Document Title	Specification of CAN Transceiver Driver
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	071
Document Classification	Standard

Document Version	1.3.0
Document Status	Final
Part of Release	3.0
Revision	7

Document Change History

Date	Version	Changed by	Change Description
15.09.2010	1.3.0	AUTOSAR Administration	• Explanation added to chapter 7.4 • Updated CanTrcv150 • Legal disclaimer revised
05.12.2007	1.2.0	AUTOSAR Administration	• Changed API name CanIf_TrcvWakeupByBus to CanIf_SetWakeupEvent • New error code CANTRCV_E_PARAM_TRCV_WAKEUP_MODE has been added. • Output parameter in the API's CanTrcv_GetOpMode, CanTrcv_GetBusWuReason and CanTrcv_GetVersionInfo is changed to pointer type. • API CanTrcv_CB_WakeupByBus has been modified • Document meta information extended • Small layout adaptations made

30.01.2007	1.1.0	AUTOSAR Administration	• CAN transceiver driver is below CAN interface. All API access from higher layers are routed through CAN interface. • One CAN transceiver driver used per CAN transceiver hardware type. For different CAN transceiver hardware types different CAN transceiver drivers are used. One CAN transceiver driver supports all CAN transceiver hardware of same type • Legal disclaimer revised • Release Notes added • “Advice for users” revised • “Revision Information” added
16.05.2006	1.0.0	AUTOSAR Administration	Initial release

Disclaimer

Advice for users

AUTOSAR Specification Documents may contain exemplary items (exemplary reference models, "use cases", and/or references to exemplary technical solutions, devices, processes or software).

Any such exemplary items are contained in the Specification Documents for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such Specification Documents, nor any later documentation of AUTOSAR conformance of products actually implementing such exemplary items, imply that intellectual property rights covering such exemplary items are licensed under the same rules as applicable to the AUTOSAR Standard.

Table of Content

1 Introduction

This specification specifies functionality, API and configuration of module CAN transceiver driver. The driver is responsible to handle the CAN transceiver hardware chips on an ECU.

The CAN bus transceiver is a hardware device, which mainly transforms the logical 1/0 signals of the µC ports or the information given by SPI connection to the bus compliant electrical voltage, current and timing.

Within an automotive environment, there are mainly three different CAN bus physics used. These physics are ISO11898 for high-speed CAN (up to 1Mbd), ISO11519 for low-speed CAN (up to 125kBd) and SAE J2411 for single-wire CAN.

In addition, the transceivers are often able to detect electrical malfunctions like wiring issues, ground offsets or transmission of too long dominant signals. Depending on the interface, they flag the detected error summarized by a single port pin or very detailed via SPI.

Some transceivers also support power supply control and wake up via the bus. A lot of different wake up/sleep and power supply concepts are usual on the market.

Latest developments are so called system basis chips (SBC) where not only the CAN but also power supply control and advanced watchdogs are implemented in one housing and are controlled via one interface (e.g. via SPI).

1.1 Goal of CAN transceiver driver

The target of this document is to specify interfaces and behavior which are applicable to most current and future CAN transceiver hardware chips and for nearly all use cases.

The CAN transceiver driver abstracts used CAN transceiver hardware. It offers a hardware independent interface to the higher layers. It abstracts also from ECU layout by using APIs of MCAL layer to access CAN transceiver hardware.

1.2 Explicitly uncovered CAN transceiver functionality

Some CAN bus transceivers offer additional functionality as, for example, ECU self test or error detection capability for diagnostics.

ECU self test and error detection are not defined within AUTOSAR and requiring such functionality in general would lock out most currently used transceiver hardware chips. Therefore, features like “ground shift detection”, “selective wake up”, “slope control” and others are not supported.

1.3 System basis chips

System basis chips (SBCs) are not supported by AUTOSAR.

1.4 Single wire CAN transceivers according SAE J2411

Single wire CAN according SAE J2411 is not supported by AUTOSAR.

2 Acronyms and abbreviations

Abbreviation	*Description*
ComM	Communication Manager
Dem	Diagnostic Event Manager
Det	Development Error Tracer
Dio	Digital input output, one of the SPAL SW modules
EB	Externally buffered channels. Buffers containing data to transfer are outside the SPI Handler/Driver.
EcuM	ECU State Manager
Frt	Free Running Timer
IB	Internally buffered channels. Buffers containing data to transfer are inside the SPI Handler/Driver.
ISR	Interrupt Service Routine
MCAL	Micro Controller Abstraction Layer
Port	Port, one of the SPAL SW modules
n/a	Not applicable
SBC	System Basis Chip; a device, which integrates e.g. CAN and/or LIN transceiver, watchdog and power control.
SPAL	Standard Peripheral Abstraction Layer
SPI Channel	A channel is a software exchange medium for data that are defined with the same criteria: configuration parameters, number of data elements with same size and data pointers (source & destination) or location. See specification of SPI driver for more details.
SPI Job	A job is composed of one or several channels with the same chip select. A job is considered to be atomic and therefore cannot be interrupted. A job has also an assigned priority. See specification of SPI driver for more details.
SPI Sequence	A sequence is a number of consecutive jobs to be transmitted. A sequence depends on a static configuration. See specification of SPI driver for more details.

3 Related documentation

3.1 Input documents

[1] List of Basic Software Modules
AUTOSAR_BasicSoftwareModules.pdf

[2] Layered Software Architecture
AUTOSAR_LayeredSoftwareArchitectur.pdf

[3] Specification of ECU Configuration
AUTOSAR_RS_ECU_Configuration.pdf

[4] General Requirements on Basic Software
AUTOSAR_SRS_General.pdf

[5] Specification of Specification of CAN Interface
AUTOSAR_SWS_CANInterface.pdf

[6] AUTOSAR Basic Software Module Description Template,
AUTOSAR_BSW_Module_Description.pdf

3.2 Related standards and norms

[7] ISO11898 – Road vehicles - Controller area network (CAN)

4 Constraints and assumptions

4.1 Limitations

CanTrcv098: The CAN bus transceiver hardware shall provide functionality and an interface which can be mapped to the operation mode model of the AUTOSAR CAN transceiver driver.

See also Chapter 7.1.

The used APIs of underlying drivers (SPI and DIO) shall be synchronous.

Implementiations of underlying drivers which does not support synchronous behaviour cannot be used together with CAN transceiver driver.

4.2 Applicability to car domains

This driver might be applicable in all car domains using CAN for communication.

5 Dependencies to other modules

Module	*Dependencies*
CanIf	All CAN transceiver drivers are arranged below CanIf.
ComM	ComM steers CAN transceiver driver communication modes via CanIf. Independent steering of each single CAN transceiver channel.
Det	Det gets development error information from CAN transceiver driver.
Dem	Dem gets production error information from CAN transceiver driver.
Dio	Dio module is used to access CAN transceiver hardware connected via ports.
EcuM	EcuM gets wake up event information from CAN transceiver driver via CanIf.
Frt	Free running timer
Icu	Icu module performs CAN transceiver hardware interrupts and calls appropriate callback function inside CAN transceiver driver.
SPI	SPI module is used to access CAN transceiver hardware connected via SPI.

5.1 File structure

5.1.1 Naming convention for transceiver driver implementation

CanTrcv070: In case different CAN transceiver hardware chips are used in one ECU, the function names of the different CAN transceiver drivers must be modified such that no two functions with the same names are generated. It is the responsibility of the user to take care that no two functions with the same names are configured. The names may be extended with a vendor ID or a type ID. Any combination of these extensions is possible.

5.1.2 Code file structure

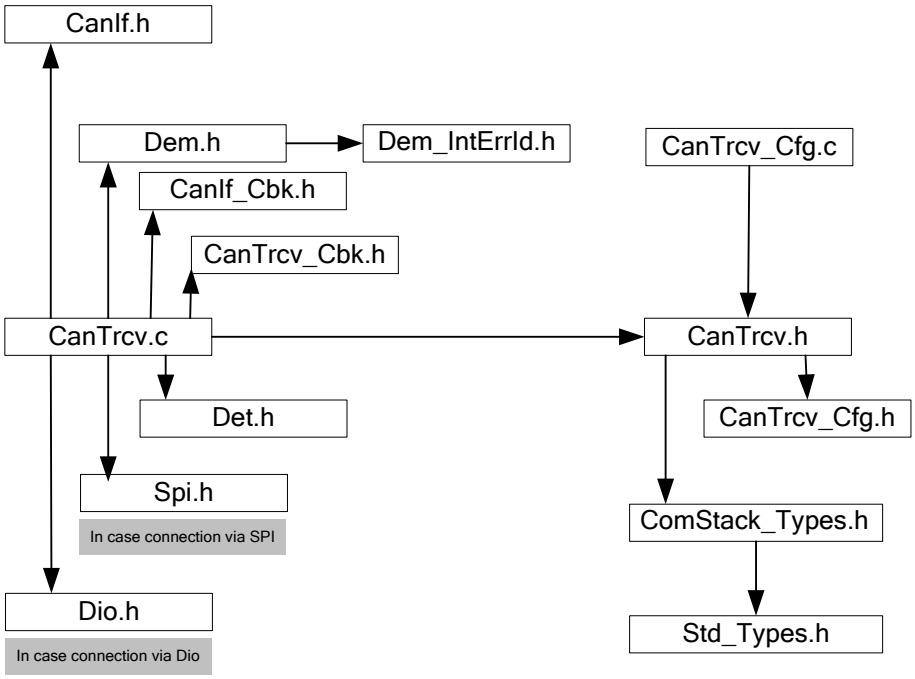
CanTrcv064: The naming convention is applied to all files of the CanTrcv module.

CanTrcv065: The CanTrcv module consists of the following files:

File name	*Requirements*	*Description*
CanTrcv.c	CanTrcv069	The implementation general c file. It does not contain interrupt routines.
CanTrcv.h	CanTrcv052	It contains only information relevant for other BSW modules (API). Differences in API depending in configuration are encapsulated.
CanTrcv_Cbk.h	CanTrcv071	CanTrcv_Cbk.h contains callback functions implemented in CanTrcv.c and called by other modules.
CanTrcv_Cfg.h	CanTrcv083	Pre compile time configuration parameter file. It's generated by the configuration tool.
CanTrcv_Cfg.c	CanTrcv062	Pre compile time configuration code file. It's generated by the configuration tool.

5.1.3 Header file structure

CanTrcv067:



CanTrcv068: For AUTOSAR standard data types, header file `Std_Types.h` is included.

CanTrcv061: The name of the compiler specific header file is `Compiler.h`. All mappings of not standardized keywords of compiler specific scope shall be placed and organized in this compiler specific type and keyword header.

CanTrcv063: The name of the platform specific header file is `Platform_Types.h`. All integer type definitions of target and compiler specific scope shall be placed and organized in this single type header.

6 Requirements Traceability

Document: AUTOSAR requirements on Basic Software, general

Requirement	***Satisfied by***
[BSW003] Version identification	CanTrcv021
[BSW00300] Module naming convention.	CanTrcv064
[BSW00301] Limit imported information	CanTrcv067
[BSW00302] Limit exported information.	CanTrcv052
[BSW00304] AUTOSAR integer data types	not applicable (general implementation requirement)
[BSW00305] Self-defined data types naming convention	not applicable (no self defined data types)
[BSW00306] Avoid direct use of compiler and platform specific keyword	not applicable (general implementation requirement)
[BSW00307] Naming convention for global variables	not applicable (general implementation requirement)
[BSW00308] Definition of global data	not applicable (general implementation requirement)
[BSW00309] Global read only data with read only constraint	not applicable (general implementation requirement)
[BSW00310] API naming convention	CanTrcv001, CanTrcv002, CanTrcv005, CanTrcv007, CanTrcv008, CanTrcv009, CanTrcv012, CanTrcv013
[BSW00312] Shared code shall be reentrant	not applicable (general implementation requirement)
[BSW00314] Separation of interrupt frames and services routines	CanTrcv069
[BSW00318] Format of module version numbers	CanTrcv021
[BSW00321] Enumeration of module version numbers	not applicable (general implementation requirement)
[BSW00323] API parameter checking	CanTrcv048
[BSW00325] Runtime of interrupt service routines	not applicable (CAN transceiver driver implements no ISRs)
[BSW00326] Transition from ISRs to OS tasks	not applicable (no such transitions are performed)
[BSW00327] Error values naming convention	CanTrcv050
[BSW00328] Avoid duplication of code	not applicable (general implementation requirement)
[BSW00329] Avoidance of generic interfaces	CanTrcv001, CanTrcv002, CanTrcv005, CanTrcv007, CanTrcv008, CanTrcv009, CanTrcv012, CanTrcv013
[BSW00330] Use of macros and inline functions	not applicable (general implementation requirement)
[BSW00331] Separation of error and status values	not applicable (no such values defined)
[BSW00333] Documentation of callback function context	not applicable (general documentation requirement)
[BSW00334] Provision of XML file	not applicable (general implementation requirement)
[BSW00335] Status values naming convention	not applicable
[BSW00336] Shut down interface	not applicable (no need for such interfaces)
[BSW00337] Classification of errors	CanTrcv057
[BSW00338] Detection and reporting of development errors	CanTrcv040,CanTrcv090
[BSW00339] Reporting of production relevant error status	CanTrcv024,CanTrcv058
[BSW00341] Mircocontroller compatibility documentation	not applicable

	(general documentation requirement)
[BSW00342] Use of source code and object code	not applicable (general implementation requirement)
[BSW00343] Specification and configuration of time	CanTrcv090
[BSW00344] Reference to link time configuration	not applicable (only pre compile time configuration supported)
[BSW00345] Pre compile time configuration	CanTrcv062, CanTrcv083
[BSW00346] Basic set of module files	CanTrcv065
[BSW00347] Naming separation of different instances of BSW drivers	CanTrcv016, CanTrcv070
[BSW00348] Standard type header	CanTrcv068
[BSW00350] Development error detection keyword	CanTrcv023, CanTrcv090
[BSW00353] Platform specific type header	CanTrcv063
[BSW00355] Do not redefine AUTOSAR integer data types	not applicable (general implementation requirement)
[BSW00357] Standard API return type	CanTrcv002
[BSW00358] Return type of init() functions	CanTrcv001
[BSW00359] Return type of callback functions	CanTrcv012
[BSW00360] Parameters of callback functions	CanTrcv012
[BSW00361] Compiler specific language extension header	CanTrcv061
[BSW00369] Do not return development error codes via API	CanTrcv001, CanTrcv002, CanTrcv005, CanTrcv007, CanTrcv008, CanTrcv009, CanTrcv012, CanTrcv013
[BSW00370] Separation of callback interfaces from API	CanTrcv071
[BSW00371] Do not pass function pointers via API	CanTrcv001, CanTrcv002, CanTrcv005, CanTrcv007, CanTrcv008, CanTrcv009, CanTrcv012, CanTrcv013
[BSW00373] Main processing function naming convention	CanTrcv013
[BSW00374] Module vendor identification	CanTrcv021
[BSW00375] Notification of wake-up reason	CanTrcv012
[BSW00376] Return type and parameters of main functions	CanTrcv013
[BSW00377] Module specific API return types	CanTrcv005, CanTrcv007
[BSW00378] AUTOSAR boolean type	not applicable (general implementation requirement)
[BSW00379] Module identification	CanTrcv021
[BSW00380] Separate C file for configuration parameters	CanTrcv062
[BSW00381] Separate configuration H file for pre compile time parameters	CanTrcv083
[BSW00383] List dependencies of configuration elements	not applicable (general documentation requirement)
[BSW00384] List dependencies to other modules	not applicable (general documentation requirement)
[BSW00385] List possible error notifications	CanTrcv050
[BSW00386] Configuration for detecting an error	CanTrcv050
[BSW00387] Specify the configuration class of callbacks	CanTrcv012
[BSW00388] Introduce containers	CanTrcv090, CanTrcv091, CanTrcv092, CanTrcv093, CanTrcv094, CanTrcv095
[BSW00389] Container shall have names	CanTrcv090, CanTrcv091, CanTrcv092, CanTrcv093, CanTrcv094, CanTrcv095
[BSW00390] Parameter content unique within the module	CanTrcv090, CanTrcv091, CanTrcv092, CanTrcv093, CanTrcv094, CanTrcv095
[BSW00391] Parameters shall have unique names	CanTrcv090, CanTrcv091, CanTrcv092, CanTrcv093, CanTrcv094, CanTrcv095
[BSW00392] Parameters shall have unique types	CanTrcv090, CanTrcv091, CanTrcv092, CanTrcv093, CanTrcv094, CanTrcv095
[BSW00393] Parameters shall have a range	CanTrcv090, CanTrcv091, CanTrcv092, CanTrcv093, CanTrcv094, CanTrcv095
[BSW00394] Specify the scope of the parameters	CanTrcv090, CanTrcv091, CanTrcv092,

	CanTrcv093, CanTrcv094, CanTrcv095
[BSW00395] List the required parameters (per parameter)	CanTrcv091, CanTrcv092, CanTrcv093, CanTrcv094, CanTrcv095
[BSW00396] Configuration classes	CanTrcv017
[BSW00397] Pre compile time parameters	CanTrcv062, CanTrcv083
[BSW00398] Link time parameters	not applicable (only pre compile time configuration supported)
[BSW00399] Loadable post build time parameters	not applicable (only pre compile time configuration supported)
[BSW004] Version check	not applicable (general implementation requirement)
[BSW00400] Selectable post build time parameters	not applicable (only pre compile time configuration supported)
[BSW00401] Documentation of multiple instances of configuration parameters	not applicable (general documentation requirement)
[BSW00402] Published information	CanTrcv021
[BSW00404] Reference to post build time configuration	not applicable (only pre compile time configuration supported)
[BSW00405] Reference to multiple configuratin sets	not applicable (only pre compile time configuration supported)
[BSW00406] Check module initialization	CanTrcv002, CanTrcv005, CanTrcv007, CanTrcv008, CanTrcv009, CanTrcv012, CanTrcv013
[BSW00407] Function to read out published parameters	CanTrcv008
[BSW00408] Configuration Parameter naming convention	CanTrcv090, CanTrcv091, CanTrcv092 CanTrcv093, CanTrcv094, CanTrcv095
[BSW00409] Header files for production code error	CanTrcv067
[BSW00410] Compiler switches shall have defined values	not applicable (general implementation requirement)
[BSW00411] Get version information keyword	CanTrcv090
[BSW00412] Separate H file for configuration parameters	CanTrcv083
[BSW00413] Accessing instances of BSW modules	CanTrcv016
[BSW00414] Parameters of init function	CanTrcv001
[BSW00415] User dependent include files	CanTrcv052
[BSW00416] Sequence of initialization	not applicable (this is out of CAN transceiver driver's scope)
[BSW00417] Preporting of error events by non basic software	not applicable (Requirement concerns application components only)
[BSW00419] Separate C file for pre compile time configuration parameters	CanTrcv062
[BSW00420] Production relevant error event rate detection	not applicable (it's an Dem requirement)
[BSW00421] Reporting of production relevant error events	CanTrcv058
[BSW00422] Debouncing of production relevant error status	not applicable (it's an Dem requirement)
[BSW00423] Usage of SW C template to describe BSW modules with AUTOSAR interfaces	not applicable (general implementation requirement)
[BSW00424] BSW main processing function task allocation	CanTrcv013
[BSW00425] Trigger condition for schedulable objects	CanTrcv090
[BSW00426] Exclusive areas in BSW modules	not applicable (CAN transceiver driver is part of ECU abstraction layer)

[BSW00427] ISR description for BSW modules	not applicable (No such areas or function in CAN transceiver driver)
[BSW00428] Execution order dependencies of main processing function	CanTrcv013
[BSW00429] Restricted BSW OS functionality access	not applicable (general implementation requirement)
[BSW00431] The BSW scheduler module implements task bodies	not applicable (requirement concerns BSW scheduler module)
[BSW00432] Modules should have separate main processing functions for read/receive and write/transmit data path	not applicable (CAN transceiver driver does not propagate data)
[BSW00433] Calling of main processing functions	not applicable (requirement concerns BSW scheduler module)
[BSW00434] The schedule module shall provide an API for exclusive areas	not applicable (requirement concerns BSW scheduler module)
[BSW005] No hard coded horizontal interfaces within MCAL	not applicable (CAN transceiver driver is part of ECU abstraction layer)
[BSW006] Platform independency	not applicable (general implementation requirement)
[BSW007] HIS Misra C	not applicable (general implementation requirement)
[BSW009] Module user documentation	not applicable (general documentation requirement)
[BSW010] Memory resource documentation	not applicable (general documentation requirement)
[BSW101] Initialization interface	CanTrcv001
[BSW158] Separation of configuration from implementation	CanTrcv065
[BSW159] Tool-based configuration	
[BSW160] Human readable configuration data	CanTrcv090, CanTrcv091, CanTrcv092 CanTrcv093, CanTrcv094, CanTrcv095
[BSW161] Microcontroller abstraction	not applicable (CAN transceiver driver is part of ECU abstraction layer)
[BSW162] ECU layout abstraction	
[BSW164] Implementation of interrupt service routines	not applicable (CAN transceiver driver implements no ISRs)
[BSW167] Static configuration checking	
[BSW168] Diagnostic Interface of SW components	not applicable (CAN transceiver driver has no such needs)
[BSW170] Data for reconfiguration of AUTOSAR SW components	
[BSW171] Configurability of optional functionality	CanTrcv012, CanTrcv013
[BSW172] Compatibility and documentation of scheduling strategy	CanTrcv001, CanTrcv013, CanTrcv090 CanTrcv091, CanTrcv098, CanTrcv099

Document: AUTOSAR requirements on Basic Software, cluster CAN

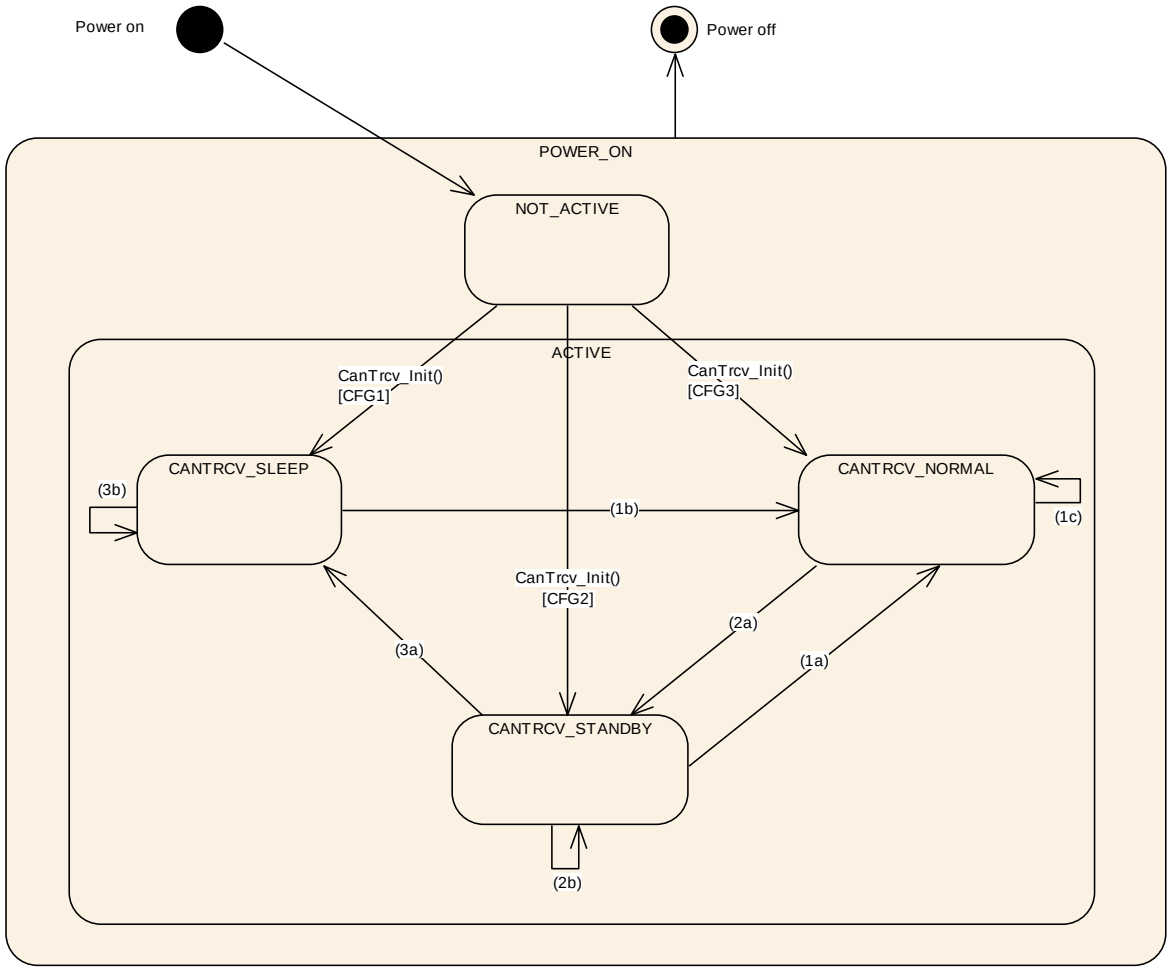
Requirement	***Satisfied by***
[BSW01090] Configuration Data for CAN Bus Transceiver	CanTrcv090, CanTrcv091, CanTrcv092 CanTrcv093, CanTrcv094, CanTrcv095
[BSW01091] Support for more than one CAN transceiver. Only pre-compile time configuration	CanTrcv002, CanTrcv005, CanTrcv007, CanTrcv009, CanTrcv012, CanTrcv016,

allowed.	CanTrcv017
[BSW01092] Configuration of bus operation mode after initialization for each CAN bus transceiver	CanTrcv091
[BSW01095] Configuration “Notification for Wakeup by bus”	CanTrcv091
[BSW01096] API to initialize the CAN bus transceiver driver	CanTrcv001
[BSW01097] CAN bus transceiver driver API shall be synchronous	CanTrcv001, CanTrcv002, CanTrcv005, CanTrcv007, CanTrcv009, CanTrcv012, CanTrcv013
[BSW01098] API to request operation mode Standby	CanTrcv002, CanTrcv055
[BSW01099] API to request operation mode Sleep	CanTrcv002, CanTrcv055
[BSW01100] API to request operation mode Normal	CanTrcv002, CanTrcv055
[BSW01101] API to read out current operation mode	CanTrcv005
[BSW01103] API to read out wake up reason	CanTrcv007
[BSW01106] Wake up by bus notification to upper layer	CanTrcv066
[BSW01107] Support for wake up during sleep transition	CanTrcv012
[BSW01109] CAN bus transceiver driver must check transceiver control	CanTrcv001, CanTrcv002, CanTrcv005, CanTrcv007, CanTrcv009, CanTrcv012, CanTrcv013
[BSW01110] Handle timing requirements of transceiver	CanTrcv001, CanTrcv002, CanTrcv005, CanTrcv007, CanTrcv009, CanTrcv012, CanTrcv013
[BSW01115] Support API for enable/disable and clear wake up event	CanTrcv009
[BSW01138] Wake up by bus callback for lower layers	CanTrcv012
BSW01108] Safe system start up and shut down for CAN bus transceiver driver	CanTrcv001, CanTrcv002

7 Functional specification

7.1 CAN transceiver driver operation modes

CanTrcv055: The CanTrcv module shall implement the state diagram shown below independently for each configured channel.



Legend:

1a, 1b, 1c: CanTrcv_SetOpMode(CANIF_TRCV_MODE_NORMAL)
2a, 2b: CanTrcv_SetOpMode(CANIF_TRCV_MODE_STANDBY)
3a, 3b: CanTrcv_SetOpMode(CANIF_TRCV_MODE_SLEEP)

The main idea behind this diagram is to support a lot of up to now available CAN bus transceivers in a common model view. Depending on the CAN transceiver hardware, the model may have one or two states more than necessary for a given CAN transceiver hardware but this will clearly decouple the ComM and EcuM from the used hardware.

The function `CanTrcv_Init` causes a state change to either CANTRCV_SLEEP, CANTRCV_NORMAL or CANTRCV_STANDBY. This depends on the configuration and is independently configurable for each channel.

State	*Description*
POWER_ON	ECU is fully powered.
NOT_ACTIVE	State of CAN transceiver hardware depends on ECU hardware and on Dio and Port driver configuration. CAN transceiver driver is not initialized and therefore not active.
ACTIVE	The function `CanTrcv_Init` has been called. It carries CAN transceiver driver to active state. Depending on configuration CAN transceiver driver enters state CANTRCV_SLEEP, CANTRCV_STANDBY or CANTRCV_NORMAL.
CANTRCV_NORMAL	Full bus communication. If CAN transceiver hardware controls ECU power supply, ECU is fully powered. The CAN transceiver driver detects no further wake up information.
CANTRCV_STANDBY	No communication is possible. ECU is still powered if CAN transceiver hardware controls ECU power supply. A transition to CANTRCV_SLEEP is only valid from this mode. A wake up by bus or by a local wake up event is possible.
CANTRCV_SLEEP	No communication is possible. ECU may be unpowered depending on responsibility to handle power supply. A wake up by bus or by a local wake up event is possible.

If a CAN transceiver driver covers more than one CAN channel, all channels are either in state NOT_ACTIVE or in state ACTIVE. In state ACTIVE each channel may be in a different sub state.

7.1.1 Operation mode switching

A mode switch is requested with a call to the function `CanTrcv_SetOpMode`.

CanTrcv150 A mode switch request to the current mode is allowed and shall not lead to an error, even if DET is enabled.

7.2 CAN transceiver hardware operation modes

The CAN transceiver hardware may support more mode transitions than shown in the state diagram above. The dependencies and the recommended implementations behaviour are explained in this chapter.

It is up to the implementation to decide which CAN transceiver hardware state is covered by which CAN transceiver driver software state. An implementation has to guarantee that the whole functionality of the described CAN transceiver driver software state is realized by the implementation.

7.2.1 Example for temporary “Go-To-Sleep” mode

The mode often referred to as "Go-to-sleep” is a temporary mode when switching from Normal to Sleep. The driver encapsulates such a temporary mode within one of the CAN transceiver driver software states. In addition, the CAN transceiver driver switches first from Normal to Standby and then with an additional API call from Standby to Sleep.

7.2.2 Example for “PowerOn/ListenOnly” mode

The mode often refered to as “PowerOn“ or “ListenOnly” is a mode where the CAN transceiver hardware is only able to receive messages but not able to send messages. Also, transmission of the acknowledge bit during reception of a message is supressed. This mode is not supported because it is outside of the CAN standard and not supported by all CAN transceiver hardware chips.

7.3 CAN transceiver wake up types

There are three different scenarios which are often called wake up:

Scenario 1:

- MCU is not powered.
- Parts of ECU including CAN transceiver hardware are powered.
- The considered CAN transceiver channel is in SLEEP mode.
- A wake up event on CAN is detected by CAN transceiver hardware.
- The CAN transceiver hardware causes powering of MCU.

In terms of AUTOSAR, this is kept as a cold start and NOT as a wake up.

Scenario 2:

- MCU is in low power mode.
- Parts of ECU including CAN transceiver hardware are powered.
- The considered CAN transceiver channel is in STANDBY mode.
- A wake up event on CAN is detected by CAN transceiver hardware.
- The CAN transceiver hardware causes a SW interrupt for waking up.

In terms of AUTOSAR, this is kept as a wake up of the CAN channel and of the MCU.

Scenario 3:

- MCU is in full power mode.
- At least parts of ECU including CAN transceiver hardware are powered.
- The considered CAN transceiver channel is in STANDBY mode.
- A wake up event on CAN is detected by CAN transceiver hardware.
- The CAN transceiver hardware either causes a SW interrupt for waking up or is polled cyclically for wake up events.

In terms of AUTOSAR, this is kept as a wake up of a CAN channel.

7.4 CAN transceiver wake up modes

CAN transceiver driver offers three wake up modes:

CanTrcv090: NO mode

In mode NO, no wake ups are generated by CAN transceiver driver. This mode is supported by all CAN transceiver hardware types.

CanTrcv091: POLLING mode

In mode POLLING, wake ups generated by CAN transceiver driver may cause CAN channel wake ups. In this mode, no MCU wake ups are possible. This mode presumes a support by used CAN transceiver hardware type. Wake up mode POLLING requires callback function `CanTrcv_CB_WakeupByBus` and main function `CanTrcv_Main` to be present in source code and and main function `CanTrcv_Main` to be called by CanIf.

CanTrcv092: ISR mode

In mode ISR, wake ups generated by CAN transceiver driver may cause CAN channel wake ups and MCU wake ups. This mode pressumes a support by used CAN transceiver hardware type. Wake up mode ISR requires callback function `CanTrcv_CB_WakeupByBus` to be present in source code.

The selection of the wake up mode is done by the configuration parameter `CanTrcvWakeUpSupport`. The support of wake ups may be switched on and off for each CAN transceiver channel individually by the configuration parameter `CanTrcvWakeupByBusUsed`.

Implementation Hint:

If a CAN transceiver needs a specific state transition (e.g. CANTRCV_SLEEP -> CANTRCV_NORMAL) initiated by the software after detection of a wake-up, this may be accomplished by the CanTrcv module, during the execution of `CanTrcv_CB_WakeupByBus`. This behaviour is implementation specific.

It has to be assured by configuration of modules, which are involved in wake-up process (EcuM, CanIf, ICU etc…) that `CanTrcv_CB_WakeupByBus` is called, when a transceiver needs a specific state transition.

7.5 Error classification

Values for production code event IDs are assigned externally by the configuration of the Dem. They are published in the file `Dem_IntErrId.h` and included via `Dem.h`.

CanTrcv057: Development error values are of type uint8.

CanTrcv050:

Type or error	*Relevance*	*Related error code*	*Value [hex]*
API called with wrong parameter for CAN network	Development	CANTRCV_E_INVALID_CAN_NETWORK	1
API called with with null pointer parameter	Development	CANTRCV_E_PARAM_POINTER	2
API service used without initialization	Development	CANTRCV_E_UNINIT	11
API service called in wrong transceiver operation mode	Development	CANTRCV_E_TRCV_NOT_STANDBY CANTRCV_E_TRCV_NOT_NORMAL	21 22
API service called with invalid parameter for TrcvWakeupMode	Development	CANTRCV_E_PARAM_TRCV_WAKEUP_MODE	23
No/incorrect communication to transceiver.	Production	CANTRCV_E_NO_TRCV_CONTROL	*

* Assignment is done in a header file of module Dem.

7.6 Error detection

CanTrcv023: The detection of all development errors is configurable (ON/OFF) at pre compile time. The switch `CanTrcvDevErrorDetect` shall activate or deactivate the detection of all development errors.

CanTrcv048: If the `CanTrcvDevErrorDetect` switch is enabled API parameter checking is enabled. The detailed description of the detected errors can be found in chapter 0.

CanTrcv058: The detection of production code errors cannot be switched off.

CanTrcv040: Detected development errors will be reported to the error hook of the Development Error Tracer (Det) if the pre-processor switch `CanTrcvDevErrorDetect` is set.

CanTrcv024: Production errors shall be reported to Diagnostic Event Manager (Dem). Only error cases are reported to the Dem.

7.7 Preconditions for driver initialization

CanTrcv099: The environment of the CanTrcv module shall make sure that all necessary BSW drivers (used by the CanTrcv module) have been initialized and are usable before `CanTrcv_Init` is called.

The CAN bus transceiver driver uses drivers for SPI, Dio and/or Icu to control the CAN bus transceiver hardware. Thus, these drivers must be available and ready to operate before the CAN bus transceiver driver is initialized.

The CAN transceiver driver may have timing requirements for the initialization sequence and the access to the transceiver device which must be fulfilled by these used underlaying drivers.

The timing requirements might be that

1) The call of the CAN bus transceiver driver initialization has to be performed very early after power up to be able to read all necessary information out of the transceiver hardware in time for all other users within the ECU.
2) The runtime of the used underlying services is very short and synchronous to enable the driver to keep his own timing requirements limited by the used hardware device.
3) The runtime of the driver may be enlarged due to some hardware devices requiring the port pin level to be valid for e.g. 50µs before changing it again to reach a specific state (e.g. sleep).

7.8 Instance concept

CanTrcv016: For each different CAN transceiver hardware type, an ECU has one CAN transceiver driver instance. One instance serves all CAN transceiver hardware of same type.

7.9 Wait states

For changing operation modes, the CAN transceiver hardware may have to perform wait states.

CanTrcv138: The module CanTrcv shall perform wait states by accessing the module Frt.

The API of the Frt module is currently not specified. Thus, no closer description in this specification is possible. Access to the Frt module is up to this future development.

8 API specification

8.1 Imported types

In this chapter all types included from the following files are listed:

CanTrcv084:

Header file	*Imported Type*
Dem_Types.h	Dem_EventIdType
Spi_Types.h	Spi_NumberOfDataType
	Spi_SequenceType
	Spi_DataType
	Spi_ChannelType
	Spi_StatusType
Dio_Types.h	Dio_LevelType
	Dio_ChannelGroupType
	Dio_ChannelType
	Dio_PortType
	Dio_PortLevelType
CanIf_Types.h	CanIf_TransceiverModeType
	CanIf_TrcvWakeupReasonType
	CanIf_TrcvWakeupModeType
Std_Types.h	Std_ReturnType
	Std_VersionInfoType

8.2 Type definitions

CanTrcv does not define types.

8.3 Function definitions

8.3.1 CanTrcv_Init

CanTrcv001:

Service name:	CanTrcv_Init

Syntax:	`void CanTrcv_Init(`
	
	`)`

Service ID[hex]:	0x00
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (in-out):	None
Parameters (out):	None
Return value:	None
Description:	Initializes the CanTrcv module.

CanTrcv100: The function `CanTrcv_Init` shall set the CAN transceiver hardware to the state configured by the configuration parameter `CanTrcvInitState`.

Note that in the time span between power up and the call to `CanTrcv_Init`, the CAN transceiver hardware may be in a different state. This depends on hardware and SPAL driver configuration.

The initialization sequence after reset (e.g. power up) is a critical phase for the CAN transceiver driver.

This API also validates whether there has been a wake up due to transceiver activity and if TRUE, reporting will be done to CanIf by calling CanIf_setWakeupEvent, which in turns reports to EcuM via API EcuM_SetWakeupEvent.

See also requirement CanTrcv099.

CanTrcv113: If there is no/incorrect communication towards the transceiver, the function `CanTrcv_Init` shall report the production error `CANTRCV_E_NO_TRCV_CONTROL`. For Eg., there are different transceiver types and different access ways (port connection, SPI). This production error should be signaled if you detect any miscommunication with your hardware. Depending on connection type and depending on your transceiver hardware you may not run in situations where you have to signal this error.

8.3.2 CanTrcv_SetOpMode

CanTrcv002:

Service name:	CanTrcv_SetOpMode	
Syntax:	`Std_ReturnType CanTrcv_SetOpMode( CanIf_TransceiverModeType OpMode, uint8 CanNetwork )`	
Service ID[hex]:	0x01	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	OpMode	This parameter contains the desired operating mode
	CanNetwork	CAN network to which API call has to be applied.
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: will be returned if the transceiver state has been changed to the requested mode. E_NOT_OK: will be returned if the transceiver state change has failed or the parameter is out of the allowed range. The previous state has not been changed.
Description:	Sets the mode of the channel CanNetwork to the value OpMode.	

CanTrcv102: The function `CanTrcv_SetOpMode` shall switch the internal state of channel CanNetwork to the value of the parameter OpMode which can be CANTRCV_NORMAL, CANTRCV_STANDBY or CANTRCV_SLEEP.

CanTrcv103: The user of the CanTrcv module shall call the function `CanTrcv_SetOpMode` with OpMode == CANTRCV_STANDBY or CANTRCV_NORMAL, if the channel CanNetwork is in mode CANTRCV_NORMAL.

CanTrcv104: The user of the CanTrcv module shall call the function `CanTrcv_SetOpMode` with OpMode == CANTRCV_ SLEEP or CANTRCV_STANDBY, if the channel CanNetwork is in mode CANTRCV_STANDBY.

This API is applicable to each transceiver with each value for parameter `CanTrcv_SetOpMode` regardless of whether the transceiver hardware supports these modes or not. This is to simplify the view of the CanIf to the assigned bus.

CanTrcv105: If the requested mode is not supported by the underlying transceiver hardware, the function `CanTrcv_SetOpMode` shall return E_NOT_OK.

The number of supported busses is set up in the configuration phase.

CanTrcv114: If there is no/incorrect communication to the transceiver, the function `CanTrcv_SetOpMode` shall report production error `CANTRCV_E_NO_TRCV_CONTROL` and return `E_NOT_OK`.

CanTrcv120: If development error detection for the module CanTrcv is enabled: If the function `CanTrcv_SetOpMode` is called with OpMode == CANTRCV_ STAND-

BY and the channel CanNetwork is not in mode CANTRCV_NORMAL or CANTRCV_ STANDBY, the function `CanTrcv_SetOpMode` shall raise the development error `CANTRCV_E_TRCV_NOT_NORMAL` and return `E_NOT_OK`.

CanTrcv121: If development error detection for the module CanTrcv is enabled: If the function `CanTrcv_SetOpMode` is called with OpMode == CANTRCV_ SLEEP and the channel CanNetwork is not in mode CANTRCV_STANDBY or CANTRCV_ SLEEP, the function `CanTrcv_SetOpMode` shall raise the development error `CANTRCV_E_TRCV_NOT_STANDBY` and return `E_NOT_OK`.

CanTrcv122: If development error detection for the module CanTrcv is enabled: If called before the CanTrcv module has been initialized, the function `CanTrcv_SetOpMode` shall raise the development error `CANTRCV_E_UNINIT` and return `E_NOT_OK`.

CanTrcv123: If development error detection for the module CanTrcv is enabled: If called with an invalid network number CanNetwork, the function `CanTrcv_SetOpMode` shall raise the development error `CANTRCV_E_INVALID_CAN_NETWORK` and return `E_NOT_OK`.

CanTrcv087: If development error detection for the module CanTrcv is enabled: If called with an invalid TrcvWakeupMode, the function `CanTrcv_SetOpMode` shall raise the development error CANTRCV_E_PARAM_TRCV_WAKEUP_MODE and return `E_NOT_OK`.

8.3.3 CanTrcv_GetOpMode

CanTrcv005:

Service name:	CanTrcv_GetOpMode	
Syntax:	`CanTrcv_OpModeType CanTrcv_GetOpMode( CanIf_TransceiverModeType OpMode, uint8 CanNetwork )`	
Service ID[hex]:	0x02	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	CanNetwork	CAN network to which API call has to be applied.
Parameters (in-out):	None	
Parameters (out):	OpMode	Pointer to operation mode of the bus the API is applied to.
Return value:	CanTrcv_OpModeType	E_OK: will be returned if the operation mode was detected. E_NOT_OK: will be returned if the operation mode was not detected.
Description:	Gets the mode of the channel CanNetwork and returns it in OpMode.	

CanTrcv106: The function `CanTrcv_GetOpMode` shall return the actual state of the CAN transceiver driver in the parameter OpMode.

See function `CanTrcv_Init` for the provided state after the CAN transceiver driver initialization till the first operation mode change request.

The number of supported busses is statically set in the configuration phase.

CanTrcv115: If there is no/incorrect communication to the transceiver, the function `CanTrcv_GetOpMode` shall report the production error `CANTRCV_E_NO_TRCV_CONTROL` and return `E_NOT_OK`.

CanTrcv124: If development error detection for the module CanTrcv is enabled: If called before the CanTrcv module has been initialized, the function `CanTrcv_GetOpMode` shall raise the development error `CANTRCV_E_UNINIT` and return `E_NOT_OK`.

CanTrcv129: If development error detection for the module CanTrcv is enabled: If called with an invalid network number CanNetwork, the function `CanTrcv_GetOpMode` shall raise the development error `CANTRCV_E_INVALID_CAN_NETWORK` and return `E_NOT_OK.`

CanTrcv132: If development error detection for the module CanTrcv is enabled: If called with OpMode==NULL, the function `CanTrcv_GetOpMode` shall raise the development error `CANTRCV_E_PARAM_POINTER` and return `E_NOT_OK`.

CanTrcv088: If development error detection for the module CanTrcv is enabled: If called with an invalid TrcvWakeupMode, the function `CanTrcv_SetOpMode` shall raise the development error CANTRCV_E_PARAM_TRCV_WAKEUP_MODE and return `E_NOT_OK`.

8.3.4 CanTrcv_GetBusWuReason

CanTrcv007:

Service name:	CanTrcv_GetBusWuReason	
Syntax:	`Std_ReturnType CanTrcv_GetBusWuReason( uint8 CanNetwork, CanIf_TrcvWakeupReasonType Reason )`	
Service ID[hex]:	0x03	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	CanNetwork	CAN network to which API call has to be applied.
Parameters (in-out):	None	
Parameters (out):	Reason	Pointer to wake up reason of the bus the API is applied to.
Return value:	Std_ReturnType	E_OK: will be returned if the wake up reason was detected. E_NOT_OK: will be returned if the wake up reason was not detected.
Description:	Gets the wakup reason for the channel CanNetwork and returns it in reason.	

CanTrcv107: The function `CanTrcv_GetBusWuReason` shall return the reason for the wake up that the CAN transceiver has detected in the parameter Reason

The ability to detect and differentiate the possible wake up reasons depends strongly on the CAN transceiver hardware.

Be aware if more than one bus is available, each bus may report a different wake up reason. E.g. if an ECU has CAN, a wake up by CAN may occur and the incoming data may cause an internal wake up for another CAN bus.
The CAN transceiver driver has a “per bus” view and does not vote the more important reason or sequence internally. The same may be true if e.g. one transceiver controls the power supply and the other is just powered or un-powered.

The number of supported busses is statically set in the configuration phase.

CanTrcv116: If there is no/incorrect communication to the transceiver, the function `CanTrcv_GetBusWuReason` shall report the production error `CANTRCV_E_NO_TRCV_CONTROL` and return `E_NOT_OK`.

CanTrcv125: If development error detection for the module CanTrcv is enabled: If called before the CanTrcv module has been initialized, the function `CanTrcv_GetBusWuReason` shall raise development error `CANTRCV_E_UNINIT` and return `E_NOT_OK`.

CanTrcv130: If development error detection for the module CanTrcv is enabled: If called with an invalid network number CanNetwork, the function `CanTrcv_GetBusWuReason` shall raise development error `CANTRCV_E_INVALID_CAN_NETWORK` and return `E_NOT_OK`.

CanTrcv133: If development error detection for the module CanTrcv is enabled: If called with Reason==NULL, the function `CanTrcv_GetBusWuReason` shall raise the development error `CANTRCV_E_PARAM_POINTER` and return `E_NOT_OK`.

8.3.5 CanTrcv_GetVersionInfo

CanTrcv008:

Service name:	CanTrcv_GetVersionInfo	
Syntax:	`void CanTrcv_GetVersionInfo( Std_VersionInfoType Versioninfo )`	
Service ID[hex]:	0x04	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	None	
Parameters (in-out):	None	
Parameters (out):	Versioninfo	Pointer to version information of this module.
Return value:	None	
Description:	Gets the version of the module and returns it in VersionInfo.	

CanTrcv108: The function `CanTrcv_GetVersionInfo` shall return the version information of this module. The version information includes:

- Module Id
- Vendor Id
- Vendor specific version numbers

CanTrcv109: The function `CanTrcv_GetVersionInfo` shall be pre-compile time configurable On/Off by the configuration parameter `CanTrcvGetVersionInfo`.

CanTrcv110: If source code for caller and callee of this function is available, the CanTrcv module should realize this function as a macro defined in the module's header file.

CanTrcv126: If development error detection for the module CanTrcv is enabled: If called before the CanTrcv has been initialized, the function `CanTrcv_GetVersionInfo` shall raise the development error `CANTRCV_E_UNINIT`.

CanTrcv134: If development error detection for the module CanTrcv is enabled: If called with VersionInfo==NULL, the function `CanTrcv_GetVersionInfo` shall raise development error `CANTRCV_E_PARAM_POINTER` and return `E_NOT_OK`.

8.3.6 CanTrcv_ SetWakeupMode

CanTrcv009:

Service name:	CanTrcv_SetWakeupMode	
Syntax:	`Std_ReturnType CanTrcv_SetWakeupMode( CanIf_TrcvWakeupModeType TrcvWakeupMode, uint8 CanNetwork )`	
Service ID[hex]:	0x05	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	TrcvWakeupMode	Requested transceiver wakeup reason
	CanNetwork	CAN network to which API call has to be applied.
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Will be returned, if the wakeup state has been changed to the requested mode. E_NOT_OK: Will be returned, if the wakeup state change has failed or the parameter is out of the allowed range. The previous state has not been changed.
Description:	Enables, disables or clears wake-up events of the channel CanNetwork.	

CanTrcv111: If the function `CanTrcv_SetWakeupMode is` called with TrcvWakupMode==CANIF_TRCV_WU_ENABLE and if the CanTrcv module has a stored wakeup event pending for the addressed bus, the CanTrcv module shall execute the notification within the API call or immediately after (depending on the implementation).

CanTrcv093: Disabled: If the function `CanTrcv_SetWakeupMode is` called with TrcvWakeupMode==CANIF_TRCV_WU_DISABLE, then the notifications for wakeup events are disabled on the addressed network. It is required by the transceiver device and the underlying communication driver to detect the wakeup events and store it internally in order to raise the event when the wakeup notification is enabled again.

CanTrcv094: Clear: If the function `CanTrcv_SetWakeupMode is` called with TrcvWakeupMode==CANIF_TRCV_WU_CLEAR, then a stored wakeup event is cleared on the addressed network. Clearing of wakeup events have to be used when the wake up notification is disabled to clear all stored wake up events under control of the higher layer.

CanTrcv095: The implementation can either enable or disable interrupt source for the wake up and also it may clear wake up events from the last communication cycle. If the interrupt is level triggered, a pending interrupt is automatically stored and raised after enabling the notification again. It is very important not to lose wake up events during the disabled period.

The number of supported busses is statically set in the configuration phase.

CanTrcv117: If there is no/incorrect communication to the transceiver, the function `CanTrcv_SetWakupMode` shall report the production error `CANTRCV_E_NO_TRCV_CONTROL` and return `E_NOT_OK`.

CanTrcv127: If development error detection for the module CanTrcv is enabled: If called before the CanTrcv has been initialized, the function

`CanTrcv_SetWakeupMode` shall raise development error `CANTRCV_E_UNINIT` and return `E_NOT_OK`.

CanTrcv131: If development error detection for the module CanTrcv is enabled: If called with an invalid network number CanNetwork, the function `CanTrcv_SetWakeupMode` shall raise development error `CANTRCV_E_INVALID_CAN_NETWORK` and return `E_NOT_OK`.

CanTrcv089: If development error detection for the module CanTrcv is enabled: If called with an invalid TrcvWakeupMode, the function `CanTrcv_SetOpMode` shall raise the development error CANTRCV_E_PARAM_TRCV_WAKEUP_MODE and return `E_NOT_OK`.

8.4 Scheduled functions

This chaper lists all functions provided by the CanTrcv module and called directly by the Basic Software Module Scheduler.

8.4.1 CanTrcv_MainFunction

CanTrcv013:

Service name:	CanTrcv_MainFunction

Syntax:	`void CanTrcv_MainFunction(`
	
	`)`

Service ID[hex]:	0x06
Timing:	FIXED_CYCLIC
Description:	Service to scan all busses for wake up events and perform these event.

The CAN bus transceiver driver may have cyclic jobs like polling for wake up events (if configured).

CanTrcv112: The `CanTrcv_MainFunction` shall scan all busses in STANDBY and SLEEP for wake up events and shall perform these events by calling the appropriate callback function.

According to [BSW00424], main processing functions shall be allocated by basic tasks. No special call order to be kept. Function is called within `CanIf_MainFunction_Wakeup`.

See configuration parameter `CanTrcvWakeUpSupport`.

CanTrcv128: If development error detection for the module CanTrcv is enabled: If called before the CanTrcv has been initialized, the function `CanTrcv_MainFunction` shall raise development error `CANTRCV_E_UNINIT`.

8.5 Call-back notifications

This chapter lists all functions provided by the CanTrcv module for lower layer modules.

CanTrcv139: The CanTrcv module shall provide function prototypes of the callback functions in the file `CanTrcv_Cbk.h`.

8.5.1 CanTrcv_CB_WakeupByBus

CanTrcv012:

Service name:	CanTrcv_CB_WakeupByBus
Syntax:	`Std_ReturnType CanTrcv_CB_WakeupByBus(` `  uint8 CanNetwork`

	)	
Service ID[hex]:	0x07	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	CanNetwork	CAN network to which API call has to be applied.
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK when a valid interrupt is detected E_NOT_OK when a no interrupt is detected
Description:	Service is called by underlying CANIF in case a wake up interrupt is detected.	

This API is called by the underlying SPAL CANIF in case a wake up interrupt is detected. The API validates wake up reason in terms whether it is a wake up or not.

Wake up by bus is always asynchronous to the transition to sleep and standby. In the worst case, wake up occurs during transition to sleep. In such a case, the driver shall create a wake up by bus notification immediately after the API calls to enter standby or sleep has finished. The EcuM must be able to handle the wake up event immediately after requesting the standby or sleep mode.

See configuration parameter `CanTrcvWakeUpSupport`.

The call context of this API is expected to be within the ISR handler of the underlying driver.

CanTrcv137: The function `CanTrcv_CB_WakeUpByBus` shall be callable in interrupt context.

CanTrcv135: If development error detection for the module CanTrcv is enabled: If called before the CanTrcv has been initialized, the function `CanTrcv_CB_WakeUpByBus` shall raise the development error `CANTRCV_E_UNINIT`.

CanTrcv136: If development error detection for the module CanTrcv is enabled: If called with an invalid parameter CanNetwork, the function `CanTrcv_CB_WakeUpByBus` shall raise the development error `CANTRCV_E_INVALID_CAN_NETWORK`.

8.6 Expected Interfaces

This chapter lists all functions the module CanTrcv requires from other modules.

8.6.1 Mandatory Interfaces

This chapter defines all interfaces which are required to fulfill the core functionality of the module.

CanTrcv085:

API function	*Description*
Dem_ReportErrorStatus	Reports errors to the DEM.

8.6.2 Optional Interfaces

This chapter defines all interfaces which are required to fulfill an optional functionality of the module.

CanTrcv066: `CanIf_SetWakeupEvent:` Called in operation modes such as sleep and standby. Not called if call to `CanTrcv_Goto_NormalMode` has caused wake up.CanIf_SetWakeupEvent is called in case of a mode change notification of the CAN transceiver.

CanTrcv086:

API function	*Description*
Spi_SetupEB	Service to setup the buffers and the length of data for the EB SPI Handler/Driver Channel specified.
Dio_ReadChannelGroup	This Service reads a subset of the adjoining bits of a port.
Dio_ReadChannel	Returns the value of the specified DIO channel.
Dio_WritePort	Service to set a value of the port.
Spi_ReadIB	Service for reading synchronously one or more data from an IB SPI Handler/Driver Channel specified by parameter.
Dio_WriteChannel	Service to set a level of a channel.
Spi_SyncTransmit	Service to transmit data on the SPI bus
Det_ReportError	Service to report development errors.
Spi_WriteIB	Service for writing one or more data to an IB SPI Handler/Driver Channel specified by parameter.
Dio_ReadPort	Returns the level of all channels of that port.
Spi_GetStatus	Service returns the SPI Handler/Driver software module status.
Dio_WriteChannelGroup	Service to set a subset of the adjoining bits of a port to a specified level.

1. The interfaces of the SPI module are used by the CanTrcv module if there are instances of the container CanTrcvSpiSequence.

2. The interfaces of the DIO module are used by the CanTrcv module if there are instances of the container CanTransceiverDIOAccess.

After finalization of the specification of the Frt module, the used Frt interfaces will be added. They will be used to perform wait states which are necessary for some transceiver types to perform.

8.6.3 Configurable interfaces

There are no configurable interfaces for CAN transceiver driver.

9 Sequence diagram

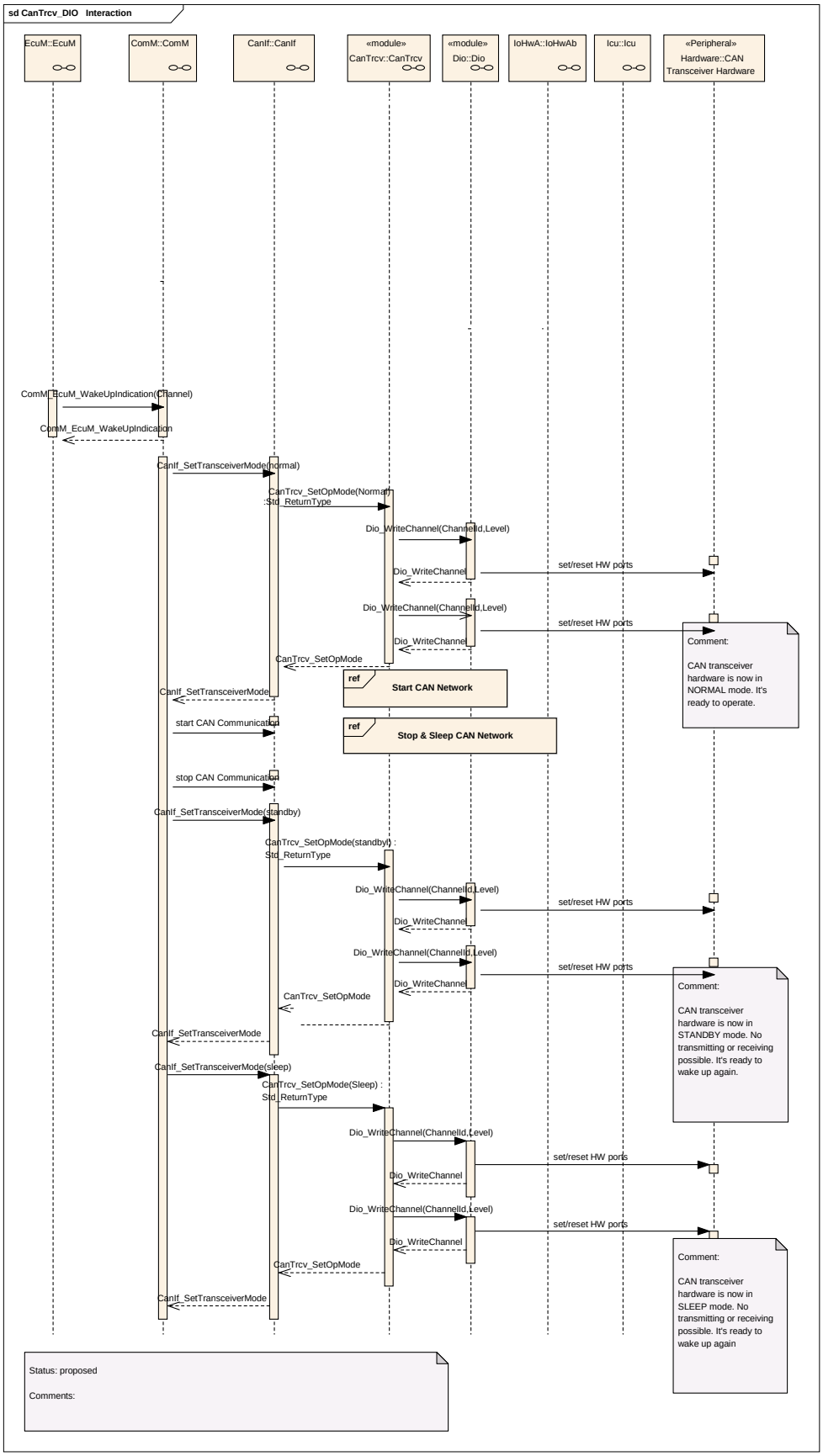
The focus of the following diagrams is on the interaction between the CAN transceiver driver and the BSW modules CanIf, ComM, EcuM, Icu and Dio. Depending on the CAN transceiver hardware, one or more calls to `Dio_WriteChannels` may be necessary.

Depending on the transceiver hardware, there may be wait states for some transitions necessary. For these wait states, a call to the Frt module will be performed. Due to the hardware dependency, these calls are not shown in the following sequence charts.

9.1 Wake up with valid validation

For all wakeup related sequence diagrams please refer to chapter 9 of ECU State Manager.

9.2 Interaction with DIO module



10 Configuration specification

In general this chapter defines configuration parameters and their clustering into containers. In order to support the specification Chapter 10.1 describes fundamentals.

Chapter 10.2 specifies the structure (containers) and the parameters of the module CanTrcv.

Chapter 0 specifies published information of the module CanTrcv.

10.1 How to read this chapter

In addition to this section, it is highly recommended to read the documents:

- AUTOSAR Layered Software Architecture [2]
- AUTOSAR ECU Configuration Specification [3]
  This document describes the AUTOSAR configuration methodology and the AUTOSAR configuration metamodel in detail.

The following is only a short survey of the topic and it will not replace the ECU Configuration Specification document.

10.1.1 Configuration class and configuration parameters

Configuration parameters define the variability of the generic part(s) of an implementation of a module. This means that only generic or configurable module implementation can be adapted to the environment (software/hardware) in use during system and/or ECU configuration.

The configuration of parameters can be achieved at different times during the software process: before compile time, before link time or after build time. In the following, the term “configuration class” (of a parameter) shall be used in order to refer to a specific configuration point in time.

10.1.2 Variants

Variants describe sets of configuration parameters. E.g., variant 1: only pre-compile time configuration parameters; variant 2: mix of pre-compile- and post build time-configuration parameters. In one variant a parameter can only be of one configuration class.

Each Variant must have a unique name which could be referenced to in later chapters. The maximum number of allowed variants is 3.

10.1.3 Containers

Containers structure the set of configuration parameters. This means:

- *all* configuration parameters are kept in containers.
- (sub-) containers can reference (sub-) containers. It is possible to assign a multiplicity to these references. The multiplicity then defines the possible number of instances of the contained parameters.

Configuration parameters shall be clustered into a container whenever

- the configuration parameters logically belong together (e.g. general parameters which are valid for the entire module NVRAM manager)
- the configuration parameters need to be instantiated (e.g. parameters of the memory block specification of the NVRAM manager – those parameters must be instantiated for each memory block)

10.2 Containers and configuration parameters

The following chapters summarize all configuration parameters. The detailed meanings of the parameters are described in preeding hapters.

10.2.1 Variants

Variant 1: Only pre compile time parameters.
Variant 2: Mix of pre compile- and link time parameters.
Variant 3: Mix of pre compile-, link time and post build time parameters.

CanTrcv017: Only pre compile time configuration is allowed. Thus only Variant1 is allowed.

10.2.2 CanTrcv

Module Name	**CanTrcv**
Module Description	Configuration of the CanTrcv (CAN Transceiver driver) module.

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CanTrcvChannel	1..*	Container gives CAN transceiver driver information about a single CAN transceiver channel. Any CAN transceiver driver has such CAN transceiver channels.
CanTrcvGeneral	1	Container gives CAN transceiver driver basic information.

10.2.3 CanTrcvGeneral

SWS Item	**CanTrcv090 :**
Container Name	CanTrcvGeneral{CanTransceiverDriverBasic}
Description	Container gives CAN transceiver driver basic information.
Configuration Parameters	

SWS Item	**CanTrcv107 :**		
Name	CanTrcvWakeUpSupport {CANTRCV_GENERAL_WAKE_UP_SUPPORT}		
Description	Informs whether wake up is supported by ISR, by polling or whether it is not supported. In case no wake up is supported by CAN transceiver hardware setting has to be always NO. Only in case wake up is supported by polling main function CanTrcv_main has to be present in source code and called by CanIf. Only in case wake up is supported by ISR callback function CanTrcv_CB_WakeupByBus has to be present in source code. In case of support for wake up either by ISR or by polling wake up ability may be switched on or off for each channel of one CAN transceiver channel independently by CanTrcvWakeupByBusUsed.		
Multiplicity	1		
Type	EnumerationParamDef		
Range	CANTRCV_WAKEUP_ BY_ISR	Wake up by interrupt	
	CAVTRCV_WAKEUP_BY_POLLING	Wake up by polling	
	CANTRCV_WAKEUP_NOT_SUPPORTED	Wake up is not supported	
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module dependency: CanTrcvWakeupByBusUsed		

SWS Item	**CanTrcv106 :**		
Name	CanTrcvGetVersionInfo {CANTRCV_GET_VERSION_INFO}		
Description	Switches version information API on and off. If switched off, function need not be present in compiled code.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**CanTrcv140 :**		
Name	CanTrcvIndex		
Description	Specifies the InstanceId of this module instance. If only one instance is present it shall have the Id 0.		
Multiplicity	1		
Type	IntegerParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**CanTrcv105 :**		
Name	CanTrcvDevErrorDetect {CANTRCV_DEV_ERROR_DETECT }		
Description	Switches development error detection and notification on and off. If switched on, #define CANTRCV_DEV _ERROR_DETECT ON shall be generated. If switched off, #define CANTRCV_DEV_ERROR _DETECT OFF shall be generated. Define shall be part of file CanTrcv_Cfg.h.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

No Included Containers

10.2.4 CanTrcvChannel

SWS Item	**CanTrcv091**
Container Name	CanTrcvChannel{CanTranceiverChannels}
Description	Container gives CAN transceiver driver information about a single CAN transceiver channel. Any CAN transceiver driver has such CAN transceiver channels.
Configuration Parameters	

SWS Item	**CanTrcv098**	
Name	CanTrcvInitState {CANTRCV_INIT_STATE}	
Description	State of CAN transceiver after call to CanTrcv_Init.	
Multiplicity	1	
Type	EnumerationParamDef	
Range	CANTRCV_OP_MODE_STANDBY	Standby operation mode
	CANTRCV_OP_MODE_SLEEP	Sleep operation mode

	CANTRCV_OP_MODE_NORMAL	Normal operation mode	
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Instance		

SWS Item	**CanTrcv099**		
Name	CanTrcvMaxBaudrate {CANTRCV_MAX_BAUDRATE}		
Description	Max baudrate for transceiver hardware type. Only used for validation purposes. Value shall be configured by configuration tool based on transceiver hardware type.		
Multiplicity	1		
Type	IntegerParamDef		
Range	0 .. 1000		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Instance		

SWS Item	**CanTrcv097**		
Name	CanTrcvControlsPowerSupply {CANTRCV_CONTROLS_POWER_SUPPLY}		
Description	Is ECU power supply controlled by this transceiver? TRUE = Controlled by transceiver. FALSE = Not controlled by transceiver.		
Multiplicity	1		
Type	BooleanParamDef		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Instance		

SWS Item	**CanTrcv096**		
Name	CanTrcvChannelUsed {CANTRCV_CHANNEL_USED}		
Description	Shall the related CAN transceiver channel be used?		
Multiplicity	1		
Type	BooleanParamDef		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Instance		

SWS Item	**CanTrcv100**		
Name	CanTrcvWakeupByBusUsed {CANTRCV_WAKEUP_BY_BUS_USED}		
Description	Is wake up by bus supported? If CAN transceiver hardware does not support wake up by bus value is always FALSE. If CAN transceiver hardware supports wake up by bus value is TRUE or FALSE depending whether it is used or not. TRUE = Is used. FALSE = Is not used.		
Multiplicity	1		
Type	BooleanParamDef		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Instance dependency: CanTrcvWakeUpSupport		

SWS Item	**CanTrcv140**		
Name	CanTrcvChannelId {CANTRCV_CHANNEL_ID}		
Description	Unique identifier of the CAN Transceiver Channel.		
Multiplicity	1		
Type	IntegerParamDef (Symbolic Name generated for this parameter)		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**CanTrcv141**		
Name	CanTrcvWakeupSourceRef {CANTRCV_WAKEUP_SOURCE_REF}		
Description	Reference to a wakeup source in the EcuM configuration. This reference is only needed if CanTrcvWakeupByBusUsed is true. Implementation Type: reference to EcuM_WakeupSourceType		
Multiplicity	1		
Type	Reference to EcuMWakeupSource		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU dependency: CanTrcvWakeupByBusUsed		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CanTrcvAccess	1..*	--

10.2.5 CanTrcvAccess

SWS Item	**CanTrcv101 :**
Choice Container Name	CanTrcvAccess
Description	--

Container Choices		
Container Name	***Multiplicity***	***Scope / Dependency***
CanTrcvDioAccess	0..1	Container gives CAN transceiver driver information about accessing ports and port pins. In addition relation between CAN transceiver hardware pin names and Dio port access information is given. If a CAN transceiver hardware has no Dio interface, there is no instance of this container.
CanTrcvSpiSequence	0..1	Container gives CAN transceiver driver information about one SPI sequence. One SPI sequence used by CAN transceiver driver is in exclusive use for it. No other driver is allowed to access this sequence. CAN transceiver driver may use one sequence to access n CAN transceiver hardwares chips of the same type or n sequences are used to access one single CAN transceiver hardware chip. If a CAN transceiver hardware has no SPI interface, there is no instance of this container.

10.2.6 CanTrcvDioAccess

SWS Item	**CanTrcv094 :**
Container Name	CanTrcvDioAccess{CanTransceiverDioAccess}
Description	Container gives CAN transceiver driver information about accessing ports and port pins. In addition relation between CAN transceiver hardware pin names and Dio port access information is given. If a CAN transceiver hardware has no Dio interface, there is no instance of this container.

Configuration Parameters

SWS Item	**CanTrcv103 :**		
Name	CanTrcvHardwareInterfaceName {CANTRCV_HARDWARE_INTERFACE_NAME}		
Description	CAN transceiver hardware interface name. It is typically the name of a pin. From a Dio point of view it is either a port, a single channel or a channel group. Depending on this fact either CANTRCV_DIO_PORT_SYMBOLIC_NAME or CANTRCV_DIO_CHANNEL_SYMBOLIC_NAME or CANTRCV_DIO_CHANNEL_GROUP_SYMBOLIC_NAME shall reference a Dio configuration. The CAN transceiver driver implementation description shall list up this name for the appropriate CAN transceiver hardware.		
Multiplicity	1		
Type	StringParamDef		
Default value	--		
ConfigurationClass	*Pre-compile time*	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Instance		

SWS Item	**CanTrcv102 :**		
Name	CanTrcvDioSymNameRef		
Description	Choice Reference to a DIO Port, DIO Channel or DIO Channel Group. This reference replaces the CANTRCV_DIO_PORT_SYM_NAME, CANTRCV_DIO_CHANNEL_SYM_NAME and CANTRCV_DIO_GROUP_SYM_NAME references in the Can Trcv SWS.		
Multiplicity	1		
Type	Choice Reference to DioChannel,DioChannelGroup,DioPort		
ConfigurationClass	*Pre-compile time*	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

No Included Containers

10.2.7 CanTrcvSpiSequence

SWS Item	**CanTrcv092 :**
Container Name	CanTrcvSpiSequence{CanTransceiverSPISequences}
Description	Container gives CAN transceiver driver information about one SPI sequence. One SPI sequence used by CAN transceiver driver is in exclusive use for it. No other driver is allowed to access this sequence. CAN transceiver driver may use one sequence to access n CAN transceiver hardwares chips of the same type or n sequences are used to access one single CAN transceiver hardware chip. If a CAN transceiver hardware has no SPI interface, there is no instance of this container.
Configuration Parameters	

SWS Item	**CanTrcv104 :**		
Name	CanTrcvSpiSequenceName {CANTRCV_SPI_SEQUENCE_NAME}		
Description	Reference to a Spi sequence configuration container.		
Multiplicity	1		
Type	Reference to SpiSequence		
ConfigurationClass	*Pre-compile time*	X	All Variants
	Link time	--	

	Post-build time	--	
Scope / Dependency	scope: Instance dependency: SpiSequence		

No Included Containers

10.3 Published Information

Published information contains data defined by the implementer of the SW module that does not change when the module is adapted (i.e. configured) to the actual HW/SW environment. It thus contains version and manufacturer information.

The standard common published information like

vendorId (<Module>_VENDOR_ID),
moduleId (<Module>_MODULE_ID),
arMajorVersion (<Module>_AR_MAJOR_VERSION),
arMinorVersion (<Module>_ AR_MINOR_VERSION),
arPatchVersion (<Module>_ AR_PATCH_VERSION),
swMajorVersion (<Module>_SW_MAJOR_VERSION),
swMinorVersion (<Module>_ SW_MINOR_VERSION),
swPatchVersion (<Module>_ SW_PATCH_VERSION),
vendorApiInfix (<Module>_VENDOR_API_INFIX)

is provided in the BSW Module Description Template (see 3 Figure 4.1 and Figure 7.1).

Additional published parameters are listed below if applicable for this module.

11 Changes to Release 1

CAN Transceiver Driver was not part of AUTOSAR release 1. Thus this chapter is not applicable for AUTOSAR release 2.

12 Changes during TO SWS Improvement

12.1 Deleted SWS Items

SWS Item	*Rationale*
CanTrcv042	not a requirement but a general description of the goal of the module
CanTrcv056	not a requirement but an example
CanTrcv073	redundant with CanTrcv040
CanTrcv034	should not be required specifically for this module
CanTrcv035	should not be required specifically for this module
CanTrcv080	should not be required specifically for this module
CanTrcv060	Reference does not exist.

12.2 Replaced SWS Items

NONE

12.3 Changed SWS Items

Many requirements have been changed to improve understanding without changing the technical contents.

SWS Item	*Rationale*
CanTrcv100	Added additional checks to validate whether there has been a wake up due to transceiver activity and if true report this to the EcuM.
CanTrcv067 CanTrcv066	Changed API name CanIf_TrcvWakeupByBus to CanIf_SetWakeupEvent and rephrased the sentence.
CanTrcv005 CanTrcv007 CanTrcv008	Output parameter in the API's CanTrcv_GetOpMode, CanTrcv_GetBusWuReason and CanTrcv_GetVersionInfo is changed to pointer type.
CanTrcv70	Rephrased the requirement.
CanTrcv105	Changed the requirement to return E_NOT_OK.
CanTrcv012	Modified the API CanTrcv_CB_WakeupByBus.

12.4 Added SWS Items

SWS Item	*Rationale*
CanTrcv100	Gave explicit id's to requirements out of CanTrcv_Init table
CanTrcv113	Gave explicit id's to requirements out of CanTrcv_Init table
CanTrcv102	Gave explicit id's to requirements out of CanTrcv_SetOpMode table
CanTrcv103	Gave explicit id's to requirements out of CanTrcv_SetOpMode table
CanTrcv104	Gave explicit id's to requirements out of CanTrcv_SetOpMode table
CanTrcv105	Gave explicit id's to requirements out of CanTrcv_SetOpMode table
CanTrcv114	Gave explicit id's to requirements out of CanTrcv_SetOpMode table
CanTrcv120	Gave explicit id's to requirements out of CanTrcv_SetOpMode table
CanTrcv121	Gave explicit id's to requirements out of CanTrcv_SetOpMode table
CanTrcv122	Gave explicit id's to requirements out of CanTrcv_SetOpMode table
CanTrcv123	Gave explicit id's to requirements out of CanTrcv_SetOpMode table
CanTrcv106	Gave explicit id's to requirements out of CanTrcv_GetOpMode table
CanTrcv115	Gave explicit id's to requirements out of CanTrcv_GetOpMode table
CanTrcv124	Gave explicit id's to requirements out of CanTrcv_GetOpMode table
CanTrcv129	Gave explicit id's to requirements out of CanTrcv_GetOpMode table
CanTrcv132	Gave explicit id's to requirements out of CanTrcv_GetOpMode table

CanTrcv107	Gave explicit id's to requirements out of CanTrcv_GetBusWuReason table
CanTrcv116	Gave explicit id's to requirements out of CanTrcv_GetBusWuReason table
CanTrcv125	Gave explicit id's to requirements out of CanTrcv_GetBusWuReason table
CanTrcv130	Gave explicit id's to requirements out of CanTrcv_GetBusWuReason table
CanTrcv133	Gave explicit id's to requirements out of CanTrcv_GetBusWuReason table
CanTrcv108	Gave explicit id's to requirements out of CanTrcv_GetVersionInfo table
CanTrcv109	Gave explicit id's to requirements out of CanTrcv_GetVersionInfo table
CanTrcv110	Gave explicit id's to requirements out of CanTrcv_GetVersionInfo table
CanTrcv126	Gave explicit id's to requirements out of CanTrcv_GetVersionInfo table
CanTrcv134	Gave explicit id's to requirements out of CanTrcv_GetVersionInfo table
CanTrcv111	Gave explicit id's to requirements out of CanTrcv_SetWakeupMode table
CanTrcv117	Gave explicit id's to requirements out of CanTrcv_SetWakeupMode table
CanTrcv127	Gave explicit id's to requirements out of CanTrcv_SetWakeupMode table
CanTrcv131	Gave explicit id's to requirements out of CanTrcv_SetWakeupMode table
CanTrcv112	Gave explicit id's to requirements out of CanTrcv_MainFunction table
CanTrcv128	Gave explicit id's to requirements out of CanTrcv_MainFunction table
CanTrcv139	Gave explicit id to requirement
CanTrcv87	Gave explicit id to requirement
CanTrcv88	Gave explicit id to requirement
CanTrcv89	Gave explicit id to requirement
CanTrcv135	Gave explicit id's to requirements out of CanTrcv_CB_WakeupByBus table
CanTrcv136	Gave explicit id's to requirements out of CanTrcv_CB_WakeupByBus table
CanTrcv137	Gave explicit id's to requirements out of CanTrcv_CB_WakeupByBus table
CanTrcv084	UML Model linking of imported types
CanTrcv001	UML Model linking of CanTrcv_Init
CanTrcv002	UML Model linking of CanTrcv_SetOpMode
CanTrcv005	UML Model linking of CanTrcv_GetOpMode
CanTrcv007	UML Model linking of CanTrcv_GetBusWuReason
CanTrcv008	UML Model linking of CanTrcv_GetVersionInfo
CanTrcv009	UML Model linking of CanTrcv_SetWakeupMode
CanTrcv012	UML Model linking of CanTrcv_CB_WakeupByBus
CanTrcv013	UML Model linking of CanTrcv_MainFunction
CanTrcv085	UML Model linking of mandatory interfaces
CanTrcv086	UML Model linking of optional interfaces
CanTrcv90	Gave explicit id to requirement
CanTrcv91	Gave explicit id to requirement
CanTrcv92	Gave explicit id to requirement
CanTrcv93	Gave explicit id to requirement
CanTrcv94	Gave explicit id to requirement
CanTrcv95	Gave explicit id to requirement
CanTrcv142	Unique identifier of the CAN Transceiver Channel added.
CanTrcv141	CanTrcvWakeup structure added.