VECTOR

CAN Interface

Technical Reference

Version 6.11.01

Authors	**Rüdiger Naas, Eugen Stripling**
Versions:	6.11.01
Status:	**Released**

1 Document Information

1.1 History

Author	Date	Version	Remarks
Eugen Stripling Rüdiger Naas	2012-07-17	5.00	ASR R4.0 Rev 3
Eugen Stripling	2013-04-03	5.01.00	ESCAN00065368 ESCAN00066338 ESCAN00066340 Adapted according to ESCAN00066285 Adapted according to ESCAN00065289 ESCAN00066396 Adapted according to ESCAN00064304
Rüdiger Naas	2013-07-24	5.01.01	ESCAN00066794
Eugen Stripling	2013-09-27	6.00.00	Adapted due to: AR4-307: J1939 support AR4-438: Dynamic address lookup table AR4-397: CAN FD support
Eugen Stripling	2014-05-19	6.01.00	CAN FD support extended: Rx-FD and Rx- and Tx-PDUs with up to 64 bytes payload
Rüdiger Naas	2014-07-10	6.02.00	Multiple CAN driver support
Eugen Stripling	2014-08-25	6.02.00	ESCAN00077304, Restriction concerning the handling of FD/Not-FD FullCAN-Rx-PDUs added
Eugen Stripling	2014-09-22	6.02.00	ESCAN00078524, CanTSyn added, Post-build selectable
Eugen Stripling	2014-11-25	6.03.00	Channel specific J1939 dynamic address
Eugen Stripling	2015-01-26	6.04.00	Chapter 3.8 adapted to changed implementation
Eugen Stripling	2015-05-18	6.05.00	Adapted due to FEAT-366
Eugen Stripling	2015-11-20	6.06.00	Adapted due to FEAT-1429
Eugen Stripling	2016-01-09	6.06.00	ESCAN00087340
Eugen Stripling	2016-02-22	6.07.00	Feature Extended RAM-check added, ESCAN00087587
Eugen Stripling	2016-06-24	6.08.00	Feature: Data checksum added
Eugen Stripling	2016-09-14	6.09.00	Adapted due to FEAT-2076:

			Behavior of Tx-PDU filter extended
Eugen Stripling	2016-09-26	6.09.00	Adapted due to FEAT-2024: Set reception mode
Eugen Stripling	2017-01-09	6.10.00	Improved due to ESCAN00093454
Eugen Stripling	2017-02-13		Adapted due to: FEAT-2140: TMC Checksum - Release feature FEAT-1914
Eugen Stripling	2017-02-28		ESCAN00094196, deviation from AUTOSAR documented by ESCAN00094121 added
Eugen Stripling	2017-08-04	6.11.00	ESCAN00096181
Eugen Stripling	2017-08-30	6.11.01	Typos corrected

Table 1-1 History of the Document

1.2 Reference Documents

No.	Title	Version
[1]	AUTOSAR_SWS_CANInterface.pdf	4.2.2 5.0.0 6.0.0
[2]	AUTOSAR_SWS_DevelopmentErrorTracer.pdf	3.2.0
[3]	AUTOSAR_SRS_BSWGeneral.pdf	3.2.0

Table 1-2 References Documents

Please note
We have configured the programs in accordance with your specifications in the questionnaire. Whereas the programs do support other configurations than the one specified in your questionnaire, Vector's release of the programs delivered to your company is expressly restricted to the configuration you have specified in the questionnaire.

Contents

Illustrations

Tables

2 Introduction

This document describes the functionality, API and configuration of the AUTOSAR CAN Interface as specified in [1]. It is based on the AUTOSAR specification release 4.0.3. The CAN Interface is a hardware independent layer with a standardized interface to the CAN Driver and CAN Transceiver Driver layer and upper layers like PDU Router, Communication Manager and the Network Management.

Supported AUTOSAR Release:	4.0.3	
Supported Configuration Variants:	Pre-compile, Link-time, Post-build-loadable	
Vendor ID:	CANIF_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	CANIF_MODULE_ID	60 (according to ref.[3])

2.1 Architecture Overview

The following figure shows where the CAN Interface is located in the AUTOSAR architecture.

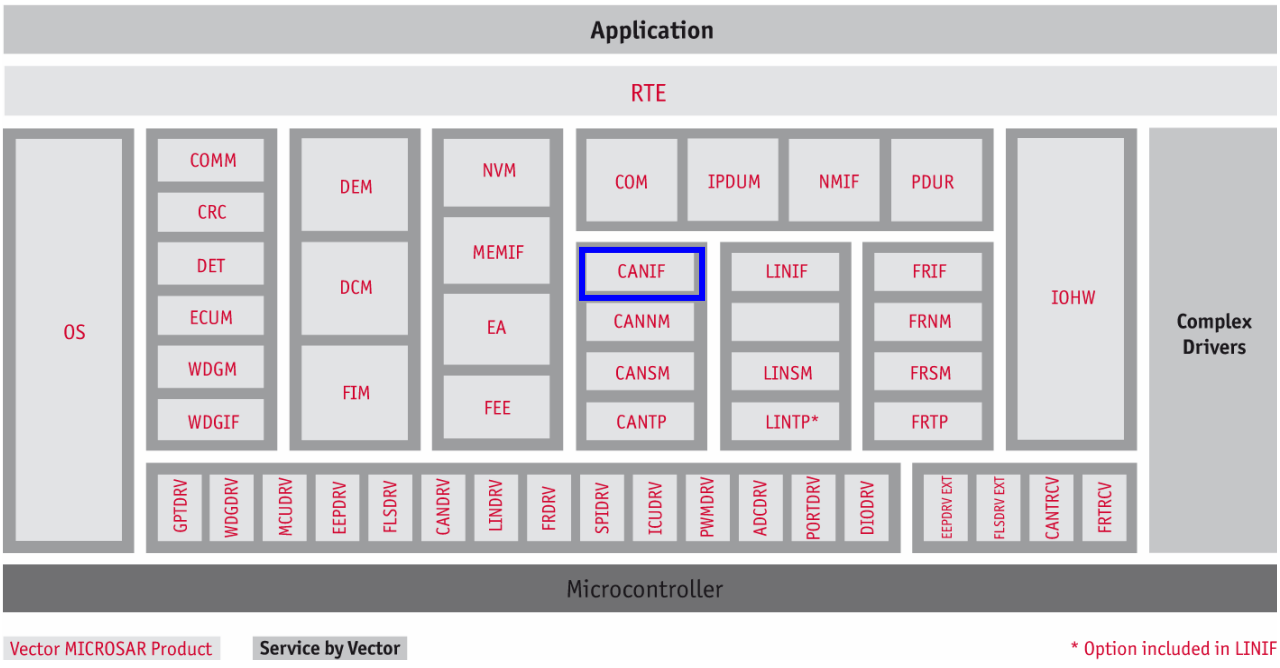

Figure 2-1 AUTOSAR layer model

The CAN Interface provides a standardized interface for all upper layers which require CAN communication. Therefore these upper layers have to communicate with the CAN

Interface which is responsible for the CAN communication. This includes the transmission and the reception of messages and the state handling of the CAN controllers as well.

The next figure shows the interfaces to adjacent modules of the CAN Interface. These interfaces are described in chapter 6.

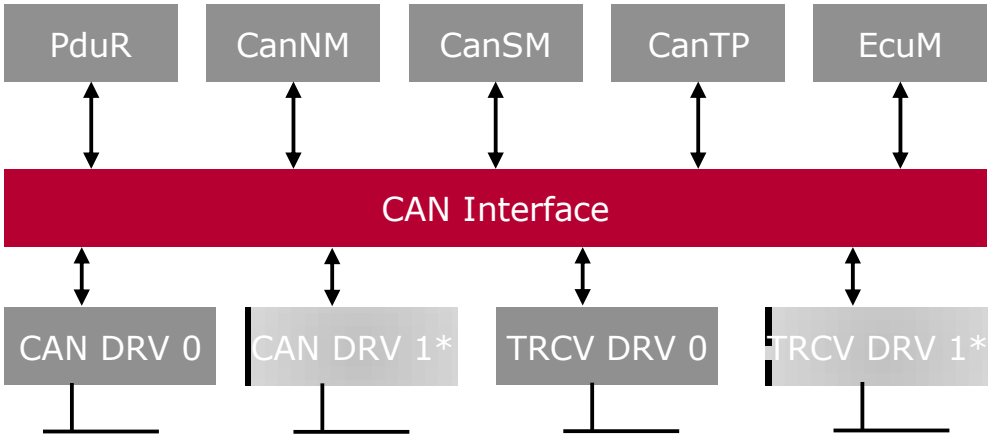

Figure 2-2 Interfaces to adjacent modules of the CAN Interface (* optional[1])

[1] NOTE: Multiple CAN driver and TRCV driver are supported optional

3 Functional Description

3.1 Deviations regarding AUTOSAR standard

Please note that the CAN Interface is tailored by Vector Informatik according to customer requirements before delivery. As a result not all features listed below might be supported by a delivered module.

For deviations and extensions regarding the AUTOSAR standard [1], please see chapter 7.

3.2 Feature List

Available Features For This BSW Module:

Feature Naming	Supported	Short Description
Initialization		
Generic Initialization	■	General initialization of the CAN Interface (CanIf_Init())
Communication		
Transmission	■	Transmission of PDUs
Dynamic transmission	■	Transmission of PDUs with changeable CAN IDs
Transmit-buffer	■	Buffering (send request and data) of Tx-PDUs mapped to a Tx-buffer in the CAN Interface. Two handling types of Tx-buffer are supported: FIFO and prioritized by CAN identifier.
Multiple Tx-BasicCAN hardware objects	■	Per CAN channel multiple Tx-BasicCAN hardware objects may be configured. This feature can only be used if the underlying CAN driver supports this feature as well.
Multiple transmit-buffers per CAN channel	■	Per CAN channel multiple independent transmit-buffers may be configured with different handling types: FIFO or prioritization by CAN identifier. This feature can only be used in combination with above mentioned feature “Multiple Tx-BasicCAN hardware objects”.
Cancellation of Tx-PDUs	■	Cancellation of PDUs and requeueing. (Feature to avoid inner priority inversion)
Transmit confirmation	■	Call back for successful transmission
Reception	■	Reception of PDUs
Receive indication	■	Call back for reception of PDUs
Control of reception mode of a Rx-PDU	■	This feature provides the ability to control the reception mode of a Rx-PDU individually at runtime.
DLC check	■	Check DLC of received PDUs against predefined values
CAN FD support	■	CAN with flexible data-rate
Meta data Rx- / Tx-support	■	Support for dynamic CAN identifier handling by using of SDU meta data
J1939 Dynamic Address Support	■	Translating of addresses according to J1939 by using of dynamic address lookup tables which are maintained by J1939Nm.
Data checksum Rx	■	Verification of checksum of Rx-PDUs
Data checksum Tx	■	Appending of checksum to Tx-PDUs
Controller Modes		
Sleep mode	■	Support sleep mode
External wake up (CAN)	■	Support external wake up by CAN Driver
External wake up (Transceiver)	■	Support external wake up by Transceiver Driver
Wake up validation	■	Support wake up validation for external wake up events
Internal wake up	■	Internal wake up by calling CanIf_SetControllerMode()

Stop mode	■	Support stop mode
BusOff detection	■	Handling of bus off notifications
Error Reporting		
DET	■	Support Development Error Detection (error notification)
Mailbox objects		
Tx BasicCAN	■	Standard mailbox to send CAN frames (Used by CAN Interface data queue)
Tx FullCAN	■	Separate mailbox for special Tx message used
Rx BasicCAN	■	Standard mailbox to receive CAN frames (depending on hardware, FIFO or shadow buffer supported)
Rx FullCAN	■	Separate mailbox for special Rx message used
Miscellaneous		
Transceiver handling	■	API for upper layers to set and read transceiver states; Interface to the Transceiver Driver
Version API	■	API to read out component version
Supported ID types - Standard Identifiers - Extended Identifiers - Mixed Identifiers	 ■ ■ ■	 Support of CAN Standard (11 bits) identifiers Support of CAN Extended (29 bits) identifiers Support standard as well as extended identifiers
Multiple CAN networks	■	Each CAN network has to be connected to exactly one controller
Multiple CAN driver	■	Supports multiple CAN driver
Partial Networking	■	Handling of partial networking transceiver Tx-PDU filter during wake-up
Tx Confirmation Polling Support	■	This service provides the information on whether any Tx confirmation has occurred for a CAN channel since the last start of that CAN channel at all.
Post-build loadable	■	Post-build loadable allows the re-configuration of an ECU at Post-build time
Post-build selectable	■	MICROSAR identity manager using Post-build selectable
Extended RAM-check	■	This service provides the ability in order to request an underlying CAN-channel to execute a check of CAN-controller-HW-registers. The usage of this feature requires a corresponding license.

Table 3-1 List of supported features

3.3 Initialization

Several functions are available to initialize the CAN Interface. The following code example shows which functions have to be called to initialize the CAN Interface and to allow transmission and reception.

```
CanIf_InitMemory();
      /* Mandatory call which reinitializes global variables to
                  set the CAN Interface back to uninitialized
                  state. */
CanTrcv_xxx_InitMemory() and CanTrcv_xxx_Init()
   /* have to be called to initialize the CAN Transceiver Driver
                  and set the CAN Transceiver to the preconfigured
```

```
                    state. For some CAN Controllers it is necessary
                    to have a recessive signal on the Rx Pin to be
                    able to initialize the CAN Controller. This
                    means the transceiver has to be set to “normal
                    mode” before CanIf_Init() is called. */
Can_InitMemory() and Can_Init();
   /* have to be called before CanIf_Init is called. */
CanIf_Init(<PtrToCanIfConfiguration>);
   /* Global initialization of the CAN Interface: all available CAN
                    Interface channels are initialized within this
                    call. If postbuild-selectable configuration is
                    active a valid configuration has to be passed to
                    CanIf_Init. In other cases the parameter is
                    ignored and a NULL pointer can be used */
CanIf_SetControllerMode(0, CANIF_CS_STARTED);
   /* The controller mode of CAN-channel 0 is set to started mode.
                    This means the CAN controller is initialized and
                    ready to communicate (acknowledge of the CAN
                    controller is activated). Communication is not
                    yet possible because the CAN Interface will
                    neither pass Tx PDUs from higher layers to the
                    CAN Driver nor accept Rx PDUs from the CAN
                    Driver. */
CanIf_SetPduMode(0, CANIF_SET_ONLINE);
   /* The PDU mode in the CAN Interface of the CAN-channel 0 is
                    switched to online mode. After initialization
                    this mode remains in the state CANIF_GET_OFFLINE
                    until the CanIf_SetPduMode function is called.
                    Now transmission requests will be passed from
                    the upper layer to the CAN Driver and Rx PDUs
                    are forwarded from the CAN Driver to the
                    corresponding higher layer. */
```

3.4 Transmission

The transmission of PDUs is only possible after the CAN Interface and CAN Driver are initialized and the CAN Interface resides in the CANIF_CS_STARTED / CANIF_GET_ONLINE or CANIF_CS_STARTED / CANIF_GET_TX_ONLINE mode. In all other states the Tx requests are rejected by the CAN Interface.

The Tx request has to be initiated by a call to the function:

```
CanIf_Transmit(<TxPduId>, <PduInfoPtr>);
```

The CAN Interface uses the PDU ID (<TxPduId>) to acquire more information from the generated data to be able to transmit the message. This data is used to call the function

Can_Write of the CAN Driver which needs information about the PDU like the CAN identifier, length of data, data by itself and the hardware transmit handle which represents the mailbox used for transmission of the PDU.

After a successful transmission of the message on the bus a confirmation function is called by the CAN Driver either from interrupt context or in case of Tx polling from task context. This confirmation is dispatched in the CAN Interface to notify the corresponding higher layer about the transmission of the PDU. For this purpose for each PDU a call back function has to be specified at configuration time.

The transmission request is rejected by returning E_NOT_OK in the following cases:

- The CAN Interface is not in the controller state CANIF_CS_STARTED
- The CAN Interface is not in the PDU mode CANIF_GET_ONLINE or CANIF_GET_TX_ONLINE
- The transmit buffer is not active and the corresponding mailbox used for transmission is occupied (BasicCAN Tx messages only).
- An error occurred during transmission (DET will be informed)

3.4.1 Dynamic transmission

The feature is activated by the parameter “Dynamic Tx Objects”.

The adjustments for the dynamic objects are the same as for the static with the exception that the CAN ID and the attribute whether extended or standard CAN ID can be selected manually.

By default the dynamic object has the CAN ID parameterized during configuration time until it is changed by the call of the API CanIf_SetDynamicTxId(). In order to set an extended CAN ID the most significant bit of its value passed to the API shall be set.

The PDU IDs of the dynamic objects are represented as symbolic handles in the file CanIf_Cfg.h.

3.4.2 Transmit-buffer

The CAN Interface provides a mechanism to buffer Tx-PDUs (including data) which are mapped to a Tx-buffer. This means if the Tx-hardware-object of such Tx-PDU is occupied the Tx-PDU-instance is stored within the CAN Interface until the Tx-hardware-object becomes free. Two handling types of a transmit-buffer are supported:

1. FIFO
2. Prioritization by CAN-identifier

The handling type defines in which manner the Tx-PDUs stored within the Tx-buffer are transmitted in case of the underlying Tx-hardware-object becomes free.

FIFO: The stored Tx-PDUs are transmitted in manner First-In-First-Out. Each Tx-PDU-instance is stored. If the FIFO is full then NO Tx-PDUs are stored until the FIFO becomes free.

Caution
In case of transmit-buffer of handling type FIFO only one instance of a Tx-PDU (the last one stored within the FIFO) can be and is cancelled from the FIFO via usage of API `CanIf_CancelTransmit`! (Feature: “Cancellation of Tx-PDUs”, see chapter 6.1.19).

Prioritization by CAN-identifier: The stored Tx-PDUs are transmitted in manner: Tx-PDU with high priority is sent before those one with lower priority. The priority is given by the CAN-identifier of the Tx-PDU. A Tx-PDU with a low CAN-identifier has higher priority than a one with greater CAN-identifier. The priority of a Tx-PDU is static and is determined from values of parameters `CanIfTxPduCanId` and `CanIfTxPduCanIdType`. Please consider this aspect in case of configuration of Tx-PDUs with dynamic CAN-identifier. Only one instance of each Tx-PDU is stored within such Tx-buffer: If a Tx-PDU is requested to be transmitted and the Tx-buffer of this Tx-PDU is already in use the already stored data of this Tx-PDU is overwritten in order to ensure the transmission of most newest data.

This handling type can be used to avoid inner priority inversion. This means if the CAN Interface passes a transmit request to the CAN Driver while all Tx-hardware-objects are occupied and at least one hardware object is occupied by a CAN message with lower priority than the message used for the current transmit request the CAN Driver initiates the cancellation of the message with the lowest priority. The cancelled CAN-message is stored in the Tx-buffer of the CAN Interface if the corresponding Tx-buffer is free. Otherwise it is discarded to ensure the transmission of most newest data. By this way a Tx-hardware message object becomes free and allows the CAN Interface to pass the CAN-message with the highest priority to the CAN Driver.

Caution
The described: “inner priority inversion” is only supported if at most only one Tx-buffer of handling type: Prioritization by CAN-identifier is configured per CAN-channel!

At all the Tx-PDUs stored within a Tx-buffer are processed either in context of the Tx-confirmation interrupt or in context of CAN Driver’s Tx-main-function in case of polling mode.

The configuration of multiple transmit-buffers is described in chapter 3.4.3.

3.4.3 Multiple Transmit-buffers

This feature can only be used if the underlying CAN driver supports the feature “Multiple Tx-BasicCAN hardware objects”. The Figure 3 shows the objects which are needed to be

configured within the EcuC-configuration and the relationship among themselves. For each Transmit-buffer a triple of objects: `CanHardwareObject`, `CanIfHthCfg` and `CanIfBufferCfg` must be configured and linked with each other and to corresponding CAN-channel (objects: `CanController` and `CanIfCtrlCfg`).

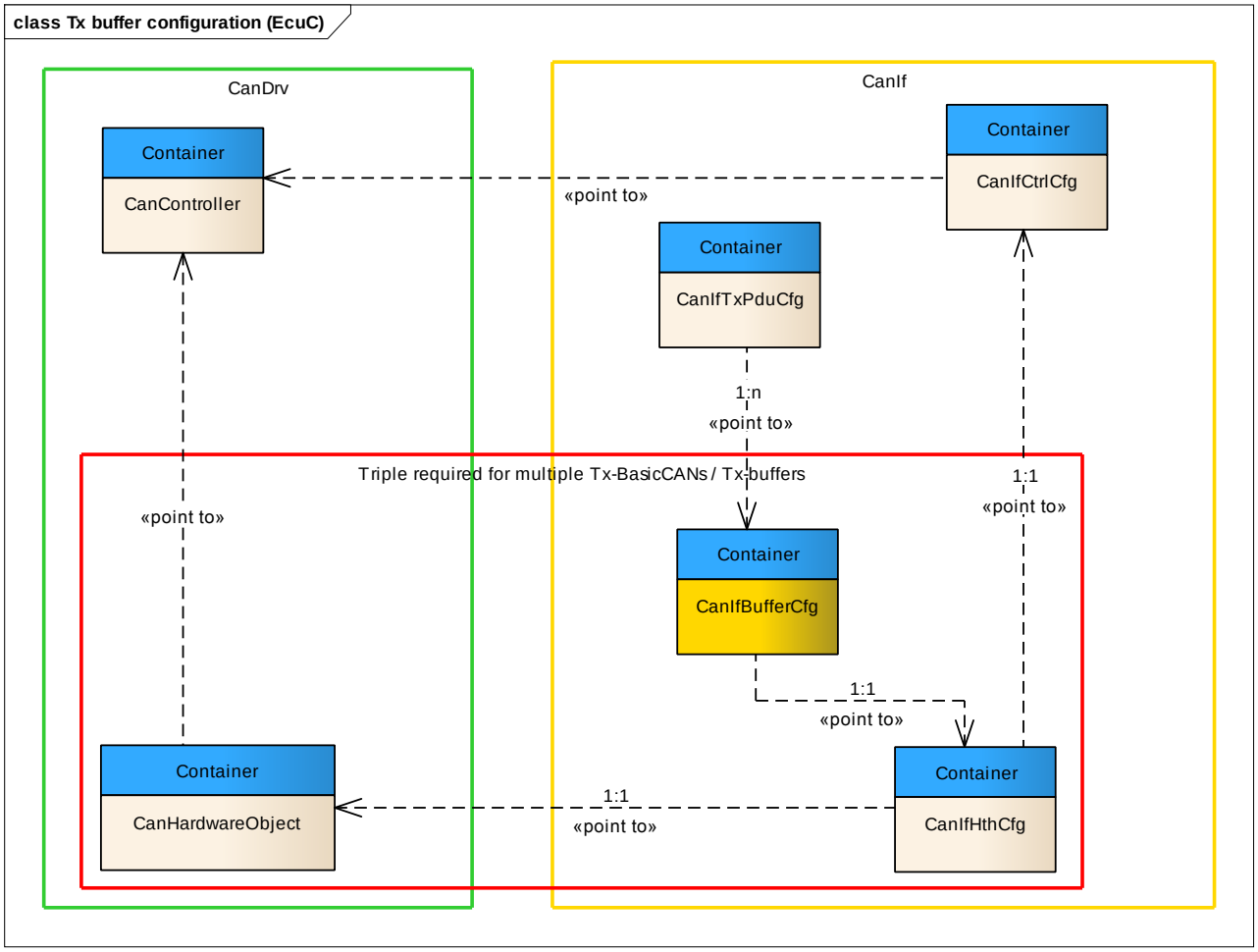

Figure 3 Configuration of multiple Transmit-buffers

After this step you can map Tx-PDUs to configured Transmit-buffer (object: `CanIfBufferCfg`). The described handling type of a transmit-buffer (see chapter 3.4.2) can be configured via the parameter `CanIfTxBufferHandlingType`. For further information about configuration of a Transmit-buffer please refer to the help which can be found in the GUI of the DaVinci Configurator 5 and to the descriptions of attributes of container `CanIfBufferCfg`.

3.4.4 Tx confirmation polling support

The CAN Interface supports a service which provides the information on whether any Tx confirmation has occurred for a CAN channel since the last start of that CAN channel at all. This feature can be enabled via the parameter `CanIfPublicTxConfirmPollingSupport`. If enabled the API `CanIf_GetTxConfirmation()` is provided and can be used for this service.

3.4.5 Data checksum Tx

This feature can be used to append a checksum to data of a Tx-PDU. The configuration of such Tx-PDU can be done individually via the parameter `CanIfTxPduDataChecksumPdu`. The appending of checksum is application specific and must be implemented within the API `CanIf_TransmitSubDataChecksumTxAppend()`. For further information please see the description of the prototype of this API in chapter 6.2.3.

For further information about configuration of this feature at all please refer to the help which can be found in the GUI of the DaVinci Configurator 5 and to the description of mentioned parameters.

3.5 Reception

Reception of PDUs is only possible in the states

- `CANIF_CS_STARTED` and `CANIF_GET_ONLINE`

or

- `CANIF_CS_STARTED` and `CANIF_GET_RX_ONLINE`.

In all other states the PDUs received by the CAN Driver are discarded by the CAN Interface without notification to the upper layers.

The CAN Interface supports reception of FullCAN- as well as BasicCAN-messages. The upper layers do not notice any differences between these two reception types as in both cases a call back function is called which was configured for the specified PDU in the generation tool.

The upper layer is notified about the PDU ID given by the corresponding upper layer at configuration time, the received data and depending on the used indication function about the length of the received data.

In case of BasicCAN reception the CAN Interface has to search through a list of all known Rx messages and compare the received CAN ID with the CAN ID in the Rx message list.

The CAN Interface offers three different search algorithms:

- **Linear search**: The list of all Rx PDUs is searched from high priority (Low CAN Identifier) to low priority (High CAN Identifier). This algorithm is efficient for a small amount of Rx messages.
- **Double Hash search**: The Rx PDU is calculated via two special hash functions. The algorithm is very efficient for a high amount of Rx messages and always takes the same time.

Note
The Double Hash search algorithm uses the mathematical operation modulo.

- **Binary search**: The list of Rx PDUs is split in two equal sized parts and the search is continued recursively on a list of PDUs which contains half the messages. This search algorithm terminates faster for big amounts of Rx messages than the linear search.

Caution
The binary search algorithm cannot be used for mixed ID systems.

3.5.1 Ranges

The BasicCAN message object can be used to receive groups of CAN messages called ranges. A range can be defined either by an upper and a lower CAN identifier or by a mask and a code.

The definition of a range by an upper and a lower CAN identifier is performed by the following parameters:

- `CanIfRxPduCanIdRangeLowerCanId` and
- `CanIfRxPduCanIdRangeUpperCanId`.

A mask-code-range is defined by parameters:

- `CanIfRxPduCanId` (code) and
- `CanIfRxPduCanIdMask` (mask).

In case of a mask-code-range each CAN identifier which fulfills the following equation pass the range and the reception of the corresponding Rx PDU is reported to the upper layer.

- `<CAN identifier> & <mask> == <code> & <mask>`

One PDU ID is assigned to all messages which pass the configured range. Hence the upper layer is not able to get additional message properties like the CAN identifier. For each range an indication function can be assigned in the generation tool in order to notify the higher layer about the reception of a message.

A range defined by an upper and a lower CAN identifier can be converted into a mask-code-range. Therefor please see the following example.

Example: How to convert a lower CAN ID and an upper CAN ID into mask and code?

Lower CAN ID: 0x400
Upper CAN ID: 0x43F
The code is same as the lower CAN ID:
code = 0x400

You need the count which is upper CAN ID – lower CAN ID → 0x43F – 0x400 = 0x3F
The count 0x3F is 000 0011 1111b in 11-bit binary format. For a range with extended CAN IDs the count needs to be 29-bit wide.

The mask is calculated out of negated count and a 11-bit mask:
mask = ~0x3F & 0x7FF = 0x7C0
For extended IDs you need a 29-bit mask:
mask = ~0x3F & 0x1FFF FFFF = 0x1FFF FFC0

Note:
If for count the first set bit is followed by unset bits on lower significant positions for the calculation of the mask these bits need to be set. For example a count of 0xA3 (1010 0011b) you need to calculate with the count 0xFF (1111 1111b). The consequence is that more CAN IDs are received as intended.

3.5.2 DLC check

The DLC check is executed for all received messages after they pass the search algorithm (PDU is in Rx list) or if they are defined to be received in FullCAN message objects. The feature DLC check can be activated only at Pre-compile time at all. If activated the DLC check can be configured for each Rx-PDU individually and can be reconfigured in the Post-build-loadable configuration phase.

The DLC check verifies if the received DLC is greater or equal to the DLC specified during configuration time. If the DLC is less than the configured one a DET error is raised and the reception of the PDU is abandoned.

3.5.3 Data checksum Rx

This feature can be used to verify the validity of a Rx-PDU after reception. The Rx-PDU which shall be verified can be configured individually via the parameter `CanIfRxPduDataChecksumPdu`. The verification is application specific and must be implemented within the API `CanIf_RxIndicationSubDataChecksumRxVerify()`. For further information please see the description of the prototype of this API in chapter 6.2.2.

In addition an indication function may be configured which signals about invalidity of a Rx-PDU. This indication function can be configured via the parameter `CanIfDispatchDataChecksumRxErrorIndicationName`. The call of this indication

function is application specific too and if required must be invoked within the implementation of `CanIf_RxIndicationSubDataChecksumRxVerify()`.

The prototype of the indication function must match following signature:

- `void My_DataChecksumRxErrFct (PduIdType CanIfRxPduId)`

and can be accessed via the macro: `CanIf_GetDataChecksumRxErrFctPtr()` (see file `CanIf_Cfg.h`)

It is recommended to call this indication function with the identifier of affected Rx-PDU. Therefor the value of parameter `CanIfRxPduId` should be used which is passed by call of `CanIf_RxIndicationSubDataChecksumRxVerify()`. The value of this parameter is a CAN interface internal identifier which corresponds to value of configuration parameter `CanIfRxPduId`. Corresponding macros are generated per Rx-PDU into file `CanIf_Cfg.h`. These ones can be used by application (s. example below).

```
/**********************************************************************************
  \def  AUTOSAR Rx PDU handles
**********************************************************************************/

#define CanIfConf_CanIfRxPduCfg_RxRange2_0                                    0U
#define CanIfConf_CanIfRxPduCfg_RxRange1_0                                    1U
#define CanIfConf_CanIfRxPduCfg_RxMSG00000711_0                               2U
#define CanIfConf_CanIfRxPduCfg_RxMSG95555311_0                               3U
#define CanIfConf_CanIfRxPduCfg_RxMSG00000511_0                               4U
#define CanIfConf_CanIfRxPduCfg_RxMSG91111151_0                               5U
```

For further information about configuration of this feature at all please refer to the help which can be found in the GUI of the DaVinci Configurator 5 and to the description of mentioned parameters.

3.5.4 Control of reception mode of a Rx-PDU

This feature provides the ability to control the reception mode of a Rx-PDU at runtime. The reception mode can be set per Rx-PDU individually at runtime via the API: `CanIf_SetPduReceptionMode()`. In order to address a Rx-PDU you can use the corresponding symbolic name value which can be found in file `CanIf_Cfg.h` (s. example below).

```
/**********************************************************************************
  \def  AUTOSAR Rx PDU handles
**********************************************************************************/

#define CanIfConf_CanIfRxPduCfg_RxRange2_0                                    0U
#define CanIfConf_CanIfRxPduCfg_RxRange1_0                                    1U
#define CanIfConf_CanIfRxPduCfg_RxMSG00000711_0                               2U
#define CanIfConf_CanIfRxPduCfg_RxMSG95555311_0                               3U
#define CanIfConf_CanIfRxPduCfg_RxMSG00000511_0                               4U
#define CanIfConf_CanIfRxPduCfg_RxMSG91111151_0                               5U
```

For further information about this API please see chapter 6.1.38. This feature can be used for e.g. either to receive a CAN-message as a Rx-PDU with an explicit CAN-identifier or as

a Rx-range-PDU. In case of the configured CAN-identifier of a Rx-PDU fits the range of CAN-identifiers of a Rx-range-PDU on the same CAN-channel as well. In case of a FullCAN-Rx-PDU the reception can be controlled at runtime at all.

This feature can be enabled via the parameter `CanIfSetPduReceptionModeSupport`. In addition Rx-PDUs whose reception mode is intended to be controlled at runtime must be configured accordingly via the parameter `CanIfRxPduSetReceptionModePdu`.

For further information about configuration of this feature at all please refer to the help which can be found in the GUI of the DaVinci Configurator 5 and to the description of mentioned parameters.

3.6 Communication Modes

The CAN Interface knows two main types of communication modes.

3.6.1 Controller Mode

The controller mode represents the physical state of the CAN controller. The following modes are available:

- `CANIF_CS_STOPPED`
- `CANIF_CS_STARTED`
- `CANIF_CS_SLEEP`
- `CANIF_CS_UNINIT`

There is no state called bus off. Bus off is treated as a transition from STARTED to STOPPED mode. All transitions have to be initiated using the API function `CanIf_SetControllerMode()`. The controller mode can be switched for each controller independent of the state of other controllers in the system.

The state `CANIF_CS_UNINIT` is left after `CanIf_InitController()` is called and can only be entered by a reset of the ECU.

The modes `CANIF_CS_SLEEP` and `CANIF_CS_STARTED` can only be entered from `CANIF_CS_STOPPED`. This means a transition from STARTED to SLEEP and vice versa is not possible without requesting the STOPPED mode first.

It is always possible to request the current active controller mode by calling the API `CanIf_GetControllerMode()`.

3.6.2 PDU Mode

The other type of communication mode is completely processed by software (it does not represent any state of the hardware). Transitions of the PDU mode are only possible if the controller mode is set to `CANIF_CS_STARTED`.

The following PDU modes are available:

- `CANIF_GET_OFFLINE`

  Rx and Tx path are switched offline

- `CANIF_GET_RX_ONLINE`

  Rx path online, Tx path offline

- `CANIF_GET_TX_ONLINE`

  Rx path offline, Tx path online

- `CANIF_GET_ONLINE`

  Rx and Tx path are switched online

- `CANIF_GET_OFFLINE_ACTIVE`

  Rx and Tx path offline, confirmation is emulated by the CAN Interface

- `CANIF_GET_OFFLINE_ACTIVE_RX_ONLINE`

  Rx path online, Tx path offline, confirmation is emulated by the CAN Interface

If parameter `CanIfPnWakeupTxPduFilterSupport` (s. chapter 3.16) is enabled then the following two further modes are available:

- `CANIF_GET_TX_ONLINE_WAKF`

  Rx path offline, tx path online

- `CANIF_GET_ONLINE_WAKF`

  Rx and Tx path are switched online

The difference to the modes `CANIF_GET_ONLINE` and `CANIF_GET_TX_ONLINE` is that the Tx-PDU filter is activated if the PDU mode is changed to one of these two modes. (s. chapter 3.16).

Caution
If one of the modes `CANIF_GET_TX_ONLINE_WAKF` or `CANIF_GET_ONLINE_WAKF` is left by calling of `CanIf_SetPduMode()` with parameter `PduModeRequest` which equals `CANIF_SET_OFFLINE` or `CANIF_SET_TX_OFFLINE` or `CANIF_SET_TX_OFFLINE_ACTIVE` or `CANIF_SET_ONLINE` or `CANIF_SET_TX_ONLINE` then the Tx-PDU Filter is **deactivated**!

The PDU modes can be set via the function `CanIf_SetPduMode()` and can be retrieved via the function `CanIf_GetPduMode()`.

3.7 Polling

The CAN Interface can process events in polling and interrupt mode. As the polling of events is executed by other layers (e.g. CAN Driver, Transceiver Driver) the CAN Interface is notified by call back functions which are called in the corresponding context.

Info
There is no need for changes in the configuration to run the CAN Interface in polling mode.

3.8 CAN FD

The CAN Interface supports CAN FD. The configuration can be performed both for Rx- and Tx-PDUs. Therefor please configure the attribute `CanIfRxPduCanIdType` (Rx-PDU) and `CanIfTxPduCanIdType` (Tx-PDU) accordingly as required by your application. In case of Rx-PDUs the message type (e.g. FD or not-FD) is evaluated during the Rx-search algorithm. Hence it is possible to handle two messages with the same CAN identifier, at which one is configured as FD and one as not-FD and to map them to different Rx-PDUs.

Expert Knowledge
If you intend to switch the baudrate of the CAN hardware at runtime it is suggested to use the API `CanIf_SetBaudrate` instead of `CanIf_ChangeBaudrate`.

Rx- and Tx-FD-PDUs with up to 64 bytes payload are supported.

Basic Knowledge
If you intend to configure BasicCAN-FD-Tx-PDUs and the Tx-buffer is enabled in your configuration please ensure that attribute `CanIfStaticFdTxBufferSupport` is enabled.

3.9 Meta data Rx- / Tx-support

If this feature is enabled the CAN Interface supports the handling of dynamic CAN-identifiers by using of SDU meta data. Such dynamic PDU can be configured by parameter `MetaDataLength`. This parameter can be found in the container of corresponding global PDU.

In case of configuration variant Link-time or Post-build loadable please enable this feature by setting of parameter `CanIfMetaDataSupport` to true. In case of configuration variant Pre-compile the activation/deactivation of this feature is determined from the configuration of Rx- and Tx-PDUs. If there is any PDU which has configured the parameter `MetaDataLength` then this feature is enabled else disabled.

3.10 J1939 dynamic address support

If this feature is enabled the CAN Interface translates the addresses (CAN identifiers) of Rx- and Tx-PDUs according to J1939 by using of dynamic address lookup tables. These tables are maintained by J1939Nm by using of following APIs:

- `CanIf_SetAddressTableEntry` and
- `CanIf_ResetAddressTableEntry`.

This feature has to be configured for each CAN channel individually by the parameter `CanIfCtrlJ1939DynAddrSupport`. Please consider that in case of configuration

variant Post-build loadable and configuration phase Post-build the value which you can select by `CanIfCtrlJ1939DynAddrSupport` is limited by value of `CanIfJ1939DynAddrSupport` which was set at configuration phase Pre-compile. Therefore in case of configuration variant Post-build loadable please first enable this feature as far as you need at all by the parameter `CanIfJ1939DynAddrSupport` and then configure the channel specific parameter of this feature. In case of configuration variant Pre-compile it is only possible to configure the channel specific parameter.

Caution
The feature `J1939 dynamic address support` works only if all Rx-PDUs of the CAN channel at which this feature is enabled are configured as BasicCANs and if all the corresponding hardware filters are opened completely!

3.11 Error Notification

AUTOSAR specifies two mechanisms of error notification and reporting. Only DET reporting is supported by the CAN Interface and can be activated at configuration time (Pre-compile configuration).

Development errors are reported to DET using the service `Det_ReportError()`.This feature is normally activated during the development phase to detect fatal errors in configuration and integration of the CAN Interface with other layers.

The reported CAN Interface ID is 60.

The reported service IDs identify the services which are described in chapter 6. The following table presents the service IDs and the related services:

Service ID	Service
1	`CanIf_Init`
2	`CanIf_InitController`
3	`CanIf_SetControllerMode`
4	`CanIf_GetControllerMode`
5	`CanIf_Transmit`
6	`CanIf_ReadRxPduData`
9	`CanIf_SetPduMode`
10	`CanIf_GetPduMode`
11	`CanIf_GetVersionInfo`
12	`CanIf_SetDynamicTxId`
13	`CanIf_SetTrcvMode`
14	`CanIf_GetTrcvMode`
15	`CanIf_GetTrcvWakeupReason`
16	`CanIf_SetTrcvWakeupMode`
17	`CanIf_CheckWakeup`
18	`CanIf_CheckValidation`

Service ID	Service
19	CanIf_TxConfirmation
20	CanIf_RxIndication
21	CanIf_CancelTxConfirmation
22	CanIf_ControllerBusoff
23	CanIf_ControllerModeIndication
24	CanIf_TrcvModeIndication
25	CanIf_GetTxConfirmationState
26	CanIf_ConfirmPnAvailability
27	CanIf_ChangeBaudrate
28	CanIf_CheckBaudrate
30	CanIf_ClearTrcvWufFlag
31	CanIf_CheckTrcvWakeFlag
32	CanIf_ClearTrcvWufFlagIndication
33	CanIf_CheckTrcvWakeFlagIndication
39	CanIf_SetBaudrate
246	CanIf_SetPduReceptionMode
247	CanIf_RamCheckEnableController
248	CanIf_RamCheckEnableMailbox
249	CanIf_RamCheckExecute
250	CanIf_CancelTransmit
251	CanIf_CancelTxNotification
252	CanIf_SetAddressTableEntry
253	CanIf_ResetAddressTableEntry

Table 3-2 Mapping of service IDs to services

The errors reported to DET are described in the following table:

Error Code		Description
10	CANIF_E_PARAM_CANID	Used in context of following functions if an invalid CAN identifier is passed: - CanIf_RxIndication - CanIf_SetDynamicTxId
11	CANIF_E_PARAM_DLC	Used in context of following functions if a PDU with invalid data length is passed: - CanIf_RxIndication - CanIf_Transmit - CanIf_CancelTxConfirmation
12	CANIF_E_PARAM_HRH	Used in context of following function if an invalid hardware receive handle is passed: - CanIf_RxIndication

Error Code		Description
13	CANIF_E_PARAM_LPDU	Used in context of following functions if an invalid Tx-PDU is passed: - CanIf_TxConfirmation - CanIf_CancelTxConfirmation - CanIf_CancelTxNotification
14	CANIF_E_PARAM_CONTROLLER	Used in context of following functions if an invalid CAN channel is passed: - CanIf_ControllerBusOff - CanIf_ControllerModeIndication - CanIf_GetTxConfirmationState - CanIf_SetTrcvMode - CanIf_GetTrcvMode - CanIf_GetTrcvWakeupReason - CanIf_SetTrcvWakeupMode - CanIf_TrcvModeIndication - CanIf_ConfirmPnAvailability - CanIf_ClearTrcvWufFlag - CanIf_ClearTrcvWufFlagIndication - CanIf_CheckTrcvWakeFlag - CanIf_CheckTrcvWakeFlagIndication
15	CANIF_E_PARAM_CONTROLLERID	Used in context of following functions if an invalid CAN channel is passed: - CanIf_SetControllerMode - CanIf_GetControllerMode - CanIf_SetPduMode - CanIf_GetPduMode - CanIf_CheckBaudrate - CanIf_ChangeBaudrate - CanIf_SetBaudrate - CanIf_SetAddressTableEntry - CanIf_ResetAddressTableEntry - CanIf_Transmit - CanIf_TxConfirmation - CanIf_CancelTxConfirmation - CanIf_CancelTransmit - CanIf_CheckWakeup - CanIf_RamCheckExecute - CanIf_RamCheckEnableMailbox - CanIf_RamCheckEnableController

Error Code		Description
16	CANIF_E_PARAM_WAKEUPSOURCE	Used in context of following functions if an invalid wakeup source is passed: - CanIf_CheckValidation - CanIf_CheckWakeup
17	CANIF_E_PARAM_TRCV	Used in context of following functions if an invalid transceiver channel is passed: - CanIf_TrcvModeIndication - CanIf_GetTrcvWakeupReason - CanIf_GetTrcvMode - CanIf_SetTrcvMode - CanIf_SetTrcvWakeupMode - CanIf_ConfirmPnAvailability - CanIf_ClearTrcvWufFlagIndication - CanIf_CheckTrcvWakeFlagIndication - CanIf_ClearTrcvWufFlag - CanIf_CheckTrcvWakeFlag - CanIf_CheckWakeup
18	CANIF_E_PARAM_TRCVMODE	Used in context of following function if an invalid transceiver mode is passed: - CanIf_SetTrcvMode
19	CANIF_E_PARAM_TRCVWAKEUPMODE	Used in context of following function if an invalid transceiver wakeup mode is passed: - CanIf_SetTrcvWakeupMode
20	CANIF_E_PARAM_POINTER	Used in context of following functions if an invalid pointer is passed: - CanIf_Init - CanIf_GetControllerMode - CanIf_Transmit - CanIf_RxIndication - CanIf_GetPduMode - CanIf_GetVersionInfo - CanIf_GetTrcvWakeupReason - CanIf_GetTrcvMode - CanIf_CancelTxConfirmation
21	CANIF_E_PARAM_CTRLMODE	Used in context of following function if an invalid CAN controller mode is passed: - CanIf_SetControllerMode
30	CANIF_E_UNINIT	Used in context of following functions if called before the CAN Interface is initialized: - CanIf_InitController - CanIf_Transmit - CanIf_TxConfirmation - CanIf_RxIndication

Error Code		Description
		- CanIf_ControllerBusOff - CanIf_SetPduMode - CanIf_GetPduMode - CanIf_CancelTxConfirmation - CanIf_CheckWakeup - CanIf_CheckValidation - CanIf_GetTrcvWakeupReason - CanIf_SetTrcvWakeupMode - CanIf_ControllerModeIndication - CanIf_SetDynamicTxId - CanIf_TrcvModeIndication - CanIf_SetControllerMode - CanIf_GetControllerMode - CanIf_CancelTxNotification - CanIf_SetTrcvMode - CanIf_GetTrcvMode - CanIf_CancelTransmit - CanIf_ConfirmPnAvailability - CanIf_ClearTrcvWufFlagIndication - CanIf_CheckTrcvWakeFlagIndication - CanIf_ClearTrcvWufFlag - CanIf_CheckTrcvWakeFlag - CanIf_GetTxConfirmationState - CanIf_CheckBaudrate - CanIf_ChangeBaudrate - CanIf_SetBaudrate - CanIf_SetPduReceptionMode - CanIf_SetAddressTableEntry - CanIf_ResetAddressTableEntry - CanIf_RamCheckExecute - CanIf_RamCheckEnableMailbox - CanIf_RamCheckEnableController - CanIf_SetPduReceptionMode
40	CANIF_E_NOK_NOSUPPORT	Not used.
44	CANIF_E_INVALID_PDURECEPTIONMODE	Used in context of following function if an invalid reception mode is passed: - CanIf_SetPduReceptionMode
50	CANIF_E_INVALID_TXPDUID	Used in context of following functions if an invalid Tx-PDU is passed: - CanIf_CancelTransmit - CanIf_SetDynamicTxId - CanIf_Transmit

Error Code		Description
		- CanIf_TxConfirmation
60	CANIF_E_INVALID_RXPDUID	Used in context of following function if an invalid Rx-PDU is passed: - CanIf_SetPduReceptionMode
61	CANIF_E_INVALID_DLC	Used in context of following function if the length of received PDU is invalid (smaller than the configured one): - CanIf_HlIndication
70	CANIF_E_STOPPED	Used in context of following function if it is called while either the controller mode is STOPPED or the PDU mode is OFFLINE: - CanIf_Transmit
71	CANIF_E_NOT_SLEEP	Used in context of following function if it is called while the CAN controller mode is neither in SLEEP nor in STOPPED. - CanIf_CheckWakeup
Additionally defined error codes (not AUTOSAR compliant)		
45	CANIF_E_CONFIG	Used to detect inconsistent data in the generated files due to misconfiguration. Used in context of following functions: - CanIf_RxIndication - CanIf_Transmit
46	CANIF_E_FATAL	Used to detect either an invalid (out of bounce) write access to a variable or an invalid read access to function pointer tables in order to prevent undefined behaviour at runtime. Used in context of following functions: - CanIf_Init - CanIf_Transmit - CanIf_CancelTxConfirmation - CanIf_CancelTransmit - CanIf_CancelTxNotification - CanIf_TxConfirmation
47	CANIF_E_INVALID_SA	Used in context of following functions if an invalid J1939 source address (SA) is determined: - CanIf_Transmit - CanIf_RxIndication
48	CANIF_E_INVALID_DA	Used in context of following functions if an invalid J1939 destination address (DA) is determined:

Error Code		Description
		- CanIf_Transmit - CanIf_RxIndication
49	CANIF_E_INVALID_CANIDTYPESIZE	Used in context of following function if the size [bytes] of type Can_IdType is inconsistent between static and generated code: - CanIf_Init
50	CANIF_E_INVALID_DLC_METADATA	Used in context of following function if a Rx-PDU of type: meta data is received with invalid length - CanIf_RxIndication
51	CANIF_E_FULL_TX_BUFFER_FIFO	Used to inform that the transmit-buffer of handling type FIFO is full and that no further Tx-PDUs can be buffered. Used in context of following function: - CanIf_Transmit
52	CANIF_E_INVALID_DOUBLEHASH_CALC	Used in context of following function if the calculated match via the double hash algorithm for a received CAN message is not in valid range: - CanIf_RxIndication

Table 3-3 Errors reported to DET

Caution
If the development error detection is disabled not only the reporting of the errors is suppressed but also the detection i.e. the verification of valid function parameters.

3.12 Transceiver handling

The CAN Interface provides APIs and call back functions to control as many transceivers as CAN controllers are available in the system. The transceiver handling has to be activated at pre-compile time.

The CAN Interface provides the following functions for higher layers to control the behavior of the transceiver.

- CanIf_SetTrcvMode()
- CanIf_TrcvModeIndication()
- CanIf_GetTrcvMode()
- CanIf_GetTrcvWakeupReason()
- CanIf_SetTrcvWakeupMode()

Additionally the following APIs are provided in order to control a partial networking CAN transceiver.

- `CanIf_CheckTrcvWakeFlag()`
- `CanIf_CheckTrcvWakeFlagIndication()`
- `CanIf_ClearTrcvWufFlag()`
- `CanIf_ClearTrcvWufFlagIndication()`
- `CanIf_ConfirmPnAvailability()`

The initialization of the transceiver driver itself is not executed by the CAN Interface. This means the calling layer has to make sure the transceiver driver is initialized before using the listed API functions.

If more than one different transceiver driver is used in the system the CAN Interface provides a mapping to address the correct transceiver driver with the correct parameters. The parameter `CanIfTransceiverMapping` has to be activated to control more than one transceiver driver.

It is also allowed to activate the parameter `CanIfTransceiverMapping` if only one transceiver driver is used in the system. Because of additional runtime it is suggested to deactivate this feature in this use case.

The CAN Interface supports the detection of wake up events raised by a transceiver. The feature “Wakeup Support” has to be activated and a wakeup source has to be configured for the corresponding transceiver channel.

Within the API `CanIf_CheckWakeup()` the CAN Interface analyses the passed wakeup source parameter and decides whether a CAN Controller or a CAN Transceiver has to be requested for a pending wake up event.

For more details refer to the chapter 3.13 Sleep / WakeUp.

3.13 Sleep / WakeUp

The CAN Interface controls the modes of the underlying CAN driver and transceiver driver.

The API `CanIf_SetControllerMode()` has to be used to change the mode of the CAN controller while the CAN transceiver can be controlled with the API `CanIf_SetTrcvMode()`.

Caution
The CAN Interface itself does not perform any checks whether the CAN controller and the CAN transceiver are set to sleep consistently and in the correct sequence. It is up to the higher layer to call `CanIf_SetControllerMode()` and `CanIf_SetTrcvMode()` in the correct sequence.

Wake up events can be raised either by the CAN controller or by the CAN transceiver. In both cases the CAN Interface is not directly informed about state changes. This means the higher layers (normally the EcuM) has to call the API `CanIf_CheckWakeup()` with the wakeup sources configured for CAN transceiver or CAN controller (1).

The CAN Interface decides by analyzing the passed wakeup source whether the CAN controller or the CAN transceiver driver has to be checked for a pending wakeup (2 or 2').

The following figure illustrates the described wake up sequence:

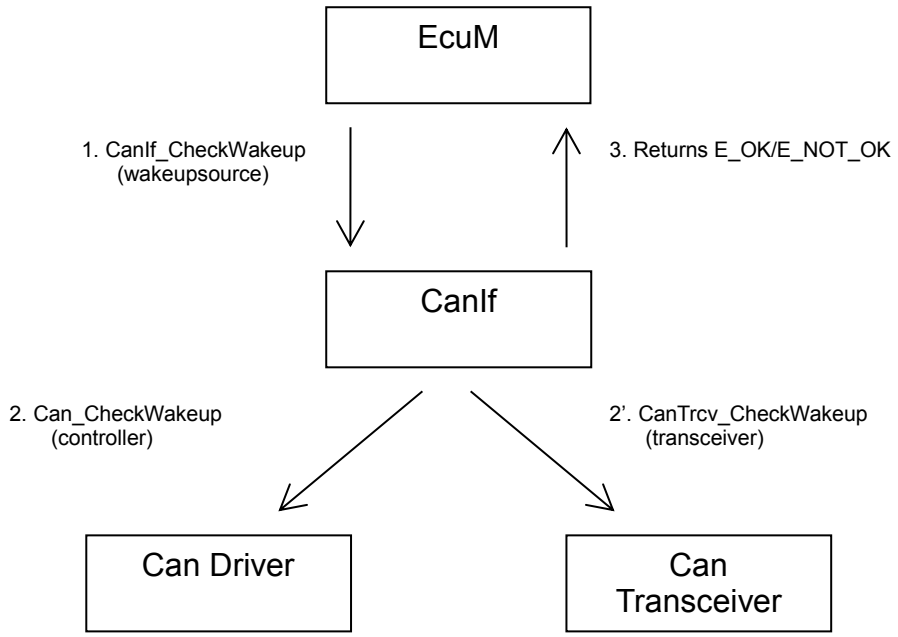

Figure 3-4 Wake up sequence (No validation)

If the parameter "CanIfPublicWakeupCheckValidSupport" is enabled the following figure shows the sequence which has to be executed for a valid wake up. Steps 1 to 3 take place as described above.

After the call of `EcuM_SetWakeupEvent()` the CAN Interface has to be set to the state `CANIF_CS_STARTED` to be able to receive messages. These messages won't be passed to upper layers by the CAN Interface because the PDU-mode is still set to `OFFLINE`. The state change which sets the CAN Interface to the mode `STARTED` has to be realized by the call of the API `CanIf_SetControllerMode()` with mode `CANIF_CS_STARTED` (5) from the function `EcuM_StartWakeupSources()` (4). If the wake up was detected by the transceiver the CAN controller has to be woken up internally. This means the call `CanIf_SetControllerMode()` with mode `CANIF_CS_STOPPED` is necessary in (5) before the transition to mode `STARTED` is executed.

If the wake up is initiated by the CAN controller the corresponding transceiver channel has to be set to mode `NORMAL` and the CAN controller has to be set to mode `STARTED`.

If the wake up is initiated by a transceiver channel the CAN controller has to be woken up internally. This means an additional call of `CanIf_SetControllerMode()` with mode `CANIF_CS_STOPPED` has to be executed to wake up the CAN controller before the transition to mode `STARTED` is initiated. (Depending on the behavior of the transceiver the

CAN controller and the configuration itself it is possible to wake up both the CAN controller and the transceiver channel externally.)

Next the EcuM starts a time out for the wake up validation. This means if a message is received within this timeout (6) the call of `CanIf_CheckValidation()` executed by the EcuM (7) will result in a successful validation. The CAN Interface checks for a recent Rx event (6) which occurred after the wake up and notifies the EcuM by calling of `EcuM_ValidationWakeupEvent()`.

If there is no message reception after (5) the function `CanIf_CheckValidation()` has been called no successful wake up validation won't be notified and the EcuM will run into a timeout. In this case the EcuM calls `EcuM_StopWakeupSources()` (8') and the CAN Driver and CAN transceiver have to be set to mode `SLEEP` again.

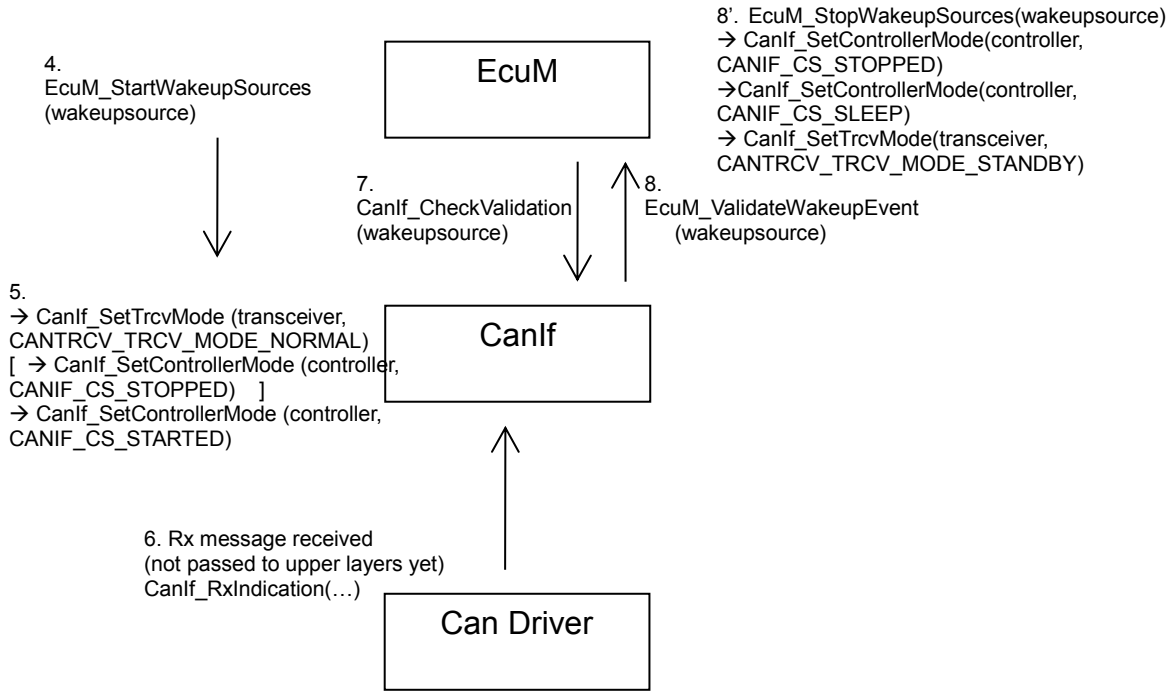

Figure 3-5 Wake up sequence (Wakeup validation)

During the wake up sequence as well as during the transition to mode `SLEEP`, the higher layers have to take care about the sequence of the state transitions affecting the CAN controller (CAN driver) and the Transceiver driver.

Since ASR4.0R3 it is configurable on whether only a received CanNm-message is able to do the validation.

3.14 Bus Off

The CAN Interface handles bus off events notified by the CAN Driver in interrupt driven or polling systems. If a bus off event is raised the CAN Driver forwards it to the CAN Interface by calling the function `CanIf_ControllerBusOff()`.

The CAN Interface switches its internal controller state from STARTED to STOPPED and the PDU mode is set to OFFLINE.

In this state no reception and no transmission is possible until the CAN Interface's controller state and as a result the CAN Controller's bus off state is recovered by the call of the function CanIf_SetControllerMode() for the affected channel by the higher layer.

After the controller state is switched the bus off state is recovered. For successful reception and transmission the PDU mode has to be switched to RX_ONLINE, TX_ONLINE or ONLINE by the higher layer.

3.15 Version Info

The version of the CAN Interface module can be acquired in three different ways. The first possibility is by calling of the function CanIf_GetVersionInfo(). This function returns the module's version in the structure Std_VersionInfoType which includes the VendorID and the ModuleID additionally.

The second possibility is the access of version defines which are specified in the header file CanIf.h.

The following defines can be evaluated to access different versions:

- **AUTOSAR version:**
  - CANIF_AR_RELEASE_MAJOR_VERSION
  - CANIF_AR_RELEASE_MINOR_VERSION
  - CANIF_AR_RELEASE_PATCH_VERSION
- **Module version:**
  - CANIF_SW_MAJOR_VERSION
  - CANIF_SW_MINOR_VERSION
  - CANIF_SW_PATCH_VERSION
- **Module ID:**
  - CANIF_MODULE_ID
- **Vendor ID:**
  - CANIF_VENDOR_ID

There is a third possibility to at least acquire the SW version by accessing globally visible constants:

- CanIf_MainVersion
- CanIf_SubVersion

- CanIf_ReleaseVersion

Info
The API CanIf_GetVersionInfo() is only available if enabled at Pre-compile time. The definitions can be accessed independent of the configuration.

3.16 Partial Networking

This feature consists of two sub-features:

- Wakeup Tx-PDU filter (parameter: CanIfPnWakeupTxPduFilterSupport)
- Handling of a partial networking transceiver (parameter: CanIfPnTrcvHandlingSupport)

The mentioned sub-features can be used only if the attribute CanIfPublicPnSupport is enabled. See the following table for more information about mentioned sub-features.

Feature	Description
CanIfPnWakeupTxPduFilterSupport	Tx-PDU filter which is activated if the PDU mode is changed either to CANIF_SET_ONLINE_WU_FILTER or to CANIF_SET_TX_ONLINE_WU_FILTER. This filter is active until the first Tx-confirmation / Rx-indication of the corresponding CAN channel arrives. Only certain Tx-PDUs which are labeled as Tx wakeup filter PDUs (s. parameter CanIfTxPduPnFilterPdu) can pass the filter. All Tx-requests of other Tx-PDUs are refused by CAN Interface until the filter is disabled.
CanIfPnTrcvHandlingSupport	Handling of a partial networking transceiver

Table 3-4 Sub-features of feature Partial Networking

The parameter CanIfPnTrcvHandlingSupport is enabled automatically if at least one underlying transceiver driver supports partial networking. In case of using the feature CanIfPnWakeupTxPduFilterSupport the Tx-PDUs which are allowed to pass the filter have to be configured accordingly. This kind of configuration can be performed individually for every Tx-PDU via the parameter CanIfTxPduPnFilterPdu.

Note
Please consider that the filter of a certain CAN channel is only active if at least one Tx-PDU of this CAN channel has the parameter CanIfTxPduPnFilterPdu enabled.

The feature `CanIfPnWakeupTxPduFilterSupport` is configurable in all three configuration variants:

- Pre-compile
- Link-time
- Post-build-loadable

Except the restriction that this feature has to be enabled at Pre-compile time at all there are no any further restrictions concerning the reconfiguration of this feature in accordance with the Tx-PDUs which may pass the filter in case of a Link-time or a Post-build-loadable configuration variant.

3.17 Services used by the CAN Interface

In the following table services provided by other components which are used by the CAN Interface are listed. For details about prototype and functionality refer to the documentation of the corresponding component.

Component	API
DET	`Det_ReportError`
CanDrv	`Can_SetControllerMode` `Can_Write`
PduR, CanNm, CanTp, CDD	`User_TxConfirmation (*)` `User_RxIndication (*)`
CanNm, EcuM, CDD	`User_ControllerBusOff (*)` `User_ValidationWakeupEvent (*)`
SchM	`SchM_Enter_CanIf_##area` `SchM_Exit_CanIf_##area`

CanTrcv	`CanTrcv_SetOpMode`
	`CanTrcv_GetOpMode`
	`CanTrcv_GetBusWuReason`
	`CanTrcv_SetWakeupMode`
	`CanTrcv_CheckWakeup`

MICROSAR extension (optional)	`EcuM_BswErrorHook`

Table 3-5 API functions used by the CAN Interface

* Names of the call back functions can be configured freely.

3.18 Multiple CAN drivers

The CAN Interface supports multiple CAN drivers which are implemented according to AUTOSAR specification 4.1.1.

Different CAN drivers are addressed by using the values of attributes "VendorId" and "VendorApiInfix" defined in BSWMD file of corresponding CAN driver.

In order to ensure compatibility with this CAN Interface the following naming convention of APIs of CAN driver need to be provided.

```
<Bsw>_<VendorId>_<VendorApiInfix>_<ApiName>
```

The APIs of used CAN driver has to be named as follows:

Basic CAN Driver APIs
Can_<VendorId>_<VendorApiInfix>_SetControllerMode
Can_<VendorId>_<VendorApiInfix>_Write
Can_<VendorId>_<VendorApiInfix>_CancelTx(*)
Can_<VendorId>_<VendorApiInfix>_CheckWakeup(*)
Can_<VendorId>_<VendorApiInfix>_CheckBaudrate(*)
Can_<VendorId>_<VendorApiInfix>_ChangeBaudrate(*)
Can_<VendorId>_<VendorApiInfix>_SetBaudrate(*)

Table 3-6 Adapted CAN driver APIs (* optional)

The following table lists APIs of CAN Interface which have to be called by a CAN driver in case of multiple CAN drivers are configured.

Basic CAN Driver APIs
CanIf_<VendorId>_<VendorApiInfix>_RxIndication
CanIf_<VendorId>_<VendorApiInfix>_TxConfirmation
CanIf_<VendorId>_<VendorApiInfix>_ControllerBusOff
CanIf_<VendorId>_<VendorApiInfix>_ControllerModeIndication
CanIf_<VendorId>_<VendorApiInfix>_CancelTxNotification
CanIf_<VendorId>_<VendorApiInfix>_CancelTxConfirmation

Table 3-7 APIs of CAN Interface which have to be used in multiple CAN driver configurations

Caution
In case of using of a CAN driver which is not provided by Vector Informatik please pay attention to chapter 7.2.4.

3.19 Extended RAM-check

This feature is configured via the parameter `CanIfExtendedRamCheckSupport`. For further information about configuration of this feature please refer to the help which can be found in the GUI of the DaVinci Configurator 5 and to the description of mentioned parameter.

3.20 Critical Sections

The AUTOSAR standard provides with the BSW Scheduler a BSW module, which handles entering and leaving critical sections.

For more information about the BSW Scheduler please refer to [3]. When the BSW Scheduler is used the CAN Interface provides critical section codes that have to be mapped by the BSW Scheduler to following mechanism:

Critical Section Define	Description
CANIF_EXCLUSIVE_AREA_0	Usage inside `CanIf_SetControllerMode()` > Duration is short (< 10 instructions) > Call to `Can_SetControllerMode()`
CANIF_EXCLUSIVE_AREA_1	Usage inside `CanIf_CancelTxConfirmation()`, `CanIf_CancelTransmit()`, `CanIf_ClearQueue()` > Duration is short (< 10 instructions). > No calls inside
CANIF_EXCLUSIVE_AREA_2	Usage inside `CanIf_TxConfirmation()` and `CanIf_CancelTxConfirmation()` > Duration is medium (< 50 instructions). > Call to `CanIf_TxQueueTreatment()`, `CanIf_TxQueueTransmit()`, `Can_Write()`, .
CANIF_EXCLUSIVE_AREA_3	Usage inside `CanIf_SetPduMode()` > Duration is short (< 10 instructions). > Call to `CanIf_ClearQueue()`
CANIF_EXCLUSIVE_AREA_4	Usage inside `CanIf_Transmit()` > Duration is medium (< 50 instructions). > Call to `Can_Write()`
CANIF_EXCLUSIVE_AREA_5	Usage inside `CanIf_SetDynamicTxId()` > Duration is short (< 10 instructions). > Setting of dynamic CAN identifier
CANIF_EXCLUSIVE_AREA_6	Usage inside `CanIf_SetAddressTableEntry()` and `CanIf_ResetAddressTableEntry()` > Duration is short (< 10 instructions). > Setting of J1939 Rx- and Tx-address > No calls inside

CANIF_EXCLUSIVE_AREA_7	Usage inside `CanIf_RxIndication()` > Duration is short (< 10 instructions). > Consistent reading from J1939 Rx-address table > No calls inside

Table 3-8 Critical Section Codes

If the exclusive areas are entered the upper layer needs to make sure that the CAN interrupts are disabled. Additionally the following table describes which API of the CAN Interface must not be called during the corresponding area is entered. The CAN Interface API `CanIf_CancelTxNotification()` / `CanIf_CancelTxConfirmation()` is entered mostly via the CAN interrupt. In case of a platform which confirmation for a transmit cancellation needs to be polled the corresponding API (for example `Can_MainFunction_Write()`) must not be called if the corresponding lock area is entered.

	CANIF_EXCLUSIVE_AREA_0	CANIF_EXCLUSIVE_AREA_1	CANIF_EXCLUSIVE_AREA_2	CANIF_EXCLUSIVE_AREA_3	CANIF_EXCLUSIVE_AREA_4	CANIF_EXCLUSIVE_AREA_5	CANIF_EXCLUSIVE_AREA_6	CANIF_EXCLUSIVE_AREA_7
CanIf_Init	■	■	■	■	■	■	■	■
CanIf_InitMemory	■							
CanIf_Transmit	■	■	■	■	■	■	■	
CanIf_CancelTransmit	■	■	■	■	■			
CanIf_SetControllerMode	■	■	■	■	■			
CanIf_CancelTxNotification/ CanIf_CancelTxConfirmation	■	■	■	■	■			
CanIf_SetPduMode	■	■	■	■	■			
CanIf_TxConfirmation	■	■	■	■	■			
CanIf_ControllerBusOff	■	■	■	■	■			
CanIf_RxIndication							■	
CanIf_SetAddressTableEntry					■		■	■
CanIf_ResetAddressTableEntry					■		■	■

Table 3-9 Restrictions for the different lock areas

4 Integration

This chapter gives necessary information for the integration of the MICROSAR CAN Interface into an application environment of an ECU.

4.1 Files and include structure

The CAN Interface consists of the following files:

The delivery of the CAN Interface contains the files which are described in the chapters 4.1.1 and 4.1.2:

4.1.1 Static Files

File Name	Description
`CanIf.c`	Implementation
`CanIf.h`	Header file, has to be included by higher layers to access the API
`CanIf_Cbk.h`	Header file, has to be included by underlying layers to access call back functions provided by the CAN Interface
`CanIf_Types.h`	Definition of types provided by the CAN Interface which have to be used by other layers. This file is included automatically if either `CanIf.h` or `CanIf_Cbk.h` is included.
`CanIf_Hooks.h`	This header file is included by `CanIf.c` and defines so called hook-macros. Every API of the CAN interface has an own pair of hook-macro. One of them is called at the beginning of each API and the other one at the end. The intention of these hook-macros is the ability to measure the execution time of an API. The hook-macros are defined to nothing by default. So they do not influence the execution of code by default.
`CanIf_GeneralTypes.h`	This header file is included by `Can_GeneralTypes.h` and contains the public types of the CAN Interface.

Table 4-1 Static files

4.1.2 Dynamic Files

The dynamic files are generated by the configuration tool.

File Name	Description
`CanIf_Cfg.h`	Generated header file (included automatically by `CanIf.h` and `CanIf_Cbk.h`)
`CanIf_Lcfg.c`	Contains link time configuration data. Contains data in case of Pre-compile, Link-time and Post-build configuration variant.
`CanIf_PBcfg.c`	Contains post build configuration data. In case of Link-time variant is used, this file is empty.
`CanIf_CanTrcv.h`	Generated header file which includes the necessary header files of the transceiver drivers used in the system.

Table 4-2 Generated files

4.2 Include Structure

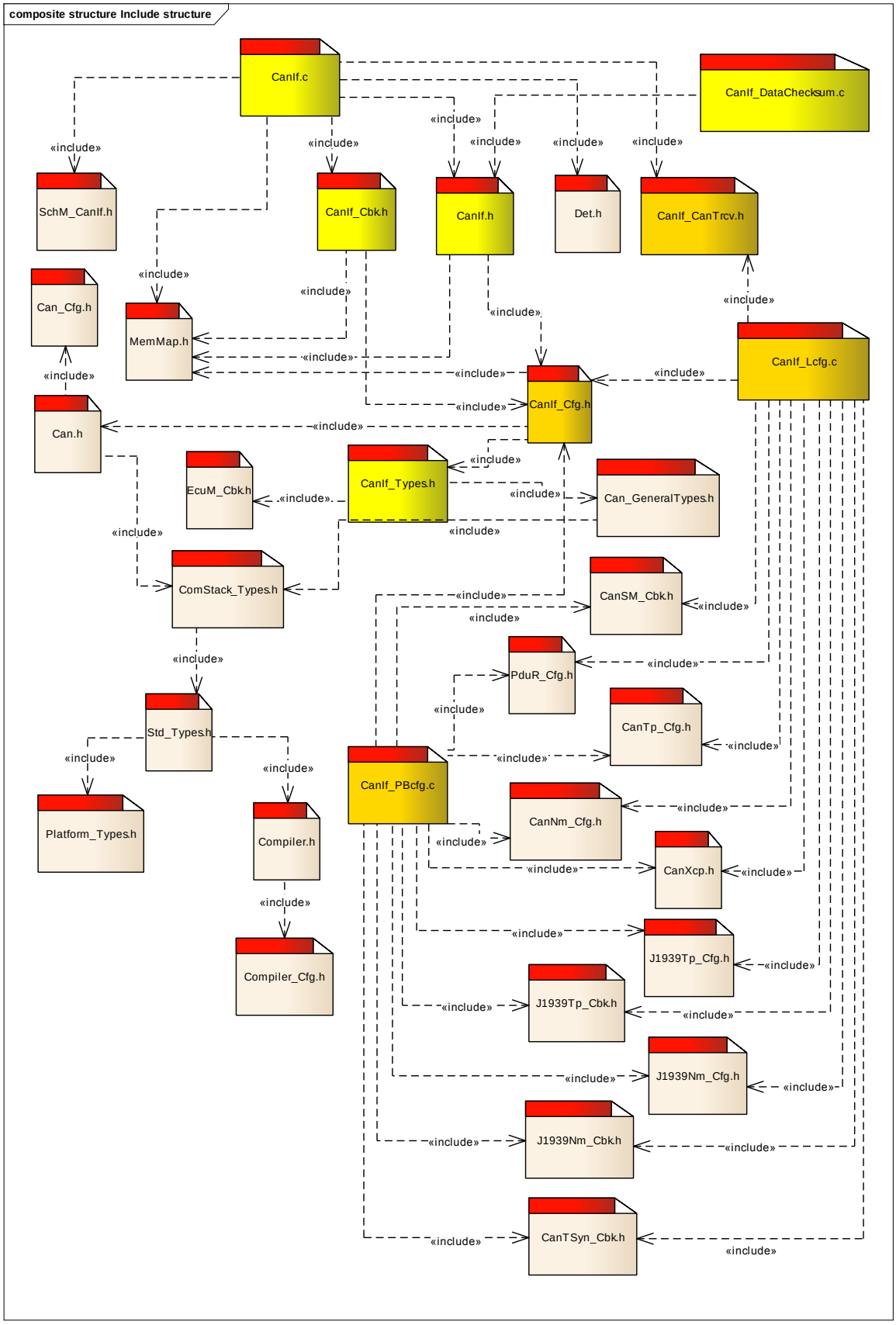

Figure 4-1 Include structure

4.3 Compiler Abstraction and Memory Mapping

The objects (e.g. variables, functions, constants) are declared by compiler independent definitions – the compiler abstraction definitions. Each compiler abstraction definition is assigned to a memory section.

The following table contains the memory section names and the compiler abstraction definitions defined for the CAN Interface and illustrates their assignment among each other.

Memory Mapping Sections \ Compiler Abstraction Definitions	CANIF_VAR_ZEROINIT	CANIF_VAR_INIT	CANIF_VAR_NOINIT	CANIF_CONST	CANIF_PBCFG	CANIF_CODE	CANIF_APPL_CODE	CANIF_APPL_VAR	CANIF_APPL_PBCFG	CANIF_VAR_PBCFG
CANIF_START_SEC_CODE CANIF_STOP_SEC_CODE						■				
CANIF_START_SEC_PBCFG CANIF_STOP_SEC_PBCFG					■					
CANIF_START_SEC_CONST_8BIT CANIF_STOP_SEC_CONST_8BIT				■						
CANIF_START_SEC_CONST_32BIT CANIF_STOP_SEC_CONST_32BIT				■						
CANIF_START_SEC_CONST_UNSPECIFIED CANIF_STOP_SEC_CONST_UNSPECIFIED				■						
CANIF_START_SEC_VAR_NOINIT_UNSPECIFIED CANIF_STOP_SEC_VAR_NOINIT_UNSPECIFIED		■								
CANIF_START_SEC_VAR_ZERO_INIT_UNSPECIFIED CANIF_STOP_SEC_VAR_ZERO_INIT_UNSPECIFIED		■								
CANIF_START_SEC_VAR_INIT_UNSPECIFIED CANIF_STOP_SEC_VAR_INIT_UNSPECIFIED		■								
CANIF_START_SEC_VAR_PBCFG CANIF_STOP_SEC_VAR_PBCFG										■

Table 4-3 Compiler abstraction and memory mapping

The Compiler Abstraction Definitions `CANIF_APPL_CODE`, `CANIF_APPL_VAR` and `CANIF_APPL_PBCFG` are used to address code, variables and constants which are declared by other modules and used by the CAN Interface.

These definitions are not mapped by the CAN Interface but by the memory mapping realized in the CAN Driver, CAN Transceiver Driver, PDU Router, Network management, Transport Protocol Layer, ECU State Manager and the CAN State manager.

5 Configuration

The CAN Interface is configured with DaVinci Configurator 5. Please refer to the help which can be found in the GUI of the configurator and to the descriptions of attributes in BSWMD file of CAN Interface.

5.1 Configuration of Post-Build

The configuration of post-build loadable is described in `TechnicalReference_PostBuildLoadable.pdf`.

6 API Description

6.1 Services provided by the CAN Interface

6.1.1 CanIf_GetVersionInfo

Prototype	
`void CanIf_GetVersionInfo( Std_VersionInfoType *VersionInfo );`	
Parameter	
`VersionInfo`	Pointer to the structure including the version information.
Return code	
-	-
Functional Description	
CanIf_GetVersionInfo() returns version information, vendor ID and AUTOSAR module ID of the component. The versions are BCD-coded.	
Particularities and Limitations	
The function is only available if enabled at Pre-compile time (`CANIF_VERSION_INFO_API = STD_ON`)	

Table 6-1 API CanIf_GetVersionInfo

6.1.2 CanIf_Init

Prototype	
`void CanIf_Init( const CanIf_ConfigType *ConfigPtr )`	
Parameter	
`ConfigPtr`	Pointer to the structure including configuration data.
Return code	
-	-
Functional Description	
This function initializes global CAN Interface variables during ECU start-up.	
Particularities and Limitations	
This API has to be called during start-up before any CAN communication. `Can_Init()` has to be executed successfully.	

Table 6-2 API CanIf_Init

6.1.3 CanIf_SetControllerMode

Prototype	
`Std_ReturnType CanIf_SetControllerMode(uint8 ControllerId, CanIf_ControllerModeType ControllerMode)`	
Parameter	
`ControllerId` `ControllerMode`	The Controller to change mode. Mode request.
Return code	
`Std_ReturnType`	Returns whether the state transition was successful.
Functional Description	
Request the mode of the specified channel. Supported modes: `CANIF_CS_SLEEP`, `CANIF_CS_STOPPED`, `CANIF_CS_STARTED`.	
Particularities and Limitations	
CAN Interface has to be initialized.	

Table 6-3 API CanIf_SetControllerMode

6.1.4 CanIf_GetControllerMode

Prototype	
`Std_ReturnType CanIf_GetControllerMode(uint8 ControllerId, CanIf_ControllerModeType *ControllerModePtr)`	
Parameter	
`ControllerId` `ControllerModePtr`	Request mode of specified Controller. Pointer to data type the information is stored in.
Return code	
`Std_ReturnType`	Returns whether the state request was successful or not.
Functional Description	
Acquire the current controller mode of the specified channel	
Particularities and Limitations	
CAN Interface has to be initialized.	

Table 6-4 API CanIf_GetControllerMode

6.1.5 CanIf_Transmit

Prototype	
`Std_ReturnType CanIf_Transmit(PduIdType CanTxPduId, const PduInfoType *PduInfoPtr)`	
Parameter	
`CanTxPduId` `PduIndoPtr`	Handle of the Tx PDU which will be transmitted. Pointer to a struct containing the properties of the Tx PDU.
Return code	
`Std_ReturnType`	Returns if the transmit request was accepted.
Functional Description	
Requests the transmission of the specified Tx PDU.	
Particularities and Limitations	
CAN Interface has to be initialized.	

Table 6-5 API CanIf_Transmit

6.1.6 CanIf_TxConfirmation

Prototype	
`void CanIf_TxConfirmation(PduIdType CanTxPduId)`	
Parameter	
`CanTxPduId`	ID of the successfully transmitted PDU.
Return code	
-	-
Functional Description	
Confirms the successful transmission of a Tx PDU	
Particularities and Limitations	
CAN Interface has to be initialized.	

Table 6-6 API CanIf_TxConfirmation

6.1.7 CanIf_RxIndication

Prototype	
`void CanIf_RxIndication(CanIf_HwHandleType Hrh, Can_IdType CanId, uint8 CanDlc, const uint8 *CanSduPtr)`	
Parameter	
`Hrh`	Hardware handle the PDU was received in.
`CanId`	CAN identifier of the received PDU.
`CanDlc`	Data length code of the received PDU.
`CanSduPtr`	Pointer to hardware or temporary buffer containing the data of the received PDU.
Return code	
-	-
Functional Description	
The CAN Driver notifies the CAN Interface about a received Rx PDU.	
Particularities and Limitations	
CAN Interface has to be initialized.	

Table 6-7 API CanIf_RxIndication

6.1.8 CanIf_ControllerBusOff

Prototype	
`void CanIf_ControllerBusOff(uint8 Controller)`	
Parameter	
`Controller`	Affected controller.
Return code	
-	-
Functional Description	
Indicates a BusOff for the specified controller to the CAN Interface.	
Particularities and Limitations	
CAN Interface has to be initialized.	

Table 6-8 API CanIf_ControllerBusOff

6.1.9 CanIf_SetPduMode

Prototype	
`Std_ReturnType CanIf_SetPduMode(uint8 ControllerId, CanIf_PduSetModeType PduModeRequest)`	
Parameter	
`ControllerId` `PduModeRequest`	Controller which will be affected by the new Pdu mode. Requested Pdu mode
Return code	
`Std_ReturnType`	Returns whether the state request was successful.
Functional Description	

Change mode for specified controller. Possible states are:
`CANIF_SET_OFFLINE,`
`CANIF_SET_RX_OFFLINE,`
`CANIF_SET_RX_ONLINE,`
`CANIF_SET_TX_OFFLINE,`
`CANIF_SET_TX_ONLINE,`
`CANIF_SET_ONLINE,`
`CANIF_SET_TX_OFFLINE_ACTIVE`	

Particularities and Limitations	
CAN Interface has to be initialized. Controller has to be in state `CANIF_CS_STARTED`.	

Table 6-9 API CanIf_SetPduMode

6.1.10 CanIf_GetPduMode

Prototype	
`Std_ReturnType CanIf_GetPduMode(uint8 ControllerId, CanIf_PduGetModeType * PduModePtr)`	
Parameter	
`ControllerId` `PduModePtr`	Request mode of the specified Controller. Pointer to a data buffer the current mode will be stored in.
Return code	
`Std_ReturnType`	Returns whether the request of the current state was successful.
Functional Description	
Request the current mode of the specified controller.	
Particularities and Limitations	
CAN Interface has to be initialized.	

Table 6-10 API CanIf_GetPduMode

6.1.11 CanIf_InitMemory

Prototype	
`void CanIf_InitMemory(void)`	
Parameter	
–	-
Return code	
-	-
Functional Description	
Initializes global RAM variables, which have to be available before any call to the CanIf API.	
Particularities and Limitations	
May only be called once before `CanIf_Init()`.	

Table 6-11 API CanIf_InitMemory

6.1.12 CanIf_CancelTxConfirmation

Prototype	
`void CanIf_CancelTxconfirmation(PduIdType CanTxPduId, const Can_PduType *PduInfoPtr)`	
Parameter	
`CanTxPduId` `PduInfoPtr`	Handle of the Tx PDU which was cancelled. Contains information about cancelled PDU
Return code	
-	-
Functional Description	
Called by the CAN Driver to notify the CAN Interface about a cancelled PDU which has to be re-queued.	
Particularities and Limitations	
Only available if `CANIF_TRANSMIT_CANCELLATION = STD_ON` is set.	

Table 6-12 API CanIf_CancelTxConfirmation

6.1.13 CanIf_SetTrcvMode

Prototype	
`StdReturnType CanIf_SetTrcvMode(uint8 TransceiverId, CanTrcv_TrcvModeType TransceiverMode)`	
Parameter	
`TransceiverId` `TransceiverMode`	Address the transceiver by a transceiver index. Requested mode transition
Return code	
`Std_ReturnType`	Returns whether the state transition was successful.
Functional Description	
Called by an upper layer to set the transceiver to another mode.	
Particularities and Limitations	
Only available if transceiver handling is activated at configuration time. (`CANIF_TRCV_HANDLING = STD_ON`)	

Table 6-13 API CanIf_SetTrcvMode

6.1.14 CanIf_GetTrcvMode

Prototype	
`StdReturnType CanIf_GetTrcvMode(CanTrcv_TrcvModeType *TransceiverModePtr, uint8 TransceiverId)`	
Parameter	
`TransceiverId` `TransceiverModePtr`	Address the transceiver by a transceiver index. Pointer to a buffer where current transceiver mode can be stored in.
Return code	
`Std_ReturnType`	Returns whether the request of the current transceiver mode was successful.
Functional Description	
Called by an upper layer to request the current mode of the transceiver.	
Particularities and Limitations	
Only available if transceiver handling is activated at configuration time. (`CANIF_TRCV_HANDLING = STD_ON`)	

Table 6-14 API CanIf_GetTrcvMode

6.1.15 CanIf_GetTrcvWakeupReason

Prototype	
`StdReturnType CanIf_GetTrcvWakeupReason(uint8 TransceiverId, CanIf_TrcvWakeupReasonType *TrcvWuReasonPtr)`	
Parameter	
`TransceiverId`	Address the transceiver by a transceiver index.
`TrcvWuReasonPtr`	Pointer to a buffer where the transceiver's wake up reason can be stored in.
Return code	
`Std_ReturnType`	Returns whether the request of the wake up reason was successful.
Functional Description	
Called by an upper layer to request the wake up reason stored in the transceiver.	
Particularities and Limitations	
Only available if transceiver handling is activated at configuration time. (`CANIF_TRCV_HANDLING = STD_ON`)	

Table 6-15 API CanIf_GetTrcvWakeupReason

6.1.16 CanIf_SetTrcvWakeupMode

Prototype	
`StdReturnType CanIf_SetTrcvWakeupMode(uint8 TransceiverId, CanTrcv_TrcvWakeupModeType TrcvWakeupMode)`	
Parameter	
`TransceiverId`	Address the transceiver by a transceiver index.
`TrcvWakeupMode`	Enable, disable or clear notification for wake up events.
Return code	
`Std_ReturnType`	Returns whether the requested mode was set successfully.
Functional Description	
Called by an upper layer to enable, disable or clear the wake up event notification of the transceiver.	
Particularities and Limitations	
Only available if transceiver handling is activated at configuration time. (`CANIF_TRCV_HANDLING = STD_ON`)	

Table 6-16 API CanIf_SetTrcvWakeupMode

6.1.17 CanIf_CheckWakeup

Prototype	
`Std_ReturnType CanIf_CheckWakeup(EcuM_WakeupSourceType WakeupSource)`	
Parameter	
`WakeupSource`	Wakeup source which identifies the possible wakeup source (Transceiver / CAN Controller)
Return code	
`Std_ReturnType`	Returns whether the request to the Transceiver/ CAN Controller was successful.
Functional Description	
Called by an upper layer to check if a transceiver or CAN controller recently raised a wakeup.	
Particularities and Limitations	
CAN Interface has to be initialized.	

Table 6-17 API CanIf_CheckWakeup

6.1.18 CanIf_CheckValidation

Prototype	
`Std_ReturnType CanIf_CheckValidation(EcuM_WakeupSourceType WakeupSource)`	
Parameter	
`WakeupSource`	Wakeup source which identifies the possible wakeup source (Transceiver / CAN Controller)
Return code	
`Std_ReturnType`	Returns whether the requested mode was set successfully.
Functional Description	
Called by an upper layer to check if a Rx message was received after a wake up occurred from one of the supported sources. If a message was received between the call of `CanIf_CheckWakeup` and `CanIf_CheckValidation` the configurable EcuM call back function `EcuM_ValidationWakeupEvent` is called from the context of this function.	
Particularities and Limitations	
CAN Interface has to be initialized. CanIf_CheckWakeup has to be called before and a wake up event has to be detected. CAN Interface has to be set to `CANIF_CS_STARTED` mode before a validation is possible.	

Table 6-18 API CanIf_CheckValidation

6.1.19 CanIf_CancelTransmit

Prototype	
`void CanIf_CancelTransmit (PduIdType CanTxPduId)`	
Parameter	
`CanTxPduId`	PduId of the message which has to be cancelled
Return code	
-	-
Functional Description	
Initiates the cancellation / suppression of the confirmation of a Tx message.	
Particularities and Limitations	
CAN Interface has to be initialized. AUTOSAR only defines a dummy function. For MICROSAR this function has the functionality to cancel an ordered Tx PDU. This API is provided only in case of `CANIF_CANCEL_SUPPORT_API = STD_ON`.	

Table 6-19 API CanIf_CancelTransmit

6.1.20 CanIf_CancelTxNotification

Prototype	
`void CanIf_CancelTxNotification (PduIdType PduId, CanIf_CancelResultType IsCancelled)`	
Parameter	
`PduId` `IsCancelled`	Id of the Tx message which was cancelled Parameter currently not evaluated.
Return code	
-	-
Functional Description	
Called by the CAN Driver to notify about a cancelled message. No confirmation is raised for this message.	
Particularities and Limitations	
CAN Interface has to be initialized. Non-AUTOSAR compliant API function which is enabled in case of `CANIF_CANCEL_SUPPORT_API = STD_ON`.	

Table 6-20 API CanIf_CancelTxNotification

6.1.21 CanIf_SetDynamicTxId

Prototype	
`void CanIf_SetDynamicTxId(PduIdType CanTxPduId, Can_IdType CanId)`	
Parameter	
`CanTxPduId` `CanId`	PDU ID of the Tx message CAN ID of the messageParameter
Return code	
-	-
Functional Description	
Called by the application to set the CAN Id of the corresponding Tx PDU.	
Particularities and Limitations	
CAN Interface has to be initialized. Shall not be interrupted by a call of `CanIf_Transmit()` for the same Tx PDU.	

Table 6-21 API CanIf_SetDynamicTxId

6.1.22 CanIf_ControllerModeIndication

Prototype	
`void CanIf_ControllerModeIndication(uint8 Controller, CanIf_ControllerModeType ControllerMode)`	
Parameter	
`Controller` `ControllerMode`	Channel where the mode transition happened Controller mode to which the CAN controller transitioned
Return code	
-	-
Functional Description	
Called by the CAN driver to notify about successful controller mode transition	
Particularities and Limitations	
CAN Interface has to be initialized.	

Table 6-22 API CanIf_ControllerModeIndication

6.1.23 CanIf_TrcvModeIndication

Prototype	
`void CanIf_TrcvModeIndication(uint8 TransceiverId, CanTrcv_TrcvModeType TransceiverMode)`	
Parameter	
`TransceiverId` `TransceiverMode`	Transceiver where the mode transition happened Transceiver mode to which the transceiver transitioned
Return code	
-	-
Functional Description	
Called by the transceiver driver to notify about successful transceiver mode transition	
Particularities and Limitations	
CAN Interface has to be initialized.	

Table 6-23 API CanIf_TrcvModeIndication

6.1.24 CanIf_ConfirmPnAvailability

Prototype	
`void CanIf_ConfirmPnAvailability(uint8 TransceiverId)`	
Parameter	
`TransceiverId`	`CAN transceiver, which was checked for PN availability`
Return code	
-	-
Functional Description	
This service indicates that the transceiver is running in PN communication mode	
Particularities and Limitations	
This API is only available in case of `CANIF_PN_TRCV_HANDLING = STD_ON`.	

Table 6-24 API CanIf_ConfirmPnAvailability

6.1.25 CanIf_ClearTrcvWufFlagIndication

Prototype	
`void CanIf_ClearTrcvWufFlagIndication(uint8 TransceiverId)`	
Parameter	
`TransceiverId`	CAN transceiver, for which the API was called
Return code	
-	-
Functional Description	
This service indicates that the transceiver has cleared the WufFlag.	
Particularities and Limitations	
`CanIf_Init()` has already been called and all transceiver driver have been initialized. This API is only available in case of `CANIF_PN_TRCV_HANDLING = STD_ON`.	

Table 6-25 API CanIf_ClearTrcvWufFlagIndication

6.1.26 CanIf_CheckTrcvWakeFlagIndication

Prototype	
`void CanIf_CheckTrcvWakeFlagIndication(uint8 TransceiverId)`	
Parameter	
`TransceiverId`	CAN transceiver, for which the API was called
Return code	
-	-
Functional Description	
This service indicates the reason for the wake up that the CAN transceiver has detected	
`CanIf_Init()` has already been called and all transceiver driver have been initialized. This API is only available in case of `CANIF_PN_TRCV_HANDLING = STD_ON`.	

Table 6-26 API CanIf_CheckTrcvWakeFlagIndication

6.1.27 CanIf_SetBaudrate

Prototype	
`Std_ReturnType CanIf_SetBaudrate(uint8 ControllerId, uint16 BaudRateConfigID)`	
Parameter	
`ControllerId` `BaudRateConfigID`	Abstracted CanIf ControllerId which is assigned to a CAN References a baud rate configuration by ID
Return code	
`E_OK` `E_NOT_OK`	Service request accepted, baudrate change started. Service request not accepted.
Functional Description	
This service shall set the baud rate configuration of the CAN controller.	
Particularities and Limitations	
CAN Interface has to be initialized. This API is provided in case of `CANIF_SET_BAUDRATE_API = STD_ON.`	

Table 6-27 API CanIf_SetBaudrate

6.1.28 CanIf_ChangeBaudrate

Prototype	
`Std_ReturnType CanIf_ChangeBaudrate(uint8 ControllerId, const uint16 Baudrate)`	
Parameter	
`ControllerId` `Baudrate`	The Controller the Baudrate shall be changed for Baudrate to which shall be changed
Return code	
`E_OK` `E_NOT_OK`	Service request accepted, change started Service request not accepted
Functional Description	
This service changes the baudrate of the CAN controller	
Particularities and Limitations	
CAN Interface has to be initialized. This API is provided in case of `CANIF_CHANGE_BAUDRATE_SUPPORT = STD_ON.`	

Table 6-28 API CanIf_ChangeBaudrate

6.1.29 CanIf_ChangeBaudrate

Prototype	
`Std_ReturnType CanIf_ChangeBaudrate(uint8 ControllerId, const uint16 Baudrate)`	
Parameter	
`ControllerId` `Baudrate`	The Controller the Baudrate shall be changed for Baudrate to which shall be changed
Return code	
`E_OK` `E_NOT_OK`	Service request accepted, change started Service request not accepted
Functional Description	
This service changes the baudrate of the CAN controller	
Particularities and Limitations	
CAN Interface has to be initialized. This API is provided in case of `CANIF_CHANGE_BAUDRATE_SUPPORT = STD_ON`.	

Table 6-29 API CanIf_ChangeBaudrate

6.1.30 CanIf_GetTxConfirmationState

Prototype	
`CanIf_NotifStatusType CanIf_GetTxConfirmationState (uint8 ControllerId)`	
Parameter	
`ControllerId`	Controller to be checked
Return code	
`CANIF_NO_NOTIFICATION` `CANIF_TX_RX_NOTIFICATION`	No transmit event occurred for requested CAN Controller The CAN Controller has successfully transmitted any message
Functional Description	
This service reports, if any TX confirmation has been done for the whole CAN controller since the last CAN controller start.	
Particularities and Limitations	
CAN Interface has to be initialized. This API is provided in case of `CANIF_PUBLIC_TX_CONFIRM_POLLING_SUPPORT = STD_ON`.	

Table 6-30 API CanIf_GetTxConfirmationState

6.1.31 CanIf_SetAddressTableEntry

Prototype	
`void CanIf_SetAddressTableEntry (uint8 ControllerId, uint8 intAddr, uint8 busAddr)`	
Parameter	

`ControllerId`
`intAddr`
`busAddr`
	The channel at which a J1939 address shall be set.
	J1939 internal address.
	J1939 bus address.

Return code	
–	-
Functional Description	
The service will be called to describe the relation between internal and external ID. Only used in J1939 environment.	
Particularities and Limitations	
CAN Interface has to be initialized. This API is provided in case of `CANIF_J1939_DYN_ADDR_SUPPORT != CANIF_J1939_DYN_ADDR_DISABLED`.	

Table 6-31 API CanIf_SetAddressTableEntry

6.1.32 CanIf_ResetAddressTableEntry

Prototype	
`void CanIf_ResetAddressTableEntry (uint8 ControllerId, uint8 intAddr)`	
Parameter	
`ControllerId` `intAddr`	The channel at which a J1939 internal address shall be reset. J1939 internal address.
Return code	
–	-
Functional Description	
The service will be called to reset the relation between internal and external ID. Only used in J1939 environment.	
Particularities and Limitations	
CAN Interface has to be initialized. This API is provided in case of `CANIF_J1939_DYN_ADDR_SUPPORT != CANIF_J1939_DYN_ADDR_DISABLED`.	

Table 6-32 API CanIf_ResetAddressTableEntry

6.1.33 CanIf_RamCheckExecute

Prototype	
`void CanIf_RamCheckExecute (uint8 ControllerId)`	
Parameter	
`ControllerId`	The CAN-channel for which the RAM-check shall be executed.
Return code	
–	-
Functional Description	
This service requests an underlying CAN-channel to execute the RAM-check of CAN-controller-HW-registers.	
Particularities and Limitations	
CAN Interface has to be initialized. This API is provided in case of `CANIF_EXTENDED_RAM_CHECK_SUPPORT == STD_ON`.	

Table 6-33 API CanIf_RamCheckExecute

6.1.34 CanIf_RamCheckEnableMailbox

Prototype	
`void CanIf_RamCheckEnableMailbox (uint8 ControllerId, CanIf_HwHandleType HwHandle)`	
Parameter	
`ControllerId` `HwHandle`	The CAN-channel to which the mailbox (`<HwHandle>`) belongs to. The mailbox which shall be enabled.
Return code	
–	-
Functional Description	
This service requests an underlying CAN-channel to enable a mailbox.	
Particularities and Limitations	
CAN Interface has to be initialized. This API is provided in case of `CANIF_EXTENDED_RAM_CHECK_SUPPORT == STD_ON`.	

Table 6-34 API CanIf_RamCheckEnableMailbox

6.1.35 CanIf_RamCheckEnableController

Prototype	
`void CanIf_RamCheckEnableController (uint8 ControllerId)`	
Parameter	
`ControllerId`	The CAN-channel which shall be enabled.
Return code	
–	-
Functional Description	
This service requests to enable an underlying CAN-channel.	
Particularities and Limitations	
CAN Interface has to be initialized. This API is provided in case of `CANIF_EXTENDED_RAM_CHECK_SUPPORT == STD_ON`.	

Table 6-35 API CanIf_RamCheckEnableController

6.1.36 CanIf_RamCheckCorruptMailbox

Prototype	
`void CanIf_RamCheckCorruptMailbox (uint8 ControllerId, CanIf_HwHandleType HwHandle)`	
Parameter	
`ControllerId` `HwHandle`	The CAN-channel to which the corrupt mailbox (`<HwHandle>`) belongs to. The corrupt mailbox.
Return code	
–	-
Functional Description	
This service indicates about a corrupt mailbox.	
Particularities and Limitations	
This service may be used also if CAN Interface is NOT initialized. This API is provided in case of `CANIF_EXTENDED_RAM_CHECK_SUPPORT == STD_ON`.	

Table 6-36 API CanIf_RamCheckCorruptMailbox

6.1.37 CanIf_RamCheckCorruptController

Prototype	
`void CanIf_RamCheckCorruptController (uint8 ControllerId)`	
Parameter	
`ControllerId`	The corrupt CAN-channel.
Return code	
-	-
Functional Description	
This service indicates about a corrupt CAN-channel.	
Particularities and Limitations	
This service may be used also if CAN Interface is NOT initialized. This API is provided in case of `CANIF_EXTENDED_RAM_CHECK_SUPPORT == STD_ON`.	

Table 6-37 API CanIf_RamCheckCorruptController

6.1.38 CanIf_SetPduReceptionMode

Prototype	
`Std_ReturnType CanIf_SetPduReceptionMode (PduIdType id, CanIf_ReceptionModeType mode)`	
Parameter	
`id`	The handle of Rx-PDU whose reception mode shall be changed.
`mode`	The reception mode which shall be set. Following reception modes are possible: `1) CANIF_RMT_IGNORE_CONTINUE`: In case of a match the received Rx-PDU is not forwarded to configured upper layer and the search for a potential match continues. `2) CANIF_RMT_RECEIVE_STOP`: In case of a match the received Rx-PDU is forwarded to configured upper layer.
Return code	
`E_OK` `E_NOT_OK`	Service request accepted, reception mode was changed Service request not accepted, reception mode was not changed
Functional Description	
Via this API the reception mode of a Rx-PDU can be set.	
Particularities and Limitations	
CAN Interface has to be initialized. During the initialization the reception mode of all affected Rx-PDUs is set to `CANIF_RMT_RECEIVE_STOP`. This API is provided in case of `CANIF_SET_PDU_RECEPTION_MODE_SUPPORT == STD_ON`.	

Table 6-38 API CanIf_SetPduReceptionMode

6.2 Callout Functions

6.2.1 EcuM_BswErrorHook

Prototype	
`void EcuM_BswErrorHook(uint16 CanIfModuleId, uint8 CanIfInstanceId)`	
Parameter	
`CanIfModuleId`	Contains the `CANIF_MODULE_ID` (60) as defined by AUTOSAR.
`CanIfInstanceId`	For the CanIf only one instance is available, so this value is always zero.
Return code	
None	
Functional Description	
Called once by the CanIf during the initialization phase to indicate one of the following possible errors: - Abort initialization as generator is not compatible	
Particularities and Limitations	
None	
Call Context	
This function is called in context of `CanIf_Init()`.	

Table 6-39 EcuM_BswErrorHook

6.2.2 CanIf_RxIndicationSubDataChecksumRxVerify

Prototype	
`Std_ReturnType CanIf_RxIndicationSubDataChecksumRxVerify (PduIdType CanIfRxPduId, Can_IdType CanId, uint8 CanDlc, const uint8 *CanSduPtr)`	
Parameter	
`CanIfRxPduId`	CanIf-internal unique handle ID of Rx-PDU
`CanId`	CAN identifier of received Rx-PDU
`CanDlc`	Data length of received Rx-PDU
`CanSduPtr`	Pointer to data of received Rx-PDU
Return code	
`E_OK`	Verification of checksum passed. In this case the Rx-PDU is forwarded to upper layer.
`E_NOT_OK`	Verification of checksum failed. In this case the Rx-PDU is discarded and NOT forwarded to upper layer.
Functional Description	
API called by CanIf in case of a data checksum PDU was received in order to verify its correctness.	
Particularities and Limitations	
This API is called only if `CANIF_DATA_CHECKSUM_RX_SUPPORT == STD_ON`.	
Call Context	
This function is called in context of `CanIf_RxIndication()`.	

6.2.3 CanIf_TransmitSubDataChecksumTxAppend

Prototype	
`void CanIf_TransmitSubDataChecksumTxAppend (const Can_PduType *txPduInfoPtr, uint8 *txPduDataWithChecksumPtr)`	
Parameter	
`txPduInfoPtr`	Pointer to Tx-PDU-parameters: CAN identifier, data length, data.
`txPduDataWithChecksumPtr`	Pointer to data buffer where the data of Tx-PDU incl. the checksum shall be stored in. The data checksum PDU is transmitted with data stored in this buffer. Note: Parameter "txPduDataWithChecksumPtr" may only be written with index >= 0 and < CANIF_CFG_MAXTXDLC_PLUS_DATACHECKSUM (see file CanIf_Cfg.h). The length of data can not be changed hence the checksum must only be added within valid data-length of the Tx-PDU which is given by range: 0 - (txPduInfoPtr->length - 1).
Return code	
None	
Functional Description	
API called by CanIf before transmission of a data checksum Tx-PDU in order to add a checksum to its data.	
Particularities and Limitations	
This API is called only if `CANIF_DATA_CHECKSUM_TX_SUPPORT == STD_ON`.	
Call Context	
This function is called in context of `CanIf_Transmit()`.	

7 AUTOSAR Standard Compliance

Following restrictions apply to the current CAN Interface implementation.

7.1 Not supported AUTOSAR features

The following features which are specified by the AUTOSAR CAN Interface SWS ([1]) are not supported.

7.1.1 Tx notification status

This feature is specified by the requirements: CANIF202, CANIF393, CANIF472, CANIF331, CANIF391, CANIF334, CANIF335, CANIF609_Conf and CANIF589_Conf.

7.1.2 Rx notification status

This feature is specified by the requirements: CANIF230, CANIF336, CANIF339, CANIF340, CANIF392, CANIF394, CANIF473, CANIF595_Conf and CANIF608_Conf.

7.1.3 Rx buffer

This feature is specified by the requirements: CANIF194, CANIF198, CANIF199, CANIF324, CANIF325, CANIF326, CANIF330, CANIF329, CANIF600_Conf and CANIF607_Conf.

7.2 Deviations

7.2.1 Tx buffer

At least and at most one Tx buffer is supported per each BasicCAN-Tx-PDU. Hence no configuration can be performed by the user as intended by the attribute `CanIfBufferSize`.

7.2.2 Partial networking

Against the requirement CANIF749 the Partial Networking Wakeup Tx Pdu Filter is enabled only if the PDU mode of CAN Interface is set either to mode `CANIF_GET_TX_ONLINE_WU_FILTER` or to mode `CANIF_GET_ONLINE_WU_FILTER`.

7.2.3 AUTOSAR version check

The CAN Interface does not perform AUTOSAR release version check in accordance with other modules because the version check is not specified by AUTOSAR clearly.

7.2.4 Check wakeup

According to AUTOSAR the API `Can_CheckWakeup` must have the following signature:

> `Can_ReturnType Can_CheckWakeup(uint8 Controller)` [2]

and must return the values `CAN_OK` or `CAN_NOT_OK`.

However the CAN Interface supports only the following signature:

> `Std_ReturnType Can_CheckWakeup(uint8 Controller)`

and supports only the return values `E_OK` and `E_NOT_OK` to be returned by this API.
Affected requirements are: CANIF720, CANIF678.

Note
Please ignore this deviation if you use a CAN driver provided by Vector Informatik. In this case both CAN driver and CAN interface are compatible and there is no malfunction in this regard.

Furthermore you may ignore this deviation and you will not have any malfunction too if:

1) you use a CAN driver which is not provided by Vector Informatik but the value of `CAN_OK` equals to `E_OK`

or

2) you do not evaluate the return value of API `CanIf_CheckWakeup` in your application at all.

7.2.5 Usage of a CAN driver implemented according to AUTOSAR 4.2.2

This CAN interface is implemented basing on AUTOSAR 4.0.3 specification. However this CAN interface can be used in conjunction with a CAN driver which is implemented according to AUTOSAR 4.2.2 specification as well. Therefor please consider the following aspects in order to avoid malfunction.

7.2.5.1 API: CanIf_RxIndication()

The prototype of `CanIf_RxIndication()` was changed in AUTOSAR 4.2.2. In case of you are using a CAN driver which is implemented according to AUTOSAR 4.2.2 then please set the configuration parameter `CanIfRxIndicationPrototype` to `CANIF_RX_IND_ASR422`. Otherwise you have not to configure this parameter at all.

7.2.5.2 HW cancellation

The API `CanIf_CancelTxConfirmation()` was removed in AUTOSAR 4.2.2 and by the way the feature HW cancellation at all. Hence if you use a CAN driver which does not support this feature please ensure that the parameter `CanIfCtrlDrvTxCancellation` is disabled in your configuration.

[2] Defined by requirement CAN360.

7.2.5.3 SW cancellation

The API `CanIf_CancelTransmit()` is according to the AUTOSAR specification just a dummy API which does not include any functionality. This CAN interface supports this service in real under the condition that the CAN driver supports this service as well. In this case this CAN interface expects the API `Can_CancelTx()` to be provided by the underlying CAN driver. If your CAN driver does not provide this API then please ensure that the configuration parameter `CanIfPublicCancelTransmitSupport` is disabled in your configuration. However if your configuration requires this parameter to be enabled at all but your CAN driver does not provide this API then please create an empty stub which matches the following signature:

- `void Can_CancelTx (Can_HwHandleType Hth, PduIdType PduId)`

and embed it via a user configuration file [Parameter: `CanIfUserConfigFile`].

7.2.5.4 Pretended networking

Pretended networking is a new feature in AUTOSAR 4.2.2. This feature is not supported by this CAN interface and hence cannot be used at all.

7.2.5.5 Trigger transmit

Trigger transmit is a new feature in AUTOSAR 4.2.2. This feature is not supported by this CAN interface and hence cannot be used at all.

7.3 Limitations

The priority of a dynamic Tx-PDU is determined from the initial configured CAN identifier and not from the CAN identifier set by using the API `CanIf_SetDynamicTxId()`.

8 Glossary and Abbreviations

8.1 Glossary

Term	Description
DaVinci Configurator 5	Configuration and generation tool for MICROSAR software components

Table 8-1 Glossary

8.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
CanNm	CAN Network Manager
CanSM	CAN State Manager
CanTp	CAN Transport Protocol
CanTrcv	CAN Transceiver
CCMSM	CAN Interface Controller Mode State Machine (for one controller)
CDD	Complex Device Driver
ComM	Communication Manager
DEM	Diagnostic Event Manager
DET	Development Error Tracer
DLC	Data Length Code
ECU	Electronic Control Unit
EcuM	ECU State Manager
FD	Flexible Data-rate
FIFO	First In First Out
HRH	Hardware Receive Handle
HTH	Hardware Transmit Handle
HW	Hardware
ISR	Interrupt Service Routine
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
PDU	Protocol Data Unit
PduR	PDU Router
SchM	Schedule Manager
SDU	Service Data Unit
SRS	Software Requirement Specification
SWC	Software Component

SWS	Software Specification

Table 8-2 Abbreviations

9 Contact

Visit our website for more information on

- News
- Products
- Demo software
- Support
- Training data
- Addresses

www.vector-informatik.com