectTimeMark

Description	Defines the point in time, when the trigger will be activated. Value is expressed in cycle time. The TTCAN is not supported by the CAN driver and, therefore, the set of configurations related to TTCAN are made non-editable and not used in the generator files. The TTCAN related configurations are kept for following the AUTOSAR schema.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 65535		
Default value	0		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.13.4 CanTTHardwareObjectTriggerId

Table 84 Specification for CanTTHardwareObjectTriggerId

Name	`CanTTHardwareObjectTriggerId`		
Description	Sequential number which allows separation of different TTCAN triggers configured for one and the same hardware object. The TTCAN is not supported by the CAN driver and, therefore, the set of configurations related to TTCAN are made non-editable and not used in the generator files. The TTCAN related configurations are kept for following the AUTOSAR schema.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 63		
Default value	0		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.13.5 CanTTHardwareObjectTriggerType

Table 85 Specification for CanTTHardwareObjectTriggerType

Name	`CanTTHardwareObjectTriggerType`		
Description	Defines the type of the trigger associated with the hardware object. This parameter depends on plain CAN parameter CAN_OBJECT_TYPE. The TTCAN is not supported by the CAN driver and, therefore, the set of configurations related to TTCAN are made non-editable and not used in the generator files. The TTCAN related configurations are kept for following the AUTOSAR schema.		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	CAN_TT_RX_TRIGGER: TT CAN with receive triggering CAN_TT_TX_REF_TRIGGER: TTCAN with reference triggered transmission CAN_TT_TX_REF_TRIGGER_GAP: TTCAN with reference triggered gap in transmission CAN_TT_TX_TRIGGER_EXCLUSIVE: TTCAN with excusive trigger transmission CAN_TT_TX_TRIGGER_MERGED: TTCAN with merged triggered transmission CAN_TT_TX_TRIGGER_SINGLE: TTCAN with single trigger transmission		
Default value	CAN_TT_RX_TRIGGER		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	CanObjectType		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.14 Container: CommonPublishedInformation

General configuration of CAN driver common container, aggregated by all modules. It contains published information about vendor and versions.

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

1.3.1.14.1 ArMajorVersion

Table 86 Specification for ArMajorVersion

Name	`ArMajorVersion`		
Description	This parameter provides the major version of the AUTOSAR specification. The default value is set to 4 as the CAN driver is following the AUTOSAR version 4.x.x.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		

(table continues...)

Table 86 (continued) Specification for ArMajorVersion

Default value	4		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.14.2 ArMinorVersion

Table 87 Specification for ArMinorVersion

Name	`ArMinorVersion`		
Description	This parameter provides the minor version of the AUTOSAR specification.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	As per AUTOSAR minor version		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.14.3 ArPatchVersion

Table 88 Specification for ArPatchVersion

Name	`ArPatchVersion`		
Description	This parameter provides the patch version of the AUTOSAR specification.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	As per the AUTOSAR patch version		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-

(table continues...)

Table 88 (continued) Specification for ArPatchVersion

Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.14.4 ModuleId

Table 89 Specification for ModuleId

Name	`ModuleId`		
Description	This parameter provides the module Id. The default value is set to 80 as this is the module ID of the CAN driver.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 65535		
Default value	80		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.14.5 Release

Table 90 Specification for Release

Name	`Release`		
Description	This parameter indicates the TC3xx device derivative used for the implementation. The default value is derived from the property file and represents the hardware derivative of the micro controller for which the CAN driver is being configured.		
Multiplicity	1..1	**Type**	EcucStringParamDef
Range	String		
Default value	As per the hardware derivative		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-

(table continues...)

Table 90 **(continued) Specification for Release**

Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.14.6 SwMajorVersion

Table 91 **Specification for SwMajorVersion**

Name	`SwMajorVersion`		
Description	This parameter provides the major version of the software. The default value is set to the software version that will be incremented per release of the code.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	As per the software version		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.14.7 SwMinorVersion

Table 92 **Specification for SwMinorVersion**

Name	`SwMinorVersion`		
Description	This parameter provides the minor version of the software. The default value is set to the software version that will be incremented per update of the code.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	As per the software version		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-

(table continues...)

Table 92 (continued) Specification for SwMinorVersion

Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.14.8 SwPatchVersion

Table 93 Specification for SwPatchVersion

Name	`SwPatchVersion`		
Description	This parameter provides the patch version of the software. The default value is set to the software version that will be incremented per patch set of the code after release.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	As per the software version		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.14.9 VendorApiInfix

Table 94 Specification for VendorApiInfix

Name	`VendorApiInfix`		
Description	This parameter is used to specify the vendor specific name. The default value is set to McmCan as this is the unique name of the CAN driver provided by Infineon.		
Multiplicity	1..1	**Type**	EcucStringParamDef
Range	String		
Default value	McmCan		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-

(table continues...)

Table 94 **(continued) Specification for VendorApiInfix**

Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.14.10 VendorId

Table 95 **Specification for VendorId**

Name	`VendorId`		
Description	This parameter provides the vendor Id The default value is set to 17 as this is the Infineon vendor ID.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 65535		
Default value	17		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.15 Container: Can

This container holds the configuration of a single CAN driver.

Post-Build Variant Multiplicity: TRUE

Multiplicity Configuration Class: Post-Build

1.3.1.16 Container: CanGeneral

This container contains the parameters related each CAN driver unit.

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

1.3.1.16.1 CanDeInitApi

Table 96 **Specification for CanDeInitApi**

Name	`CanDeInitApi`

(table continues...)

Table 96 (continued) Specification for CanDeInitApi

Description	The parameter switches the Can_17_McmCan_DeInit () API to ON or OFF. By default, the optional interface APIs are disabled to minimize the executable code size. In AUTOSAR 4.4.0 the parameter would be made editable FALSE and always generate the macro value as ON. This is because the Can_17_McmCan_DeInit() is not an optional API in AUTOSAR 4.4.0.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	ECU
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.16.2 CanDevErrorDetect

Table 97 Specification for CanDevErrorDetect

Name	`CanDevErrorDetect`		
Description	Switches the DET detection and notification to ON or OFF - TRUE: enabled (ON) - FALSE: disabled (OFF)		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar version 4.4.0.		

1.3.1.16.3 CanDevErrorDetection

Table 98 Specification for CanDevErrorDetection

Name	`CanDevErrorDetection`		
Description	Switches the DET detection and notification to ON or OFF - TRUE: enabled (ON) - FALSE: disabled (OFF)		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar version 4.2.2.		

1.3.1.16.4 CanEcucPartitionRef

Table 99 Specification for CanEcucPartitionRef

Name	`CanEcucPartitionRef`		
Description	The parameter maps the CAN driver to zero or multiple ECUC partitions to make the modules API available in this partition. The CAN driver will operate as an independent instance in each of the partitions. *Note: Parameter support is added only for AUTOSAR schema compliance. This parameter is not used in code generation logic, hence this parameter is made editable false.*		
Multiplicity	0..*	**Type**	EcucReferenceDef
Range	Reference to Node:		
Default value	NULL		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	Pre-Compile
Origin	AUTOSAR_ECUC	**Scope**	ECU
Dependency	-		
Autosar Version	Applicable for Autosar version 4.4.0.		

1.3.1.16.5 CanIndex

Table 100 **Specification for CanIndex**

Name	`CanIndex`		
Description	Specifies the InstanceId of the module instance. If only one instance is present it shall have the Id 0. The default value is set as 0 assuming there is only one instance of the CAN driver.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	0		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.16.6 CanInitDeInitApiMode

Table 101 **Specification for CanInitDeInitApiMode**

Name	`CanInitDeInitApiMode`		
Description	Defines the mode in which the Init and DeInit APIs will be used. The default value of this parameter is set to Supervisor to enable maximum access rights to the registers used by the CAN driver.		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	CAN_17_MCMCAN_MCAL_SUPERVISOR: Operating mode used is Supervisory CAN_17_MCMCAN_MCAL_USER1: Operating mode used is USER-1		
Default value	CAN_17_MCMCAN_MCAL_SUPERVISOR		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.16.7 CanLPduReceiveCalloutFunction

Table 102 Specification for CanLPduReceiveCalloutFunction

Name	`CanLPduReceiveCalloutFunction`		
Description	Specifies the name of a callout function that is called after a successful reception of a received CAN Rx L-PDU. If this parameter is configured with NULL_PTR, no callout will take place. The L-PDU callout function is mapped in a separate memory section. The L-PDU call out configuration parameter is set to non-editable as the CAN driver implemented is not an external CAN controller using any form of communication for interaction with the hardware. The default value is set to NULL_PTR as this configuration parameter is not being used.		
Multiplicity	0..1	**Type**	EcucFunctionNameDef
Range	String		
Default value	NULL		
Post-build variant value	FALSE	**Post-build variant multiplicity**	FALSE
Value configuration class	Pre-Compile	**Multiplicity configuration class**	Pre-Compile
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.16.8 CanMainFunctionBusoffPeriod

Table 103 Specification for CanMainFunctionBusoffPeriod

Name	`CanMainFunctionBusoffPeriod`		
Description	Describes the period for cyclic call to Can_17_McmCan_MainFunction_Busoff. The unit is expressed in seconds. The default value is set to 5 ms. This is done to keep all the communication module main function periodicity to a common value.		
Multiplicity	0..1	**Type**	EcucFloatParamDef
Range	0.001 - 65.535		
Default value	0.005		
Post-build variant value	FALSE	**Post-build variant multiplicity**	FALSE
Value configuration class	Pre-Compile	**Multiplicity configuration class**	Pre-Compile

(table continues...)

Table 103 (continued) Specification for CanMainFunctionBusoffPeriod

Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.16.9 CanMainFunctionModePeriod

Table 104 Specification for CanMainFunctionModePeriod

Name	`CanMainFunctionModePeriod`		
Description	Describes the period for the cyclic call to Can_17_McmCan_MainFunction_Mode. The unit is expressed in seconds. The default value is set to 5 ms. This is done to keep all the communication module main function periodicity to a common value. The parameter is made non-editable as the CAN driver has a synchronous mode setting mechanism and does not support the Can_17_McmCan_MainFunction_Mode() function. The configuration parameter, even though not used, shall be present in the schema to maintain the AUTOSAR schema.		
Multiplicity	1..1	**Type**	EcucFloatParamDef
Range	0.001 - 65.535		
Default value	0.005		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.16.10 CanMainFunctionWakeupPeriod

Table 105 Specification for CanMainFunctionWakeupPeriod

Name	`CanMainFunctionWakeupPeriod`		
Description	Describes the period for the cyclic call to Can_17_McmCan_MainFunction_Wakeup. Unit is expressed in seconds. The default value is set to 5 ms. This is done to keep all the communication module main function periodicity to a common value.		
Multiplicity	0..1	**Type**	EcucFloatParamDef
Range	0.001 - 65.535		
Default value	0.005		

(table continues...)

Table 105 (continued) Specification for CanMainFunctionWakeupPeriod

Post-build variant value	FALSE	**Post-build variant multiplicity**	FALSE
Value configuration class	Pre-Compile	**Multiplicity configuration class**	Pre-Compile
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.16.11 CanMultiCoreErrorDetect

Table 106 Specification for CanMultiCoreErrorDetect

Name	`CanMultiCoreErrorDetect`		
Description	Switches the multi-core error detection and notification to ON or OFF. - TRUE: enabled (ON) - FALSE: disabled (OFF) *Note: If the CanMultiCoreErrorDetect parameter is set to TRUE with the CanDevErrorDetection parameter set to FALSE, an error is generated.*		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	CanDevErrorDetection		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.16.12 CanMultiplexedTransmission

Table 107 Specification for CanMultiplexedTransmission

Name	`CanMultiplexedTransmission`
Description	Enables/disables multiplexed transmission feature support. By default, the optional interface APIs are disabled to minimize the executable code size.

(table continues...)

Table 107 (continued) Specification for CanMultiplexedTransmission

Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	ECU
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.16.13 CanOsCounterRef

Table 108 Specification for CanOsCounterRef

Name	`CanOsCounterRef`		
Description	Contains a reference to the OsCounter, which can be used by the CAN driver. The CanOsCounterRef configuration parameter is made non-editable as the CAN driver should make use of the internal counter values. The configuration parameter, even though not used, should be present in the schema to maintain the AUTOSAR schema.		
Multiplicity	0..1	**Type**	EcucReferenceDef
Range	Reference to Node: OsCounter		
Default value	NULL		
Post-build variant value	FALSE	**Post-build variant multiplicity**	FALSE
Value configuration class	Pre-Compile	**Multiplicity configuration class**	Pre-Compile
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.16.14 CanPublicIcomSupport

Table 109 Specification for CanPublicIcomSupport

Name	`CanPublicIcomSupport`

(table continues...)

Table 109 (continued) Specification for CanPublicIcomSupport

Description	Selects the support of pretended network features in the CAN driver. TRUE: enabled FALSE: disabled The CAN driver uses this parameter for enabling/disabling the pretended network feature support API Can_17_McmCan_SetIcomConfiguration (). By default, the optional interface APIs are disabled to minimize the executable code size.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	ECU
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.16.15 CanRunTimeErrorDetect

Table 110 Specification for CanRunTimeErrorDetect

Name	`CanRunTimeErrorDetect`		
Description	The parameter is used to enable or disable the runtime error checks of the CAN module.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	TRUE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.16.16 CanSetBaudrateApi

Table 111 Specification for CanSetBaudrateApi

Name	`CanSetBaudrateApi`		
Description	Used for enabling/disabling the support of Can_17_McmCan _SetBaudrate () and Can_17_McmCan_CheckBaudrate () APIs. It is applicable only when both CanControllerActivation and CanSetBaudrateApi are set to TRUE. By default, the optional interface APIs are disabled to minimize the executable code size.		
Multiplicity	0..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	FALSE
Value configuration class	Pre-Compile	**Multiplicity configuration class**	Pre-Compile
Origin	AUTOSAR_ECUC	**Scope**	ECU
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.16.17 CanSupportTTCANRef

Table 112 Specification for CanSupportTTCANRef

Name	`CanSupportTTCANRef`		
Description	Refers to the CanIfSupportTTCAN parameter in the CAN interface module configuration. The CanIfSupportTTCAN parameter defines whether TTCAN is supported. The CanSupportTTCANRef configuration parameter is made non-editable as the CAN driver should not support TTCAN. The configuration parameter, even though not used, should be present in the schema to maintain the AUTOSAR schema.		
Multiplicity	0..1	**Type**	EcucReferenceDef
Range	Reference to Node: CanIfPrivateCfg		
Default value	NULL		
Post-build variant value	FALSE	**Post-build variant multiplicity**	FALSE
Value configuration class	Pre-Compile	**Multiplicity configuration class**	Pre-Compile

(table continues...)

Table 112 (continued) Specification for CanSupportTTCANRef

Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.16.18 CanTimeoutDuration

Table 113 Specification for CanTimeoutDuration

Name	`CanTimeoutDuration`		
Description	Specifies the maximum time for the blocking function until a timeout is detected. The unit is expressed in seconds. The default value is set to 1ms for the CanTimeoutDuration configuration parameter considering that no hardware action should take more than 1ms to execute.		
Multiplicity	1..1	**Type**	EcucFloatParamDef
Range	0.000001 - 65.535		
Default value	0.001		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.16.19 CanVersionInfoApi

Table 114 Specification for CanVersionInfoApi

Name	`CanVersionInfoApi`		
Description	Switches the Can_17_McmCan_GetVersionInfo() API to ON or OFF. The default value is set as FALSE to reduce the code foot print as version information is seldom used in the development phase.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-

(table continues...)

Table 114 (continued) Specification for CanVersionInfoApi

Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.17 Container: CanMainFunctionRWPeriods

This container contains the parameter for configuring the period for the cyclic call to Can_17_McmCan_MainFunction_Read or Can_17_McmCan_MainFunction_Write depending on the referring item.

The multiplicity range of the CanMainFunctionRWPeriods configuration parameter has been altered to 254 to keep a controllable upper limit to the number of instances.

Post-Build Variant Multiplicity: FALSE

Multiplicity Configuration Class: Pre-Compile

1.3.1.17.1 CanMainFunctionPeriod

Table 115 Specification for CanMainFunctionPeriod

Name	`CanMainFunctionPeriod`		
Description	Describes the period for the cyclic call to Can_17_McmCan_MainFunction_Read or Can_17_McmCan_MainFunction_Write depending on the referring item. The unit is expressed in seconds. The different poll-cycles will be configurable when more than one CanMainFunctionPeriod is configured. In this case, multiple Can_17_McmCan_MainFunction_Read() or Can_17_McmCan_MainFunction_Write() will be provided by the CAN driver. The default value is set to 5 ms. This is done to keep all the communication module main function periodicity to a common value.		
Multiplicity	1..1	**Type**	EcucFloatParamDef
Range	0.001 - 65.535		
Default value	0.005		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.18 Container: CanHardwareObject

This container contains the configuration (parameters) of the CAN hardware objects. The lower multiplicity of the container is 1 and upper multiplicity id till the maximum number of hardware objects.

Post-Build Variant Multiplicity: FALSE

Multiplicity Configuration Class: Pre-Compile

1.3.1.18.1 CanControllerRef

Table 116 Specification for CanControllerRef

Name	`CanControllerRef`		
Description	Reference to the CAN controller to which the HOH (hardware object handle) is associated to		
Multiplicity	1..1	**Type**	EcucReferenceDef
Range	Reference to Node: CanController		
Default value	NULL		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.18.2 CanFdPaddingValue

Table 117 Specification for CanFdPaddingValue

Name	`CanFdPaddingValue`		
Description	The parameter specifies the value which is used to pad unspecified data in the CAN FD frames greater than 8 bytes for transmission. This is necessary due to the discrete possible values of the DLC (data length count) if greater than 8 bytes. If the length of a PDU which was requested to be sent does not match the allowed DLC values, the remaining bytes up to the next possible value should be padded with this value. It is applicable only when CanObjectType is of transmit type and CAN FD is enabled.		
Multiplicity	0..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	0		
Post-build variant value	TRUE	**Post-build variant multiplicity**	TRUE
Value configuration class	Post-Build	**Multiplicity configuration class**	Post-Build

(table continues...)

Table 117 (continued) Specification for CanFdPaddingValue

Origin	AUTOSAR_ECUC	**Scope**	ECU
Dependency	CanControllerRef, CanControllerFdBaudrateConfig, CanObjectType		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.18.3 CanHandleType

Table 118 Specification for CanHandleType

Name	`CanHandleType`		
Description	Specifies the type (FULL-CAN or BASIC-CAN) of a hardware object. As FULL CAN feature is most commonly used, the default value of the CanHandleType configuration parameter is set to FULL.		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	BASIC: for several L-PDUs handled by the hardware object FULL: for only one L-PDU (identifier) handled by the hardware object		
Default value	FULL		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	ECU
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.18.4 CanHardwareObjectUsesPolling

Table 119 Specification for CanHardwareObjectUsesPolling

Name	`CanHardwareObjectUsesPolling`		
Description	The parameter indicates that polling for a particular hardware object is enabled. This parameter is enabled if CanTxProcessing or CanRxProcessing is set to MIXED for the particular controller to which these hardware objects belong to. In this case, the hardware objects which have this parameter value set as TRUE will have the polling for that object enabled. *Note: In case Rxprocessing or Txprocessing is configured as MIXED and if all hardware objects are configured to use polling or interrupt then a warning will be generated in configuration tool. Hence, in case the user wants to use only polling or only interrupt for the hardware objects associated with a certain controller, the user should select CanRxProcessing or CanTxProcessing as POLLING or INTERRUPT respectively.*		
Multiplicity	0..1	**Type**	EcucBooleanParamDef

(table continues...)

Table 119 (continued) Specification for CanHardwareObjectUsesPolling

Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX FOR AS4.2.2 VARIANT AND AUTOSAR_ECUC FOR AS4.4.0 VARIANT	**Scope**	LOCAL
Dependency	CanObjectType, CanTxProcessing, CanRxProcessing		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.18.5 CanHwFIFOThreshold

Table 120 Specification for CanHwFIFOThreshold

Name	`CanHwFIFOThreshold`		
Description	The parameter specifies the threshold size at which interrupt is triggered to copy the data CanHwFIFOThreshold should be less than or equal to CanFifoSize CanObjectType should be RECIEVE type and CanHwObjectCount should be greater than 1. The parameter specifies the threshold size at which interrupt is triggered to copy the data		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	1 - 64		
Default value	1		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	CanObjectType, CanHwObjectCount		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.18.6 CanHwObjectCount

Table 121 Specification for CanHwObjectCount

Name	`CanHwObjectCount`

(table continues...)

Table 121 (continued) Specification for CanHwObjectCount

Description	Number of the hardware objects used to implement one HOH. In case of an HRH this parameter defines the number of elements in the hardware FIFO (for HRH objects the range is from 1 to 64). In case of a HTH it defines the number of elements in the Tx queue used for multiplexed transmission (for HTH objects the range is from 1 to 32). The maximum hardware object count is limited to 64 per controller. The limitation comes from the memory assigned per controller.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	1 - 64		
Default value	1		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	ECU
Dependency	CanMultiplexedTransmission, CanObjectType		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.18.7 CanIdType

Table 122 Specification for CanIdType

Name	`CanIdType`		
Description	Specifies whether the CanHwFilterCode value is of following type: - standard identifier - extended identifier - mixed mode The default value of the CanIdType configuration parameter is set to STANDARD as it is the commonly used CanId.		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	EXTENDED: all the CANIDs are of extended type only (29 bit). MIXED: The type of CANIDs can be both standard and extended type. STANDARD: all the CANIDs are of standard type only (11bit).		
Default value	STANDARD		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-

(table continues...)

Table 122 (continued) Specification for CanIdType

Origin	AUTOSAR_ECUC	**Scope**	ECU
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.18.8 CanMainFunctionRWPeriodRef

Table 123 Specification for CanMainFunctionRWPeriodRef

Name	`CanMainFunctionRWPeriodRef`		
Description	Reference to CanMainFunctionPeriod It is dependent on CanMainFunctionRWPeriods. It is applicable only when the referenced CAN controllers CanRxProcessing or CanTxProcessing or both are POLLING.		
Multiplicity	0..1	**Type**	EcucReferenceDef
Range	Reference to Node: CanMainFunctionRWPeriods		
Default value	NULL		
Post-build variant value	TRUE	**Post-build variant multiplicity**	TRUE
Value configuration class	Post-Build	**Multiplicity configuration class**	Post-Build
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	CanObjectType, CanTxProcessing, CanRxProcessing		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.18.9 CanObjectId

Table 124 Specification for CanObjectId

Name	`CanObjectId`

(table continues...)

Table 124 (continued) Specification for CanObjectId

Description	Holds the handle ID of HRH or HTH. The value of this parameter is unique in a given CAN driver, and it should start with 0 and continue without any gaps. The HRH and HTH Ids share a common ID range. Example: HRH0-0, HRH1-1, HTH0-2, HTH1-3 Configuration rules to be followed: HRHs belonging to a controller should be grouped together HTHs belonging to a controller should be grouped together All HRHs should have lower CanObjectId than all HTHs Configuration example: HRHs of Controller0 is from 0 to 4 HRHs of Controller1 is from 5 to 9 HRHs of Controller2 is from 10 to 14 HRHs of Controller3 is from 15 to 19 HTHs of Controller0 is from 20 to 24 HTHs of Controller1 is from 25 to 29 HTHs of Controller2 is from 30 to 34 HTHs of Controller3 is from 35 to 39 *Note: 'N' is the maximum number of hardware objects that can be configured and depends on the hardware device being used.*		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - N-1		
Default value	0		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	ECU
Dependency	CanObjectType		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.18.10 CanObjectType

Table 125 Specification for CanObjectType

Name	`CanObjectType`		
Description	Specifies if the HardwareObject is used as a transmit or receive object The default value is set to RECEIVE because when configuring hardware objects, first the RECEIVE objects should be configured followed by the TRANSMIT objects.		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef

(table continues...)

Table 125 (continued) Specification for CanObjectType

Range	RECEIVE: Receive HOH TRANSMIT: Transmit HOH		
Default value	RECEIVE		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	ECU
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.18.11 CanTriggerTransmitEnable

Table 126 Specification for CanTriggerTransmitEnable

Name	`CanTriggerTransmitEnable`		
Description	Defines whether or not the CAN supports the trigger-transmit API for this handle. By default, the optional interface APIs are disabled to minimize the executable code size.		
Multiplicity	0..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	FALSE
Value configuration class	Pre-Compile	**Multiplicity configuration class**	Pre-Compile
Origin	AUTOSAR_ECUC	**Scope**	ECU
Dependency	CanObjectType		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.2 Functions - Type definitions

1.3.2.1 Can_17_McmCan_LPduRxCalloutFnPtrType

Table 127 Specification for Can_17_McmCan_LPduRxCalloutFnPtrType

Syntax	`Can_17_McmCan_LPduRxCalloutFnPtrType`

(table continues...)

Table 127 (continued) Specification for Can_17_McmCan_LPduRxCalloutFnPtrType

Type	Pointer to a function of type boolean Function_Name (const Can_HwHandleType Hrh, const Can_IdType CanId, const uint8 CanDataLength, const uint8 * const CanSduPtr)
File	`Can_17_McmCan_PBcfg.c`
Description	Pointer to the L-PDU Callout function
Source	AUTOSAR
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.2.2 CanTrcv_TrcvModeType

Table 128 Specification for CanTrcv_TrcvModeType

Syntax	`CanTrcv_TrcvModeType`	
Type	Enumeration	
File	`Can_GeneralTypes.h`	
Range	0 - CANTRCV_TRCVMODE_NORMAL	Transceiver mode Normal
	1 - CANTRCV_TRCVMODE_SLEEP	Transceiver mode Sleep
	2 - CANTRCV_TRCVMODE_STANDBY	Transceiver mode StandBy
Description	The data type defines the operating modes of the CAN transceiver driver.	
Source	AUTOSAR	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.2.3 CanTrcv_TrcvWakeupModeType

Table 129 Specification for CanTrcv_TrcvWakeupModeType

Syntax	`CanTrcv_TrcvWakeupModeType`	
Type	Enumeration	
File	`Can_GeneralTypes.h`	
Range	0 - CANTRCV_WUMODE_ENABLE	The notification for wakeup events are enabled on the addressed transceiver.
	1 - CANTRCV_WUMODE_DISABLE	The notification for wakeup events are disabled on the addressed transceiver.
	2 - CANTRCV_WUMODE_CLEAR	The stored notification events are cleared on the addressed transceiver.
Description	The data type is used to control the CanTrcv concerning the wakeup events and wakeup notifications.	
Source	AUTOSAR	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.2.4 CanTrcv_TrcvWakeupReasonType

Table 130 Specification for CanTrcv_TrcvWakeupReasonType

Syntax	`CanTrcv_TrcvWakeupReasonType`	
Type	Enumeration	
File	`Can_GeneralTypes.h`	
Range	0 - CANTRCV_WU_ERROR	Due to an error wake up reason is not detected. This value may only be reported when the production error is reported to the Mcal_Wrapper before.
	1 - CANTRCV_WU_NOT_SUPPORTED	The transceiver does not support any information for the wake up reason.
	2 - CANTRCV_WU_BY_BUS	The transceiver has detected that the network has caused the wake up of the ECU.
	3 - CANTRCV_WU_INTERNALLY	The transceiver has detected that the network has been woken up by the ECU through a request to the NORMAL mode.
	4 - CANTRCV_WU_RESET	The transceiver has detected, that the wakeup is due to an ECU reset.
	5 - CANTRCV_WU_POWER_ON	The transceiver has detected, that the wakeup is due to an ECU reset after power on.
	6 - CANTRCV_WU_BY_PIN	The transceiver has detected, that the wakeup is due to a state held at the pin.
	7 - CANTRCV_WU_BY_SYSERR	The transceiver has detected, that the wake up of the ECU was caused by a hardware related device failure.
Description	The data type denotes the wake up reason detected by the CanTrcv.	
Source	AUTOSAR	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.2.5 Can_ControllerStateType

Table 131 Specification for Can_ControllerStateType

Syntax	`Can_ControllerStateType`	
Type	Enumeration	
File	`Can_GeneralTypes.h`	
Range	0 - CAN_CS_UNINIT	CAN controller state UNINIT.
	1 - CAN_CS_STARTED	CAN controller state STARTED.
	2 - CAN_CS_STOPPED	CAN controller state STOPPED.
	3 - CAN_CS_SLEEP	CAN controller state SLEEP.

(table continues...)

Table 131 (continued) Specification for Can_ControllerStateType

Description	The data type represents the CAN controller state types as defined by the CAN controller state machine.
Source	AUTOSAR
Autosar Version	Applicable for Autosar version 4.4.0.

1.3.2.6 Can_ErrorStateType

Table 132 Specification for Can_ErrorStateType

Syntax	`Can_ErrorStateType`	
Type	Enumeration	
File	`Can_GeneralTypes.h`	
Range	0 - CAN_ERRORSTATE_ACTIVE	The CAN controller takes fully part in communication.
	1 - CAN_ERRORSTATE_PASSIVE	When in Passive does not send any frame, but controller can still receive packets.
	2 - CAN_ERRORSTATE_BUSOFF	The CAN controller does not take part in communication.
Description	The data type defines the error state of the CAN controller.	
Source	AUTOSAR	
Autosar Version	Applicable for Autosar version 4.4.0.	

1.3.2.7 Can_HwHandleType

Table 133 Specification for Can_HwHandleType

Syntax	`Can_HwHandleType`	
Type	uint16	
File	`Can_GeneralTypes.h`	
Range	0x00 - 0xFFFF	By default, extended type is defined
Description	The data type represents the hardware object handles of a CAN hardware unit. For CAN hardware units with more than 255 hardware objects, uses the extended range.	
Source	AUTOSAR	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.2.8 Can_HwType

Table 134 Specification for Can_HwType

Syntax	`Can_HwType`
Type	Structure

(table continues...)

Table 134 (continued) Specification for Can_HwType

File	`Can_GeneralTypes.h`	
Range	Can_IdType CanId	Standard/Extended CAN ID of CAN L-PDU
	Can_HwHandleType Hoh	ID of the corresponding Hardware Object Range
	uint8 ControllerId	ControllerId provided by CanIf clearly identify the corresponding controller
Description	The data type defines a data structure which clearly provides a hardware object handle including its corresponding CAN controller and therefore CanDrv as well as the specific CanId.	
Source	AUTOSAR	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.2.9 Can_PduType

Table 135 Specification for Can_PduType

Syntax	`Can_PduType`	
Type	Structure	
File	`Can_GeneralTypes.h`	
Range	PduIdType swPduHandle	Software PDU handle
	uint8 length	Number of SDU data bytes
	Can_IdType id	Formatted CAN message identifier
	uint8 * sdu	Pointer to data bytes
Description	The data type unites PduId (swPduHandle), SduLength (length), SduData (sdu), and CanId (id) for any CAN L-SDU.	
Source	AUTOSAR	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.2.10 Can_IdType

Table 136 Specification for Can_IdType

Syntax	`Can_IdType`	
Type	uint32	
File	`Can_GeneralTypes.h`	
Range	0x00- 0xDFFFFFFF	By default, extended 32-bit is defined

(table continues...)

Table 136 (continued) Specification for Can_IdType

Description	The data type represents the identifier of an L-PDU. The two most significant bits specify the frame type: 00 CAN message with Standard CAN ID 01 CAN FD frame with Standard CAN ID 10 CAN message with Extended CAN ID 11 CAN FD frame with Extended CAN ID The type can be either uint16 or uint32 (type can be uint16 when all HOH's are of STANDARD type otherwise the type should be uint32). The CAN driver should support both uint16 and uint32. Standard32Bit - 0 to 0x400007FF Standard16Bit - 0 to 0x47FF Extended32Bit - 0 to 0xDFFFFFFF
Source	AUTOSAR
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.2.11 Can_StateTransitionType

Table 137 Specification for Can_StateTransitionType

Syntax	`Can_StateTransitionType`	
Type	Enumeration	
File	`Can_GeneralTypes.h`	
Range	0 - CAN_T_START	CAN controller transition value to request state STARTED.
	1 - CAN_T_STOP	CAN controller transition value to request state STOPPED.
	2 - CAN_T_SLEEP	CAN controller transition value to request state SLEEP.
	3 - CAN_T_WAKEUP	CAN controller transition value to request state STOPPED from state SLEEP.
Description	The data type denotes the CAN controller state transitions.	
Source	AUTOSAR	
Autosar Version	Applicable for Autosar version 4.2.2.	

1.3.2.12 Can_ReturnType

Table 138 Specification for Can_ReturnType

Syntax	`Can_ReturnType`
Type	Enumeration

(table continues...)

Table 138 (continued) Specification for Can_ReturnType

File	`Can_GeneralTypes.h`	
Range	0 - CAN_OK	Success
	1 - CAN_NOT_OK	Error or wakeup event occurred during sleep transition
	2 - CAN_BUSY	Transmit request could not be processed because no transmit object was available
Description	The data type represents the return values of the CAN driver APIs	
Source	AUTOSAR	
Autosar Version	Applicable for Autosar version 4.2.2.	

1.3.2.13 Can_17_McmCan_ConfigType

Table 139 Specification for Can_17_McmCan_ConfigType

Syntax	`Can_17_McmCan_ConfigType`	
Type	Structure	
File	`Can_17_McmCan.h`	
Range	--	The elements of the data structure are specific to the micro-controller
Description	The data 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. It contains the definition of the implementation-specific post build configuration structure of the CAN driver.	
Source	AUTOSAR	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.2.14 Can_17_Mcmcan_DrvStateMachine

Table 140 Specification for Can_17_Mcmcan_DrvStateMachine

Syntax	`Can_17_Mcmcan_DrvStateMachine`	
Type	Enumeration	
File	`Can_17_McmCan.h`	
Range	0 - CAN_17_MCMCA_UNINIT	The driver state is UNINIT.
	1 - CAN_17_MCMCAN_READY	The driver state is READY
Description	The data type specifies the CAN driver state machine states.	
Source	IFX	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.3 Functions - APIs

This section lists all the APIs of the CAN driver.

1.3.3.1 Can_17_McmCan_Init

Table 141 Specification for `Can_17_McmCan_Init` API

Syntax	`void  Can_17_McmCan_Init`
	`(`
	`    const Can_17_McmCan_ConfigType * const Config`
	`)`	

Service ID	0x0	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Non Reentrant	
Parameters (in)	Config	Pointer to the CAN driver root configuration
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	The function initializes all global variables and relevant registers of the MCMCAN (based on configuration) assigned to that particular core with the values of structure referenced by the parameter Config. Successful execution of this API will trigger a state transition of the CAN Driver state machine from CAN_UNINIT to CAN_READY state. The controllers initialized shall be configured to reject reception of CAN frames with remote transmission requests (i.e. Frames with RTR bit set) This API must be invoked from all the cores using the CAN driver, as each call initializes only the SFRs and global variables of the CAN controllers used by the invoking core. The kernel clocks and common resource initialization are initialized by the MCALs master core. The CAN initialization status is set at the end of Initialization function execution.	
Source	AUTOSAR	
Error handling	CAN_17_MCMCAN_E_MASTER_CORE_UNINIT, CAN_17_MCMCAN_E_NOT_CONFIGURED, CAN_17_MCMCAN_E_TRANSITION, CAN_17_MCMCAN_E_INIT_FAILED	
Configuration dependencies	-	
User hints	None	

(table continues...)

Table 141 (continued) Specification for `Can_17_McmCan_Init` API

SFR accessed	CAN_CLC(rw), CAN_MCR(rw), CAN_N_CCCR(rw), CAN_N_DBTP(w), CAN_N_GFC(ex_w), CAN_N_GRINT1(ex_w), CAN_N_GRINT2(ex_w), CAN_N_IR(w), CAN_N_NBTP(w), CAN_N_NDAT1(w), CAN_N_NDAT2(w), CAN_N_NPCR(ex_w), CAN_N_PSR(r), CAN_N_RWD(ex_w), CAN_N_RX_BC(ex_w), CAN_N_RX_ESC(ex_w), CAN_N_RX_F0C(ex_w), CAN_N_RX_F0S(r), CAN_N_RX_F1C(ex_w), CAN_N_RX_F1S(r), CAN_N_SIDFC(ex_w), CAN_N_TDCR(w), CAN_N_TX_BC(w), CAN_N_TX_BTIE(ex_w), CAN_N_TX_EFC(ex_w), CAN_N_TX_EFS(r), CAN_N_TX_ESC(ex_w), CAN_N_TX_FQS(r), CAN_N_XIDFC(ex_w), CPU_CORE_ID(r), SCU_CCUCON0(r), SCU_EICON0(rw), SCU_OSCCON(r), SCU_SYSPLLCON0(r), SCU_SYSPLLCON1(r), STM_TIM0(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.3.2 Can_17_McmCan_DeInit

Table 142 Specification for `Can_17_McmCan_DeInit` API

Syntax	`void  Can_17_McmCan_DeInit`
	`(`
	`  void`
	`)`	

Service ID	0x10	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Non Reentrant	
Parameters (in)	-	-
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	The function de-initializes all global variables and relevant registers of the MCMCAN (based on configuration) assigned to that particular core with the values of structure referenced by the parameter ConfigPtr. Successful execution of this API will trigger a state transition of the CAN Driver state machine from CAN_READY to CAN_UNINIT state. The Can_17_McmCan_DeInit() function is available only when CanDeInitApi is enabled. in case of AUTOSAR 4.2.2, the parameter can be enabled or disabled. In AUTOSAR 4.4.0 the parameter will always generate TRUE and will be disabled.	
Source	AUTOSAR	
Error handling	CAN_17_MCMCAN_E_UNINIT, CAN_17_MCMCAN_E_SLAVE_CORE_INIT, CAN_17_MCMCAN_E_TRANSITION	

(table continues...)

Table 142 (continued) Specification for `Can_17_McmCan_DeInit` API

Configuration dependencies	CanDeInitApi
User hints	None
SFR accessed	CAN_CLC(rw), CAN_KRST0(rw), CAN_KRST1(rw), CAN_KRSTCLR(rw), CPU_CORE_ID(r), SCU_CCUCON0(r), SCU_EICON0(rw), SCU_OSCCON(r), SCU_SYSPLLCON0(r), SCU_SYSPLLCON1(r), STM_TIM0(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.3.3 Can_17_McmCan_SetControllerMode

Table 143 Specification for `Can_17_McmCan_SetControllerMode` API

Syntax	`Can_ReturnType  Can_17_McmCan_SetControllerMode`
	`(`
	`    const uint8 Controller,`
	`    const Can_StateTransitionType Transition`
	`)`	

Service ID	0x03	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Non Reentrant	
Parameters (in)	Controller Transition	CAN controller for which the controller mode status shall be changed Transition value to request new CAN controller state
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Can_ReturnType	CAN_OK: Request accepted CAN_NOT_OK: Request not accepted, or, a development error
Description	The function performs software triggered state transitions of the CAN controller state machine. The function is implemented synchronous as the change in the mode is done synchronously by the hardware. This is a deviation from AUTOSAR. Also there is no HW support to wakeup the controller, it is only logical sleep which is implemented in driver.	
Source	AUTOSAR	

(table continues...)

Table 143 (continued) Specification for `Can_17_McmCan_SetControllerMode` API

Error handling	CAN_17_MCMCAN_E_TRANSITION, CAN_17_MCMCAN_E_PARAM_CONTROLLER, CAN_17_MCMCAN_E_UNINIT, CAN_17_MCMCAN_E_NOT_CONFIGURED
Configuration dependencies	-
User hints	None
SFR accessed	CAN_CLC(r), CAN_MCR(r), CAN_N_CCCR(rw), CAN_N_IE(w), CAN_N_IR(rw), CAN_N_NDAT1(rw), CAN_N_NDAT2(rw), CAN_N_PSR(r), CAN_N_RX_F0A(w), CAN_N_RX_F0S(r), CAN_N_RX_F1A(w), CAN_N_RX_F1S(r), CAN_N_TX_BCR(w), CAN_N_TX_BRP(r), CPU_CORE_ID(r), STM_TIM0(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*
Autosar Version	Applicable for Autosar version 4.2.2.

1.3.3.4 Can_17_McmCan_SetControllerMode

Table 144 Specification for `Can_17_McmCan_SetControllerMode` API

Syntax	`Std_ReturnType Can_17_McmCan_SetControllerMode`
	`(`
	`    const uint8 Controller,`
	`    const Can_ControllerStateType Transition`
	`)`	

Service ID	0x3	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Non Reentrant	
Parameters (in)	Controller Transition	CAN controller for which the controller mode status shall be changed Transition value to request new CAN controller state
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Std_ReturnType	E_OK: Request accepted E_NOT_OK: Request not accepted, or, a development error occurred.

(table continues...)

Table 144 **(continued) Specification for `Can_17_McmCan_SetControllerMode` API**

Description	The function performs software triggered state transitions of the CAN controller State machine. The function is implemented synchronous as the change in the mode is done synchronously by the hardware. This is a deviation from AUTOSAR. Also there is no HW support to wakeup the controller, it is only logical sleep which is implemented in driver.
Source	AUTOSAR
Error handling	CAN_17_MCMCAN_E_NOT_CONFIGURED, CAN_17_MCMCAN_E_UNINIT, CAN_17_MCMCAN_E_PARAM_CONTROLLER, CAN_17_MCMCAN_E_TRANSITION
Configuration dependencies	-
User hints	None
SFR accessed	CAN_CLC(r), CAN_MCR(r), CAN_N_CCCR(rw), CAN_N_IE(w), CAN_N_IR(rw), CAN_N_NDAT1(rw), CAN_N_NDAT2(rw), CAN_N_PSR(r), CAN_N_RX_F0A(w), CAN_N_RX_F0S(r), CAN_N_RX_F1A(w), CAN_N_RX_F1S(r), CAN_N_TX_BCR(w), CAN_N_TX_BRP(r), CPU_CORE_ID(r), STM_TIM0(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*
Autosar Version	Applicable for Autosar version 4.4.0.

1.3.3.5 Can_17_McmCan_SetBaudrate

Table 145 **Specification for `Can_17_McmCan_SetBaudrate` API**

Syntax	`Std_ReturnType  Can_17_McmCan_SetBaudrate`
	`(`
	`    const uint8 Controller,`
	`    const uint16 BaudRateConfigID`
	`)`	

Service ID	0x0F	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant for different controllers. Non reentrant for the same controller.	
Parameters (in)	Controller BaudRateConfigID	CAN controller for which the, baud rate needs to be set Unique Id with a specific baud rate configuration
Parameters (out)	-	-
Parameters (in - out)	-	-

(table continues...)

Table 145 (continued) Specification for `Can_17_McmCan_SetBaudrate` API

Return	Std_ReturnType	E_OK: Service request accepted, setting of new baud rate started E_NOT_OK: Service request not accepted, or, development error reported.
Description	The function sets the baud rate configuration of the CAN controller during runtime when the CAN controller is in STOPPED state. The Can_17_McmCan_SetBaudrate() function is available only when CanSetBaudrateApi is enabled.	
Source	AUTOSAR	
Error handling	CAN_17_MCMCAN_E_PARAM_CONTROLLER, CAN_17_MCMCAN_E_UNINIT, CAN_17_MCMCAN_E_PARAM_BAUDRATE, CAN_17_MCMCAN_E_NOT_CONFIGURED	
Configuration dependencies	CanSetBaudrateApi	
User hints	None	
SFR accessed	CAN_CLC(r), CAN_MCR(r), CAN_N_CCCR(rw), CAN_N_DBTP(w), CAN_N_NBTP(w), CAN_N_PSR(r), CAN_N_TDCR(w), CPU_CORE_ID(r), STM_TIM0(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.3.6 Can_17_McmCan_DisableControllerInterrupts

Table 146 Specification for `Can_17_McmCan_DisableControllerInterrupts` API

Syntax	`void  Can_17_McmCan_DisableControllerInterrupts`
	`(`
	`    const uint8 Controller`
	`)`	

Service ID	0x04	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant	
Parameters (in)	Controller	CAN controller for which interrupts need to be disabled
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	The function disables all interrupts for the given CAN controller	

(table continues...)

Table 146 (continued) Specification for `Can_17_McmCan_DisableControllerInterrupts` API

Source	AUTOSAR
Error handling	CAN_17_MCMCAN_E_UNINIT, CAN_17_MCMCAN_E_NOT_CONFIGURED, CAN_17_MCMCAN_E_PARAM_CONTROLLER
Configuration dependencies	-
User hints	None
SFR accessed	CAN_N_IE(rw), CPU_CORE_ID(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.3.7 Can_17_McmCan_EnableControllerInterrupts

Table 147 Specification for `Can_17_McmCan_EnableControllerInterrupts` API

Syntax	`void  Can_17_McmCan_EnableControllerInterrupts`
	`(`
	`    const uint8 Controller`
	`)`	

Service ID	0x05	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant	
Parameters (in)	Controller	CAN controller for which interrupts shall be re-enabled
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	The functions re-enables the allowed interrupts of the given CAN controller	
Source	AUTOSAR	
Error handling	CAN_17_MCMCAN_E_PARAM_CONTROLLER, CAN_17_MCMCAN_E_UNINIT, CAN_17_MCMCAN_E_NOT_CONFIGURED	
Configuration dependencies	-	
User hints	None	

(table continues...)

Table 147 **(continued) Specification for `Can_17_McmCan_EnableControllerInterrupts` API**

SFR accessed	CAN_N_IE(rw), CPU_CORE_ID(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.3.8 Can_17_McmCan_SetIcomConfiguration

Table 148 **Specification for `Can_17_McmCan_SetIcomConfiguration` API**

Syntax	`Std_ReturnType  Can_17_McmCan_SetIcomConfiguration`
	`(`
	`    const uint8 Controller,`
	`    const IcomConfigIdType ConfigurationId`
	`)`	

Service ID	0x21	
Sync/Async	Asynchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant for different Controllers. Non reentrant for the same Controller.	
Parameters (in)	Controller ConfigurationId	CAN controller for which the status shall be changed. Requested configuration. An ID greater than 0 identifies a configuration in which pretended networking is activated for the Controller. An ID value of 0 deactivates the pretended networking identifier that is activated for the Controller.
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Std_ReturnType	E_OK: CAN driver succeeded in setting a configuration with a valid Configuration id. E_NOT_OK: CAN driver failed to set a configuration with a valid Configuration id, or, development error occurred
Description	The API should change the Icom configuration of a CAN controller to the requested one. The Can_17_McmCan_SetIcomConfiguration() function is available only when CanPublicIcomSupport is enabled. *Note: For the API Can_SetIcomConfiguration, as per AUTOSAR 4.2.2 has a service ID 0xf which is also the service ID for Can_SetBaudRate.* *In AUTOSAR 4.4.0 the service ID correction for Can_SetIcomConfiguration was done and was modified to 0x21 so that it did not conflict with Can_SetBaudRate service ID (0xf).* *Hence the Can_SetIcomConfiguration shall have the service ID 0x21 in both AUTOSAR versions as per A2GT-PRQ-12538.*	

(table continues...)

Table 148 **(continued) Specification for `Can_17_McmCan_SetIcomConfiguration` API**

Source	AUTOSAR
Error handling	CAN_17_MCMCAN_E_ICOM_CONFIG_INVALID, CAN_17_MCMCAN_E_PARAM_CONTROLLER, CAN_17_MCMCAN_E_UNINIT, CAN_17_MCMCAN_E_NOT_CONFIGURED
Configuration dependencies	CanPublicIcomSupport
User hints	None
SFR accessed	CAN_CLC(r), CAN_MCR(r), CAN_N_CCCR(rw), CAN_N_IE(w), CAN_N_PSR(r), CPU_CORE_ID(r), STM_TIM0(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.3.9 Can_17_McmCan_Write

Table 149 **Specification for `Can_17_McmCan_Write` API**

Syntax	`Std_ReturnType  Can_17_McmCan_Write`
	`(`
	`    const Can_HwHandleType Hth,`
	`    const Can_PduType * const PduInfo`
	`)`	

Service ID	0x06	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant.	
Parameters (in)	Hth PduInfo	Information which hardware transmit handle should be used for transmit. Implicitly this is also the information about the controller to use because the Hth numbers are unique inside a hardware unit. Pointer to the SDU user memory, DLC and Identifier
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Std_ReturnType	E_OK: Write command has been accepted E_NOT_OK: Development error occurred CAN_BUSY: No TX hardware buffer available or pre-emptive call of Can_Write that cannot be implemented re-entrant.
Description	This function is used to transmit CAN/CAN FD frame based on the information passed to it. The CAN driver will only transmit messages with remote transmission request (RTR) bit at reset state (that is, no remote transmission request will be accepted by the CAN driver).	

(table continues...)

Table 149 (continued) Specification for `Can_17_McmCan_Write` API

Source	AUTOSAR
Error handling	CAN_17_MCMCAN_E_PARAM_MSGID, CAN_17_MCMCAN_E_NOT_CONFIGURED, CAN_17_MCMCAN_E_PARAM_POINTER, CAN_17_MCMCAN_E_PARAM_HANDLE, CAN_17_MCMCAN_E_UNINIT, CAN_17_MCMCAN_E_PARAM_DATA_LENGTH, CAN_17_MCMCAN_E_PARAM_CONTROLLER
Configuration dependencies	-
User hints	-
SFR accessed	CAN_N_CCCR(r), CAN_N_TX_BAR(w), CAN_N_TX_BC(r), CAN_N_TX_FQS(r), CPU_CORE_ID(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*
Autosar Version	Applicable for Autosar version 4.4.0.

1.3.3.10 Can_17_McmCan_Write

Table 150 Specification for `Can_17_McmCan_Write` API

Syntax	`Can_ReturnType  Can_17_McmCan_Write`
	`(`
	`    const Can_HwHandleType Hth,`
	`    const Can_PduType * const PduInfo`
	`)`	

Service ID	0x06	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant.	
Parameters (in)	Hth PduInfo	Information which hardware transmit handle should be used for transmit. Implicitly this is also the information about the controller to use because the Hth numbers are unique inside a hardware unit. Pointer to the SDU user memory, DLC and Identifier
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Can_ReturnType	CAN_OK: Write command has been accepted CAN_NOT_OK: Development error occurred CAN_BUSY: No TX hardware buffer available or pre-emptive call of Can_Write that cannot be implemented re-entrant.

(table continues...)

Table 150 **(continued) Specification for `Can_17_McmCan_Write` API**

Description	This function is used to transmit CAN/CAN FD frame based on the information passed to it. The CAN driver will only transmit messages with remote transmission request (RTR) bit at reset state (that is, no remote transmission request will be accepted by the CAN driver).
Source	AUTOSAR
Error handling	CAN_17_MCMCAN_E_PARAM_POINTER, CAN_17_MCMCAN_E_PARAM_DLC, CAN_17_MCMCAN_E_PARAM_HANDLE, CAN_17_MCMCAN_E_PARAM_MSGID, CAN_17_MCMCAN_E_UNINIT, CAN_17_MCMCAN_E_NOT_CONFIGURED, CAN_17_MCMCAN_E_PARAM_CONTROLLER
Configuration dependencies	-
User hints	None
SFR accessed	CAN_N_CCCR(r), CAN_N_TX_BAR(w), CAN_N_TX_BC(r), CAN_N_TX_FQS(r), CPU_CORE_ID(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*
Autosar Version	Applicable for Autosar version 4.2.2.

1.3.3.11 Can_17_McmCan_GetControllerMode

Table 151 **Specification for `Can_17_McmCan_GetControllerMode` API**

Syntax	`Std_ReturnType Can_17_McmCan_GetControllerMode`
	`(`
	`const uint8 Controller,`
	`Can_ControllerStateType * const ControllerModePtr`
	`)`	

Service ID	0x12	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Non Reentrant	
Parameters (in)	Controller	CAN controller for which the status shall be requested.
Parameters (out)	ControllerModePtr	Pointer to a memory location, where the current mode of the CAN controller will be stored.
Parameters (in - out)	-	-
Return	Std_ReturnType	E_OK: Controller mode request has been accepted. E_NOT_OK: Development error has been reported.

(table continues...)

Table 151 (continued) Specification for `Can_17_McmCan_GetControllerMode` API

Description	The function reports about the current controller status of the requested CAN controller. Note: In case if driver is in uninitialized state and DET is off, this API will report controller mode as CAN_CS_UNINIT and returns E_OK. if DET is on then a DET will be raised and E_NOT_OK will be returned.
Source	AUTOSAR
Error handling	CAN_17_MCMCAN_E_PARAM_POINTER, CAN_17_MCMCAN_E_PARAM_CONTROLLER, CAN_17_MCMCAN_E_UNINIT, CAN_17_MCMCAN_E_NOT_CONFIGURED
Configuration dependencies	-
User hints	None
SFR accessed	CPU_CORE_ID(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*
Autosar Version	Applicable for Autosar version 4.4.0.

1.3.3.12 Can_17_McmCan_GetControllerErrorState

Table 152 Specification for `Can_17_McmCan_GetControllerErrorState` API

Syntax	`Std_ReturnType  Can_17_McmCan_GetControllerErrorState`
	`(`
	`    const uint8 ControllerId,`
	`    Can_ErrorStateType * const ErrorStatePtr`
	`)`	

Service ID	0x11	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant for different controller. Non Reentrant for the same controller	
Parameters (in)	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN controller, which is requested for ErrorState.
Parameters (out)	ErrorStatePtr	Pointer to a memory location, where the error state of the CAN controller will be stored.
Parameters (in - out)	-	-
Return	Std_ReturnType	E_OK: Error state request has been accepted. E_NOT_OK: Error state request has not been accepted or development error has been reported.
Description	The function obtains the error state of the CAN controller by reading the error state register.	
Source	AUTOSAR	

(table continues...)

Table 152 **(continued) Specification for `Can_17_McmCan_GetControllerErrorState` API**

Error handling	CAN_17_MCMCAN_E_UNINIT, CAN_17_MCMCAN_E_PARAM_CONTROLLER, CAN_17_MCMCAN_E_PARAM_POINTER, CAN_17_MCMCAN_E_NOT_CONFIGURED
Configuration dependencies	-
User hints	None
SFR accessed	CAN_N_PSR(r), CPU_CORE_ID(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*
Autosar Version	Applicable for Autosar version 4.4.0.

1.3.3.13 Can_17_McmCan_GetControllerTxErrorCounter

Table 153 **Specification for `Can_17_McmCan_GetControllerTxErrorCounter` API**

Syntax	`Std_ReturnType  Can_17_McmCan_GetControllerTxErrorCounter`
	`(`
	`    const uint8 ControllerId,`
	`    uint8 * const TxErrorCounterPtr`
	`)`	

Service ID	0x31	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant for different controller. Non Reentrant for the same controller.	
Parameters (in)	ControllerId	CAN controller, whose current Tx error counter shall be acquired.
Parameters (out)	TxErrorCounterPtr	Pointer to a memory location, where the current Tx error counter of the CAN controller will be stored.
Parameters (in - out)	-	-
Return	Std_ReturnType	E_OK: Tx error counter available. E_NOT_OK: Development error occurred.
Description	The API returns the Tx error counter for a CAN controller. *Note: The value of the counter might not be correct at the moment the API returns it, because the Tx counter is handled asynchronously in hardware. Applications should not trust this value for any assumption about the current bus state.*	
Source	AUTOSAR	
Error handling	CAN_17_MCMCAN_E_UNINIT, CAN_17_MCMCAN_E_PARAM_CONTROLLER, CAN_17_MCMCAN_E_PARAM_POINTER, CAN_17_MCMCAN_E_NOT_CONFIGURED	
Configuration dependencies	-	

(table continues...)

Table 153 (continued) Specification for `Can_17_McmCan_GetControllerTxErrorCounter` API

User hints	None
SFR accessed	CAN_N_ECR(r), CPU_CORE_ID(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*
Autosar Version	Applicable for Autosar version 4.4.0.

1.3.3.14 Can_17_McmCan_GetControllerRxErrorCounter

Table 154 Specification for `Can_17_McmCan_GetControllerRxErrorCounter` API

Syntax	`Std_ReturnType Can_17_McmCan_GetControllerRxErrorCounter`
	`(`
	`    const uint8 ControllerId,`
	`    uint8 * const RxErrorCounterPtr`
	`)`	

Service ID	0x30	
Sync/Async	Asynchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant for different controller. Non Reentrant for the same controller.	
Parameters (in)	ControllerId	CAN controller, whose current Rx error counter shall be acquired.
Parameters (out)	RxErrorCounterPtr	Pointer to a memory location, where the current Rx error counter of the CAN controller will be stored.
Parameters (in - out)	-	-
Return	Std_ReturnType	E_OK: Rx error counter available. E_NOT_OK: Development error occurred.
Description	The API returns the Rx error counter for a particular CAN controller. *Note: In passive state the counter value will be always 128 due to hardware limitation.* *Note: The value of the counter might not be correct at the moment the API returns it, because the Rx counter is handled asynchronously in hardware. Applications should not trust this value for any assumption about the current bus state.*	
Source	AUTOSAR	
Error handling	CAN_17_MCMCAN_E_PARAM_POINTER, CAN_17_MCMCAN_E_PARAM_CONTROLLER, CAN_17_MCMCAN_E_UNINIT, CAN_17_MCMCAN_E_NOT_CONFIGURED	
Configuration dependencies	-	
User hints	None	

(table continues...)

Table 154 (continued) Specification for `Can_17_McmCan_GetControllerRxErrorCounter` API

SFR accessed	CAN_N_ECR(r), CPU_CORE_ID(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*
Autosar Version	Applicable for Autosar version 4.4.0.

1.3.3.15 Can_17_McmCan_GetVersionInfo

Table 155 Specification for `Can_17_McmCan_GetVersionInfo` API

Syntax	`void  Can_17_McmCan_GetVersionInfo`
	`(`
	`    Std_VersionInfoType * const versioninfo`
	`)`	

Service ID	0x07	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant	
Parameters (in)	-	-
Parameters (out)	versioninfo	Pointer to the location to store the version information of this module.
Parameters (in - out)	-	-
Return	void	-
Description	This functions provides the version information of the CAN driver The Can_17_McmCan_GetVersionInfo() function is available only when CanVersionInfoApi is enabled.	
Source	AUTOSAR	
Error handling	CAN_17_MCMCAN_E_PARAM_POINTER	
Configuration dependencies	CanVersionInfoApi	
User hints	None	
SFR accessed	-	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.3.16 Can_17_McmCan_CheckBaudrate

Table 156 Specification for `Can_17_McmCan_CheckBaudrate` API

Syntax	`Std_ReturnType Can_17_McmCan_CheckBaudrate`
	`(`
	`    const uint8 Controller,`
	`    const uint16 Baudrate`
	`)`	

Service ID	0x0E	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant for different controller. Non reentrant for same controller.	
Parameters (in)	Controller Baudrate	Associated CAN controller Baudrate to be checked
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Std_ReturnType	E_OK: Service request accepted, checking of baud rate started. E_NOT_OK: Service request not accepted or development error occured.
Description	This function checks the baud rate of the CAN controller.	
Source	AUTOSAR	
Error handling	CAN_17_MCMCAN_E_PARAM_CONTROLLER, CAN_17_MCMCAN_E_UNINIT, CAN_17_MCMCAN_E_PARAM_BAUDRATE, CAN_17_MCMCAN_E_NOT_CONFIGURED	
Configuration dependencies	CanSetBaudrateApi	
User hints	None	
SFR accessed	CPU_CORE_ID(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	
Autosar Version	Applicable for Autosar version 4.2.2.	

1.3.4 Notifications and Callbacks

The CAN driver does not provide any notification or callbacks.

1.3.5 Scheduled functions

This section lists all the scheduled functions of the CAN driver.

1.3.5.1 Can_17_McmCan_MainFunction_Read

Table 157 Specification for `Can_17_McmCan_MainFunction_Read` API

Syntax	`void  Can_17_McmCan_MainFunction_Read`
	`(`
	`  void`
	`)`	

Service ID	0x08	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Non Reentrant	
Parameters (in)	-	-
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	This main function performs the task of processing all the HRH objects configured as polling and if respective messages are received will provide notification to upper layer. The function performs the polling of receive indication when CanRxProcessing is set to POLLING or MIXED. In case of MIXED processing only the hardware objects for which CanHardwareObjectUsesPolling is set to TRUE shall be polled. The function is implemented as an empty define if none of the RX processing for any of the configured controllers or hardware objects (in case of mixed mode) is chosen as POLLING. In case the value of CanMainFunctionRWPeriod is 0 or 1, Can_17_McmCan_MainFunctionRead is used. In case it is greater than 1, Can_17_McmCan_MainFunctionRead_(x) is used.	
Source	AUTOSAR	
Error handling	CAN_17_MCMCAN_E_DATALOST	
Configuration dependencies	CanHardwareObjectUsesPolling,CanMainFunctionRWPeriods	
User hints	None	
SFR accessed	CAN_N_IE(r), CAN_N_IR(rw), CAN_N_NDAT1(rw), CAN_N_NDAT2(rw), CAN_N_RX_F0A(rw), CAN_N_RX_F0S(r), CAN_N_RX_F1A(rw), CAN_N_RX_F1S(r), CPU_CORE_ID(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	
Autosar Version	Applicable for Autosar version 4.2.2.	

1.3.5.2 Can_17_McmCan_MainFunction_Read

Table 158 Specification for `Can_17_McmCan_MainFunction_Read` API

Syntax	`void  Can_17_McmCan_MainFunction_Read`
	`(`
	`  void`
	`)`	

Service ID	0x08	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Non Reentrant	
Parameters (in)	-	-
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	The main function performs the task of processing all the HRH objects configured as polling and if respective messages are received will provide notification to upper layer. The function performs the polling of receive indication when CanRxProcessing is set to POLLING or MIXED. In case of MIXED processing only the hardware objects for which CanHardwareObjectUsesPolling is set to TRUE shall be polled. The function is implemented as an empty define if none of the RX processing for any of the configured controllers or hardware objects (in case of mixed mode) is chosen as POLLING. In case the value of CanMainFunctionRWPeriod is 0 or 1, Can_17_McmCan_MainFunctionRead is used. In case it is greater than 1, Can_17_McmCan_MainFunctionRead_(x) is used.	
Source	AUTOSAR	
Error handling	CAN_17_MCMCAN_E_DATALOST	
Configuration dependencies	CanMainFunctionRWPeriods,CanHardwareObjectUsesPolling	
User hints	-	
SFR accessed	-	
Autosar Version	Applicable for Autosar version 4.4.0.	

1.3.5.3 Can_17_McmCan_MainFunction_Read_(x)

Table 159 Specification for `Can_17_McmCan_MainFunction_Read_(x)` API

Syntax	`void  Can_17_McmCan_MainFunction_Read_(x)`
	`(`
	`  void`
	`)`	

Service ID	0x08	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Non Reentrant	
Parameters (in)	-	-
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	The function performs the polling of receive indication when CanRxProcessing is set to POLLING or MIXED. In case of MIXED processing only the hardware objects for which CanHardwareObjectUsesPolling is set to TRUE shall be polled. The function name shall be appended with _x, when the number of elements in the parameter list CanMainFunctionRWPeriods is greater than 1 that is referenced by at least one RECEIVE CanHardwareObject. e.g.: Elements in the parameter list CanMainFunctionRWPeriods is 2 (i.e. greater than 1), then two functions will be generated namely: Can_17_McmCan_MainFunction_Read_0 and Can_17_McmCan_MainFunction_Read_1 these functions will poll for the HRH configured for their respective periods. In case the value of CanMainFunctionRWPeriod is 0 or 1, Can_17_McmCan_MainFunctionRead is used. In case it is greater than 1, Can_17_McmCan_MainFunctionRead_(x) is used. Note that _x represent the periodicity with which this function needs to be polled. Only the HRH objects associated with this period is only processed in this function.	
Source	AUTOSAR	
Error handling	CAN_17_MCMCAN_E_DATALOST	
Configuration dependencies	CanHardwareObjectUsesPolling,CanMainFunctionRWPeriods	
User hints	None	
SFR accessed	CAN_N_IR(rw), CAN_N_NDAT1(rw), CAN_N_NDAT2(rw), CAN_N_RX_F0A(r), CAN_N_RX_F0S(r), CAN_N_RX_F1A(r), CAN_N_RX_F1S(r), CPU_CORE_ID(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	

(table continues...)

Table 159 (continued) Specification for `Can_17_McmCan_MainFunction_Read_(x)` API

Autosar Version	Applicable for Autosar version 4.2.2.

1.3.5.4 Can_17_McmCan_MainFunction_Read_(x)

Table 160 Specification for `Can_17_McmCan_MainFunction_Read_(x)` API

Syntax	`void  Can_17_McmCan_MainFunction_Read_(x)`
	`(`
	`  void`
	`)`	

Service ID	0x08	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Non Reentrant	
Parameters (in)	-	-
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	The function performs the polling of receive indication when CanRxProcessing is set to POLLING or MIXED. In case of MIXED processing only the hardware objects for which CanHardwareObjectUsesPolling is set to TRUE shall be polled. The function name shall be appended with _x, when the number of elements in the parameter list CanMainFunctionRWPeriods is greater than 1 that is referenced by at least one RECEIVE CanHardwareObject. e.g.: Elements in the parameter list CanMainFunctionRWPeriods is 2 (i.e. greater than 1), then two functions will be generated namely: Can_17_McmCan_MainFunction_Read_0 and Can_17_McmCan_MainFunction_Read_1 these functions will poll for the HRH configured for their respective periods. In case the value of CanMainFunctionRWPeriod is 0 or 1, Can_17_McmCan_MainFunctionRead is used. In case it is greater than 1, Can_17_McmCan_MainFunctionRead_(x) is used. The RX processing starts when the threshold value of FIFO reaches watermark. Note that _x represent the periodicity with which this function needs to be polled. Only the HRH objects associated with this period is only processed in this function.	
Source	AUTOSAR	
Error handling	CAN_17_MCMCAN_E_DATALOST	
Configuration dependencies	CanHardwareObjectUsesPolling	

(table continues...)

Table 160 (continued) Specification for `Can_17_McmCan_MainFunction_Read_(x)` API

User hints	-
SFR accessed	-
Autosar Version	Applicable for Autosar version 4.4.0.

1.3.5.5 Can_17_McmCan_MainFunction_Write

Table 161 Specification for `Can_17_McmCan_MainFunction_Write` API

Syntax	`void  Can_17_McmCan_MainFunction_Write`
	`(`
	`  void`
	`)`	

Service ID	0x01	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Non Reentrant	
Parameters (in)	-	-
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	The function shall perform the polling of TX confirmation when CanTxProcessing is set to POLLING or MIXED. In case of MIXED processing only the hardware objects for which CanHardwareObjectUsesPolling is set to TRUE shall be polled. The function is implemented as an empty define in case no polling at all is used. In case the value of CanMainFunctionRWPeriod is 0 or 1, Can_17_McmCan_MainFunctionWrite is used. In case it is greater than 1, Can_17_McmCan_MainFunctionWrite_(x) is used. The Tx slots are not freed until transmit notifications is not provided to the upper layer.	
Source	AUTOSAR	
Error handling	CAN_17_MCMCAN_E_DATALOST	
Configuration dependencies	CanHardwareObjectUsesPolling	
User hints	-	
SFR accessed	-	
Autosar Version	Applicable for Autosar version 4.4.0.	

1.3.5.6 Can_17_McmCan_MainFunction_Write_(x)

Table 162 Specification for `Can_17_McmCan_MainFunction_Write_(x)` API

Syntax	`void  Can_17_McmCan_MainFunction_Write_(x)`
	`(`
	`  void`
	`)`	

Service ID	0x01	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Non Reentrant	
Parameters (in)	-	-
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	The function shall perform the polling of Tx confirmation when CanTxProcessing is set to POLLING or MIXED. In case of MIXED processing only the hardware objects for which CanHardwareObjectUsesPolling is set to TRUE shall be polled. The function name shall be appended with _x, when the number of elements in the parameter list CanMainFunctionRWPeriods is greater than 1. Note that _x represent the periodicity with which this function needs to be polled. Only the HTH objects associated with this period is only processed in this function. For example: Elements in the CanMainFunctionRWPeriods parameter list are two (that is, greater than 1), then the following two functions are generated: Can_17_McmCan_MainFunction_Write_0 and Can_17_McmCan_MainFunction_Write_1. These functions poll for the HTH configured for their respective periods. In case the value of CanMainFunctionRWPeriod is 0 or 1, Can_17_McmCan_MainFunctionWrite is used. In case it is greater than 1, Can_17_McmCan_MainFunctionWrite_(x) is used. The Tx slots are not freed until transmit notifications is not provided to the upper layer.	
Source	AUTOSAR	
Error handling	CAN_17_MCMCAN_E_DATALOST	
Configuration dependencies	-	
User hints	-	
SFR accessed	-	
Autosar Version	Applicable for Autosar version 4.4.0.	

1.3.5.7 Can_17_McmCan_MainFunction_Write

Table 163 Specification for `Can_17_McmCan_MainFunction_Write` API

Syntax	`void Can_17_McmCan_MainFunction_Write`
	`(`
	`void`
	`)`	

Service ID	0x01	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Non Reentrant	
Parameters (in)	-	-
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	The function shall perform the polling of TX confirmation when CanTxProcessing is set to POLLING or MIXED. In case of MIXED processing only the hardware objects for which CanHardwareObjectUsesPolling is set to TRUE shall be polled. The function is implemented as an empty define in case no polling at all is used. In case the value of CanMainFunctionRWPeriod is 0 or 1, Can_17_McmCan_MainFunctionWrite is used. In case it is greater than 1, Can_17_McmCan_MainFunctionWrite_(x) is used. The Tx slots are not freed until transmit notifications is not provided to the upper layer.	
Source	AUTOSAR	
Error handling	CAN_17_MCMCAN_E_DATALOST	
Configuration dependencies	CanHardwareObjectUsesPolling,CanMainFunctionRWPeriods	
User hints	None	
SFR accessed	CAN_N_IR(rw), CAN_N_TX_BTO(r), CAN_N_TX_EFA(rw), CAN_N_TX_EFS(r), CPU_CORE_ID(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	
Autosar Version	Applicable for Autosar version 4.2.2.	

1.3.5.8 Can_17_McmCan_MainFunction_Write_(x)

Table 164 Specification for `Can_17_McmCan_MainFunction_Write_(x)` API

Syntax	`void  Can_17_McmCan_MainFunction_Write_(x)`
	`(`
	`  void`
	`)`	

Service ID	0x01	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Non Reentrant	
Parameters (in)	-	-
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	The function shall perform the polling of Tx confirmation when CanTxProcessing is set to POLLING or MIXED. In case of MIXED processing only the hardware objects for which CanHardwareObjectUsesPolling is set to TRUE shall be polled. The function name shall be appended with _x, when the number of elements in the parameter list CanMainFunctionRWPeriods is greater than 1. Note that _x represent the periodicity with which this function needs to be polled. Only the HTH objects associated with this period is only processed in this function. For example: Elements in the CanMainFunctionRWPeriods parameter list are two (that is, greater than 1), then the following two functions are generated: Can_17_McmCan_MainFunction_Write_0 and Can_17_McmCan_MainFunction_Write_1. These functions poll for the HTH configured for their respective periods. In case the value of CanMainFunctionRWPeriod is 0 or 1, Can_17_McmCan_MainFunctionWrite is used. In case it is greater than 1, Can_17_McmCan_MainFunctionWrite_(x) is used. The Tx slots are not freed until transmit notifications is not provided to the upper layer.	
Source	AUTOSAR	
Error handling	CAN_17_MCMCAN_E_DATALOST	
Configuration dependencies	CanHardwareObjectUsesPolling,CanMainFunctionRWPeriods	
User hints	-	
SFR accessed	CAN_N_IR(rw), CAN_N_TX_BTO(r), CAN_N_TX_EFA(w), CAN_N_TX_EFS(r), CPU_CORE_ID(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	

(table continues...)

Table 164 (continued) Specification for `Can_17_McmCan_MainFunction_Write_(x)` API

Autosar Version	Applicable for Autosar version 4.2.2.

1.3.5.9 Can_17_McmCan_MainFunction_BusOff

Table 165 Specification for `Can_17_McmCan_MainFunction_BusOff` API

Syntax	`void  Can_17_McmCan_MainFunction_BusOff`
	`(`
	`  void`
	`)`	

Service ID	0x09	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Non Reentrant	
Parameters (in)	-	-
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	The function performs the polling of bus-off events that are configured statically as 'to be polled'. Bus-off notification will be provided to upper layer only once when the hardware detects bus-off. If bus-off remains after the first notification, no further notifications will be provided to upper layer. The function is implemented as an empty define in case no polling at all is used.	
Source	AUTOSAR	
Error handling	-	
Configuration dependencies	-	
User hints	-	
SFR accessed	-	
Autosar Version	Applicable for Autosar version 4.4.0.	

1.3.5.10 Can_17_McmCan_MainFunction_BusOff

Table 166 Specification for `Can_17_McmCan_MainFunction_BusOff` API

Syntax	`void  Can_17_McmCan_MainFunction_BusOff`
	`(`
	`  void`
	`)`	

Service ID	0x09	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Non Reentrant	
Parameters (in)	-	-
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	The function performs the polling of bus-off events that are configured statically as 'to be polled'. Bus-off notification will be provided to upper layer only once when the hardware detects bus-off. If bus-off remains after the first notification, no further notifications will be provided to upper layer. The function is implemented as an empty if the RX processing for none of the configured controllers is chosen as POLLING	
Source	AUTOSAR	
Error handling	-	
Configuration dependencies	-	
User hints	None	
SFR accessed	CAN_N_CCCR(r), CAN_N_PSR(r), CAN_N_TX_BCR(w), CAN_N_TX_BRP(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	
Autosar Version	Applicable for Autosar version 4.2.2.	

1.3.5.11 Can_17_McmCan_MainFunction_Wakeup

Table 167 Specification for `Can_17_McmCan_MainFunction_Wakeup` API

Syntax	`void Can_17_McmCan_MainFunction_Wakeup`
	`(`
	`  void`
	`)`	

Service ID	0x0A	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Non Reentrant	
Parameters (in)	-	-
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	The function performs the polling of wake-up events that are configured statically as 'to be polled'. The function is implemented as an empty define in case no polling at all is used.	
Source	AUTOSAR	
Error handling	-	
Configuration dependencies	-	
User hints	None	
SFR accessed	CAN_N_IR(rw), CAN_N_NDAT1(rw), CAN_N_NDAT2(rw), CAN_N_RX_F0A(rw), CAN_N_RX_F0S(r), CAN_N_RX_F1A(rw), CAN_N_RX_F1S(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	
Autosar Version	Applicable for Autosar version 4.2.2.	

1.3.5.12 Can_17_McmCan_MainFunction_Wakeup

Table 168 Specification for `Can_17_McmCan_MainFunction_Wakeup` API

Syntax	`void Can_17_McmCan_MainFunction_Wakeup`
	`(`
	`  void`
	`)`

Service ID	0x0A

(table continues...)

Table 168 (continued) Specification for `Can_17_McmCan_MainFunction_Wakeup` API

Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Non Reentrant	
Parameters (in)	-	-
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	The function performs the polling of wake-up events that are configured statically as 'to be polled'. The function is implemented as an empty define in case no polling at all is used.	
Source	AUTOSAR	
Error handling	-	
Configuration dependencies	-	
User hints	-	
SFR accessed	-	
Autosar Version	Applicable for Autosar version 4.4.0.	

1.3.5.13 Can_17_McmCan_MainFunction_Mode

Table 169 Specification for `Can_17_McmCan_MainFunction_Mode` API

Syntax	`void Can_17_McmCan_MainFunction_Mode`
	`(`
	`  void`
	`)`	

Service ID	0x0c	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Non Reentrant	
Parameters (in)	-	-
Parameters (out)	-	-
Parameters (in - out)	-	-

(table continues...)

Table 169 (continued) Specification for `Can_17_McmCan_MainFunction_Mode` API

Return	void	-
Description	The function is supposed to poll for the CAN controller mode transitions. The CAN driver has a synchronous mode setting mechanism and does not support the Can_17_McmCan_MainFunction_Mode() function. It is implemented as an empty function.	
Source	AUTOSAR	
Error handling	-	
Configuration dependencies	-	
User hints	None	
SFR accessed	-	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.6 Interrupt service routines

This section lists all the interrupt handlers of CAN driver.

1.3.6.1 Can_17_McmCan_IsrBusOffHandler

Table 170 Specification for `Can_17_McmCan_IsrBusOffHandler` API

Syntax	`void Can_17_McmCan_IsrBusOffHandler`
	`(`
	`    const uint8 HwKernelId,`
	`    const uint8 NodeIdIndex`
	`)`	

Service ID	-	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant	
Parameters (in)	HwKernelId NodeIdIndex	The CAN controller which is to be processed, is associated with the passed Kernel The CAN node which is to be processed
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	The function checks the occurrence of bus-off events on the given CAN controller and gives corresponding notification to the upper layer. It resets the controller state to the STOPPED. The Can_17_McmCan_IsrBusOffHandler() handler is available only when, CanBusoffProcessing is enabled	

(table continues...)

Table 170 (continued) Specification for `Can_17_McmCan_IsrBusOffHandler` API

Source	IFX
Error handling	-
Configuration dependencies	CanBusoffProcessing
User hints	None
SFR accessed	CAN_N_CCCR(r), CAN_N_IR(rw), CAN_N_PSR(r), CAN_N_TX_BCR(w), CAN_N_TX_BRP(r), CPU_CORE_ID(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.6.2 Can_17_McmCan_IsrReceiveHandler

Table 171 Specification for `Can_17_McmCan_IsrReceiveHandler` API

Syntax	`void  Can_17_McmCan_IsrReceiveHandler`
	`(`
	`    const uint8 HwKernelId,`
	`    const uint8 NodeIdIndex`
	`)`	

Service ID	-	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant	
Parameters (in)	HwKernelId NodeIdIndex	The CAN controller which is to be processed, is associated with the passed Kernel The CAN node which is to be processed
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	The function should handle receive interrupts from dedicated receive buffers during CAN controller STARTED state. For dedicated reception the hardware filter code alone is considered, the receive mask available shall not be used during the filtering or processing of the message. In case of dedicated each hardware object can be configured as INTERRUPT or POLLING. However as the interrupt lines are shared, if one of the HRH is configured as INTERRUPT all dedicated objects on reception would trigger an interrupt.	
Source	IFX	

(table continues...)

Table 171 (continued) Specification for `Can_17_McmCan_IsrReceiveHandler` API

Error handling	-
Configuration dependencies	CanRxProcessing
User hints	None
SFR accessed	CAN_N_IR(rw), CAN_N_NDAT1(rw), CAN_N_NDAT2(rw), CAN_N_RX_F0A(rw), CAN_N_RX_F0S(r), CAN_N_RX_F1A(rw), CAN_N_RX_F1S(r), CAN_N_TX_BAR(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.6.3 Can_17_McmCan_IsrRxFIFOHandler

Table 172 Specification for `Can_17_McmCan_IsrRxFIFOHandler` API

Syntax	`void Can_17_McmCan_IsrRxFIFOHandler`
	`(`
	`    const uint8 HwKernelId,`
	`    const uint8 NodeIdIndex`
	`)`	

Service ID	-	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant	
Parameters (in)	HwKernelId NodeIdIndex	The CAN controller which is to be processed, is associated with the passed Kernel The CAN node which is to be processed
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-

(table continues...)

Table 172 **(continued) Specification for `Can_17_McmCan_IsrRxFIFOHandler` API**

Description	The function shall handle receive interrupts from FIFO 0 and FIFO 1 during CAN controller STARTED state. The ISR is triggered for FIFO0/ FIFO 1 on Watermark or on FIFO full event. Messages are read through FIFO and freed by acknowledging the slot to receive successive packet. Rx FIFO interrupt processes maximum of configured FIFO elements. In case the messages are received while the Rx FIFO messages are in progress and if number of messages received is greater than the configured threshold level; on exit of interrupt handler; watermark interrupt will not be triggered. Therefore all messages will be processed only on FULL interrupt. If FIFO overflow is set, an error CAN_17_MCMCAN_E_DATALOST is raised to indicate that few messages may be lost. RXFIFO 0 and 1 can be separately configured as INTERRUPT or polling in case mixed mode is used.
Source	IFX
Error handling	CAN_17_MCMCAN_E_DATALOST
Configuration dependencies	CanRxProcessing
User hints	None
SFR accessed	CAN_N_IR(rw), CAN_N_NDAT1(rw), CAN_N_NDAT2(rw), CAN_N_RX_F0A(rw), CAN_N_RX_F0S(r), CAN_N_RX_F1A(rw), CAN_N_RX_F1S(r), CAN_N_TX_BAR(rw), CPU_CORE_ID(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.6.4 Can_17_McmCan_IsrTransmitHandler

Table 173 **Specification for `Can_17_McmCan_IsrTransmitHandler` API**

Syntax	`void Can_17_McmCan_IsrTransmitHandler`
	`(`
	`    const uint8 HwKernelId,`
	`    const uint8 NodeIdIndex`
	`)`	

Service ID	-	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant	
Parameters (in)	HwKernelId NodeIdIndex	The CAN controller which is to be processed, is associated with the passed Kernel The CAN node which is to be processed

(table continues...)

Table 173 (continued) Specification for `Can_17_McmCan_IsrTransmitHandler` API

Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	The function identifies the message object belonging to the given CAN controller for which the transmission request was successful. It extracts the corresponding software PDU handle and gives notification to upper layer. The Can_17_McmCan_IsrTransmitHandler() handler is available only when CanTxProcessing is enabled Due to Mixed mode support, if one of the HTH is configured in INTERRUPT mode every successful transmission will trigger interrupt.	
Source	IFX	
Error handling	CAN_17_MCMCAN_E_DATALOST	
Configuration dependencies	CanTxProcessing	
User hints	None	
SFR accessed	CAN_N_IR(rw), CAN_N_TX_BAR(rw), CAN_N_TX_BTO(r), CAN_N_TX_EFA(rw), CAN_N_TX_EFS(r), CPU_CORE_ID(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.7 Callout

This section lists all the callout of the CAN driver.

1.3.7.1 LPDU_CalloutName

Table 174 Specification for `LPDU_CalloutName` API

Syntax	`boolean  LPDU_CalloutName`
	`(`
	`    const Can_HwHandleType Hrh,`
	`    const Can_IdType CanId,`
	`    const uint8 CanDataLength,`
	`    const uint8 * const CanSduPtr`
	`)`

Service ID	0x20
Sync/Async	Asynchronous
Safety Level	Refer to the release notes for the safety related info

(table continues...)

Table 174 (continued) Specification for `LPDU_CalloutName` API

Re-entrancy	Non Reentrant	
Parameters (in)	Hrh CanId CanDataLength CanSduPtr	The hardware receive handle which will be passed to the upper layer The CAN message ID The data length of the message Pointer to the SDU structure which indicates the message data
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	boolean	TRUE: L PDU Callout function is successful FALSE: L PDU callout function is not successful
Description	The AUTOSAR CAN module supports optional L-PDU callouts on every reception of L-PDU where LPDU_CalloutName has to be substituted with the concrete L-PDU callout name which is configurable. If the L-PDU callout returns false, the L-PDU shall not be processed any further. The L-PDU callout function is mapped in a separate memory section. *Note: The prototype is deviated from the AUTOSAR prototype. The Hrh is of type uint8 as per AUTOSAR however the number of HRH which is configured is more than 255. Hence, the type of Hrh is modified to be of Can_HwHandleType which can hold values uint8 or uint16.*	
Source	AUTOSAR	
Error handling	-	
Configuration dependencies	CanLPduReceiveCalloutFunction	
User hints	-	
SFR accessed	-	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.8 Errors Handling

Error Name: Description	Source	Error ID (AS422)	Type (AS422)	Error ID (AS440)	Type (AS440)
CAN_17_MCMCAN_E_PARAM_DLC: The error is reported in case Can_17_McmCan_Write () API service is called with a data length which is not within the range. The ranges are describes below: 1. If the length is more than 64 byte. 2. If the length is more than 8 byte and the CAN controller is not in CAN FD mode 3. If the length is more than 8 byte and the CAN controller is in CAN FD mode but the CAN FD flag in Can_PduType->id is not set	AUTOSAR	0x3	DET	NA	NA
CAN_17_MCMCAN_E_PARAM_DATA_LENGTH: The error is reported in case Can_17_McmCan_Write () API service is called with a data length which is not within the range. The ranges are describes below: 1. If the length is more than 64 byte. 2. If the length is more than 8 byte and the CAN controller is not in CAN FD mode 3. If the length is more than 8 byte and the CAN controller is in CAN FD mode but the CAN FD flag in Can_PduType->id is not set	AUTOSAR	NA	NA	0x3	DET
CAN_17_MCMCAN_E_PARAM_POINTER: The error is reported when an API service is called with a NULL pointer as its parameter.	AUTOSAR	0x1	DET	0x1	DET

Error Name: Description	Source	Error ID (AS422)	Type (AS422)	Error ID (AS440)	Type (AS440)
CAN_17_MCMCAN_E_PARAM_HANDLE: The error is reported in case the Can_17_McmCan_Write() API service is called with an Hth parameter which is not configured as a hardware transmit handle.	AUTOSAR	0x2	DET	0x2	DET
CAN_17_MCMCAN_E_PARAM_CONTROLLER: The error is reported in case the API services are called with the parameter controller which is out of the range or not configured for the particular core.	AUTOSAR	0x4	DET	0x4	DET
CAN_17_MCMCAN_E_UNINIT: The error is reported in case the API service is called without being initialized	AUTOSAR	0x5	DET	0x5	DET
CAN_17_MCMCAN_E_TRANSITION: The error is reported in case the API services are called with a state which triggers an invalid transition of the controller state machine.	AUTOSAR	0x6	DET	0x6	DET
CAN_17_MCMCAN_E_DATALOST: The error is triggered in case an API service is called when received CAN message in FIFO or Tx event is lost.	AUTOSAR	0x1	RUNTIME	0x1	RUNTIME
CAN_17_MCMCAN_E_PARAM_BAUDRATE: The error is reported in case an API service is called with invalid baudrate as a parameter.	AUTOSAR	0x8	DET	0x7	DET
CAN_17_MCMCAN_E_ICOM_CONFIG_INVALID: The error is called in case the Can_17_McmCan_SetIcomConfiguration () API service is called with 0 or an unconfigured ICOM configuration ID	AUTOSAR	0x9	DET	0x8	DET

Error Name: Description	Source	Error ID (AS422)	Type (AS422)	Error ID (AS440)	Type (AS440)
CAN_17_MCMCAN_E_INIT_FAILED: The error is reported in case the Can_17_McmCan_Init() API service is called with a configuration pointer which is NULL or the configuration is not the same as the intended core.	AUTOSAR	0xA	DET	0x9	DET
CAN_17_MCMCAN_E_PARAM_MSGID: The error is reported in case the Can_17_McmCan_Write API service is called with an invalid CAN message identifier as its parameter, which is neither STANDARD nor EXTENDED.	IFX	0xD	DET	0xD	DET
CAN_17_MCMCAN_E_NOT_CONFIGURED: The error is reported in case the API service tries to use a controller which is not configured to the core which has invoked the service.	IFX	0x64	DET	0x64	DET
CAN_17_MCMCAN_E_MASTER_CORE_UNINIT: The error is reported by the Can_17_McmCan_Init() API service in case the master core is in uninitialized state and a slave core initialization is invoked.	IFX	0x66	DET	0x66	DET
CAN_17_MCMCAN_E_SLAVE_CORE_INIT: The error is reported in case the Can_17_McmCan_Deinit() API is invoked by the master core before de-initialization of the slave cores is completed.	IFX	0x67	DET	0x67	DET

1.3.9 Deviations and limitations

This section describes the deviations and limitations of the CAN driver.

1.3.9.1 Deviations

This section describes the deviations of the CAN driver.

1.3.9.1.1 Software specification deviations

This section describes the deviations from software specification.

Table 175 Known deviations

Reference	Deviation
Specification of CAN Driver AUTOSAR Release 4.2.2 - [SWS_Can_00360] : Can_CheckWakeup. Specification of CAN Driver AUTOSAR Release 4.4.0 - [SWS_Can_00360] : Can_CheckWakeup.	The CAN driver does not support the Can_17_McmCan_CheckWakeup() API listed in AUTOSAR. The CAN driver does not support wake up from sleep over the CAN bus. Hence, the configuration parameter CanWakeupFunctionalityAPI that is associated with the enabling of the Can_17_McmCan_CheckWakeup() API is made non-editable.
Specification of CAN Driver AUTOSAR Release 4.2.2 - [ECUC_Can_00480]: CanControllerTrcvDelayCompensationOffset	For CanControllerTrcvDelayCompensationOffset parameter, the range has been extended to 65535 in order to accommodate CAN FD with the higher baudrates of > 2Mbps.
Specification of CAN Driver AUTOSAR Release 4.2.2 - [SWS_Can_00420] : The Can module shall reset the interrupt flag at the end of the ISR (if not done automatically by hardware). Specification of CAN Driver AUTOSAR Release 4.4.0 - [SWS_Can_00420] : The Can module shall reset the interrupt flag at the end of the ISR (if not done automatically by hardware).	RF0WE, RF1WE, RF0FE, RF1FE, TEFN, BO and DRX are the interrupts configured for the CAN module, in-order to ensure that the successive events are registered in IR during the processing of interrupts of previous received messages, interrupt flags are cleared at the start of the ISR. Note that the BO is the only exception and clears the interrupt at end of the ISR.
Specification of CAN Driver AUTOSAR Release 4.2.2 - [SWS_Can_00443] : The L-PDU-Callout prototype. Specification of CAN Driver AUTOSAR Release 4.4.0 - [SWS_Can_00443] : The L-PDU-Callout prototype.	As per AUTOSAR SWS, the LPDU callout prototype should be :</p>`boolean LPDU_CalloutName ( uint8 Hrh, Can_IdType CanId, uint8 CanDataLegth, const uint8* CanSduPtr )`<p><textarea rows="9">uint8 cannot hold all the HRH ids as it is possible to configure more than 255 hardware objects. Hence the proposal is to have HRH of Can_HwHandleType. This will be inline with the number of hardware objects which can be configured. Hence the prototype is defined as:</p>`boolean LPDU_CalloutName ( Can_HwHandleType Hrh, Can_IdType CanId, uint8 CanDataLegth, const uint8* CanSduPtr )`
Specification of CAN Driver AUTOSAR Release 4.2.2 - Can_SetIcomConfiguration	For the API Can_SetIcomConfiguration, as per AUTOSAR 4.2.2 has a service ID 0xf which is also the service ID for Can_SetBaudRate. In AUTOSAR 4.4.0 the service ID correction for Can_SetIcomConfiguration was done and was modified to 0x21 so that it did not conflict with Can_SetBaudRate service ID (0xf). Hence the Can_SetIcomConfiguration has the service ID 0x21 in both AUTOSAR versions.

(table continues...)

Table 175 (continued) Known deviations

Reference	Deviation
Specification of CAN Driver AUTOSAR Release 4.2.2 - 7.11.1 Development Errors	The error CAN_E_DATALOST indicates that receive FIFO is full and loss of received messages. Similarly, for transmitted objects, the transmit event FIFO is full and loss of transmit notification event. Since this error can happen during runtime as well, the CAN driver report the error CAN_E_DATALOST as the runtime error instead development errors. Note: In ‘Specification of CAN Driver AUTOSAR Release 4.4.0’, the error CAN_E_DATALOST is of type runtime error.
Specification of CAN Driver AUTOSAR Release 4.2.2 - [SWS_Can_00416]: Can_IdType	The CAN driver does not support uint16 Can_IdType. Only unit32 Can_IdType is supported. Note: For ‘Specification of CAN Driver AUTOSAR Release 4.4.0’, only unit32 is specified for Can_IdType.
For all requirements related to Runtime errors	Reporting of Runtime error: Det_ReportRuntimeError is done through Mcal_Wrapper_Det_ReportRuntimeError interface. This is applicable for only AUTOSAR 4.4.0. All runtime error related datatypes and modified interfaces inclusion shall be done via Mcal_Wrapper.h.

1.3.9.1.2 AMDC Violations

This section describes the violations reported by the Vector AMDC checker tool with respect to AUTOSAR.

Table 176 Violations reported by AMDC checker tool for A207

AMDC Rule	A207
Description	Maximum value of parameter 'Can/CanConfigSet/CanController/CanControllerBaudrateConfig/CanControllerFdBaudrateConfig/CanControllerTrcvDelayCompensationOffset' in VSMD (65535) may not be larger than maximum value defined in StMD (400). [Can_17_McmCan.bmd]

1.3.9.1.3 VSMD Violations

This section describes the violations reported by the EB VSMD checker tool with respect to AUTOSAR.

1.3.9.2 Limitations

This section describes the limitations of the CAN driver.

Table 177 Known limitations

Reference	Limitation
CanIf_RxIndication	CanIf_RxIndication contains the pdu information stored in the CAN driver internal memory so the upper layer must make a copy of pdu information once the CanIf_RxIndication is called by the CAN driver and should not be reusing the pointer passed by CAN driver.
RX FIFO handling in Interrupt mode	In the CAN driver for handling Rx FIFO through interrupt mode, watermark and FIFO full conditions are enabled. Rx FIFO interrupt processes maximum of configured FIFO elements. If messages are received while the Rx FIFO messages are being processed and if the number of messages in FIFO always stay above the configured threshold level during interrupt handler processing then watermark interrupt will not be triggered again. Therefore all messages will be processed only on FULL interrupt.
Hardware errata MCMCAN_AI.022	If Can_Write() API is invoked multiple times with different HTH(hardware objects) number and with same message ID, the message transmitted in the CAN bus are not in the increasing order of the HTH number. The order of the message transmitted in the CAN bus depends on the delay between Can_Write() API invocation. It is recommended to maintain the message id's different for different HW objects. Note: The CAN drivers chose the transmit buffer number based on HTH number. The HTH number and chosen transmit buffer number will be same.
Configuration of receive hardware objects (Dedicated and FIFO)	During CAN hardware object configuration, the receive objects of a controller shall be configured in Tresos in the increasing order of CanObjectId. If the receive FIFO objects are to be used then the receive FIFO objects shall be configured as the last receive type objects for each controller. That is, below order shall be followed for index and CanObjectId of receive objects per controller 1. Rx Dedicated 2. Rx FIFO0 3. Rx FIFO1

Revision history

Table 178 Revision History

Date	Version	Description
2022-06-13	7.0	Document is released.
2023-06-12	6.1	• 1.1.3.1 C file Structure section Figure 2 Can_C_File_Structure-1.png updated to show Mcal_Wrapper.h and Det.h inclusion by Can_17_McmCan.c file. • 1.1.3.1 C File Structure section Table 2 C File Structure updated to include Mcal_Wrapper.h and Det.h files. • 1.1.2 Hardware-software mapping section, Figure 1 Mapping of hardware-software interfaces updated to add Mcal_Wrapper. • DEM module removed and Mcal_Wrapper module added in 1.1.4.1 Integration with AUTOSAR stack section. • Runtime error information removed from DET module and added in Mcal_Wrapper module in 1.1.4.1 Integration with AUTOSAR stack section. • 1.3.2.4 CanTrcv_TrcvWakeupReasonType section updated DEM to Production Error for CANTRCV_WU_ERROR. • ASIL Level field changed to Safety Level with value as 'Refer to the release notes for the safety related info' for all functions under 1.3.3 Functions - APIs, 1.3.5 Scheduled functions, 1.3.6 Interrupt service routines, and 1.3.7 Callout • 1.3.1.5.2 CanHwFilterMask section description updated • 1.3.9.1.1 Software specification deviations section, Table 175 Known Deviations updated for following changes: - Added Specification of CAN Driver AUTOSAR Release 4.2.2 - [SWS_Can_00416]: Can_IdType deviation for uint16 Can_IdType not being supported. - Added the Reference "For all requirements related to Runtime errors" for Autosar requirements. Updated Description to add Mcal_Wrapper module information.
2022-06-21	6.0	Document is released.
2022-06-20	5.1	• Corrected the upper limit of the range of CanControllerBaudrateConfig parameter from 100 to 1000 kbps. • HSI of functions updated for SFRs used during Init and DeInit.
2021-12-03	5.0	Document is released.
2021-12-03	4.1	Limitation added for configuration of receive hardware objects.
2021-11-17	4.0	Document is released.
2021-11-17	3.1	• Deviations and limitations section updated about CAN_E_DATALOST error
2021-03-15	3.0	Document is released.
2021-03-12	2.1	• Enhanced information about limitation in Rx FIFO handling in interrupt mode. • Added information about deviation related to the CAN_E_DATALOST development error.
2020-12-07	2.0	Document is released.

(table continues...)

Table 178 (continued) Revision History

2020-12-02	1.1	• Added information about the order for the CanObjectId to MO mapping • Changed information about the source file which contains prototype of the L-PDU callout • Added information about issuing Can_Write requests after the occurrence of bus off
2020-08-17	1.0	Document is released.
2020-08-12	0.1	• Initial version • CAN driver chapter moved from MC-ISAR_TC3xx_UM_Basic to this document • CAN driver information updated as per AUTOSAR 4.2.2 and AUTOSAR 4.4.0

Trademarks

All referenced product or service names and trademarks are the property of their respective owners.

Edition 2023-06-13

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about any aspect of this document?

Email: erratum@infineon.com

Document reference
IFX-ocr1484806431059

Important notice

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics (“Beschaffenheitsgarantie”).

With respect to any examples, hints or any typical values stated herein and/or any information regarding the application of the product, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party.

In addition, any information given in this document is subject to customer’s compliance with its obligations stated in this document and any applicable legal requirements, norms and standards concerning customer’s products and any use of the product of Infineon Technologies in customer’s applications.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer’s technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

Warnings

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies’ products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.