

MICROSAR CAN Transceiver Driver

Technical Reference

Generic

Version 3.02.00

Authors	Matthias Fleischmann, Senol Cendere, Mihai Olariu, Timo Vanoni
Status	Released

1 Document Information

1.1 History

Author	Date	Version	Remarks
Matthias Fleischmann	2008-04-21	1.00	Creation
Senol Cendere	2008-06-10	1.01	Rework file names Rework tool description
Mihai Olariu	2008-07-10	1.02	Rename the placeholders
Mihai Olariu	2008-10-13	1.03	Minor changes in the GENy GUI
Mihai Olariu	2008-12-15	1.04	Add support for the platforms which cannot wakeup by CAN bus activity
Matthias Fleischmann	2009-07-01	1.05	Updated description for ICU notification function and GENy configuration. Added description for BSWMD file configuration.
Timo Vanoni	2009-11-12	1.06	Added API CanTrcv_30_<Your_Trcv>_Wait() Filenames were renamed to match BSW00347. Change of initialization flow.
Timo Vanoni	2010-05-06	1.06.1	Fixed example in chapter 4.5
Timo Vanoni	2011-01-26	1.07.00	Added support for Identity Manager Configurations
Timo Vanoni	2011-07-19	2.00.00	Implementation according ASR3.2.1
Timo Vanoni	2011-12-06	2.01.00	Add support for selective wakeup (partial networking) Add support for SPI Interface
Timo Vanoni	2012-08-22	3.00.00	Add support for SetPNActivationState API (ch. 6.1.16) Change wakeup detection of CB_WakeupByBus to conform to ASR3.2.2 (ch.6.1.8)
Timo Vanoni	2012-09-28	3.01.00	Add support for ASR4.0.3
Timo Vanoni	2013-04-12	3.02.00	Small reworks, add note in chapter 4.4

Table 1-1 History of the document

1.2 Reference Documents

No.	Title	Version
[1]	AUTOSAR_SWS_CAN_TransceiverDriver.pdf	2.0.0 ¹ , 3.0.0
[2]	AUTOSAR_SWS_DET.pdf	2.2.1
[3]	AUTOSAR_SWS_DEM.pdf	2.2.0
[4]	AUTOSAR_BasicSoftwareModules.pdf	V1.0.0

Table 1-2 Reference documents

1.3 Scope of the Document

This technical reference describes the specific use of the Generic CAN transceiver driver. Note that the substrings “__Your_Trcv__” and “__YOUR_TRCV__” are just placeholders for the real name of the CAN transceiver (e.g. Tja1041 and TJA1041).



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.

¹ From R3.2 Rev 2

Contents

1	Document Information	2
1.1	History	2
1.2	Reference Documents	3
1.3	Scope of the Document.....	3
2	Introduction.....	8
3	Functional Description	9
3.1	Features	9
3.2	Initialization	9
3.3	Set operation mode.....	10
3.4	Get operation mode	10
3.5	Get version info.....	10
3.6	Wakeup by bus event detection	10
3.6.1	Get bus wakeup reason	11
3.6.2	Set wakeup mode	11
3.6.3	Development Error Reporting.....	11
3.6.4	Production Code Error Reporting	12
4	Integration.....	13
4.1	Scope of Delivery.....	13
4.1.1	Static Files	13
4.1.2	Dynamic Files	13
4.2	Compiler Abstraction and Memory Mapping.....	14
4.3	Data consistency.....	15
4.4	Integration.....	15
4.5	Timers.....	15
4.6	Generated Data	17
4.6.1	CanTrcv_30___Your_Trcv___Channel.....	17
4.6.2	Partial networking data.....	18
4.6.2.1	CanTrcv_30___Your_Trcv___PnData.....	18
4.6.2.2	CanTrcv_30___Your_Trcv___PnActivationState.....	18
4.7	Handling of asynchronous APIs	19
4.7.1	Request handling	20
4.7.1.1	Concurrent operation modes.....	20
4.7.1.2	Request to CheckWakeFlag / ClearTrcvWufFlag / CB_WakeupByBus	22
4.7.1.3	Status Flag handling	22
4.7.1.4	Indication handling	23

5	Dependencies to other components	24
5.1	Dio driver	24
5.2	SPI driver	24
5.2.1	SPI Configuration	24
5.3	Icu driver	25
5.3.1	ICU configuration	25
5.3.2	Implementation of the signal notification function	27
6	API Description	28
6.1	Services provided by CANTRCV	28
6.1.1	CanTrcv_30__Your_Trcv__InitMemory	28
6.1.2	CanTrcv_30__Your_Trcv__Init	29
6.1.3	CanTrcv_30__Your_Trcv__SetOpMode	30
6.1.4	CanTrcv_30__Your_Trcv__GetOpMode	31
6.1.5	CanTrcv_30__Your_Trcv__GetBusWuReason	32
6.1.6	CanTrcv_30__Your_Trcv__SetWakeupMode	33
6.1.7	CanTrcv_30__Your_Trcv__GetVersionInfo	34
6.1.8	CanTrcv_30__Your_Trcv__CB_WakeupByBus	35
6.1.9	CanTrcv_30__Your_Trcv__GetTrcvSystemData	36
6.1.10	CanTrcv_30__Your_Trcv__ClearTrcvWufFlag	37
6.1.11	CanTrcv_30__Your_Trcv__ReadTrcvTimeoutFlag	38
6.1.12	CanTrcv_30__Your_Trcv__ClearTrcvTimeoutFlag	38
6.1.13	CanTrcv_30__Your_Trcv__ReadTrcvSilenceFlag	39
6.1.14	CanTrcv_30__Your_Trcv__CheckWakeFlag	40
6.1.15	CanTrcv_30__Your_Trcv__MainFunction	41
6.1.16	CanTrcv_30__Your_Trcv__SetPNActivationState	42
6.2	Services used by CANTRCV	43
7	Configuration	44
7.1	Configuration with GENy	44
7.1.1	Component Selection	44
7.1.2	General settings	44
7.1.3	Channel Specific Configuration Options	46
7.2	Configuration with DaVinci Configurator 5	47
8	AUTOSAR Standard Compliance	48
8.1	Additions/ Extensions	48
8.1.1	Memory initialization	48
8.2	Deviations	48
8.2.1	Notification functions	48
8.2.2	CanIf_CheckTrcvWakeFlagIndication is always called	48

8.2.3	No re-initialization will occur if POR is set in CanTrcv_SetOpMode..	48
8.2.4	Const removed from API CanTrcv_GetTrcvSystemData	49
8.2.5	Unused BSWMD parameters	49
8.2.6	Modified return value of CanTrcv_CB_WakeupByBus.....	49
9	Glossary and Abbreviations	50
9.1	Glossary	50
9.2	Abbreviations	50
10	Contact.....	51

Illustrations

Figure 5-1	Add an ICU channel.....	25
Figure 5-2	ICU channel configuration for wakeup via transceiver.....	26
Figure 5-3	ICU wakeup capability	26
Figure 7-1	CANTRCV component selection.....	44
Figure 7-2	CANTRCV general configuration	44
Figure 7-3	CANTRCV channel dependent configuration	46

Tables

Table 1-1	History of the document.....	2
Table 1-2	Reference documents.....	3
Table 3-1	Supported SWS features	9
Table 3-2	Not supported SWS features	9
Table 3-3	Mapping of service IDs to services	11
Table 3-4	Errors reported to DET	12
Table 3-5	Errors reported to DEM.....	12
Table 4-1	Static files	13
Table 4-2	Generated files	13
Table 4-3	Compiler abstraction and memory mapping.....	14
Table 4-4	Timer indexes and their wait times.....	16
Table 6-1	CanTrcv_30__Your_Trcv__InitMemory	28
Table 6-2	CanTrcv_30__Your_Trcv__Init.....	29
Table 6-3	CanTrcv_30__Your_Trcv__SetOpMode.....	30
Table 6-4	CanTrcv_30__Your_Trcv__GetOpMode	31
Table 6-5	CanTrcv_30__Your_Trcv__GetBusWuReason	32
Table 6-6	CanTrcv_30__Your_Trcv__SetWakeupMode.....	33
Table 6-7	CanTrcv_30__Your_Trcv__GetVersionInfo	34
Table 6-8	CanTrcv_30__Your_Trcv__CB_WakeupByBus.....	35
Table 6-9	CanTrcv_30__Your_Trcv__GetTrcvSystemData	36
Table 6-10	CanTrcv_30__Your_Trcv__ClearTrcvWufFlag	37
Table 6-11	CanTrcv_30__Your_Trcv__ReadTrcvTimeoutFlag.....	38
Table 6-12	CanTrcv_30__Your_Trcv__ClearTrcvTimeoutFlag.....	38
Table 6-13	CanTrcv_30__Your_Trcv__ReadTrcvSilenceFlag	39
Table 6-14	CanTrcv_30__Your_Trcv__CheckWakeFlag.....	40
Table 6-15	CanTrcv_30__Your_Trcv__MainFunction	41
Table 6-16	CanTrcv_30__Your_Trcv__SetPNActivationState.....	42
Table 6-17	Services used by the CANTRCV	43

Table 8-1	Deviation of APIs used by CanTrcv.....	48
Table 9-1	Glossary	50
Table 9-2	Abbreviations.....	50

2 Introduction

This document describes the functionality, API and configuration of the AUTOSAR BSW module CANTRCV as specified in [1]. The CAN transceiver driver provides an abstraction layer for the used CAN transceiver hardware. It offers a hardware independent interface to the upper layer components.

Supported AUTOSAR Release*:	3* and 4.0.3	
Supported Configuration Variants:	Pre-compile	
Vendor ID:	CANTRCV_30__YOUR_TRCV__VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	CANTRCV_30__YOUR_TRCV__MODULE_ID	[ModuleID] (according to ref. [4])
Instance ID:	CANTRCV_30__YOUR_TRCV__INSTANCE_ID	Specified as pre-compile parameter in the Generation Tool.

* For the precise AUTOSAR Release 3.x please see the release specific documentation.



Info

Replace the placeholders `__Your_Trcv__` and `__YOUR_TRCV__` with the according name of the used transceiver. This must be done in the source code and the BSWMD files.

`__YOUR_TRCV__` is used for definitions in upper case (e.g. TJA1041).

`__Your_Trcv__` is used for variables in camel case (e.g. Tja1041).

3 Functional Description

3.1 Features

The features listed in this chapter cover the complete functionality specified in [1].

The supported and not supported features are presented in the following two tables.

Supported features
CAN Transceiver initialization.
CAN Transceiver control via DIO.
Detection of wakeup (optional: incl. partial networking)
Getting and setting operation mode of CAN transceiver.
CAN Transceiver control via SPI. (optional)
CAN Transceiver self-diagnostics. (optional)

Table 3-1 Supported SWS features

Not supported features
-

Table 3-2 Not supported SWS features

The CAN transceiver driver provides service functions for initialization, operation mode change and operation mode detection of the used CAN transceiver hardware. Optional service functions and callback functions are provided to detect wakeup by bus events and report them to the upper layer components.

3.2 Initialization

After power on the CAN transceiver hardware has to be initialized. Therefore the CAN transceiver driver provides two service functions.

The function `VcIgXkT(TTTNd gTIgXkTTT>c iB bdgn` initializes all necessary values for the transceiver driver. This function has to be called first after power on or reset. The function `VcIgXkT(TTTNd gTIgXkTTT>c i` initializes all CAN transceiver channels which are selected by the generation tool. Each CAN transceiver is switched into the configured channel specific init mode. Operation mode after power on or reset can be normal, standby or sleep. `VcIgXkT(TTTNd gTIgXkTTT>c iB bdgn` has to be called before this function is called.

If a wakeup event was pending before the initialization the CAN transceiver driver stores this event and notifies it with the `VcIgXkT(TTTNd gTIgXkTTT TLV e n h` function.

3.3 Set operation mode

The operation mode of the CAN transceiver hardware is changed by service function

`VcIgXkT(TTTNd gTIgXkTTTH iDeBdY` . It can be switched from normal into standby mode, from standby into sleep mode and from standby or sleep mode into normal mode. Mode change from normal into sleep mode or from sleep into standby mode is not supported by this function corresponding to the specification.

3.4 Get operation mode

To retrieve the current operation mode of a specified CAN transceiver hardware, the service function `VcIgXkT(TTTNd gTIgXkTTT< iDeBdY` has to be called.

3.5 Get version info

The service function `VcIgXkT(TTTNd gTIgXkTTT< iK gh dc>c[d` can be called to get the version info of the software module. This function must be enabled in the configuration tool by setting the checkbox **Version Info Api**.

The version of the CAN transceiver driver module can be acquired in two different ways. Calling the function `VcIgXkT(TTTNd gTIgXkTTT< iK gh dc>c[d` will return the version of the module in the structure `Std_VersionInfoType` which additionally includes the VendorID and the ModuleID. Accessing the version defines which are specified in the header file `VcIgXkT(TTTNd gTIgXkTT #] :`

AUTOSAR Revision:

```
CIG KT( TTTNDJGTIG KTTT GTB DGTK: GH>DC
CIG KT( TTTNDJGTIG KTTT GTB>CDGTK: GH>DC
CIG KT( TTTNDJGTIG KTTT GTE I =TK: GH>DC
```

Module Version:

```
CIG KT( TTTNDJGTIG KTTTHLTB DGTK: GH>DC
CIG KT( TTTNDJGTIG KTTTHLTB>CDGTK: GH>DC
CIG KT( TTTNDJGTIG KTTTHLTE I =TK: GH>DC
```

3.6 Wakeup by bus event detection

If wakeup by bus detection is enabled in the configuration tool the callback function

`VcIgXkT(TTTNd gTIgXkTTT TLV e n h` has to be called by the lower layer in case of a wakeup. This function checks the specified CAN transceiver channel. If the transceiver hardware is in sleep mode and a wakeup by bus event is detected then the function returns : `TD` , otherwise : `TCDITD` .

3.6.1 Get bus wakeup reason

The service function `VcIgXkT(TTTNd gTIgXkTTT< i hLV eG Vhdc` returns the reason which caused the wakeup.

3.6.2 Set wakeup mode

The service function `VcIgXkT(TTTNd gTIgXkTTTH iLV eBdY` sets the wakeup mode which is required by the CAN interface.

3.6.3 Development Error Reporting

Development errors are reported to DET using the service `iTG edgi: ggdg`, if the pre-compile parameter `CIG KT(TTTNDJGTIG KTTT :KT: GGDGT :I: I 22 HI TDC`.

The reported CANTRCV instance ID must be specified in GENy. For details please refer to Configuration with GENy.

The reported service IDs identify the services which are described in 6.1. The following table presents the service IDs and the related services:

Service ID	Service
0x00	CanTrcv_30__Your_Trcv__Init
0x01	CanTrcv_30__Your_Trcv__SetOpMode
0x02	CanTrcv_30__Your_Trcv__GetOpMode
0x03	CanTrcv_30__Your_Trcv__GetBusWuReason
0x05	CanTrcv_30__Your_Trcv__SetWakeupMode
0x07	AUTOSAR3: CanTrcv_30__Your_Trcv__CB_WakeupByBus AUTOSAR4: CanTrcv_30__Your_Trcv__CheckWakeup
0x04	CanTrcv_30__Your_Trcv__GetVersionInfo
0x06	CanTrcv_30__Your_Trcv__MainFunction
0x08	CanTrcv_30__Your_Trcv__MainFunctionDiagnostics
0x09	CanTrcv_30__Your_Trcv__GetTrcvSystemData
0x0A	CanTrcv_30__Your_Trcv__ClearTrcvWufFlag
0x0B	CanTrcv_30__Your_Trcv__ReadTrcvTimeoutFlag
0x0C	CanTrcv_30__Your_Trcv__ClearTrcvTimeoutFlag
0x0D	CanTrcv_30__Your_Trcv__ReadTrcvSilenceFlag
0x0E	CanTrcv_30__Your_Trcv__CheckWakeFlag
0x0F	CanTrcv_30__Your_Trcv__SetPNActivationState

Table 3-3 Mapping of service IDs to services

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

Error Code	Description
0x01 CANTRCV_30__YOUR_TRCV__E_INVALID_CAN_NETWORK CANTRCV_30__YOUR_TRCV__E_INVALID_TRANSCEIVER	Invalid channel index is used in function argument list.
0x02 CANTRCV_30__YOUR_TRCV__E_PARAM_POINTER	Invalid pointer NULL_PTR is used in function argument list.
0x11 CANTRCV_30__YOUR_TRCV__E_UNINIT	CAN transceiver hardware is not initialized.
0x21 CANTRCV_30__YOUR_TRCV__E_TRCV_NOT_STANDBY	CAN transceiver hardware is not in standby mode.
0x22 CANTRCV_30__YOUR_TRCV__E_TRCV_NOT_NORMAL	CAN transceiver hardware is not in normal operation mode.
0x23 CANTRCV_30__YOUR_TRCV__E_PARAM_TRCV_WAKEUP_MODE	The requested wakeup mode is not valid.
0x24 CANTRCV_30__YOUR_TRCV__E_PARAM_TRCV_OPMODE	The requested operation mode is not supported by the underlying transceiver hardware.
0x25 CANTRCV_30__YOUR_TRCV__E_BAUDRATE_NOT_SUPPORTED	The selected baudrate is not supported by the underlying transceiver hardware.
0x40 AUTOSAR4 only: CANTRCV_30__YOUR_TRCV__E_NO_TRCV_CONTROL	If the CAN transceiver is not under control, which means that the transceiver does remains in an invalid state, this production error is raised.

Table 3-4 Errors reported to DET

3.6.4 Production Code Error Reporting

Production code related errors are reported to DEM using the service `bTG edgi:ggdgHiVi h` (specified in [3]), if the pre-compile parameter `CIG KT(TTTNDJGTIG KTTTEGD T:GGDGT :I: I 22 HI TDC`.

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

Error Code	Description
AUTOSAR3 only: CANTRCV_30__YOUR_TRCV__E_NO_TRCV_CONTROL	If the CAN transceiver is not under control, which means that the transceiver remains in an invalid state, this production error is raised.
CANTRCV_30__YOUR_TRCV__E_BUS_ERROR	A physical BUS error occurred.

Table 3-5 Errors reported to DEM

4 Integration

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

4.1 Scope of Delivery

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

4.1.1 Static Files

File Name	Description
CanTrcv_30___Your_Trcv__.h	Header file which has to be included by higher layers.
CanTrcv_30___Your_Trcv__.c	Implementation
AUTOSAR3 only: CanTrcv_30___Your_Trcv___CbK.h	Header file which has to be included by lower layers to access the supported callback functions.

Table 4-1 Static files

4.1.2 Dynamic Files

The dynamic files are generated by the configuration tool GENy (see 7.1.27.1.1).

File Name	Description
CanTrcv_30___Your_Trcv___Cfg.h	Header file contains type definitions and external data declarations. It is included by CanTrcv_30___Your_Trcv__.h.
CanTrcv_30___Your_Trcv___Cfg.c	Contains the configuration data.
AUTOSAR4 only: CanTrcv_GeneralTypes.h	Header file that contains all CanTrcv types specified by SWS. This file can be included by Can_GeneralTypes.h.

Table 4-2 Generated files

4.2 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 that are defined for the CANTRCV and illustrates their assignment among each other.

Memory Mapping Sections	Compiler Abstraction Definitions				
	CANTRCV_30__YOUR_TRCV__VAR	CANTRCV_30__YOUR_TRCV__APPL_VAR	CANTRCV_30__YOUR_TRCV__CONST	CANTRCV_30__YOUR_TRCV__CODE	CANTRCV_30__YOUR_TRCV__APPL_CODE
CANTRCV_30__YOUR_TRCV__START_SEC_CODE				■	■
CANTRCV_30__YOUR_TRCV__STOP_SEC_CODE					
CANTRCV_30__YOUR_TRCV__START_SEC_CONST_UNSPECIFIED			■		
CANTRCV_30__YOUR_TRCV__STOP_SEC_CONST_UNSPECIFIED					
CANTRCV_30__YOUR_TRCV__START_SEC_VAR_NOINIT_UNSPECIFIED	■	■			
CANTRCV_30__YOUR_TRCV__STOP_SEC_VAR_NOINIT_UNSPECIFIED					

Table 4-3 Compiler abstraction and memory mapping

4.3 Data consistency

The CAN transceiver driver calls service functions of upper layers in order to prevent interruption when accessing the CAN transceiver pins.

These service functions have to be provided by the components VStdLib, Schedule Manager or OSEK OS depending on which of these components is used for interrupt disable/restore handling. The component for interrupt control handling has to be selected in the configuration tool.

4.4 Integration

The driver has to be extended by hardware specific functionality. Within CanTrcv_30__Your_Trcv__.c and CanTrcv_30__Your_Trcv__.h, the parts that have to be implemented are marked with the following token:

<Your_Trcv_Code>

Please refer to the comments at the dedicated code part for integration tips. Also refer to the following chapters for details.



Info

Replace the placeholders __Your_Trcv__ and __YOUR_TRCV__ with the according name of the used transceiver. This procedure has to be done in both, the source code and the BSWMD files.

__YOUR_TRCV__ is used for definitions in upper case (e.g. TJA1041).

__Your_Trcv__ is used for variables in camel case (e.g. Tja1041).

4.5 Timers

As the underlying transceiver hardware may have some time constraints that must be met, the transceiver driver sometimes needs to wait some time until the next request to the hardware can be made.

An application function which handles this wait states is declared in CanTrcv_30__Your_Trcv__Cbk.h and has to be implemented by the user. To enable or disable this callback function and all predefined timers, you have to specify the following constant in the CanTrcv_30__Your_Trcv__.h:

```
Y [ c      CIG KT( TTTNDJGTIG KTTTJH: TI>B: GH      HI TDC
```

Declaration:

```
;JC [kd Y!    CIG KT( TTTNDJGTIG KTTT D :    eeaT VcIgXkT( TTTNd gTIgXkTTTLV i[] ci-
I b g>cY m 0
```

The parameter `IBgCYm` is used to distinguish between the timers needed by the transceiver driver. `IBgCYm` is represented by a symbolic constant that is defined in the `CanTrcv_30_Your_Trcv.h`. The following table lists all available timer indexes:

Timer Index (symbolic constant)	Wait Time	Description
<code>VcIgXkT(TTTNd gTIgXkTTTAdd e>c i</code>	<code>> 50µs</code>	This timer is called by the transceiver driver in function <code>>c i</code> after the transition to normal mode was made and before switching to the

4.6 Generated Data

The configuration tool generates various tables that can be used within the implementation. Also refer to the following files:

CanTrcv_30___Your_Trcv___Cfg.h (Type definitions, declarations)

CanTrcv_30___Your_Trcv___Cfg.c (Initializers, definitions)

4.6.1 CanTrcv_30___Your_Trcv___Channel

This array contains generic data for each channel. It has the following elements.

Vc>[TIGVchX k gBdY Ine VcIgXk>c iHiVi	State that shall be reached after CanTrcv_Init() has finished.
Wdda Vc VcIgXkLV e n hJh Y	TRUE if Wakeup By Bus over CanTrcv is activated for the given channel, FALSE otherwise.
Wdda Vc VcIgXk] Vcc aJh Y	TRUE if the channel has to be used, FALSE otherwise.
: X BTLV eHd gX Ine LV eHd gX	Wakeup Source of the given channel that will be reported to EcuM after a wakeup was detected.
Only if DBO D D F D ! ! F:	
Wdda Vc >X] Vcc aH i	TRUE if an ICU channel has been configured for the given channel, FALSE otherwise.
>X T] Vcc aIne >X] Vcc a	Configured ICU channel for the given CanTrcv channel. Only valid if IcuChannelSet == TRUE.
Only for DIO Interface:	
dT] Vcc aIne 1E c3	
□ # # >DTE>CTIG KTGM	

4.6.2 Partial networking data

If the underlying CanTrcv supports selective wakeup the following tables can be generated.

4.6.2.1 CanTrcv_30___Your_Trcv___PnData

This array contains the configured partial networking data for each channel. It has the following elements:

ci(>	Contains the ID as entered in the corresponding field in the generation tool.
ci(B H	Contains the MASK as entered in the corresponding field in the generation tool.
ci- I P-R	Contains the DATA bytes. The index is the same as of the corresponding field in the generation tool
ci- A	Contains the configured DLC.
ci- B I =T: MI	Has value "1" if the ID/MASK shall be interpreted as extended ID.
ci- J G I:	Configurator 5 only: Baudrate (e.g. 125, 250, 500, ...)

None of these values are used in the unmodified template driver. It is up to the user to implement the code necessary to download the data to the underlying CanTrcv hardware. The data must be written at >c iT] Vcc aTEc ViV in CanTrcv_30___Your_Trcv__.c.

4.6.2.2 CanTrcv_30___Your_Trcv___PnActivationState

This array indicates per channel whether selective wakeup is enabled or not. It contains a "1" if selective wakeup is activated

4.7 Handling of asynchronous APIs



Please note

This chapter only affects CanTrcv hardware that uses an asynchronous SPI Interface. Transceivers with DIO interface are always synchronous and thus do not need any special behavior.

Due to the complexity of the implementation it is NOT RECOMMENDED to use asynchronous SPI access.

Throughout this chapter, `CanTrcv_*` instead of `CanTrcv_30____Your_Trcv__*` is used in order to improve readability.

When using asynchronous SPI access, all APIs except `VcIgXkT>c i` shall behave asynchronously. This usually means that a call to an API only *requests* the dedicated action but does not actually *perform* it. Instead the request shall be held pending until the requested action is either completely performed or another concurrent action is requested.

As there are no restrictions basically each request can be interrupted by another request. The driver must thus handle concurrent and interrupted requests in order to guarantee proper functionality.

It is also required by the driver to inform upper layers about finished or cancelled requests according to the SWS.

The template driver contains a few mechanisms to help the user to implement a correct driver. Depending on the underlying CanTrcv hardware the actual requirements for implementation varies.

4.7.1 Request handling

The template driver already implements a mechanism to handle requests in a correct way. If asynchronous SPI access is used, the template driver uses `VcIgXkTH i G f` to store requests. These requests can be queried with the `VcIgXkT< i G f` macros. Refer to `CanTrcv_30___Your_Trcv__.c` for details.

Example:

If `CanTrcv_SetOpMode` is called, the template driver stores the request with `VcIgXkTH iDeBdY G f`. In context of `VcIgXkTBV c; cXi dc` the user implementation has to check the pending request with `VcIgXkT< iDeBdY G f` and has to perform all necessary tasks to complete it.

4.7.1.1 Concurrent operation modes

It is required that always the last pending operation mode request is completed. If another operation mode request is currently pending the driver has to safely cancel this old request before completing the new request.

Higher layer has to be informed only for the last operation mode request. Thus the implementation has to take care that no invalid indication is called.

The template driver takes care about this in a limited manner. To query a new request the user has to implement `H iDeBdY CdgbVa`, `H iDeBdY HiVcYWn` and `H iDeBdY Ha e` in `CanTrcv_30___Your_Trcv__.c`.

Within `VcIgXkTH iDeBdY` it is NOT ALLOWED to perform any SPI request. Instead all hardware access must be done in context of `VcIgXkTBV c; cXi dc`.

As soon as the request is finished the internal callback `VcIgXkTIgXkBdY >cY XVi dc` MUST be called by the user implementation.

The following table illustrates the necessary steps to change an operation mode.

Requested Mode	Actions
NORMAL	<p>Call <code>VcIgXkTH i dc[gbEc kV W a inG f hi</code></p> <p>Read the status flags (via <code>GetStatusFlags</code>)</p> <p>Call <code>VcIgXkT< iHiVi h; aV h>cY XVi dc</code> as soon as the status flags are read.</p> <p>Perform the actual mode change</p> <p>Call <code>VcIgXkTIgXkBdY >cY XVi dc</code> as soon as the mode change is complete</p>
STANDBY	<p>Perform the actual mode change</p> <p>Call <code>VcIgXkTIgXkBdY >cY XVi dc</code> as soon as the mode change is complete</p>
SLEEP	<p>Perform the actual mode change</p> <p>Call <code>VcIgXkTIgXkBdY >cY XVi dc</code> as soon as the mode change is complete</p>

If a mode change request is not yet completed when another new mode is requested the template driver executes the code located at `VcX aDeBdY G f`. Here, the user has to implement a function to safely cancel the old request. If it is necessary to wait for any event (e.g. SPI Indication) `VcIgXkTBV c; cXi dc` must handle the cancel request.

While a request is marked as “to be cancelled” the user implementation has to take care that the internal callback `VcIgXkTIgXkBdY >cY XVi dc` is NOT being called.

As soon as the old request is cancelled, the user implementation MUST call the macro `VcIgXkTEgdX hhC miG f hi`. And check if a new request is pending.

If an action other than mode change is requested the user implementation must handle the other request AFTER the mode change request has completed.

If another action is currently performed while a mode change is requested the user implementation must handle the operation mode change AFTER the other request has completed.

4.7.1.2 Request to CheckWakeFlag / ClearTrcvWufFlag / CB_WakeupByBus

If a request to `] X LV ;aV` or `a VgIgXkL [;aV` is pending, the corresponding code in `CanTrcv_30__Your_Trcv__.c` is executed.

The user implementation MUST NOT ignore any of these requests. If there is more than one request of the same type pending, the confirmation must be done only once.

Within the APIs it is not allowed to use SPI access. Instead the handling must be done in context of `VcIgXkTBV c[cXi dc`.

As soon as a request has completed the user implementation MUST call the corresponding internal callback `(VcIgXkT] X LV ;aV >cY XVi dc`

`VcIgXkT a VgIgXkL [;aV >cY XVi dc`

`VcIgXkT WLV e n h>cY XVi dc`).

If the hardware implementation allows handling multiple requests with the same SPI communication, it is allowed to call multiple internal callbacks at the same time.

4.7.1.3 Status Flag handling

When status flags are queried (at `GetStatusFlags` in `CanTrcv_30__Your_Trcv__.c`), the driver has to read out all status flags of the underlying `CanTrcv` hardware as soon as possible.

Upon request, the current status flags have to be marked as invalid by setting

`VcIgXkT(TTTNd gTIgXkTTTEgdWP cY mR#hiVi h;aV hGYn 2 ; AH: #`

If the SPI is currently blocked by another request, the request to read the status flags shall be held pending.

As soon as the status flags are completely read, the user implementation has to call the internal callback `VcIgXkT< iHiVi h;aV h>cY XVi dc`.

4.7.1.4 Indication handling

Most internal callbacks can be executed directly from a SPI callback. Depending on the last executed SPI command, the pending request and the state of the driver, the corresponding callback should be called.

Below is an example of the Elmos E520.13 implementation:

```
;JC [kd Y! CIG KT( T: * &(T D : VcIgXkT( T: * &(THe >cY XVi dc [ ci- VcIgXk>cY m
p
ci- k ci 2 CIG KT( T: * &(TLDG :GT:KTCDC: 0
hl iX] [ VcIgXkT( T: * &(He bY [[ gP VcIgXk>cY mR
p
XVh CIG KT( T: * &(TG THNHT > </
k ci 2 CIG KT( T: * &(TLDG :GT:KTG THNHT > <0
Wg V 0

XVh CIG KT( T: * &(TG THNHTH: I /
k ci 2 CIG KT( T: * &(TLDG :GT:KTG THNHTH: I 0
Wg V 0

XVh CIG KT( T: * &(TLGTHNHTH: I /
k ci 2 CIG KT( T: * &(TLDG :GT:KTLGTHNHTH: I 0
Wg V 0

Y [V ai/
Wg V 0
r

[ [k ci 2 CIG KT( T: * &(TLDG :GT:KTCDC:
p
[ [k ci 22 CIG KT( T: * &(TLDG :GT:KTG THNHT > <
p
C l hiVi h [aV h g X k Y###
VcIgXkT( T: * &(T< iHiVi h;aV h>cY XVi dc [ VcIgXk>cY m 0
r
BdY Ldg g
[ [ VcIgXkT( T: * &(TEgdWP VcIgXk>cY mR#Ldg gHiVi 2 CIG KT( T: * &(TLDG :GTHIDEE:
[ VcIgXkT( T: * &(TEgdWP VcIgXk>cY mR# h>c i 22 CIG KT( T: * &(T>HT>C>I

p
>c[ dgb ldg g VWd i i] h k ci
[kd Y VcIgXkT( T: * &(TBdY Ldg g [ VcIgXk>cY m! k ci 0
r

"" >cY XVi dch
[ [k ci 22 CIG KT( T: * &(TLDG :GT:KTG THNHT > <
p
[ [ VcIgXkT( T: * &(T< i agL [;aV G f [ VcIgXk>cY m 2 m
p
VcIgXkT( T: * &(T a VgIgXkL [;aV >cY XVi dc [ VcIgXk>cY m 0
r
[ [ VcIgXkT( T: * &(T>hG f;aV EcY [ VcIgXk>cY m
p
[ [ VcIgXkT( T: * &(T< i ] LV ;aV G f [ VcIgXk>cY m 2 m
p
VcIgXkT( T: * &(T ] X LV ;aV >cY XVi dc [ VcIgXk>cY m 0
r
[ [ VcIgXkT( T: * &(T< i W n h;aV G f [ VcIgXk>cY m 2 m
p
VcIgXkT( T: * &(T WL V e n h>cY XVi dc [ VcIgXk>cY m 0
r
G a Vh GY V g f hi ] g Vh c l g f hih id g VY > < h] d aY W f g Y
VcIgXkT( T: * &(TEgdWP VcIgXk>cY mR#G VY V G f hiHiVi 2
CIG KT( T: * &(TG T > <THI I:T> A: 0
r
r
r
;g HE>
VcIgXkT( T: * &(He bY [[ gP VcIgXk>cY mR 2 CIG KT( T: * &(TCDTG:F0
r
```

5 Dependencies to other components

5.1 Dio driver

Depending on the configuration, the CanTrcv driver performs hardware access by calling service functions of the lower layer component Dio driver:

- > Function Dio_WriteChannel() is used to set the logical level of the channel pins to which the CAN transceiver hardware is connected.
- > Function Dio_ReadChannel() is used to get the logical level of the channel pins to which the CAN transceiver hardware is connected.
- > The Dio driver has to provide the channel dependent assignment for the CAN transceiver hardware pins which are specified in the tool. These pins are referred by the CAN transceiver driver by using the symbolic names specified in the tool.

5.2 SPI driver

Depending on the configuration, the CanTrcv driver performs hardware access by calling service functions of the lower layer component SPI driver:

- Function Spi_SyncTransmit() / Spi_AsyncTransmit() is used to issue a sequence in order to read from or write to the transceiver hardware.
- The SPI driver must be configured to allow access to the transceiver. This means that sequences, jobs and channels must be defined prior using the transceiver driver.

5.2.1 SPI Configuration

The configuration of the SPI driver strongly depends on the underlying CanTrcv hardware. There is no tool support for specifying which sequences / channels / jobs are used by the CanTrcv driver.

5.3 Icu driver

The CAN transceiver driver performs hardware access by calling service functions of the lower layer component Icu driver:

- > Function `>X T: hVWa Cdi [XVi dc` is used by the transceiver driver to disable the ICU notification after wakeup event notification.
- > Function `>X T: cVWa Cdi [XVi dc` is used by the transceiver driver to enable the ICU notification during the transition into the STANDBY mode.



Please note

The following chapter applies only to this use case:

- the used CAN controller does not support the feature "wakeup by CAN bus", i.e. the CAN controller cannot detect incoming CAN messages and generate a so-called wakeup-interrupt
- the ECU shall wake up and start its own communication due to detected communication on the CAN bus

The proposal described in this chapter enables the user to support the use-case by using an additional μ C I/O port to generate the wakeup information via the ICU driver. This I/O port is connected either parallel to the CAN Rx port or is directly attached to the CAN transceivers ERR port (depending on the used CAN transceiver). The following steps have to be done for every CAN channel that shall be able to wake up via the CAN bus:

5.3.1 ICU configuration

Add an ICU channel

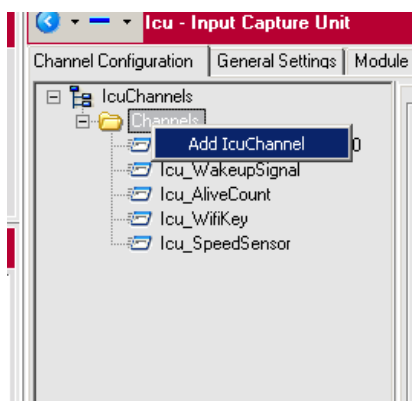


Figure 5-1 Add an ICU channel

The user has to configure the ICU channel and to add a signal notification function.

Additionally the wakeup source for the CAN channel which shall be handled via this ICU channel have to be chosen.

In dependency of the provided ICU configuration options it is possible to configure the "IcuDefaultStartEdge". This option should be configured to ICU_FALLING_EDGE, because

the wakeup event is provided by the Rx pin and/ or the ERR pin via a level change from recessive to dominant.

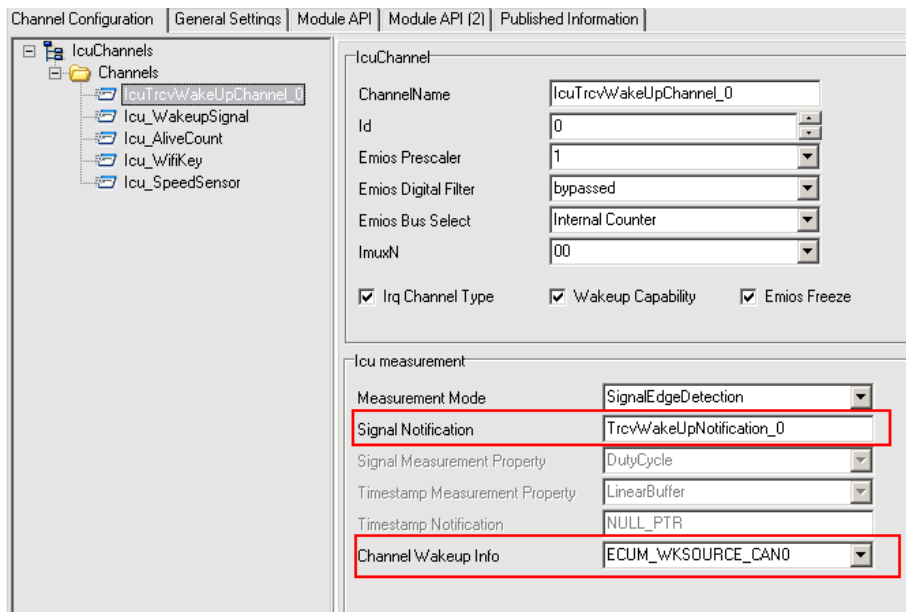


Figure 5-2 ICU channel configuration for wakeup via transceiver

If the transceiver shall also wake up the ECU and not only the CAN channel then it is necessary to activate the “Wakeup Capability” for this ICU channel.

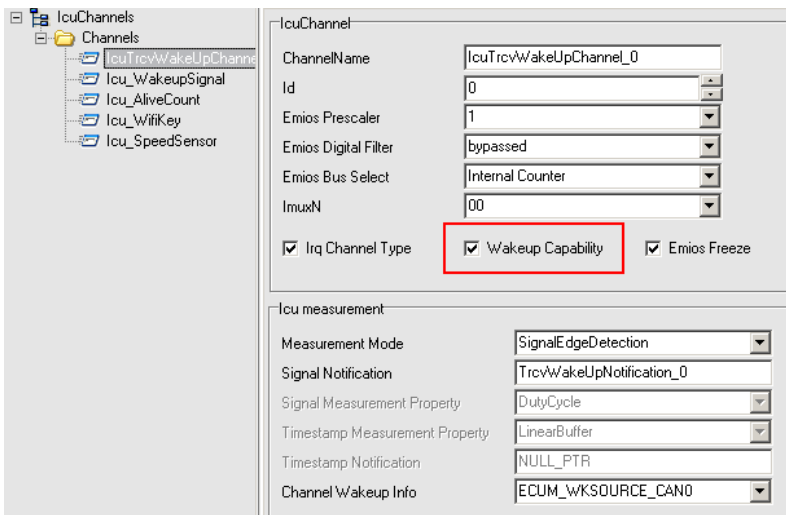


Figure 5-3 ICU wakeup capability

5.3.2 Implementation of the signal notification function

The user has to implement the signal notification function as shown in the following example:

Example

```
kd Y >X TIgXkLV JeCdi [ XVi dcT kd Y
p
    c[dgb i] :X B VWd i i] lV e k ci! i] eVgVb i g
    h i] Xdc[ g Y igVchX k g lV e hd gX
: X BT ] X LV Je JBTL HDJG :T C 0
r
```



Please note

If no wakeup validation is used for the configured wakeup source, then it is possible to call `EcuM_SetWakeUpEvent()` instead of `EcuM_CheckWakeUp()`.

6 API Description

6.1 Services provided by CANTRCV

The CANTRCV API consists of services, which are realized by function calls.

6.1.1 CanTrcv_30___Your_Trcv___InitMemory

Prototype	
k d Y D	N ! □ k d Y
Parameter	
-	-
Return code	
-	-
Functional Description	
This function initializes the memory and needed values of the CAN transceiver driver.	
Particularities and Limitations	
> This function must be called before any other functionality of the CAN transceiver driver.	
Expected Caller Context	
> This function must be called from task level and is not reentrant.	

Table 6-1 CanTrcv_30___Your_Trcv___InitMemory

6.1.2 CanTrcv_30___Your_Trcv___Init

Prototype	
<pre>kd Y D VcIgXkT(TTTNd gTIgXkTTT dc[Ine dc[Eig</pre>	
Parameter	
dc[Eig	Pointer to the VcIgXkT(TTTNd gTIgXkTTT dc[struct. If multiple configurations are available, the active configuration can be selected by using the related VcIgXkT(TTTNd gTIgXkTTT dc[T1>Y ci inCVb 3 struct
Return code	
-	-
Functional Description	
> This function initializes all channels of the CAN Transceiver driver which are configured in the configuration tool.	
Particularities and Limitations	
> The function VcIgXkT(TTTNd gTIgXkTTT>c iB bdgn must be called before the function VcIgXkT(TTTNd gTIgXkTTT>c i can be called. > This function must be called before any other service functionality of the Transceiver driver.	
Expected Caller Context	
> This function must be called from task level and is not reentrant.	

Table 6-2 CanTrcv_30___Your_Trcv___Init

6.1.3 CanTrcv_30__Your_Trcv__SetOpMode

Prototype

```

JIDH G( /
HiYTG i gcIne D N ! □
Vc>[TIGVchX k gBdY Ine DeBdY ! ci- VcIgXk>cY m
JIDH G) /
HiYTG i gcIne D N ! □ ci- VcIgXk>cY m!
VcIgXkTIgXkBdY Ine DeBdY

```

Parameter

OpMode	Pointer which contains the desired operation mode.
CanTrcvIndex	Index of the selected transceiver.

Return code

E_OK / E_NOT_OK	<p>E_OK: is returned if the transceiver state has been changed to the requested mode.</p> <p>E_NOT_OK: is returned if the transceiver state change has failed or the parameter is out of the allowed range. The previous state has not been changed.</p>
-----------------	--

Functional Description

Sets the CAN transceiver to the requested operation mode. These operation modes are:

AUTOSAR3: C>;TIG KTBD :T

AUTOSAR4: CIG KTIG KBD :T

- CDGB A
- HI C N
- HA: : E

If the transceiver is requested to change into NORMAL mode and selective wakeup is enabled on channel VcIgXk>cY m the hardware is queried for error flags. If no error was detected the driver calls the notification function Vc>[T(TTTNd gTIgXkTTT dc[gbEc kV aVW a in

6.1.4 CanTrcv_30__Your_Trcv__GetOpMode

Prototype	
<pre> JIDH G(/ HiYTG i gcIne D N ! □ Vc>[TIGVchX k gBdY Ine DeBdY ! ci- VcIgXk>cY m JIDH G) / HiYTG i gcIne D N ! □ ci- VcIgXk>cY m! VcIgXkTTIgXkBdY Ine DeBdY </pre>	
Parameter	
OpMode	Pointer to operation mode of the bus the API is applied to.
CanTrcvIndex	Index of the selected transceiver.
Return code	
E_OK / E_NOT_OK	E_OK: is returned if the operation mode was detected. E_NOT_OK: is returned if the operation mode was not detected.
Functional Description	
Stores the current operation mode of the selected CAN transceiver to DeBdY . These operation modes are: AUTOSAR3: C>;TIG KTBD :T AUTOSAR4: CIG KTIG KBD :T > CDGB A > HI C N > HA: : E	
Particularities and Limitations	
> The CAN transceiver driver must be initialized. > If a mode change was requested before, the reported operation mode may not be valid until the mode change is completed and Vc>[T(TTTNd gTIgXkTTTIgXkBdY>cY XVi dc was called.	
Expected Caller Context	
> This function can be called from task or interrupt level and is not reentrant.	

Table 6-4 CanTrcv_30__Your_Trcv__GetOpMode

6.1.5 CanTrcv_30___Your_Trcv___GetBusWuReason

Prototype	
<pre> JIDH G(/ HiYTG i gcIne D C ! □ ci- VcIgXk>cY m! Vc>[TIgXkLV eG VhdcIne G Vhdc JIDH G) / HiYTG i gcIne D C ! □ ci- VcIgXk>cY m! VcIgXkTIgXkLV eG VhdcIne G Vhdc </pre>	
Parameter	
CanTrcvIndex	Index of the selected transceiver.
Reason	Pointer to wake up reason of the bus the API is applied to.
Return code	
E_OK / E_NOT_OK	E_OK: is returned if the wake up reason was detected. E_NOT_OK: is returned if the wake up reason was not detected.
Functional Description	
Stores the last the wakeup reason for the channel VcIgXk>cY m to G Vhdc. These wakeup reasons are: AUTOSAR3: C>;TIG KT AUTOSAR4: CIG KT > LJT>CI: GC AAN: The wakeup was caused by setting the CAN transceiver to normal operation mode via VcIgXkT(TTTNd gTIgXkTTTH iDeBdY . > LJT: GGDG: No wakeup was detected by the transceiver and no reason is stored. The function returns : TCDITD . > LJTCDITHJEEDGI: : No wakeup detection supported by this CAN transceiver. The function returns : TCDITD . > LJT NT JH: The wakeup was caused by an external bus wakeup. > LJTG: H: I: The wakeup was detected after a reset. > LJTEDL: GTDC: The transceiver detected a power-on wakeup > LJT NTHNH: GG: An internal hardware error occurred that caused the transceiver to wake up. > LJT NTE>C: The wakeup was caused by applying an edge to the WAKE pin.	
Particularities and Limitations	
> The CAN transceiver driver must be initialized. > The wakeup reason represents always the last detected wakeup reason. If there was more than one wakeup detected only the last one will be reported.	
Expected Caller Context	
> This function can be called from task or interrupt level and is not reentrant.	

Table 6-5 CanTrcv_30___Your_Trcv___GetBusWuReason

6.1.7 CanTrcv_30___Your_Trcv___GetVersionInfo

Prototype	
kd Y D ! HiYTK gh dc>c[dIne K gh dc>c[d	
Parameter	
VersionInfo	Pointer to version information of this module.
Return code	
-	-
Functional Description	
Gets the version of the module and returns it in VersionInfo.	
Particularities and Limitations	
> The CAN transceiver driver must be initialized.	
Expected Caller Context	
> This function can be called from task or interrupt level and is not reentrant.	

Table 6-7 CanTrcv_30___Your_Trcv___GetVersionInfo

6.1.8 CanTrcv_30__Your_Trcv__CB_WakeupByBus

Prototype	
<pre> JIDH G(/ HiYTG i gcIne D VcIgXk>cY m JIDH G) / HiYTG i gcIne / D </pre>	
Parameter	
CanTrcvIndex	Index of the selected transceiver.
Return code	
:TD :TCDITD	<p>In synchronous mode:</p> <p>:TD a wakeup-by-bus event was detected</p> <p>:TCDITD no wakeup was detected or an error occurred</p> <p>In asynchronous mode:</p> <p>:TD : the request to check for wakeup was acknowledged.</p> <p>:TCDITD : an error occurred.</p>
Functional Description	
<p>This function requests the CanTrcv driver to check for wakeups and report them. If a wakeup was detected, the CanTrcv reports it by calling of : X BTH iLV e:k ci.</p> <p>In asynchronous mode, if no wakeup was detected : X BT:cY] X LV e is called.</p>	
Particularities and Limitations	
<ul style="list-style-type: none"> > The CAN transceiver driver must be initialized. > Do not use return value :TD for wakeup detection. A wakeup can be considered as valid only if : X BTH iLV e:k ci was called for the corresponding wakeup source. 	
Expected Caller Context	
<ul style="list-style-type: none"> > This function can be called from task or interrupt level and is not reentrant. 	

Table 6-8 CanTrcv_30__Your_Trcv__CB_WakeupByBus

6.1.9 CanTrcv_30___Your_Trcv___GetTrcvSystemData

Prototype	
<pre>HiYTG i gcIne D E ! □ ci- VcIgXk>cY m! ci(IgXkHnh ViV</pre>	
Parameter	
VcIgXk>cY m	Index of the selected transceiver
IgXkHnh ViV	Pointer to the diagnosis data buffer to store the information in.
Return code	
:TD / :TCDITD	:TD if the diagnosis register was successfully read, :TCDITD otherwise.
Functional Description	
Reads the diagnosis registers of the underlying transceiver hardware and stores them in IgXkHnh ViV.	
Particularities and Limitations	
<ul style="list-style-type: none"> ■ The CAN transceiver driver must be initialized. ■ The underlying transceiver hardware must support Selective Wakeup / Partial Networking ■ Hnh V ViV contains only valid information if :TD was returned. ■ The function will not accept any requests if there are still SPI requests pending. In this case :TCDITD is returned. 	
Expected Caller Context	
This function can be called from task or interrupt level and is not reentrant.	

Table 6-9 CanTrcv_30___Your_Trcv___GetTrcvSystemData

6.1.10 CanTrcv_30___Your_Trcv___ClearTrcvWufFlag

Prototype	
<pre>HiYTG i gcIne D D G ! □ ci- VcIgXk>cY m</pre>	
Parameter	
VcIgXk>cY m	Index of the selected transceiver
Return code	
: TD / : TCDITD	: TD if the WUF flag has been cleared, : TCDITD otherwise.
Functional Description	
<p>This service requests the transceiver driver to clear all wakeup flags from the underlying transceiver hardware.</p> <p>If E_OK is returned, the driver will call the notification function</p> <pre>Vc>[T(TTTNd gTIgXkTTT a VgIgXkL [;aV >cY XVi dc</pre> <p>as soon as the request has been completed. Note that the notification may occur in interrupt context, in task context or in context of!</p> <pre>VcIgXkT(TTTNd gTIgXkTTT a VgIgXkL [;aV .</pre>	
Particularities and Limitations	
<ul style="list-style-type: none"> ■ The CAN transceiver driver must be initialized. ■ The underlying transceiver hardware must support Selective Wakeup / Partial Networking 	
Expected Caller Context	
This function can be called from task or interrupt level and is not reentrant.	

Table 6-10 CanTrcv_30___Your_Trcv___ClearTrcvWufFlag

6.1.11 CanTrcv_30___Your_Trcv___ReadTrcvTimeoutFlag

Prototype	
<pre>HiYTG i gcIne D G ! □ ci- VcIgXk>cY m! VcIgXkT(TTTNd gTIgXkTTT;aV HiVi Ine ;aV HiVi</pre>	
Parameter	
VcIgXk>cY m	Index of the selected transceiver
; aV HiVi	State of the timeout flag.
Return code	
: TD / : TCDITD	: TD if status of the timeout flag is successfully read. : TCDITD otherwise.
Functional Description	
Reads the status of the timeout flag from the underlying transceiver hardware and stores it to ; aV HiVi .	
Particularities and Limitations	
<ul style="list-style-type: none"> The CAN transceiver driver must be initialized. 	
Expected Caller Context	
This function can be called from task or interrupt level and is not reentrant.	

Table 6-11 CanTrcv_30___Your_Trcv___ReadTrcvTimeoutFlag

6.1.12 CanTrcv_30___Your_Trcv___ClearTrcvTimeoutFlag

Prototype	
<pre>HiYTG i gcIne D D G ! □ ci- VcIgXk>cY m</pre>	
Parameter	
VcIgXk>cY m	Index of the selected transceiver
Return code	
: TD / : TCDITD	: TD if the timeout flag has been cleared, : TCDITD otherwise.
Functional Description	
Clears the timeout flag from the underlying transceiver hardware.	
Particularities and Limitations	
<ul style="list-style-type: none"> The CAN transceiver driver must be initialized. 	
Expected Caller Context	
This function can be called from task or interrupt level and is not reentrant.	

Table 6-12 CanTrcv_30___Your_Trcv___ClearTrcvTimeoutFlag

6.1.13 CanTrcv_30__Your_Trcv__ReadTrcvSilenceFlag

Prototype	
<pre>HiYTG i gcIne D G ! □ ci- VcIgXk>cY m! VcIgXkT(TTTNd gTIgXkTTT;aV HiVi Ine ;aV HiVi</pre>	
Parameter	
VcIgXk>cY m	Index of the selected transceiver
; aV HiVi	State of the silence flag.
Return code	
: TD / : TCDITD	: TD if status of the silence flag is successfully read. : TCDITD otherwise.
Functional Description	
Reads the status of the silence flag from the underlying transceiver hardware and stores it to ; aV HiVi .	
Particularities and Limitations	
■ The CAN transceiver driver must be initialized.	
Expected Caller Context	
This function can be called from task or interrupt level and is not reentrant.	

Table 6-13 CanTrcv_30__Your_Trcv__ReadTrcvSilenceFlag

6.1.14 CanTrcv_30___Your_Trcv___CheckWakeFlag

Prototype	
<pre>HiYTG i gcIne D D G ! □ ci- VcIgXk>cY m</pre>	
Parameter	
VcIgXk>cY m	Index of the selected transceiver
Return code	
:TD /:TCDITD	:TD if request for checking wakeup flag has been accepted, :TCDITD otherwise.
Functional Description	
<p>Requests the driver to check the status of the wake flag of the underlying transceiver hardware. Any detected wakeup will be reported through service EcuM_SetWakeupEvent. The correct wakeup reason can be determined by using VcIgXkT(TTTNd gTIgXkTTT< i hL G Vhdc#</p> <p>If E_OK is returned, the driver will call the notification function Vc>[T(TTTNd gTIgXkTTT] X IgXkLV ;aV >cY XVi dc as soon as the request has been completed. Note that the notification may occur in interrupt context, in task context or in context of VcIgXkT(TTTNd gTIgXkTTT] X LV ;aV .</p>	
Particularities and Limitations	
<ul style="list-style-type: none"> ■ The CAN transceiver driver must be initialized. 	
Expected Caller Context	
This function can be called from task or interrupt level and is not reentrant.	

Table 6-14 CanTrcv_30___Your_Trcv___CheckWakeFlag

6.1.15 CanTrcv_30___Your_Trcv___MainFunction

Prototype	
kd Y D	N G ! kd Y
Parameter	
"	-
Return code	
"	"
Functional Description	
This service can be called from task level and periodically checks if a wakeup was detected by the underlying transceiver hardware.	
Particularities and Limitations	
■ The CAN transceiver driver must be initialized.	
Expected Caller Context	
This function can be called from task context and is not reentrant.	

Table 6-15 CanTrcv_30___Your_Trcv___MainFunction

6.1.16 CanTrcv_30___Your_Trcv___SetPNActivationState

Prototype		
HiYTG i gcIne D OB ! □ VcIgXkT(TTTNd gTIgXkTTTEC Xi kVi dcIne Xi kVi dcHiVi		
Parameter		
Xi kVi dcHiVi	Specifies whether to enable or disable selective wakeup.	
Return code		
: TD : TCDITD	: TD is returned if PN activation state was successfully changed. : TCDITD is returned if PN activation state could not be changed or an error occurred.	
Functional Description		
<p>This service changes the state of the selective wakeup feature. If called with ECT: C A: CanTrcv hardware will be configured to wake up on configured frames only (WUF).</p> <p>If called with ECT >H A: CanTrcv hardware will be configured to wake up on any bus activity (WUP).</p> <p>In order to enable selective wakeup it must be also configured as enabled in generation tool.</p>		
Particularities and Limitations		
<ul style="list-style-type: none">■ The CAN transceiver driver must be initialized.■ State change to ECT >H A: will be effective after next mode change.■ Selective wakeup can be enabled only for channels that have PN activated in generation tool.■ It is strongly recommended to use this API only directly after VcIgXkT(TTTNd gTIgXkTTT>c i □ .		
Expected Caller Context		
This function must be called from task and is not reentrant. The API must not be interrupted by any other CanTrcv API.		

Table 6-16 CanTrcv_30___Your_Trcv___SetPNActivationState

6.2 Services used by CANTRCV

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

Component	API
DET	Det_ReportError
DEM	Dem_SetEventStatus
SPI	Spi_SetupEB
SPI	Spi_SyncTransmit
SPI	Spi_AsyncTransmit
DIO	Dio_WriteChannel
DIO	Dio_ReadChannel
CANIF	CanIf_TrcvModeIndication
CANIF	CanIf_ConfirmPnAvailability
CANIF	CanIf_ClearTrcvWufFlagIndication
CANIF	CanIf_CheckTrcvWakeFlagIndication
CANIF	CanIf_TrcvModeIndication
ICU	Icu_EnableNotification
ICU	Icu_DisableNotification
ECUM	EcuM_SetWakeupEvent
ECUM	EcuM_CheckWakeup
ECUM	EcuM_EndCheckWakeup

Table 6-17 Services used by the CANTRCV

7 Configuration

7.1 Configuration with GENy

The CANTRCV is configured with the help of the configuration tool GENy.

7.1.1 Component Selection

Software Components	ECU: CH0_Node0	Channel0	Channel1	Channel2	Channel3	Channel4	Channel5
CanIf	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
CanNm	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CanSM	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
CanTp	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
CanTrcv_Generic1	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CanTrcv_Generic2	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CanTrcv_Generic3	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
CanTrcv_Tja1041	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Figure 7-1 CANTRCV component selection

The component CAN transceiver generic driver can be added to the system channel configuration by enabling the checkbox CanTrcv_Generic<x> (x = 1..3) in the software component selection. Note that up to 3 generic transceivers can be used in the system, but it is not mandatory to use all of them.

7.1.2 General settings

The general settings affect all enabled CAN transceiver channels.

ECU: DUT_CH1

- Components
 - Can_V85xFcn
 - CanIf
 - CanTrcv_Generic1
 - Channels
 - GenTool_GenyAsrBase
 - GenTool_GenyPluginAsrIfEc
 - GenTool_GenyPluginConfigD
 - Hw_V85xFcnCpu
 - NameDecorator
 - zBrs_EmbeddedRunTimeSys
- Tx Messages
- Rx Messages
- Tx Signals
- Rx Signals

Configurable Options

CanTrcv_Generic1

Common

Configuration Variant: Variant 1 (Pre-compile Configuration)

Version Info Api: ☐ *

Dev Error Detect: ☒ *

Prod Error Detect: ☒ *

User Config File: *

Miscellaneous

Component specific Name: Generic1*

Use SPI: ☐

Use synchronous SPI: ☒ *

Supports Partial Networking: ☐ *

Index: 0*

Wake Up Support: None

DIO Pin: Add

DIO Pin: Delete DIO_TRCV_RX

DEM Event: Add

DEMEvent: Delete DEM_TRCV_CUSTOM_1

Figure 7-2 CANTRCV general configuration

- > **Version Info Api:** Enable this checkbox if service function CanTrcv___Your_Trcv___GetVersionInfo() is used.
- > **Dev Error Detection:** Enable/ disable the development error notification to DET.

- > **Prod Error Detection:** Enable/ disable the product error notification to DEM.
- > **User Config File:** The content of this specified user file is added to the file CanTrcv_30___Your_Trcv___dio_Cfg.h.
- > **Component specific Name:** Insert user specific name of the transceiver component. Names of files CanTrcv_30___Your_Trcv___c, CanTrcv_30___Your_Trcv___h and CanTrcv_30_Xdio_Cbk.h c have to be adapted. Replace "X" by the component specific name in camel case. In each generic transceiver driver file (static or generated) replace every "__YOUR_TRCV__" by the component specific name in upper case and replace every "__Your_Trcv__" by the component specific name in camel case.
- > **Use SPI:** Enable if the underlying CanTrcv hardware is controlled via SPI
- > **Use synchronous SPI:** Specifies whether accesses over SPI should be done asynchronously or not. If disabled it is required that the SPI driver is configured as level 1 or level 2 driver. Additionally it is required that the function `VcIgXkTBV c[cXi dc` is called periodically.
- > **Supports Partial Networking:** Enable if the underlying CanTrcv hardware supports selective wakeup.
- > **Index:** Specify the instance id which is used for DET error reporting for this module. A default of 0 is set.
- > **Wake Up Support:** Select the type of wakeup event detection, "ISR" or "Polling". Select "None" if wakeup event detection is not used.

**Please note**

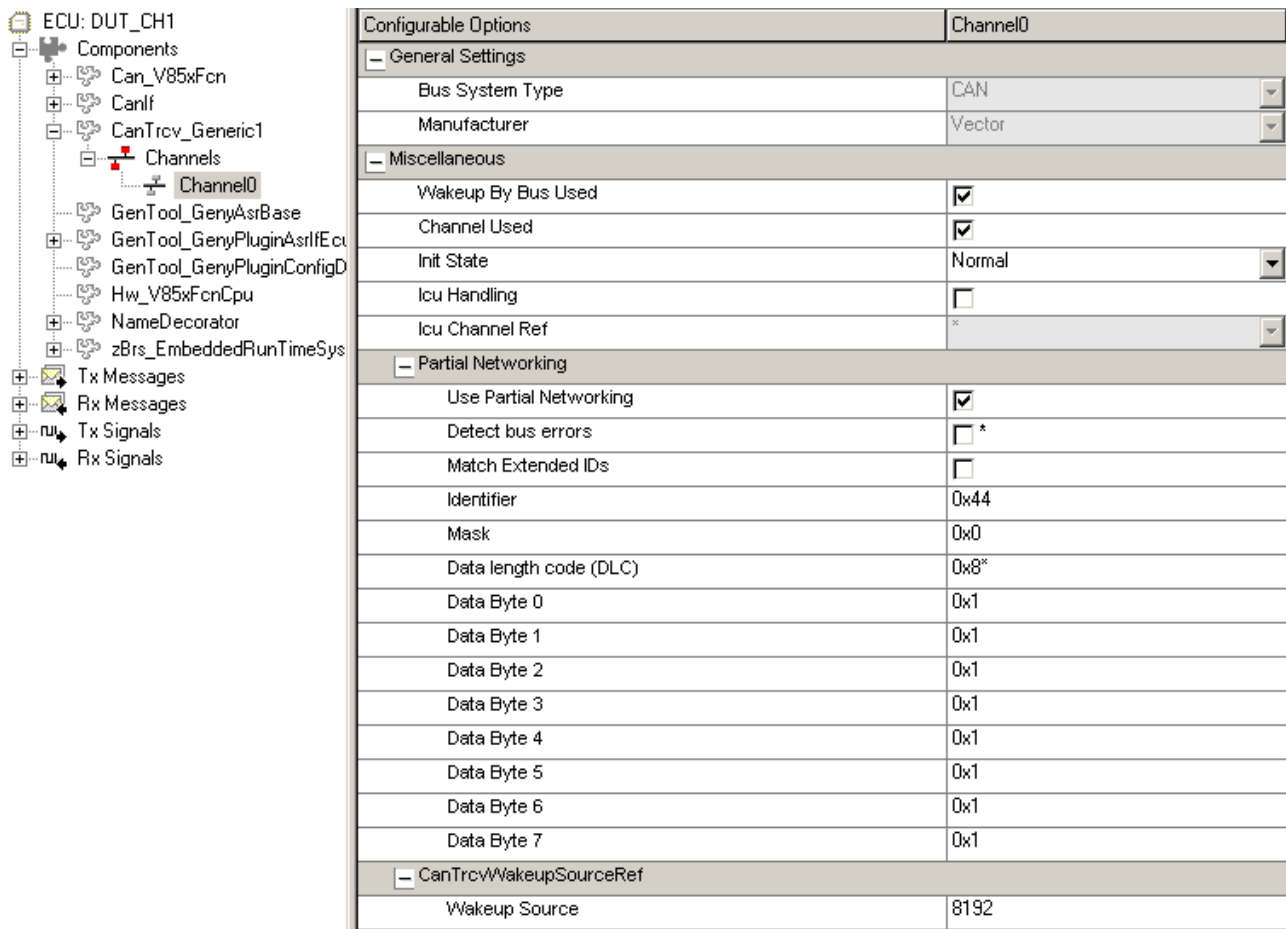
In order to enable the extension regarding support for the CAN controllers which do not support the wakeup by bus (see 5.3), a user config file has to be created. The user will contain the following definition:

```
#define CANTRCV_30___YOUR_TRCV___USE_ICU STD_ON
```

- > **DIO Pin:** Press button "Add" to get as many pins as required. Insert a unique pin name into each field. The CAN transceiver driver refers the DIO pin channel assignment by using these pin names. The component DIO has to provide these pins assignment with channel postfix, e.g. PIN_STB_0 is used for transceiver channel 0 and PIN_STB_1 is used for transceiver channel 1.
- > **DEM Event:** Press button "Add" to add a custom DEM Event for this transceiver driver instance. The DEM event will also be allocated at DEM module

7.1.3 Channel Specific Configuration Options

Select the transceiver channel view to set the channel specific configuration options.



Configurable Options		Channel0
General Settings		
Bus System Type		CAN
Manufacturer		Vector
Miscellaneous		
Wakeup By Bus Used	<input checked="" type="checkbox"/>	
Channel Used	<input checked="" type="checkbox"/>	
Init State		Normal
Icu Handling	<input type="checkbox"/>	
Icu Channel Ref		*
Partial Networking		
Use Partial Networking	<input checked="" type="checkbox"/>	
Detect bus errors	<input type="checkbox"/>	*
Match Extended IDs	<input type="checkbox"/>	
Identifier		0x44
Mask		0x0
Data length code (DLC)		0x8
Data Byte 0		0x1
Data Byte 1		0x1
Data Byte 2		0x1
Data Byte 3		0x1
Data Byte 4		0x1
Data Byte 5		0x1
Data Byte 6		0x1
Data Byte 7		0x1
CanTrcvWakeupSourceRef		
Wakeup Source		8192

Figure 7-3 CANTRCV channel dependent configuration

- > **Wakeup By Bus Used:** Enable this checkbox if wakeup by bus is used on the selected channel.
- > **Channel Used:** Enable this checkbox if the selected channel has to be handled by the CAN transceiver driver.
- > **Init State:** Select the operation mode which has to be set when the CAN transceiver is first time initialized after power on or reset.



Caution

It is not recommended to select any other Init State than Normal if the attached CAN Controller is unable to initialize with a permanent dominant Rx pin.

- > **Icu Handling:** If enabled, the transceiver will enable or disable the selected Icu channel on certain mode changes. See chapter 5.3 for details.
- > **Icu Channel Ref:** Select the Icu channel for this transceiver. The value can only be specified if Icu handling is enabled. See chapter 5.3 for details.

- > **Use Partial Networking:** Enables or disables the partial networking feature which allows the transceiver to be woken up only by a specific wakeup frame specified by Identifier, Mask, DLC and Data Byte 0-7.
- > **Detect bus errors:** Enables for handling of BUSERR flag. When activated on at least one channel, the API `VcIgXkT(TTTNd gTIgXkTTTBV c; cXi dc V cdhi Xh` can be used to poll the state of the BUSERR flag for all channels where this attribute is set to TRUE.
- > **Match Extended IDs:** Enables support for extended IDs. If activated, the configured Identifier and Mask is considered to be 29 instead of 11bit.
- > **Identifier:** Specifies the ID of the wakeup frame used for selective wakeup. If "Match Extended IDs" is set to false, the ID has to be entered as 11bit ID (0x0 - 0x7ff). Otherwise, you have to enter a 29 bit ID (0x0 - 0x1ffffff).
- > **Mask:** You may set a mask to specify an ID range used for selective wakeup. If "Match Extended IDs" is set to false, the Mask has to be entered as 11bit Mask (0x0 - 0x7ff). Otherwise, you have to enter a 29bit Mask (0x0 - 0x1ffffff). If a bit is set to 0, the corresponding bit of the received frame must match to the corresponding bit value specified in ID. If a bit is set to 1, the bit of the received frame is ignored and considered as always correct.
- > **Data length code (DLC):** Specified the length of the wakeup frame used for selective wake up the transceiver.
- > **Data Byte [0-7]:** Defines the n-th byte (Byte0 = LSB) of the data payload mask to be used on the received payload in order to determine if the Transceiver must be woken up by the received Remote Wakeup Frame. The payload matches if the first bit with value one of the received frame matches the first bit with value one of the configured payload.
- > **Wakeup Source:** Select the wakeup source for the according CAN transceiver channel.

7.2 Configuration with DaVinci Configurator 5

Refer to the integrated online help and parameter descriptions of Configurator 5.

8 AUTOSAR Standard Compliance

8.1 Additions/ Extensions

8.1.1 Memory initialization

To have an independent memory initialization for this BSW module the additional function `VcIgXkT(TTTNd gTIgXkTTT>c iB bdgn` was added.

8.2 Deviations

While the driver is implemented according [1], some requirements could not be fulfilled in order to ensure proper functionality. The following chapter lists these deviations.

8.2.1 Notification functions

According to SWS [1] the transceiver shall call notification functions in CanIf. As the given CanTrcvChannelId is valid only for one transceiver driver instance, it was decided to call the following notification functions instead:

SWS API	Used API
CanIf_TrvcModelIndication	CanIf_30___Your_Trvc___TrvcModelIndication
CanIf_ConfirmPnAvailability	CanIf_30___Your_Trvc___ConfirmPnAvailability
CanIf_ClearTrcvWufFlagIndication	CanIf_30___Your_Trvc___ClearTrcvWufFlagIndication
CanIf_CheckTrcvWakeFlagIndication	CanIf_30___Your_Trvc___CheckTrcvWakeFlagIndication

Table 8-1 Deviation of APIs used by CanTrcv

Within these functions recalculation of the given CanTrcvIndex has to be performed so that the local index of the driver matches the global index of the CanIf.

Affected requirements: CanTrcv086, CanTrcv222, CanTrcv239, CanTrcv238

8.2.2 CanIf_CheckTrcvWakeFlagIndication is always called

According to SWS [1], `Vc>[T] X IgXkLV ; aV >cY XVi dc` shall be called only if a wakeup was detected. As this would lock CanSm, it was decided to call this notification function in every case, even if no wakeup was detected. If a wakeup was detected, `: X BTH iLV e:k ci` is called by the transceiver driver.

Affected requirements: CanTrcv238

8.2.3 No re-initialization will occur if POR is set in CanTrcv_SetOpMode

According to SWS [1], the transceiver driver shall perform a re-initialization if POR is set when entering normal mode. This situation can only happen if V_s and V_{cc} drops below the lower boundary and recovers again while MCU is still active. In this case, it is always required to perform a complete re-initialization of the whole MCU as correct functionality cannot be ensured anymore. During this re-initialization, `VcIgXkT>c i` has to be called that reinitializes the underlying transceiver hardware.

Affected requirements: CanTrcv221

8.2.4 Const removed from API CanTrcv_GetTrcvSystemData

According to SWS [1], the API `VcIgXkT<iIgXkHnh>bViV` shall have a parameter `IgXkHnhViV` of type `Xdchi`. As this parameter has to be filled by the transceiver driver it was decided to remove this `Xdchi` qualifier.

Affected requirements: CanTrcv152

8.2.5 Unused BSWMD parameters

According to SWS [1], the parameters `VcIgXkHE>dbbGig` and `VcIgXkHE>dbbIbdi` shall have the multiplicity 1. As the SWS does not describe where to use these values, it was decided to set multiplicity to 0..1 so they do not have to be used.

Affected requirements: CanTrcv172, CanTrcv171

8.2.6 Modified return value of CanTrcv_CB_WakeupByBus

According to SWS [1] the API `CanTrcv_CB_WakeupByBus` shall return `E_OK` if a wakeup was detected. As SPI can be asynchronous the return value was modified so that `E_OK` means that the request to check for Wakeups has been acknowledged. The actual completion of this request may be done in `MainFunction`.

9 Glossary and Abbreviations

9.1 Glossary

Term	Description
GENy	Generation tool for CANbedded and MICROSAR components

Table 9-1 Glossary

9.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
DEM	Diagnostic Event Manager
DET	Development Error Tracer
ECU	Electronic Control Unit
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
SRS	Software Requirement Specification
SWC	Software Component
SWS	Software Specification
WUP	Wakeup Pattern
WUF	Wakeup Frame
AUTOSAR3	AUTOSAR R3.2 Rev1 + Rev2
AUTOSAR4	AUTOSAR R4.0 Rev3
POR	Power-on reset
MCU	Microcontroller Unit
SPI	Serial Peripheral Interface

Table 9-2 Abbreviations

10 Contact

Visit our website for more information on

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

www.vector-informatik.com