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

Document Version	2.1.0
Document Status	Final
Part of Release	3.2
Revision	3

Document Change History

Date	Version	Changed by	Change Description
28.02.2014	2.1.0	AUTOSAR Release Management	• Revised configuration for SPI Interface • Added an API “CanTrcv_SetPNActivationState” • Editorial changes
17.05.2012	2.0.0	AUTOSAR Administration	• Added a new requirement for Partial Networking. • Changed the sequence diagrams of Interaction with DIO module,De-Init with synchronous SPI sequence and De-Init with asynchronous SPI sequence. • Added the container "CanTrcvPnFrameDataMaskSpec".
07.04.2011	1.4.0	AUTOSAR Administration	• Changed feature name 'Partial Networking' to 'Selective Wakeup'. • Added option to enable/disable selective wakeup in the CanTrcv driver. • Modified the wakeup mechanism, initialization and mode requests, to adapt the changes in selective wakeup feature. • Modified few APIs as re-entrant to support selective wakeup feature. • Legal disclaimer revised
15.09.2010	1.3.0	AUTOSAR Administration	• Explanation added to chapter 7.4 • Updated CanTrcv150 • Legal disclaimer revised
04.08.2008	1.2.2	AUTOSAR Administration	Layout adaptations
23.06.2008	1.2.1	AUTOSAR Administration	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

Known Limitations

The sequence chart in chap 9.2 "Wake up with Validation" of the CanTrcvDrv contains errors:
1. CanSm is missing as intermediate module for the interactions between ComM and CanIf.
2. After wakeup indication, full communication is requested and CanSM has to request CANIF_TRCV_MODE_NORMAL (so CanIf_SetTransceiverMode and CanTrcvSetOpMode are called with CANIF_TRCV_MODE_NORMAL instead of CANIF_TRCV_MODE_STDBY as indicated in the current figure).
3. The asynchronous access of the transceiver is not shown: CanSm has to wait until CanTrcv indicates the successful mode transition to CANIF_TRCV_MODE_NORMAL before the Start CAN Network sequence is started. This is mandatory to ensure that all implementations of CAN controller hardware are able to switch the controller mode.

The asynchronous handling will be added to the sequence chart in the next revision.

1 Introduction

This specification describes the functionality, APIs and configuration of CAN Transceiver Driver module. The CAN Transceiver Driver module is responsible for handling the CAN transceiver hardware chips on an ECU.

The CAN Transceiver is a hardware device, which adapts the signal levels that are used on the CAN bus to the logical (digital) signal levels recognized by a microcontroller.

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

Some transceivers support power supply control and wake up via the CAN bus. Different wake up/sleep and power supply concepts are usual on the market.

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

Latest developments include System Basis Chips (SBCs) where power supply control and advanced watchdogs are implemented in addition to CAN. These are enclosed in one housing and controlled through single interface (e.g. via SPI).

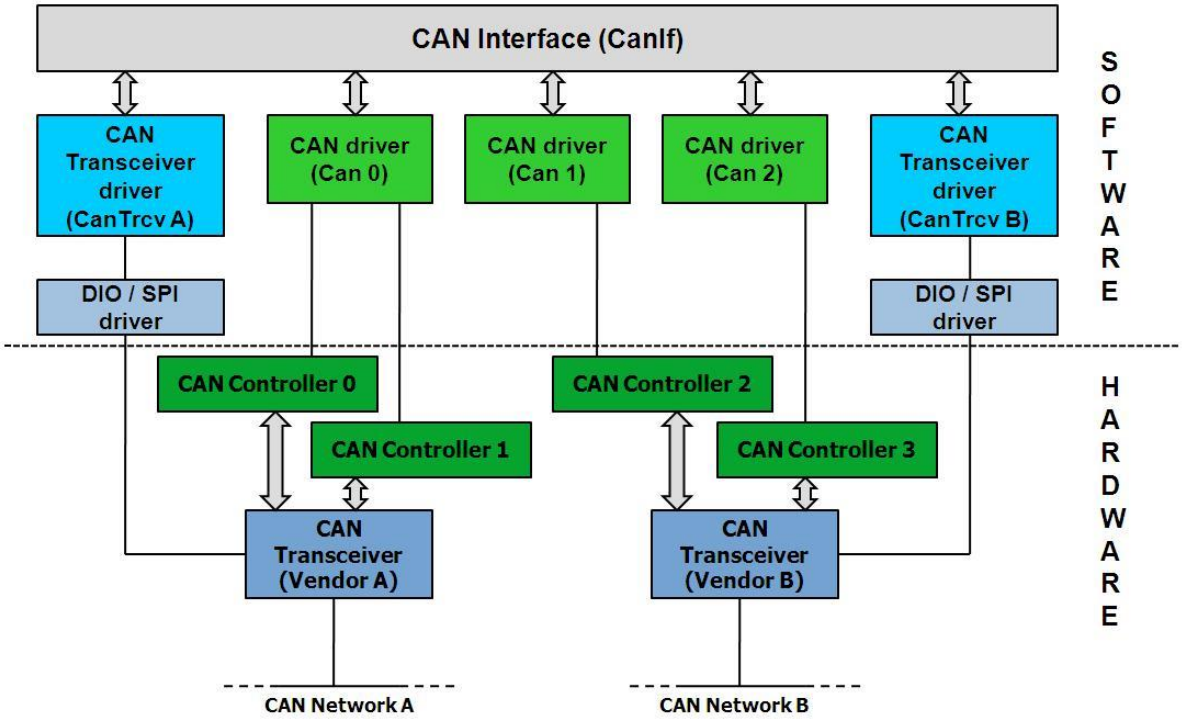


Figure 1.1: Example diagram for CanTrcv hardware and software interfaces

1.1 Goal of CAN transceiver driver

The target of this document is to specify the interfaces and behavior which are applicable to most current and future CAN transceiver devices.

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

1.2 Explicitly uncovered CAN transceiver functionality

Some CAN bus transceivers offer additional functionality, 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 would lock out most currently used transceiver hardware chips. Therefore, features like “ground shift detection”, “selective wake up”, “slope control” are not supported.

1.3 System basis chips

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

1.4 Single wire CAN transceivers

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

2 Acronyms and abbreviations

Abbreviation	*Description*
API	Application Programming Interface
CAN Channel	A physical channel which is connected to a CAN network from a CAN controller through a CAN transceiver.
CanSM	Can State Manager
ComM	Communication Manager
DEM	Diagnostic Event Manager
DET	Development Error Tracer
DIO	Digital Input Output (an AUTOSAR module)
EB	Externally Buffered channels. Buffers containing data to transfer are outside the SPI driver.
EcuM	Ecu state Manager
IB	Internally Buffered channels. Buffers containing data to transfer are inside the SPI driver.
ICU	Input Capture Unit
ID/id	Identifier
ISR	Interrupt Service Routine
MCAL	Micro Controller Abstraction Layer
Port	An AUTOSAR module.
POR	Power On Reset (Flag within the transceiver)
n/a	Not Applicable
SBC	System Basis Chip; a device which integrates a CAN and/or LIN transceiver, watchdog and power control.
SPAL	Standard Peripheral Abstraction Layer
SPI	Serial Peripheral Interface (an AUTOSAR module)
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.
SYSERR	System Error (Flag within the transceiver)
WUF	Wake Up Frame
WUP	Wake Up Pattern

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.

Implementation of an underlying driver which does not support synchronous behavior 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. Each CAN transceiver driver is steered independently.
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 device connected via ports.
EcuM	EcuM is notified of the wake up events by CAN transceiver driver.
ICU	CAN transceiver controls/accesses the hardware interrupts through the ICU module. ICU module invokes callbacks in CAN transceiver driver when interrupts occur.
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: If 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

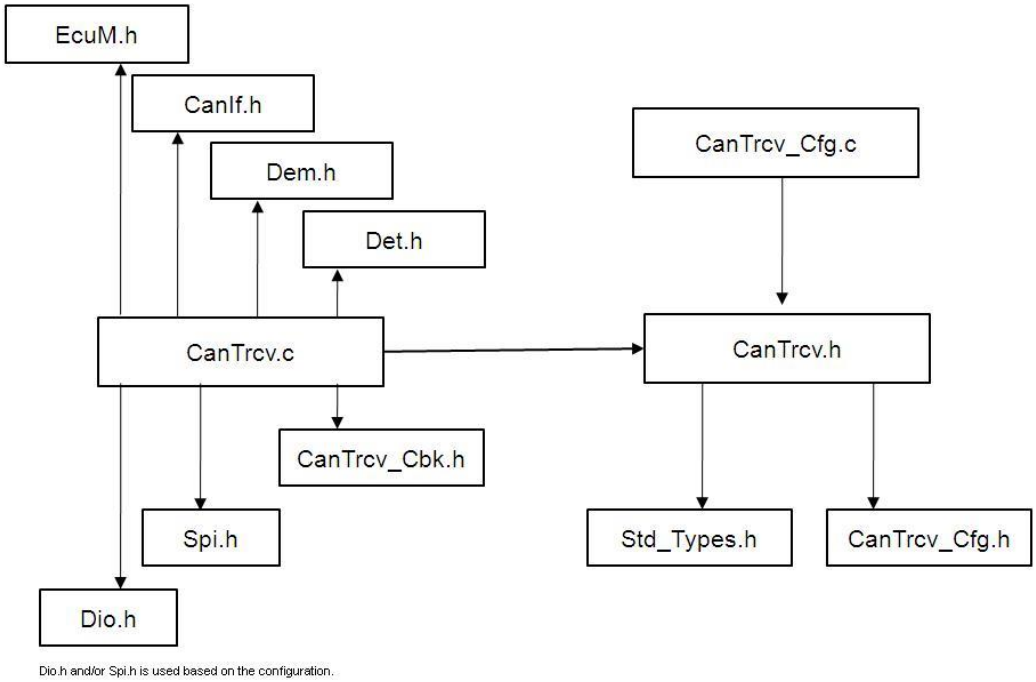
CanTrcv064: The naming convention is applied to all files of the CanTrcv module.

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	CanTrcv108
[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, CanTrcv154, CanTrcv153, CanTrcv152
[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	CanTrcv108
[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, CanTrcv154, CanTrcv153, CanTrcv152
[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, CanTrcv154, CanTrcv153, CanTrcv152
[BSW00370] Separation of callback interfaces from API	CanTrcv139
[BSW00371] Do not pass function pointers via API	CanTrcv001, CanTrcv002, CanTrcv005, CanTrcv007, CanTrcv008, CanTrcv009, CanTrcv012, CanTrcv013, CanTrcv154, CanTrcv153, CanTrcv152
[BSW00373] Main processing function naming convention	CanTrcv013
[BSW00374] Module vendor identification	CanTrcv108
[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	CanTrcv108
[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	Chapter 10
[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	CanTrcv108
[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, CanTrcv154, CanTrcv153, CanTrcv152
[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	not applicable

modules with AUTOSAR interfaces	(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	
[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 allowed.	CanTrcv002, CanTrcv005, CanTrcv007, CanTrcv009, CanTrcv012, CanTrcv016, CanTrcv017
[BSW01092] Configuration of bus operation mode after initialization for each CAN bus trans-ceiver	CanTrcv091
[BSW01095] Configuration “Notification for Wakeup by bus”	CanTrcv091
[BSW01096] API to initialize the CAN bus trans-ceiver driver	CanTrcv001
[BSW01097] CAN bus transceiver driver API shall be synchronous	CanTrcv001, CanTrcv002, CanTrcv005, CanTrcv007, CanTrcv009, CanTrcv012, CanTrcv013, CanTrcv154, CanTrcv153, CanTrcv152
[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 up-per layer	CanTrcv204, CanTrcv224
[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, CanTrcv154, CanTrcv153, CanTrcv152
[BSW01110] Handle timing requirements of transceiver	CanTrcv001, CanTrcv002, CanTrcv005, CanTrcv007, CanTrcv009, CanTrcv012, CanTrcv013, CanTrcv154, CanTrcv153, CanTrcv152
[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.

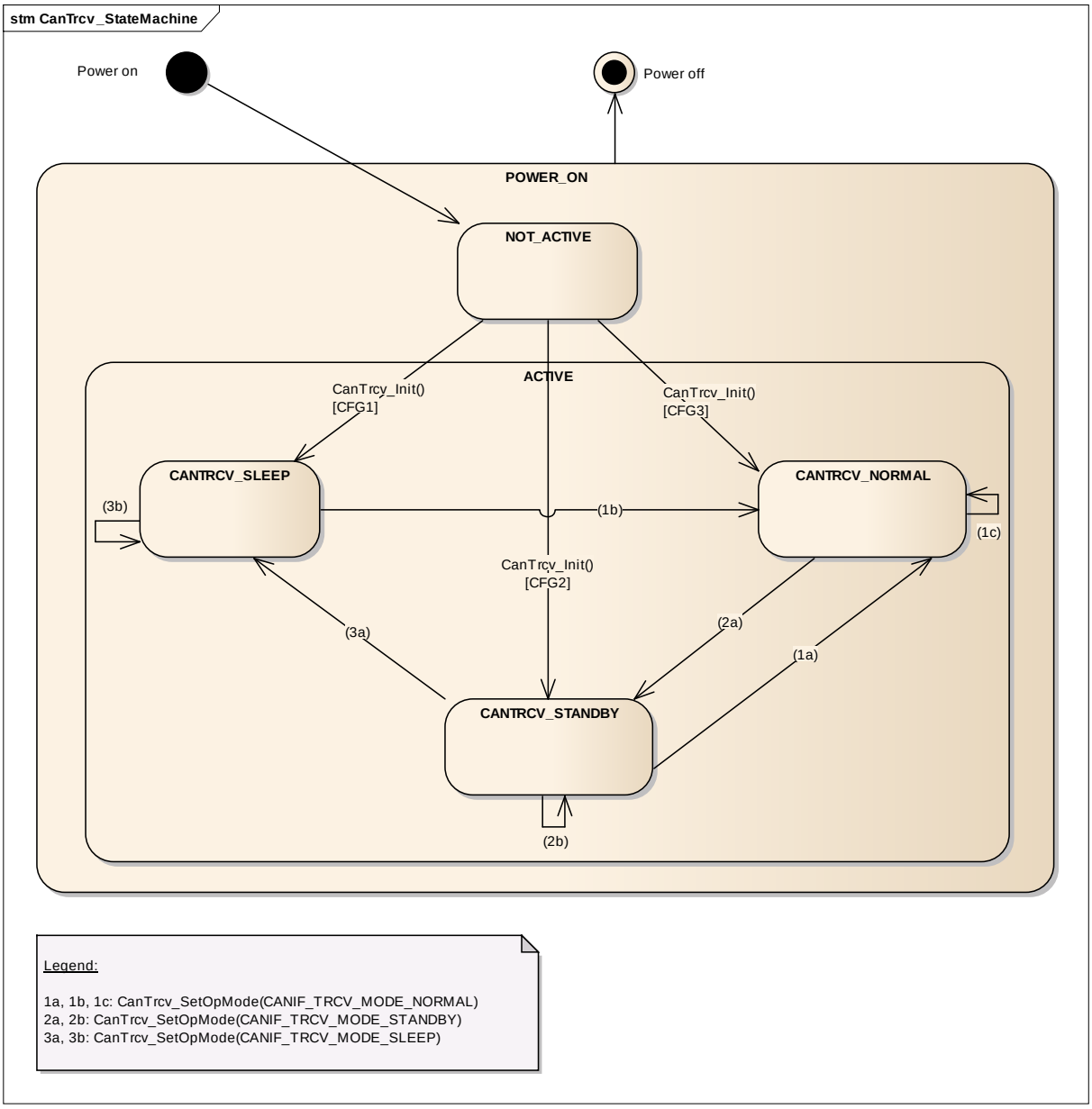


Figure 7.1: CanTrcv state diagram

The main idea intended by Figure 7.1, is to support a lot of currently available CAN transceivers in a generic view. Depending on the CAN transceiver hardware, the model may have one or two states more/less than necessary for a given CAN transceiver hardware. The standardization of transceiver driver as shown above 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 are explained in this chapter.

The implementer may decide the mapping between CAN transceiver hardware states and software states. Nevertheless, the implementation should guarantee that the whole functionality of the described software states of the CAN transceiver driver is realized.

7.2.1 Example: 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: “PowerOn/ListenOnly” mode

The mode often referred 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 suppressed. This mode is not supported because it is outside of the CAN standard and not supported by all CAN transceiver hardware chips.

7.3 Wakeup support

7.3.1 Wakeup 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.

7.3.2 Wakeup modes

CAN transceiver driver offers three wake up modes:

CanTrcv190: NOT_SUPPORTED mode

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

CanTrcv191: 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 support by the CAN transceiver hardware. Wake up mode POLLING requires callback function `CanTrcv_CB_WakeupByBus` and main function `CanTrcv_Main` to be present in source code.

CanTrcv192: ISR mode

In mode ISR, wake ups generated by CAN transceiver driver may cause CAN channel wake ups and MCU wake ups. This mode presumes support by used CAN transceiver hardware. 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 up may be switched on and off for each CAN transceiver channel individually by the configuration parameter `CanTrcvWakeupByBusUsed`.

Implementation Hints:

If a CAN transceiver needs a specific state transition (e.g. `CANTRCV_SLEEP` -> `CANTRCV_NORMAL`) initiated by the software after detection of a wakeup, 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 wakeup process (EcuM, CanIf, ICU etc…) that `CanTrcv_CB_WakeupByBus` is called, when a transceiver needs a specific state transition.

7.3.3 Enabling/Disabling wakeup notification

CanTrcv158: CanTrcv driver shall use the following APIs provided by ICU driver, to enable and disable the wakeup event notification:

- `Icu_EnableNotification`
- `Icu_DisableNotification`

CanTrcv driver shall ensure the following to avoid the loss of wakeup events:

CanTrcv159: It shall enable the ICU channels when the transceiver transitions to the Standby mode (`CANTRCV_STANDBY`).

CanTrcv219: It shall disable the ICU channels when the transceiver transitions to the Normal mode (`CANTRCV_NORMAL`).

7.4 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 of error	*Relevance*	*Related error code*	*Value [hex]*
API called with wrong parameter for CAN transceiver	Development	CANTRCV_E_INVALID_TRANSCEIVER	1
API called with NULL pointer as parameter	Development	CANTRCV_E_PARAM_POINTER	2
API used without initialization	Development	CANTRCV_E_UNINIT	11
API called in wrong transceiver operation mode	Development	CANTRCV_E_TRCV_NOT_STANDBY CANTRCV_E_TRCV_NOT_NORMAL	21 22
API called with invalid parameter for Wakeup Mode	Development	CANTRCV_E_PARAM_TRCV_WAKEUP_MODE	23
API called with invalid parameter for OpMode	Development	CANTRCV_E_PARAM_TRCV_OP_MODE	24
Configured baud rate is not supported by the transceiver	Development	CANTRCV_E_BAUDRATE_NOT_SUPPORTED	25
No/incorrect communication to transceiver.	Production	CANTRCV_E_NO_TRCV_CONTROL	*

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

7.5 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 7.4.

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.

7.6 Preconditions for driver initialization

CanTrcv196: The environment of the CanTrcv driver 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 transceiver driver uses functions of SPI, DIO and/or ICU drivers to control the CAN transceiver hardware. These drivers must be available and ready to operate before the CAN bus transceiver driver is initialized.

CanTrcv203: 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 underlying drivers.

The timing requirements might be like:

1) `CanTrcv_Init` API has to be called in pre-determined time after power up, for making all necessary information (like wakeup information) available to other modules.
2) The runtime of the modules used by CanTrcv has to be very short and synchronous to enable CanTrcv to keep its own timing requirements limited by hardware.
3) The runtime of the CanTrcv driver has to be enlarged due to some hardware devices requiring a port pin level to be valid for a particular time (e.g. 50µs), before changing it again to reach a specific state (e.g. Sleep).

7.7 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.

7.8 Wait states

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

CanTrcv138: CanTrcv driver may undergo wait states for switching between operation modes, based on the hardware capability of the transceiver.

7.9 Transceivers with selective wakeup functionality

This section describes requirements for CAN transceivers with selective wakeup functionality.

Partial Networking is a state in a CAN system where some nodes are in low power mode while other nodes are communicating. This reduces the power consumption by the entire network. Nodes in the low-power modes are woken up by pre-defined wakeup frames.

Transceivers which support selective wakeup can be woken up by Wake Up Frame/ Frames (WUF), in addition to the wakeup by Wake Up Pattern (WUP) offered by normal transceivers.

CanTrcv182: If selective wakeup is supported by the transceiver hardware, it shall be indicated with the configuration parameter `CanTrcvHwPnSupport`.

CanTrcv184: The configuration container for selective wakeup functionality (`CanTrcvPartialNetwork`) and the APIs:
`CanTrcv_GetTrcvSystemData`,
`CanTrcv_ClearWufFlag`,
`CanTrcv_ReadTrcvTimeoutFlag`,
`CanTrcv_ClearTrcvTimeoutFlag` and
`CanTrcv_ReadTrcvSilenceFlag`,
shall exist only if `CanTrcvHwPnSupport = TRUE`.

CanTrcv185: If selective wakeup is supported, CAN transceivers shall be configured to wake up on a particular CAN frame or a group of CAN frames using the parameters `CanTrcvPnFrameCanId`, `CanTrcvPnFrameCanIdMask` and `CanTrcvPnFrameDataMask`.

CanTrcv240: In order to implement the AUTOSAR Partial Networking mechanism Can Transceivers shall support the definition of a data mask for the Wake Up Frame (the configuration structure `of CanTrcvPnFrameDataMask` is mandatory).

CanTrcv186: If the transceiver has the ability to identify bus failures (and distinguish between bus failures and other hardware failures), it shall be indicated using the configuration parameter `CanTrcvBusErrFlag` for bus diagnostic purposes.

Note:

For CAN transceivers supporting selective wakeup functionality, detection of wakeup frames is possible during Normal mode (`CANTRCV_NORMAL`). Detected wakeup frames are signaled by the transceiver WUF flag. This ensures that no wakeup frame is lost during a transition to Standby mode (`CANTRCV_STANDBY`).

8 API specification

8.1 Imported types

This section describes the types imported from modules other than CanTrcv.

CanTrcv084:

Module	*Imported Type*
CanIf	CanIf_TransceiverModeType
	CanIf_TrcvWakeupModeType
	CanIf_TrcvWakeupReasonType
Dem	Dem_EventIdType
Dio	Dio_ChannelType
	Dio_LevelType
	Dio_PortLevelType
	Dio_PortType
	Dio_ChannelGroupType
EcuM	EcuM_WakeupSourceType
Icu	Icu_ChannelType
Spi	Spi_ChannelType
	Spi_DataType
	Spi_NumberOfDataType
	Spi_SequenceType
	Spi_StatusType
Std_Types	Std_ReturnType
	Std_VersionInfoType

8.2 Type definitions

This section specifies the types defined in the CanTrcv module.

8.2.1 CanTrcv_ConfigType

CanTrcv151:

Name:	`CanTrcv_ConfigType`	
Type:	`Structure`	
Range:	`Implementation specific`	--
Description:	This is the type of the external data structure containing the overall initialization data for the CAN transceiver driver and settings affecting all transceivers. Furthermore it contains pointers to transceiver configuration structures. The contents of the initialization data structure are CAN transceiver hardware specific.	

8.2.2 CanTrcv_PNActivationType

CanTrcv241:

Name:	`CanTrcv_PNActivationType`	
Type:	`Enumeration`	
Range:	`CANTRCV_PN_ENABLED`	PN wakeup functionality in CanTrcv is enabled.
	`CANTRCV_PN_DIABLED`	PN wakeup functionality in CanTrcv is disabled.
Description:	Datatype used for describing whether PN wakeup functionality in CanTrcv is ena-	

	bled or disabled.

8.2.3 CanTrcv_FlagStateType

CanTrcv229:

Name:	`CanTrcv_TrcvFlagStateType`	
Type:	`Enumeration`	
Range:	`CANTRCV_FLAG_SET`	The flag is set in the transceiver hardware.
	`CANTRCV_FLAG_CLEARED`	The flag is cleared in the transceiver hardware.
Description:	Provides the state of a flag in the transceiver hardware.	

8.3 Function definitions

8.3.1 CanTrcv_Init

CanTrcv001:

Service name:	CanTrcv_Init	
Syntax:	`void CanTrcv_Init( const CanTrcv_ConfigType* ConfigPtr )`	
Service ID[hex]:	0x00	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	ConfigPtr	Pointer to driver configuration.
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	None	
Description:	Initializes the CanTrcv module.	

CanTrcv181: The function `CanTrcv_Init` shall initialize all the connected CAN transceivers based on their initialization sequences and configuration (provided by parameter `ConfigPtr`). Meanwhile, it shall support the configuration sequence of the AUTOSAR stack also.

CanTrcv197: The function `CanTrcv_Init` shall set the CAN transceiver hardware to the state configured by the configuration parameter `CanTrcvInitState`.

Please 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 driver configurations.

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

Please refer CanTrcv196 also.

CanTrcv204: If supported by hardware, `CanTrcv_Init` shall validate whether there has been a wake up due to transceiver activity and if TRUE, reporting shall be done to EcuM via API `EcuM_SetWakeupEvent`.

CanTrcv205: If selective wakeup is enabled and supported by hardware: POR and SYSERR flags of the transceiver status shall be checked by `CanTrcv_Init` API.

CanTrcv206: If the POR flag or SYSERR flag is set, transceiver shall be re-configured for selective wakeup functionality by running the configuration sequence.

If the POR flag or SYSERR flag is not set, the configuration stored in the transceiver memory will be still valid and re-configuration is not necessary.

CanTrcv180: If the POR flag is set, wakeup shall be reported to EcuM through API `EcuM_SetWakeupEvent` with `CANIF_TRCV_WU_POWERON` as the wakeup reason.

CanTrcv208: If the SYSERR flag is set, wakeup shall be reported to EcuM through API `EcuM_SetWakeupEvent` with `CANIF_TRCV_WU_BY_SYSERR` as the wakeup reason.

CanTrcv113: If there is no/incorrect communication towards the transceiver, the function `CanTrcv_Init` shall report the production error `CANTRCV_E_NO_TRCV_CONTROL`.

CanTrcv223: If DET is enabled for CanTrcv module: the function `CanTrcv_Init` shall raise the development error `CANTRCV_E_BAUDRATE_NOT_SUPPORTED`, if the configured baud rate (parameter `CanTrcvBaudRate`) is not supported by the transceiver.

CanTrcv207: If DET is enabled for CanTrcv module: the function `CanTrcv_Init` shall raise the development error `CANTRCV_E_PARAM_POINTER`, if NULL pointer is passed as `ConfigPtr` parameter.

8.3.2 CanTrcv_SetOpMode

CanTrcv002:

Service name:	CanTrcv_SetOpMode	
Syntax:	`Std_ReturnType CanTrcv_SetOpMode( CanIf_TransceiverModeType OpMode, uint8 Transceiver )`	
Service ID[hex]:	0x01	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant for different transceivers	
Parameters (in):	OpMode	This parameter contains the desired operating mode
	Transceiver	CAN transceiver ID.
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 Transceiver to the value OpMode.	

CanTrcv198: The function CanTrcv_SetOpMode shall switch the internal state of channel Transceiver to the value of the parameter OpMode which can be CANTRCV_NORMAL, CANTRCV_STANDBY or CANTRCV_SLEEP.

CanTrcv199: The user of the CanTrcv module shall call the function CanTrcv_SetOpMode with OpMode = CANTRCV_STANDBY or CANTRCV_NORMAL, if the channel Transceiver is in mode CANTRCV_NORMAL.

CanTrcv200: The user of the CanTrcv module shall only call the function CanTrcv_SetOpMode with OpMode = CANTRCV_SLEEP or CANTRCV_STANDBY, if the channel Transceiver 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.

CanTrcv201: 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 buses is set up in the configuration phase.

CanTrcv220: If selective wakeup is supported by hardware: the flags POR and SYSERR of the transceiver status shall be checked by CanTrcv_SetOpMode API.

CanTrcv221: If the POR flag is set, transceiver shall be re-initialized to run the transceiver's configuration sequence.

CanTrcv222: If the SYSERR flag is NOT set, PN wakeup functionality has not been disabled and the requested mode is CANTRCV_NORMAL, transceiver shall call the API CanIf_ConfirmPnAvailability(Transceiver) for the corresponding Transceiver.

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 DET for the module CanTrcv is enabled: If the function CanTrcv_SetOpMode is called with OpMode = CANTRCV_ STANDBY and the channel Transceiver 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 DET for the module CanTrcv is enabled: If the function CanTrcv_SetOpMode is called with OpMode = CANTRCV_ SLEEP and the channel Transceiver 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 DET 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 DET for the module CanTrcv is enabled: If called with an invalid Transceiver ID, the function `CanTrcv_SetOpMode` shall raise the development error `CANTRCV_E_INVALID_TRANSCEIVER` and return `E_NOT_OK`.

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

8.3.3 CanTrcv_GetOpMode

CanTrcv005:

Service name:	CanTrcv_GetOpMode	
Syntax:	`Std_ReturnType CanTrcv_GetOpMode( CanIf_TransceiverModeType* OpMode, uint8 Transceiver )`	
Service ID[hex]:	0x02	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Transceiver	CAN transceiver ID.
Parameters (in-out):	None	
Parameters (out):	OpMode	Pointer to operation mode of the bus the API is applied to.
Return value:	Std_ReturnType	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 Transceiver and returns it in OpMode.	

CanTrcv202: 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 buses 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 DET 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 DET for the module CanTrcv is enabled: If called with an invalid Transceiver ID, the function `CanTrcv_GetOpMode` shall raise the development error `CANTRCV_E_INVALID_TRANSCEIVER` and return `E_NOT_OK`.

CanTrcv132: If DET 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`.

8.3.4 CanTrcv_GetBusWuReason

CanTrcv007:

Service name:	CanTrcv_GetBusWuReason	
Syntax:	`Std_ReturnType CanTrcv_GetBusWuReason( uint8 Transceiver, CanIf_TrcvWakeupReasonType* Reason )`	
Service ID[hex]:	0x03	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Transceiver	CAN transceiver ID.
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 wakeup reason for the Transceiver and returns it in parameter Reason.	

CanTrcv178: 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 buses 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 DET 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 DET for the module CanTrcv is enabled: If called with an invalid Transceiver ID, the function `CanTrcv_GetBusWuReason` shall raise development error `CANTRCV_E_INVALID_TRANSCEIVER` and return `E_NOT_OK`.

CanTrcv133: If DET 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 DET 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 DET 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 Transceiver )`
Service ID[hex]:	0x05
Sync/Async:	Synchronous

Reentrancy:	Reentrant for different transceivers	
Parameters (in):	TrcvWakeupMode	Requested transceiver wakeup reason
	Transceiver	CAN transceiver ID.
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 Transceiver according to TrcvWakeupMode.	

CanTrcv111: If the function `CanTrcv_SetWakeupMode` is called with `TrcvWakupMode = CANIF_TRCV_WU_ENABLE`, then the notifications for wakeup events are enabled on the addressed Transceiver. If the CanTrcv module has a stored wakeup event on the addressed Transceiver, the notification shall be sent within or immediately after `CanTrcv_SetWakeupMode`.

CanTrcv193: If the function `CanTrcv_SetWakeupMode` is called with `TrcvWakeupMode = CANIF_TRCV_WU_DISABLE`, then the notifications for wakeup events are disabled on the addressed Transceiver. The CAN transceiver module shall detect the wakeup events and store it internally in order to raise the event when the wakeup notification is enabled again.

CanTrcv194: If the function `CanTrcv_SetWakeupMode` is called with `TrcvWakeupMode = CANIF_TRCV_WU_CLEAR`, then a stored wakeup event is cleared on the addressed Transceiver. Clearing of wakeup events have to be used when the wake up notification is disabled, to clear all stored wake up events.

CanTrcv195: 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 buses is statically set in the configuration phase.

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

CanTrcv127: If DET 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 DET for the module CanTrcv is enabled: If called with an invalid Transceiver ID, the function `CanTrcv_SetWakeupMode` shall raise development error `CANTRCV_E_INVALID_TRANSCEIVER` and return `E_NOT_OK`.

CanTrcv089: If DET 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`.

8.3.7 CanTrcv_GetTrcvSystemData

CanTrcv152:

Service name:	CanTrcv_GetTrcvSystemData	
Syntax:	`Std_ReturnType CanTrcv_GetTrcvSystemData( uint8 Transceiver, const uint32* TrcvSysData )`	
Service ID[hex]:	0x09	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	Transceiver	CAN transceiver ID.
Parameters (in-out):	None	
Parameters (out):	TrcvSysData	Configuration/Status data of the transceiver.
Return value:	Std_ReturnType	E_OK: will be returned if the transceiver status is successfully read. E_NOT_OK: will be returned if the transceiver status data is not available or a development error occurs.
Description:	Reads the transceiver configuration/status data and returns it through parameter TrcvSysData.	

CanTrcv210: The function `CanTrcv_GetTrcvSystemData` shall read the configuration/status of the CAN transceiver and store the read data in the out parameter `TrcvSysData`. If this is successful, `E_OK` shall be returned.

Hint: This API can be invoked through diagnostic services or during initialization to determine the transceiver status and its availability.

Note: Currently an agreement on the parameter set for the transceiver HW specification has not been reached. For this reason, the diagnostic data is now returned as a uint32 (as stored in the transceiver registers). When a definitive and standard parameter set is defined, a data structure may be defined for abstracting the diagnostic data.

CanTrcv211: If there is no/incorrect communication to the transceiver, the function `CanTrcv_GetTrcvSystemData` shall report the production error `CANTRCV_E_NO_TRCV_CONTROL` and return `E_NOT_OK`.

CanTrcv216: If DET is enabled for the CanTrcv module: if called before the CanTrcv has been initialized, the function `CanTrcv_GetTrcvSystemData` shall raise development error `CANTRCV_E_UNINIT` and return `E_NOT_OK`.

CanTrcv212: If DET is enabled for the CanTrcv module: if called with an invalid Transceiver ID, function `CanTrcv_GetTrcvSystemData` shall raise the development error `CANTRCV_E_INVALID_TRANSCEIVER` and return `E_NOT_OK`.

CanTrcv213: If DET is enabled for the CanTrcv module: if called with NULL pointer for parameter `TrcvSysData`, function `CanTrcv_GetTrcvSystemData` shall raise the development error `CANTRCV_E_PARAM_POINTER` and return `E_NOT_OK`.

8.3.8 CanTrcv_ClearTrcvWufFlag

CanTrcv153:

Service name:	CanTrcv_ClearTrcvWufFlag	
Syntax:	`Std_ReturnType CanTrcv_ClearTrcvWufFlag( uint8 Transceiver )`	
Service ID[hex]:	0x0a	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant for different transceivers	
Parameters (in):	Transceiver	CAN transceiver ID.
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: will be returned if the WUF flag has been cleared. E_NOT_OK: will be returned if the WUF flag has not been cleared or a development error occurs.
Description:	Clears the WUF flag in the transceiver hardware.	

CanTrcv214: The function `CanTrcv_ClearTrcvWufFlag` shall clear the wakeup flag in the CAN transceiver. If successful, `E_OK` shall be returned.

Implementation Hints:

This API shall be used by the CanSM module for ensuring that no frame wakeup event is lost, during entering a low-power mode. This API clears the WUF flag.

The CAN transceiver shall be shall be put into Standby mode (`CANTRCV_STANDBY`) after clearing of the WUF flag.

If a system error (SYSERR, e.g. configuration error) occurs while selective wakeup functionality is being enabled, transceiver will disable the functionality. Transceiver will wake up on the next CAN wake pattern (WUP).
In case of any other hardware error (e.g. frame detection error), transceiver will wake up if the error counter inside the transceiver overflows.

CanTrcv239: CanTrcv shall inform CanIf that the wakeup flag has been cleared for the requested `Transceiver`, through the callback notification `CanIf_ClearTrcvWufFlagIndication`.

CanTrcv215: If there is no/incorrect communication to the transceiver, the function `CanTrcv_ClearTrcvWufFlag` shall report the production error `CANTRCV_E_NO_TRCV_CONTROL` and return `E_NOT_OK`.

CanTrcv217: If DET is enabled for the CanTrcv module: if called before the CanTrcv has been initialized, the function `CanTrcv_ClearTrcvWufFlag` shall raise development error `CANTRCV_E_UNINIT` and return `E_NOT_OK`

CanTrcv173: If DET is enabled for the CanTrcv module: if called with an invalid Transceiver ID, function `CanTrcv_ClearTrcvWufFlag` shall raise the development error `CANTRCV_E_INVALID_TRANSCEIVER` and return `E_NOT_OK`.

8.3.9 CanTrcv_ReadTrcvTimeoutFlag

CanTrcv226:

Service name:	CanTrcv_ReadTrcvTimeoutFlag	
Syntax:	`Std_ReturnType CanTrcv_ReadTrcvTimeoutFlag( uint8 Transceiver, CanTrcv_TrcvFlagStateType* FlagState )`	
Service ID[hex]:	0x0b	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	Transceiver	CAN transceiver ID.
Parameters (in-out):	None	
Parameters (out):	FlagState	State of the timeout flag.
Return value:	Std_ReturnType	E_OK: Will be returned, if status of the timeout flag is successfully read. E_NOT_OK: Will be returned, if status of the timeout flag could not be read.
Description:	Reads the status of the timeout flag from the transceiver hardware.	

CanTrcv230: If DET for the module CanTrcv is enabled: If called with an invalid Transceiver ID, the function `CanTrcv_ReadTrcvTimeoutFlag` shall raise the development error `CANTRCV_E_INVALID_TRANSCEIVER` and return `E_NOT_OK`.

CanTrcv231: If DET for the module CanTrcv is enabled: If called with `FlagState = NULL`, the function `CanTrcv_ReadTrcvTimeoutFlag` shall raise the development error `CANTRCV_E_PARAM_POINTER` and return `E_NOT_OK`.

8.3.10 CanTrcv_ClearTrcvTimeoutFlag

CanTrcv227:

Service name:	CanTrcv_ClearTrcvTimeoutFlag	
Syntax:	`Std_ReturnType CanTrcv_ClearTrcvTimeoutFlag( uint8 Transceiver )`	
Service ID[hex]:	0x0c	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	Transceiver	CAN transceiver ID.
Parameters (in-	None	

out):		
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Will be returned, if the timeout flag is successfully cleared. E_NOT_OK: Will be returned, if the timeout flag could not be cleared.
Description:	Clears the status of the timeout flag in the transceiver hardware.	

CanTrcv232: If DET for the module CanTrcv is enabled: If called with an invalid Transceiver ID, the function `CanTrcv_ClearTrcvTimeoutFlag` shall raise the development error `CANTRCV_E_INVALID_TRANSCEIVER` and return `E_NOT_OK`.

8.3.11 CanTrcv_ReadTrcvSilenceFlag

CanTrcv228:

Service name:	CanTrcv_ReadTrcvSilenceFlag	
Syntax:	`Std_ReturnType CanTrcv_ReadTrcvSilenceFlag( uint8 Transceiver, CanTrcv_TrcvFlagStateType* FlagState )`	
Service ID[hex]:	0x0d	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	Transceiver	CAN transceiver ID.
Parameters (in-out):	None	
Parameters (out):	FlagState	State of the silence flag.
Return value:	Std_ReturnType	E_OK: Will be returned, if status of the silence flag is successfully read. E_NOT_OK: Will be returned, if status of the silence flag could not be read.
Description:	Reads the status of the silence flag from the transceiver hardware.	

CanTrcv233: If DET for the module CanTrcv is enabled: If called with an invalid Transceiver ID, the function `CanTrcv_ReadTrcvSilenceFlag` shall raise the development error `CANTRCV_E_INVALID_TRANSCEIVER` and return `E_NOT_OK`.

CanTrcv234: If DET for the module CanTrcv is enabled: If called with `FlagState = NULL`, the function `CanTrcv_ReadTrcvSilenceFlag` shall raise the development error `CANTRCV_E_PARAM_POINTER` and return `E_NOT_OK`.

8.3.12 CanTrcv_CheckWakeFlag

CanTrcv236:

Service name:	CanTrcv_CheckWakeFlag
Syntax:	`Std_ReturnType CanTrcv_CheckWakeFlag( uint8 Transceiver`

	)	
Service ID[hex]:	0x0e	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	Transceiver	CAN transceiver ID.
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Will be returned, if the request for checking the wakeup flag has been accepted. E_NOT_OK: Will be returned, if the request for checking the wakeup flag has not been accepted.
Description:	Requests to check the status of the wakeup flag from the transceiver hardware.	

CanTrcv238: CanTrcv shall inform CanIf that a wakeup has been detected in the requested `Transceiver`, through the callback notification `CanIf_CheckTrcvWakeFlagIndication`.

CanTrcv237: If DET for the module CanTrcv is enabled: If called with an invalid Transceiver ID, the function `CanTrcv_CheckWakeFlag` shall raise the development error `CANTRCV_E_INVALID_TRANSCEIVER` and return `E_NOT_OK`.

8.3.13 CanTrcv_ SetPNActivationState

CanTrcv242:

Service name:	CanTrcv_SetPNActivationState	
Syntax:	`Std_ReturnType CanTrcv_SetPNActivationState( CanTrcv_PNActivationType ActivationState )`	
Service ID[hex]:	0x0f	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	ActivationState	CANTRCV_PN_ENABLED: PN wakeup functionality in CanTrcv shall be enabled. CANTRCV_PN_DIABLED: PN wakeup functionality in CanTrcv shall be disabled.
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Will be returned, if the PN wake-up functionality has been changed to the requested setting. E_NOT_OK: Will be returned, if the PN wake-up functionality change has failed.
Description:	The API configures the wake-up behavior of the transceiver for Standby and Sleep Mode: Either the CAN transceiver is woken up by a remote wake-up pattern (standard CAN wake-up) or by the configured remote wake-up frame.	

CanTrcv243: CanTrcv shall enable the PN wakeup functionality when function `CanTrcv_SetPNActivationState` is called with ActivationState= `PN_ENABLED` and return `E_OK`.

CanTrcv244: CanTrcv shall disable the PN wakeup functionality when function `CanTrcv_SetPNActivationState` is called with ActivationState= `PN_DISABLED` and return `E_OK`.

8.4 Scheduled functions

This chapter 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 buses 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 is to be kept. Function is called within `CanIf_MainFunction_Wakeup`.

See configuration parameter `CanTrcvWakeUpSupport`.

CanTrcv128: If DET 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`.

8.4.2 CanTrcv_MainFunctionDiagnostics

CanTrcv154:

Service name:	CanTrcv_MainFunctionDiagnostics

Syntax:	`void CanTrcv_MainFunctionDiagnostics(`
	
	`)`

Service ID[hex]:	0x08

Timing:	FIXED_CYCLIC
Description:	Reads the transceiver diagnostic status periodically and sets product/development accordingly.

CanTrcv174: The cyclic function `CanTrcv_MainFunctionDiagnostics` shall read the transceiver status periodically and report production/development errors accordingly.

CanTrcv183: The cyclic function `CanTrcv_MainFunctionDiagnostics` shall exist only if `CanTrcvBusErrFlag = TRUE`.

CanTrcv175: If configured and supported by hardware: if the BUSERR flag is set, function `CanTrcv_MainFunctionDiagnostics` shall set the production error `CANTRCV_E_BUS_ERROR`.

CanTrcv218: If DET for the module CanTrcv is enabled: If called before the CanTrcv has been initialized, the function `CanTrcv_MainFunctionDiagnostics` 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 Transceiver )`	
Service ID[hex]:	0x07	
Sync/Async:	Synchronous if wakeup timer is not configured, otherwise Asynchronous	
Reentrancy:	Reentrant	
Parameters (in):	Transceiver	CAN transceiver ID.
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 callback function is invoked by the CanIf module, if a wake up interrupt is detected. CanIf uses this callback to validate the wakeup interrupt.

Wakeup by bus is asynchronous to the transition to Sleep and Standby modes. In a worst possible case, the wakeup can occur during transition to Sleep

(CANTRCV_SLEEP) mode. Then, the CanTrcv driver shall create a wake up by bus notification immediately after CanTrcv_SetOpMode has finished.

The EcuM module must be able to handle the wake up event immediately after requesting the Standby or Sleep mode.

EcuM_EndCheckWakeup(WakeupSource) is called by canTrcv_CB_WakeupByBus for checking the wakeup source asynchronously.

Refer configuration parameter CanTrcvWakeUpSupport.

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

CanTrcv224: If supported by hardware, CanTrcv_CB_WakeupByBus shall validate whether there has been a wake up due to transceiver activity and if TRUE, reporting shall be done to EcuM via API EcuM_SetWakeupEvent.

CanTrcv135: If DET 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 and return E_NOT_OK.

CanTrcv136: If DET for the module CanTrcv is enabled: If called with an invalid Transceiver ID, the function CanTrcv_CB_WakeUpByBus shall raise the development error CANTRCV_E_INVALID_TRANSCEIVER.

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.
EcuM_SetWakeupEvent	Sets the wakeup event.

8.6.2 Optional Interfaces

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

CanTrcv086:

API function	*Description*
CanIf_CheckTrcvWakeFlagIndication	This service informs the upper layer modules that Wakeflag has

	been checked with the result that it either was not set or has been reset.
CanIf_ClearTrcvWufFlagIndication	This service indicates that the transceiver has cleared the WufFlag.
CanIf_ConfirmPnAvailability	This service indicates that the transceiver is running in PN communication mode.
CanIf_TrcvModeIndication	This service indicates that the transceiver mode has changed to TransceiverMode.
Det_ReportError	Service to report development errors.
Dio_ReadChannel	Returns the value of the specified DIO channel.
Dio_ReadChannelGroup	This Service reads a subset of the adjoining bits of a port.
Dio_ReadPort	Returns the level of all channels of that port.
Dio_WriteChannel	Service to set a level of a channel.
Dio_WriteChannelGroup	Service to set a subset of the adjoining bits of a port to a specified level.
Dio_WritePort	Service to set a value of the port.
Icu_DisableNotification	This function disables the notification of a channel.
Icu_EnableNotification	This function enables the notification on the given channel.
Spi_GetStatus	Service returns the SPI Handler/Driver software module status.
Spi_ReadIB	Service for reading synchronously one or more data from an IB SPI Handler/Driver Channel specified by parameter.
Spi_SetupEB	Service to setup the buffers and the length of data for the EB SPI Handler/Driver Channel specified.
Spi_SyncTransmit	Service to transmit data on the SPI bus
Spi_WriteIB	Service for writing one or more data to an IB SPI Handler/Driver Channel specified by parameter.

CanTrcv155: CanTrcv driver shall access the interfaces of the SPI module only if one or more instances of the container `CanTrcvSpiSequence` are configured.

CanTrcv156: CanTrcv driver shall access the interfaces of the DIO module only if one or more instances of the container `CanTrcvDioAccess` are configured.

CanTrcv157: CanTrcv driver shall enable/disable ICU channels only if reference is configured for the parameter `CanTrcvIcuChannelRef`.

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 other BSW modules.

Generic function call sequence has been provided in these sequence diagrams. Depending on the CAN transceiver hardware, there may be deviations to the sequences in the diagrams shown below (For e.g.: additional calls may be needed if a function returns ‘busy’ status on initial call).

9.1 Wake up sequence

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

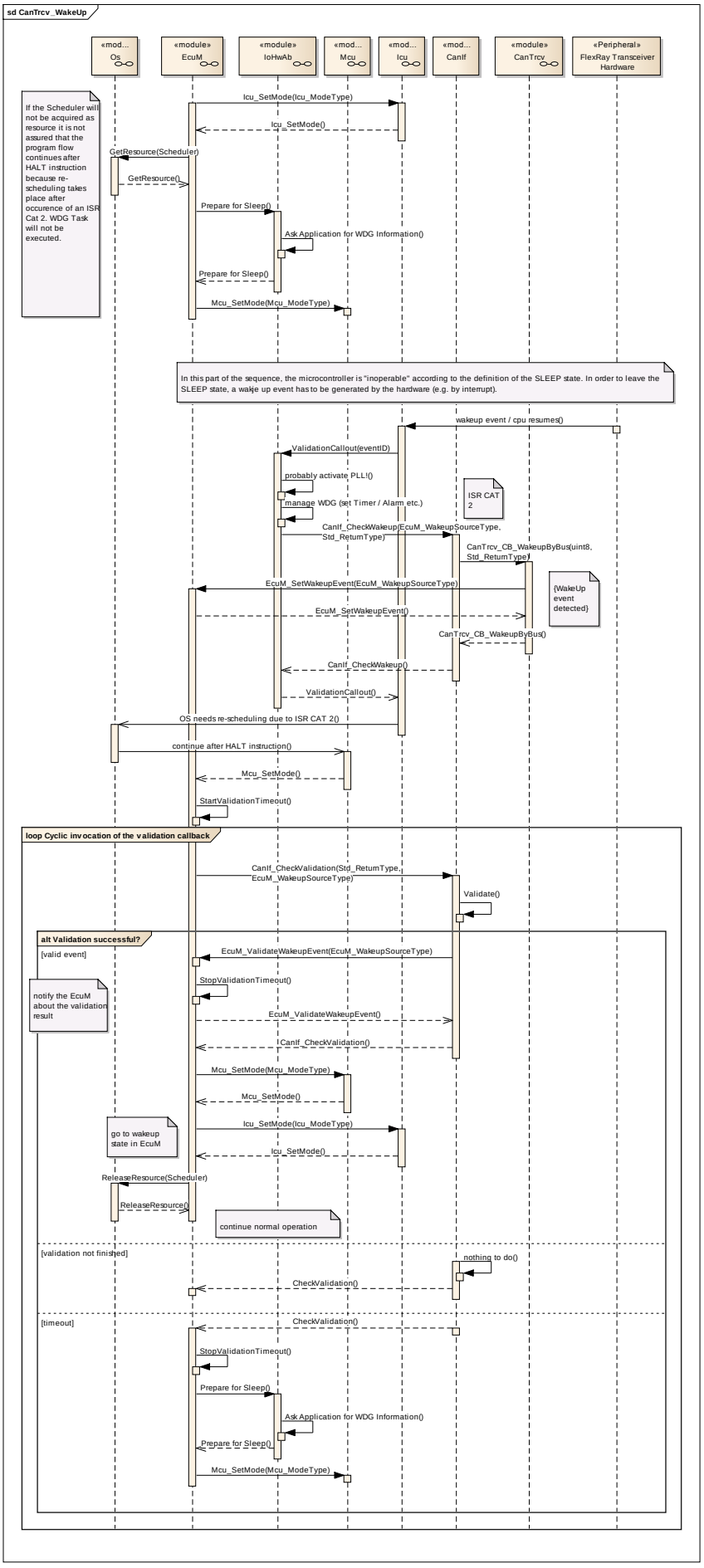

sd CanTrcv_WakeUp
«mod... Os
«module» EcuM
«module» IoHwAb
«mod... Mcu
«mod... Icu
«mod... CanIf
«module» CanTrcv
«Peripheral» FlexRay Transceiver Hardware
If the Scheduler will not be acquired as resource it is not assured that the program flow continues after HALT instruction because re-scheduling takes place after occurence of an ISR Cat 2. WDG Task will not be executed.
Icu_SetMode(Icu_ModeType)
Icu_SetMode()
GetResource(Scheduler)
GetResource()
Prepare for Sleep()
Ask Application for WDG Information()
Prepare for Sleep()
Mcu_SetMode(Mcu_ModeType)
In this part of the sequence, the microcontroller is "inoperable" according to the definition of the SLEEP state. In order to leave the SLEEP state, a wake up event has to be generated by the hardware (e.g. by interrupt).
wakeup event / cpu resumes()
ValidationCallout(eventID)
probably activate PLL!()
manage WDG (set Timer / Alarm etc.)
ISR CAT 2
CanIf_CheckWakeup(EcuM_WakeupSourceType, Std_ReturnType)
CanTrcv_CB_WakeupByBus(uint8, Std_ReturnType)
{WakeUp event detected}
EcuM_SetWakeupEvent(EcuM_WakeupSourceType)
EcuM_SetWakeupEvent()
CanTrcv_CB_WakeupByBus()
CanIf_CheckWakeup()
ValidationCallout()
OS needs re-scheduling due to ISR CAT 2()
continue after HALT instruction()
Mcu_SetMode()
StartValidationTimeout()
loop Cyclic invocation of the validation callback
CanIf_CheckValidation(Std_ReturnType, EcuM_WakeupSourceType)
Validate()
alt Validation successful?
[valid event]
EcuM_ValidateWakeupEvent(EcuM_WakeupSourceType)
StopValidationTimeout()
notify the EcuM about the validation result
EcuM_ValidateWakeupEvent()
CanIf_CheckValidation()
Mcu_SetMode(Mcu_ModeType)
Mcu_SetMode()
go to wakeup state in EcuM
Icu_SetMode(Icu_ModeType)
Icu_SetMode()
ReleaseResource(Scheduler)
ReleaseResource()
continue normal operation
[validation not finished]
nothing to do()
CheckValidation()
[timeout]
CheckValidation()
StopValidationTimeout()
Prepare for Sleep()
Ask Application for WDG Information()
Prepare for Sleep()
Mcu_SetMode(Mcu_ModeType)

Figure 9.1: Wake Up

9.2 Wake up with Validation

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

9.3 De-Initialization (SPI Synchronous)

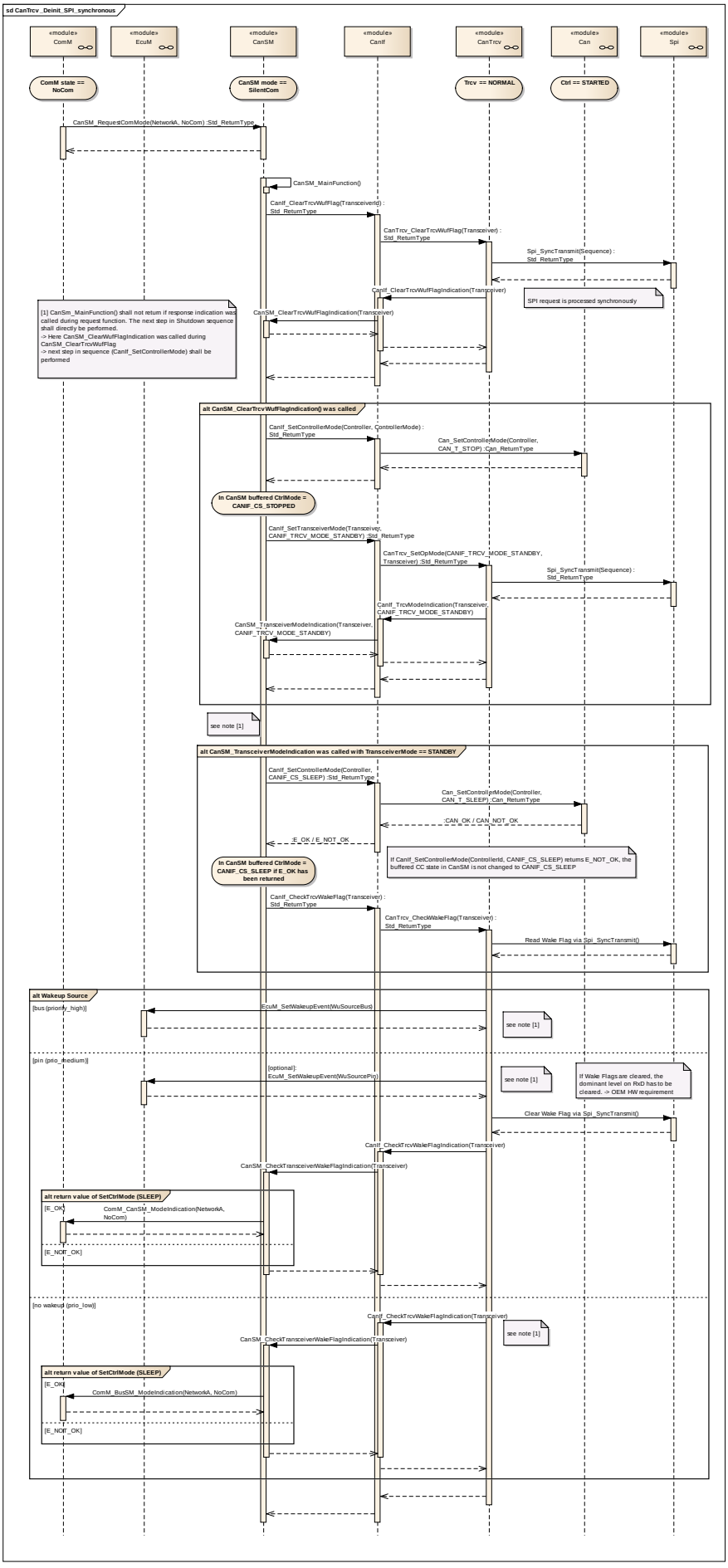

sd CanTrcv_Deinit_SPI_synchronous
«module» ComM
«module» EcuM
«module» CanSM
«module» CanIf
«module» CanTrcv
«module» Can
«module» Spi
ComM state == NoCom
CanSM mode == SilentCom
Trcv == NORMAL
Ctrl == STARTED
CanSM_RequestComMode(NetworkA, NoCom) :Std_ReturnType
CanSM_MainFunction()
CanIf_ClearTrcvWufFlag(TransceiverId) : Std_ReturnType
CanTrcv_ClearTrcvWufFlag(Transceiver) : Std_ReturnType
Spi_SyncTransmit(Sequence) : Std_ReturnType
CanIf_ClearTrcvWufFlagIndication(Transceiver)
SPI request is processed synchronously
[1] CanSm_MainFunction() shall not return if response indication was called during request function. The next step in Shutdown sequence shall directly be performed.
-> Here CanSM_ClearWufFlagIndication was called during CanSM_ClearTrcvWufFlag
-> next step in sequence (CanIf_SetControllerMode) shall be performed
CanSM_ClearTrcvWufFlagIndication(Transceiver)
alt CanSM_ClearTrcvWufFlagIndication() was called
CanIf_SetControllerMode(Controller, ControllerMode) : Std_ReturnType
Can_SetControllerMode(Controller, CAN_T_STOP) :Can_ReturnType
In CanSM buffered CtrlMode = CANIF_CS_STOPPED
CanIf_SetTransceiverMode(Transceiver, CANIF_TRCV_MODE_STANDBY) :Std_ReturnType
CanTrcv_SetOpMode(CANIF_TRCV_MODE_STANDBY, Transceiver) :Std_ReturnType
Spi_SyncTransmit(Sequence) : Std_ReturnType
CanIf_TrcvModeIndication(Transceiver, CANIF_TRCV_MODE_STANDBY)
CanSM_TransceiverModeIndication(Transceiver, CANIF_TRCV_MODE_STANDBY)
see note [1]
alt CanSM_TransceiverModeIndication was called with TransceiverMode == STANDBY
CanIf_SetControllerMode(Controller, CANIF_CS_SLEEP) :Std_ReturnType
Can_SetControllerMode(Controller, CAN_T_SLEEP) :Can_ReturnType
:CAN_OK / CAN_NOT_OK
:E_OK / E_NOT_OK
In CanSM buffered CtrlMode = CANIF_CS_SLEEP if E_OK has been returned
If CanIf_SetControllerMode(ControllerId, CANIF_CS_SLEEP) returns E_NOT_OK, the buffered CC state in CanSM is not changed to CANIF_CS_SLEEP
CanIf_CheckTrcvWakeFlag(Transceiver) : Std_ReturnType
CanTrcv_CheckWakeFlag(Transceiver) : Std_ReturnType
Read Wake Flag via Spi_SyncTransmit()
alt Wakeup Source
[bus (priority_high)]
EcuM_SetWakeupEvent(WuSourceBus)
see note [1]
[pin (prio_medium)]
[optional]: EcuM_SetWakeupEvent(WuSourcePin)
see note [1]
If Wake Flags are cleared, the dominant level on RxD has to be cleared. -> OEM HW requirement
Clear Wake Flag via Spi_SyncTransmit()
CanIf_CheckTrcvWakeFlagIndication(Transceiver)
CanSM_CheckTransceiverWakeFlagIndication(Transceiver)
alt return value of SetCtrlMode (SLEEP)
[E_OK]
ComM_CanSM_ModeIndication(NetworkA, NoCom)
[E_NOT_OK]
[no wakeup (prio_low)]
CanIf_CheckTrcvWakeFlagIndication(Transceiver)
see note [1]
CanSM_CheckTransceiverWakeFlagIndication(Transceiver)
alt return value of SetCtrlMode (SLEEP)
[E_OK]
ComM_BusSM_ModeIndication(NetworkA, NoCom)
[E_NOT_OK]

Figure 9.3: De-Init with synchronous SPI sequence

9.4 De-Initialization (SPI Asynchronous)

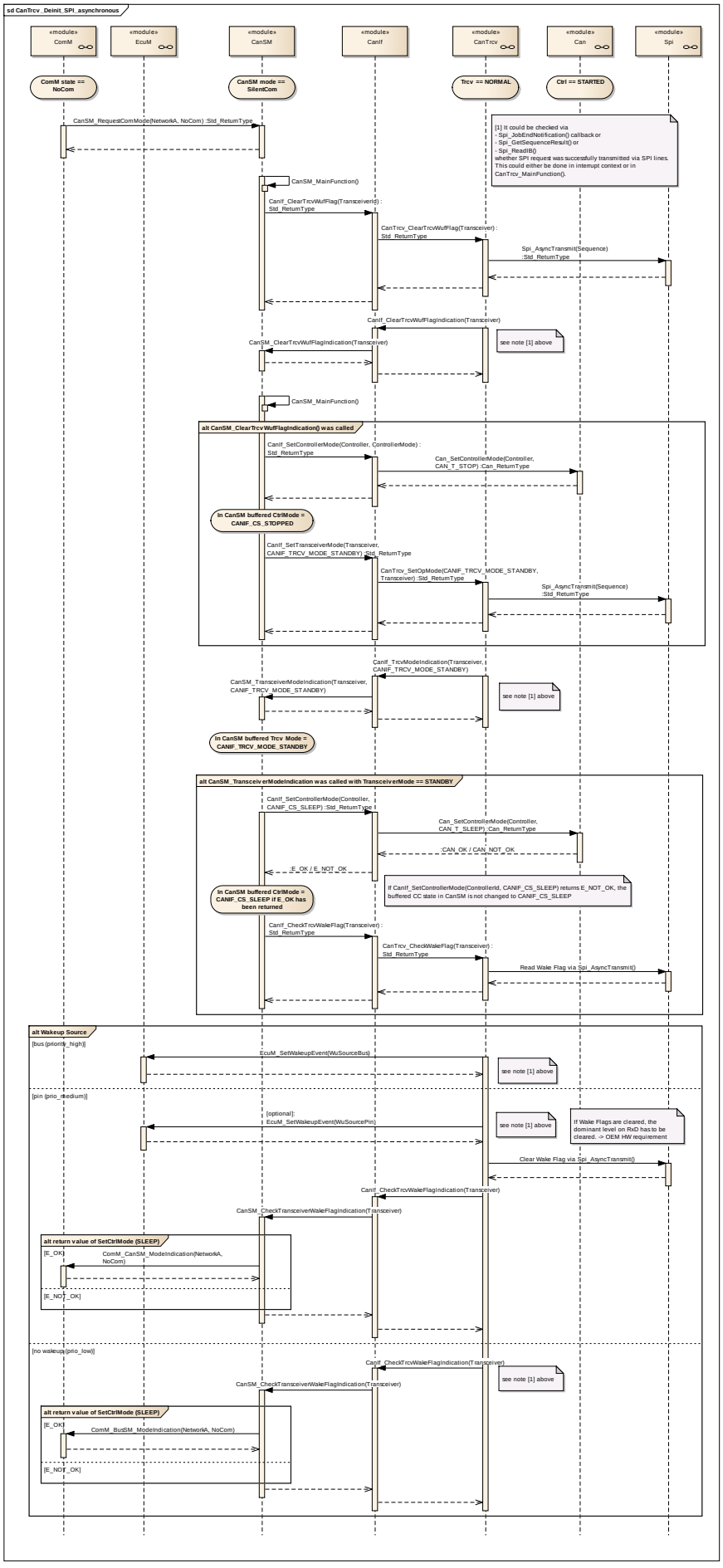

sd CanTrcv_DeInit_SPI_asynchronous
«module» ComM
«module» EcuM
«module» CanSM
«module» CanIf
«module» CanTrcv
«module» Can
«module» Spi
ComM state == NoCom
CanSM mode == SilentCom
Trcv == NORMAL
Ctrl == STARTED
CanSM_RequestComMode(NetworkA, NoCom) :Std_ReturnType
[1] It could be checked via
- Spi_JobEndNotification() callback or
- Spi_GetSequenceResult() or
- Spi_ReadIB()
whether SPI request was successfully transmitted via SPI lines.
This could either be done in interrupt context or in CanTrcv_MainFunction().
CanSM_MainFunction()
CanIf_ClearTrcvWufFlag(TransceiverId) : Std_ReturnType
CanTrcv_ClearTrcvWufFlag(Transceiver) : Std_ReturnType
Spi_AsyncTransmit(Sequence) :Std_ReturnType
CanIf_ClearTrcvWufFlagIndication(Transceiver)
CanSM_ClearTrcvWufFlagIndication(Transceiver)
see note [1] above
CanSM_MainFunction()
alt CanSM_ClearTrcvWufFlagIndication() was called
CanIf_SetControllerMode(Controller, ControllerMode) : Std_ReturnType
Can_SetControllerMode(Controller, CAN_T_STOP) :Can_ReturnType
In CanSM buffered CtrlMode = CANIF_CS_STOPPED
CanIf_SetTransceiverMode(Transceiver, CANIF_TRCV_MODE_STANDBY) :Std_ReturnType
CanTrcv_SetOpMode(CANIF_TRCV_MODE_STANDBY, Transceiver) :Std_ReturnType
Spi_AsyncTransmit(Sequence) :Std_ReturnType
CanIf_TrcvModeIndication(Transceiver, CANIF_TRCV_MODE_STANDBY)
CanSM_TransceiverModeIndication(Transceiver, CANIF_TRCV_MODE_STANDBY)
see note [1] above
In CanSM buffered Trcv_Mode = CANIF_TRCV_MODE_STANDBY
alt CanSM_TransceiverModeIndication was called with TransceiverMode == STANDBY
CanIf_SetControllerMode(Controller, CANIF_CS_SLEEP) :Std_ReturnType
Can_SetControllerMode(Controller, CAN_T_SLEEP) :Can_ReturnType
:CAN_OK / CAN_NOT_OK
:E_OK / E_NOT_OK
In CanSM buffered CtrlMode = CANIF_CS_SLEEP if E_OK has been returned
If CanIf_SetControllerMode(ControllerId, CANIF_CS_SLEEP) returns E_NOT_OK, the buffered CC state in CanSM is not changed to CANIF_CS_SLEEP
CanIf_CheckTrcvWakeFlag(Transceiver) : Std_ReturnType
CanTrcv_CheckWakeFlag(Transceiver) : Std_ReturnType
Read Wake Flag via Spi_AsyncTransmit()
alt Wakeup Source
[bus (priority_high)]
EcuM_SetWakeupEvent(WuSourceBus)
see note [1] above
[pin (prio_medium)]
[optional]:
EcuM_SetWakeupEvent(WuSourcePin)
see note [1] above
If Wake Flags are cleared, the dominant level on RxD has to be cleared. -> OEM HW requirement
Clear Wake Flag via Spi_AsyncTransmit()
CanIf_CheckTrcvWakeFlagIndication(Transceiver)
CanSM_CheckTransceiverWakeFlagIndication(Transceiver)
alt return value of SetCtrlMode (SLEEP)
[E_OK]
ComM_CanSM_ModeIndication(NetworkA, NoCom)
[E_NOT_OK]
[no wakeup (prio_low)]
CanIf_CheckTrcvWakeFlagIndication(Transceiver)
see note [1] above
CanSM_CheckTransceiverWakeFlagIndication(Transceiver)
alt return value of SetCtrlMode (SLEEP)
[E_OK]
ComM_BusSM_ModeIndication(NetworkA, NoCom)
[E_NOT_OK]

Figure 9.4: De-Init with asynchronous SPI sequence

10 Configuration specification

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 meta model 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-PRE-COMPILE: only pre-compile time configuration parameters; VARIANT-POST-BUILD: 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 sections summarize all configuration parameters for CanTrcv module. Detailed meanings of the parameters are described in preceding chapters.

10.2.1 Variants

CanTrcv176: VARIANT-PRE-COMPILE: Only pre-compile time configuration parameters.

CanTrcv177: VARIANT-POST-BUILD: Mix of pre-compile and post-build time configuration parameters.

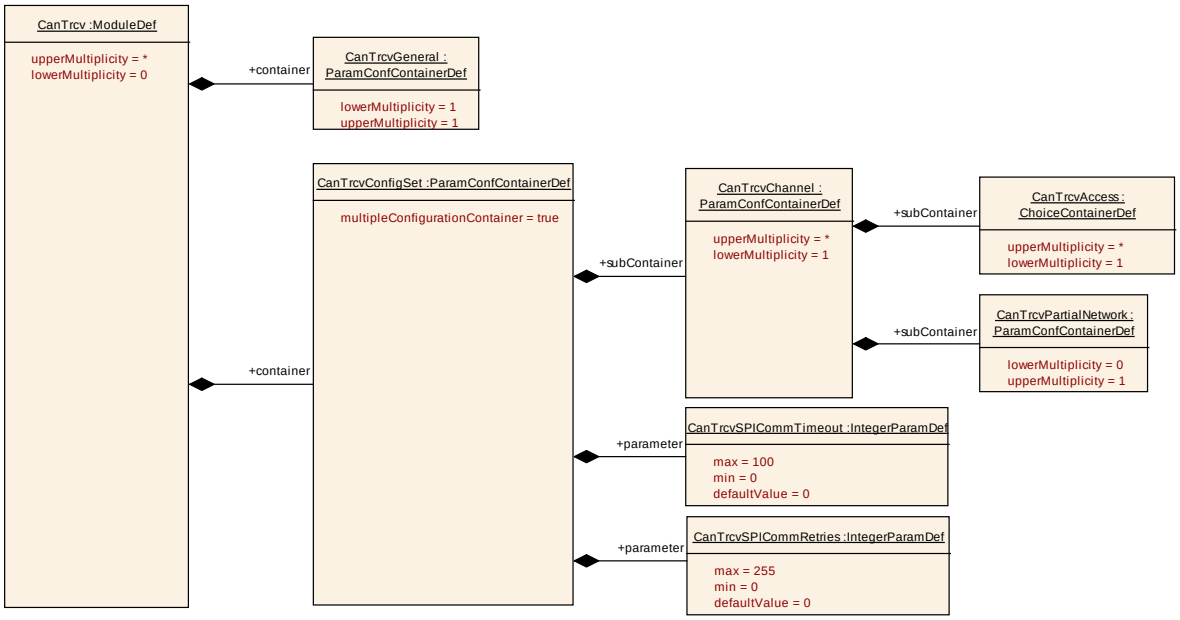


Figure 10.1: CanTrcv module configuration layout

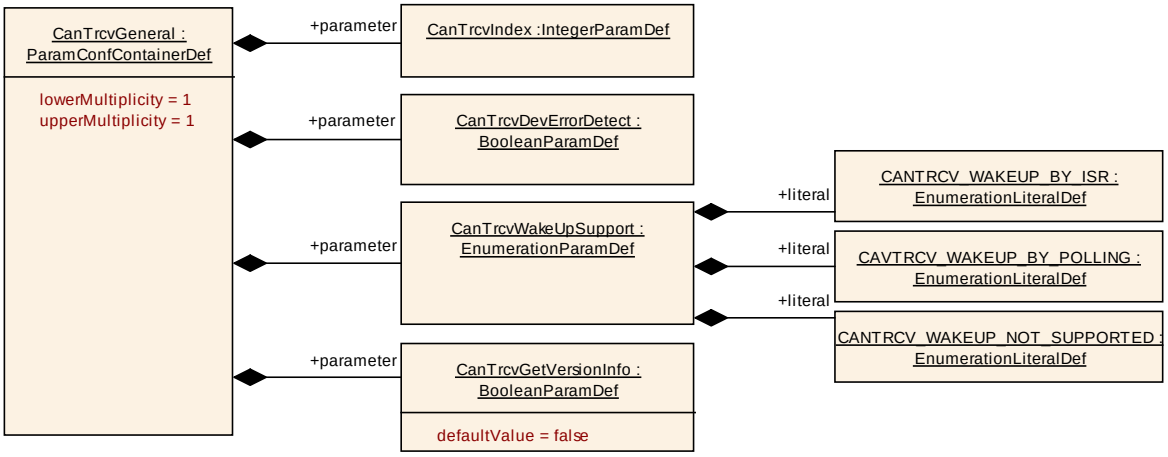


Figure 10.2: CanTrcvGeneral configuration layout

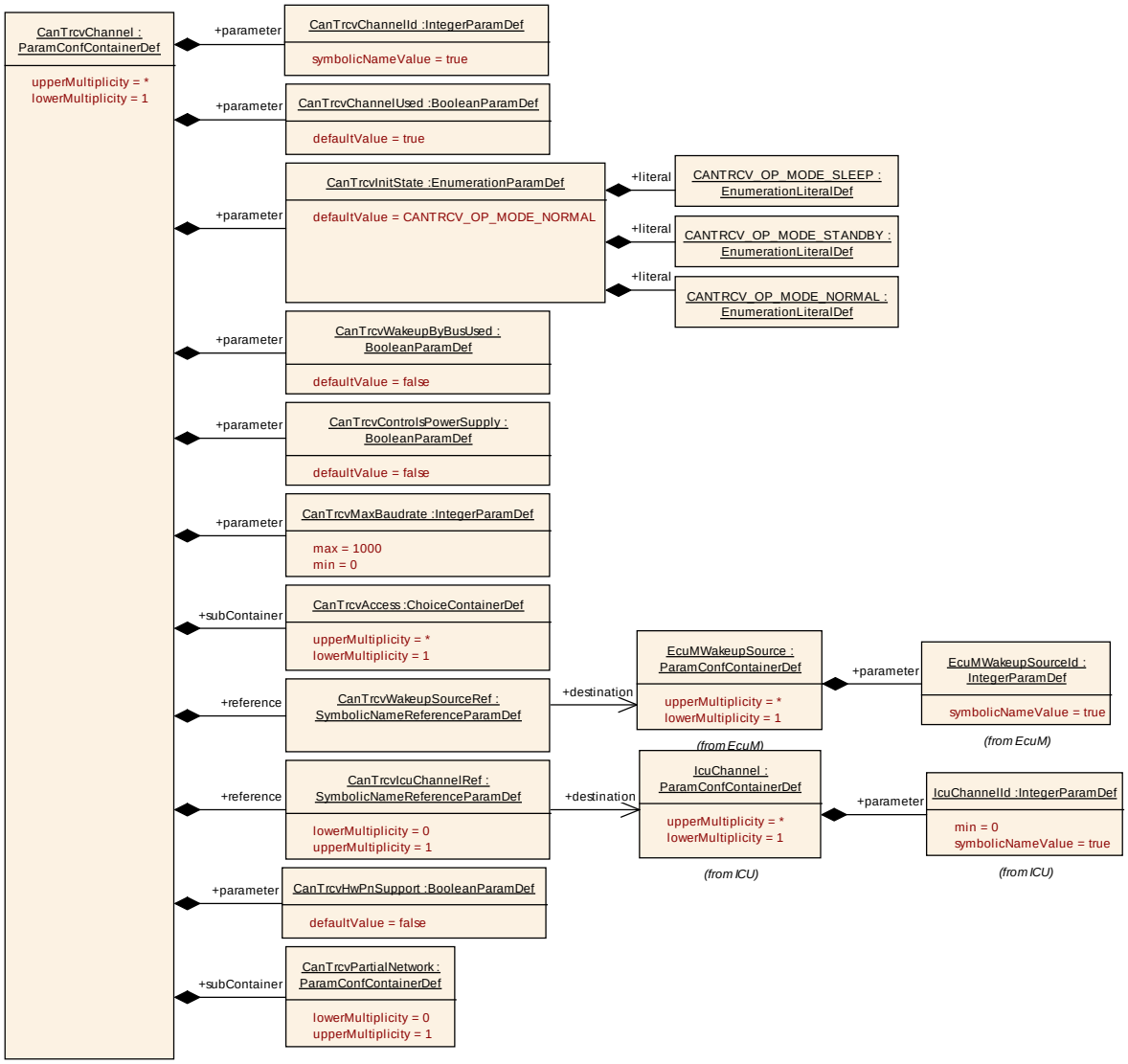


Figure 10.3: CanTrcvChannel configuration layout

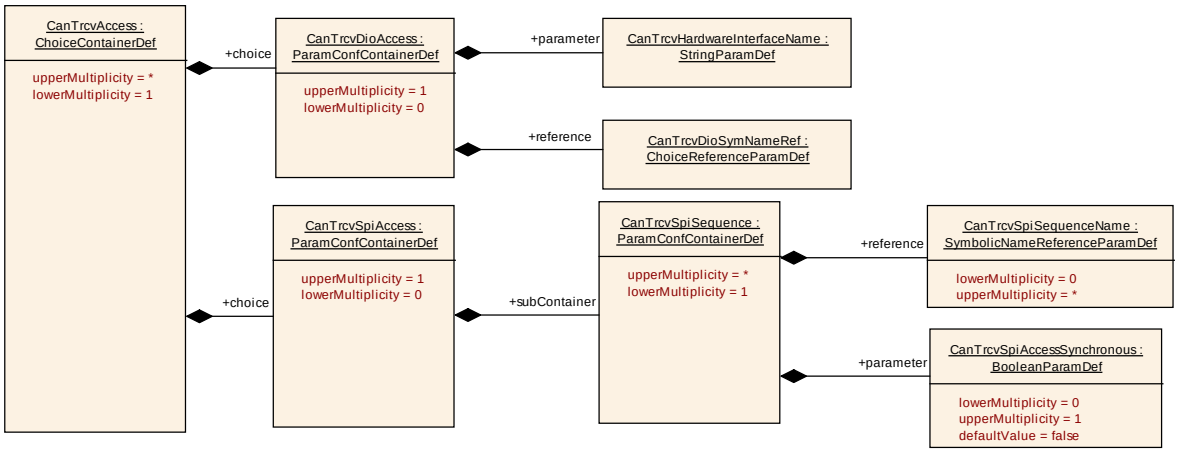
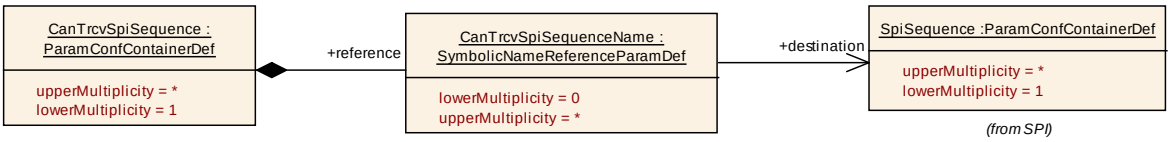


Figure 10.4: CanTrcvAccess configuration layout



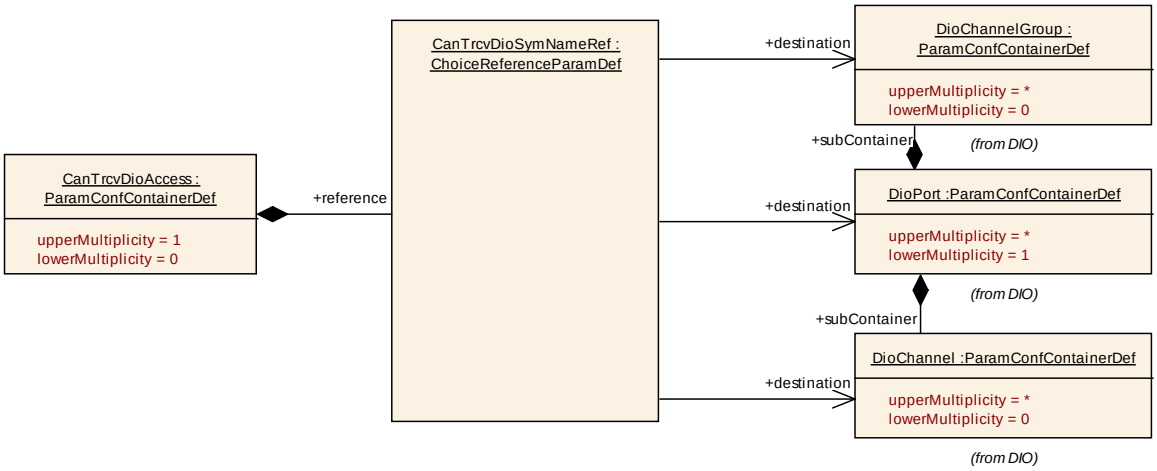


Figure 10.5: CanTrcv References configuration layout

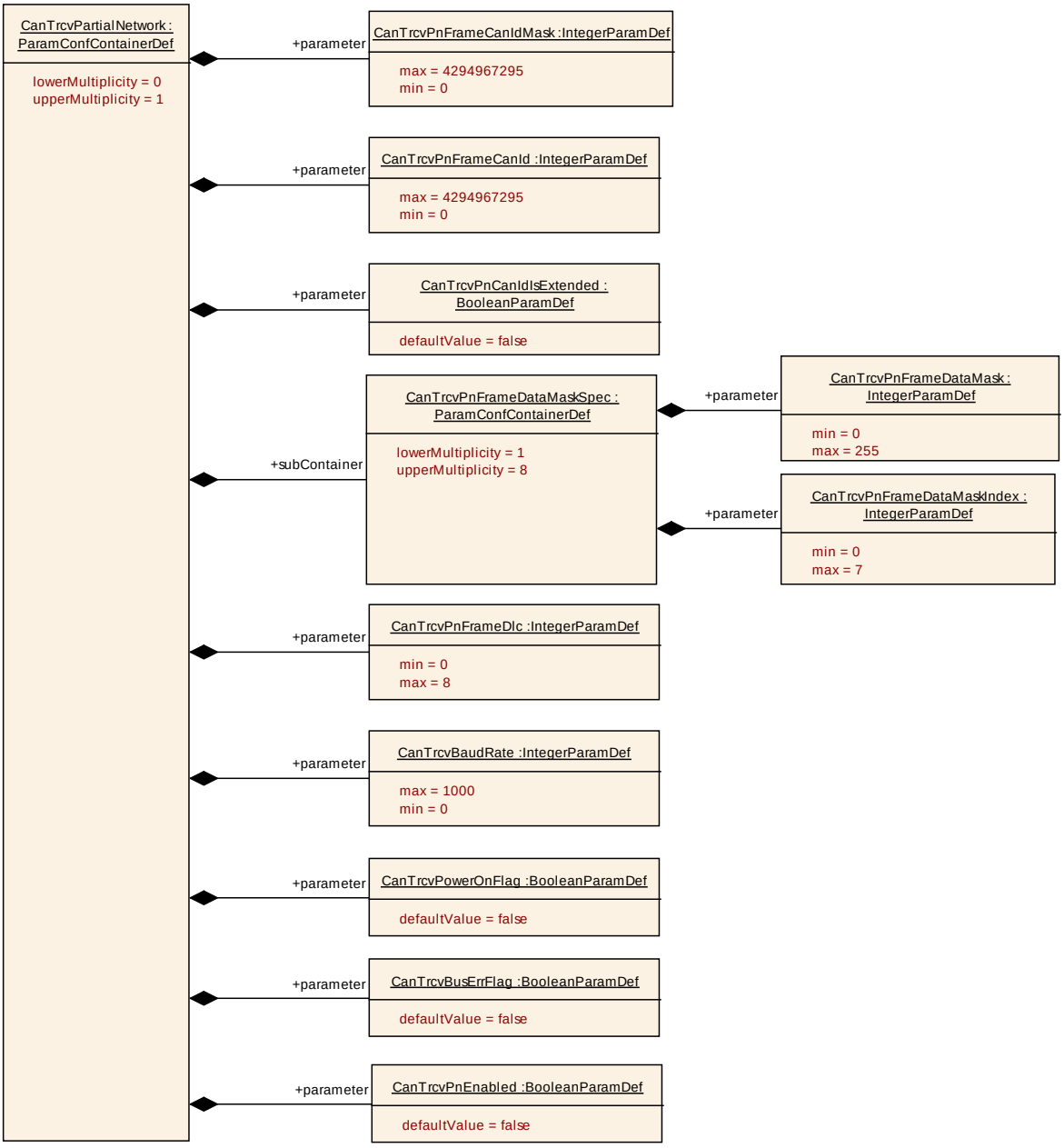


Figure 10.6: CanTrcvPartialNetwork configuration layout

10.2.2 CanTrcv

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

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CanTrcvConfigSet	1	This is the multiple configuration set container for CAN Transceiver.
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	**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		

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		
Range	..		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

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.
Multiplicity	1

Type	EnumerationParamDef		
Range	CANTRCV_WAKEUP_BY_ISR	Wake up by interrupt	
	CANTRCV_WAKEUP_NOT_SUPPORTED	Wake up is not supported	
	CAVTRCV_WAKEUP_BY_POLLING	Wake up by polling	
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module dependency: CanTrcvWakeupByBusUsed		

No Included Containers

10.2.4 CanTrcvConfigSet

SWS Item	**ECUC_CanTrcv_00173 :**
Container Name	CanTrcvConfigSet{CanTranceiverChannels} [Multi Config Container]
Description	This is the multiple configuration set container for CAN Transceiver.
Configuration Parameters	

SWS Item	**CanTrcv172 :**		
Name	CanTrcvSPICommRetries {CANTRCV_SPI_COMM_RETRIES}		
Description	Indicates the maximum number of communication retries in case of a failed SPI communication (applies both to timed out communication and to errors/NACK in the response data). If configured value is '0', no retry is allowed (communication is expected to succeed at first try).		
Multiplicity	1		
Type	IntegerParamDef		
Range	0 .. 255		
Default value	0		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Local dependency: This parameter exists only if at least one SPI Sequence is referenced in CanTrcvSpiSequence.		

SWS Item	**CanTrcv171 :**		
Name	CanTrcvSPICommTimeout {CANTRCV_SPI_COMM_TIMEOUT}		
Description	Indicates the maximum time allowed to the CanTrcv for replying (either positively or negatively) to a SPI command. Timeout is configured in milliseconds. Timeout value of '0' means that no specific timeout is to be used by CanTrcv and the communication is executed at the best of the SPI HW capacity.		
Multiplicity	1		
Type	IntegerParamDef		
Range	0 .. 100		
Default value	0		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Local dependency: This parameter exists only if at least one SPI Sequence is referenced in CanTrcvSpiSequence.		

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.

10.2.5 CanTrcvChannel

SWS Item	**ECUC_CanTrcv_00143 :**
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	**ECUC_CanTrcv_00155 :**		
Name	CanTrcvChannelId {CANTRCV_CHANNEL_ID}		
Description	Unique identifier of the CAN Transceiver Channel.		
Multiplicity	1		
Type	IntegerParamDef (Symbolic Name generated for this parameter)		
Range	..		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

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

SWS Item	**CanTrcv160 :**
Name	CanTrcvHwPnSupport {CANTRCV_HW_PN_SUPPORT}
Description	Indicates whether the transceiver has the ability to be selectively woken up by a

	Wake-up Frame (WUF). TRUE = Selective wakeup functionality is available in transceiver. FALSE = Selective wakeup functionality is not available in transceiver.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Local		

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_NORMAL	Normal operation mode (default)	
	CANTRCV_OP_MODE_SLEEP	Sleep operation mode	
	CANTRCV_OP_MODE_STANDBY	Standby 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		
Default value	--		
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		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Instance dependency: CanTrcvWakeUpSupport		

SWS Item	CanTrcv143 :		
Name	CanTrcvIcuChannelRef {CANTRCV_ICU_CHANNEL_REF}		
Description	Reference to the IcuChannel to enable/disable the interrupts for wakeups.		
Multiplicity	0..1		
Type	Reference to [IcuChannel]		
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..*	--
CanTrcvPartialNetwork	0..1	Container gives CAN transceiver driver information about the configuration of Partial Networking functionality.

10.2.6 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.
CanTrcvSpiAccess	0..1	Container gives CAN transceiver driver information about accessing Spi. If a CAN transceiver hardware has no Spi interface, there is no instance of this container.

10.2.7 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	--		
regularExpression	--		
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

10.2.8 CanTrcvSpiAccess

SWS Item	**CanTrcv183_CONF :**
Container Name	CanTrcvSpiAccess
Description	Container gives CAN transceiver driver information about accessing Spi. If a CAN transceiver hardware has no Spi interface, there is no instance of this container.
Configuration Parameters	

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CanTrcvSpiSequence	1..*	Container gives information about the SPI sequences used to

		access a CAN Transceiver device. It is assumed that one SPI sequence used by a CAN transceiver device is for its exclusive use.

10.2.9 CanTrcvSpiSequence

SWS Item	**CanTrcv092 :**
Container Name	CanTrcvSpiSequence{CanTransceiverSPISequences}
Description	Container gives information about the SPI sequences used to access a CAN Transceiver device. It is assumed that one SPI sequence used by a CAN transceiver device is for its exclusive use.
Configuration Parameters	

SWS Item	**CanTrcv235 :**		
Name	CanTrcvSpiAccessSynchronous {CANTRCV_SPI_ACCESS_SYNCHRONOUS}		
Description	This parameter is used to define whether the access to the Spi sequence is synchronous or asynchronous. true: SPI access is synchronous. false: SPI access is asynchronous.		
Multiplicity	0..1		
Type	BooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**CanTrcv104 :**		
Name	CanTrcvSpiSequenceName {CANTRCV_SPI_SEQUENCE_NAME}		
Description	Reference to a Spi sequence configuration container.		
Multiplicity	0..*		
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

10.2.10 CanTrcvPartialNetwork

SWS Item	**CanTrcv161 :**
Container Name	CanTrcvPartialNetwork
Description	Container gives CAN transceiver driver information about the configuration of Partial Networking functionality.
Configuration Parameters	

SWS Item	**CanTrcv167 :**
Name	CanTrcvBaudRate {CANTRCV_BAUD_RATE}

Description	Indicates the CAN Bus communication baud rate in kbps.		
Multiplicity	1		
Type	IntegerParamDef		
Range	0 .. 1000		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Local dependency: Although RWUF with DLC=0 is technically possible, it is explicitly not wanted.		

SWS Item	**CanTrcv169 :**		
Name	CanTrcvBusErrFlag {CANTRCV_BUS_ERR_FLAG}		
Description	Indicates if the Bus Error (BUSERR) flag is managed by the BSW. This flag is set if a bus failure is detected by the transceiver. TRUE = Supported by transceiver and managed by BSW. FALSE = Not managed by BSW.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Local		

SWS Item	**CanTrcv164 :**		
Name	CanTrcvPnCanIdIsExtended {CANTRCV_PN_CAN_ID_IS_EXTENDED}		
Description	Indicates whether extended or standard ID is used. TRUE = Extended Can identifier is used. FALSE = Standard Can identifier is used.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Local		

SWS Item	**ECUC_CanTrcv_00172 :**		
Name	CanTrcvPnEnabled {CANTRCV_PN_ENABLED}		
Description	Indicates whether the selective wake-up function is enabled or disabled in HW. TRUE = Selective wakeup feature is enabled in the transceiver hardware FALSE = Selective wakeup feature is disabled in the transceiver hardware		
Multiplicity	1		
Type	BooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Local		

SWS Item	**CanTrcv163 :**

Name	CanTrcvPnFrameCanId {CANTRCV_PN_FRAME_CAN_ID}		
Description	CAN ID of the Wake-up Frame (WUF).		
Multiplicity	1		
Type	IntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Local		

SWS Item	**CanTrcv162 :**		
Name	CanTrcvPnFrameCanIdMask {CANTRCV_PN_FRAME_CAN_ID_MASK}		
Description	ID Mask for the selective activation of the transceiver. It is used to enableFrame Wake-up (WUF) on a group of IDs.		
Multiplicity	1		
Type	IntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Local		

SWS Item	**CanTrcv166 :**		
Name	CanTrcvPnFrameDlc {CANTRCV_PN_FRAME_DLC}		
Description	Data Length of the Wake-up Frame (WUF).		
Multiplicity	1		
Type	IntegerParamDef		
Range	0 .. 8		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Local		

SWS Item	**CanTrcv168 :**		
Name	CanTrcvPowerOnFlag {CANTRCV_POWER_ON_FLAG}		
Description	Description: Indicates if the Power On Reset (POR) flag is available and is managed by the transceiver. TRUE = Supported by Hardware. FALSE = Not supported by Hardware.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Local		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CanTrcvPnFrameData-MaskSpec	1..8	Defines data payload mask to be used on the received payload in order to determine if the transceiver must be woken up by the received Wake-up Frame (WUF).

10.2.11 CanTrcvPnFrameDataMaskSpec

SWS Item	**CanTrcv165 :**
Container Name	CanTrcvPnFrameDataMaskSpec{CANTRCV_PN_FRAME_DATA_MASK_SPEC}
Description	Defines data payload mask to be used on the received payload in order to determine if the transceiver must be woken up by the received Wake-up Frame (WUF).
Configuration Parameters	

SWS Item	**CanTrcv170_CONF :**		
Name	CanTrcvPnFrameDataMask {CANTRCV_PN_FRAME_DATA_MASK}		
Description	Defines the n byte (Byte0 = LSB) of the data payload mask to be used on the received payload in order to determine if the transceiver must be woken up by the received Wake-up Frame (WUF).		
Multiplicity	1		
Type	IntegerParamDef		
Range	0 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Local		

SWS Item	**CanTrcv171_CONF :**		
Name	CanTrcvPnFrameDataMaskIndex {CANTRCV_PN_FRAME_DATA_MASK_INDEX}		
Description	holds the position n in frame of the mask-part		
Multiplicity	1		
Type	IntegerParamDef		
Range	0 .. 7		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Local		

No Included Containers

10.3 Published Information

The standardized published parameters as required by [BSW00402] in the General Requirements on Basic Software Modules (See Chapter 3), shall be published within the header file of this module and shall be provided in the BSW Module Description.

Corresponding module abbreviation can be found in the List of Basic Software Modules (See Chapter 3).

Additional module-specific published parameters are listed below if applicable.