Document Title	Specification of CAN Transceiver Driver
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	71

Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R23-11

Document Change History

Date	Release	Changed by	Description
2023-11-23	R23-11	AUTOSAR Release Management	• Editorial changes
2022-11-24	R22-11	AUTOSAR Release Management	• Added support for CanXL
2021-11-25	R21-11	AUTOSAR Release Management	• Updated state machine behavior for CanTrcv_Init • Editorial changes
2020-11-30	R20-11	AUTOSAR Release Management	• Modeling of Development Errors,Runtime Errors, and Transient Faults. • SOME/IP transformation props miss-ing is added. • Clean up of APIs with return type void, that specify a return value. • CanTrcv Operation Mode Inconsistencies corrected.

▽

△

2019-11-28	R19-11	AUTOSAR Release Management	• Sequence diagram De-Initialization (SPI Synchronous) and De-Initialization (SPI Asynchronous)split into different pages • Minor correction in CanTrcv initializa-tion functionality. • Changed Document Status from Fi-nal to published
2018-10-31	4.4.0	AUTOSAR Release Management	• Removed DET reporting behavior for the APIs CanTrcv_MainFunctionDiagnostics and CanTrcv_MainFunction during uninitialized state.
2017-12-08	4.3.1	AUTOSAR Release Management	• CanTrcv_DeInit API added in state machine diagram • Editorial changes 'Runtime errors'added
2016-11-30	4.3.0	AUTOSAR Release Management	• Added CanTrcv_DeInit API • Sequence diagram updated • CanTrcvGetVersionInfo renamed to CanTrcvVersionInfoApi • Updated Configuration class for configuration parameters • Minor corrections in the MainFunction periods
2014-10-31	4.2.1	AUTOSAR Release Management	• Revised the configuration of CAN Transceiver. • Minor corrections in wait state func-tionality. • Clarification regarding the wakeup sources.
2014-03-31	4.1.3	AUTOSAR Release Management	• Revised configuration for SPI inter-face. • Revised naming convention for transceiver driver
2013-10-31	4.1.2	AUTOSAR Release Management	• Removed 'Timing' row from sched-uled functions API table. • Editorial changes • Removed chapter(s) on change doc-umentation

▽

△

2013-03-15	4.1.1	AUTOSAR Release Management	• Updated sequence diagrams • Reworked according to the new SWS_BSWGeneral
2011-12-22	4.0.3	AUTOSAR Release Management	• Added support for Partial Networking • Implemented Production error concept • Updated Baud rate configuration pa-rameter handling • Added support to detect that power-on was caused by CAN communication • Reentrancy attribute is corrected for APIs • Corrections in few requirements • Optional Interfaces Table is corrected
2009-12-18	4.0.1	AUTOSAR Release Management	• CanTrcv state names changed and state diagram modified • Usage of SBCs are no longer restricted • Mode switch requests to the current mode are allowed • CanTrvc driver has to invoke CanIf_TrcvModeIndication after each mode switch request, when the requested mode has been reached
2010-02-02	3.1.4	AUTOSAR Release Management	• Wakeup event reporting: In R4.0,CanTrcv stores wakeup events.CanIf invokes function CanTrcv_CheckWakeup() periodical-ly to check for wakeup events. • Wakeup modes: In R4.0, wakeup through interrupt mechanism is not supported. Only POLLING and NOT_SUPPORTED wakeup modes are available in CanTrcv. • Sleep Wait Count added: Wait count for transitioning into sleep mode (CanTrcvSleepWaitCount) added. • Legal disclaimer revised

▽

△

2008-08-13	3.1.1.	AUTOSAR Release Management	• Legal disclaimer revised
2007-12-21	3.0.1	AUTOSAR Release Management	• 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
2007-01-24	2.1.15	AUTOSAR Release Management	• 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
2006-05-16	2.0	AUTOSAR Release Management	• Initial release

Disclaimer

Contents

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 recognised 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).

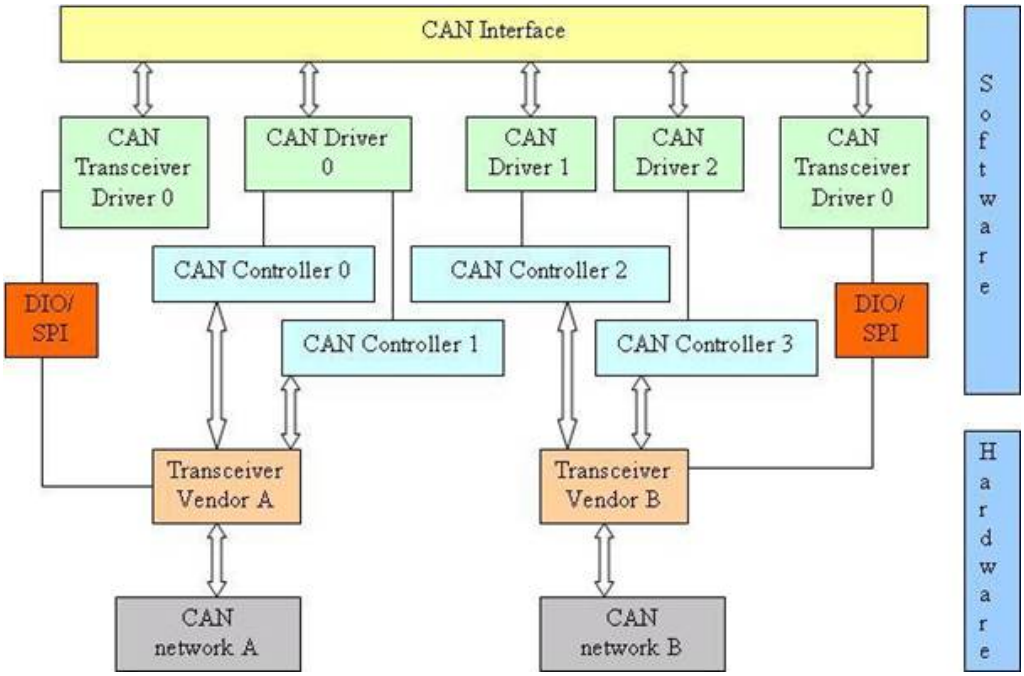

Figure 1.1: CAN Network Block Diagram

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 Single wire CAN transceivers according SAE J2411

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

2 Acronyms and Abbreviations

The glossary below includes acronyms and abbreviations relevant to the CAN Transceiver Driver module that are not included in the [1, AUTOSAR glossary].

Abbreviation:	Description:
ComM	Communication Manager
DEM	Diagnostic Event Manager
DET	Default Error Tracer
DIO	Digital Input Output (SPAL module)
EB	Externally Buffered channels. Buffers containing data to transfer are outside the SPI Handler/Driver.
EcuM	ECU State Manager
IB	Internally Buffered channels. Buffers containing data to transfer are inside the SPI Handler/Driver.
ISR	Interrupt Service Routine
MCAL	The MCAL, Microcontroller Abstraction Layer, is defined in AUTOSAR Layered Software Architecture [2]
Port	Port module (SPAL module)
n/a	Not Applicable
SBC	System Basis Chip; a device, which integrates e.g. CAN and/or LIN transceiver, watchdog and power control.
SPAL	Standard Peripheral Abstraction Layer
SPI Channel	A channel is a software exchange medium for data that are defined with the same criteria: configuration parameters, number of data elements with same size and data pointers (source & destination) or location. See specification of SPI driver for more details.
SPI Job	A job is composed of one or several channels with the same chip select. A job is considered to be atomic and therefore cannot be interrupted. A job has also an assigned priority. See specification of SPI driver for more details.
SPI Sequence	A sequence is a number of consecutive jobs to be transmitted. A sequence depends on a static configuration. See specification of SPI driver for more details.
CAN Channel	A physical channel which is connected to a CAN network from a CAN controller through a CAN transceiver.
API	Application Programming Interface

Table 2.1: Abbreviations used in the scope of this Document

3 Related documentation

3.1 Input documents & related standards and norms

[1] Glossary
AUTOSAR_FO_TR_Glossary

[2] Layered Software Architecture
AUTOSAR_CP_EXP_LayeredSoftwareArchitecture

[3] General Specification of Basic Software Modules
AUTOSAR_CP_SWS_BSWGeneral

[4] Specification of CAN Interface
AUTOSAR_CP_SWS_CANInterface

[5] Specification of CAN XL Transceiver Driver
AUTOSAR_CP_SWS_CANXLTransceiverDriver

[6] Specification of ECU State Manager
AUTOSAR_CP_SWS_ECUStateManager

3.2 Related specification

AUTOSAR provides a *General Specification on Basic Software modules* [3, SWS BSW General], which is also valid for CAN Transceiver Driver.

Thus, the specification SWS BSW General shall be considered as additional and required specification for CAN Transceiver Driver.

4 Constraints and assumptions

4.1 Limitations

[SWS_CanTrcv_00098] ⌈The CAN bus transceiver hardware shall provide the functionality and an interface which can be mapped to the operation mode model of the AUTOSAR CAN transceiver driver.⌋ *(SRS_BSW_00172)*

See also Chapter [7].

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 gets information about wake up events from CAN transceiver driver via CanIf.
SPI	SPI module is used to access CAN transceiver device connected via SPI.

5.1 File structure

5.1.1 Code file structure

[SWS_CanTrcv_00064] ⌈The naming convention prescribed by AUTOSAR is applied to all files of the CanTrcv module.⌋*(SRS_BSW_00300)*

[SWS_CanTrcv_00065] ⌈

File name	Requirements	Description
CanTrcv.c	SWS_CanTrcv_00069	The implementation general c file. It does not contain interrupt routines.
CanTrcv.h	SWS_CanTrcv_00052	It contains only information relevant for other BSW modules (API). Differences in API depending in configuration are encapsulated.

The CanTrcv module consists of these files.

⌋*()*

6 Requirements Tracability

Requirement	Description	Satisfied by
[SRS_BSW_00101]	The Basic Software Module shall be able to initialize variables and hardware in a separate initialization function	[SWS_CanTrcv_00001]
[SRS_BSW_00160]	Configuration files of AUTOSAR Basic SW module shall be readable for human beings	[SWS_CanTrcv_00090] [SWS_CanTrcv_00091] [SWS_CanTrcv_00093] [SWS_CanTrcv_00095]
[SRS_BSW_00171]	Optional functionality of a Basic-SW component that is not required in the ECU shall be configurable at pre-compile-time	[SWS_CanTrcv_00013]
[SRS_BSW_00172]	The scheduling strategy that is built inside the Basic Software Modules shall be compatible with the strategy used in the system	[SWS_CanTrcv_00001] [SWS_CanTrcv_00013] [SWS_CanTrcv_00090] [SWS_CanTrcv_00091] [SWS_CanTrcv_00098] [SWS_CanTrcv_00099]
[SRS_BSW_00300]	All AUTOSAR Basic Software Modules shall be identified by an unambiguous name	[SWS_CanTrcv_00064]
[SRS_BSW_00310]	API naming convention	[SWS_CanTrcv_00001] [SWS_CanTrcv_00002] [SWS_CanTrcv_00005] [SWS_CanTrcv_00007] [SWS_CanTrcv_00008] [SWS_CanTrcv_00009] [SWS_CanTrcv_00013]
[SRS_BSW_00327]	Error values naming convention	[SWS_CanTrcv_00050] [SWS_CanTrcv_00206] [SWS_CanTrcv_00227]
[SRS_BSW_00331]	All Basic Software Modules shall strictly separate error and status information	[SWS_CanTrcv_00206] [SWS_CanTrcv_00227]
[SRS_BSW_00336]	Basic SW module shall be able to shutdown	[SWS_CanTrcv_91001]
[SRS_BSW_00337]	Classification of development errors	[SWS_CanTrcv_00206] [SWS_CanTrcv_00227]
[SRS_BSW_00339]	Reporting of production relevant error status	[SWS_CanTrcv_00228]
[SRS_BSW_00343]	The unit of time for specification and configuration of Basic SW modules shall be preferably in physical time unit	[SWS_CanTrcv_00112]
[SRS_BSW_00347]	A Naming seperation of different instances of BSW drivers shall be in place	[SWS_CanTrcv_00016]
[SRS_BSW_00350]	All AUTOSAR Basic Software Modules shall allow the enabling/ disabling of detection and reporting of development errors.	[SWS_CanTrcv_00050]
[SRS_BSW_00357]	For success/failure of an API call a standard return type shall be defined	[SWS_CanTrcv_00002]
[SRS_BSW_00358]	The return type of init() functions implemented by AUTOSAR Basic Software Modules shall be void	[SWS_CanTrcv_00001]
[SRS_BSW_00369]	All AUTOSAR Basic Software Modules shall not return specific development error codes via the API	[SWS_CanTrcv_00001] [SWS_CanTrcv_00002] [SWS_CanTrcv_00005] [SWS_CanTrcv_00007] [SWS_CanTrcv_00008] [SWS_CanTrcv_00009] [SWS_CanTrcv_00013] [SWS_CanTrcv_91004] [SWS_CanTrcv_91005]

△

Requirement	Description	Satisfied by
[SRS_BSW_00373]	The main processing function of each AUTOSAR Basic Software Module shall be named according the defined convention	[SWS_CanTrcv_00013]
[SRS_BSW_00375]	Basic Software Modules shall report wake-up reasons	[SWS_CanTrcv_00007]
[SRS_BSW_00377]	A Basic Software Module can return a module specific types	[SWS_CanTrcv_00005] [SWS_CanTrcv_00007]
[SRS_BSW_00385]	List possible error notifications	[SWS_CanTrcv_00050] [SWS_CanTrcv_00206] [SWS_CanTrcv_00227] [SWS_CanTrcv_00228]
[SRS_BSW_00386]	The BSW shall specify the configuration and conditions for detecting an error	[SWS_CanTrcv_00050]
[SRS_BSW_00388]	Containers shall be used to group configuration parameters that are defined for the same object	[SWS_CanTrcv_00090] [SWS_CanTrcv_00091] [SWS_CanTrcv_00093] [SWS_CanTrcv_00095]
[SRS_BSW_00389]	Containers shall have names	[SWS_CanTrcv_00090] [SWS_CanTrcv_00091] [SWS_CanTrcv_00093] [SWS_CanTrcv_00095]
[SRS_BSW_00390]	Parameter content shall be unique within the module	[SWS_CanTrcv_00090] [SWS_CanTrcv_00091] [SWS_CanTrcv_00093] [SWS_CanTrcv_00095]
[SRS_BSW_00392]	Parameters shall have a type	[SWS_CanTrcv_00090] [SWS_CanTrcv_00091] [SWS_CanTrcv_00093] [SWS_CanTrcv_00095]
[SRS_BSW_00393]	Parameters shall have a range	[SWS_CanTrcv_00090] [SWS_CanTrcv_00091] [SWS_CanTrcv_00093] [SWS_CanTrcv_00095]
[SRS_BSW_00394]	The Basic Software Module specifications shall specify the scope of the configuration parameters	[SWS_CanTrcv_00090] [SWS_CanTrcv_00091] [SWS_CanTrcv_00093] [SWS_CanTrcv_00095]
[SRS_BSW_00395]	The Basic Software Module specifications shall list all configuration parameter dependencies	[SWS_CanTrcv_00091] [SWS_CanTrcv_00093] [SWS_CanTrcv_00095]
[SRS_BSW_00406]	A static status variable denoting if a BSW module is initialized shall be initialized with value 0 before any APIs of the BSW module is called	[SWS_CanTrcv_00002] [SWS_CanTrcv_00005] [SWS_CanTrcv_00007] [SWS_CanTrcv_00008] [SWS_CanTrcv_00009] [SWS_CanTrcv_00013]
[SRS_BSW_00407]	Each BSW module shall provide a function to read out the version information of a dedicated module implementation	[SWS_CanTrcv_00008]
[SRS_BSW_00408]	All AUTOSAR Basic Software Modules configuration parameters shall be named according to a specific naming rule	[SWS_CanTrcv_00090] [SWS_CanTrcv_00091] [SWS_CanTrcv_00093] [SWS_CanTrcv_00095]
[SRS_BSW_00411]	All AUTOSAR Basic Software Modules shall apply a naming rule for enabling/disabling the existence of the API	[SWS_CanTrcv_00008]
[SRS_BSW_00413]	An index-based accessing of the instances of BSW modules shall be done	[SWS_CanTrcv_00016]
[SRS_BSW_00414]	Init functions shall have a pointer to a configuration structure as single parameter	[SWS_CanTrcv_00001]
[SRS_BSW_00424]	BSW module main processing functions shall not be allowed to enter a wait state	[SWS_CanTrcv_00013]

▽

△

Requirement	Description	Satisfied by
[SRS_BSW_00425]	The BSW module description template shall provide means to model the defined trigger conditions of schedulable objects	[SWS_CanTrcv_00090]
[SRS_BSW_00428]	A BSW module shall state if its main processing function(s) has to be executed in a specific order or sequence	[SWS_CanTrcv_00013]
[SRS_Can_01090]	The bus transceiver driver package shall offer configuration parameters that are needed to configure the driver for a given bus and the supported notifications	[SWS_CanTrcv_00090] [SWS_CanTrcv_00091] [SWS_CanTrcv_00093] [SWS_CanTrcv_00095]
[SRS_Can_01091]	The CAN bus transceiver driver shall support the configuration for more than one bus	[SWS_CanTrcv_00002] [SWS_CanTrcv_00005] [SWS_CanTrcv_00007] [SWS_CanTrcv_00009] [SWS_CanTrcv_00016]
[SRS_Can_01095]	The bus transceiver driver shall support the compile time configuration of one notification to an upper layer for change notification for "wakeup by bus" events	[SWS_CanTrcv_00007]
[SRS_Can_01096]	The bus transceiver driver shall provide an API to initialize the driver internally	[SWS_CanTrcv_00001]
[SRS_Can_01097]	CAN Bus Transceiver driver API shall be synchronous	[SWS_CanTrcv_00001] [SWS_CanTrcv_00002] [SWS_CanTrcv_00005] [SWS_CanTrcv_00007] [SWS_CanTrcv_00009] [SWS_CanTrcv_00013]
[SRS_Can_01098]	The bus transceiver driver shall support an API to send the addressed transceiver into its Standby mode	[SWS_CanTrcv_00002] [SWS_CanTrcv_00055]
[SRS_Can_01099]	The bus transceiver driver shall support an API to send the addressed transceiver into its Sleep mode	[SWS_CanTrcv_00002] [SWS_CanTrcv_00055]
[SRS_Can_01100]	The bus transceiver driver shall support an API to send the addressed transceiver into its Normal mode	[SWS_CanTrcv_00002] [SWS_CanTrcv_00055]
[SRS_Can_01101]	The bus transceiver driver shall support an API to read out the current operation mode of the transceiver of a specified bus within the ECU	[SWS_CanTrcv_00005]
[SRS_Can_01103]	The bus transceiver driver shall support an API to read out the reason of the last wakeup of a specified bus within the ECU	[SWS_CanTrcv_00007]
[SRS_Can_01106]	The bus transceiver driver shall call the appropriate callback function of EcuM in case a wakeup by bus event is detected	[SWS_CanTrcv_00007]
[SRS_Can_01108]	The bus transceiver driver shall support the AUTOSAR ECU state manager in a way that a safe system startup and shutdown is possible	[SWS_CanTrcv_00001] [SWS_CanTrcv_00002] [SWS_CanTrcv_91001] [SWS_CanTrcv_91002] [SWS_CanTrcv_91003]
[SRS_Can_01109]	The bus transceiver driver shall check the control communication to the transceiver and the reaction of the transceiver for correctness	[SWS_CanTrcv_00001] [SWS_CanTrcv_00002] [SWS_CanTrcv_00005] [SWS_CanTrcv_00007] [SWS_CanTrcv_00009] [SWS_CanTrcv_00013]

▽

△

Requirement	Description	Satisfied by
[SRS_Can_01110]	CAN Bus Transceiver driver shall handle the transceiver specific timing requirements internally	[SWS_CanTrcv_00001] [SWS_CanTrcv_00002] [SWS_CanTrcv_00005] [SWS_CanTrcv_00007] [SWS_CanTrcv_00009] [SWS_CanTrcv_00013]
[SRS_Can_01115]	The bus transceiver driver shall support an API to enable and disable the wakeup notification for each bus separately	[SWS_CanTrcv_00009]
[SRS_Can_01157]	The bus transceiver driver shall provide an API for clearing the WUF bit in the tranceiver hardware	[SWS_CanTrcv_00214]

Table 6.1: RequirementsTracing

7 Functional specification

7.1 CAN transceiver driver operation modes

[SWS_CanTrcv_00055] ⌈The CanTrcv module shall implement the state diagram shown below, independently for each configured transceiver.⌋*(SRS_Can_01098, SRS_Can_01099, SRS_Can_01100)*

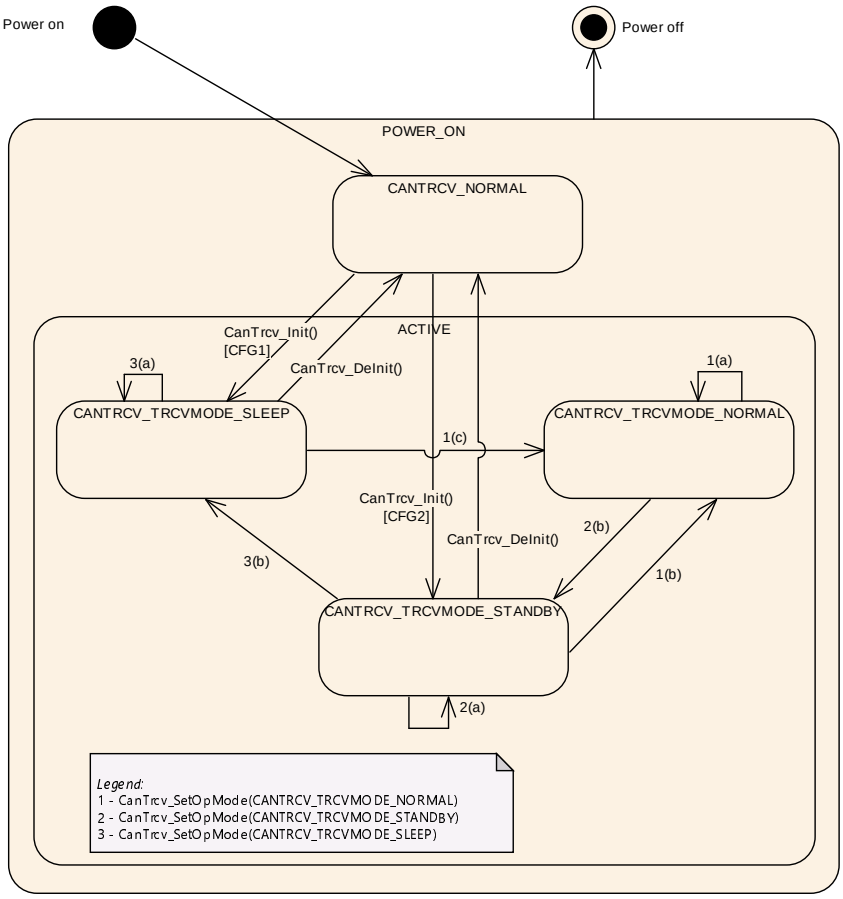

Figure 7.1: CanTrcv State Machine

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

[SWS_CanTrcv_00148] ⌈The function `CanTrcv_Init` causes a state change to either CANTRCV_TRCVMODE_SLEEP or CANTRCV_TRCVMODE _STANDBY. This depends on the configuration and is independently configurable for each transceiver.⌋ *()*

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 the state CANTRCV_TRCVMODE_SLEEP or CANTRCV_TRCVMODE_STANDBY.
CANTRCV_TRCVMODE_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_TRCVMODE_STANDBY	No communication is possible. ECU is still powered if CAN transceiver hardware controls ECU power supply. A transition to CANTRCV_TRCVMODE_SLEEP is only valid from this mode. A wake up by bus or by a local wake up event is possible.
CANTRCV_TRCVMODE_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 transceiver (configured as channels), all transceivers (channels) are either in the state NOT_ACTIVE or in the state ACTIVE.

In state ACTIVE, each transceiver 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`.

[SWS_CanTrcv_00161] ⌈A mode switch request to the current mode is allowed and shall not lead to an error, even if DET is enabled.⌋*()*

[SWS_CanTrcv_00158] ⌈The CanTrcv module shall invoke the callback function `CanIf_TrcvModeIndication`, for each mode switch request with call to `CanTrcv_SetOpMode`, after the requested mode has been reached referring to the corresponding CAN transceiver with the abstract CanIf TransceiverId. See see [4, Specification of Can Interface].⌋*()*

7.2 CAN transceiver hardware operation modes

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

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

7.2.1 Example for temporary "Go-To-Sleep" mode

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

7.2.2 Example for "PowerOn/ListenOnly" mode

The mode often 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 CAN transceiver wake up types

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

Scenario 1:

- MCU is not powered.
- Parts of ECU including CAN transceiver hardware are powered.
- The considered CAN transceiver is in SLEEP mode.
- A wake up event on CAN bus 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 is in STANDBY mode.
- A wake up event on CAN bus 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 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 the CAN channel.

7.4 Enabling/Disabling wakeup notification

[SWS_CanTrcv_00171] ⌈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 enable/disable ICU channels only if reference is configured for the parameter `CanTrcvIcuChannelRef`.⌋*()*

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

[SWS_CanTrcv_00172] ⌈It shall enable the ICU channels when the transceiver transitions to the Standby mode (CANTRCV_STANDBY).⌋*()*

[SWS_CanTrcv_00173] ⌈It shall disable the ICU channels when the transceiver transitions to the Normal mode (CANTRCV_NORMAL).⌋*()*

7.5 CAN transceiver wake up modes

CAN transceiver driver offers two wake up modes:

[SWS_CanTrcv_00090] ⌈NOT_SUPPORTED mode⌋*(SRS_BSW_00388, SRS_BSW_00389, SRS_BSW_00390, SRS_BSW_00392, SRS_BSW_00393, SRS_BSW_00394, SRS_BSW_00408, SRS_BSW_00425, SRS_BSW_00160, SRS_BSW_00172, SRS_Can_01090)*

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

[SWS_CanTrcv_00091] ⌈POLLING mode⌋*(SRS_BSW_00388, SRS_BSW_00389, SRS_BSW_00390, SRS_BSW_00392, SRS_BSW_00393, SRS_BSW_00394, SRS_-*

BSW_00395, SRS_BSW_00408, SRS_BSW_00160, SRS_BSW_00172, SRS_Can_-01090)

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

The main function `CanTrcv_MainFunction` shall be called by BSW scheduler and `CanTrcv_CheckWakeup` by CanIf.

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

Note: In both modes the function `CanTrcv_CheckWakeup` shall be present, but the functionality shall be based on the configured wakeup mode (NOT_SUPPORTED OR POLLING).

Implementation Hint:

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

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

7.6 Error Classification

Section "Error Handling" of the document [3] "General Specification of Basic Software Modules" describes the error handling of the Basic Software in detail. Above all, it constitutes a classification scheme consisting of five error types which may occur in BSW modules.

Based on this foundation, the following section specifies particular errors arranged in the respective subsections below.

7.6.1 Development Errors

[SWS_CanTrcv_00050] Definiton of development errors in module CanTrcv ⌈

Type of error	*Related error code*	*Error value*
API called with wrong parameter for the CAN transceiver	CANTRCV_E_INVALID_TRANSCEIVER	1
API called with null pointer parameter	CANTRCV_E_PARAM_POINTER	2
API service used without initialization	CANTRCV_E_UNINIT	11
API service called in wrong transceiver operation mode (STANDBY expected)	CANTRCV_E_TRCV_NOT_STANDBY	21
API service called in wrong transceiver operation mode (NORMAL expected)	CANTRCV_E_TRCV_NOT_NORMAL	22
API service called with invalid parameter for Trcv WakeupMode	CANTRCV_E_PARAM_TRCV_WAKEUP_MODE	23
API service called with invalid parameter for Op Mode	CANTRCV_E_PARAM_TRCV_OPMODE	24
Configured baud rate is not supported by the transceiver	CANTRCV_E_BAUDRATE_NOT_SUPPORTED	25
Module initialization has failed, e.g. CanTrcv_Init() called with an invalid pointer in postbuild.	CANTRCV_E_INIT_FAILED	27

⌋*(SRS_BSW_00327, SRS_BSW_00350, SRS_BSW_00385, SRS_BSW_00386)*

7.6.2 Runtime Errors

[SWS_CanTrcv_91006] Definiton of runtime errors in module CanTrcv ⌈

Type of error	*Related error code*	*Error value*
No/incorrect communication to transceiver.	CANTRCV_E_NO_TRCV_CONTROL	26

⌋*()*

7.6.3 Transient Faults

There are no transient faults.

7.6.4 Production Errors

There are no production errors.

7.6.5 Extended Production Errors

[SWS_CanTrcv_00228] ⌈

Error Name:	CANTRCV_E_BUS_ERROR	
Short Description:	A CAN bus error occured during communication,	
Long Description:	This Extended Production Error shall be issued when a bus failure is detected by the transceiver during the CAN communication.	
Detection Criteria:	Fail	When the flag corresponding to bus failure is set, Dem_SetEventStatus shall be reported with parameters EventId as CANTRCV_E_BUS_ERROR and EventStatus as DEM_EVENT_STATUS_FAILED. [SWS_CanTrcv_00206], [SWS_CanTrcv_00229]
	Pass	When the flag corresponding to bus failure is not set, Dem_SetEventStatus shall be reported with parameters EventId as CANTRCV_E_BUS_ERROR and EventStatus as DEM_EVENT_STATUS_PASSED. [SWS_CanTrcv_00227], [SWS_CanTrcv_00229]
Secondary Parameters:	N/A	
Time Required:	N/A	
Monitor Frequency:	continuous	

⌋*(SRS_BSW_00339, SRS_BSW_00385)*

[SWS_CanTrcv_00229] ⌈The extended production error CANTRCV_E_BUS_ERROR (value assigned by DEM) shall be detectable by the CAN tranceiver module when Bus Error (BUSERR) flag is set, depending on whether it is configured and supported by hardware.⌋*()*

7.7 Preconditions for driver initialization

[SWS_CanTrcv_00099] ⌈The environment of the CanTrcv module shall make sure that all necessary BSW drivers (used by the CanTrcv module) have been initialized and are usable before `CanTrcv_Init` is called.⌋*(SRS_BSW_00172)*

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

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

The timing requirements might be that

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

7.8 Instance concept

[SWS_CanTrcv_00016] ⌈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.⌋*(SRS_BSW_00347, SRS_BSW_00413, SRS_Can_01091)*

7.9 Wait states

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

[SWS_CanTrcv_00230] ⌈The CAN Tranceiver Driver shall use the Time service Tm_BusyWait1us16bit to realize the wait time for transceiver state changes.⌋*()*

7.10 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.

[SWS_CanTrcv_00174] ⌈If selective wakeup is supported by the transceiver hardware, it shall be indicated with the configuration parameter `CanTrcvHwPnSupport`.⌋*()*

[SWS_CanTrcv_00175] ⌈The configuration container for selective wakeup functionality (CanTrcvPartialNetwork) and for the following APIs:

- `CanTrcv_GetTrcvSystemData`,
- `CanTrcv_ClearTrcvWufFlag`,
- `CanTrcv_ReadTrcvTimeoutFlag`,
- `CanTrcv_ClearTrcvTimeoutFlag` and
- `CanTrcv_ReadTrcvSilenceFlag`

shall exist only if CanTrcvHwPnSupport = TRUE.⌋*()*

[SWS_CanTrcv_00177] ⌈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`.⌋*()*

[SWS_CanTrcv_00178] ⌈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_TRCVMODE_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_TRCVMODE_STANDBY).

7.11 CAN XL Extension

The CAN XL Transceiver Driver is implemented as an extension for the existing CAN Transceiver Driver (see [5, Specification of Can XL]), non CAN XL hardware will still use basic CAN Transceiver Driver implementation.

The CAN XL Transceiver Driver is an extension of CAN Transceiver Driver and introduces an additional API to support Ethernet interface and provides a mode interface to CAN XL Transceiver Driver (see [5, Specification of Can XL] for further details).

7.12 Security Events

The module does not report security events.

8 API specification

8.1 Imported types

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

[SWS_CanTrcv_00084] Definition of imported datatypes of module CanTrcv ⌈

Module	*Header File*	*Imported Type*
Dem	Rte_Dem_Type.h	Dem_EventIdType
	Rte_Dem_Type.h	Dem_EventStatusType
Dio	Dio.h	Dio_ChannelGroupType
	Dio.h	Dio_ChannelType
	Dio.h	Dio_LevelType
	Dio.h	Dio_PortLevelType
	Dio.h	Dio_PortType
EcuM	EcuM.h	EcuM_WakeupSourceType
Icu	Icu.h	Icu_ChannelType
Spi	Spi.h	Spi_ChannelType
	Spi.h	Spi_DataBufferType
	Spi.h	Spi_NumberOfDataType
	Spi.h	Spi_SequenceType
	Spi.h	Spi_StatusType
Std	Std_Types.h	Std_ReturnType
	Std_Types.h	Std_VersionInfoType

⌋()

8.2 Type definitions

[SWS_CanTrcv_00209] Definition of datatype CanTrcv_ConfigType ⌈

Name	CanTrcv_ConfigType	
Kind	Structure	
Elements	Implementation specific	
	Type	–
	Comment	–
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.	
Available via	CanTrcv.h	

⌋()

[SWS_CanTrcv_00210] Definition of datatype CanTrcv_PNActivationType ⌈

Name	CanTrcv_PNActivationType		
Kind	Enumeration		
Range	PN_ENABLED	–	PN wakeup functionality in CanTrcv is enabled.
	PN_DISABLED	–	PN wakeup functionality in CanTrcv is disabled.
Description	Datatype used for describing whether PN wakeup functionality in CanTrcv is enabled or disabled.		
Available via	CanTrcv.h		

⌋()

[SWS_CanTrcv_00211] Definition of datatype CanTrcv_TrcvFlagStateType ⌈

Name	CanTrcv_TrcvFlagStateType		
Kind	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.		
Available via	CanTrcv.h		

⌋()

[SWS_CanTrcv_00163] Definition of datatype CanTrcv_TrcvModeType ⌈

Name	CanTrcv_TrcvModeType		
Kind	Enumeration		
Range	CANTRCV_TRCVMODE_ SLEEP	–	Transceiver mode SLEEP
	CANTRCV_TRCVMODE_ STANDBY	–	Transceiver mode STANDBY
	CANTRCV_TRCVMODE_ NORMAL	0x00	Transceiver mode NORMAL
Description	Operating modes of the CAN Transceiver Driver.		
Available via	Can_GeneralTypes.h		

⌋()

[SWS_CanTrcv_00164] Definition of datatype CanTrcv_TrcvWakeupModeType ⌈

Name	CanTrcv_TrcvWakeupModeType		
Kind	Enumeration		
Range	CANTRCV_WUMODE_ ENABLE	0x00	The notification for wakeup events is enabled on the addressed transceiver.
	CANTRCV_WUMODE_ DISABLE	0x01	The notification for wakeup events is disabled on the addressed transceiver.
	CANTRCV_WUMODE_ CLEAR	0x02	A stored wakeup event is cleared on the addressed transceiver.
Description	This type shall be used to control the CAN transceiver concerning wake up events and wake up notifications.		
Available via	Can_GeneralTypes.h		

⌋()

[SWS_CanTrcv_00165] Definition of datatype CanTrcv_TrcvWakeupReasonType ⌈

Name	CanTrcv_TrcvWakeupReasonType		
Kind	Enumeration		
Range	CANTRCV_WU_ERROR	0x00	Due to an error wake up reason was not detected. This value may only be reported when error was reported to DEM before.
	CANTRCV_WU_NOT_ SUPPORTED	0x01	The transceiver does not support any information for the wake up reason.
	CANTRCV_WU_BY_BUS	0x02	The transceiver has detected, that the network has caused the wake up of the ECU.
	CANTRCV_WU_ INTERNALLY	0x03	The transceiver has detected, that the network has woken up by the ECU via a request to NORMAL mode.
	CANTRCV_WU_RESET	0x04	The transceiver has detected, that the "wake up" is due to an ECU reset.
	CANTRCV_WU_POWER_ ON	0x05	The transceiver has detected, that the "wake up" is due to an ECU reset after power on.
	CANTRCV_WU_BY_PIN	0x06	The transceiver has detected a wake-up event at one of the transceiver's pins (not at the CAN bus).
	CANTRCV_WU_BY_ SYSERR	0x07	The transceiver has detected, that the wake up of the ECU was caused by a HW related device failure.
Description	This type denotes the wake up reason detected by the CAN transceiver in detail.		
Available via	Can_GeneralTypes.h		

⌋()

8.3 Function definitions

8.3.1 CanTrcv_Init

[SWS_CanTrcv_00001] Definition of API function CanTrcv_Init ⌈

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 (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Initializes the CanTrcv module.	
Available via	CanTrcv.h	

⌋(*SRS_BSW_00310*, *SRS_BSW_00358*, *SRS_BSW_00369*, *SRS_BSW_00414*, *SRS_BSW_00101*, *SRS_BSW_00172*, *SRS_Can_01096*, *SRS_Can_01097*, *SRS_Can_01109*, *SRS_Can_01110*, *SRS_Can_01108*)

[SWS_CanTrcv_00180] ⌈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.⌋()

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

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

This API shall store the wake up event, if any, during initialization time.

See also requirement [SWS_CanTrcv_00099].

[SWS_CanTrcv_00167] ⌈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` with the wakeup source referenced in `CanTrcvWakeupSourceRef`.⌋()

[SWS_CanTrcv_00181] ⌈If selective wakeup is enabled and supported by hardware: POR and SYSERR flags of the transceiver status shall be checked by `CanTrcv_Init` API.⌋()

[SWS_CanTrcv_00182] ⌈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.⌋()

[SWS_CanTrcv_00183] ⌈If the POR flag is set, wakeup shall be reported to EcuM through API `EcuM_SetWakeupEvent` with a wakeup source value, which has a "1" at the bit position according to the symbolic name value referred by `CanTrcvPorWakeupSourceRef`, and "0" on all others.⌋()

[SWS_CanTrcv_00184] ⌈If the SYSERR flag is set, wakeup shall be reported to EcuM through API `EcuM_SetWakeupEvent` with a wakeup source value, which has a "1" at the bit position according to the symbolic name value referred by `CanTrcvSyserrWakeupSourceRef`, and "0" on all others.⌋()

[SWS_CanTrcv_00113] ⌈If there is no/incorrect communication towards the transceiver, the function `CanTrcv_Init` shall report the runtime error code `CANTRCV_E_NO_TRCV_CONTROL` to the Default Error Tracer.

For Eg., there are different transceiver types and different access ways (port connection, SPI). This runtime error should be signalled if you detect any miscommunication

with your hardware. Depending on connection type and depending on your transceiver hardware you may not run in situations where you have to signal this error.⌋*()*

[SWS_CanTrcv_00168] ⌈If development error detection is enabled for CanTrcv module: the function `CanTrcv_Init` shall raise the development error `CANTRCV_E_BAUDRATE_NOT_SUPPORTED`, if the configured baud rate is not supported by the transceiver.⌋*()*

[SWS_CanTrcv_00226] ⌈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).⌋*()*

8.3.2 CanTrcv_SetOpMode

[SWS_CanTrcv_00002] Definition of API function CanTrcv_SetOpMode ⌈

Service Name	CanTrcv_SetOpMode	
Syntax	`Std_ReturnType CanTrcv_SetOpMode ( uint8 Transceiver, CanTrcv_TrcvModeType OpMode )`	
Service ID [hex]	0x01	
Sync/Async	Asynchronous	
Reentrancy	Reentrant for different transceivers	
Parameters (in)	Transceiver	CAN transceiver to which API call has to be applied.
	OpMode	This parameter contains the desired operating mode
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: will be returned if the request for transceiver mode change has been accepted. `E_NOT_OK`: will be returned if the request for transceiver mode change has not been accepted or any parameter is out of the allowed range.
Description	Sets the mode of the Transceiver to the value OpMode.	
Available via	CanTrcv.h	

⌋*(SRS_BSW_00310, SRS_BSW_00357, SRS_BSW_00369, SRS_BSW_00406, SRS_Can_01091, SRS_Can_01097, SRS_Can_01098, SRS_Can_01099, SRS_Can_01100, SRS_Can_01109, SRS_Can_01110, SRS_Can_01108)*

[SWS_CanTrcv_00102] ⌈The function `CanTrcv_SetOpMode` shall switch the internal state of `Transceiver` to the value of the parameter OpMode, which can be CANTRCV_TRCVMODE_NORMAL, CANTRCV_TRCVMODE_STANDBY or CANTRCV_TRCVMODE_SLEEP.⌋*()*

Note: The user of the CanTrcv module may call the function `CanTrcv_SetOpMode` with `OpMode` = CANTRCV_TRCVMODE_STANDBY or CANTRCV_TRCVMODE_NORMAL, if the Transceiver is in mode CANTRCV_TRCVMODE_NORMAL.

Note: The user of the CanTrcv module may call the function CanTrcv_SetOpMode with OpMode = CANTRCV_TRCVMODE_SLEEP, CANTRCV_TRCVMODE_STANDBY or CANTRCV_TRCVMODE_NORMAL, if the Transceiver is in mode CANTRCV_TRCVMODE_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.

[SWS_CanTrcv_00105] ⌈If the requested mode is not supported by the underlying transceiver hardware, the function CanTrcv_SetOpMode shall return E_NOT_OK.⌋*()*

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

[SWS_CanTrcv_00186] ⌈If selective wakeup is supported by hardware: the flags POR and SYSERR of the transceiver status shall be checked by CanTrcv_SetOpMode API.⌋*()*

[SWS_CanTrcv_00187] ⌈If the POR flag is set, transceiver shall be re-initialized to run the transceiver's configuration sequence.⌋*()*

[SWS_CanTrcv_00188] ⌈If the SYSERR flag is NOT set and the requested mode is CANTRCV_NORMAL, transceiver shall call the API CanIf_ConfirmPnAvailability for the corresponding abstract CanIf TransceiverId. CanIf_ConfirmPnAvailability informs CanNm (through CanIf and CanSm) that selective wakeup is enabled.⌋*()*

[SWS_CanTrcv_00114] ⌈If there is no/incorrect communication to the Transceiver, the function CanTrcv_SetOpMode shall report runtime error code CANTRCV_E_NO_TRCV_CONTROL to the Default Error Tracer and return E_NOT_OK.⌋*()*

[SWS_CanTrcv_00120] ⌈If development error detection for the module CanTrcv is enabled:

If the function CanTrcv_SetOpMode is called with OpMode = CANTRCV_TRCVMODE_STANDBY, and the Transceiver is not in mode CANTRCV_TRCVMODE_NORMAL or CANTRCV_TRCVMODE_STANDBY, the function CanTrcv_SetOpMode shall raise the development error CANTRCV_E_TRCV_NOT_NORMAL otherwise (if DET is disabled) return E_NOT_OK.⌋*()*

[SWS_CanTrcv_00121] ⌈If development error detection for the module CanTrcv is enabled:

If the function CanTrcv_SetOpMode is called with OpMode = CANTRCV_TRCVMODE_SLEEP, and the Transceiver is not in mode CANTRCV_TRCVMODE_STANDBY or CANTRCV_TRCVMODE_SLEEP, the function CanTrcv_SetOpMode shall raise the development error CANTRCV_E_TRCV_NOT_STANDBY otherwise (if DET is disabled) return E_NOT_OK.⌋*()*

[SWS_CanTrcv_00122] ⌈If development error detection for the module CanTrcv is enabled:

If called before the CanTrcv module has been initialized, the function CanTrcv_SetOpMode shall raise the development error CANTRCV_E_UNINIT otherwise (if DET is disabled) return E_NOT_OK.⌋()

[SWS_CanTrcv_00123] ⌈If development error detection for the module CanTrcv is enabled: If called with an invalid Transceiver number, the function CanTrcv_SetOpMode shall raise the development error CANTRCV_E_INVALID_TRANSCEIVER otherwise (if DET is disabled) return E_NOT_OK.⌋()

[SWS_CanTrcv_00087] ⌈If development error detection 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_OPMODE otherwise (if DET is disabled) return E_NOT_OK.⌋()

8.3.3 CanTrcv_GetOpMode

[SWS_CanTrcv_00005] Definition of API function CanTrcv_GetOpMode ⌈

Service Name	CanTrcv_GetOpMode	
Syntax	`Std_ReturnType CanTrcv_GetOpMode ( uint8 Transceiver, CanTrcv_TrcvModeType* OpMode )`	
Service ID [hex]	0x02	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Transceiver	CAN transceiver to which API call has to be applied.
Parameters (inout)	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.	
Available via	CanTrcv.h	

⌋(*SRS_BSW_00310*, *SRS_BSW_00369*, *SRS_BSW_00377*, *SRS_BSW_00406*, *SRS_Can_01091*, *SRS_Can_01097*, *SRS_Can_01101*, *SRS_Can_01109*, *SRS_Can_01110*)

[SWS_CanTrcv_00106] ⌈The function CanTrcv_GetOpMode shall collect the actual state of the CAN transceiver driver in the out parameter OpMode.⌋()

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

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

[SWS_CanTrcv_00115] ⌈If there is no/incorrect communication to the transceiver, the function CanTrcv_GetOpMode shall report the runtime error code CANTRCV_E_NO_TRCV_CONTROL to the Default Error Tracer and return E_NOT_OK.⌋()

[SWS_CanTrcv_00124] ⌈If development error detection for the module CanTrcv is enabled: If called before the CanTrcv module has been initialized, the function `CanTrcv_GetOpMode` shall raise the development error `CANTRCV_E_UNINIT` otherwise (if DET is disabled) return E_NOT_OK.⌋*()*

[SWS_CanTrcv_00129] ⌈If development error detection for the module CanTrcv is enabled: If called with an invalid `Transceiver` number, the function `CanTrcv_GetOpMode` shall raise the development error `CANTRCV_E_INVALID_TRANSCEIVER` otherwise (if DET is disabled) return E_NOT_OK.⌋*()*

[SWS_CanTrcv_00132] ⌈If development error detection for the module CanTrcv is enabled: If called with `OpMode` = NULL, the function `CanTrcv_GetOpMode` shall raise the development error `CANTRCV_E_PARAM_POINTER` otherwise (if DET is disabled) return E_NOT_OK.⌋*()*

8.3.4 CanTrcv_GetBusWuReason

[SWS_CanTrcv_00007] Definition of API function CanTrcv_GetBusWuReason ⌈

Service Name	CanTrcv_GetBusWuReason	
Syntax	`Std_ReturnType CanTrcv_GetBusWuReason ( uint8 Transceiver, CanTrcv_TrcvWakeupReasonType* reason )`	
Service ID [hex]	0x03	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Transceiver	CAN transceiver to which API call has to be applied.
Parameters (inout)	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 transceiver wakeup reason was provided. `E_NOT_OK`: will be returned if no wake up reason is available or if the service request failed due to development errors.
Description	Gets the wakeup reason for the Transceiver and returns it in parameter Reason.	
Available via	CanTrcv.h	

⌋*(SRS_BSW_00310, SRS_BSW_00369, SRS_BSW_00375, SRS_BSW_00377, SRS_BSW_00406, SRS_Can_01091, SRS_Can_01095, SRS_Can_01097, SRS_Can_01103, SRS_Can_01106, SRS_Can_01109, SRS_Can_01110)*

[SWS_CanTrcv_00107] ⌈The function `CanTrcv_GetBusWuReason` shall collect the reason for the wake up that the CAN transceiver has detected in the parameter Reason.⌋*()*

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

Be aware if more than one bus is available, each bus may report a different wake up reason. E.g. if an ECU has CAN, a wake up by CAN may occur and the incoming data may cause an internal wake up for another CAN bus.

The CAN transceiver driver has a "per bus" view and does not vote the more important reason or sequence internally. The same may be true if e.g. one transceiver controls the power supply and the other is just powered or un-powered.

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

[SWS_CanTrcv_00116] ⌈If there is no/incorrect communication to the transceiver, the function `CanTrcv_GetBusWuReason` shall report the runtime error code `CANTRCV_E_NO_TRCV_CONTROL` to the Default Error Tracer and return E_NOT_OK.⌋*()*

[SWS_CanTrcv_00125] ⌈If development error detection for the module CanTrcv is enabled: If called before the CanTrcv module has been initialized, the function `CanTrcv_GetBusWuReason` shall raise development error `CANTRCV_E_UNINIT` otherwise (if DET is disabled) return E_NOT_OK.⌋*()*

[SWS_CanTrcv_00130] ⌈If development error detection for the module CanTrcv is enabled: If called with an invalid `Transceiver` number, the function `CanTrcv_GetBusWuReason` shall raise development error `CANTRCV_E_INVALID_TRANSCEIVER` otherwise (if DET is disabled) return E_NOT_OK.⌋*()*

[SWS_CanTrcv_00133] ⌈If development error detection for the module CanTrcv is enabled: If called with `reason` = NULL, the function `CanTrcv_GetBusWuReason` shall raise the development error `CANTRCV_E_PARAM_POINTER` otherwise (if DET is disabled) return E_NOT_OK.⌋*()*

8.3.5 CanTrcv_VersionInfo

[SWS_CanTrcv_00008] Definition of API function CanTrcv_GetVersionInfo ⌈

Service Name	CanTrcv_GetVersionInfo	
Syntax	`void CanTrcv_GetVersionInfo ( Std_VersionInfoType* versioninfo )`	
Service ID [hex]	0x04	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	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.	
Available via	CanTrcv.h	

⌋*(SRS_BSW_00310, SRS_BSW_00369, SRS_BSW_00406, SRS_BSW_00407, SRS_BSW_00411)*

8.3.6 CanTrcv_ SetWakeupMode

[SWS_CanTrcv_00009] Definition of API function CanTrcv_SetWakeupMode ⌈

Service Name	CanTrcv_SetWakeupMode	
Syntax	`Std_ReturnType CanTrcv_SetWakeupMode ( uint8 Transceiver, CanTrcv_TrcvWakeupModeType TrcvWakeupMode )`	
Service ID [hex]	0x05	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different transceivers	
Parameters (in)	Transceiver	CAN transceiver to which API call has to be applied.
	TrcvWakeupMode	Requested transceiver wakeup reason
Parameters (inout)	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.	
Available via	CanTrcv.h	

⌋*(SRS_BSW_00310, SRS_BSW_00369, SRS_BSW_00406, SRS_Can_01091, SRS_Can_01097, SRS_Can_01109, SRS_Can_01110, SRS_Can_01115)*

[SWS_CanTrcv_00111] ⌈Enabled: If the function `CanTrcv_SetWakeupMode` is called with `TrcvWakeupMode` = CANTRCV_ WUMODE_ENABLE and if the CanTrcv module has a stored wakeup event pending for the addressed bus, the CanTrcv module shall update its wakeup event as 'present'.⌋*()*

[SWS_CanTrcv_00093] ⌈Disabled: If the function `CanTrcv_SetWakeupMode` is called with `TrcvWakeupMode` = CANTRCV_ WUMODE_DISABLE, the wakeup events are disabled on the addressed transceiver. It is required by the transceiver device and the transceiver driver to detect the wakeup events and store it internally, in order to raise the wakeup events when the wakeup mode is enabled again.⌋*(SRS_BSW_00388, SRS_BSW_00389, SRS_BSW_00390, SRS_BSW_00392, SRS_BSW_00393, SRS_BSW_00394, SRS_BSW_00395, SRS_BSW_00408, SRS_BSW_00160, SRS_Can_01090)*

[SWS_CanTrcv_00094] ⌈Clear: If the function `CanTrcv_SetWakeupMode` is called with `TrcvWakeupMode` = CANTRCV_ WUMODE_CLEAR, then a stored wakeup event is cleared on the addressed `Transceiver`.⌋*()*

[SWS_CanTrcv_00150] ⌈Clearing of wakeup events have to be used when the wake up notification is disabled to clear all stored wake up events under control of the higher layer.⌋*()*

[SWS_CanTrcv_00095] ⌈The implementation can enable, disable or clear wake up events from the last communication cycle. It is very important not to lose wake up events during the disabled period.⌋*(SRS_BSW_00388, SRS_BSW_00389,*

SRS_BSW_00390, *SRS_BSW_00392*, *SRS_BSW_00393*, *SRS_BSW_00394*, *SRS_BSW_00395*, *SRS_BSW_00408*, *SRS_BSW_00160*, *SRS_Can_01090*)

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

[SWS_CanTrcv_00117] ⌈If there is no/incorrect communication to the transceiver, the function `CanTrcv_SetWakeupMode` shall report the runtime error code `CANTRCV_E_NO_TRCV_CONTROL` to the Default Error Tracer and return E_NOT_OK.⌋*()*

[SWS_CanTrcv_00127] ⌈If development error detection for the module CanTrcv is enabled: If called before the CanTrcv has been initialized, the function `CanTrcv_SetWakeupMode` shall raise development error `CANTRCV_E_UNINIT` otherwise (if DET is disabled) return E_NOT_OK.⌋*()*

[SWS_CanTrcv_00131] ⌈If development error detection for the module CanTrcv is enabled: If called with an invalid `Transceiver` number, the function `CanTrcv_SetWakeupMode` shall raise development error `CANTRCV_E_INVALID_TRANSCEIVER` otherwise (if DET is disabled) return E_NOT_OK.⌋*()*

[SWS_CanTrcv_00089] ⌈If development error detection for the module CanTrcv is enabled: If called with an invalid `TrcvWakeupMode`, the function `CanTrcv_SetWakeupMode` shall raise the development error `CANTRCV_E_PARAM_TRCV_WAKEUP_MODE`

POINTER otherwise (if DET is disabled) return E_NOT_OK.⌋*()*

8.3.7 CanTrcv_GetTrcvSystemData

[SWS_CanTrcv_00213] Definition of API function CanTrcv_GetTrcvSystemData ⌈

Service Name	CanTrcv_GetTrcvSystemData	
Syntax	`Std_ReturnType CanTrcv_GetTrcvSystemData ( uint8 Transceiver, uint32* TrcvSysData )`	
Service ID [hex]	0x09	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	Transceiver	CAN transceiver ID.
Parameters (inout)	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. This API shall exist only if CanTrcvHwPnSupport = TRUE.	
Available via	CanTrcv.h	

⌋*()*

[SWS_CanTrcv_00189] ⌈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.⌋*()*

[SWS_CanTrcv_00190] ⌈If there is no/incorrect communication to the transceiver, the function CanTrcv_GetTrcvSystemData shall report the runtime error code CANTRCV_E_NO_TRCV_CONTROL to the default Error Tracer and return E_NOT_OK.⌋ *()*

[SWS_CanTrcv_00191] ⌈If development error detection 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 otherwise (if DET is disabled) return E_NOT_OK.⌋*()*

[SWS_CanTrcv_00192] ⌈If development error detection is enabled for the CanTrcv module: if called with an invalid transceiver ID for parameter Transceiver, function CanTrcv_GetTrcvSystemData shall raise the development error CANTRCV_E_INVALID_TRANSCEIVER otherwise (if DET is disabled) return E_NOT_OK.⌋*()*

[SWS_CanTrcv_00193] ⌈If development error detection 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 otherwise (if DET is disabled) return E_NOT_OK.⌋*()*

8.3.8 CanTrcv_ClearTrcvWufFlag

[SWS_CanTrcv_00214] Definition of API function CanTrcv_ClearTrcvWufFlag ⌈

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 (inout)	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. This API shall exist only if CanTrcvHwPn Support = TRUE.	
Available via	CanTrcv.h	

⌋(*SRS_Can_01157*)

[SWS_CanTrcv_00194] ⌈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.⌋()

[SWS_CanTrcv_00195] ⌈CanTrcv shall inform CanIf that the wakeup flag has been cleared for the requested Transceiver, through the callback notification CanIf_ClearTrcvWufFlagIndication referring to the corresponding CAN transceiver with the abstract CanIf TransceiverId.⌋()

[SWS_CanTrcv_00196] ⌈If there is no/incorrect communication to the transceiver, the function CanTrcv_ClearTrcvWufFlag shall report the runtime error CANTRCV_E_NO_TRCV_CONTROL to the Default Error Tracer and return E_NOT_OK.⌋()

[SWS_CanTrcv_00197] ⌈If development error detection 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 otherwise (if DET is disabled) return E_NOT_OK.⌋()

[SWS_CanTrcv_00198] ⌈If development error detection is enabled for the CanTrcv module: if called with an invalid transceiver ID for parameter Transceiver, function CanTrcv_ClearTrcvWufFlag shall raise the development error CANTRCV_E_INVALID_TRANSCEIVER otherwise (if DET is disabled) return E_NOT_OK.⌋()

8.3.9 CanTrcv_ReadTrcvTimeoutFlag

[SWS_CanTrcv_00215] Definition of API function CanTrcv_ReadTrcvTimeoutFlag ⌈

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 (inout)	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 success-fully 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. This API shall exist only if CanTrcvHwPnSupport = TRUE.	
Available via	CanTrcv.h	

⌋()

[SWS_CanTrcv_00199] ⌈If development error detection is enabled for the module CanTrcv: If called with an invalid transceiver ID `Transceiver`, the function `CanTrcv_ReadTrcvTimeoutFlag` shall raise the development error `CANTRCV_E_INVALID_TRANSCEIVER` otherwise (if DET is disabled) return E_NOT_OK.⌋()

[SWS_CanTrcv_00200] ⌈If development error detection is enabled for the module CanTrcv: If called with `FlagState` = NULL, the function `CanTrcv_ReadTrcvTimeoutFlag` shall raise the development error `CANTRCV_E_PARAM_POINTER` otherwise (if DET is disabled) return E_NOT_OK.⌋()

8.3.10 CanTrcv_ClearTrcvTimeoutFlag

[SWS_CanTrcv_00216] Definition of API function CanTrcv_ClearTrcvTimeoutFlag ⌈

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 (inout)	None	
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. This API shall exist only if Can TrcvHwPnSupport = TRUE.	
Available via	CanTrcv.h	

⌋()

[SWS_CanTrcv_00201] ⌈If development error detection is enabled for the module CanTrcv: If called with an invalid transceiver ID Transceiver, the function CanTrcv_ClearTrcvTimeoutFlag shall raise the development error CANTRCV_E_INVALID_TRANSCEIVER otherwise (if DET is disabled) return E_NOT_OK.⌋()

8.3.11 CanTrcv_ReadTrcvSilenceFlag

[SWS_CanTrcv_00217] Definition of API function CanTrcv_ReadTrcvSilenceFlag ⌈

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 (inout)	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 success-fully 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. This API shall exist only if CanTrcvHwPnSupport = TRUE.	
Available via	CanTrcv.h	

⌋()

[SWS_CanTrcv_00202] ⌈If development error detection is enabled for the module CanTrcv: If called with an invalid transceiver ID Transceiver, the function CanTrcv_ReadTrcvSilenceFlag shall raise the development error CANTRCV_E_INVALID_TRANSCEIVER otherwise (if DET is disabled) return E_NOT_OK.⌋()

[SWS_CanTrcv_00203] ⌈If development error detection is enabled for the module CanTrcv: If called with FlagState = NULL, the function CanTrcv_ReadTrcvSi-

lenceFlag shall raise the development error CANTRCV_E_PARAM_POINTER otherwise (if DET is disabled) return E_NOT_OK.⌋()

8.3.12 CanTrcv_CheckWakeup

[SWS_CanTrcv_00143] Definition of API function CanTrcv_CheckWakeup ⌈

Service Name	CanTrcv_CheckWakeup	
Syntax	`Std_ReturnType CanTrcv_CheckWakeup ( uint8 Transceiver )`	
Service ID [hex]	0x07	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Transceiver	CAN transceiver to which API call has to be applied.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: API call has been accepted E_NOT_OK: API call has not been accepted
Description	Service is called by underlying CANIF in case a wake up interrupt is detected.	
Available via	CanTrcv.h	

⌋()

[SWS_CanTrcv_00144] ⌈If development error detection for the module CanTrcv is enabled: If called before the CanTrcv module has been initialized, the function CanTrcv_CheckWakeup shall raise the development error CANTRCV_E_UNINIT otherwise (if DET is disabled) return E_NOT_OK.⌋()

[SWS_CanTrcv_00145] ⌈If development error detection for the module CanTrcv is enabled: If called with an invalid Transceiver number, the function CanTrcv_CheckWakeup shall raise the development error CANTRCV_E_INVALID_TRANSCEIVER otherwise (if DET is disabled) return E_NOT_OK.⌋()

[SWS_CanTrcv_00146] ⌈If supported by hardware, CanTrcv_CheckWakeup 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 with the wakeup source referenced in CanTrcvWakeupSourceRef.⌋()

8.3.13 CanTrcv_SetPNActivationState

[SWS_CanTrcv_00219] Definition of API function CanTrcv_SetPNActivationState ⌈

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	PN_ENABLED: PN wakeup functionality in CanTrcv shall be enabled. PN_DIABLED: PN wakeup functionality in CanTrcv shall be disabled.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: Will be returned, if the PN has been changed to the requested configuration. `E_NOT_OK`: Will be returned, if the PN configuration change has failed. The previous configuration has not been changed.
Description	The API configures the wake-up 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.	
Available via	CanTrcv.h	

⌋()

[SWS_CanTrcv_00220] ⌈If development error detection for the module CanTrcv is enabled: If called before the CanTrcv module has been initialized, the function `CanTrcv_SetPNActivationState` shall raise the development error `CANTRCV_E_UNINIT` otherwise (if DET is disabled) return E_NOT_OK.⌋()

[SWS_CanTrcv_00221] ⌈CanTrcv shall enable the PN wakeup functionality when function `CanTrcv_SetPNActivationState` is called with `ActivationState`= PN_ENABLED and return E_OK.⌋()

[SWS_CanTrcv_00222] ⌈CanTrcv shall disable the PN wakeup functionality when function `CanTrcv_SetPNActivationState` is called with `ActivationState`= PN_DISABLED and return E_OK⌋()

8.3.14 CanTrcv_CheckWakeFlag

[SWS_CanTrcv_00223] Definition of API function CanTrcv_CheckWakeFlag ⌈

Service Name	CanTrcv_CheckWakeFlag
Syntax	`Std_ReturnType CanTrcv_CheckWakeFlag ( uint8 Transceiver )`
Service ID [hex]	0x0e

▽

△

Sync/Async	Asynchronous	
Reentrancy	Non Reentrant	
Parameters (in)	Transceiver	CAN transceiver ID.
Parameters (inout)	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.	
Available via	CanTrcv.h	

⌋()

[SWS_CanTrcv_00224] ⌈CanTrcv shall inform the CanIf with the callback notification `CanIf_CheckTrcvWakeFlagIndication`, that the wake flag of the CAN Transceiver with the corresponding TransceiverId has been checked.⌋()

[SWS_CanTrcv_00225] ⌈If development error detection is enabled for the module CanTrcv: If called with an invalid transceiver ID `Transceiver`, the function `CanTrcv_CheckWakeFlag` shall raise the development error `CANTRCV_E_INVALID_TRANSCEIVER` otherwise (if DET is disabled) return E_NOT_OK.⌋()

8.3.15 CanTrcv_DeInit

[SWS_CanTrcv_91001] Definition of API function CanTrcv_DeInit ⌈

Service Name	CanTrcv_DeInit

Syntax	`void CanTrcv_DeInit (`
	`  void`
	`)`

Service ID [hex]	0x10
Sync/Async	Synchronous
Reentrancy	Non Reentrant
Parameters (in)	None
Parameters (inout)	None
Parameters (out)	None
Return value	None
Description	De-initializes the CanTrcv module.
Available via	CanTrcv.h

⌋(*SRS_Can_01108*, *SRS_BSW_00336*)

[SWS_CanTrcv_91002] ⌈The function `CanTrcv_DeInit` shall de-initialize all the connected CAN transceivers based on their de-initialization sequences.⌋(*SRS_Can_01108*)

[SWS_CanTrcv_91003] ⌈The function `CanTrcv_DeInit` shall set the CAN transceiver hardware to the state NOT_ACTIVE.⌋(*SRS_Can_01108*)

In the state NOT_ACTIVE, the CAN transceiver hardware allows to be re-configured with a new configuration sequence

[SWS_CanTrcv_91004] ⌈If there is no/incorrect communication towards the transceiver, the function `CanTrcv_DeInit` shall report the runtime error `CANTRCV_E_NO_TRCV_CONTROL` code to the Default Error Tracer.

For Eg., there are different transceiver types and different access ways (port connection, SPI). This runtime error should be signaled if you detect any miscommunication with your hardware. Depending on connection type and depending on your transceiver hardware you may not run in situations where you have to signal this error.⌋*(SRS_BSW_00369)*

[SWS_CanTrcv_91005] ⌈If development error detection for the CanTrcv module is enabled: The function `CanTrcv_DeInit` shall raise the error `CANTRCV_E_TRCV_NOT_STANDBY` if the transceiver is not in mode CANTRCV_TRCVMODE_STANDBY or CANTRCV_TRCVMODE_SLEEP.⌋*(SRS_BSW_00369)*

8.4 Scheduled functions

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

8.4.1 CanTrcv_MainFunction

[SWS_CanTrcv_00013] Definition of scheduled function CanTrcv_MainFunction ⌈

Service Name	CanTrcv_MainFunction

Syntax	`void CanTrcv_MainFunction (`
	`  void`
	`)`

Service ID [hex]	0x06
Description	Service to scan all busses for wake up events and perform these event.
Available via	SchM_CanTrcv.h

⌋*(SRS_BSW_00310, SRS_BSW_00369, SRS_BSW_00373, SRS_BSW_00406, SRS_BSW_00424, SRS_BSW_00428, SRS_BSW_00171, SRS_BSW_00172, SRS_Can_01097, SRS_Can_01109, SRS_Can_01110)* The CAN bus transceiver driver may have cyclic jobs like polling for wake up events (if configured).

[SWS_CanTrcv_00112] ⌈The `CanTrcv_MainFunction` shall scan all busses in STANDBY and SLEEP for wake up events.

This function shall set a wake-up event flag to perform these events.⌋*(SRS_BSW_00343)*

According to [SRS_BSW_00424], main processing functions shall be allocated by basic tasks. No special call order to be kept. This function is directly called by Basic Software Scheduler.

See configuration parameter `CanTrcvWakeUpSupport`.

8.4.2 CanTrcv_MainFunctionDiagnostics

[SWS_CanTrcv_00218] Definition of scheduled function CanTrcv_MainFunction Diagnostics ⌈

Service Name	CanTrcv_MainFunctionDiagnostics
Syntax	`void CanTrcv_MainFunctionDiagnostics ( void )`
Service ID [hex]	0x08
Description	Reads the transceiver diagnostic status periodically and sets product/development accordingly.
Available via	SchM_CanTrcv.h

⌋*()*

[SWS_CanTrcv_00204] ⌈The cyclic function `CanTrcv_MainFunctionDiagnostics` shall read the transceiver status periodically and report production/development errors accordingly.⌋*()*

[SWS_CanTrcv_00205] ⌈The cyclic function `CanTrcv_MainFunctionDiagnostics` shall exist only if CanTrcvBusErrFlag = TRUE.⌋*()*

[SWS_CanTrcv_00206] ⌈If configured and supported by hardware: if the BUSERR flag reported from BSW is set, function `CanTrcv_MainFunctionDiagnostics` shall call the API `Dem_SetEventStatus` with parameters EventId as CANTRCV_E_BUS_ERROR and EventStatus as DEM_EVENT_STATUS_FAILED.⌋ *(SRS_BSW_00337, SRS_BSW_00385, SRS_BSW_00327, SRS_BSW_00331)*

[SWS_CanTrcv_00227] ⌈If configured and supported by hardware: if the BUSERR flag reported from BSW is reset, function `CanTrcv_MainFunctionDiagnostics` shall call the API `Dem_SetEventStatus` with parameters EventId as CANTRCV_E_BUS_ERROR and EventStatus as DEM_EVENT_STATUS_PASSED.⌋ *(SRS_BSW_00337, SRS_BSW_00385, SRS_BSW_00327, SRS_BSW_00331)*

8.5 Callback notifications

Since the CanTrcv is a driver module, it doesn't provide any callback functions for lower layer modules.

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.

[SWS_CanTrcv_00085] Definition of mandatory interfaces in module CanTrcv ⌈

API Function	*Header File*	*Description*
CanIf_TrcvModeIndication	CanIf_CanTrcv.h	This service indicates a transceiver state transition referring to the corresponding CAN transceiver with the abstract CanIf TransceiverId.
Det_ReportRuntimeError	Det.h	Service to report runtime errors. If a callout has been configured then this callout shall be called.

⌋()

8.6.2 Optional interfaces

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

[SWS_CanTrcv_00086] Definition of optional interfaces in module CanTrcv ⌈

API Function	*Header File*	*Description*
CanIf_CheckTrcvWakeFlagIndication	CanIf_CanTrcv.h	This service indicates that the check of the transceiver's wake-up flag has been finished by the corresponding CAN transceiver with the abstract CanIf TransceiverId. This indication is used to cope with the asynchronous transceiver communication.
CanIf_ClearTrcvWufFlagIndication	CanIf_CanTrcv.h	This service indicates that the transceiver has cleared the WufFlag referring to the corresponding CAN transceiver with the abstract CanIf Transceiver Id.
CanIf_ConfirmPnAvailability	CanIf_CanTrcv.h	This service indicates that the transceiver is running in PN communication mode referring to the corresponding CAN transceiver with the abstract CanIf TransceiverId.
Dem_SetEventStatus	Dem.h	Called by SW-Cs or BSW modules to report monitor status information to the Dem. BSW modules calling Dem_SetEventStatus can safely ignore the return value. This API will be available only if ({Dem/Dem ConfigSet/DemEventParameter/DemEvent ReportingType} == STANDARD_REPORTING)
Det_ReportError	Det.h	Service to report development errors.
Dio_ReadChannel	Dio.h	Returns the value of the specified DIO channel.
Dio_ReadChannelGroup	Dio.h	This Service reads a subset of the adjoining bits of a port.

▽

△

API Function	*Header File*	*Description*
Dio_ReadPort	Dio.h	Returns the level of all channels of that port.
Dio_WriteChannel	Dio.h	Service to set a level of a channel.
Dio_WriteChannelGroup	Dio.h	Service to set a subset of the adjoining bits of a port to a specified level.
Dio_WritePort	Dio.h	Service to set a value of the port.
EcuM_SetWakeupEvent	EcuM.h	Sets the wakeup event.
Icu_DisableNotification	Icu.h	This function disables the notification of a channel.
Icu_EnableNotification	Icu.h	This function enables the notification on the given channel.
Spi_GetStatus	Spi.h	Service returns the SPI Handler/Driver software module status.
Spi_ReadIB	Spi.h	Service for reading synchronously one or more data from an IB SPI Handler/Driver Channel specified by parameter.
Spi_SetupEB	Spi.h	Service to setup the buffers and the length of data for the EB SPI Handler/Driver Channel specified.
Spi_SyncTransmit	Spi.h	Service to transmit data on the SPI bus
Spi_WriteIB	Spi.h	Service for writing one or more data to an IB SPI Handler/Driver Channel specified by parameter.
Tm_BusyWait1us16bit	Tm.h	Performs busy waiting by polling with a guaranteed minimum waiting time.

⌋()

Check of the transceiver's wake-up flag has been finished by the corresponding CAN transceiver with the abstract CanIf TransceiverId. This indication is used to cope with the asynchronous transceiver communication.

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

Note: If the Can transceiver is controlled via Dio/Spi, the Dio/Spi interfaces are required to fulfill the core functionality of the module. Which interfaces are needed exactly shall not be detailed further in this specification

8.6.3 Configurable interfaces

There are no configurable interfaces for CAN transceiver driver.

9 Sequence diagrams

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

Depending on the transceiver hardware, there may be a need of wait states for some transitions.

9.1 Wake up with valid validation

For all wakeup related sequence diagrams please refer to chapter 9 of [6].

9.2 Interaction with DIO module

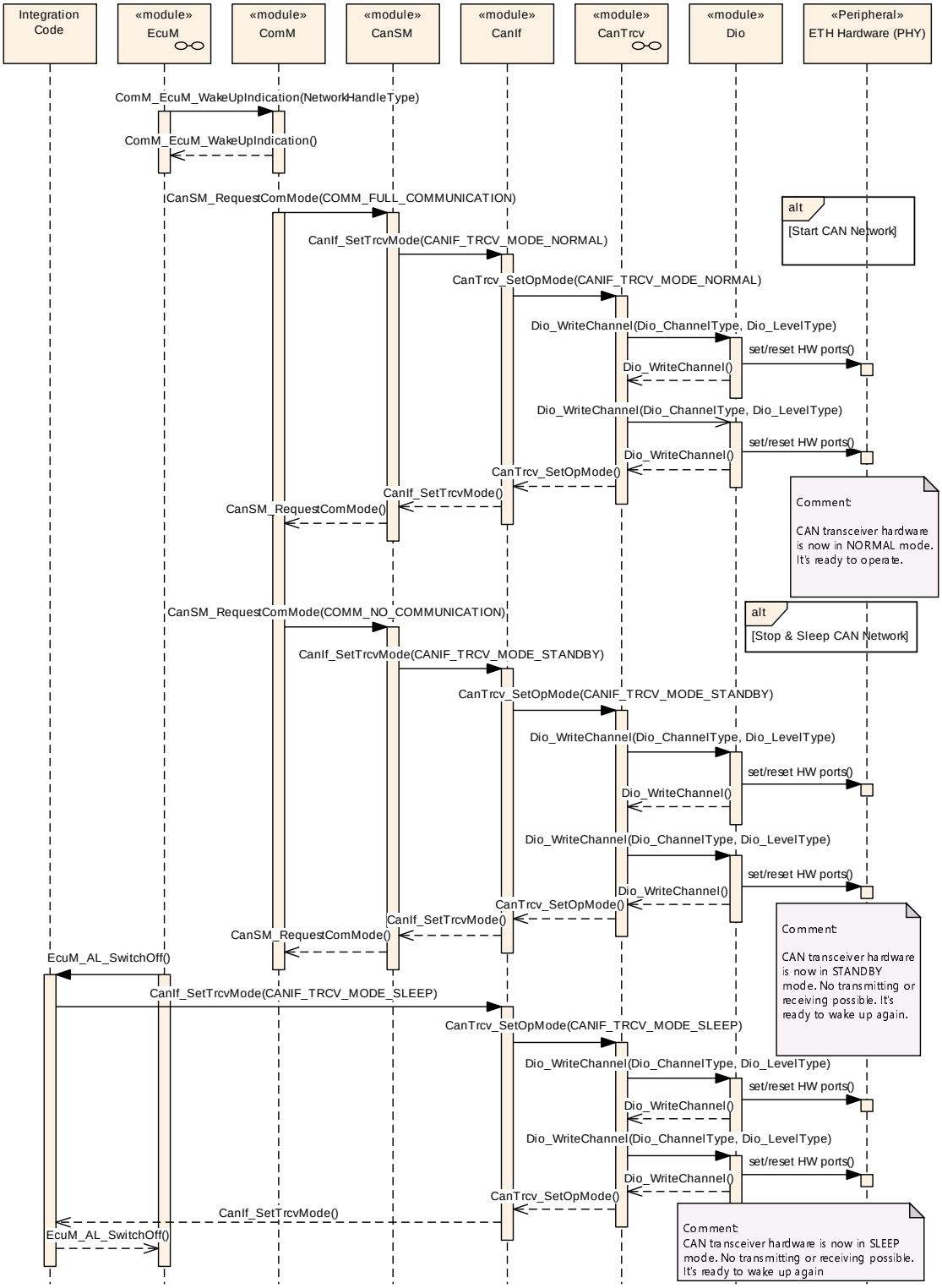

Figure 9.1: CanTrcv Dio Interaction

9.3 De-Initialization (SPI Synchronous)

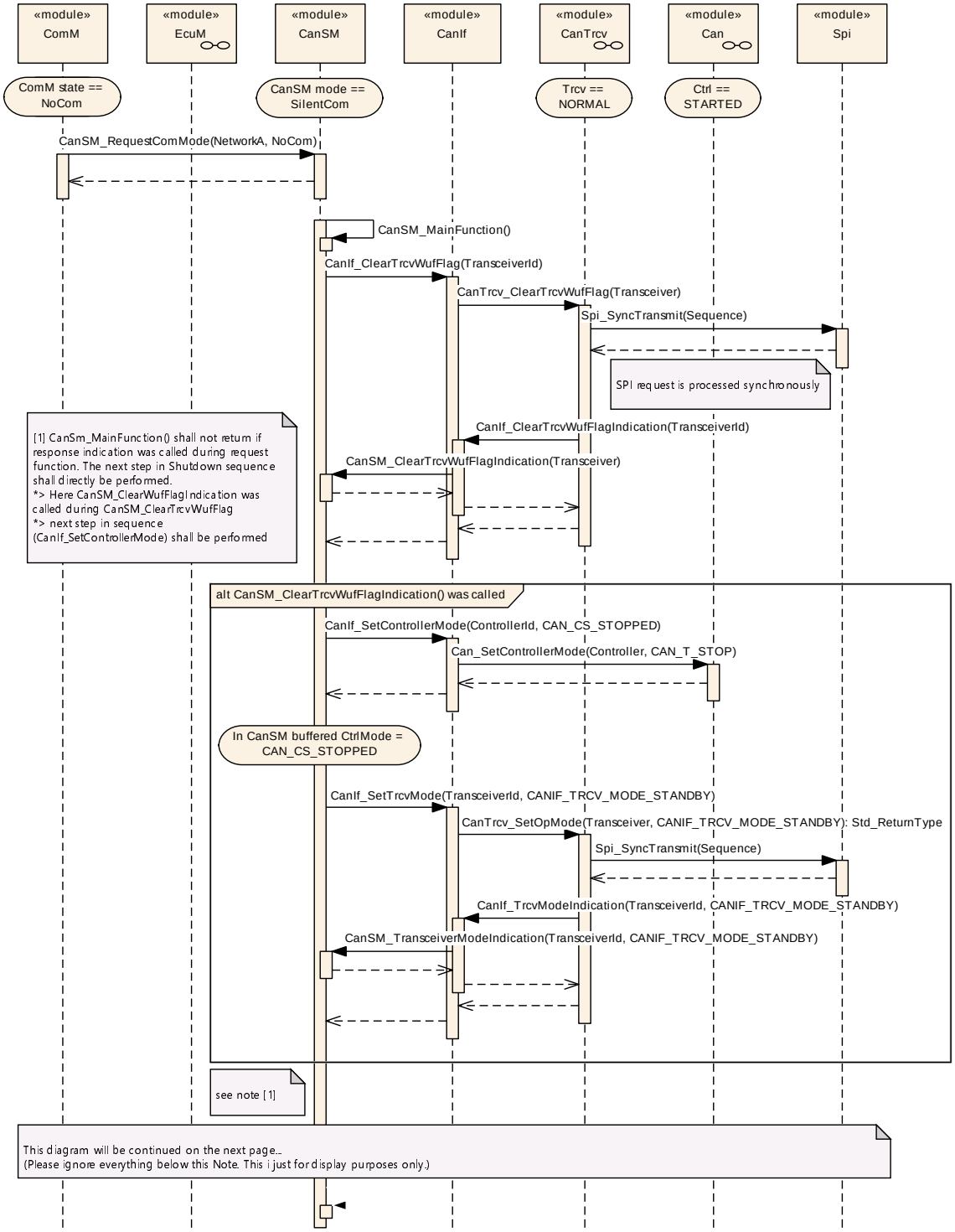

Figure 9.2: CanTrcv Deinit SPI synchronous I

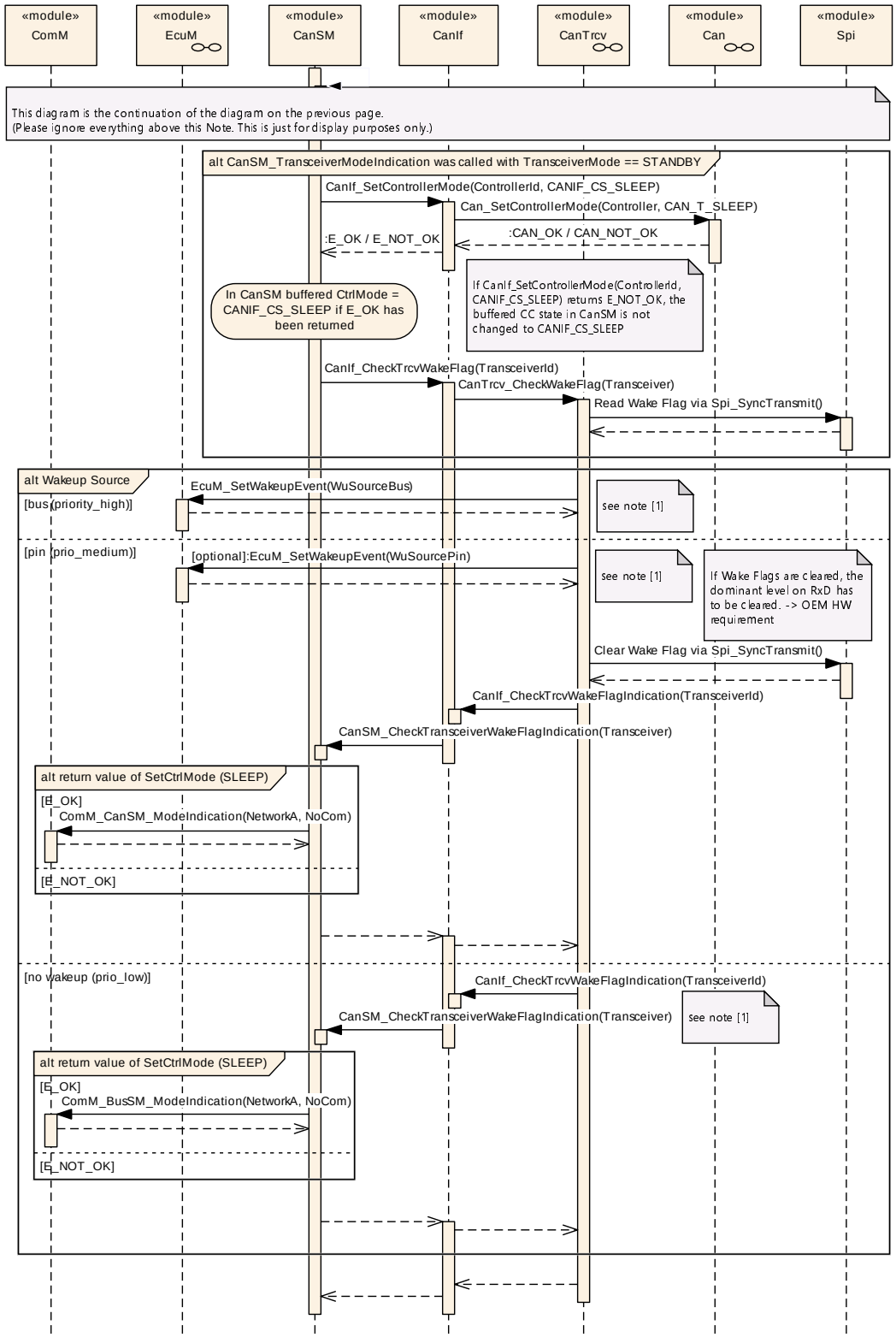

Figure 9.3: CanTrcv Deinit SPI synchronous II

9.4 De-Initialization (SPI Asynchronous)

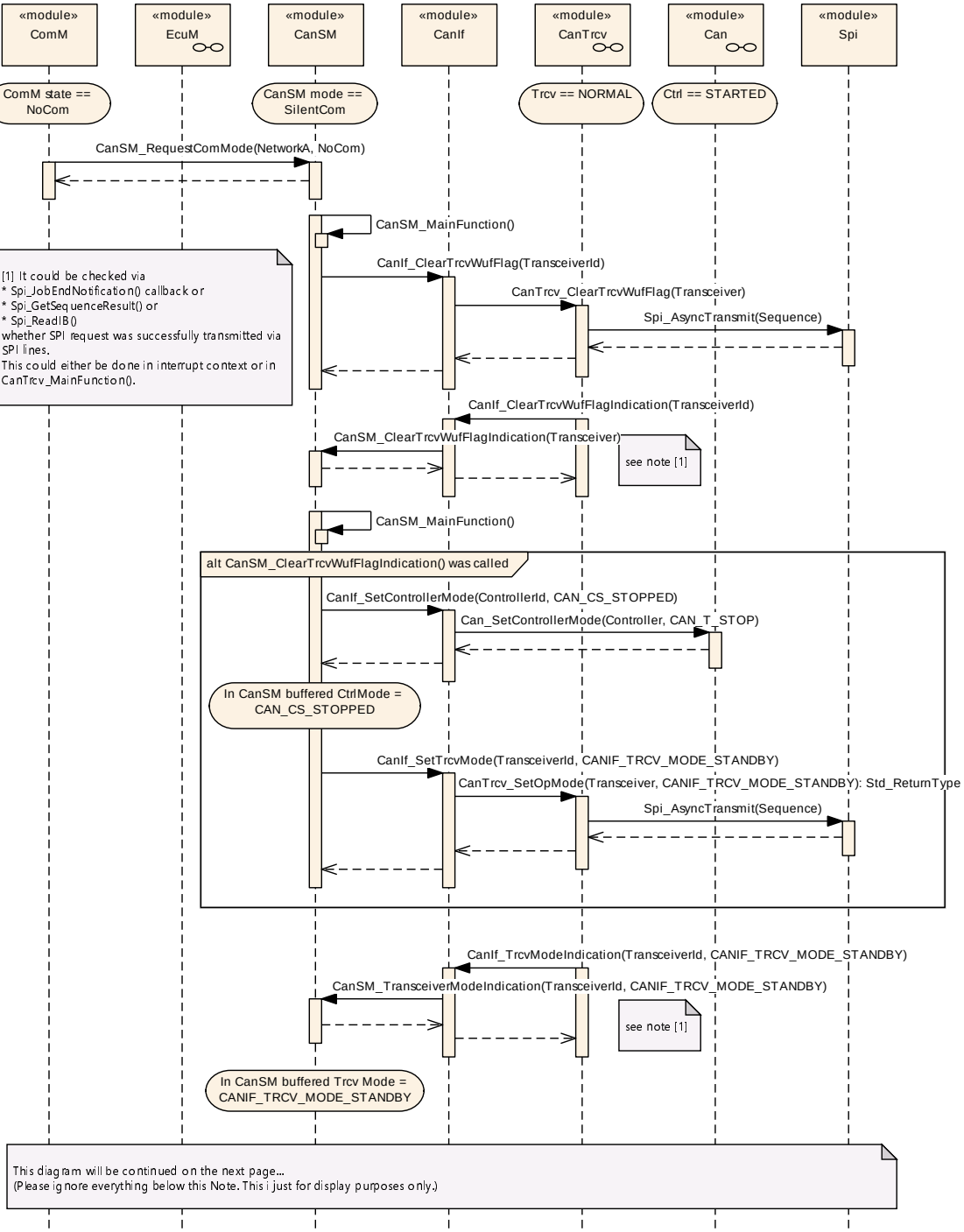

Figure 9.4: CanTrcv Deinit SPI asynchronous I

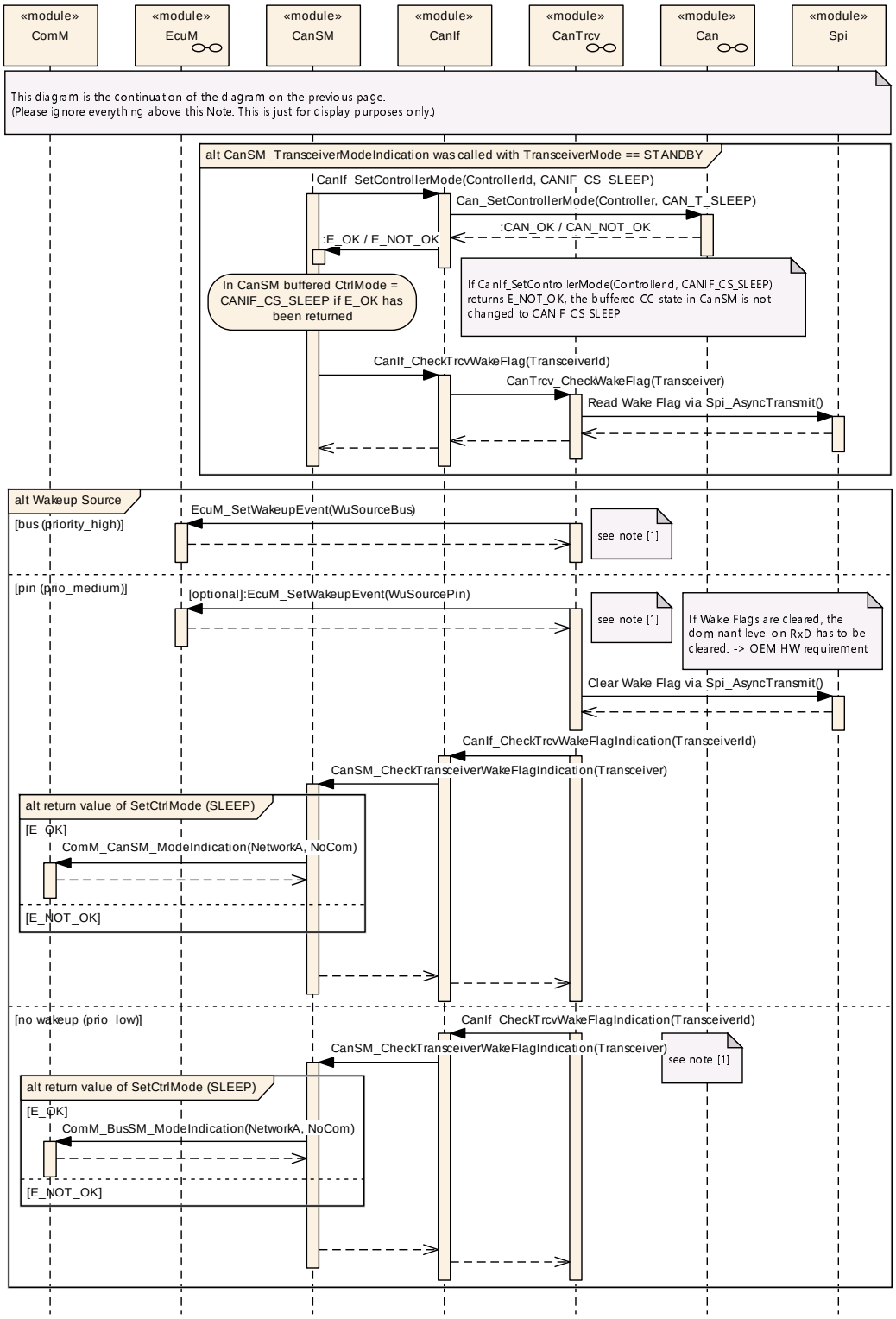

Figure 9.5: CanTrcv Deinit SPI asynchronous II

10 Configuration specification

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

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

Chapter 10.3 specifies published information of the module CanTrcv.

10.1 How to read this chapter

For details refer to the chapter 10.1 “Introduction to configuration specification” in [3].

[SWS_CanTrcv_00231] ⌈The Can Transceiver Driver module shall reject configurations with partition mappings which are not supported by the implementation.⌋ *()*

10.2 Containers and configuration parameters

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

10.2.1 CanTrcv

SWS Item	[ECUC_CanTrcv_00192]
Module Name	CanTrcv
Description	Configuration of the CanTrcv (CAN Transceiver driver) module.
Post-Build Variant Support	true
Supported Config Variants	VARIANT-LINK-TIME, VARIANT-POST-BUILD, VARIANT-PRE-COMPILE

Included Containers		
Container Name	Multiplicity	Scope / Dependency
CanTrcvConfigSet	1	This container contains the configuration parameters and sub containers of the AUTOSAR CanTrcv module.
CanTrcvGeneral	1	Container gives CAN transceiver driver basic information.

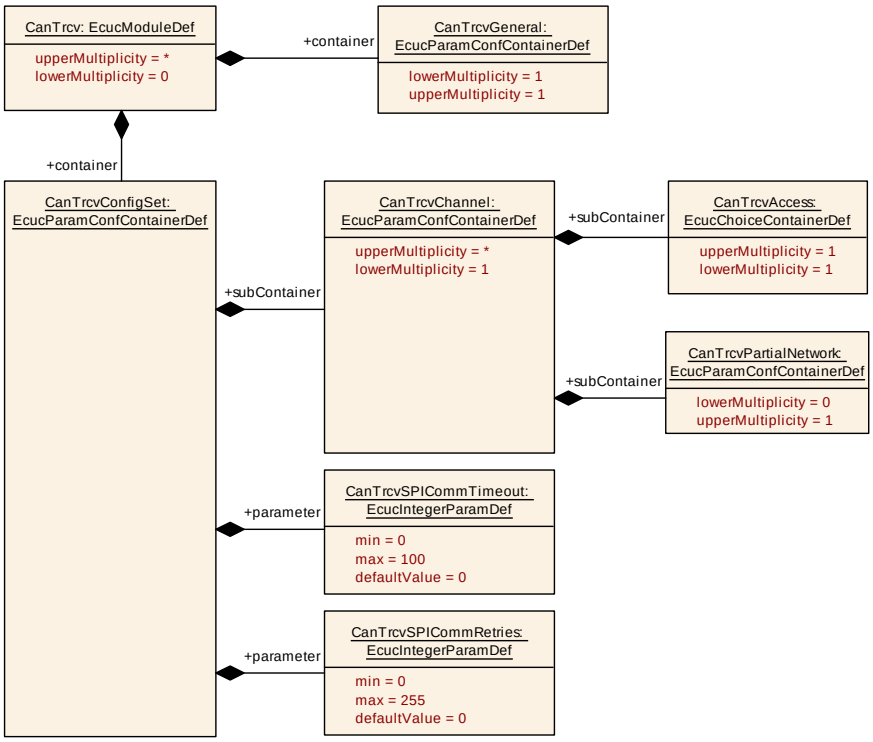

Figure 10.1: Overview of Can Tranceiver Configuration Containers

10.2.2 CanTrcvGeneral

SWS Item	**[ECUC_CanTrcv_00090]**
Container Name	CanTrcvGeneral
Parent Container	CanTrcv
Description	Container gives CAN transceiver driver basic information.
Configuration Parameters	

SWS Item	**[ECUC_CanTrcv_00152]**		
Parameter Name	CanTrcvDevErrorDetect		
Parent Container	CanTrcvGeneral		
Description	Switches the development error detection and notification on or off. • true: detection and notification is enabled. • false: detection and notification is disabled.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	[ECUC_CanTrcv_00184]		
Parameter Name	CanTrcvIndex		
Parent Container	CanTrcvGeneral		
Description	Specifies the InstanceId of this module instance. If only one instance is present it shall have the Id 0.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local withAuto = true		

SWS Item	[ECUC_CanTrcv_00187]		
Parameter Name	CanTrcvMainFunctionDiagnosticsPeriod		
Parent Container	CanTrcvGeneral		
Description	This parameter describes the period for cyclic call to CanTrcv_MainFunction Diagnostics. Unit is seconds.		
Multiplicity	0..1		
Type	EcucFloatParamDef		
Range	]0 .. INF[		
Default value	–		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	[ECUC_CanTrcv_00186]		
Parameter Name	CanTrcvMainFunctionPeriod		
Parent Container	CanTrcvGeneral		
Description	This parameter describes the period for cyclic call to CanTrcv_MainFunction. Unit is seconds.		
Multiplicity	0..1		
Type	EcucFloatParamDef		
Range	]0 .. INF[		
Default value	–		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	Pre-compile time	X	All Variants
	Link time	–	

△

	Post-build time	–	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	**[ECUC_CanTrcv_00190]**		
Parameter Name	CanTrcvTimerType		
Parent Container	CanTrcvGeneral		
Description	Type of the Time Service Predefined Timer.		
Multiplicity	0..1		
Type	EcucEnumerationParamDef		
Range	None	None	
	Timer_1us16bit	16 bit 1us timer	
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	**[ECUC_CanTrcv_00153]**		
Parameter Name	CanTrcvVersionInfoApi		
Parent Container	CanTrcvGeneral		
Description	Switches version information API on and off. If switched off, function need not be present in compiled code.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	**[ECUC_CanTrcv_00191]**	
Parameter Name	CanTrcvWaitTime	
Parent Container	CanTrcvGeneral	
Description	Wait time for transceiver state changes in seconds.	
Multiplicity	0..1	
Type	EcucFloatParamDef	
Range	[0 .. 2.55E-4]	
Default value	–	
Post-Build Variant Multiplicity	false	

▽

△

Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	**[ECUC_CanTrcv_00154]**	
Parameter Name	CanTrcvWakeUpSupport	
Parent Container	CanTrcvGeneral	
Description	Informs whether wake up is supported by polling or not supported. In case no wake up is supported by the hardware, setting has to be NOT_SUPPORTED. Only in the case of wake up supported by polling, function CanTrcv_MainFunction has to be present and to be invoked by the scheduler.	
Multiplicity	1	
Type	EcucEnumerationParamDef	
Range	CANTRCV_WAKEUP_BY_POLLING	Wake up by polling
	CANTRCV_WAKEUP_NOT_SUPPORTED	Wake up is not supported
Post-Build Variant Value	false	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local dependency: CanTrcvWakeupByBusUsed	

SWS Item	**[ECUC_CanTrcv_00193]**		
Parameter Name	CanTrcvEcucPartitionRef		
Parent Container	CanTrcvGeneral		
Description	Maps the CAN transceiver driver to zero or multiple ECUC partitions to make the modules API available in this partition. The module will operate as an independent instance in each of the partitions.		
Multiplicity	0..*		
Type	Reference to EcucPartition		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

No Included Containers

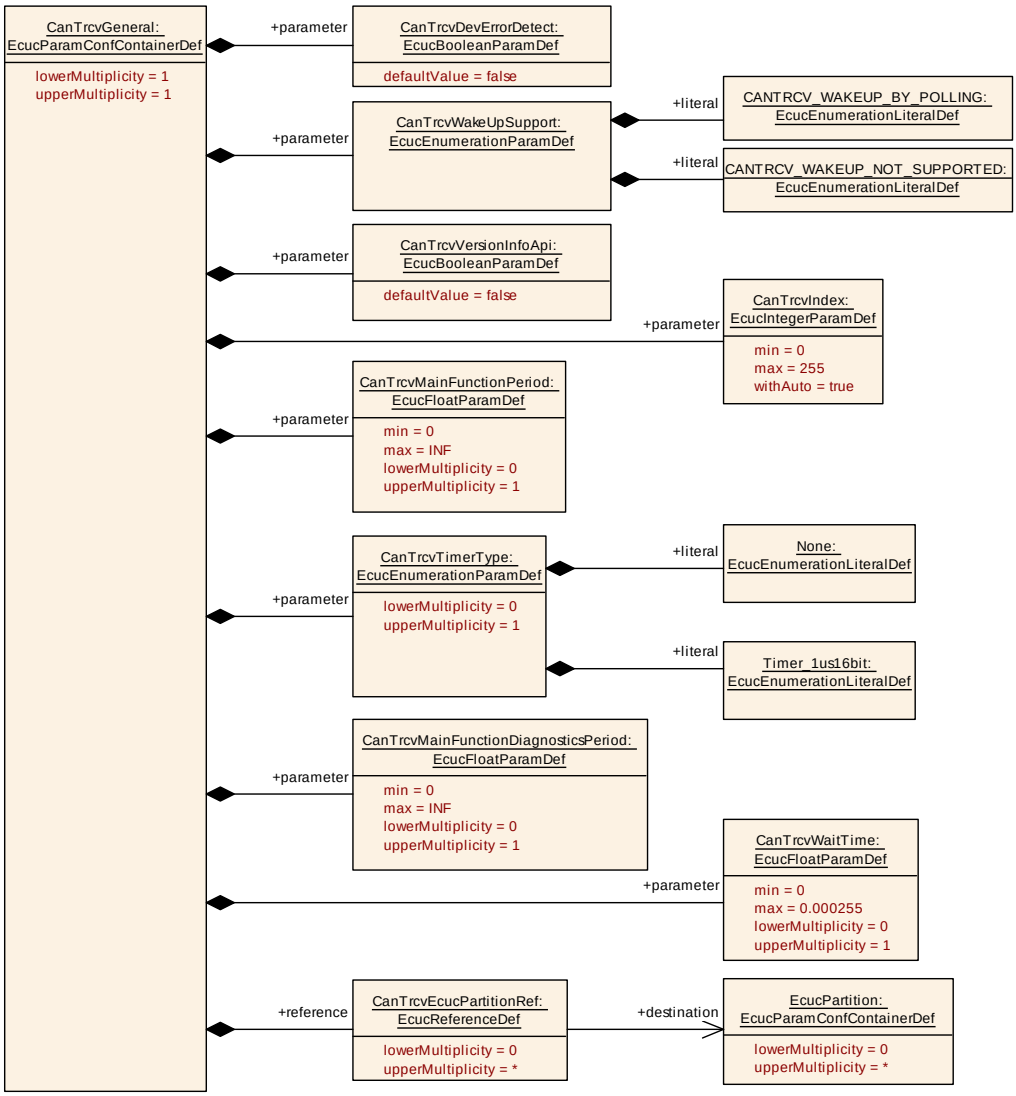

Figure 10.2: Overview of CanTrcvGeneral Container

10.2.3 CanTrcvConfigSet

SWS Item	[ECUC_CanTrcv_00173]
Container Name	CanTrcvConfigSet
Parent Container	CanTrcv
Description	This container contains the configuration parameters and sub containers of the AUTOSAR CanTrcv module.
Configuration Parameters	

SWS Item	[ECUC_CanTrcv_00175]
Parameter Name	CanTrcvSPICommRetries
Parent Container	CanTrcvConfigSet

▽

△

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	EcucIntegerParamDef		
Range	0 .. 255		
Default value	0		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local dependency: This parameter exists only if atleast one SPI Sequence is referenced in CanTrcvSpiSequence.		

SWS Item	**[ECUC_CanTrcv_00174]**		
Parameter Name	CanTrcvSPICommTimeout		
Parent Container	CanTrcvConfigSet		
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	EcucIntegerParamDef		
Range	0 .. 100		
Default value	0		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local dependency: This parameter exists only if atleast 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).

10.2.4 CanTrcvChannel

SWS Item	**[ECUC_CanTrcv_00143]**
Container Name	CanTrcvChannel
Parent Container	CanTrcvConfigSet
Description	Container gives CAN transceiver driver information about a single CAN transceiver (channel).

▽

△

Post-Build Variant Multiplicity	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Configuration Parameters			

SWS Item	**[ECUC_CanTrcv_00155]**		
Parameter Name	CanTrcvChannelId		
Parent Container	CanTrcvChannel		
Description	Unique identifier of the CAN Transceiver Channel.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 255		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU withAuto = true		

SWS Item	**[ECUC_CanTrcv_00096]**		
Parameter Name	CanTrcvChannelUsed		
Parent Container	CanTrcvChannel		
Description	Shall the related CAN transceiver channel be used?		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	true		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	**[ECUC_CanTrcv_00097]**		
Parameter Name	CanTrcvControlsPowerSupply		
Parent Container	CanTrcvChannel		
Description	Is ECU power supply controlled by this transceiver? TRUE = Controlled by transceiver. FALSE = Not controlled by transceiver.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	[ECUC_CanTrcv_00160]		
Parameter Name	CanTrcvHwPnSupport		
Parent Container	CanTrcvChannel		
Description	Indicates whether the HW supports the selective wake-up function TRUE = Selective wakeup feature is supported by the transceiver FALSE = Selective wakeup functionality is not available in transceiver		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local dependency: CanTrcvWakeUpSupport		

SWS Item	[ECUC_CanTrcv_00146]		
Parameter Name	CanTrcvInitState		
Parent Container	CanTrcvChannel		
Description	State of CAN transceiver after call to CanTrcv_Init.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	CANTRCV_OP_MODE_SLEEP	Sleep operation mode	
	CANTRCV_OP_MODE_STANDBY	Standby operation mode	
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	[ECUC_CanTrcv_00147]		
Parameter Name	CanTrcvMaxBaudrate		
Parent Container	CanTrcvChannel		
Description	Indicates the data transfer rate in kbps. Maximum data transfer rate in kbps for transceiver hardware type. Only used for validation purposes. This value can be used by configuration tools.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 20000		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	[ECUC_CanTrcv_00148]		
Parameter Name	CanTrcvWakeupByBusUsed		
Parent Container	CanTrcvChannel		
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	0..1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local dependency: CanTrcvWakeUpSupport		

SWS Item	[ECUC_CanTrcv_00194]		
Parameter Name	CanTrcvChannelEcucPartitionRef		
Parent Container	CanTrcvChannel		
Description	Maps the CAN transceiver channel to zero or one ECUC partitions. The ECUC partition referenced is a subset of the ECUC partitions where the CAN transceiver driver is mapped to.		
Multiplicity	0..1		
Type	Reference to EcucPartition		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

SWS Item	[ECUC_CanTrcv_00185]		
Parameter Name	CanTrcvIcuChannelRef		
Parent Container	CanTrcvChannel		
Description	Reference to the IcuChannel to enable/disable the interrupts for wakeups.		
Multiplicity	0..1		
Type	Symbolic name reference to IcuChannel		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	Pre-compile time	X	All Variants
	Link time	–	

△

	Post-build time	–	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency			

SWS Item	**[ECUC_CanTrcv_00181]**		
Parameter Name	CanTrcvPorWakeupSourceRef		
Parent Container	CanTrcvChannel		
Description	Symbolic name reference to specify the wakeup sources that should be used in the calls to EcuM_SetWakeupEvent as specified in [SWS_CanTrcv_00183] and [SWS_CanTrcv_00184]. This reference is mandatory if the HW supports POR or SYSERR flags		
Multiplicity	0..1		
Type	Symbolic name reference to EcuMWakeupSource		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

SWS Item	**[ECUC_CanTrcv_00182]**		
Parameter Name	CanTrcvSyserrWakeupSourceRef		
Parent Container	CanTrcvChannel		
Description	Symbolic name reference to specify the wakeup sources that should be used in the calls to EcuM_SetWakeupEvent as specified in [SWS_CanTrcv_00183] and [SWS_CanTrcv_00184] This reference is mandatory if the HW supports POR or SYSERR flags		
Multiplicity	0..1		
Type	Symbolic name reference to EcuMWakeupSource		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

SWS Item	**[ECUC_CanTrcv_00177]**
Parameter Name	CanTrcvWakeupSourceRef
Parent Container	CanTrcvChannel

▽

△

Description	Reference to a wakeup source in the EcuM configuration. This reference is only needed if CanTrcvWakeupByBusUsed is true.		
Multiplicity	0..1		
Type	Symbolic name reference to EcuMWakeupSource		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	**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	Container gives CanTrcv Driver information about access to a single CAN transceiver.
CanTrcvDemEventParameterRefs	0..1	Container for the references to DemEventParameter elements which shall be invoked using the API Dem_SetEventStatus in case the corresponding error occurs. The EventId is taken from the referenced DemEventParameter's DemEventId symbolic value. The standardized errors are provided in this container and can be extended by vendor-specific error references.
CanTrcvPartialNetwork	0..1	Container gives CAN transceiver driver information about the configuration of Partial Networking functionality.
CanXLTrcvChannel	0..1	This container is specified in the SWS CAN XL Transceiver Driver and represents a CAN XL transceiver channel. If this container is present, the CAN transceiver will provide the extended CanXLTrcv API.

[SWS_CanTrcv_00233] ⌈The ECUC partitions referenced by `CanTrcvEcucPartitionRef`. shall be a subset of the ECUC partitions referenced by `CanTrcvEcucPartitionRef`.⌋*()*

[SWS_CanTrcv_CONSTR_00235] ⌈If `CanTrcvEcucPartitionRef` references one or more ECUC partitions, `CanTrcvEcucPartitionRef` shall have a multiplicity of one and reference one of these ECUC partitions as well.⌋*()*

[SWS_CanTrcv_00234] ⌈`CanTrcvChannel` and CanController of one communication channel shall all reference the same ECUC partition.⌋*()*

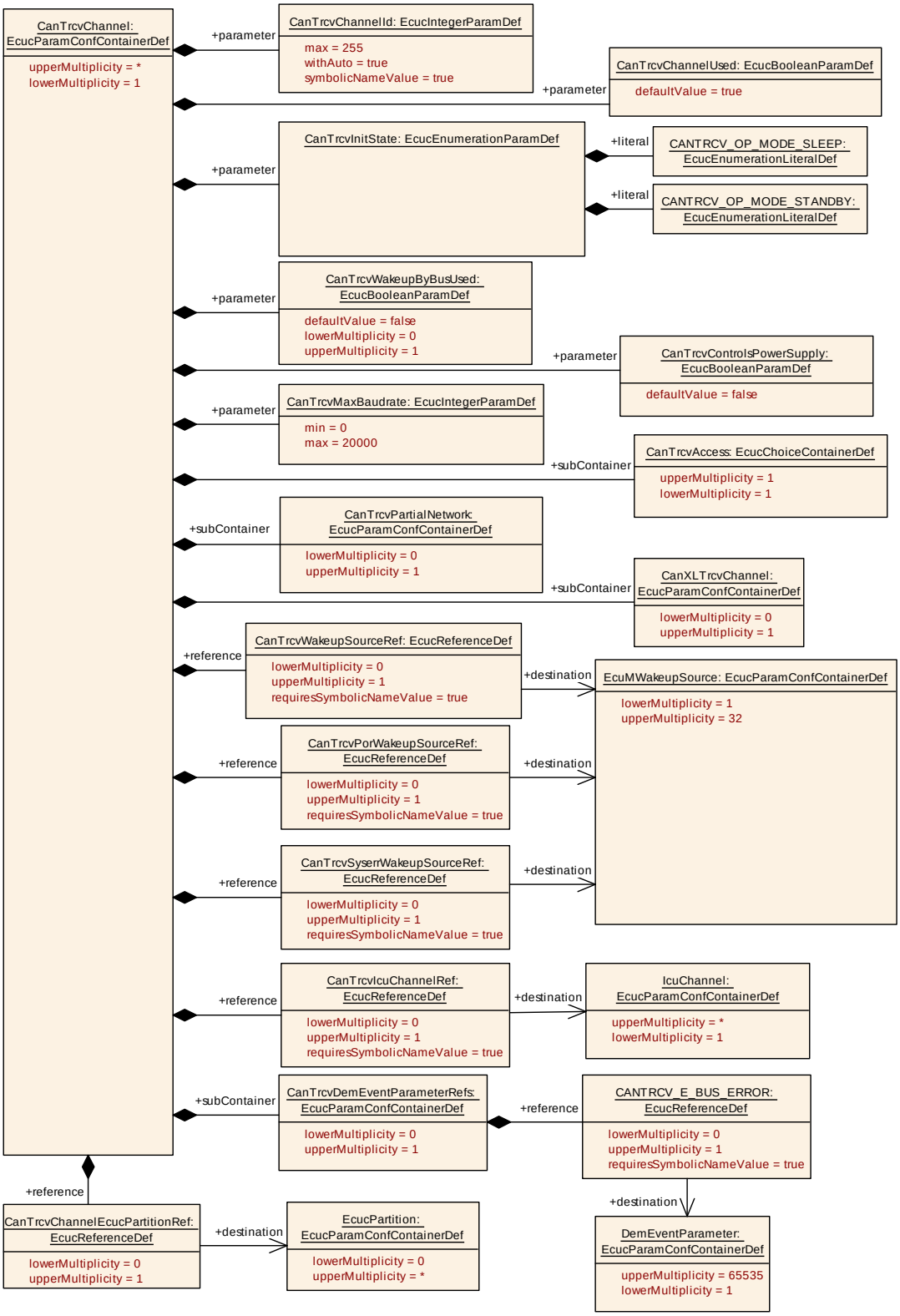

Figure 10.3: Overview of CanTrcvChannel Container

10.2.5 CanTrcvAccess

SWS Item	[ECUC_CanTrcv_00101]
Choice Container Name	CanTrcvAccess
Parent Container	CanTrcvChannel
Description	Container gives CanTrcv Driver information about access to a single CAN transceiver.

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.

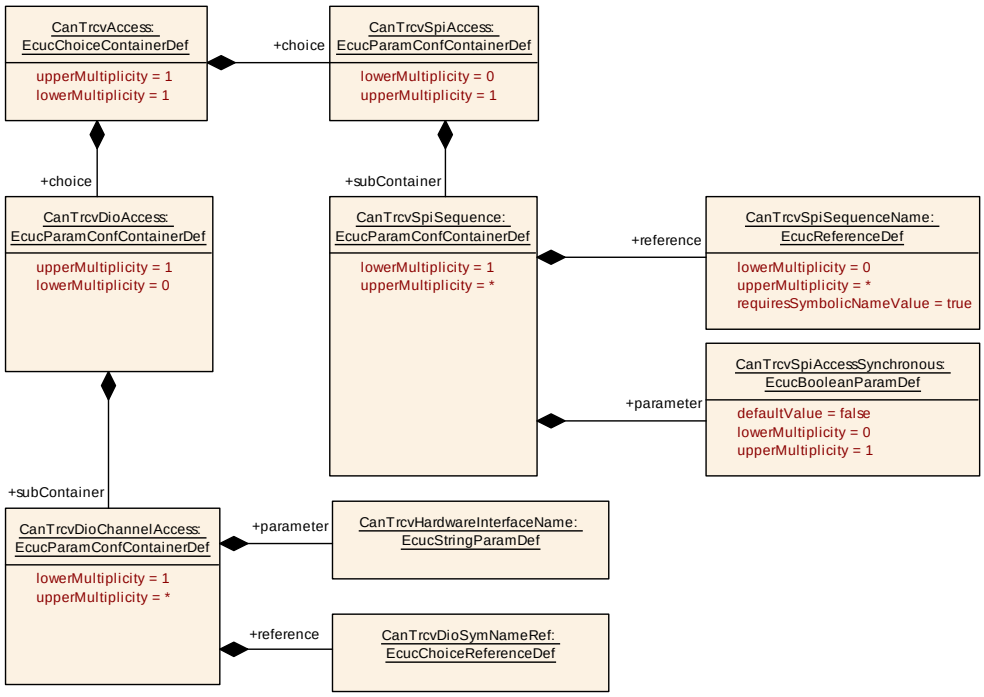

Figure 10.4: Overview of CanTrcvAccess Container

10.2.6 CanTrcvDioAccess

SWS Item	[ECUC_CanTrcv_00145]
Container Name	CanTrcvDioAccess
Parent Container	CanTrcvAccess

△

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	

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
CanTrcvDioChannelAccess	1..*	Container gives DIO channel access by single Can transceiver channel.

10.2.7 CanTrcvDioChannelAccess

SWS Item	**[ECUC_CanTrcv_00157]**
Container Name	CanTrcvDioChannelAccess
Parent Container	CanTrcvDioAccess
Description	Container gives DIO channel access by single Can transceiver channel.
Configuration Parameters	

SWS Item	**[ECUC_CanTrcv_00150]**		
Parameter Name	CanTrcvHardwareInterfaceName		
Parent Container	CanTrcvDioChannelAccess		
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	EcucStringParamDef		
Default value	–		
Regular Expression	–		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	**[ECUC_CanTrcv_00149]**
Parameter Name	CanTrcvDioSymNameRef
Parent Container	CanTrcvDioChannelAccess
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]
Post-Build Variant Value	false

▽

△

Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency			

No Included Containers

10.2.8 CanTrcvSpiAccess

SWS Item	**[ECUC_CanTrcv_00183]**
Container Name	CanTrcvSpiAccess
Parent Container	CanTrcvAccess
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 CAN transceiver driver information about one SPI sequence. One SPI sequence used by CAN transceiver driver is in exclusive use for it. No other driver is allowed to access this sequence. CAN transceiver driver may use one sequence to access n CAN transceiver hardwares chips of the same type or n sequences are used to access one single CAN transceiver hardware chip. If a CAN transceiver hardware has no SPI interface, there is no instance of this container.

10.2.9 CanTrcvSpiSequence

SWS Item	**[ECUC_CanTrcv_00144]**
Container Name	CanTrcvSpiSequence
Parent Container	CanTrcvSpiAccess
Description	Container gives CAN transceiver driver information about one SPI sequence. One SPI sequence used by CAN transceiver driver is in exclusive use for it. No other driver is allowed to access this sequence. CAN transceiver driver may use one sequence to access n CAN transceiver hardwares chips of the same type or n sequences are used to access one single CAN transceiver hardware chip. If a CAN transceiver hardware has no SPI interface, there is no instance of this container.
Configuration Parameters	

SWS Item	**[ECUC_CanTrcv_00176]**
Parameter Name	CanTrcvSpiAccessSynchronous
Parent Container	CanTrcvSpiSequence

▽

△

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	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	**[ECUC_CanTrcv_00151]**		
Parameter Name	CanTrcvSpiSequenceName		
Parent Container	CanTrcvSpiSequence		
Description	Reference to a Spi sequence configuration container.		
Multiplicity	0..*		
Type	Symbolic name reference to SpiSequence		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local dependency: SpiSequence		

No Included Containers

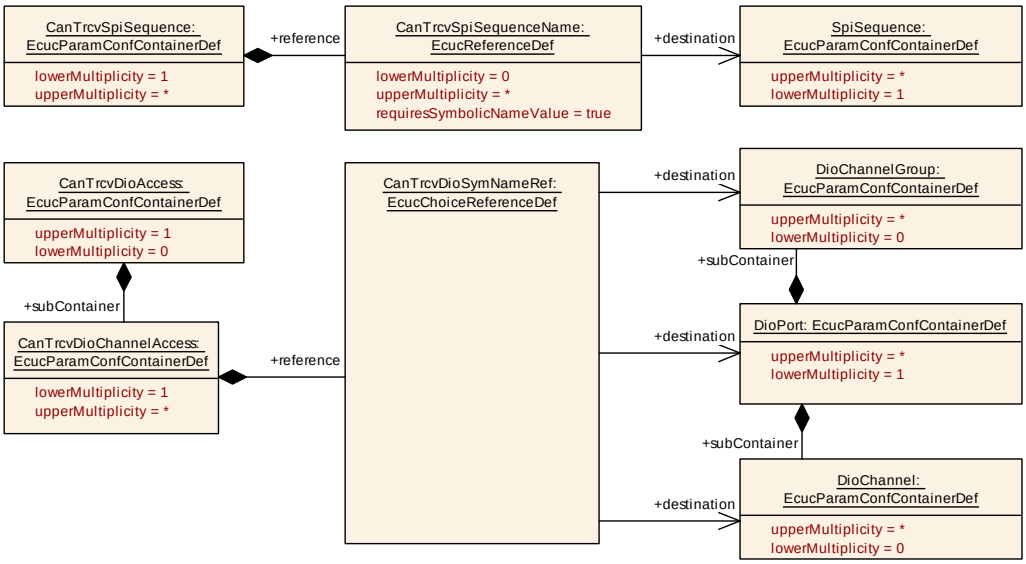

Figure 10.5: CanTrcv References to Dio and Spi

10.2.10 CanTrcvDemEventParameterRefs

SWS Item	**[ECUC_CanTrcv_00188]**
Container Name	CanTrcvDemEventParameterRefs
Parent Container	CanTrcvChannel
Description	Container for the references to DemEventParameter elements which shall be invoked using the API Dem_SetEventStatus in case the corresponding error occurs. The Event Id is taken from the referenced DemEventParameter's DemEventId symbolic value. The standardized errors are provided in this container and can be extended by vendor-specific error references.
Configuration Parameters	

SWS Item	**[ECUC_CanTrcv_00189]**		
Parameter Name	CANTRCV_E_BUS_ERROR		
Parent Container	CanTrcvDemEventParameterRefs		
Description	Reference to the DemEventParameter which shall be issued when bus error has occurred.		
Multiplicity	0..1		
Type	Symbolic name reference to DemEventParameter		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	

▽

△

Scope / Dependency	scope: ECU dependency: DEM

No Included Containers

10.2.11 CanTrcvPartialNetwork

SWS Item	**[ECUC_CanTrcv_00161]**
Container Name	CanTrcvPartialNetwork
Parent Container	CanTrcvChannel
Description	Container gives CAN transceiver driver information about the configuration of Partial Networking functionality.
Configuration Parameters	

SWS Item	**[ECUC_CanTrcv_00169]**		
Parameter Name	CanTrcvBaudRate		
Parent Container	CanTrcvPartialNetwork		
Description	Indicates the data transfer rate in kbps.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 12000		
Default value	–		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local dependency: Although WUF with DLC=0 is technically possible, it is explicitly not wanted.		

SWS Item	**[ECUC_CanTrcv_00171]**		
Parameter Name	CanTrcvBusErrFlag		
Parent Container	CanTrcvPartialNetwork		
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	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	[ECUC_CanTrcv_00164]		
Parameter Name	CanTrcvPnCanIdIsExtended		
Parent Container	CanTrcvPartialNetwork		
Description	Indicates whether extended or standard ID is used. TRUE = Extended Can identifier is used. FALSE = Standard Can identifier is used		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	[ECUC_CanTrcv_00172]		
Parameter Name	CanTrcvPnEnabled		
Parent Container	CanTrcvPartialNetwork		
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	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	[ECUC_CanTrcv_00163]		
Parameter Name	CanTrcvPnFrameCanId		
Parent Container	CanTrcvPartialNetwork		
Description	CAN ID of the Wake-up Frame (WUF).		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	–		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	[ECUC_CanTrcv_00162]
Parameter Name	CanTrcvPnFrameCanIdMask
Parent Container	CanTrcvPartialNetwork
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	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	–		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**[ECUC_CanTrcv_00168]**		
Parameter Name	CanTrcvPnFrameDlc		
Parent Container	CanTrcvPartialNetwork		
Description	Data Length of the Wake-up Frame (WUF).		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 8		
Default value	–		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**[ECUC_CanTrcv_00170]**		
Parameter Name	CanTrcvPowerOnFlag		
Parent Container	CanTrcvPartialNetwork		
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	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
CanTrcvPnFrameDataMaskSpec	0..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).

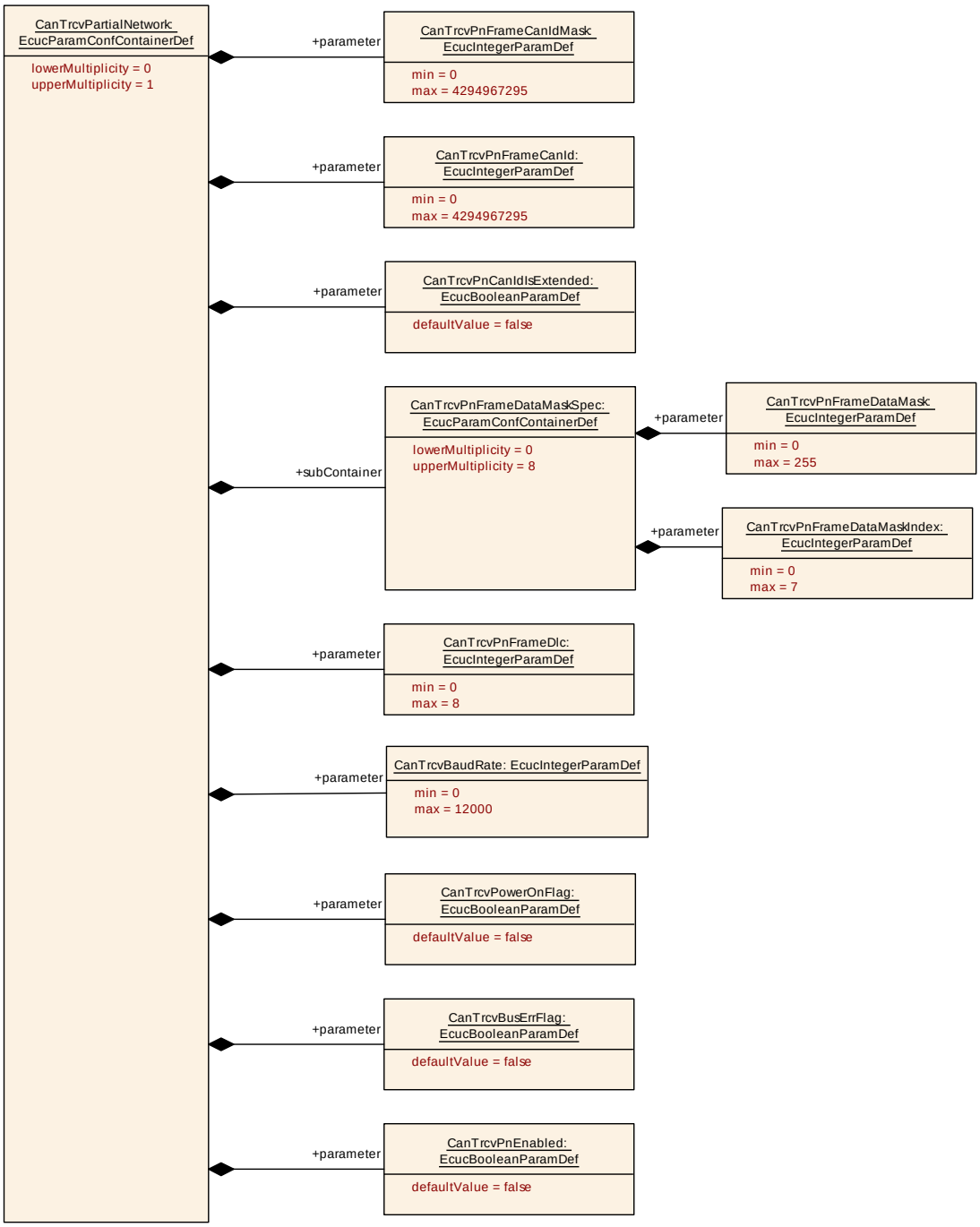

Figure 10.6: CanTrcv Partial Network

10.2.12 CanTrcvPnFrameDataMaskSpec

SWS Item	[ECUC_CanTrcv_00165]
Container Name	CanTrcvPnFrameDataMaskSpec
Parent Container	CanTrcvPartialNetwork
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	[ECUC_CanTrcv_00166]		
Parameter Name	CanTrcvPnFrameDataMask		
Parent Container	CanTrcvPnFrameDataMaskSpec		
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	EcucIntegerParamDef		
Range	0 .. 255		
Default value	–		
Post-Build Variant Value	true		
Value Configuration Class	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	[ECUC_CanTrcv_00167]		
Parameter Name	CanTrcvPnFrameDataMaskIndex		
Parent Container	CanTrcvPnFrameDataMaskSpec		
Description	holds the position n in frame of the mask-part		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 7		
Default value	–		
Post-Build Variant Value	true		
Value Configuration Class	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers

10.3 Published Information

For details refer to the [3, chapter 10.3 "Published Information" in SWS_BSWGeneral].

A Not applicable requirements

[SWS_CanTrcv_NA_00999] ⌈These requirements are not applicable to this specification.⌋*(SRS_BSW_00304, SRS_BSW_00305, SRS_BSW_00306, SRS_BSW_00307, SRS_BSW_00308, SRS_BSW_00309, SRS_BSW_00312, SRS_BSW_00321, SRS_BSW_00325, SRS_BSW_00328, SRS_BSW_00330, SRS_BSW_00333, SRS_BSW_00334, SRS_BSW_00341, SRS_BSW_00342, SRS_BSW_00344, SRS_BSW_00359, SRS_BSW_00360, SRS_BSW_00378, SRS_BSW_00383, SRS_BSW_00384, SRS_BSW_00398, SRS_BSW_00399, SRS_BSW_00400, SRS_BSW_00401, SRS_BSW_00404, SRS_BSW_00405, SRS_BSW_00410, SRS_BSW_00416, SRS_BSW_00417, SRS_BSW_00422, SRS_BSW_00423, SRS_BSW_00426, SRS_BSW_00427, SRS_BSW_00429, SRS_BSW_00432, SRS_BSW_00433, SRS_BSW_00005, SRS_BSW_00006, SRS_BSW_00007, SRS_BSW_00009, SRS_BSW_00010, SRS_BSW_00161, SRS_BSW_00164, SRS_BSW_00168, SRS_Can_01107, SRS_Can_01138)*

B Change History

Please note that the lists in this chapter also include constraints and specification items that have been removed from the specification in a later version. These constraints and specification items do not appear as hyperlinks in the document.

B.1 Change History of this document according to AUTOSAR Release R23-11

B.1.1 Added Specification Items in R23-11

Number	Heading
[SWS_CanTrcv_-00001]	Definition of API function CanTrcv_Init
[SWS_CanTrcv_-00002]	Definition of API function CanTrcv_SetOpMode
[SWS_CanTrcv_-00005]	Definition of API function CanTrcv_GetOpMode
[SWS_CanTrcv_-00007]	Definition of API function CanTrcv_GetBusWuReason
[SWS_CanTrcv_-00008]	Definition of API function CanTrcv_GetVersionInfo
[SWS_CanTrcv_-00009]	Definition of API function CanTrcv_SetWakeupMode
[SWS_CanTrcv_-00013]	Definition of scheduled function CanTrcv_MainFunction
[SWS_CanTrcv_-00016]	
[SWS_CanTrcv_-00050]	Definiton of development errors in module CanTrcv
[SWS_CanTrcv_-00055]	
[SWS_CanTrcv_-00064]	
[SWS_CanTrcv_-00065]	
[SWS_CanTrcv_-00084]	Definition of imported datatypes of module CanTrcv
[SWS_CanTrcv_-00085]	Definition of mandatory interfaces in module CanTrcv
[SWS_CanTrcv_-00086]	Definition of optional interfaces in module CanTrcv

▽

△

Number	Heading
[SWS_CanTrcv_-00087]	
[SWS_CanTrcv_-00089]	
[SWS_CanTrcv_-00090]	
[SWS_CanTrcv_-00091]	
[SWS_CanTrcv_-00093]	
[SWS_CanTrcv_-00094]	
[SWS_CanTrcv_-00095]	
[SWS_CanTrcv_-00098]	
[SWS_CanTrcv_-00099]	
[SWS_CanTrcv_-00102]	
[SWS_CanTrcv_-00105]	
[SWS_CanTrcv_-00106]	
[SWS_CanTrcv_-00107]	
[SWS_CanTrcv_-00111]	
[SWS_CanTrcv_-00112]	
[SWS_CanTrcv_-00113]	
[SWS_CanTrcv_-00114]	
[SWS_CanTrcv_-00115]	
[SWS_CanTrcv_-00116]	
[SWS_CanTrcv_-00117]	
[SWS_CanTrcv_-00120]	
[SWS_CanTrcv_-00121]	

▽

△

Number	Heading
[SWS_CanTrcv_-00122]	
[SWS_CanTrcv_-00123]	
[SWS_CanTrcv_-00124]	
[SWS_CanTrcv_-00125]	
[SWS_CanTrcv_-00127]	
[SWS_CanTrcv_-00129]	
[SWS_CanTrcv_-00130]	
[SWS_CanTrcv_-00131]	
[SWS_CanTrcv_-00132]	
[SWS_CanTrcv_-00133]	
[SWS_CanTrcv_-00143]	Definition of API function CanTrcv_CheckWakeup
[SWS_CanTrcv_-00144]	
[SWS_CanTrcv_-00145]	
[SWS_CanTrcv_-00146]	
[SWS_CanTrcv_-00148]	
[SWS_CanTrcv_-00150]	
[SWS_CanTrcv_-00158]	
[SWS_CanTrcv_-00161]	
[SWS_CanTrcv_-00163]	Definition of datatype CanTrcv_TrcvModeType
[SWS_CanTrcv_-00164]	Definition of datatype CanTrcv_TrcvWakeupModeType
[SWS_CanTrcv_-00165]	Definition of datatype CanTrcv_TrcvWakeupReasonType
[SWS_CanTrcv_-00167]	

▽

△

Number	Heading
[SWS_CanTrcv_-00168]	
[SWS_CanTrcv_-00171]	
[SWS_CanTrcv_-00172]	
[SWS_CanTrcv_-00173]	
[SWS_CanTrcv_-00174]	
[SWS_CanTrcv_-00175]	
[SWS_CanTrcv_-00177]	
[SWS_CanTrcv_-00178]	
[SWS_CanTrcv_-00180]	
[SWS_CanTrcv_-00181]	
[SWS_CanTrcv_-00182]	
[SWS_CanTrcv_-00183]	
[SWS_CanTrcv_-00184]	
[SWS_CanTrcv_-00186]	
[SWS_CanTrcv_-00187]	
[SWS_CanTrcv_-00188]	
[SWS_CanTrcv_-00189]	
[SWS_CanTrcv_-00190]	
[SWS_CanTrcv_-00191]	
[SWS_CanTrcv_-00192]	
[SWS_CanTrcv_-00193]	
[SWS_CanTrcv_-00194]	

▽

△

Number	Heading
[SWS_CanTrcv_-00195]	
[SWS_CanTrcv_-00196]	
[SWS_CanTrcv_-00197]	
[SWS_CanTrcv_-00198]	
[SWS_CanTrcv_-00199]	
[SWS_CanTrcv_-00200]	
[SWS_CanTrcv_-00201]	
[SWS_CanTrcv_-00202]	
[SWS_CanTrcv_-00203]	
[SWS_CanTrcv_-00204]	
[SWS_CanTrcv_-00205]	
[SWS_CanTrcv_-00206]	
[SWS_CanTrcv_-00209]	Definition of datatype CanTrcv_ConfigType
[SWS_CanTrcv_-00210]	Definition of datatype CanTrcv_PNActivationType
[SWS_CanTrcv_-00211]	Definition of datatype CanTrcv_TrcvFlagStateType
[SWS_CanTrcv_-00213]	Definition of API function CanTrcv_GetTrcvSystemData
[SWS_CanTrcv_-00214]	Definition of API function CanTrcv_ClearTrcvWufFlag
[SWS_CanTrcv_-00215]	Definition of API function CanTrcv_ReadTrcvTimeoutFlag
[SWS_CanTrcv_-00216]	Definition of API function CanTrcv_ClearTrcvTimeoutFlag
[SWS_CanTrcv_-00217]	Definition of API function CanTrcv_ReadTrcvSilenceFlag
[SWS_CanTrcv_-00218]	Definition of scheduled function CanTrcv_MainFunctionDiagnostics
[SWS_CanTrcv_-00219]	Definition of API function CanTrcv_SetPNActivationState

▽

△

Number	Heading
[SWS_CanTrcv_-00220]	
[SWS_CanTrcv_-00221]	
[SWS_CanTrcv_-00222]	
[SWS_CanTrcv_-00223]	Definition of API function CanTrcv_CheckWakeFlag
[SWS_CanTrcv_-00224]	
[SWS_CanTrcv_-00225]	
[SWS_CanTrcv_-00226]	
[SWS_CanTrcv_-00227]	
[SWS_CanTrcv_-00228]	
[SWS_CanTrcv_-00229]	
[SWS_CanTrcv_-00230]	
[SWS_CanTrcv_-00231]	
[SWS_CanTrcv_-00233]	
[SWS_CanTrcv_-00234]	
[SWS_CanTrcv_-91001]	Definition of API function CanTrcv_DeInit
[SWS_CanTrcv_-91002]	
[SWS_CanTrcv_-91003]	
[SWS_CanTrcv_-91004]	
[SWS_CanTrcv_-91005]	
[SWS_CanTrcv_-91006]	Definiton of runtime errors in module CanTrcv
[SWS_CanTrcv_NA_-00999]	

Table B.1: Added Specification Items in R23-11

B.1.2 Changed Specification Items in R23-11

none

B.1.3 Deleted Specification Items in R23-11

none

B.1.4 Added Constraints in R23-11

Number	Heading
[SWS_CanTrcv_CONSTR_00235]	

Table B.2: Added Constraints in R23-11

B.1.5 Changed Constraints in R23-11

none

B.1.6 Deleted Constraints in R23-11

none