Document Title	Specification of CAN State Manager
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	253
Document Classification	Standard

Document Version	1.5.0
Document Status	Final
Part of Release	3.2
Revision	3

Document Change History

Date	Version	Changed by	Change Description
28.02.2014	1.5.0	AUTOSAR Release Management	• CanSM state machine initialization does not requests any lower layer APIs anymore to avoid race conditions • CanSM considers communication mode requests also during bus-off recovery to ensure a synchronization to the NM state machine • CanSM state machine implements a new state to be involved into the wake-up validation process of the EcuM module • Editorial changes • Removed chapter(s) on change documentation
29.05.2012	1.4.0	AUTOSAR Administration	• Bus off error reported as pre failed to make bus off recovery time independent of error detection time • Different TX Online PDU modes used for the regular transition to full communication and for the transition after bus-off to avoid WUF after bus-off recovery • CanSM_TxTimeoutException recovery improved
27.04.2011	1.3.0	AUTOSAR Administration	• Partial Networking Transceiver support with specific BSW module interaction for deinitalisation • PDU group control replaced with BswM module interaction • Aggregation of the formally separated state machines into one combined

Document Change History			
Date	**Version**	**Changed by**	**Change Description**
			state machine • Legal disclaimer revised
15.09.2010	1.2.0	AUTOSAR Administration	• Add CANSM341, CANSM340, CANSM242, CANSM243 • Updated CANSM340, CANSM219, CANSM045, CANSM219, CANSM231 • Legal disclaimer revised
28.01.2010	1.1.0	AUTOSAR Administration	• Independant parameters for CAN networks. • Update of document with generated artifacts. • Legal disclaimer revised
23.06.2008	1.0.1	AUTOSAR Administration	• Legal disclaimer revised
13.11.2007	1.0.0	AUTOSAR Administration	Initial Release

Disclaimer

Advice for users

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

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

Table of Contents

1 Introduction and functional overview

This specification describes the functionality, API and the configuration for the AUTOSAR Basic Software module CAN State Manager.

The AUTOSAR BSW stack specifies for each communication bus a bus specific state manager. This module shall implement the control flow for the respective bus. Like shown in the figure below the CAN State Manager (CanSM) is member of the Communication Service Layer. It interacts with the Communication Hardware Abstraction Layer and the System Service Layer.

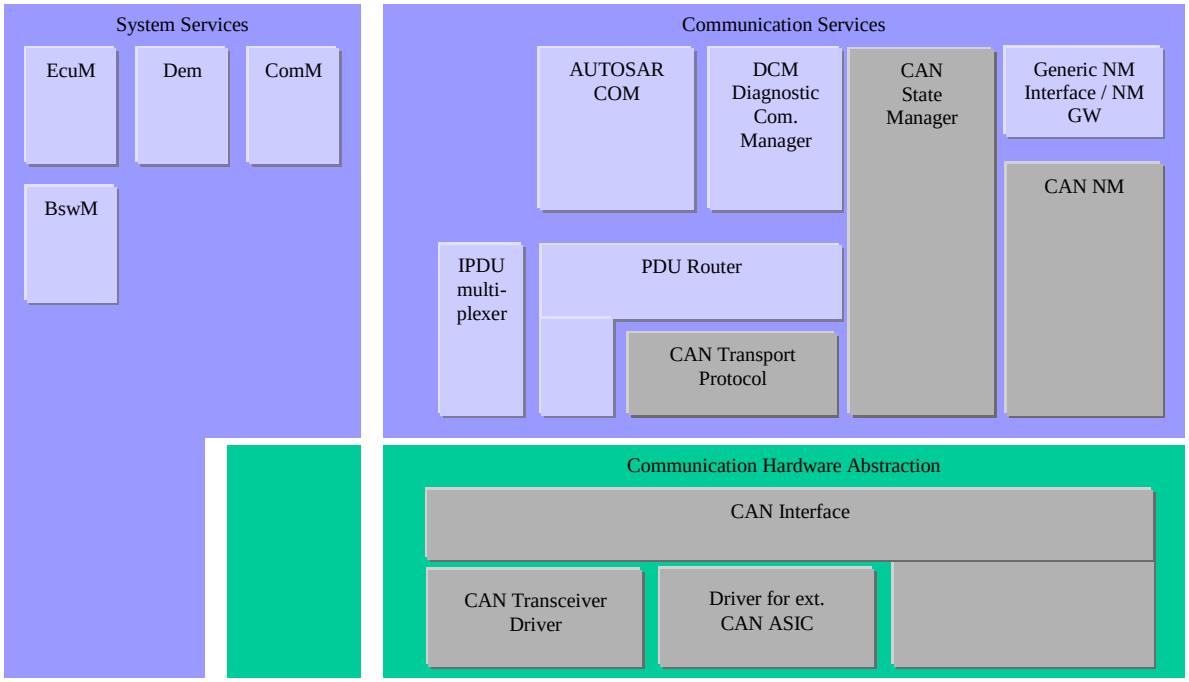

Figure 1-1: Layered Software Architecture from CanSM point of view

2 Acronyms and abbreviations

Abbreviation / Acronym:	Description:
CAN	Controller Area Network
DEM	Diagnostic Event Manager
DET	Development Error Tracer
CanSM	CAN State Manager
ComM	Communication Manager
EcuM	ECU State Manager
CanIf	CAN Interface
BSW	Basic Software
SchM	BSW Scheduler
PDU	Protocol Data Unit
RX	Receive
TX	Transmit
API	Application Program Interface
BswM	Basic Software Mode Manager
CanNm	CAN Network Management

3 Related documentation

3.1 Input documents

[1] List of Basic Software Modules
AUTOSAR_BasicSoftwareModules.pdf

[2] Layered Software Architecture
AUTOSAR_LayeredSoftwareArchitecture.pdf

[3] General Requirements on Basic Software Modules
AUTOSAR_SRS_General.pdf

[4] Specification of ECU Configuration
AUTOSAR_ECU_Configuration.pdf

[5] Specification of Standard Types
AUTOSAR_StandardTypes.pdf

[6] Specification of Communication Stack Types
AUTOSAR_SWS_ComStackType

[7] Requirements on CAN
AUTOSAR_SRS_CAN.pdf

[8] Requirements on Mode Management
AUTOSAR_SRS_ModeManagement.pdf

[9] Specification of CAN Transceiver Driver
AUTOSAR_SWS_CAN_TransceiverDriver.pdf

[10] Specification of Communication Manager
AUTOSAR_SWS_ComManager.pdf

[11] Specification of ECU State Manager

AUTOSAR_SWS_ECU_StateManager.pdf

[12] Specification of Diagnostics Event Manager
AUTOSAR_SWS_DEM.pdf

[13] Specification of CAN Interface
AUTOSAR_SWS_CAN_Interface.pdf

[14] Specification of BSW Scheduler
AUTOSAR_SWS_BSW_Scheduler.pdf

[15] Specification of Development Error Tracer
AUTOSAR_SWS_DET.pdf

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

[17] Specification of Basic Software Mode Manager,
AUTOSAR_SWS_BSWModeManager.pdf

[18] Specification of CAN Network Management,
AUTOSAR_SWS_CAN_NM.pdf

4 Constraints and assumptions

4.1 Limitations

The CanSM can be used for CAN communication only. Its dedication is to operate with the CanIf to control one ore multiple underlying CAN Controllers and CAN Transceiver Drivers. Other protocols than CAN (i.e. LIN or FlexRay) are not supported.

4.2 Applicability to car domains

The CAN State Manager can be used for all domain applications always when the CAN protocol is used.

5 Dependencies to other modules

This section describes the relations to other modules within the basic software, which is shown in the following figure.

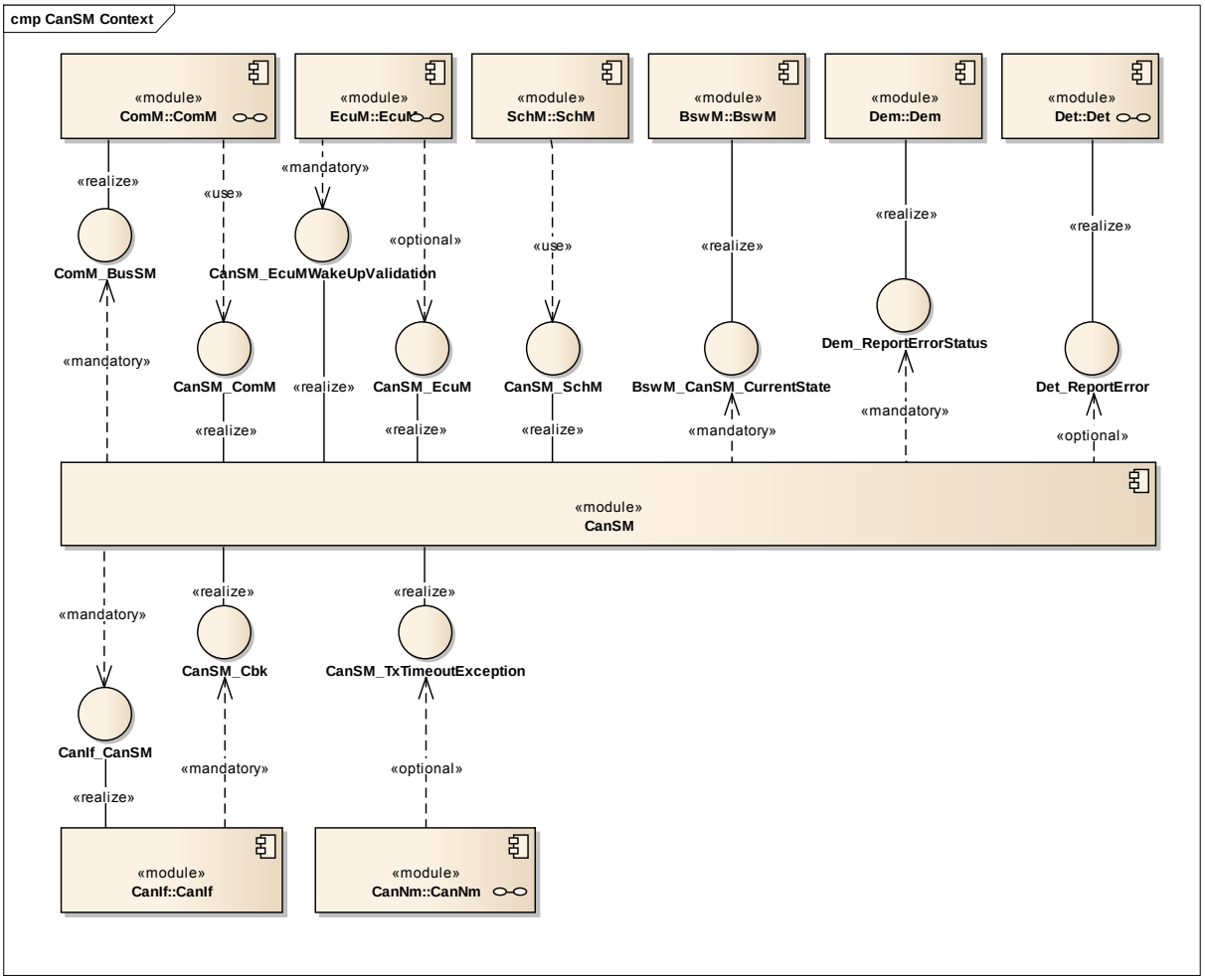

Figure 5-1: Dependencies of the CanSM to other BSW modules

The next sections give a brief description of configuration information and services the CanSM module requires from other modules.

5.1 ECU State Manager (EcuM)

The EcuM initializes the CanSM and interacts with the CanSM module for the CAN wakeup validation (refer to [11] for detailed specification of this module).

5.2 BSW Scheduler (SchM)

The BSW Scheduler calls the main function of the CanSM, which is necessary for the cyclic processes of the CanSM (refer to [14] for detailed specification of this module).

5.3 Communication Manager (ComM)

The ComM uses the API of the CanSM to request communication modes of CAN networks, which are identified with unique network handles (refer to [10] for detailed specification of this module).

The CanSM notifies the current communication mode of its CAN networks to the ComM.

5.4 CAN Interface (CanIf)

The CanSM uses the API of the CanIf to control the operating modes of the CAN controllers and CAN transceivers assigned to the CAN networks (refer to [13] for detailed specification of this module).

The CanIf notifies the CanSM about peripheral events.

5.5 Diagnostic Event Manager (DEM)

The CanSM reports bus specific production errors to the DEM (refer to [12] for detailed specification of this module).

5.6 Basic Software Mode Manager (BswM)

The CanSM module needs to notify bus specific mode changes to the BswM module (refer to [17] for detailed specification of this module).

5.7 CAN Network Management (CanNm)

The CanSM module needs to notify the partial network availability to the CanNm module (ref. to [18] for detailed specification of this module).

5.8 Development Error Tracer (DET)

The CanSM reports development errors to the DET, if development error handling is switched on by configuration (refer to [15] for detailed specification of this module).

5.9 File structure

5.9.1 Code file structure

CANSM007:

The code file structure shall not be defined within this specification completely. At this point it shall be pointed out that the code-file structure shall include the following files named:

- CanSM_Lcfg.c – for link time configurable parameters and
- CanSM_PBcfg.c – for post build time configurable parameters

These files shall contain all link time and post-build time configurable parameters.

Further more following files shall be used for implementation

- CanSM.c – for implementation of the provided functionality

5.9.2 Header file structure

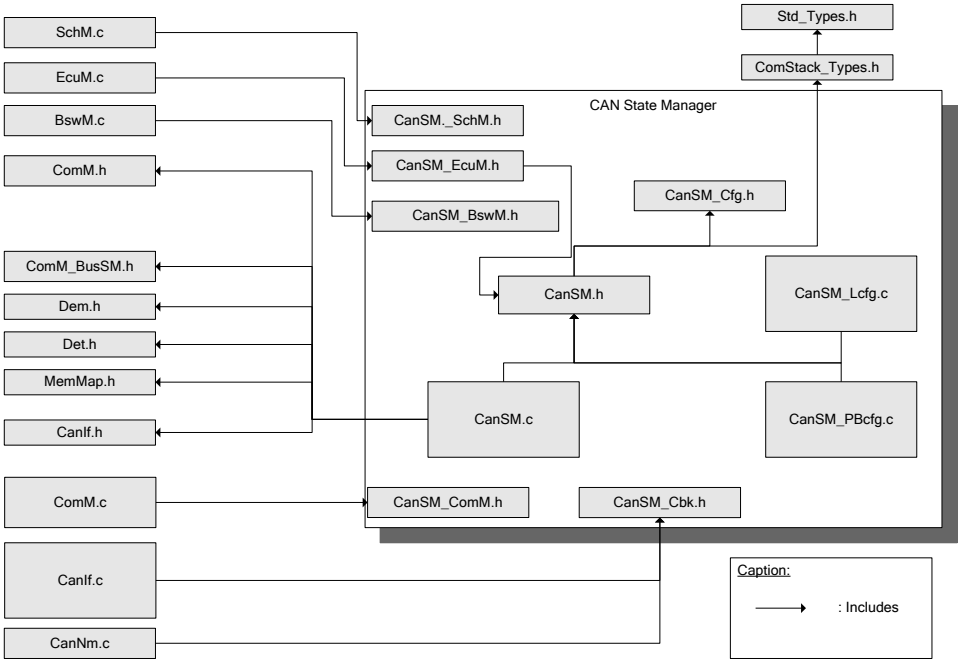

Figure 5-2: Header file structure

CANSM008:

The header file CanSM.h shall export CanSM specific types and the API of the CanSM, which is not dedicated to a certain module.

CANSM238:

The header file CanSM.h shall include the header file ComStack_Types.h.

Remark: The header file ComStack_Types.h includes the header file Std_Types.h

CANSM239:

The header file CanSM.h shall include the header file CanSM_Cfg.h.

CANSM009:

The header file CanSM_ComM.h shall export the CanSM API dedicated to the ComM.

CANSM022:

The header file CanSM_SchM.h shall export the main function of the CanSM.

Rationale: Integration of the CanSM into the BSW scheduler.

CANSM010:

The header file CanSM_Cfg.h shall contain references to the parameters of the c-source files CanSM_Lcfg.c and CanSM_PBcfg.c (see section 5.9.1 above) and pre-compile parameters, which are not declared as “const” parameter, but as defines.

CANSM011:

The header file CanSM_Cbk.h shall be dedicated to declare call-out notification functions of the CanSM.

CANSM013:

The CanSM (CanSM.c) shall reference its header file CanSM.h to get access to its own API declaration and to its configuration parameters.

CANSM014:

The CanSM shall include the header file Dem.h.

Rationale: To report production errors.

CANSM015:

The CanSM shall include the header file Det.h.

Rationale: To report development errors.

CANSM016:

The CanSM shall include the header file MemMap.h.

Rationale: To make it possible to map the code and the data of the CanSM into specific memory sections.

CANSM017:

The CanSM shall include the header file CanIf.h.

Rationale: API of the CanIf needed for peripheral control.

CANSM174:

The CanSM shall include the header file ComM.h.

Rationale: To get the type definitions of the ComM.

CANSM191:

The CanSM shall include the header file ComM_BusSM.h.

Rationale: This file provides the API of the ComM, which is exclusively intended for the bus state managers.

CANSM345:

The header file CanSM_BswM.h shall declare the interfaces, which are dedicated to the BswM module.

CANSM346:

The CanSM shall include the header file BswM_CanSM_CurrentState.h.

Rationale: This interface provides the mandatory API of the BswM module to the CanSM module.

[CANSM415:]

The CanSM module (CanSM.c) shall include the header file CanNm_Cbk.h, if Partial Networking is enabled (ref. to CANSM134_Conf).

6 Requirements traceability

According to [3] (General BSW Requirements):

Requirement	***Satisfied by***
[BSW00344] Reference to link-time configuration	CANSM007: CANSM010:
[BSW0404] Reference to post build time configuration	CANSM007: CANSM010:
[BSW0405] Reference to multiple configuration sets	CANSM061: CANSM023:
[BSW00345] Pre-compile time configuration	CANSM007: CANSM010: Chapter 10.2 **CANSM127 :**
[BSW159] Tool based configuration	CANSM155:
[BSW167] Static configuration checking	CANSM156:
[BSW171] Configurability of optional functionality	CANSM015: Chapter 10.2
[BSW170] Data for reconfiguration of SW-components	Not applicable (requirement on SWC-module)
[BSW00380] Separate C-Files for configuration parameters	CANSM007:
[BSW00419] Separate C-Files for pre-compile time configuration parameters	CANSM007:
[BSW00381] Separate configuration header file for pre-compile time parameters	CANSM010:
[BSW00412] Separate configuration header file for configuration parameters	CANSM010:
[BSW00383] List dependencies of configuration files	Chapter 10.2 **CANSM141 :**
[BSW00384] List dependencies to other modules	Chapter 5
[BSW00387] Specify the configuration class of callback function	Chapter 8.4
[BSW00388] Introduce containers	Chapter 10.2 **CANSM127 :**
[BSW00389] Containers shall have names	Chapter 10.2
[BSW00390] Parameter content shall be unique within the module	Chapter 10.2
[BSW00391] Parameter shall have unique names	Chapter 10.2
[BSW00392] Parameters shall have a type	Chapter 10.2
[BSW00393] Parameters shall have a	Chapter 10.2

range	
[BSW00394] Specify the scope of the parameters	Chapter 10.2
[BSW00395] List the required parameters (per parameter)	Not applicable
[BSW00396] Configuration classes	Chapter 10.2
[BSW00397] Pre--compile--time parameters	Chapter 10.2
[BSW00398] Link--time parameters	Chapter 10.2
[BSW00399] Loadable Post--build time parameters	Chapter 10.2
[BSW00400] Selectable Post--build time parameters	CANSM163:
[BSW00438] Post Build Configuration Data Structure	CANSM061:
[BSW00402] Published information	Chapter 10.2.5
[BSW00375] Notification of wake-up reason	Not applicable (no wake up interrupt)
[BSW101] Initialization interface	CANSM023:
[BSW00416] Sequence of Initialization	chapter 7.7
[BSW00406] Check module initialization	CANSM032:
[BSW00437] NoInit--Area in RAM	Not applicable (not in scope of this spec)
[BSW168] Diagnostic interface	Not applicable (requirement on SWC-module)
[BSW00407] Function to read out published parameters	CANSM024:
[BSW00423] Usage of SW--C template to describe BSW modules with AUTOSAR Interfaces	Not applicable (not in scope of this spec)
[BSW00424] BSW main processing function task allocation	CANSM065:
[BSW00425] Trigger conditions for schedulable objects	CANSM065:
[BSW00426] Exclusive areas in BSW modules	Not applicable (not in scope of this spec)
[BSW00427] ISR description for BSW modules	Not applicable (not in scope of this spec)
[BSW00428] Execution order dependencies of main processing functions	Not applicable (not in scope of this spec)
[BSW00429] Restricted BSW OS functionality access	Not applicable (not in scope of this spec)
[BSW00431] The BSW Scheduler module implements task bodies	Not applicable (not in scope of this spec)
[BSW00432] Modules should have separate main processing functions for read/receive and write/transmit data path	Not applicable (not in scope of this spec)
[BSW00433] Calling of main processing	Not applicable

functions	(not in scope of this spec)
[BSW00434] The Schedule Module shall provide an API for exclusive areas	Not applicable (not in scope of this spec)
[BSW00336] Shutdown interface	Not applicable (no deinitialisation function)
[BSW00337] Classification of errors	Chapter 7.8
[BSW00338] Detection and Reporting of development errors	CANSM027: CANSM028:
[BSW00369] Do not return development error codes via API	CANSM079:
[BSW00339] Reporting of production relevant errors and exceptions	CANSM074:
[BSW00422] Pre--de--bouncing of production relevant error status	Chapter 7.7
[BSW00417] Reporting of Error Events by Non--Basic Software	Not applicable (not in scope of this spec)
[BSW00323] API parameter checking	CANSM071:
[BSW004] Version check	CANSM025:
[BSW00409] Header files for production code error IDs	Chapter 7.8
[BSW00385] List possible error notifications	chapter 7.8
[BSW00386] Configuration for detecting an error	CANSM027: CANSM071: CANSM028:
[BSW161] Microcontroller abstraction	Not applicable (not in scope of this spec)
[BSW162] ECU layout abstraction	Not applicable (not in scope of this spec)
[BSW005] No hard coded horizontal interfaces within MCAL	Not applicable (not in scope of this spec)
[BSW00415] User dependent include files	Chapter 5.9.2
[BSW164] Implementation of interrupt service routines	CANSM076:
[BSW00325] Runtime of interrupt service routines	CANSM237:
[BSW00326] Transition from ISRs to OS tasks	Not applicable (not in scope of this spec)
[BSW00342] Usage of source code and object code	Chapter 10.2
[BSW00343] Specification and configuration of time	Chapter 10.2
[BSW160] Human--readable configuration data	CANSM155:
[BSW007] HIS MISRA C	CANSM079:
[BSW00300] Module naming convention	CANSM079:
[BSW00413] Accessing instances of BSW modules	CANSM079:
[BSW00347] Naming separation of	Not applicable

different instances of BSW drivers	(not in scope of this spec)
[BSW00305] Self--defined data types naming convention	Chapter 8.2
[BSW00307] Global variables naming convention	CANSM079:
[BSW00310] API naming convention	Chapter 8.2.2
[BSW00373] Main processing function naming convention	Chapter 8.5.1
[BSW00327] Error values naming convention	Chapter 7.8
[BSW00335] Status values naming convention	Chapter 8.2
[BSW00350] Development error detection keyword	Chapter 7.9
[BSW00408] Configuration parameter naming convention	Chapter 10.2
[BSW00410] Compiler switches shall have defined values	Chapter 10.2
[BSW00411] Get version info keyword	CANSM180: , Chapter 10.2
[BSW00346] Basic set of module files	Chapter 5.9
[BSW158] Separation of configuration from implementation	Chapter 5.9
[BSW00314] Separation of interrupt frames and service routines	Not applicable (not in scope of this spec)
[BSW00370] Separation of callback interface from API	Chapter 5.9
[BSW00435] Header File Structure for the Basic Software Scheduler	CANSM022:
[BSW00436] Module Header File Structure for the Basic Software Memory Mapping	CANSM016:
[BSW00348] Standard type header	Chapter 5.9
[BSW00353] Platform specific type header	Not applicable (not in scope of this spec)
[BSW00361] Compiler specific language extension header	Not applicable (not in scope of this spec)
[BSW00301] Limit imported information	Chapter 5.9
[BSW00302] Limit exported information	Chapter 5.9
[BSW00328] Avoid duplication of code	CANSM079:
[BSW00312] Shared code shall be reentrant	CANSM079:
[BSW006] Platform independency	CANSM079:
[BSW00357] Standard API return type [	Chapter 8.2.2
[BSW00377] Module specific API return types	Not applicable (not used)
[BSW00304] AUTOSAR integer data types	CANSM079:
[BSW00355] Do not redefine AUTOSAR	CANSM079:

integer data types	
[BSW00378] AUTOSAR boolean type	CANSM079:
[BSW00306] Avoid direct use of compiler and platform specific keywords [	CANSM079:
[BSW00308] Definition of global data	Not applicable (not used)
[BSW00309] Global data with read--only constraint	Not applicable (not used)
[BSW00371] Do not pass function pointers via API	Chapter 8.2.2
[BSW00358] Return type of init() functions	CANSM023:
[BSW00414] Parameter of init function	CANSM023:
[BSW00376] Return type and parameters of main processing functions	CANSM065:
[BSW00359] Return type of callback functions	CANSM064:
[BSW00360] Parameters of callback functions	Not applicable (assignment between bus-off and impacted controller id is necessary, which is transferred as parameter)
[BSW00329] Avoidance of generic interfaces	CANSM079:
[BSW00330] Usage of macros / inline functions instead of functions	CANSM079:
[BSW00331] Separation of error and status values	Chapter 7.8, Chapter 8.2, CANSM079:
[BSW009] Module User Documentation	CANSM079:
[BSW00401] Documentation of multiple instances of configuration parameters	Chapter 10.2
[BSW172] Compatibility and documentation of scheduling strategy	CANSM079:
[BSW010] Memory resource documentation	CANSM079:
[BSW00333] Documentation of callback function context	CANSM064:
[BSW00374] Module vendor identification	Chapter 10.2.5
[BSW00379] Module identification	Chapter 10.2.5
[BSW003] Version identification	Chapter 10.2.5, CANSM024:
[BSW00318] Format of module version numbers	Chapter 10.2.5
[BSW00321] Enumeration of module version numbers	CANSM079:
[BSW00341] Microcontroller compatibility documentation	Not applicable (not in scope of this spec)
[BSW00334] Provision of XML file	CANSM079:

The CAN SRS ([7]) specifies the CAN specific parent requirements for the CanSM, which are listed in the following table:

Requirement	***Satisfied by***

[BSW01014] Network configuration abstraction	CANSM037: CANSM090: CANSM089: CANSM062: CANSM063: CANSM065: Chapter 10.2
BSW01142] Control flow abstraction of CAN networks	CANSM037: CANSM090: CANSM089: CANSM062: CANSM063: CANSM065: Chapter 10.2 chapter 7.3 chapter 7.4 chapter 7.5
[BSW01143] BusOff recovery time	**CANSM128 :** **CANSM129 :**
[BSW01144] Power-On Initialization	chapter 7.7.2
[BSW01145] Management of CAN devices	chapter 7.4 chapter 7.7
[BSW01146] Bus-off recovery and error handling	chapter 7.7 CANSM064: CANSM070:

The CanSM provides services to the ComM. Because of that, the CanSM also has to consider some requirements of the Mode Management SRS [9], which specifies the parent requirements for the ComM. These requirements are listed in following table:

Requirement	***Satisfied by***
[BSW09080] Physical channel independency	CANSM032:
[BSW09081] API for requesting communication	CANSM037: CANSM062:
[BSW09083] Support of different communication modes	CANSM037:
[BSW09084] API for querying the current communication mode	CANSM090: CANSM063:
[BSW09085] Indication of communication mode changes	CANSM089: and chapter 8.6.1

7 Functional specification

An ECU can have different communication networks. Each network has to be identified with a unique network handle. The ComM requests communication modes from the networks. It knows by its configuration, which handle is assigned to what kind of network. In case of CAN, it uses the CAN state manager, which is responsible for the control flow abstraction of CAN networks. The following sections describe this in detail.

7.1 Translation of network communication mode requests

CANSM037:

The CanSM shall provide to the ComM an API, which can be used by the ComM to request communication modes[1] of CAN networks (refer to chapter 8.3.3).

CANSM240:

Depending on the parameters handed over by this API, the CanSM shall execute a state transition of the related network mode state machine (refer to section 7.7).

CANSM241:

This transition shall translate the request into the respective APIs calls to control the assigned CAN peripherals (ref. to section 7.4), to control the CAN controller PDU modes (ref. to section 7.5), to notify communication mode changes (ref. to section 7.2) and to transfer basic software mode changes (refer to section 7.3).

7.2 Output of current network communication modes

The current communication mode of a network can be different to the requested mode. The CanSM has to provide the information of the current communication mode to the ComM by the two following kind of interfaces (see section 7.7):

CANSM090:

The CanSM shall provide an API, which can be polled by the ComM to get the current communication mode of a CAN network (see section 8.3.4).

CANSM089:

The CanSM shall use a call out function of the ComM to notify the change of communication modes(refer to chapter 8.6.1).

7.3 Basic Software Mode change notification

The CanSM module shall use the BswM module API BswM_CanSM_CurrentState to transfer the current CanSM BSW mode to the BswM module (ref. to chapter 5.6).

[1] please refer to [10] for a detailed description of the communication modes

7.4 Control of peripherals

7.4.1 CAN Transceivers

Each CAN Transceiver belongs to one CAN network.

CANSM033:

The assignment between network handles and transceivers shall be part of the CanSM configuration.

CANSM142:

The CanSM shall control the CAN transceivers depending on the state transitions of its network mode state machines (see section 7.7).

CANSM043:

The CanSM shall use the API of the CanIf for the control of the CAN transceiver modes (refer to chapter 8.6.1).

7.4.2 CAN Controllers

One or more[2] CAN controllers belong to a certain CAN network (handle).

CANSM039:

Depending on the network mode state machine, the CanSM shall control the CAN controller modes of each CAN network.

CANSM044:

The CanSM shall use the API of the CanIf to control the operating modes of the assigned CAN controllers (refer to chapter 8.6.1).

7.5 Control of PDU mode

CANSM083:

The CanSM shall control the PDU modes of the assigned CAN controllers depending on the network mode. The CanSM shall use the API of the CanIf module to request PDU modes (refer to chapter 8.6.1).

Note: With the PDU mode the CanIf decides, if the RX-PDUs and TX-PDUs assigned to a CAN controller are allowed to pass the CanIf or not.

7.6 Confirm PN availability to the CanNm

CANSM401:
If the CanIf module notifies PN availability for a configured CAN Transceiver to the CanSM module with the callback function CanSM_ConfirmPnAvailability (ref. to

[2] More CAN controllers can be assigned to a network to extend the amount of hardware object handles

CANSM400), then the CanSM module shall call the API `CanNm_ConfirmPnAvailability` (ref. to chapter 8.6.1) with the related CAN network as `channel` to confirm the PN availability to the CanNm module.

7.7 State machine for each CAN network

CANSM032:
The CanSM shall implement for each configured network handle one state machine, which is specified in the chapter 7.7.1, the chapter 7.7.2 and the chapter 7.7.3.

CANSM411:
All effects of the CanSM state machine (ref. to CANSM032), which depend on the communication request from the ComM module (ref. to CANSM062), shall be operated in the context of the CanSM main function (ref. to CANSM065).

7.7.1 State machine diagram: Top level

Referenced by: CANSM032

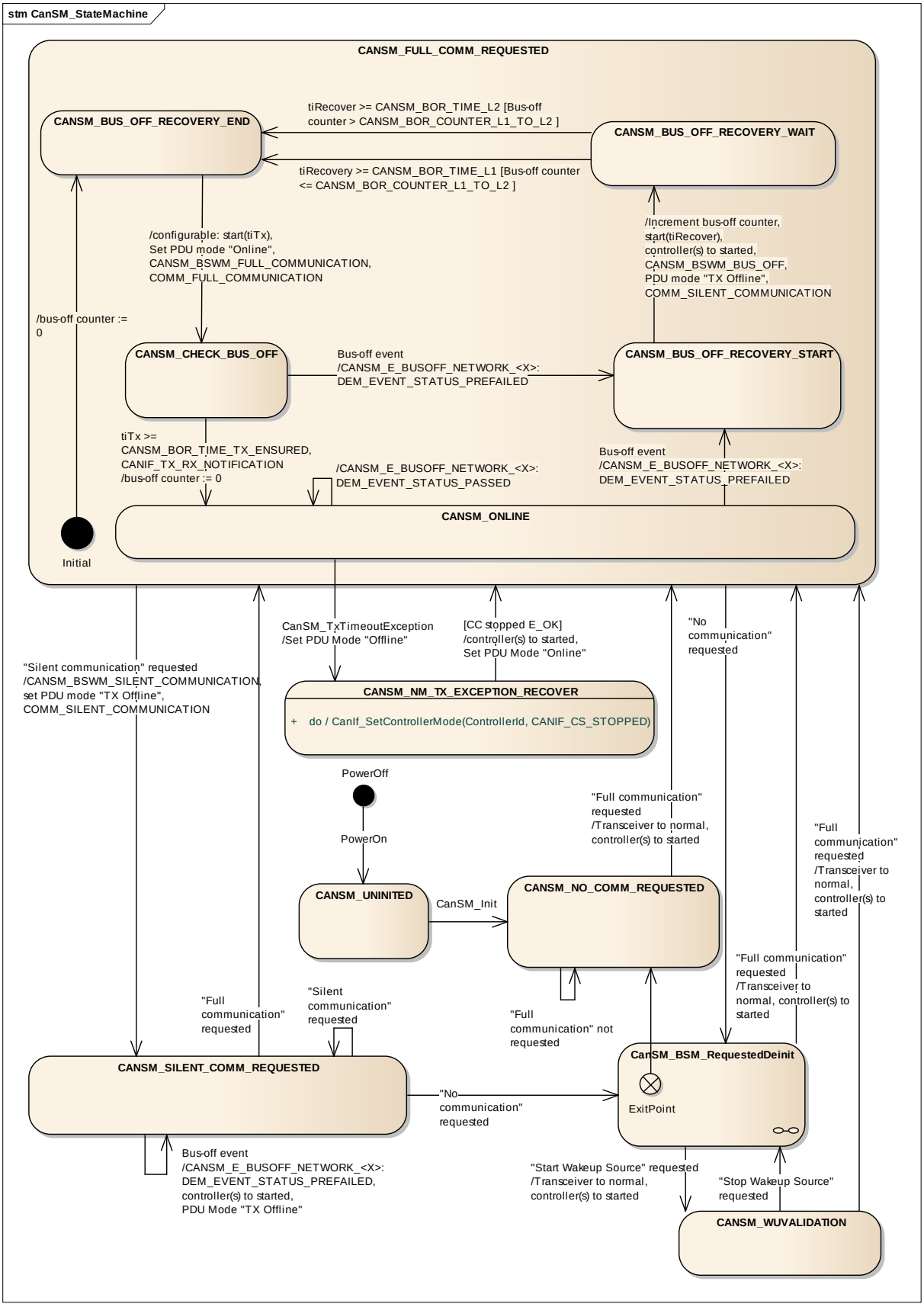

Figure 7-1: CanSM_StateMachine, state machine diagram for one CAN network

7.7.1.1 Sub state machine: CanSM_BSM_RequestedDeinit

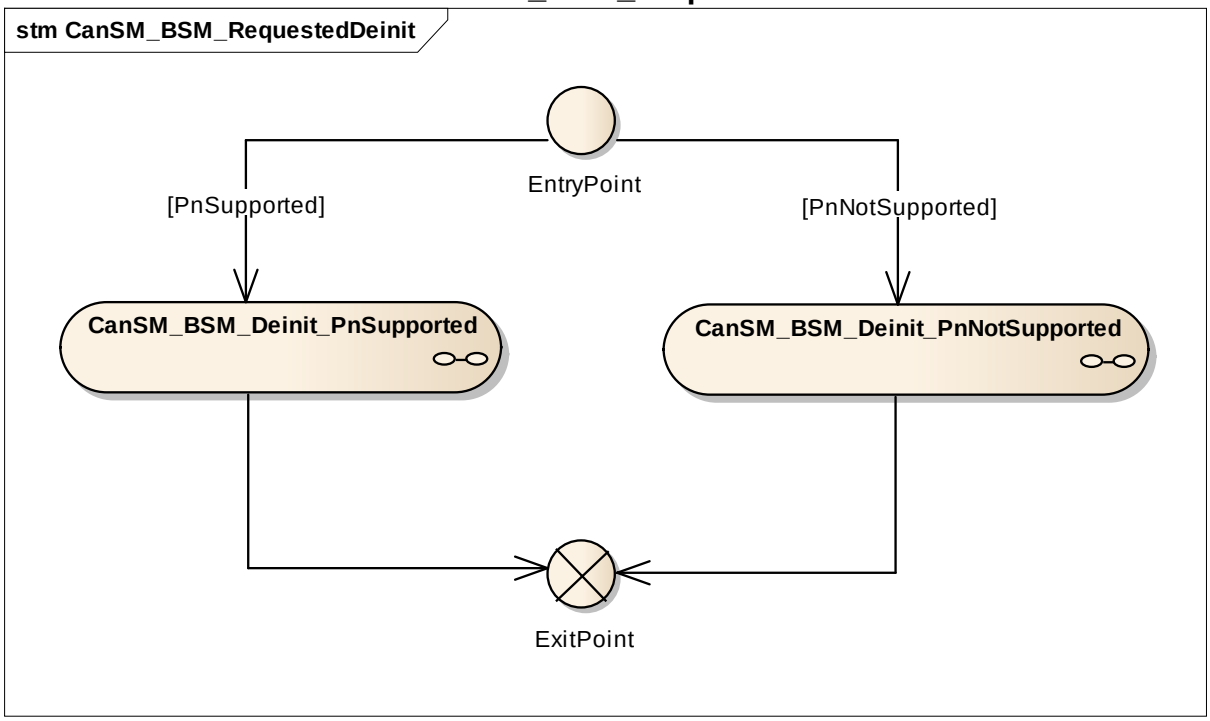

Figure 7-2: CanSM_BSM_RequestedDeinit, sub state machine of Figure 7-1

7.7.1.1.1 Sub state machine: CanSM_BSM_Deinit_PnNotSupported

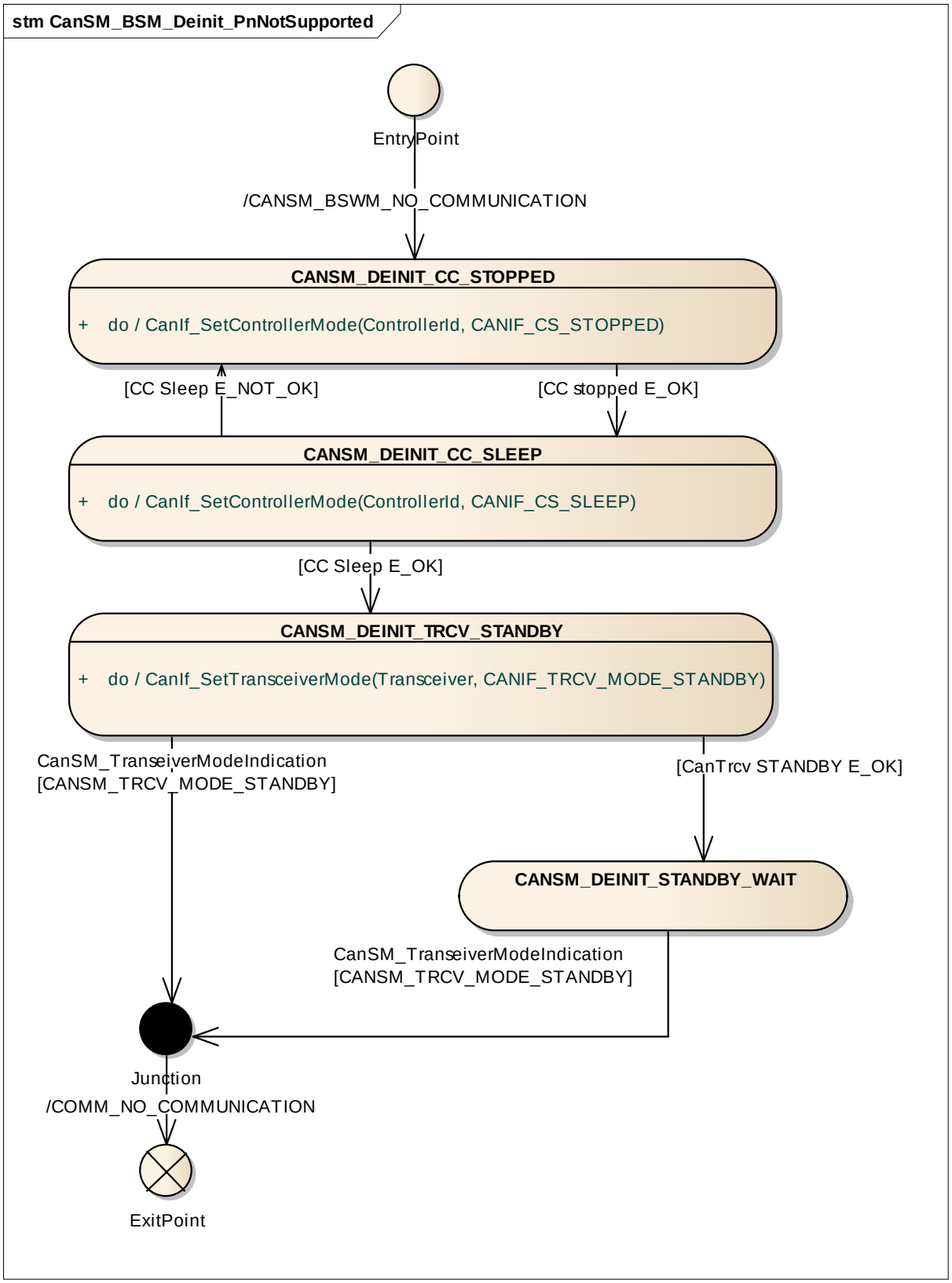

Figure 7-3: CanSM_BSM_Deinit_PnNotSupported, sub state machine of Figure 7-2

7.7.1.1.2 Sub state machine: CanSM_BSM_Deinit_PnSupported

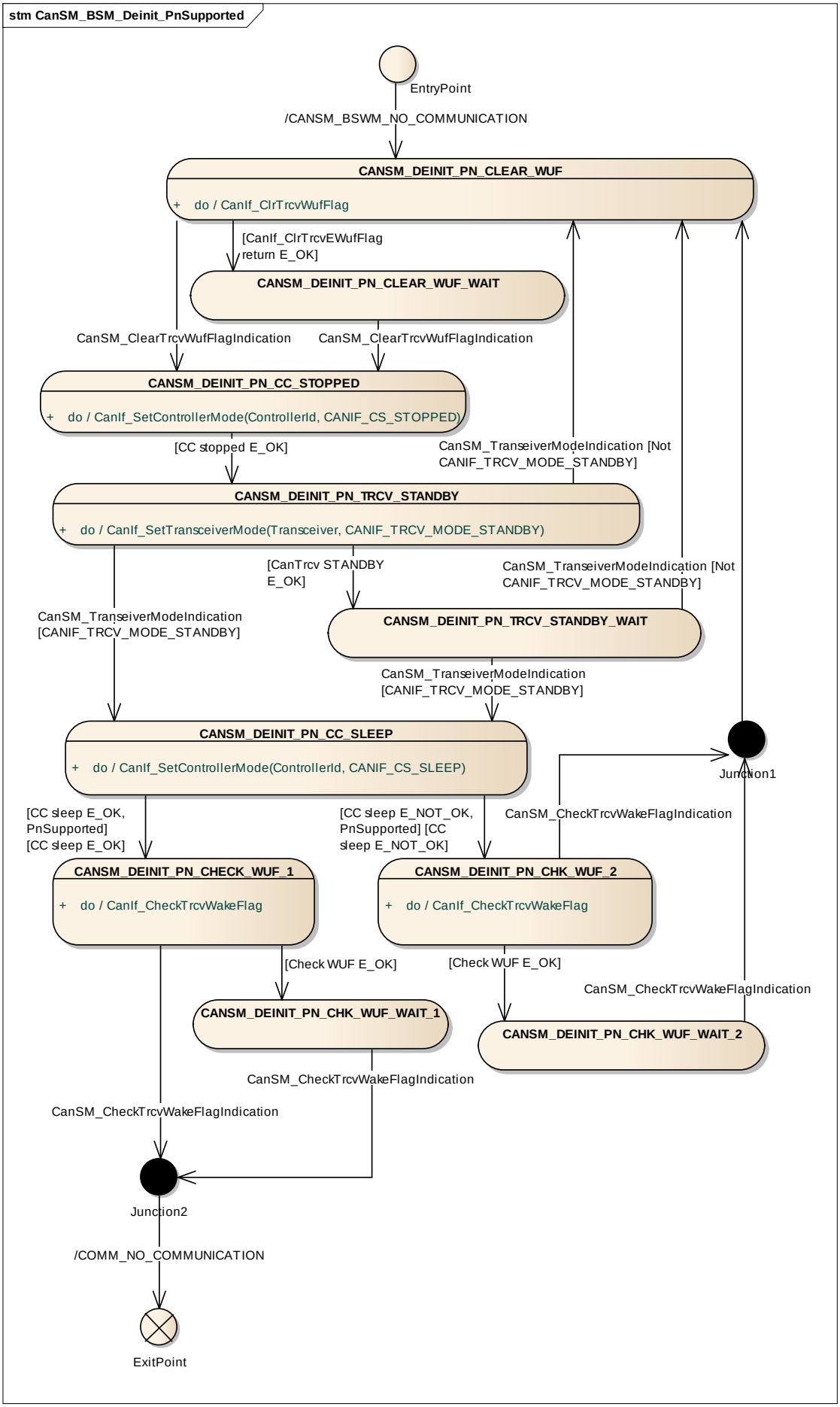
stm CanSM_BSM_Deinit_PnSupported
EntryPoint
/CANSM_BSWM_NO_COMMUNICATION
CANSM_DEINIT_PN_CLEAR_WUF
+ do / CanIf_ClrTrcvWufFlag
[CanIf_ClrTrcvEWufFlag return E_OK]
CANSM_DEINIT_PN_CLEAR_WUF_WAIT
CanSM_ClearTrcvWufFlagIndication
CanSM_ClearTrcvWufFlagIndication
CANSM_DEINIT_PN_CC_STOPPED
+ do / CanIf_SetControllerMode(ControllerId, CANIF_CS_STOPPED)
[CC stopped E_OK]
CanSM_TranseiverModeIndication [Not CANIF_TRCV_MODE_STANDBY]
CANSM_DEINIT_PN_TRCV_STANDBY
+ do / CanIf_SetTransceiverMode(Transceiver, CANIF_TRCV_MODE_STANDBY)
[CanTrcv STANDBY E_OK]
CanSM_TranseiverModeIndication [Not CANIF_TRCV_MODE_STANDBY]
CanSM_TranseiverModeIndication [CANIF_TRCV_MODE_STANDBY]
CANSM_DEINIT_PN_TRCV_STANDBY_WAIT
CanSM_TranseiverModeIndication [CANIF_TRCV_MODE_STANDBY]
CANSM_DEINIT_PN_CC_SLEEP
+ do / CanIf_SetControllerMode(ControllerId, CANIF_CS_SLEEP)
Junction1
[CC sleep E_OK, PnSupported] [CC sleep E_OK]
[CC sleep E_NOT_OK, PnSupported] [CC sleep E_NOT_OK]
CanSM_CheckTrcvWakeFlagIndication
CANSM_DEINIT_PN_CHECK_WUF_1
+ do / CanIf_CheckTrcvWakeFlag
CANSM_DEINIT_PN_CHK_WUF_2
+ do / CanIf_CheckTrcvWakeFlag
[Check WUF E_OK]
[Check WUF E_OK]
CanSM_CheckTrcvWakeFlagIndication
CANSM_DEINIT_PN_CHK_WUF_WAIT_1
CANSM_DEINIT_PN_CHK_WUF_WAIT_2
CanSM_CheckTrcvWakeFlagIndication
CanSM_CheckTrcvWakeFlagIndication
Junction2
/COMM_NO_COMMUNICATION
ExitPoint

Figure 7-4: CanSM_BSM_Deinit_PnSupported, sub state machine of Figure 7-2

7.7.2 State machine: Triggers and guards

7.7.2.1 Trigger CanSM_Init

CANSM350:
If the CanSM module is requested with the function `CanSM_Init`, this shall trigger the CanSM state machines (ref. to Figure 7-1) for all configured CAN Networks (ref. to CANSM126) with the trigger `CanSM_Init` and have the effect NO_COMMUNICATION (ref. to CANSM375).

7.7.2.2 Trigger "Full communication" requested

CANSM351:
After the CanSM module has accepted the request `CanSM_RequestComMode` (ref. to CANSM062) for a `NetworkHandle` (ref. to CANSM161) with the `ComM_Mode` `COMM_FULL_COMMUNICATION`, this shall trigger the CanSM state machine (ref. to Figure 7-1) for the requested `NetworkHandle` with the trigger `"Full communication" requested`.

7.7.2.3 Trigger "Silent communication" requested

CANSM352:
After the CanSM module has accepted the request `CanSM_RequestComMode` (ref. to CANSM062) for a `NetworkHandle` (ref. to CANSM161) with the `ComM_Mode` `COMM_SILENT_COMMUNICATION`, this shall trigger the CanSM state machine (ref. to Figure 7-1) for the requested `NetworkHandle` with the trigger `"Silent communication" requested`.

7.7.2.4 Trigger "No communication" requested

CANSM353:
After the CanSM module has accepted the request `CanSM_RequestComMode` (ref. to CANSM062) for a `NetworkHandle` (ref. to CANSM161) with the `ComM_Mode` `COMM_NO_COMMUNICATION`, this shall trigger the CanSM state machine (ref. to Figure 7-1) for the requested `NetworkHandle` with the trigger `"No communication" requested`.

7.7.2.5 Trigger "Start Wakeup Source" requested

[CANSM416:] If the API request `CanSM_StartWakeUpSource` (ref. to CANSM418) returns `E_OK` (ref. to CANSM423), it shall trigger the state machine machine (ref. to Figure 7-1) with `"Start Wakeup Source" requested`.

7.7.2.6 Trigger "Stop Wakeup Source" requested

[CANSM417:] If the API request `CanSM_StopWakeUpSource` (ref. to CANSM424) returns `E_OK` (ref. to CANSM429), it shall trigger the state machine (ref. to Figure 7-1) with `"Stop Wakeup Source" requested`.

7.7.2.7 Trigger CanSM_TxTimeoutException

CANSM354:
If the CanSM module is notified with the function `CanSM_TxTimeoutException` (ref. to CANSM348) for a configured `NetworkHandle` (ref. to CANSM161), then this shall trigger the CanSM state machine (ref. to Figure 7-1) for the requested `NetworkHandle` with the trigger `CanSM_TxTimeoutException`.

7.7.2.8 Trigger Bus-off event

CANSM355:
If the CanSM module is notified with the function `CanSM_ControllerBusOff` (ref. to CANSM064) for a configured `Controller`, then this shall trigger the CanSM state machine (ref. to Figure 7-1) with the trigger `Bus-off event` for the CAN network, which references the configured Controller (ref. to CANSM127) in its configuration.

7.7.2.9 Trigger CAN_TX_RX_NOTIFICATION

CANSM356:
To get the information, if the CanSM state machine for each CAN network is triggered with the trigger `CAN_TX_RX_NOTIFICATION` (ref. to Figure 7-1), the CanSM module shall check if `CANSM_BOR_TX_CONFIRMATION_POLLING` is enabled (ref. to CANSM339), request in this case the API function `CanIf_GetTxConfirmationState` and check, if it returns `CANIF_TX_RX_NOTIFICATION` for all configured CAN controllers of the CAN network (ref. to CANSM127).

7.7.2.10 Trigger tiTx >= CANSM_BOR_TX_ENSURED

CANSM357:
If the parameter `CANNSM_BOR_TX_CONFIRMATION_POLLING` is disabled (ref. to CANSM339) and the time duration since the effect `start(tiTx)` (ref. to CANSM358) is greater or equal the configuration parameter `CANSM_BOR_TIME_TX_ENSURED` (ref. to CANSM130), then this shall trigger the CanSM state machine (ref. to Figure 7-1) for the related CAN network with the trigger `tiTx >= CANSM_BOR_TX_ENSURED`.

Hints for configuration of CanSMBorTimeTxEnsured

The configured time `CanSMBorTimeTxEnsured` must be large enough to ensure that new PDUs are transmitted. This depends on the SWCs, which initiate the transmission of signals and the cycle times configured in COM for signals with cyclic transmission mode.

7.7.2.11 Trigger tiRecover >= CANSM_BOR_TIME_L1

CANSM359:
If the time duration since the effect `start(tiRecover)` (ref. to CANSM360) is greater or equal the configuration parameter `CANSM_BOR_TIME_L1` (ref. to CANSM128), then this shall trigger the CanSM state machine (ref. to Figure 7-1) for the related CAN network with the trigger `tiRecover >= CANSM_BOR_TIME_L1`.

7.7.2.12 Trigger tiRecover >= CANSM_BOR_TIME_L2

CANSM361:
If the time duration since the effect `start(tiRecover)` (ref. to CANSM360) is greater or equal the configuration parameter `CANSM_BOR_TIME_L2` (ref. to CANSM129), then this shall trigger the CanSM state machine (ref. to Figure 7-1) for the related CAN network with the trigger `tiRecover >= CANSM_BOR_TIME_L2`.

7.7.2.13 Trigger CanSM_ClearTrcvWufFlagIndication

CANSM362:
The function `CanSM_ClearTrcvWufFlagIndication` (ref. to CANSM363) shall trigger the CanSM state machine (ref. to Figure 7-4) for the related CAN network with the trigger `CanSM_ClearTrcvWufFlagIndication`.

7.7.2.14 Trigger under guard: CanSM_TransceiverModeIndication [CANIF_TRCV_MODE_STANDBY]

CANSM364:
The function call `CanSM_TransceiverModeIndication` (ref. to CANSM365) with the parameter `TransceiverMode` equal to `CANIF_TRCV_MODE_STANDBY` and the parameter `Transceiver` equal to the configured CAN Transceiver (ref. to CANSM137), shall trigger the CanSM state machine (ref. to Figure 7-3 and Figure 7-4) for the CAN network with the trigger under guard: `CanSM_TransceiverModeIndication [CANIF_TRCV_MODE_STANDBY]`.

7.7.2.15 Trigger under guard: CanSM_TransceiverModeIndication [not CANIF_TRCV_MODE_STANDBY]

CANSM366:
The function `CanSM_TransceiverModeIndication` (ref. to CANSM365) with the parameter `TransceiverMode` not equal to `CANIF_TRCV_MODE_STANDBY` and the parameter `Transceiver` equal to the configured CAN Transceiver (ref. to CANSM137), shall trigger the CanSM state machine (ref. to Figure 7-3 and Figure 7-4) for the CAN network with the trigger under guard: `CanSM_TransceiverModeIndication [not CANIF_TRCV_MODE_STANDBY]`.

7.7.2.16 Trigger CanSM_CheckTrcvWakeFlagIndication

CANSM368:
The function `CanSM_CheckTrcvWakeFlagIndication` (ref. to CANSM367) shall trigger the CanSM state machine (ref. to Figure 7-4) for the related CAN network with the trigger `CanSM_CheckTrcvWakeFlagIndication`.

7.7.2.17 Guarding condition Bus-Off counter > CANSM_BOR_COUNTER_L1_TO_L2

CANSM384:
The guarding condition `Bus-Off counter > CANSM_BOR_COUNTER_L1_TO_L2` of the CanSM state machine (ref. to Figure 7-1) for the related CAN network shall check if the bus-off counter (ref. to CANSM382, CANSM383) is greater than the configuration parameter `CANSM_BOR_COUNTER_L1_TO_L2` (ref. to CANSM131).

7.7.2.18 Guarding condition Bus-Off counter <= CANSM_BOR_COUNTER_L1_TO_L2

CANSM385:
The guarding condition `Bus-Off counter <= CANSM_BOR_COUNTER_L1_TO_L2` of the CanSM state machine (ref. to Figure 7-1) for the related CAN network shall check if the bus-off counter (ref. to CANSM382, CANSM383) is lower than or equal to the configuration parameter `CANSM_BOR_COUNTER_L1_TO_L2` (ref. to CANSM131).

7.7.2.19 Guarding condition PnSupported

CANSM387:
The guarding condition `PnSupported` of the CanSM state machine (ref. to Figure 7-2) shall check if the configuration parameter `CanSMTransceiverPnSupport` (ref. to CANSM344) is `TRUE`.

7.7.2.20 Guarding condition Not PnSupported

CANSM388:
The guarding condition `Not PnSupported` of the CanSM state machine (ref. to Figure 7-2) shall check if the configuration parameter `CanSMTransceiverPnSupport` (ref. to CANSM344) is `FALSE`.

7.7.2.21 Guarding condition CanIf_ClrTrcvEWufFlag return E_OK

CANSM389:
The guarding condition `CanIf_ClrTrcvEWufFlag return E_OK` of the CanSM state machine (ref. to Figure 7-4) shall check if the API call `CanIf_ClrTrcvWufFlag` (ref. to CANSM390) has returned with `E_OK`.

7.7.2.22 Guarding condition CC stopped E_OK

CANSM392:
The guarding condition `CC stopped E_OK` of the CanSM state machine (ref. to Figure 7-1, Figure 7-3 and Figure 7-4) shall check if the API call `CanIf_SetControllerMode` with the `ControllerMode` equal to `CANIF_CS_STOPPED` (ref. to CANSM391) has returned with `E_OK` for all configured CAN controllers of the CAN network.

7.7.2.23 Guarding condition CC sleep E_OK

CANSM397:
The guarding condition `CC sleep E_OK` of the CanSM state machine (ref. to Figure 7-3 and Figure 7-4) shall check if the API call `CanIf_SetControllerMode` with the `ControllerMode` equal to `CANIF_CS_SLEEP` (ref. to CANSM395) has returned with `E_OK` for all configured CAN controllers of the CAN network.

7.7.2.24 Guarding condition CC sleep E_NOT_OK

CANSM398:
The guarding condition `CC sleep E_NOT_OK` of the CanSM state machine (ref. to Figure 7-3 and Figure 7-4) shall check if the API call `CanIf_SetControllerMode` with the `ControllerMode` equal to `CANIF_CS_SLEEP` (ref. to CANSM395) has returned with `E_NOT_OK` for any of the configured CAN controllers of the CAN network.

7.7.2.25 Guarding condition CanTrcv STANDBY E_OK

CANSM394:
The guarding condition `CanTrcv STANDBY E_OK` of the CanSM state machine (ref. to Figure 7-3 and Figure 7-4) shall check if the API call `CanIf_SetTransceiverMode` with the `TransceiverMode` equal to `CAN_TRCV_MODE_STANDBY` (ref. to CANSM393) has returned with `E_OK`.

7.7.2.26 Guarding condition check WUF E_OK

CANSM399:
The guarding condition `check WUF E_OK` of the CanSM state machine (ref. to Figure 7-4) shall check if the API call `CanIf_CheckTrcvWakeFlag` (ref. to CANSM396) has returned with `E_OK`.

7.7.3 State machine: Effects and state operations

7.7.3.1 Effect start(tiTx)

CANSM358:
If the parameter `CANNSM_BOR_TX_CONFIRMATION_POLLING` is disabled (ref. to CANSM339), the effect `start(tiTx)` of the CanSM state machine (ref. to Figure 7-1) shall also be active and start an internal SW timer for the related CAN network, which is necessary for the related trigger `tiTx >= CANSM_BOR_TX_ENSURED` (ref. to section 7.7.2.10).

7.7.3.2 Effect start(tiRecover)

CANSM360:
The effect `start(tiRecover)` of the CanSM state machine (ref. to Figure 7-1) shall start an internal SW timer for the related CAN network, which is necessary for the related triggers `tiRecover >= CANSM_BOR_TIME_L1` (ref. to section 7.7.2.11) and `tiRecover >= CANSM_BOR_TIME_L1` (ref. to section 7.7.2.12).

7.7.3.3 Effect Transceiver to normal

CANSM369:
The effect Transceiver to normal of the CanSM state machine (ref. to Figure 7-1) shall request the API `CanIf_SetTransceiverMode` (ref. to chapter 8.6.1) with the Transceiver parameter equal to the configured Transceiver of the related CAN network in a repeated way until it returns `E_OK` and is notified with the call of the notification function `CanSM_TransceiverModeIndication` (ref. to CANSM365) with the TransceiverMode `CANIF_TRCV_MODE_NORMAL`.

7.7.3.4 Effect Controller(s) to started

CANSM370:
The effect Controller(s) to started of the CanSM state machine (ref. to Figure 7-1) shall request the API `CanIf_SetControllerMode` (ref. to chapter 8.6.1) for all configured CAN controllers of the CAN network (ref. to CANSM127) with the respective ControllerId parameters in a repeated way until all API calls have returned `E_OK`.

7.7.3.5 Effect CANSM_BSWM_SILENT_COMMUNICATION

CANSM371:
The effect `CANSM_BSWM_SILENT_COMMUNICATION` of the CanSM state machine (ref. to Figure 7-1) shall request the API `BswM_CanSM_CurrentState` (ref. to chapter 8.6.1) with the related `Network` (ref. to CANSM161) and `CurrentState := CANSM_BSWM_SILENT_COMMUNICATION`.

7.7.3.6 Effect CANSM_BSWM_FULL_COMMUNICATION

CANSM372:
The effect `CANSM_BSWM_FULL_COMMUNICATION` of the CanSM state machine (ref. to Figure 7-1) shall request the API `BswM_CanSM_CurrentState` (ref. to chapter 8.6.1) with the related `Network` (ref. to CANSM161) and `CurrentState := CANSM_BSWM_FULL_COMMUNICATION`.

7.7.3.7 Effect CANSM_BSWM_BUS_OFF

CANSM373:
The effect `CANSM_BSWM_BUS_OFF` of the CanSM state machine (ref. to Figure 7-1) shall request the API `BswM_CanSM_CurrentState` (ref. to chapter 8.6.1) with the related `Network` (ref. to CANSM161) and `CurrentState := CANSM_BSWM_BUS_OFF`.

7.7.3.8 Effect CANSM_BSWM_NO_COMMUNICATION

CANSM374:
The effect `CANSM_BSWM_NO_COMMUNICATION` of the CanSM state machine (ref. to Figure 7-3 and Figure 7-4) shall request the API `BswM_CanSM_CurrentState` (ref. to chapter 8.6.1) with the related `Network` (ref. to CANSM161) and `CurrentState := CANSM_BSWM_NO_COMMUNICATION`.

7.7.3.9 Effect COMM_NO_COMMUNICATION

CANSM375:
The effect `COMM_NO_COMMUNICATION` of the CanSM state machine (ref. to Figure 7-3 and Figure 7-4) shall request the API `ComM_BusSM_ModeIndication` (ref. to chapter 8.6.1) with the parameters `Channel` equal to the related CAN network (ref. to CANSM161) and `ComMode` equal to `COMM_NO_COMMUNICATION`.

7.7.3.10 Effect COMM_SILENT_COMMUNICATION

CANSM376:
The effect `COMM_SILENT_COMMUNICATION` of the CanSM state machine (ref. to Figure 7-1) shall request the API `ComM_BusSM_ModeIndication` (ref. to chapter 8.6.1) with the parameters `Channel` equal to the related CAN network (ref. to CANSM161) and `ComMode` equal to `COMM_SILENT_COMMUNICATION`.

7.7.3.11 Effect COMM_FULL_COMMUNICATION

CANSM377:
The effect `COMM_FULL_COMMUNICATION` of the CanSM state machine (ref. to Figure 7-1) shall request the API `ComM_BusSM_ModeIndication` (ref. to chapter 8.6.1) with the parameters `Channel` equal to the related CAN network (ref. to CANSM161) and `ComMode` equal to `COMM_FULL_COMMUNICATION`.

7.7.3.12 Effect Set PDU mode “Online”

CANSM378:
The effect `Set PDU mode "Online"` of the CanSM state machine (ref. to Figure 7-1) shall request for all configured CAN Controllers (ref. to CANSM127) of the related CAN network the API `CanIf_SetPduMode` (ref. to chapter 8.6.1) with the corresponding `Controller` parameter and the parameter `PduModeRequest` equal to `CANIF_SET_ONLINE`, if the configuration parameter `CanSMTransceiverPnSupport` (ref. to CANSM344) is `FALSE` for the related CAN Network.

CANSM413:
The effect `Set PDU mode "Online"` of the CanSM state machine (ref. to Figure 7-1) shall request for all configured CAN Controllers (ref. to CANSM127) of the related CAN network the API `CanIf_SetPduMode` (ref. to chapter 8.6.1) with the corresponding `Controller` parameter and the parameter `PduModeRequest` equal to `CANIF_SET_ONLINE_WAKF`, if the configuration parameter `CanSMTransceiverPnSupport` (ref. to CANSM344) is `TRUE` for the related CAN Network and if the bus off counter (ref. to CANSM382, CANSM383) is equal 0.

CANSM414:
The effect `Set PDU mode "Online"` of the CanSM state machine (ref. to Figure 7-1) shall request for all configured CAN Controllers (ref. to CANSM127) of the related CAN network the API `CanIf_SetPduMode` (ref. to chapter 8.6.1) with the corresponding `Controller` parameter and the parameter `PduModeRequest` equal to `CANIF_SET_ONLINE`, if the configuration parameter `CanSMTransceiverPnSupport` (ref. to CANSM344) is `TRUE` for the related CAN Network and if the bus off counter (ref. to CANSM382, CANSM383) is greater than 0.

7.7.3.13 Effect Set PDU mode “TX Offline”

CANSM379:
The effect `Set PDU mode "TX Offline"` of the CanSM state machine (ref. to Figure 7-1) shall request for all configured CAN Controllers (ref. to CANSM127) of the related CAN network the API `CanIf_SetPduMode` (ref. to chapter 8.6.1) with the corresponding `Controller` parameter and the parameter `PduModeRequest` equal to `CANIF_SET_TX_OFFLINE`.

7.7.3.14 Effect CANSM_E_BUSOFF_NETWORK_<X>: DEM_EVENT_STATUS_PASSED

CANSM380:
The effect `CANSM_E_BUSOFF_NETWORK_<X>: DEM_EVENT_STATUS_PASSED` of the CanSM state machine (ref. to Figure 7-1) shall request the API `Dem_ReportErrorStatus` (ref. to chapter 8.6.1) with the parameters `EventStatus := DEM_EVENT_STATUS_PASSED` and the corresponding CAN Bus-Off `EventId` for the related CAN network (ref. to CANSM070).

7.7.3.15 Effect CANSM_E_BUSOFF_NETWORK_<X>: DEM_EVENT_STATUS_PREFAILED

CANSM381:
The effect `CANSM_E_BUSOFF_NETWORK_<X>: DEM_EVENT_STATUS_PREFAILED` of the CanSM state machine (ref. to Figure 7-1) shall request the API `Dem_ReportErrorStatus` (ref. to chapter 8.6.1) with the parameters `EventStatus := DEM_EVENT_STATUS_PREFAILED` and the corresponding CAN Bus-Off `EventId` (ref. to CANSM070).for the related CAN network

7.7.3.16 Effect bus-off counter := 0

CANSM382:
The effect `bus-off counter := 0` of the CanSM state machine (ref. to Figure 7-1) shall set an internal counter variable, which has to be present for each configure CAN network (ref. to CANSM126) to 0.

7.7.3.17 Effect increment bus-off counter

CANSM383:
The effect `increment bus-off counter` of the CanSM state machine (ref. to Figure 7-1) shall increment an internal counter variable with 1, which has to be present for each configured CAN network (ref. to CANSM126).

7.7.3.18 Do action in state CANSM_DEINIT_PN_CLEAR_WUF

CANSM390:
In the state `CANSM_DEINIT_CLEAR_WUF`, the CanSM state machine (ref. to Figure 7-4) shall repeat the API request `CanIf_ClrTrcvWufFlag` with the Transceiver (ref. to CANSM137), which is configured for the related CAN network.

7.7.3.19 Do action in state CANSM_DEINIT_CC_STOPPED and CANSM_DEINIT_PN_CC_STOPPED

CANSM391:
In the states `CANSM_NM_TX_ECEPTION_RECOVER`, `CANSM_DEINIT_CC_STOPPED` and `CANSM_DEINIT_PN_CC_STOPPED`, the CanSM state machine (ref. to Figure 7-1, Figure 7-3 and Figure 7-4) shall repeat for all configured CAN controllers of the CAN network the API request `CanIf_SetControllerMode` with `ControllerMode` equal to `CANIF_CS_STOPPED`.

7.7.3.20 Do action in state CANSM_DEINIT_TRCV_STANDBY and CANSM_DEINIT_PN_TRCV_STANDBY

CANSM393:
In the state `CANSM_DEINIT_TRCV_STANDBY` and in the state `CANSM_DEINIT_PN_TRCV_STANDBY`, the CanSM state machine (ref. to Figure 7-3 and Figure 7-4) shall repeat for the configured CAN transceiver of the CAN network the API request `CanIf_SetTransceiverMode` with `TransceiverMode` equal to `CANIF_TRCV_MODE_STANDBY`.

7.7.3.21 Do action in state CANSM_DEINIT_CC_SLEEP and CANSM_DEINIT_PN_CC_SLEEP

CANSM395:

In the state `CANSM_DEINIT_CC_SLEEP` and in the state `CANSM_DEINIT_PN_CC_SLEEP`, the CanSM state machine (ref. to Figure 7-3 and Figure 7-4) shall repeat for all configured CAN controllers of the CAN network the API request `CanIf_SetControllerMode` with `ControllerMode` equal to `CANIF_CS_SLEEP`.

7.7.3.22 Do action in states CANSM_DEINIT_PN_CHECK_WUF_1 and ~WUF_2

CANSM396:

In the states `CANSM_DEINIT_CHECK_WUF_1` and `CANSM_DEINIT_CHECK_WUF_2` the CanSM state machine (ref. to Figure 7-4) shall repeat the API request `CanIf_CheckTrcvWakeFlag` with `Transceiver` equal to the configured CAN transceiver of the CAN network.

7.8 Error classification

This chapter lists and classifies all errors that can be detected by this software module. Each error is classified to relevance (development / production) and the related error code (unique label for the error). For development errors this table also specifies the unique values, which correspond to the error codes.

Values for production code Event Ids are assigned externally by the configuration of the DEM. They are published in the file Dem_IntErrId.h and included via Dem.h.

CANSM069:

Development error values shall be of type uint8.

CANSM070:

The CanSM shall report for each configured CAN network one specific bus-off error event with following naming convention for the production errors events: `CANSM_E_BUSOFF_NETWORK_<X>`, where <X> is the respective network handle.

Example: The assigned bus-off error for the network handle 5 is `CANSM_E_BUSOFF_NETWORK_5`.

Type or error	***Relevance***	***Related error code***	***Value [hex]***
API service used without module initialization	Development	`CANSM_E_UNINIT`	`0x01`
API service called with wrong pointer	Development	`CANSM_E_PARAM_POINTER`	`0x02`
API service called with wrong parameter	Development	`CANSM_E_INVALID_NETWORK_HANDLE`	`0x03`
API service called with wrong parameter	Development	`CANSM_E_PARAM_CONTROLLER`	`0x04`
API service called with wrong parameter	Development	`CANSM_E_PARAM_TRANSCEIVER`	`0x05`
The bus-off recovery state machine of a CAN	Production	Refer to `CANSM070`:	`Assigned by DEM`

network has detected a certain amount of sequential bus-offs without successful recovery			

7.9 Error detection

CANSM027:

The detection of development errors shall be configurable (*ON* / *OFF*) at pre-compile time.

The switch `CanSMDevErrorDetect` (see chapter 10) shall activate or deactivate the detection of all development errors.

CANSM071:

If the `CanSMDevErrorDetect` switch is enabled, the API parameter checking shall be enabled. The detailed description of the detected errors can be found in chapter 7.8 and chapter 8.

CANSM072:

The detection of production code errors cannot be switched off.

Remark: The detailed description of the detection production error “bus-off” can be found in section 7.7.3.15.

7.10 Error notification

CANSM028:

Detected development errors shall be reported to the `Det_ReportError` service of the Development Error Tracer (DET) if the pre-processor switch `CanSMDevErrorDetect` is set “on” (see chapter 10).

CANSM074:

Production errors shall be reported to the Diagnostic Event Manager with the API Dem_ReportErrorStatus.

Remark: For the configuration of the DEM it has to be considered, that the bus-off events are already debounced by the CanSM itself internally. The detailed description of the notification of the production error “bus-off” can be found in the section 7.7.3.15.

7.11 Non-functional design rules

CANSM025:

The CanSM files shall check the consistency between the header, C and configuration files during compilation according to BSW004. This is to guarantee the consistency of the files and the code generator to the same release.

CANSM077:

The CanSM shall not use operating system timers and resources directly.

CANSM076:
The CanSM shall not implement interrupt service routines.

CANSM078:

The CanSM shall be implemented in a way, that it can be either delivered as source code or object code into the AUTOSAR stack.

CANSM079:

The CanSM shall be implemented according the AUTOSAR Design Requirements (For details refer to Requirements on Basic Software Modules [3]).

CANSM237:

The run time of the CanSM functions, which can be called in interrupt context, should be kept short.

8 API specification

8.1 Imported types

8.1.1 Standard types

The CanSM includes the following listed types of the file Std_Types.h (refer to [5]):

- Std_ReturnType
- Std_VersionInfoType
- uint8

8.1.2 Common COM-Stack specific types

The CanSM includes the following listed types of the file ComStackTypes.h (refer to [6]):

- NetworkHandleType

8.1.3 ComM types

The CanSM includes the following listed types of the ComM module (refer to[10]):

- ComM_ModeType

8.1.4 CanIf types

The CanSM includes the following listed types of the CanIf module (refer to [13]):

- CanIf_ControllerModeType
- CanIf_ChannelSetModeType
- CanIf_TransceiverModeType

8.1.5 DEM types

The CanSM includes the following listed types of the DEM module (refer to [12]):

- Dem_EventIdType
- Dem_EventStatusType

8.2 Type definitions

The following sections specify the type definitions of the CanSM.

8.2.1 CanSM_ConfigType

CANSM061:

Name:	`CanSM_ConfigType`	
Type:	`Structure`	
Range:	`--`	--
Description:	This type defines a data structure for the post build parameters of the CanSM. At initialization the CanSM gets a pointer to a structure of this type to get access to its configuration data, which is necessary for initialization.	

8.2.2 CanSM_BswMCurrentStateType

CANSM347:

Name:	`CanSM_BswMCurrentStateType`	
Type:	`Enumeration`	
Range:	`CANSM_BSWM_NO_COMMUNICATION`	--
	`CANSM_BSWM_SILENT_COMMUNICATION`	--
	`CANSM_BSWM_FULL_COMMUNICATION`	--
	`CANSM_BSWM_BUS_OFF`	--
Description:	Can specific communication modes / states notified to the BswM module	

8.3 Function definitions

The following sections specify the provided API functions of the CanSM.

8.3.1 CanSM_Init

CANSM023:

Service name:	CanSM_Init
Syntax:	`void CanSM_Init( const CanSM_ConfigType* ConfigPtr )`
Service ID[hex]:	0x00
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	ConfigPtr: Pointer to init structure for the post build parameters of the CanSM
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This service initializes the CanSM module

CANSM198:

Only for configuration variant 1 and 2: Instead of the prototype specified in CANSM023: the CanSM shall declare following prototype for the API `CanSM_Init` and use a void parameter instead of the `ConfigPtr`:

```
void CanSM_Init(void)
```

CANSM179:

Only for configuration variant 3: The function `CanSM_Init` shall report the development error `CANSM_E_PARAM_POINTER` to the DET, if the user of this function hands over a NULL-pointer as `ConfigPtr`.

8.3.2 CanSM_GetVersionInfo

CANSM024:

Service name:	CanSM_GetVersionInfo	
Syntax:	`void CanSM_GetVersionInfo( Std_VersionInfoType* VersionInfo )`	
Service ID[hex]:	0x01	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	VersionInfo	--
Return value:	None	
Description:	This service puts out the version information of this module (module ID, vendor ID, vendor specific version numbers related to BSW00407)	

Implementation hint: If source code for caller and callee of this function is available this function should be realized as a macro. The macro should be defined in the modules header file.

CANSM180:

This function `CanSM_GetVersionInfo` shall be pre compile time configurable `On/Off` by the configuration parameter: `CANSM_VERSION_INFO_API`.

8.3.3 CanSM_RequestComMode

CANSM062:

Service name:	CanSM_RequestComMode	
Syntax:	`Std_ReturnType CanSM_RequestComMode( NetworkHandleType NetworkHandle, ComM_ModeType ComM_Mode )`	
Service ID[hex]:	0x02	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant	
Parameters (in):	NetworkHandle	Handle of destinated communication network for request
	ComM_Mode	Requested communication mode

Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Service accepted E_NOT_OK: Service denied
Description:	This service shall change the communication mode of a CAN network to the requested one.	

Remark: The function reentrancy is limited to different network handles. Reentrancy for the same network is not to be regarded here.

CANSM181:

The function `CanSM_RequestComMode` checks the network handle of the request. It only accepts the request, if the network handle of the request is a handle contained in the CanSM configuration (configuration parameter `CanSMNetworkHandle`). If it is not contained in the configuration, the function denies the request.

CANSM183:

The function `CanSM_RequestComMode` shall report `CANSM_E_INVALID_NETWORK_HANDLE` to the DET, if it does not accept the network handle of the request.

CANSM182:

If the function `CanSM_RequestComMode` accepts the request, it shall store the requested communication mode for the network handle and shall execute the corresponding network mode state machine and the bus-off recovery state machine.

CANSM184:

The function `CanSM_RequestComMode` shall report `CANSM_E_UNINIT` to the DET, if the CanSM is not initialized yet.

8.3.4 CanSM_GetCurrentComMode

CANSM063:

Service name:	CanSM_GetCurrentComMode	
Syntax:	`Std_ReturnType CanSM_GetCurrentComMode( NetworkHandleType NetworkHandle, ComM_ModeType* ComM_ModePtr )`	
Service ID[hex]:	0x03	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	NetworkHandle	Network handle, whose current communication mode shall be put out
Parameters (inout):	None	
Parameters (out):	ComM_ModePtr	Pointer, where to put out the current communication mode
Return value:	Std_ReturnType	E_OK: Service accepted E_NOT_OK: Service denied
Description:	This service shall put out the current communication mode of a CAN network.	

CANSM185:

The function `CanSM_GetCurrentComMode` checks the network handle of the service request. It only accepts the service, if the network handle of the request is a handle contained in the CanSM configuration (configuration parameter `CanSMNetworkHandle`). If it is not contained in the configuration, the function denies the request.

CANSM187:

The function `CanSM_GetCurrentComMode` shall report `CANSM_E_INVALID_NETWORK_HANDLE` to the DET, if it does not accept the network handle of the request.

CANSM186:

The function `CanSM_GetCurrentComMode` puts out the current communication mode for the network handle to the designated pointer of type `ComM_ModeType`, if it accepts the request.

Remark: Because the CAN hardware needs a certain time to proceed with the request and there is currently no notification mechanism specified, the real hardware mode and the mode notified by the CanSM might be different until the hardware is ready.

CANSM188:

The function `CanSM_GetCurrentComMode` shall report `CANSM_E_UNINIT` to the DET, if the CanSM is not initialized yet.

8.3.5 CanSM_StartWakeupSource

[CANSM418:]

Service name:	CanSM_StartWakeupSource	
Syntax:	`Std_ReturnType CanSM_StartWakeupSource( NetworkHandleType network )`	
Service ID[hex]:	0x11	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	network	Affected CAN network
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Request accepted E_NOT_OK: Request denied
Description:	This function shall be called by EcuM when a wakeup source shall be started.	

[CANSM419:]

The API function `CanSM_StartWakeupSource` shall return `E_NOT_OK`, if the CanSM module is not initialized yet with `CanSM_Init` (ref. to CANSM023).

[CANSM420:]
The function `CanSM_StartWakeupSource` shall call the function `Det_ReportError` with `ErrorId` parameter `CANSM_E_UNINIT`, if the CanSM module is not initialized yet with `CanSM_Init` (ref. to CANSM023).

[CANSM421:]
The function `CanSM_StartWakeupSource` shall return `E_NOT_OK`, if the CanSM module is initialized and the `network` parameter of the request is not a handle contained in the configuration of the CanSM module (ref. to CANSM161)

[CANSM422:]
The function `CanSM_StartWakeupSource` shall call the function `Det_ReportError` with `ErrorId` parameter `CANSM_E_INVALID_NETWORK_HANDLE`, if the CanSM module is initialized and the requested handle is invalid concerning the CanSM configuration (ref. to CANSM161).

[CANSM423:]
The function `CanSM_StartWakeupSource` shall return `E_OK` and it shall be considered as trigger (ref. to CANSM416) for the state machine of the related network, if the CanSM module is initialized and the requested handle is valid concerning the CanSM configuration (ref. to CANSM161).

8.3.6 CanSM_StopWakeupSource

[CANSM424:]

Service name:	CanSM_StopWakeupSource	
Syntax:	`Std_ReturnType CanSM_StopWakeupSource( NetworkHandleType network )`	
Service ID[hex]:	0x12	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	network	Affected CAN network
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Request accepted E_NOT_OK: Request denied
Description:	This function shall be called by EcuM when a wakeup source shall be stopped.	

[CANSM425:]
The API function CanSM_StopWakeupSource shall return E_NOT_OK, if the CanSM module is not initialized yet with CanSM_Init (ref. to CANSM023).

[CANSM426:]
The function CanSM_StopWakeupSource shall call the function Det_ReportError with ErrorId parameter CANSM_E_UNINIT, if the CanSM module is not initialized yet with CanSM_Init (ref. to CANSM023)

[CANSM427:]
The function CanSM_StopWakeupSource shall return E_NOT_OK, if the CanSM module is initialized and the network parameter of the request is not a handle contained in the configuration of the CanSM module (ref. to CANSM161)

[CANSM428:]
The function CanSM_StopWakeupSource shall call the function Det_ReportError with ErrorId parameter CANSM_E_INVALID_NETWORK_HANDLE, if the CanSM module is initialized and the requested handle is invalid concerning the CanSM configuration (ref. to CANSM161).

[CANSM429:]
The function CanSM_StopWakeupSource shall return E_OK and it shall be considered as trigger (ref. to CANSM417) for the state machine of the related network, if the CanSM module is initialized and the requested handle is valid concerning the CanSM configuration (ref. CANSM161).

8.4 Call-back notifications

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

8.4.1 CanSM_ControllerBusOff

CANSM064:

Service name:	CanSM_ControllerBusOff
Syntax:	void CanSM_ControllerBusOff(uint8 Controller

	)
Service ID[hex]:	0x04
Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	Controller: CAN controller, which detected a bus-off event
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	The CanSM is notified about a bus-off event on a certain CAN controller with this call-out function. It shall execute the bus-off recovery state machine for the corresponding network handle.

CANSM189:

If the function `CanSM_ControllerBusOff` gets a `Controller`, which is not configured as `CanSMControllerId` in the CanSM configuration, it shall report `CANSM_E_PARAM_CONTROLLER` to the DET.

CANSM190:

If the CanSM is not initialized yet, the function reports `CANSM_E_UNINIT` to the DET.

CANSM235:
If the CanSM is initialized and the CanSM configuration (`CanSMControllerId`) covers the notified `Controller` (function parameter), the function shall execute the bus-off recovery state machine for the corresponding network handle.

Addional remarks:

1.) The call context is either on interrupt level (interrupt mode) or on task level (polling mode).

2.) Reentrancy is necessary for multiple CAN controller usage.

8.4.2 CanSM_TxTimeoutException

CANSM348:

Service name:	CanSM_TxTimeoutException
Syntax:	`void CanSM_TxTimeoutException( NetworkHandleType Channel )`
Service ID[hex]:	0x07
Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	Channel: Affected CAN network
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This function shall notify the CanSM module, that the CanNm has detected for the affected partial CAN network a tx timeout exception, which shall be recovered by the CanSM module with a transition to no communication and back to the requested communication mode again.

CANSM403:
The function CanSM_TxTimeoutException shall report CANSM_E_UNINIT to the DET, if the CanSM is not initialized yet.

CANSM349:
If the function CanSM_TxTimeoutException is referenced with a Channel, which is not configured as CanSMNetworkHandle in the CanSM configuration, it shall report CANSM_E_INVALID_NETWORK_HANDLE to the DET.

Remarks: Reentrancy is necessary for different Channels.

8.4.3 CanSM_ClearTrcvWufFlagIndication

CANSM363

Service name:	CanSM_ClearTrcvWufFlagIndication
Syntax:	`void CanSM_ClearTrcvWufFlagIndication( uint8 Transceiver )`
Service ID[hex]:	0x08
Sync/Async:	Synchronous
Reentrancy:	Reentrant for different CAN Transceivers
Parameters (in):	Transceiver: Requested Transceiver
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This callback function shall indicate the CanIf_ClearTrcvWufFlag API process end for the notified CAN Transceiver.

CANSM404:
The function CanSM_ClearTrcvWufFlagIndication shall report CANSM_E_UNINIT to the DET, if the CanSM is not initialized yet.

CANSM402:
If the function CanSM_ClearTrcvWufFlagIndication gets a TransceiverId, which is not configured (ref. to CANSM137) in the configuration of the CanSM module, it shall call the function Det_ReportError with ErrorId parameter CANSM_E_PARAM_TRANSCEIVER.

8.4.4 CanSM_TransceiverModeIndication

CANSM365:

Service name:	CanSM_TransceiverModeIndication
Syntax:	`void CanSM_TransceiverModeIndication( uint8 Transceiver, CanIf_TransceiverModeType TransceiverMode )`
Service ID[hex]:	0x09
Sync/Async:	Synchronous

Reentrancy:	Reentrant for different CAN Transceivers	
Parameters (in):	Transceiver	Requested Transceiver
	TransceiverMode	Notified TransceiverMode
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	This callback function shall indicate the reached TransceiverMode for the notified CAN Transceiver after process end of the API CanIf_SetTransceiverMode.	

CANSM406:
The function `CanSM_TransceiverModeIndication` shall report `CANSM_E_UNINIT` to the DET, if the CanSM module is not initialized yet.

CANSM405:
If the function `CanSM_TransceiverModeIndication` gets a `TransceiverId`, which is not configured (ref. to CANSM137) in the configuration of the CanSM module, it shall call the function `Det_ReportError` with `ErrorId` parameter `CANSM_E_PARAM_TRANSCEIVER`.

8.4.5 CanSM_CheckTrcvWakeFlagIndication

CANSM367

Service name:	CanSM_CheckTransceiverWakeFlagIndication	
Syntax:	`void CanSM_CheckTransceiverWakeFlagIndication( uint8 Transceiver )`	
Service ID[hex]:	0x0a	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant for different CAN Transceivers	
Parameters (in):	Transceiver	Requested Transceiver
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	This callback function indicates the CheckTransceiverWakeFlag API process end for the notified CAN Transceiver.	

CANSM407:
The function `CanSM_CheckTransceiverWakeFlagIndication` shall report `CANSM_E_UNINIT` to the DET, if the CanSM module is not initialized yet.

CANSM408:
If the function `CanSM_CheckTransceiverWakeFlagIndication` gets a `TransceiverId`, which is not configured (ref. to CANSM137) in the configuration of the CanSM module, it shall call the function `Det_ReportError` with `ErrorId` parameter `CANSM_E_PARAM_TRANSCEIVER`.

8.4.6 CanSM_ConfirmPnAvailability

CANSM400:

Service name:	CanSM_ConfirmPnAvailability	
Syntax:	`void CanSM_ConfirmPnAvailability( uint8 Transceiver )`	
Service ID[hex]:	0x06	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Transceiver	CAN transceiver, which was checked for PN availability
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	This callback function indicates that the transceiver is running in PN communication mode.	

Effect of this indication:
Refer to CANSM401

DET errors:

CANSM409:
The function `CanSM_ConfirmPnAvailability` shall report `CANSM_E_UNINIT` to the DET, if the CanSM module is not initialized yet.

CANSM410:
If the function `CanSM_ConfirmPnAvailability` gets a `TransceiverId`, which is not configured (ref. to CANSM137) in the configuration of the CanSM module, it shall call the function `Det_ReportError` with `ErrorId` parameter `CANSM_E_PARAM_TRANSCEIVER`.

8.5 Scheduled functions

Basic Software Scheduler directly calls these functions. The following functions shall have no return value and no parameter. All functions shall be non-reentrant.

Terms and definitions:
Fixed cyclic: Fixed cyclic means that one cycle time is defined at configuration and shall not be changed because functionality is requiring that fixed timing (e.g. filters).
Variable cyclic: Variable cyclic means that the cycle times are defined at configuration, but might be mode dependent and therefore vary during runtime.
On pre condition: On pre condition means that no cycle time can be defined. The function will be called when conditions are fulfilled. Alternatively, the function may be called cyclically however the cycle time will be assigned dynamically during runtime by other modules.

8.5.1 CanSM_MainFunction

CANSM065:

Service name:	CanSM_MainFunction
Syntax:	`void CanSM_MainFunction( )`
Service ID[hex]:	0x05
Timing:	FIXED_CYCLIC
Description:	Scheduled function of the CanSM

CANSM167:

The main function of the CanSM shall process the CanSM state machine for each configured network handle (ref. to chapter 7.7).

8.6 Expected Interfaces

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

8.6.1 Mandatory Interfaces

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

API function	***Description***
BswM_CanSM_CurrentState	Function called by CanSM to indicate its current state.
CanIf_CheckTrcvWakeFlag	Requests the CanIf module to check the Wake flag of the designated CAN Transceiver.
CanIf_ClearTrcvWufFlag	Requests the CanIf module to clear the WUF flag of the designated CAN Transceiver.
CanIf_GetTxConfirmationState	This service reports, if any TX confirmation has been done for the whole CAN controller since the last CAN controller start.
CanIf_SetControllerMode	CANIF003: This service calls the corresponding CAN Driver service for changing of the CAN controller mode. It initiates a transition to the requested CAN controller mode of one or multiple CAN controllers. This service calls Can_SetControllerMode(Controller, Transition) for the requested CAN controller. Development errors: If the CAN Interface was not initialized before, the call of this function will be reported to the development error tracer (CANIF_E_UNINIT). The function returns with E_NOT_OK. Invalid values of Controller will be reported to the development error tracer (CANIF_E_PARAM_CONTROLLER). Caveats: Re-entrant calls of this API are allowed only for different controller Identifiers. The CAN Driver must be initialized after Power ON. The CAN Interface must be initialized after Power ON. Configuration: ID of the CAN controller is published inside the configuration description of the CAN Interface.
CanIf_SetPduMode	CANIF008: This service sets the requested mode at all L-PDUs of the

	predefined logical PDU channel. This channel parameter can be derived from Controller. Development errors: Invalid values of Controller will be reported to the development error tracer (CANIF_E_PARAM_CONTROLLLER). If the CAN Interface was not initialized before, the call of this function will be reported to the development error tracer (CANIF_E_UNINIT). The function returns with E_NOT_OK. Caveats: Re-entrant calls of this API are allowed only for different channel Identifiers. The CAN Interface must be initialized after Power ON. Configuration: The channel mode is configurable by CANIF_CANTXPDUID_CONTROLLER/ CANIF_CANRXPDUID_CONTROLLER.
CanIf_SetTransceiverMode	CANIF287: This API requests actual state of CAN Transceiver Driver. For more details, please refer to the X[9]X XSpecification of CAN Transceiver DriverX. This service calls CanTrcv_SetOpMode (Transceiver, *OpMode) for the corresponding requested CAN transceiver. Development errors: Invalid values of transceiver or transceiver mode will be reported to the development error tracer (CANIF_TRCV_E_TRANSCEIVER, CANIF_TRCV_E_TRCV_NOT_STANDBY or CANIF_TRCV_E_TRCV_NOT_NORMAL) If the CAN Interface was not initialized before, the call of this function will be reported to the development error tracer (CANIF_E_UNINIT). The function returns with E_NOT_OK. Caveats: This API shall be applicable to all CAN transceivers with all values independent, if the transceiver hardware supports these modes or not. This is to ease up the view of the Can Interface to the assigned physical CAN channel. If the mode is not supported, the return value shall be E_OK. Configuration: The number of supported transceiver types for each network is set up in the configuration phase. If no transceiver is used, this API shall not be provided.
CanNm_ConfirmPnAvailability	Enables the PN filter functionality on the indicated NM channel. Availability: The API is only available if CanNmPnEnabled is TRUE.
ComM_BusSM_ModeIndication	Indication of the actual bus mode by the corresponding Bus State Manager. ComM shall propagate the indicated state to the users with means of the RTE (see ComM661).
Com_SendSignal	The service Com_SendSignal updates the signal object identified by SignalId with the signal referenced by the SignalDataPtr parameter. If the signal has the Triggered transfer property, the update is followed by immediate transmission (within the next main function at the latest) of the I-PDU associated with the signal except when the signal is packed into an I-PDU with Periodic transmission mode; in this case, no transmission is initiated by the call to this service. If the signal has the Pending transfer property, no transmission is caused by the update.
Dem_ReportErrorStatus	Reports errors to the DEM.

8.6.2 Optional Interfaces

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

API function	*Description*
Det_ReportError	Service to report development errors.

9 Sequence diagrams

The interactions of the CanSM module with the depending modules CanIf, ComM, BswM, Dem and CanNm are specified in the state machine diagrams (ref. to Figure 7-1 - Figure 7-3). Sequence diagrams for these interactions are not provided in this revision of the CanSM SWS.

For the special use case of CAN network deinitialization with partial network support please refer to chapter 9 of [9] (Specification of CAN Transceiver Driver).

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. It also specifies a template (table) you shall use for the parameter specification. We intend to leave Chapter 10.1 in the specification to guarantee comprehension.
Chapter 10.2 specifies the structure (containers) and the parameters of the module CanSM.

Chapter 10.3 specifies published information of the module CanSM.

10.1 How to read this chapter

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

- AUTOSAR Layered Software Architecture [2]
- AUTOSAR ECU Configuration Specification [4]
  This document describes the AUTOSAR configuration methodology and the AUTOSAR configuration meta model in detail.

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

10.1.1 Configuration and configuration parameters

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

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

10.1.2 Variants

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

10.1.3 Containers

Containers structure the set of configuration parameters. This means:

- *all* configuration parameters are kept in containers.

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

10.1.4 Specification template for configuration parameters

The following tables consist of three sections:

- the general section
- the configuration parameter section
- the section of included/referenced containers

The following template (table) shall be used to specify the configuration parameters in the context of containers.

SWS Item	*<CANSM159>*
Container Name	*<Identifies the container by a name, e.g., CanDriverConfiguration>*
Description	*<Explains the intention and the content of the container .>*
Configuration Parameters	

All parameters belonging to the class have to be specified using following table template:

Name	*<Identifies the parameter by name. The naming convention shall follow BSW00408.>*		
Description	*<Explains the intention of the configuration parameter.>*		
Type	*<Specify the type of the parameter (e.g., `uint8..uint32`) if possible or mark it "–">*		
Unit	*<Specify the unit of the parameter (e.g., `ms`) if possible or mark it "–" >*		
Range	*<Specify the range (or possible values) of the parameter (e.g., `1..15`, `ON,OFF`) if possible or mark it "–">*	*<Describe the value(s) or ranges.>*	
Configuration Class	***Pre-compile***	*see*[3]	*<Refer here to (a) variant(s).>*
	Link time	*see*[4]	*<Refer here to (a) variant(s).>*
	Post Build	*see*[5]	*<Refer here to (a) variant(s).>*
Scope	*<Describe the scope of the parameter if known or mark it as "- -".* *The scope describes the impact of the configuration parameter: Does the setting affect only one instance of the module (instance), all instances of this module (module), the ECU or a network.* *Possible values of scope :* *instance, module, ECU, network>*		
Dependency	*<Describe the dependencies with respect to the scope if known not mark it as "- -".>*		

[3] see the explanation below this table - Pre-compile time
[4] see the explanation below this table - Link time
[5] see the explanation below this table - Post Build

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
<Reference a valid (sub)container by its name, e.g.,CanController>	*<Specifies the possible number of instances of the referenced container and its contained configuration parameters.* *Possible values:* *<multiplicity>* *<min_multiplicity..max_multiplicity>* *>*	*<Describe the scope of the referenced sub-container if known or mark it as "- -".* *The scope describes the impact of the configuration parameter: Does the setting affect only one instance of the module (instance), all instances of this module (module), the ECU or a network.* *Possible values of scope:* *instance, module, ECU, network>* *<Describe the dependencies with respect to the scope if known not mark it as "- -".>*

Pre-compile time - specifies whether the configuration parameter shall be of configuration class *Pre-compile time* or not

Label	***Description***
x	The configuration parameter shall be of configuration class *Pre-compile time*.
--	The configuration parameter shall never be of configuration class *Pre-compile time*.

Link time - specifies whether the configuration parameter shall be of configuration class *Link time* or not

Label	***Description***
x	The configuration parameter shall be of configuration class *Link time*.
--	The configuration parameter shall never be of configuration class *Link time*.

Post Build - specifies whether the configuration parameter shall be of configuration class *Post Build* or not

Label	***Description***
x	The configuration parameter shall be of configuration class *Post Build* and no specific implementation is required.
L	*Loadable* – the configuration parameter shall be of configuration class *Post Build* and only one configuration parameter set resides in the ECU.
M	*Multiple* – the configuration parameter shall be of configuration class *Post Build* and is selected out of a set of multiple parameters by passing a dedicated pointer to the init function of the module.
--	The configuration parameter shall never be of configuration class *Post Build*.

10.2 Containers and configuration parameters

CANSM155:

The following chapters summarize all configuration parameters of the CanSM module. The configuration of these parameters has to be tool-based (XML-format). The detailed meanings of the parameters describe chapter 7 and chapter 8.

CANSM156:

The consistency of the configuration must be checked by the configuration tool at configuration time. Configuration rules and constraints for plausibility checks shall be performed during configuration time, where possible.

10.2.1 Variants

CANSM122:

Variant 1: Only pre-compile parameters

Variant 2: Mix of pre-compile and link time parameters

Variant 3: Mix of pre compile-, link time and post build time parameters

Note:

In the generated tables below following naming is used for the variants:

- Variant 1 – VARIANT-PRE-COMPILE
- Variant 2 – VARIANT-LINK-TIME
- Variant 3 – VARIANT-POST-BUILD

CANSM163:

For post build time parameters the type “x” was chosen to allow both variants of implementations with either loadable (“L”) or multiple (“M”) types of post built parameters.

10.2.2 CanSMGeneral

SWS Item	**CANSM314_Conf :**
Container Name	CanSMGeneral
Description	Container for general pre-compile parameters of the CanSM module.
Configuration Parameters	

SWS Item	**CANSM133_Conf :**		
Name	CanSMDevErrorDetect {CANSM_DEV_ERROR_DETECT}		
Description	Enables and disables the development error detection and notification mechanism.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**CANSM312_Conf :**
Name	CanSMMainFunctionTimePeriod {CANSM_MAIN_FUNCTION_PERIOD}
Description	This parameter defines the cycle time of the function CanSM_MainFunction in seconds.

Multiplicity	1		
Type	FloatParamDef		
Range	0.001 .. 65.535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**CANSM134_Conf :**		
Name	CanSMPncSupport		
Description	Enables or disables support of partial networking. False: Partial Networking is disabled True: Partial Networking is enabled		
Multiplicity	0..1		
Type	BooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local dependency: This parameter shall be available only if ComMPncSupport is enabled in ComM		

SWS Item	**CANSM311_Conf :**		
Name	CanSMVersionInfoApi {CANSM_VERSION_INFO_API}		
Description	Activate/Deactivate the version information API (CanSM_GetVersionInfo). true: version information API activated. false: version information API deactivated.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

10.2.3 CanStateManagerConfiguration

SWS Item	**CANSM123 :**
Container Name	CanStateManagerConfiguration [Multi Config Container]
Description	This container contains the global parameters of the CanSM and sub containers, which are for the CAN network specific configuration.
Configuration Parameters	

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CanStateManagerNetworks	1..*	This container contains the CAN network specific parameters of each CAN network

10.2.4 CanStateManagerNetworks

SWS Item	**CANSM126 :**
Container Name	CanStateManagerNetworks
Description	This container contains the CAN network specific parameters of each CAN network
Configuration Parameters	

SWS Item	**CANSM131 :**		
Name	CanSMBorCounterL1ToL2 {CANSM_BOR_COUNTER_L1_TO_L2}		
Description	This threshold defines at which bus-off-counter value the bus-off recovery state machine switches from level 1 to level 2.		
Multiplicity	1		
Type	IntegerParamDef		
Range	0 .. 255		
Default value	--		
ConfigurationClass	***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	**CANSM128 :**		
Name	CanSMBorTimeL1 {CANSM_BOR_TIME_L1}		
Description	This time parameter defines in seconds the duration of the bus-off recovery time in level 1 (short recovery time).		
Multiplicity	1		
Type	FloatParamDef		
Range	0 .. 65.535		
Default value	--		
ConfigurationClass	***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	**CANSM129 :**		
Name	CanSMBorTimeL2 {CANSM_BOR_TIME_L2}		
Description	This time parameter defines in seconds the duration of the bus-off recovery time in level 2 (long recovery time).		
Multiplicity	1		
Type	FloatParamDef		
Range	0 .. 65.535		
Default value	--		
ConfigurationClass	***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	**CANSM130 :**
Name	CanSMBorTimeTxEnsured {CANSM_BOR_TIME_TX_ENSURED}
Description	This parameter defines in seconds the duration of the bus-off event check. This check assesses, if the recovery has been successful after the recovery reenables the transmit path. If a new bus-off occurs during this time period, the CanSM assesses this bus-off as sequential bus-off without successful recovery. Because

	a bus-off only can be detected, when PDUs are transmitted, the time has to be great enough to ensure that PDUs are transmitted again.		
Multiplicity	1		
Type	FloatParamDef		
Range	0 .. 65.535		
Default value	--		
ConfigurationClass	***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: CANSM_BOR_TX_CONFIRMATION_POLLING disabled		

SWS Item	**CANSM339 :**		
Name	CanSMBorTxConfirmationPolling {CANSM_BOR_TX_CONFIRMATION_POLLING}		
Description	This parameter shall configure, if the CanSM polls the CanIf_GetTxConfirmationState API to decide the bus-off state to be recovered instead of using the CanSMBorTimeTxEnsured parameter for this decision.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Local		

SWS Item	**CANSM161 :**		
Name	CanSMNetworkHandle {CANSM_NETWORK_HANDLE}		
Description	Unique handle to identify one certain CAN network. Reference to one of the network handles configured for the ComM.		
Multiplicity	1		
Type	Reference to [ComMChannel]		
ConfigurationClass	***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: ComM		

SWS Item	**CANSM137 :**		
Name	CanSMTransceiverId {CANSM_TRANSCEIVER_ID}		
Description	ID of the CAN transceiver assigned to the configured network handle. Reference to one of the transceivers of CanTrcv configuration.		
Multiplicity	1		
Type	Reference to [CanIfTransceiverDrvConfig]		
ConfigurationClass	***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: CanIf		

SWS Item	**CANSM344 :**

Name	CanSMTransceiverPnSupport		
Description	Indicates the ability of partial networking for the configured CAN transceiver. The information about the ability of the transceiver to support the selective wake-up function is hold in the parameter CanTrcvHwPnSupport.		
Multiplicity	1		
Type	Reference to [CanTrcvChannel]		
ConfigurationClass	***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: CanTrcv		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CanStateManagerControllers	1..*	This container contains the controller IDs assigned to a CAN network.

10.2.5 CanStateManagerControllers

SWS Item	**CANSM127 :**
Container Name	CanStateManagerControllers
Description	This container contains the controller IDs assigned to a CAN network.
Configuration Parameters	

SWS Item	**CANSM141 :**		
Name	CanSMControllerId {CANSM_CONTROLLER_ID}		
Description	Unique handle to identify one certain CAN controller. Reference to one of the CAN controllers of CAN driver (Can) configuration.		
Multiplicity	1		
Type	Reference to [CanIfControllerConfig]		
ConfigurationClass	***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: CanIf		

No Included Containers

10.3 Published Information

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

The standard common published information like

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

is provided in the BSW Module Description Template (see [16] Figure 4.1 and Figure 7.1).

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