Document Title	Specification of CAN Transceiver Driver
Document Owner	AUTOSAR GbR
Document Responsibility	AUTOSAR GbR
Document Version	1.1.0
Document Status	Draft
Part of Release	2.1
Revision	0014

Document Change History			
Date	**Version**	**Changed by**	**Change Description**
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

Release Notes

Errata and known deficiencies

The wakeup concept is currently neither harmonized nor consistent throughout all wakeup related specification documents and therefore subject to change.

Known and potential problems resulting from known deficiencies

Due to the fact that the wakeup concept is not harmonized, inconsistent assumptions may lead to

- the duplication of functionalities across multiple modules
- proprietary implementation extensions
- difficulties during integration

Changes planned for next release

The harmonized wakeup concept throughout all wakeup related documents will result in

- adapted specification texts in Chapter 7 of the specification documents
- adapted APIs in Chapter 8 of the specification documents
- adapted wakeup sequences in Chapter 9 of the specification documents

Disclaimer

Advice to users of AUTOSAR Specification Documents:

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. It 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 behaviour, which are applicable to most current and future CAN transceiver hardware chips and for nearly all use cases.

CanTrcv042: 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 like 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: config. 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
https://svn.autosar.org/repos/10Releases/
AUTOSAR_BasicSoftwareModules.pdf

[2] Layered Software Architecture
https://svn.autosar.org/repos/10Releases/
AUTOSAR_LayeredSoftwareArchitectur.pdf

[3] Specification of ECU Configuration
https://svn.autosar.org/repos/10Releases/
AUTOSAR_RS_ECU_Configuration.pdf

[4] General Requirements on Basic Software
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SRS_General.pdf

[5] Specification of Specification of CAN Interface
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_CANInterface.pdf

3.2 Related standards and norms

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

4 Constraints and assumptions

4.1 Limitations

CanTrcv098: The CAN bus transceiver hardware must provide functionality and an interface, which can be mapped to the operation mode model of the AUTOSAR CAN transceiver driver. See 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. The names may be extended with a vendor ID or a type ID. Any combination of these extensions is possible.

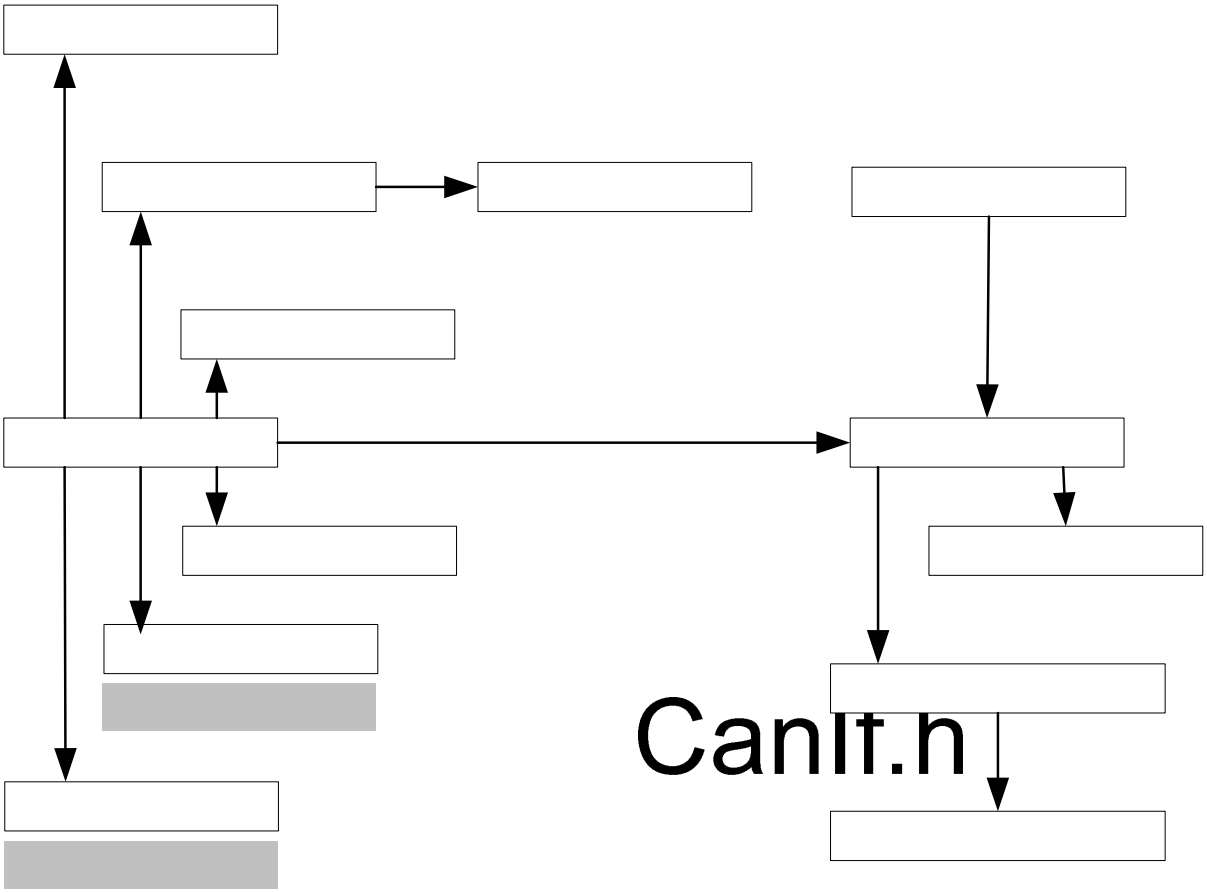
5.1.2 Code file structure

CanTrcv064: Naming convention is applied to all files.
CanTrcv065: Module consists of listed 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: Name of 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.

Dem.h

CanTrcv063: Name of 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.

CanTrcv_

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, CanTrcv073
[BSW00339] Reporting of production relevant error status	CanTrcv024, CanTrcv060, 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 CanTrcv060
[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 porpagate 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	CanTrcv034
[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	CanTrcv042
[BSW164] Implementation of interrupt service routines	not applicable (CAN transceiver driver implements no ISRs)
[BSW167] Static configuration checking	CanTrcv035
[BSW168] Diagnostic Interface of SW components	not applicable (CAN transceiver driver has no such needs)
[BSW170] Data for reconfiguration of AUTOSAR SW components	CanTrcv080
[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

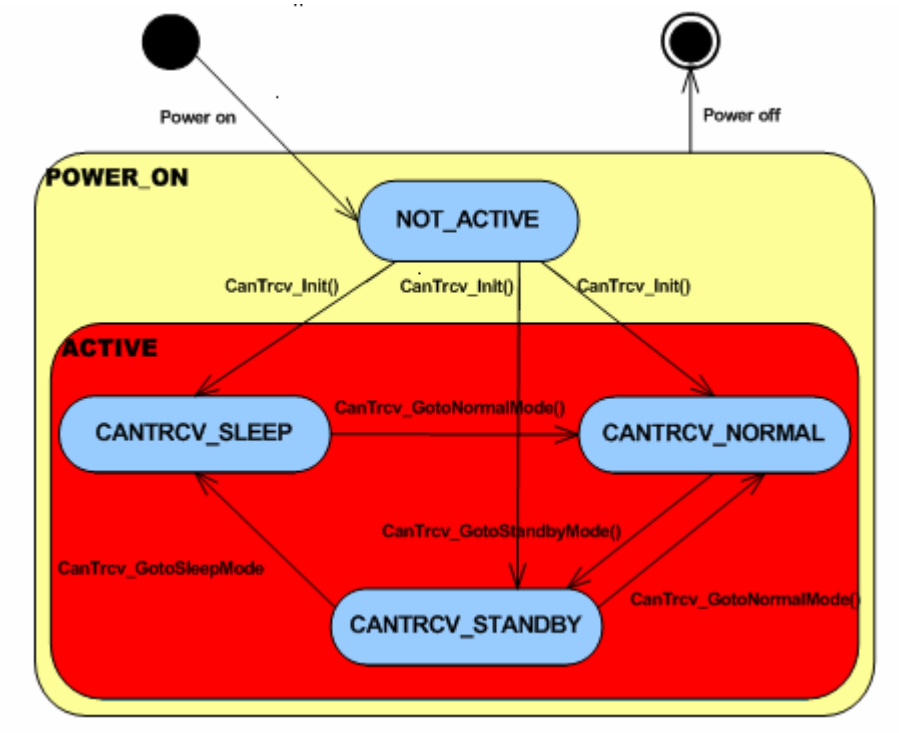
Requirement	***Satisfied by***
[BSW01090] Configuration Data for CAN Bus Transceiver	CanTrcv090, CanTrcv091, CanTrcv092 CanTrcv093, CanTrcv094, CanTrcv095

[BSW01091] Support for more than one CAN transceiver. Only pre-compile time configuration allowed.	CanTrcv002, CanTrcv005, CanTrcv007, CanTrcv009, CanTrcv012, CanTrcv016, 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, CanTrcv056
[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 CAN transceiver driver operation modes are described in the state diagram below. 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 independent configurable for each channel.

State	***Description***
POWER_ON	ECU is fully powered.
NOT_ACTIVE	State of CAN transceiver hardware depend 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() was 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.

7.2 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 whole functionality of described CAN transceiver driver software state is given by the implementation.

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

CanTrcv056: The mode often refered to as "Go-to-sleep” is a temporary mode when switching from Normal to Sleep. The driver shall encapsulate such a temporary mode within one of the CAN transceiver driver software states. In addition CAN transceiver driver switches first from Normal to Standby and than 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 acknowledge bit during reception of a message is supressed. This mode is not supported because it’s outside of 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:

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. CAN transceiver hardware causes powering of MCU. In terms of Autosar this is kept as a cold start and NOT as a wake up.
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. 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.
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. 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:

1) In mode NO no wake ups are generated by CAN transceiver driver. This mode is supported by all CAN transceiver hardware types.
2) 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.
3) 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.

Selection of wake up mode is done with configuration parameter CANTRCV_GENERAL_WAKE_UP_SUPPORT. Support of wake ups may be switched on and off for each CAN transceiver channel individually by configuration parameter CANTRCV_WAKEUP_BY_BUS_USED.

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
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 *CANTRCV_DEV_ERROR_DETECT* shall activate or deactivate the detection of all development errors.

CanTrcv048: If the *CANTRCV_DEV_ERROR_DETECT* switch is enabled API parameter checking is enabled. The detailed description of the detected errors can be found in chapter 7.5.

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 *CANTRCV_DEV_ERROR_DETECT* 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 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 underlaying 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 have need to have the port pin level 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 CAN transceiver hardware may have to perform wait states. Wait states are performed by accessing module Frt. API of Frt module is currently not specified. Thus no closer description in this specification. Access to Frt module is up to this future development.

8 API specification

8.1 Imported types

8.1.1 Standard types

Used types are Std_ReturnType and Std_VersionInfoType.

8.2 Type definitions

CanTrcv does not define types.

8.3 Function definitions

8.3.1 CanTrcv_Init

Service name:	CanTrcv_Init
Syntax:	void CanTrcv_Init ( )
Service ID [hex]:	0x0
Behavior:	Synchronous
Reentrancy:	non reentrant
Parameters (in):	-- --
Parameters (out):	none --
Return value:	none --
Description:	CanTrcv001 After this call all CAN transceiver hardware is either in state CANTRCV_NORMAL, CANTRCV_STANDBY or CANTRCV_SLEEP. In time span between power up and call of CanTrcv_Init CAN transceiver hardware maybe in a different state. This depends on hardware and SPAL driver configuration. Development errors: - Production errors: CANTRCV_E_NO_TRCV_CONTROL: incorrect transceiver access
Caveats:	The initialization sequence after reset (e.g. power up) is a critical phase for the CAN transceiver driver. The driver will use SPAL functionality (SPI or DIO) to access the transceiver hardware. Therefore all necessary BSW drivers must be initialized and usable before.
Configuration:	Configuration parameter CANTRCV_INIT_STATE specifies state after call of this function.

8.3.2 CanTrcv_SetOpMode

Service name:	CanTrcv_SetOpMode	
Syntax:	Std_ReturnType CanTrcv_SetOpMode (unit8 CanNetwork, CanIf_TransceiverModeType OpMode)	
Service ID [hex]:	0x1	
Behavior:	Synchronous	
Reentrancy:	non reentrant	
Parameters (in):	CanNetwork	CAN network to which API call has to be applied.
	OpMode	The parameter says to which operation mode the change shall be performed
Parameters (out):	none	
Return value:	E_OK	E_OK will be returned if the transceiver state has been changed to the requested mode.
	E_NOT_OK	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:	CanTrcv002: The internal state of the CAN transceiver driver is switched to one of following values, depending on parameter OpMode CANTRCV_NORMAL CANTRCV_STANDBY CANTRCV_SLEEP The call to this API with OpMode == CANTRCV_SET_OP_MODE_STANDBY is allowed only, if the CAN bus transceiver is already set to CANTRCV_NORMAL. If this is not true, the error CANTRCV_E_TRCV_NOT_NORMAL is notified. The call to this API with OpMode == CANTRCV_SET_OP_MODE_SLEEP is allowed only, if the CAN bus transceiver is already set to CANTRCV_STANDBY. If this is not true, the error CANTRCV_E_TRCV_NOT_STANDBY is notified. <u>Development errors:</u> CANTRCV_E_INVALID_CAN_NETWORK: network number invalid CANTRCV_E_UNINIT: not yet initialized CANTRCV_E_TRCV_NOT_NORMAL: invalid parameter value CANTRCV_E_TRCV_NOT_STANDBY: invalid parameter value <u>Production errors:</u> CANTRCV_E_NO_TRCV_CONTROL: incorrect transceiver access	
Caveats:	This API shall be applicable to each transceiver with each value for parameter CanTrcv_SetOpMode independent if the transceiver hardware supports this modes or not. This is to ease up the view of the CanIf to the assigned bus. If the mode is not supported, the return value shall be E_OK.	
Configuration:	The number of supported busses is set up in the configuration phase.	

8.3.3 CanTrcv_GetOpMode

Service name:	CanTrcv_GetOpMode	
Syntax:	Std_ReturnType CanTrcv_GetOpMode (unit8 CanNetwork, CanIf_TransceiverModeType* OpMode)	
Service ID [hex]:	0x2	
Behavior:	Synchronous	
Reentrancy:	reentrant	
Parameters (in):	CanNetwork	CAN network to which API call has to be applied.
Parameters (out):	OpMode	Pointer to operation mode of the bus the API is applied to.
Return value:	E_OK	E_OK will be returned if the operation mode was detected.
	E_NOT_OK	E_NOT_OK will be returned if the operation mode was not detected.
Description:	CanTrcv005 API detects actual state of CAN transceiver driver. Development errors: CANTRCV_E_INVALID_CAN_NETWORK: network number invalid CANTRCV_E_PARAM_POINTER: parameter is a null pointer. CANTRCV_E_UNINIT: not yet initialized Production errors: CANTRCV_E_NO_TRCV_CONTROL: incorrect transceiver access	
Caveats:	See CanTrcv_Init() for the provided state after the CAN transceiver driver initialization till the first operation mode change request.	
Configuration:	The number of supported busses is statically set in the configuration phase.	

8.3.4 CanTrcv_GetBusWuReason

Service name:	CanTrcv_GetBusWuReason	
Syntax:	Std_ReturnType CanTrcv_GetBusWuReason (unit8 CanNetwork, CanIf_TrcvWakeupReasonType* Reason)	
Service ID [hex]:	0x3	
Behavior:	Synchronous	
Reentrancy:	reentrant	
Parameters (in):	CanNetwork	CAN network to which API call has to be applied.
Parameters (out):	Reason	Pointer to wake up reason of the bus the API is applied to.
Return value:	E_OK	E_OK will be returned if the wake up reason was detected.
	E_NOT_OK	E_NOT_OK will be returned if the wake up reason was not detected.
Description:	CanTrcv007 This API returns the reason for the wake up that the CAN transceiver has	

	detected. The ability to detect and differentiate the possible wake up reasons depends strongly on the CAN transceiver hardware. Development errors: CANTRCV_E_INVALID_CAN_NETWORK: network number invalid CANTRCV_E_PARAM_POINTER: parameter is a null pointer. CANTRCV_E_UNINIT: not yet initialized Production errors: CANTRCV_E_NO_TRCV_CONTROL: incorrect transceiver access
Caveats:	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.
Configuration:	The number of supported busses is statically set in the configuration phase.

8.3.5 CanTrcv_GetVersionInfo

Service name:	CanTrcv_GetVersionInfo
Syntax:	void CanTrcv_GetVersionInfo (Std_VersionInfoType *versioninfo)
Service ID [hex]:	0x4
Sync/Async:	Synchronous
Reentrancy:	non reentrant
Parameters (in):	none
Parameters (out):	versioninfo Pointer to version information of this module.
Return value:	none
Description:	CanTrcv008 This service returns the version information of this module. The version information includes: - Module Id - Vendor Id - Vendor specific version numbers (BSW00407). This function shall be pre compile time configurable On/Off by the configuration parameter: CANTRCV_GET_VERSION_INFO. Development errors: CANTRCV_E_UNINIT: not yet initialized CANTRCV_E_PARAM_POINTER: parameter is a null pointer. Hint: If source code for caller and callee of this function is available this function should be realized as a macro. The macro should be defined in the modules header file.
Caveats:	--
Configuration:	CANTRCV_GET_VERSION_INFO

8.3.6 CanTrcv_ SetWakeupMode

Service name:	CanTrcv_ SetWakeupMode	
Syntax:	`Std_ReturnType CanTrcv_SetWakeupMode ( unit8 CanNetwork, CanIf_TrcvWakeupModeType TrcvWakeupMode )`	
Service ID [hex]:	0x5	
Behavior:	Synchronous	
Reentrancy:	non-reentrant	
Parameters (in):	CanNetwork	CAN network to which API call has to be applied.
	TrcvWakeupMode	Requested transceiver wakeup reason
Parameters (out):	none	--
Return value:	`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:	CanTrcv009 This API enables, disables and clears the notification for wakeup events on the addressed network. Enabled: if the CAN transceiver driver has a stored wakeup event pending for the addressed bus, the notification is executed within the API call or immediately after (depending on the implementation). Disabled: Ii it is required by the transceiver device and the underlying communication, the driver has to detect the wakeup events nevertheless and stores it internally to raise the event, when the wakeup notification is enabled again. Clear: 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. Development errors: CANTRCV_E_INVALID_CAN_NETWORK: network number invalid CANTRCV_E_UNINIT: not yet initialized Production errors: CANTRCV_E_NO_TRCV_CONTROL: incorrect transceiver access	
Caveats:	The implementation may be enable or disable interrupt source for the wake up or it clears 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.	
Configuration:	The number of supported busses is statically set in the configuration phase.	

8.4 Scheduled functions

8.4.1 CanTrcv_MainFunction

Service name:	CanTrcv_MainFunction
Service ID [hex]:	0x6
Description:	CanTrcv013 The CAN bus transceiver driver may have cyclic jobs like polling for wake up events (if configured). This cyclic called function has to scan all busses in STANDBY and SLEEP for wake up events and has to perform these events by calling appropriate callback function. According [BSW00424] main processing functions shall be allocated by basic tasks. No special call order to be kept. Function is called within CanIf_MainFunction_Wakeup. Development errors: CANTRCV_E_UNINIT: not yet initialized
Timing:	fixed cyclic
Pre condition:	--
Configuration:	See configuration parameter CANTRCV_GENERAL_WAKE_UP_SUPPORT.

8.5 Call-back notifications

This is a list of functions provided for lower layer modules. The function prototypes of the callback functions shall be provided in the file CanTrcv_Cbk.h.

8.5.1 CanTrcv_CB_WakeupByBus

Service name:	CanTrcv_CB_WakeupByBus	
Syntax:	void CanTrcv_CB_WakeupByBus (unit8 CanNetwork)	
Service ID [hex]:	0x7	
Behavior:	Synchronous	
Reentrancy:	reentrant	
Parameters (in):	CanNetwork	CAN network to which API call has to be applied.
Parameters (out):	none	--
Return value:	--	--
Description:	CanTrcv012 This API is called by underlying SPAL Icu driver in case a wake up interrupt is detected. The API validates wake up reason in terms whether it is a wake up or not. Development errors: CANTRCV_E_INVALID_CAN_NETWORK: network number invalid CANTRCV_E_UNINIT: not yet initialized	

Caveats:	Wake up by bus is always asynchronous to the transition to sleep and standby. In 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.
Configuration:	See configuration parameter CANTRCV_GENERAL_WAKE_UP_SUPPORT.
Call Context	The call context of this API is expected to be within the ISR handler of the underlying driver. This has to be documented (BSW00333).

8.6 Expected Interfaces

In this chapter all interfaces required from other modules are listed.

8.6.1 Mandatory Interfaces

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

API function	***Module***	***Description***
Dem_ReportErrorStatus	Dem	Called by CAN transceiver driver to indicate error events to DEM.

8.6.2 Optional Interfaces

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

API function	Module	Description	Configuration
CanIf_TrcvWakeup ByBus	CanIf	CanTrcv066 Wake up by bus notification. Called in operation modes sleep and standby. Not called if call to CanTrcv_Goto_ NormalMode() has caused wake up.	Container CanTransceiverChannels, parameters CANTRCV_WAKEUP _BY_BUS_USED and CANTRCV_WAKEUP _POLLING_USED
Det_ Report Error	Det	CanTrcv073 Development error notification.	Container CanTransceiver DriverBasic, parameter CANTRCV_DEVELOP MENT_ERROR.
Spi_WriteIB	Spi[1]	Write data to an IB of a SPI channel.	Container CanTransceiverSPI-Channel gives reference for appropriate SPI container with all necessary information.
Spi_ReadIB		Read to from an IB of a SPI channel.	
Spi_SetupEB		Configure EB for a SPI channel	
Spi_GetStatus		Returns current state of SPI driver	No configuration necessary.
Spi_SyncTransmit		Starts synchronous transmission on a SPI connection	Sequence name given by container CanTransceiver SPISequences.
Dio_ReadChannel	Dio[2]	Read access to one Dio channel.	Appropriate identifier names given by instances of container CanTransceiverDIOAccess.
Dio_WriteChannel		Write access to one Dio channel.	
Dio_ReadPort		Read access to one Dio port.	
Dio_WritePort		Write access to one Dio port.	
Dio_Read ChannelGroup		Read access to one Dio channel group.	
Dio_Write ChannelGroup		Write access to one Dio channel group.	

1. Interfaces of SPI module used if there are instances of container CanTransceiverSPISequeces.
2. Interfaces of DIO module used if there are instances of container CanTransceiverDIOAccess.

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

8.6.3 Configurable interfaces

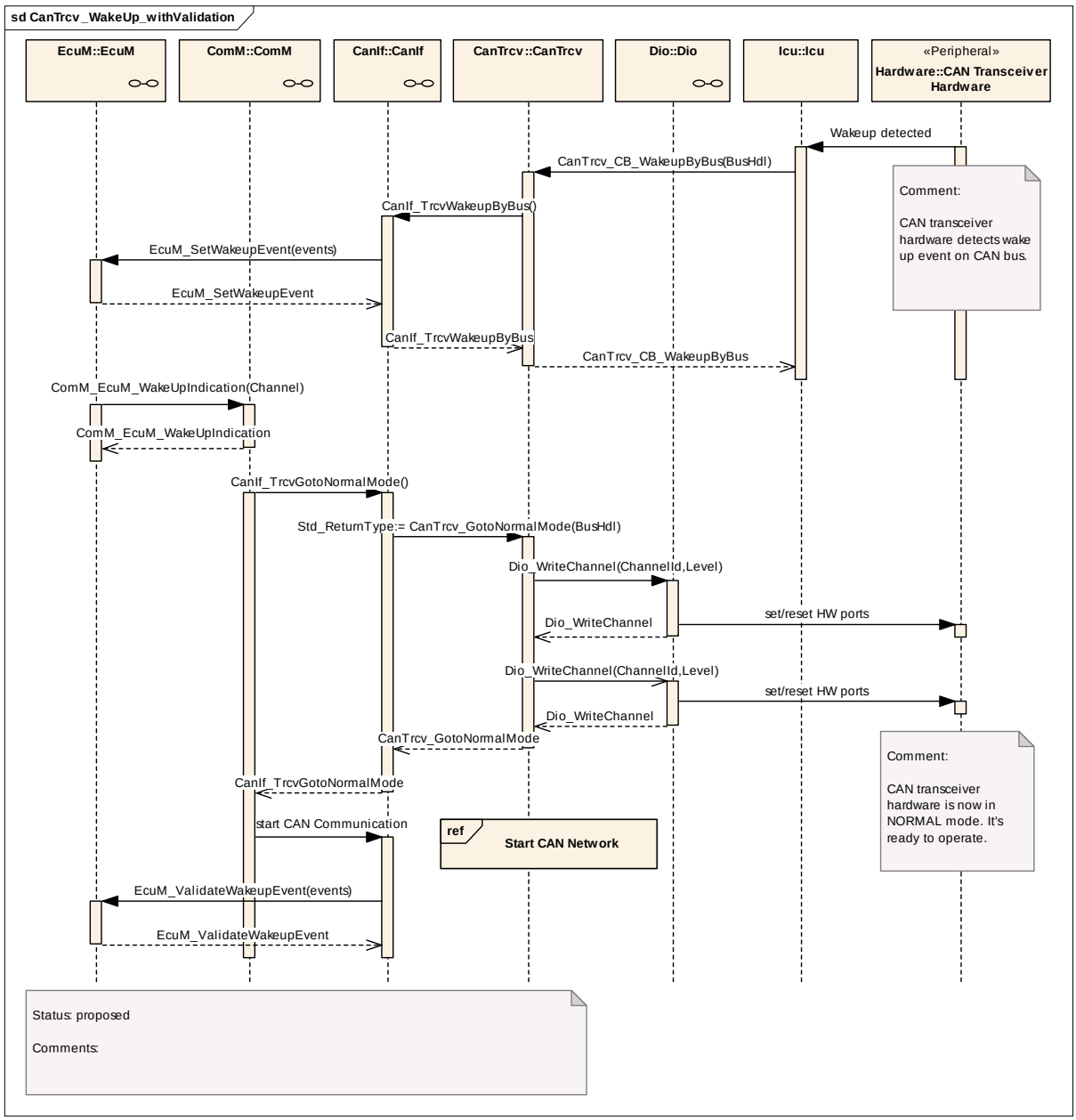
There are no configurable interfaces for CAN transceiver driver.

9 Sequence diagram

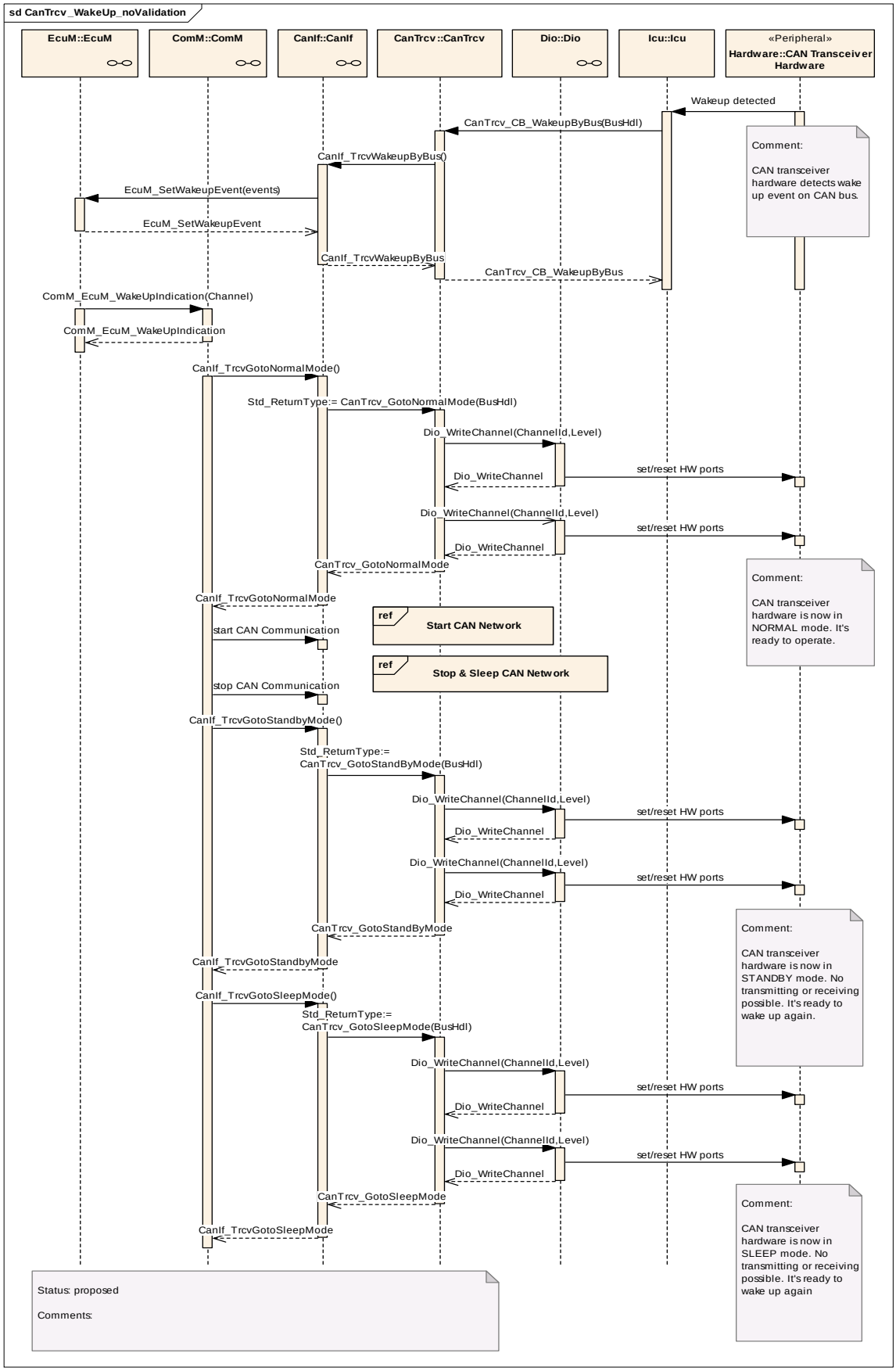
Focus of graphs is located on interaction between the CAN transceiver driver and BSW modules CanIf, ComM, EcuM, Icu and Dio. Depending on CAN transceiver hardware one or more calls to Dio_WriteChannels may be necessary.

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

9.1 Wake up with valid validation



9.2 Wakeup with missing validation



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 10.3 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 General configuration requirements

All following configuration requirements has to be fulfilled by configuration tool. Configuration information is part of files CanTrcv.h and CanTrcv_Cfg.c. See chapters 5.1.2 and 5.1.3 for details.

Requirement	***Description***
CanTrcv034	A configuration tool is used to generate the configuration data and code if any.
CanTrcv035	The configuration tool has to check the validity of the provided input data and the usability in the project context.
CanTrcv080	Provide configuration dependency information.

10.2.3 Container CanTransceiverDriverBasic

SWS Item	CanTrcv090
Container Name	CanTransceiverDriverBasic
Description	Container gives CAN transceiver driver basic information.
Configuration Parameters	

Name	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.		
Type or Unit	BooleanParamDef		
Range	TRUE	Is used.	
	FALSE (default)	Is not used.	
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	--	--
	Post Build	--	--
Scope	module		
Dependency	-		

Name	CANTRCV_GENERAL_WAKE_UP_SUPPORT		
Description	Informs whether wake up is supported by ISR, by polling or whether it is not supported. See chapter 7.9 for details.		
	EnumParamDef		
Range	ISR	Wake up supported by ISR	
	POLLING	Wake up supported by polling	
	NO	Wake up not supported.	
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	--	--
	Post Build	--	--
Scope	module		
Dependency	CANTRCV_WAKEUP_BY_BUS_USED		

Name	CANTRCV_GET_VERSION_INFO		
Description	Switches version information API on and off. If switched off, function need not be present in compiled code.		
Type or Unit	BooleanParamDef		
Range	TRUE	Is used.	
	FALSE (default)	Is not used.	
Configuration Class	**Pre-compile**	x	Variant 1
	Link time	--	--
	Post Build	--	--
Scope	module		
Dependency	-		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CanTransceiverChannel	1...n	ECU

10.2.4 Container CanTransceiverChannels

SWS Item	CanTrcv091
Container Name	CanTransceiverChannels
Description	Container gives CAN transceiver driver information about a single CAN transceiver channel.
Configuration Parameters	

Name	CANTRCV_CHANNEL_USED		
Description	Shall the related CAN transceiver channel be used?		
Type or Unit	BooleanParamDef		
Range	TRUE (default)	Is used.	
	FALSE	Is not used.	
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	--	--
	Post Build	--	--
Scope	Instance		
Dependency	-		

Name	CANTRCV_CANIF_NETWORK
Description	Unique network number. It is used by ComM by Icu and internally.

	Reference to a CanIf configuration item CANIF_NETWORK.		
Type or Unit	IntegerParamReferenceDef		
Range	0 … (n-1)		
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	--	--
	Post Build	--	--
Scope	Instance		
Dependency	--		

Name	CANTRCV_INIT_STATE		
Description	State of CAN transceiver after call to CanTrcv_Init.		
Type or Unit	EnumParamDef		
Range	CANTRCV_GET_OP_MODE_NORMAL (default)	In normal mode	
	CANTRCV_OP_MODE_STANDBY	In standby mode	
	CANTRCV_OP_MODE_SLEEP	In sleep mode	
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	--	--
	Post Build	--	--
Scope	Instance		
Dependency	--		

Name	CANTRCV_WAKEUP_BY_BUS_USED		
Description	Informs whether a wake up ability of a single CAN transceiver channel is used or not. In case CANTRCV_ GENERAL_WAKE_UP_SUPPORT is configured to NO this configuration parameter is without effect. See CANTRCV_ GENERAL_WAKE_UP_SUPPORT for more information.		
Type or Unit	BooleanParamDef		
Range	TRUE	Is used.	
	FALSE (default)	Is not used.	
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	--	--
	Post Build	--	--
Scope	instance		
Dependency	CANTRCV_GENERAL_WAKE_UP_SUPPORT		

Name	CANTRCV_CONTROLS_POWER_SUPPLY		
Description	Is ECU power supply controlled by this transceiver?		
Type or Unit	BooleanParamDef		
Range	TRUE	Controlled by transceiver.	
	FALSE (default)	Not controlled by transceiver.	
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	--	--
	Post Build	--	--
Scope	Instance		
Dependency	-		

Name	CANTRCV_MAX_BAUDRATE		
Description	Max baudrate for transceiver hardware type. Only used for validation purposes. Value shall be configured by configuration tool based on CANTRCV_HARDWARE_NAME and internal information about ability of this hardware typel.		
Type or Unit	IntegerParamDef		
Range	0-1000	Baud rate in kbaud	
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	--	--

	Post Build	--	--
Scope	Instance		
Dependency	CANTRCV_HARDWARE_NAME		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CanTransceiver SPISequence	0…n	Instance
CanTransceiver DIOAccess	0…n	Instance

10.2.5 Container CanTransceiverSPISequences

SWS Item	CanTrcv092
Container Name	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	

Name	CANTRCV_SPI_SEQUENCE_NAME		
Description	Reference to a Spi sequence configuration container.		
Type or Unit	SymbolicNameReferenceDef		
Range	ID derived from SPI configuration		
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	--	--
	Post Build	--	--
Scope	Instance		
Dependency	**Container SPISequenceConfiguration**		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CanTransceiver SPIJobs	**1...n**	**Instance**

10.2.6 Container CanTransceiverSPIJob

SWS Item	CanTrcv095
Container Name	CanTransceiverSPIJob
Description	Container gives CAN transceiver driver information about one SPI job.
Configuration Parameters	

Name	CANTRCV_SPI_JOB_NAME		
Description	Reference to a Spi job configuration container.		
Type or Unit	SymbolicNameReferenceDef		
Range	ID derived from SPI configuration		
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	--	--
	Post Build	--	--
Scope	Instance		
Dependency	**Container SPIJobConfiguration**		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CanTransceiver SPIChannel	**1...n**	**Instance**

10.2.7 Container CanTransceiverSPIChannel

SWS Item	CanTrcv093
Container Name	CanTransceiverSPIChannel
Description	Container gives CAN transceiver driver information about one SPI channel.
Configuration Parameters	

Name	CANTRCV_SPI_CHANNEL_NAME		
Description	Reference to a Spi channel configuration container.		
Type or Unit	SymbolicNameReferenceDef		
Range	ID derived from SPI configuration		
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	--	--
	Post Build	--	--
Scope	Instance		
Dependency	**Container SpiChannelConfiguration:**		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
-	**-**	**-**

10.2.8 Container CanTransceiverDIOAccess

SWS Item	CanTrcv094
Container Name	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	

Name	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's either a port, a single channel or a channel group. Depending on this fact either CANTRCV_DIO_PORT_SYM_NAME or CANTRCV_DIO_CHANNEL_SYM_NAME or CANTRCV_DIO_GROUP_SYM_NAME shall reference a DIO configuration. The CAN transceiver driver implementation description shall list up this name for the appropriate CAN transceiver hardware.		
Type or Unit	StringParamDef		
Range	-		
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	--	--
	Post Build	--	--
Scope	Instance		
Dependency	-		

Name	CANTRCV_DIO_PORT_SYM_NAME		
Description	Reference to a DIO port configuration container.Only CANTRCV_DIO_PORT_SYM_NAME or CANTRCV_DIO_CHANNEL_SYM_NAME or CANTRCV_DIO_GROUP_SYM_NAME shall reference an entity of DIO configuration. The other two parameters shall be 0x0. 0x0 means no reference.		
Type or Unit	SymbolicNameReferenceDef		
Range	-		
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	--	--
	Post Build	--	--
Scope	Instance		
Dependency	**This container:** CANTRCV_DIO_CHANNEL_SYM_NAME CANTRCV_DIO_GROUP_SYM_NAME **Container DioPort**		

Name	CANTRCV_DIO_CHANNEL_SYM_NAME		
Description	Reference to a DIO channel configuration container.Only CANTRCV_DIO_PORT_SYM_NAME or CANTRCV_DIO_CHANNEL_SYM_NAME or CANTRCV_DIO_GROUP_SYM_NAME shall reference an entity of DIO configuration. The other two parameters shall be 0x0. 0x0 means no reference.		
Type or Unit	SymbolicNameReferenceDef		
Range	-		
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	--	--
	Post Build	--	--
Scope	Instance		
Dependency	**This container:** CANTRCV_DIO_PORT_SYM_NAME CANTRCV_DIO_GROUP_SYM_NAME **Container DioChannel**		

Name	CANTRCV_DIO_GROUP_SYM_NAME		
Description	Reference to a DIO channel group configuration container.Only CANTRCV_DIO_PORT_SYM_NAME or CANTRCV_DIO_CHANNEL_SYM_NAME or CANTRCV_DIO_GROUP_SYM_NAME shall reference an entity of DIO configuration. The other two parameters shall be 0x0. 0x0 means no reference.		
Type or Unit	SymbolicNameReferenceDef		
Range	-		
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	--	--
	Post Build	--	--
Scope	Instance		
Dependency	**This container:** CANTRCV_DIO_PORT_SYM_NAME CANTRCV_DIO_CHANNEL_SYM_NAME **Container DioChannelGroup**		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
-	**-**	**-**

11 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.

SWS Item	CanTrcv021	
Information elements		
Information element name	***Type / Range***	***Information element description***
CANTRCV_VENDOR_ID	#define/ uint16	Vendor ID of the dedicated implementation of this module according to the AUTOSAR vendor list
CANTRCV_VENDOR_API_INFIX	#define/ uint16	This parameter is used to specify the vendor specific name. In total the implementation specific name is generated as follows: <ModuleName>_<VendorId>_<API name from SWS>
CANTRCV_MODULE_ID	#define/uint8	Module ID of this module from Module List
CANTRCV _AR_MAJOR_VERSION	#define/uint8	Major version number of AUTOSAR specification where the appropriate implementation is based on.
CANTRCV _AR_MINOR_VERSION	#define/uint8	Minor version number of AUTOSAR specification where the appropriate implementation is based on.
CANTRCV _AR_PATCH_VERSION	#define/uint8	Patch level version number of AUTOSAR specification where the appropriate implementation is based on.
CANTRCV _SW_MAJOR_VERSION	#define/uint8	Major version number of the vendor specific implementation of the module. The numbering is vendor specific.
CANTRCV _SW_MINOR_VERSION	#define/uint8	Minor version number of the vendor specific implementation of the module. The numbering is vendor specific.
CANTRCV _SW_PATCH_VERSION	#define/uint8	Patch level version number of the vendor specific implementation of the module. The numbering is vendor specific.

12 Changes to Release 1

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