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

Document Version	1.2.0
Document Status	Final
Part of Release	3.0
Revision	7

Document Change History

Date	Version	Changed by	Change Description
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
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.

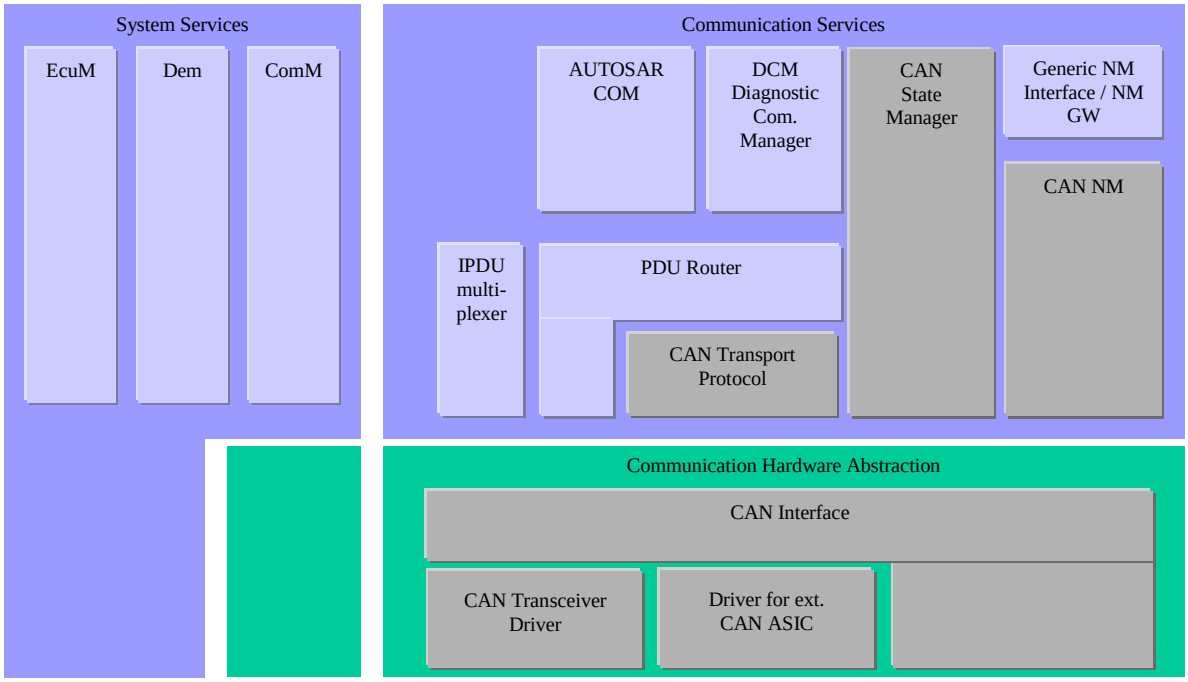

Figure 1-1: Layered Software Architecture from CanSM point of view

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
COM	Communication
PDU	Protocol Data Unit
RX	Receive
TX	Transmit
API	Application Program Interface

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 Communication
AUTOSAR_SWS_COM.pd

[15] Specification of BSW Scheduler
AUTOSAR_SWS_BSW_Scheduler.pdf

[16] Specification of Development Error Tracer
AUTOSAR_SWS_DET.pdf

[17] AUTOSAR Basic Software Module Description Template,
AUTOSAR_BSW_Module_Description.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.

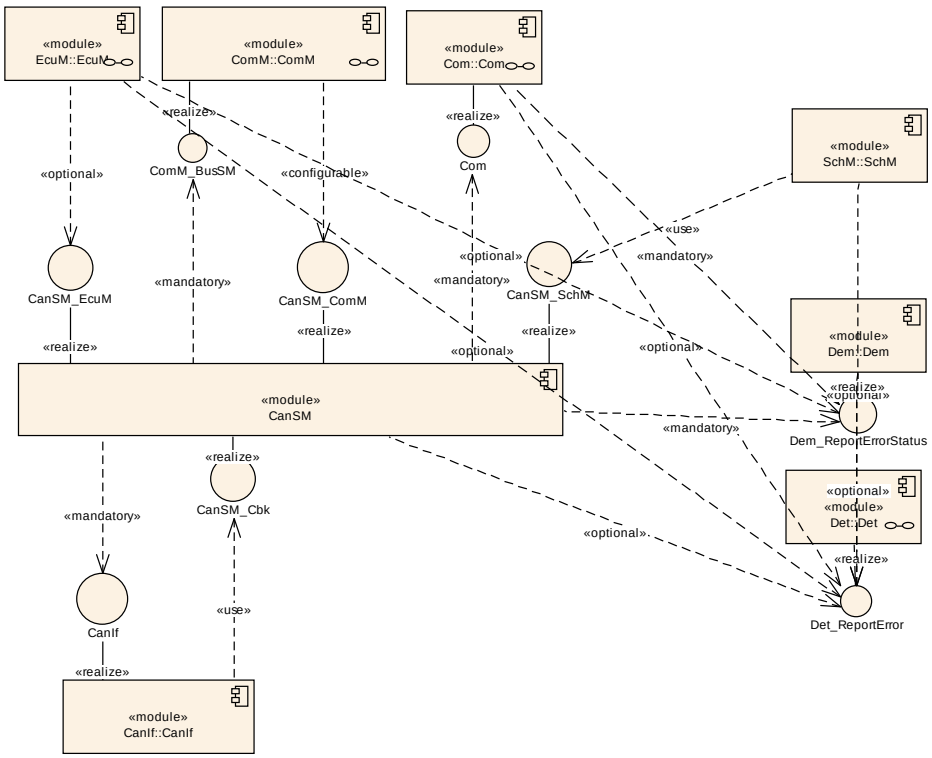

Figure 5-1: Dependencies of the CanSM to other BSW modules

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 (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 [15] 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 Communication (COM)

The CanSM uses the API of COM to control CAN related PDU groups (refer to [14] for detailed specification of this module).

5.5 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.

5.6 Diagnostic Event Manager (DEM)

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

5.7 Development Error Tracer (DET)

The CanSM reports development errors to the DET, if development error handling is switched on by configuration (refer to [16] for detailed specification of this module).

5.8 File structure

5.8.1 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

5.8.2 Header file structure

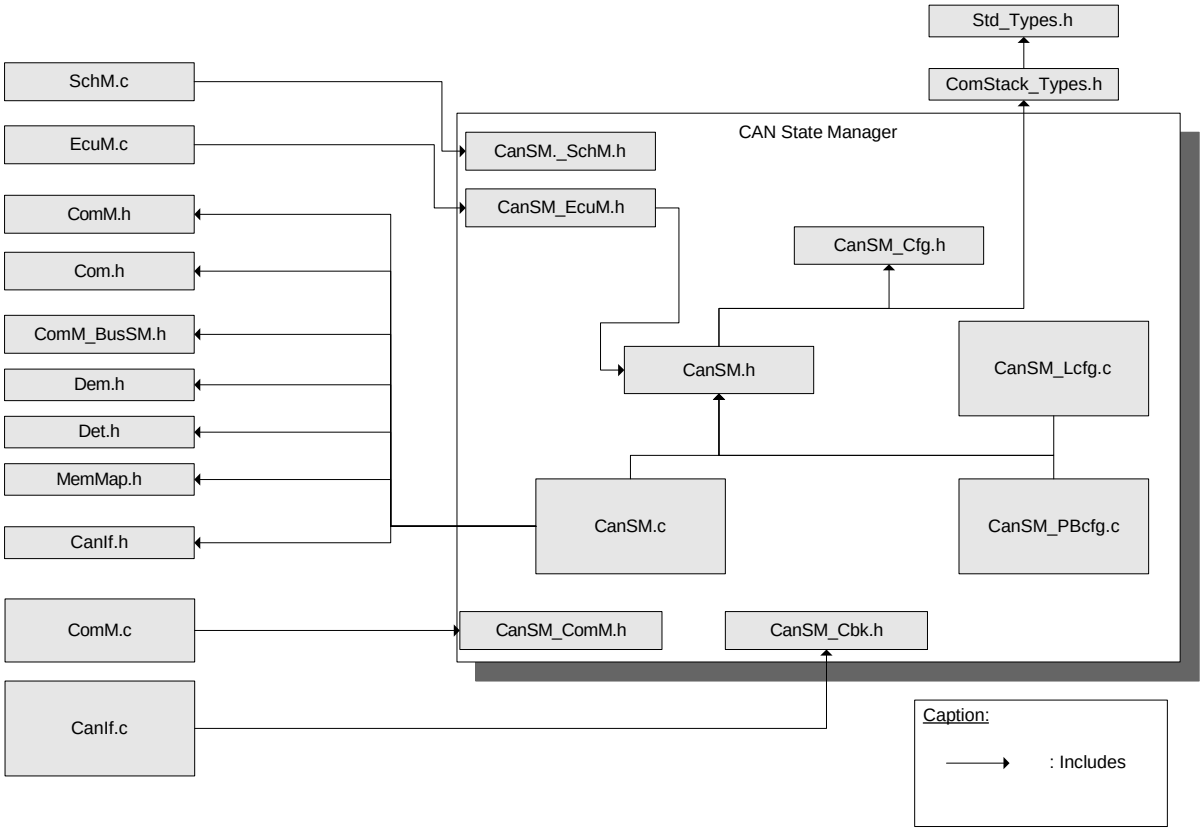

Figure 5-2: Header file structure

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 headerfile CanSM.h shall include the headerfile ComStack_Types.h.

Remark: The headerfile ComStack_Types.h includes the headerfile Std_Types.h

CANSM239:

The headerfile CanSM.h shall include the headerfile 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 5.8.1 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.

CANSM172:

The CanSM shall include the header file Com.h.

Rationale: API of the COM needed for PDU group 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.

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 10.2.4
[BSW00375] Notification of wake-up reason	Not applicable (no wake up interrupt)
[BSW101] Initialization interface	CANSM023:
[BSW00416] Sequence of Initialization	CANSM217:
[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 5.8.2
[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 0
[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 5.8
[BSW158] Separation of configuration from implementation	Chapter 5.8
[BSW00314] Separation of interrupt frames and service routines	Not applicable (not in scope of this spec)
[BSW00370] Separation of callback interface from API	Chapter 5.8
[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 5.8
[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 5.8
[BSW00302] Limit exported information	Chapter 5.8
[BSW00328] Avoid duplication of code	CANSM079:
[BSW00312] Shared code shall be reentrant	CANSM079:
[BSW006] Platform independency	CANSM079:
[BSW00357] Standard API return type [	Chapter 0
[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 0
[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 10.2.4
[BSW00379] Module identification	Chapter 10.2.4
[BSW003] Version identification	Chapter 10.2.4, CANSM024:
[BSW00318] Format of module version numbers	Chapter 10.2.4
[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 7.6.1
[BSW01145] Management of CAN devices	chapter 7.3 chapter 7.6
[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: CANSM045:
[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 7.6).

CANSM241:

This transition shall translate the request into a respective API call to control the assigned CAN peripherals and the PDU handling (refer to section 7.3 and 7.4).

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 7.6):

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 Control of peripherals

7.3.1 CAN Transceivers

Each CAN Transceiver belongs to one CAN network.

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

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 7.6).

CANSM043:

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

7.3.2 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).

7.4 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.

7.5 Control of PDU groups

The CanSM has the responsibility to control the PDU groups, which are related to the configured CAN networks.

Remark for configuration of the CanSM: The user has to configure a RX and a TX PDU group in the CanSM configuration for each used CAN network. Each RX PDU group shall contain the RX PDUs of the configured CAN network. Each TX PDU group shall contain the TX PDUs of the configured CAN network.

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

CANSM173:

The CanSM shall use the API of COM to control CAN related PDU groups depending on the network mode state machine.

CANSM091:

The configuration of the CanSM shall assign to each CAN network handle a certain RX PDU group and a certain TX PDU group.

Rationale: The API of COM, which the CanSM shall use, is related to PDU groups.

CANSM178:

It shall be configurable for the CanSM, that it disables the RX PDU group deadline monitoring depending on the bus-off recovery state machine (see section 7.7).

Rationale: Avoid RX timeouts in COM, if a bus-off is present.

7.6 Network mode state machine

CANSM032:

The CanSM shall implement for each configured network handle one network mode state machine, which is specified in Figure 7-1.

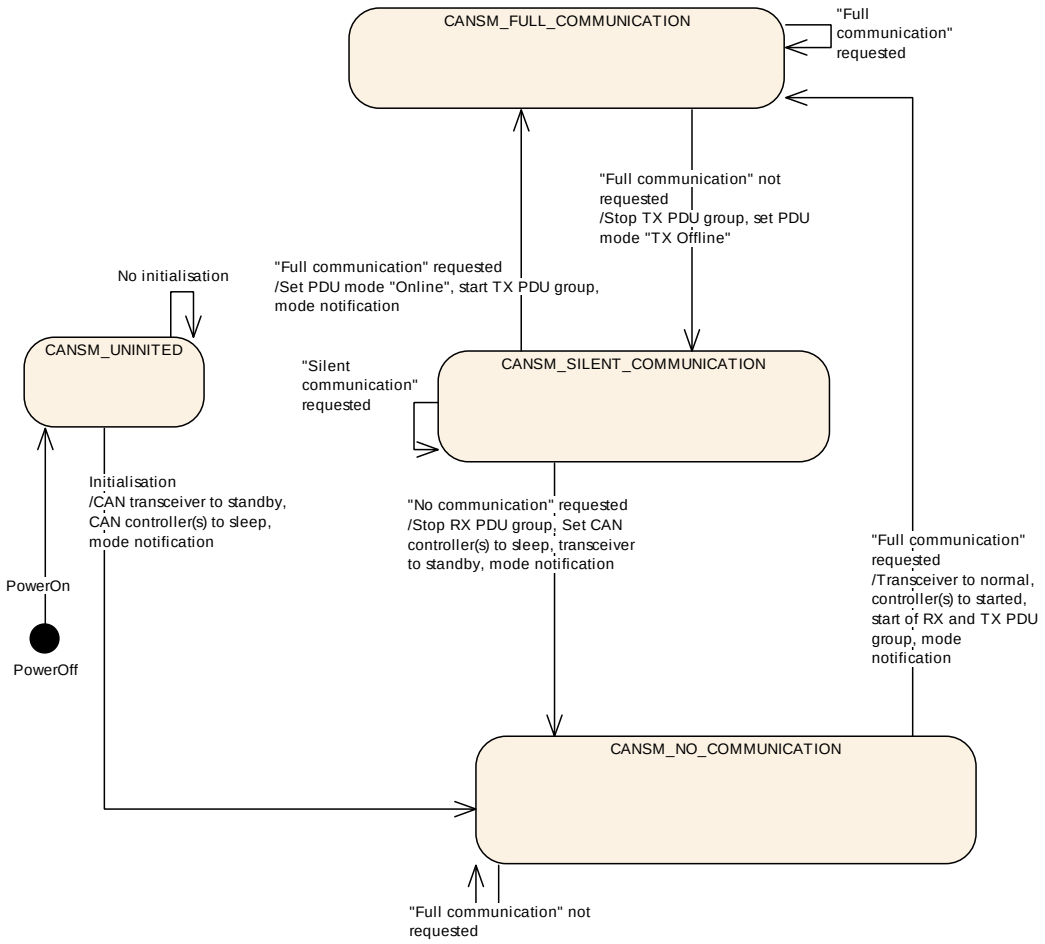

Figure 7-1: Network mode state machine of the CanSM

CANSM233:

The network mode state machines shall use the type `CanSM_NetworkModeStateType` as state values.

7.6.1 Initial transition

CANSM211:

In the state `CANSM_UNINITED` the state machine shall have a transition to `CANSM_NO_COMMUNNICATION`, if the CanSM is initialized. In this transition the CanSM shall interact like specified in the sequence diagram Figure 9-1.

Remark: This transition sets up the wake-up capability at initialization.

7.6.2 Transition from no to full communication

CANSM212:

In the state `CANSM_NO_COMMUNICATION` the state machine shall have a transition to `CANSM_FULL_COMMUNICATION`, if the ComM requests `COMM_FULL_COMMUNICATION` for the corresponding network handle. In this transition the CanSM shall interact like specified in the sequence diagram Figure 9-2.

7.6.3 Transition from silent to no communication

CANSM214:

In the state `CANSM_SILENT_COMMUNICATION` the state machine shall have a transition to `CANSM_NO_COMMUNICATION`, if the ComM requests `COMM_NO_COMMUNICATION` for the corresponding network handle. In this transition the CanSM shall interact like specified in the sequence diagram Figure 9-3.

Remark: This transition sets up the wake-up capability again, which has to be done when the bus goes sleeping.

7.6.4 Transition from silent to full communication

CANSM215:

In the state `CANSM_SILENT_COMMUNICATION` the state machine shall have a transition to `CANSM_FULL_COMMUNICATION`, if the ComM requests `COMM_FULL_COMMUNICATION` for the corresponding network handle. In this transition the CanSM shall interact like specified in the sequence diagram Figure 9-4.

7.6.5 Transition from full to silent communication

CANSM216:

In the state CANSM_FULL_COMMUNICATION the state machine shall have a transition to CANSM_SILENT_COMMUNICATION, if the ComM requests COMM_SILENT_COMMUNICATION or COMM_NO_COMMUNICATION for the corresponding network handle. In this transition the CanSM shall interact like specified in the sequence diagram Figure 9-5.

7.7 Bus-off recovery state machine

CANSM045:

The CanSM shall implement for each CAN network handle a state machine for the bus-off recovery, which is specified with the diagram in Figure 7-2 for the case that configuration parameter CANSM_BOR_TX_CONFIRMATION_POLLING (ref. to CANSM339) is FALSE.

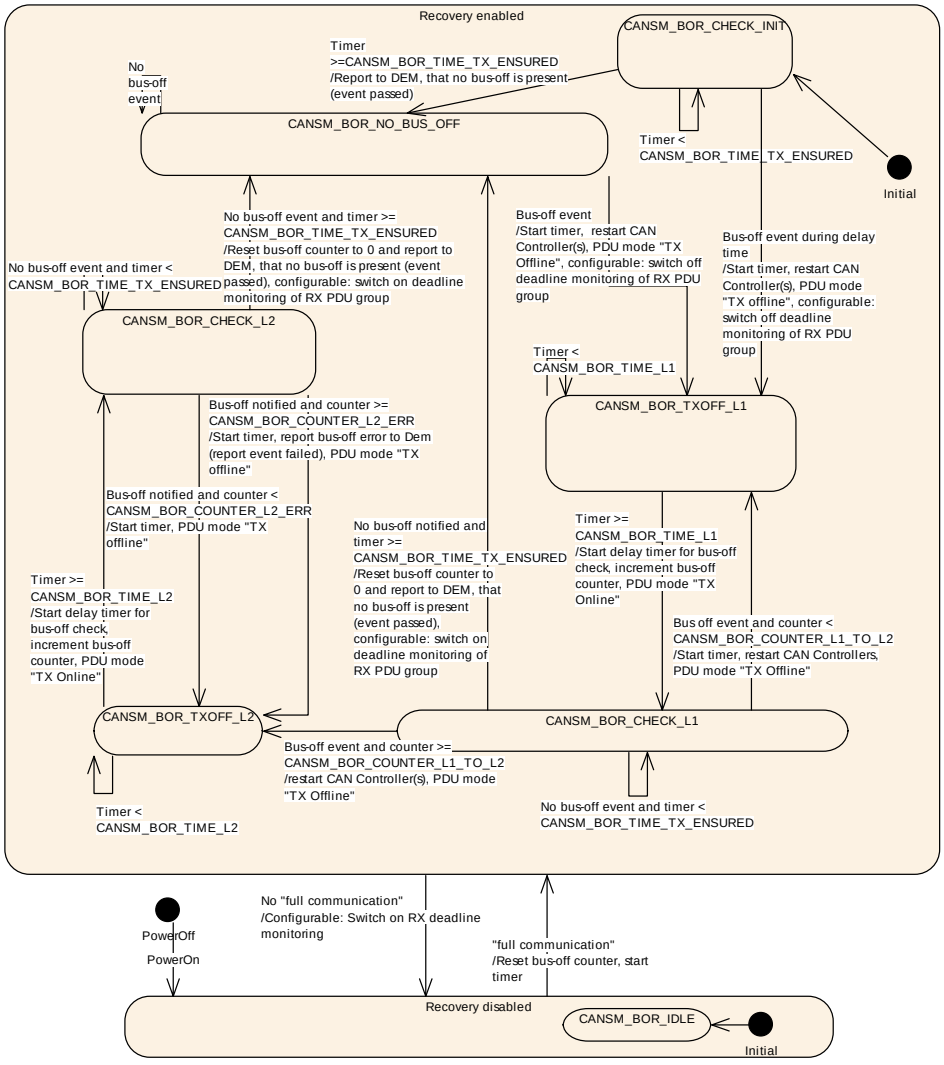

Figure 7-2: Bus-off recovery state machine of the CanSM

Remark:

For a clearer understanding the state machine is shown with an outer and inner one. The outer shall point out, when the recovery handling is active (network mode is equal to full communication mode or silent communication mode) and when it is inactive (network mode is equal to no communication).

CANSM340:

If the configuration parameter `CANSM_BOR_TX_CONFIRMATION_POLLING` is enabled (TRUE) (ref. to CANSM339), the CanSM module shall poll the CanIf API `CanIf_GetTxConfirmationState` and replace for all transitions where applicable the condition "Timer >= `CANSM_BOR_TIME_TX_ENSURED`" with "the API `CanIf_GetTxConfirmationState` returns `CANIF_TX_RX_NOTIFICATION` for all configured CAN controllers of the CAN network".

CANSM341:

If the configuration parameter `CANSM_BOR_TX_CONFIRMATION_POLLING` is enabled (TRUE) (ref. to CANSM339), the CanSM module shall poll the CanIf API `CanIf_GetTxConfirmationState` and replace for all transitions where applicable the condition "Timer < `CANSM_BOR_TIME_TX_ENSURED`" with "the API `CanIf_GetTxConfirmationState` returns `CANIF_NO_NOTIFICATION` for all configured CAN controllers of the CAN network".

CANSM168:

The implemented state machine shall only use the state values of the inner state machine, which are specified with the type `CanSM_BusOffRecoveryStateType` (see section 8.2.2).

CANSM057:

Each bus-off recovery state machine shall have a bus-off-counter. This bus-off-counter is compared with the parameters `CanSMBorCounterL1ToL2` and `CANSM_BOR_COUNTER_L2_TO_ERR`. The counter shall be of type "uint8". If the counter reaches the value 255 and is incremented, it shall remain at this value.

CANSM203:

Each bus-off recovery state machine shall have one bus-off-timer, which is compared with the time parameters `CanSMBorTimeTxEnsured`, `CanSMBorTimeL1` and `CanSMBorTimeL2`.

CANSM048:

The bus-off recovery state machine shall evaluate the information, if a bus-off event has occurred for the currently processed network handle.

Remark: The CanIf hands over the information about bus-off events detected by a certain CAN Controller to the CanSM (see chapter 8.4.1). The relationship between CAN controllers and CAN networks is part of the CanSM configuration. So it can get to know, which network is impacted.

CANSM047:

The bus-off recovery state machine shall report a production error to the DEM. The sequence diagram Figure 9-9 specifies, in which way the state machine shall report

this event to be passed. The sequence diagram Figure 9-10 specifies, in which way the state machine shall report the event to be failed.

Rationale: This production error shall provide the information, if a bus-off cannot be recovered for a longer time.

CANSM206:

To restart the CAN controllers the CanSM shall interact with the CanIf like specified in the sequence diagram Figure 9-11.

CANSM207:

To switch the PDU mode to “TX Online” the CanSM shall interact with the CanIf like specified in the sequence diagram Figure 9-13.

CANSM209:

To switch the PDU mode to “TX Offline” the CanSM shall interact with the CanIf like specified in the sequence diagram Figure 9-12.

CANSM208:

To switch on the RX deadline monitoring (configurable) the CanSM shall interact with COM like specified in the sequence diagram Figure 9-15.

CANSM210:

To switch off the RX deadline monitoring (configurable) the CanSM shall interact with COM like specified in the sequence diagram Figure 9-15.

7.7.1 Hints for configuration

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.

7.8 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_INVALID_NETWORK_MODE	0x04
The bus-off recovery state machine of a CAN network has detected a certain amount of sequential bus-offs without successful recovery	Production	Refer to CANSM070:	Assigned by DEM

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 chapter 7.7.

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 chapter 7.7.

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 COM types

The CanSM includes the following listed types of the COM module (refer to [14]):

- Com_PduGroupIdType

8.1.4 ComM types

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

- ComM_ModeType

8.1.5 CanIf types

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

- CanIf_ControllerModeType
- CanIf_ChannelSetModeType
- CanIf_TransceiverModeType

8.1.6 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:	`--`	--	
Element:	`CanSM`	`--`	--
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_NetworkModeStateType

CANSM236:

Name:	`CanSM_NetworkModeStateType`	
Type:	`Enumeration`	
Range:	`CANSM_UNINITED`	CanSM is unititialized in this state
	`CANSM_NO_COMMUNICATION`	ComM requests COMM_NO_COMMUNICATION in this state
	`CANSM_SILENT_COMMUNICATION`	ComM requests COMM_SILENT_COMMUNICATION in this state
	`CANSM_FULL_COMMUNICATION`	ComM requests COMM_FULL_COMMUNICATION in this state
Description:	This type shall define the states of the network mode state machine.	

8.2.3 CanSM_BusOffRecoveryStateType

CANSM169:

Name:	`CanSM_BusOffRecoveryStateType`	
Type:	`Enumeration`	
Range:	`CANSM_BOR_IDLE`	Idle state
	`CANSM_BOR_CHECK`	Initial bus-off check at beginning of full-communication
	`CANSM_BOR_NO_BUS_OFF`	Regular state during full-communication without detected bus-off events
	`CANSM_BOR_TXOFF_L1`	Bus-off recovery level 1 state, TX disabled
	`CANSM_BOR_CHECK_L1`	Bus-off recovery level 1 state, TX enabled again
	`CANSM_BOR_TXOFF_L2`	Bus-off recovery level 2 state, TX disabled
	`CANSM_BOR_CHECK_L2`	Bus-off recovery level 2 state, TX enabled again
Description:	This type shall define the states of the bus-off recovery state machine.	

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.

8.4 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.

8.5 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 execute the bus-off recovery state machines of each network handle, which is configured for the CanSM.

8.6 Expected Interfaces

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

8.6.1 Mandatory Interfaces

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

API function	*Module*	*Description*
CanIf_SetControllerMode	CanIf	To control operating states of CAN controllers
CanIf_SetTransceiverMode	CanIf	To control operating states of CAN Transceivers
CanIf_SetPduMode	CanIf	To control the PDU mode of a CAN controller
CanIf_GetTxConfirmationState	CanIf	This service reports, if any TX confirmation has been done for the whole CAN controller since the last CAN controller start.
Dem_ReportErrorStatus	DEM	To report production errors to DEM
ComM_BusSM_ModeIndication	ComM	The ComM provides this function to be notified about communication mode changes of its network handles.
Com_IpduGroupStart	COM	To start CanSM related PDU groups
Com_IpduGroupStop	COM	To stop CanSM related PDU groups

8.6.2 Optional Interfaces

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

API function	*Module*	*Description*	*Configuration parameter (description see chapter 10)*
Det_ReportError	Det	Development error notification	`CanSMDevErrorDetect`
Com_DisableReceptionDM	COM	Disable RX PDU Group Deadline Monitoring during bus-off	`CanSMBorDisableRxDlMonitoring`
Com_EnableReceptionDM	COM	Enable RX PDU Group Deadline	`CanSMBorDisableRxDlMonitoring`

		Monitoring after finishing bus-off recovery	

9 Sequence diagrams

9.1 Network mode state machine sequences

CANSM217:

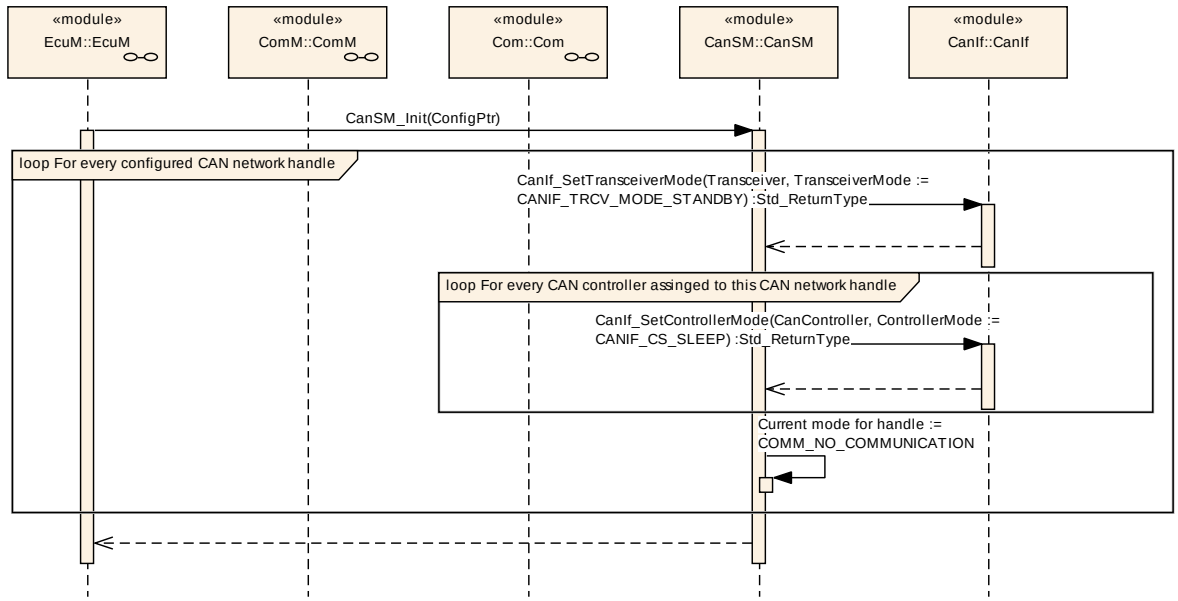

Figure 9-1: Network mode state machine – initialization

CANSM219:

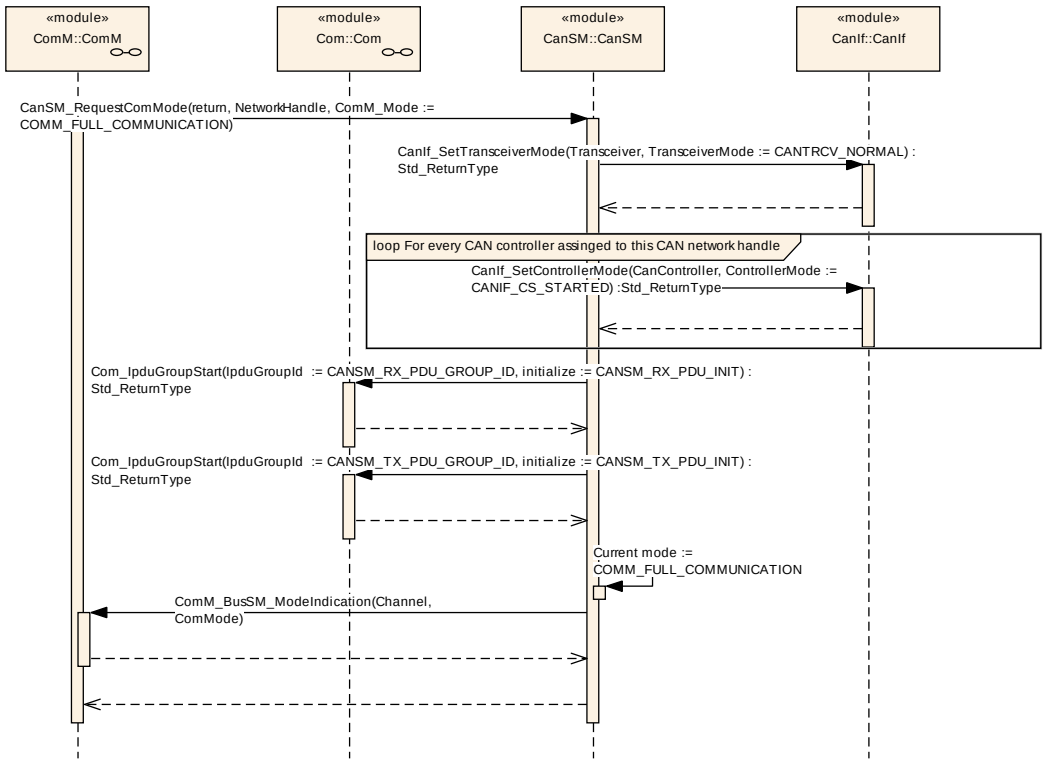

Figure 9-2: Network mode state machine – transition from no to full communication

CANSM218:

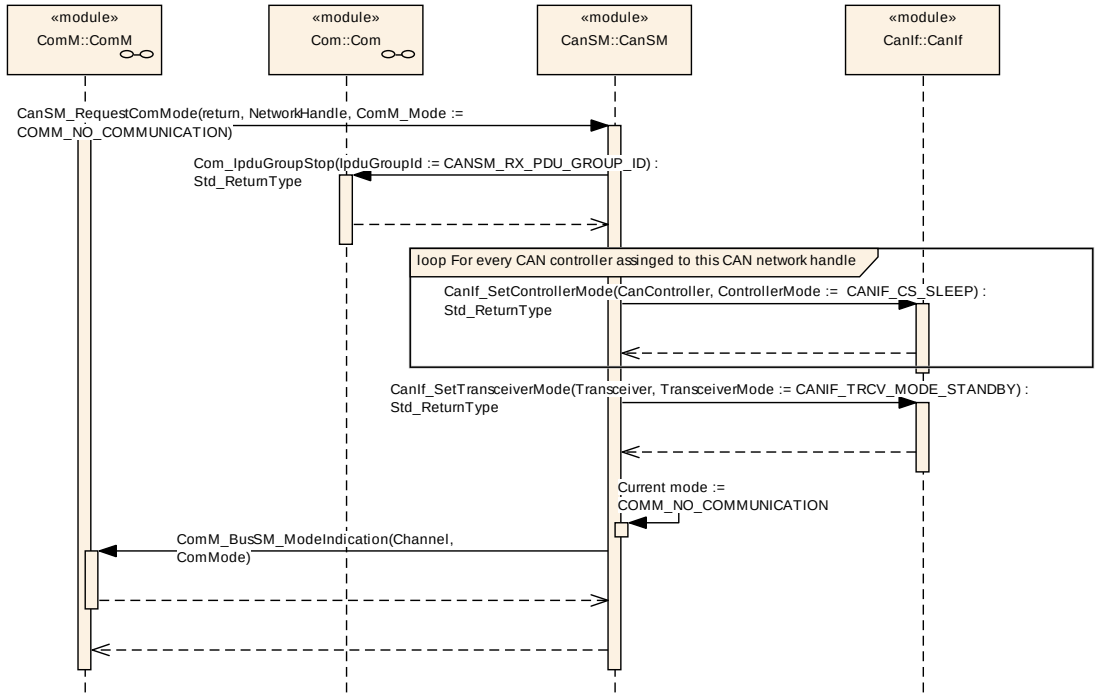

Figure 9-3: Network mode state machine – transition from silent to no communication

CANSM231:

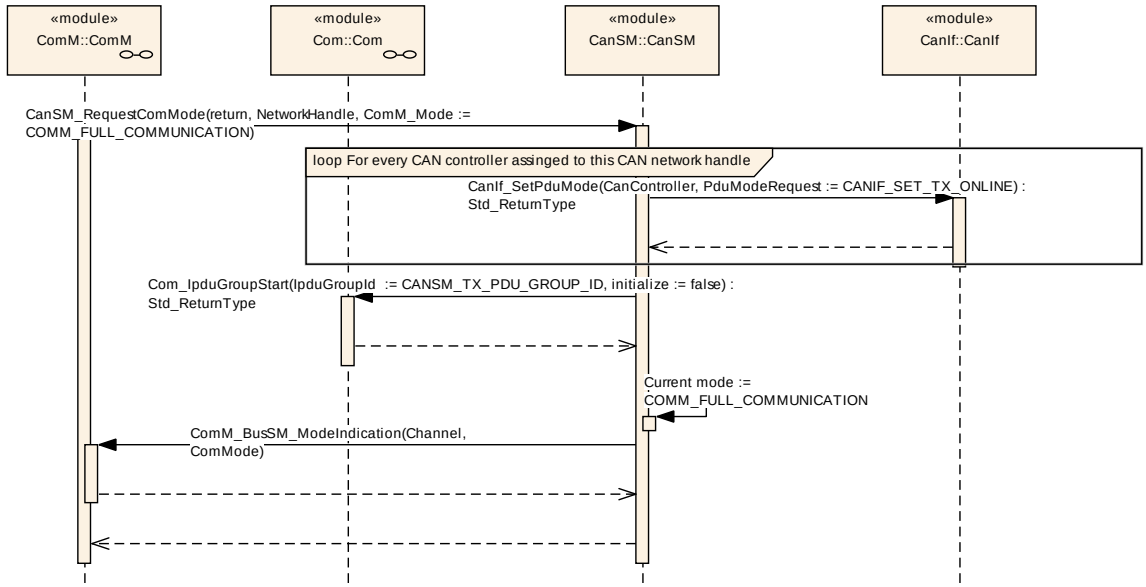

Figure 9-4: Network mode state machine – transition from silent to full communication

CANSM232:

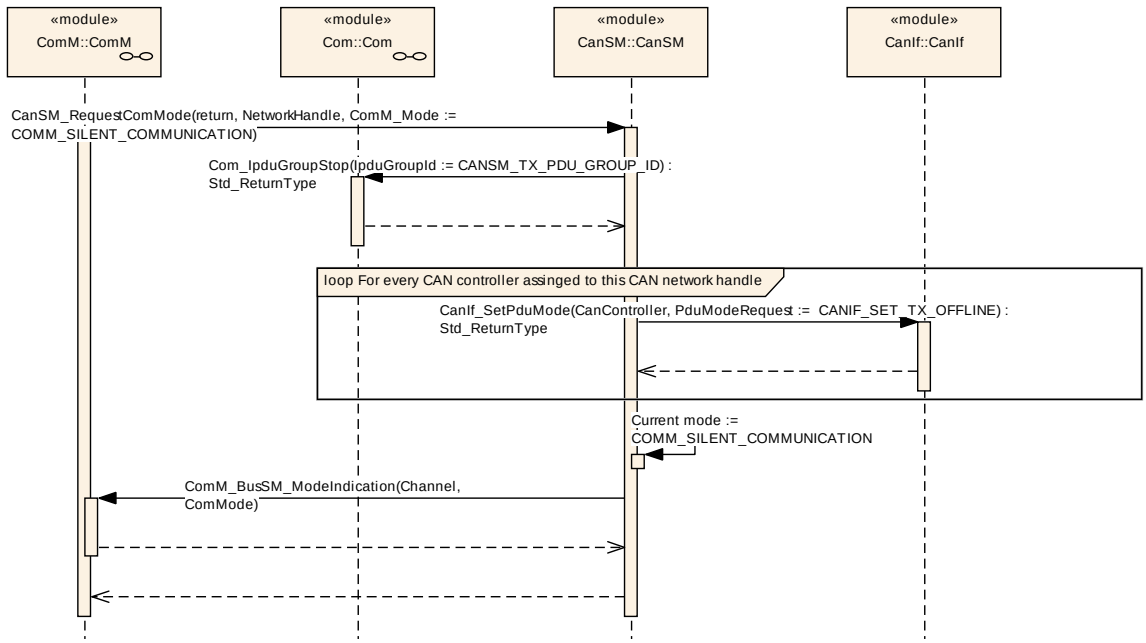

Figure 9-5: Network mode state machine – transition from full to silent to communication

9.2 Bus-off recovery state machine sequences

CANSM221:

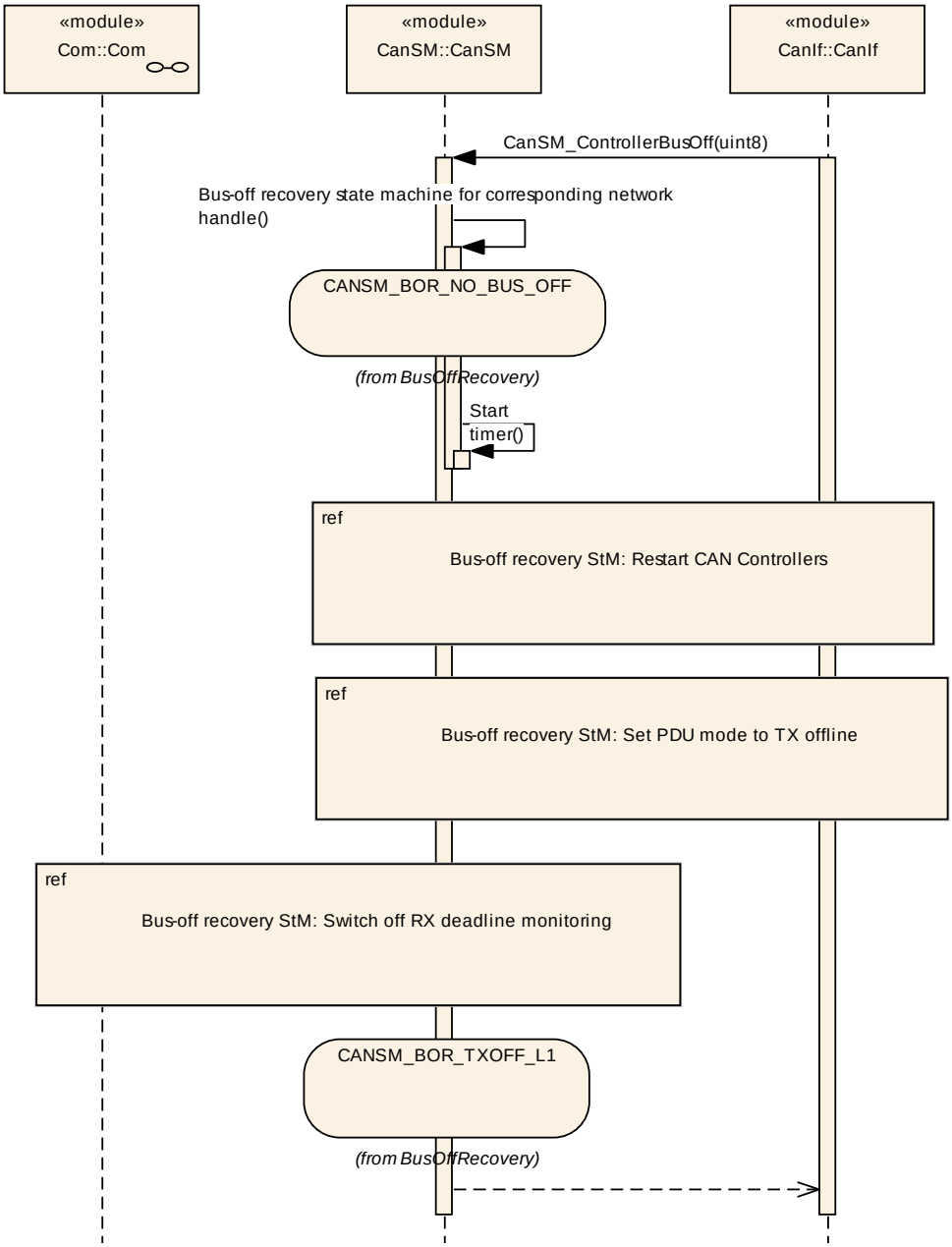

Figure 9-6: Bus-off recovery state machine – first bus-off event

CANSM222:

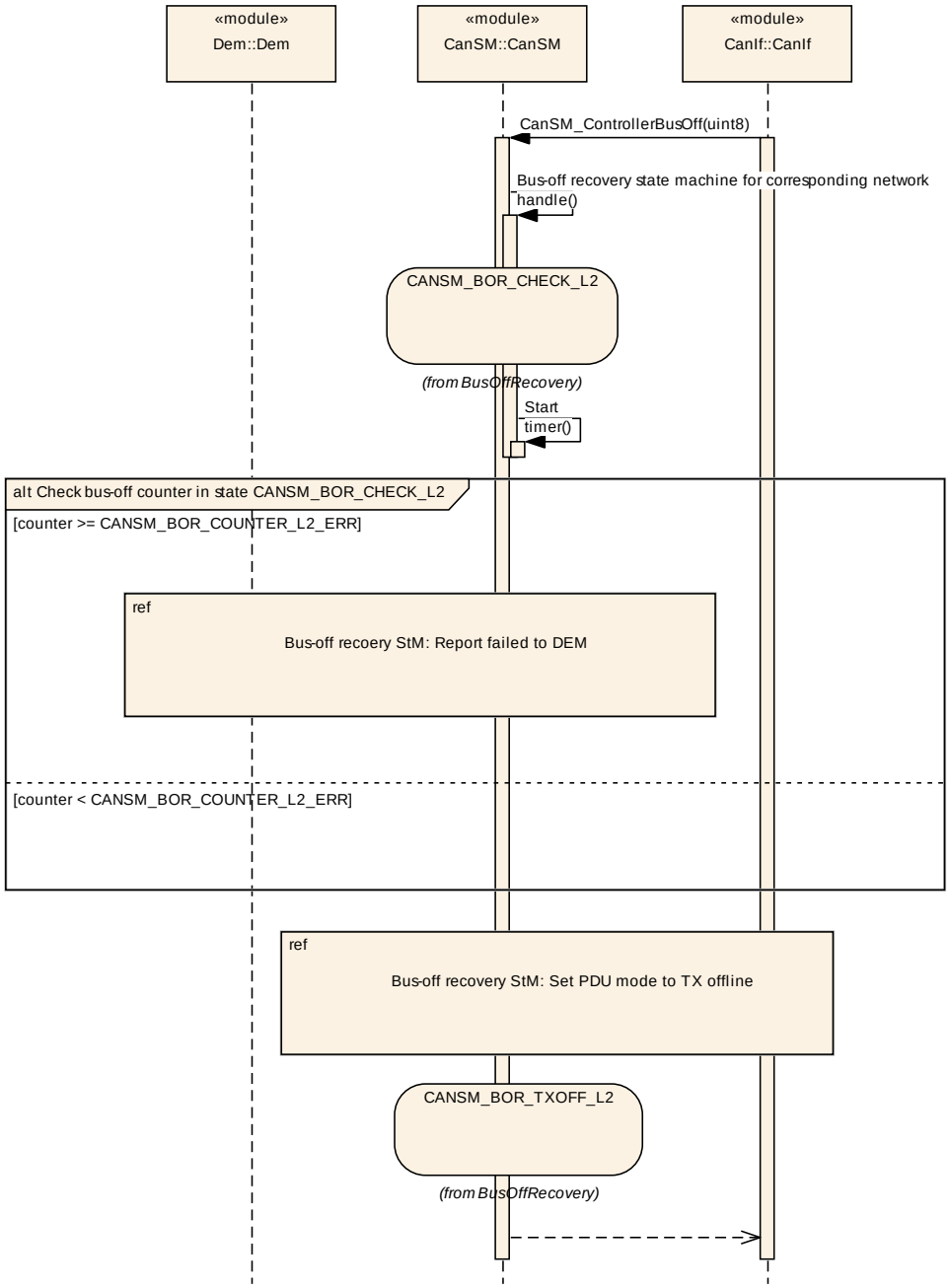

Figure 9-7: Bus-off recovery state machine – bus-off event with DEM error

CANSM223:

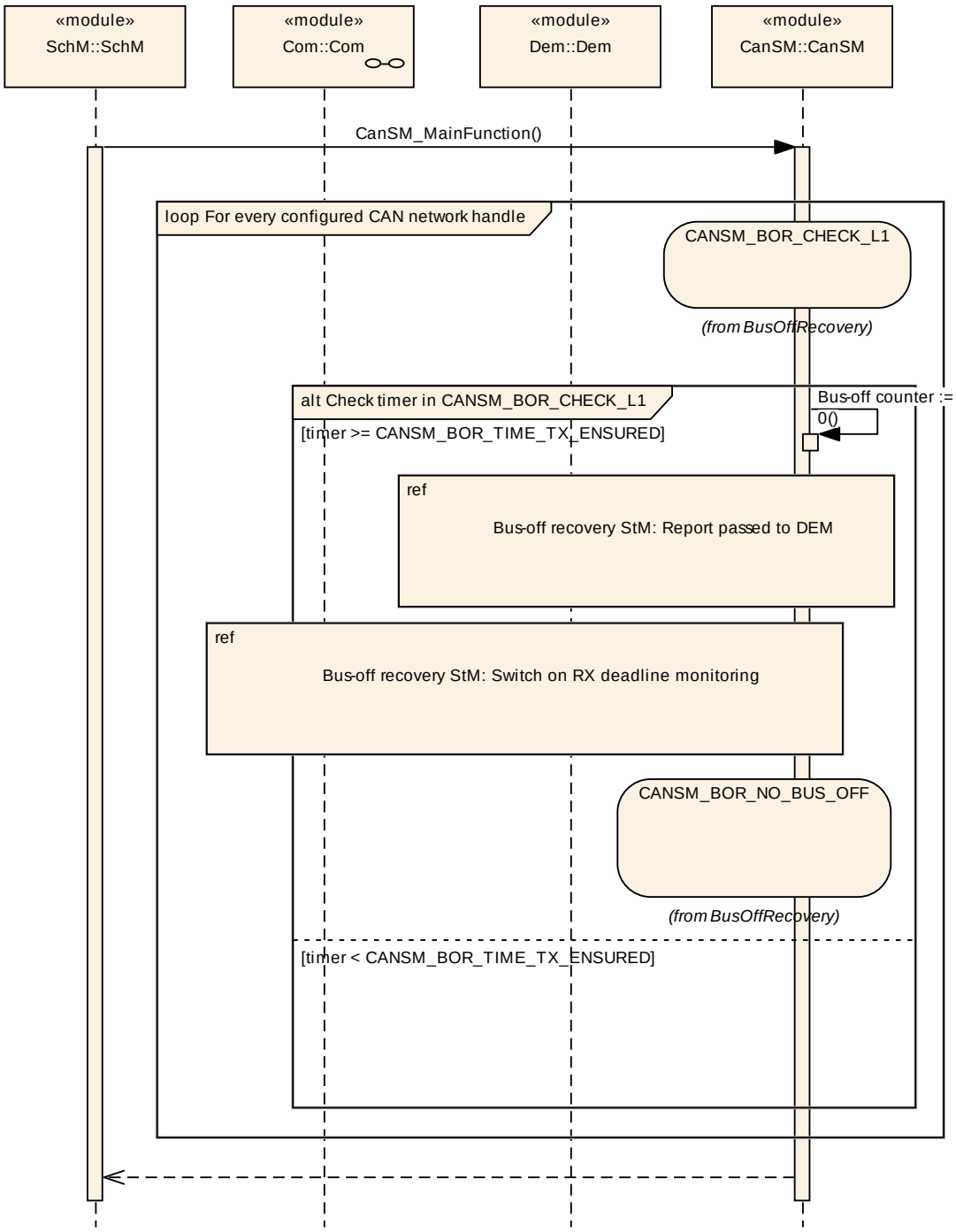

Figure 9-8: Bus-off recovery state machine – return to CANSM_BOR_NO_BUS_OFF

CANSM224:

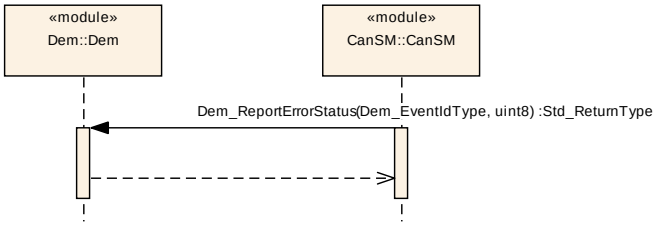

Figure 9-9: Bus-off recovery state machine – report passed to DEM

CANSM226:

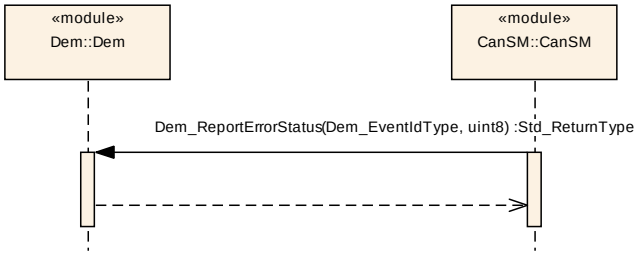

Figure 9-10: Bus-off recovery state machine – report failed to DEM

CANSM225:

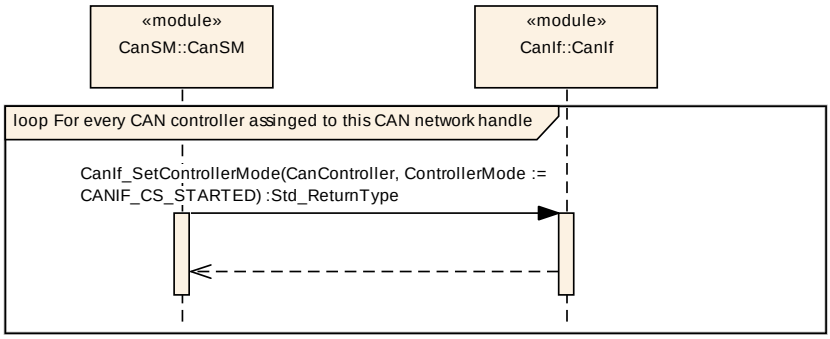

Figure 9-11: Bus-off recovery state machine – restart CAN-controllers

CANSM227:

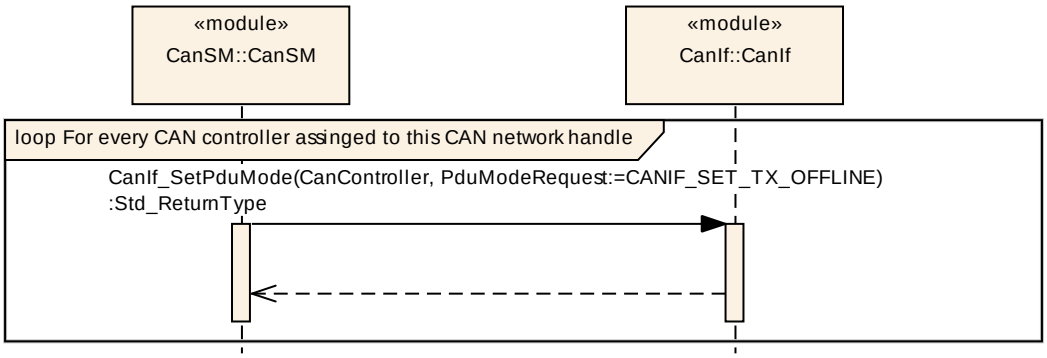

Figure 9-12: Bus-off recovery state machine – set PDU mode to TX offline

CANSM228:

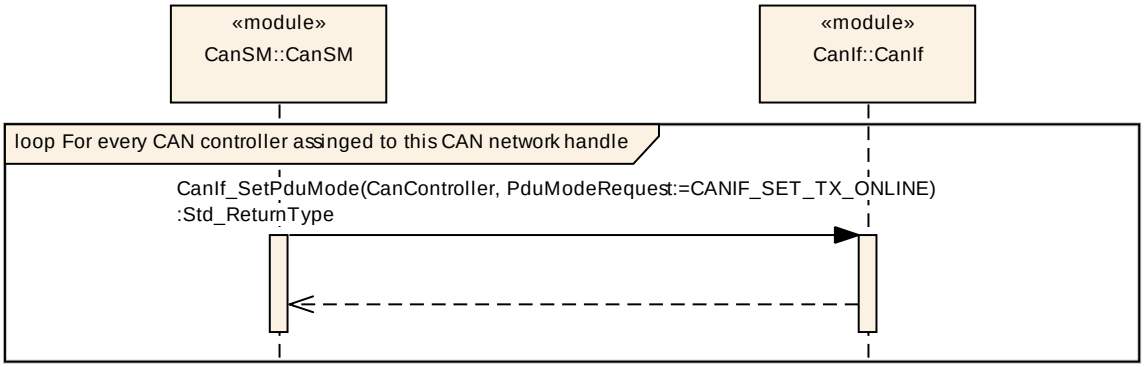

Figure 9-13: Bus-off recovery state machine – set PDU mode to TX online

CANSM229:

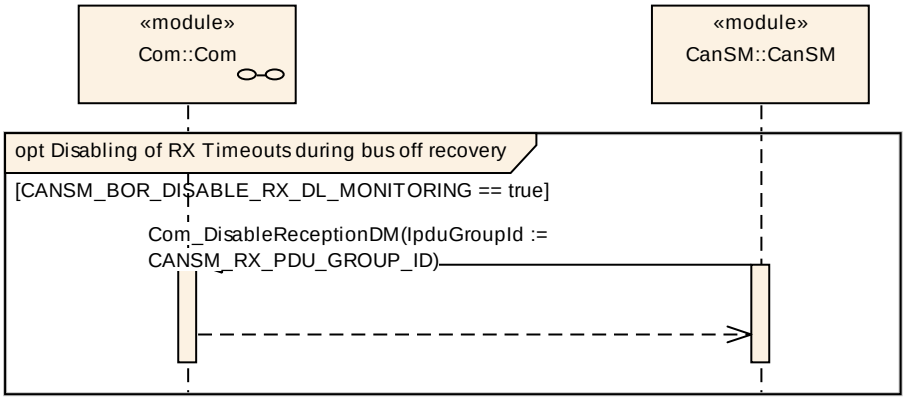

Figure 9-14: Bus-off recovery state machine – switch off RX deadline monitoring

CANSM230:

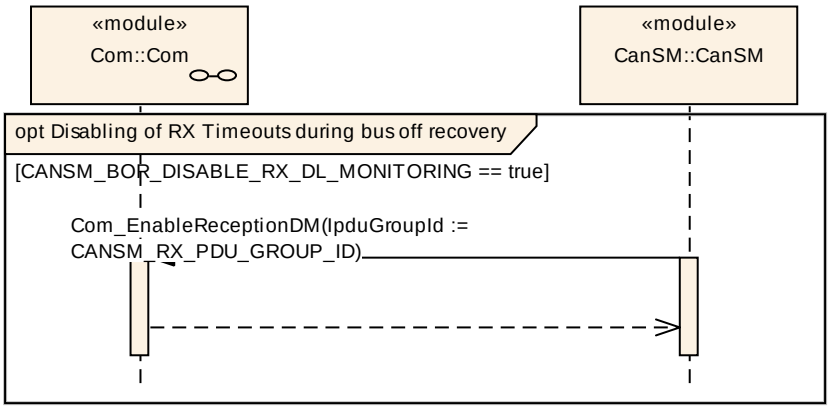

Figure 9-15: Bus-off recovery state machine – switch on RX deadline monitoring

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 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	

SWS Item	**CANSM133 :**		
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	VARIANT-PRE-COMPILE, VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Local		

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

		of each CAN network

10.2.3 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	**CANSM132 :**		
Name	CanSMBorCounterL2Err {CANSM_BOR_COUNTER_L2_ERR}		
Description	This threshold defines at which bus-off-counter value the bus-off recovery state machine shall report a failed production error state to the DEM.		
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	**CANSM175 :**		
Name	CanSMBorDisableRxDlMonitoring {CANSM_BOR_DISABLE_RX_DL_MONITORING}		
Description	If "On" the CanSM disables the deadline monitoring of RX PDU Groups during bus-off recovery to avoid timeouts (False: Off, True: On).		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE, VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
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	0.0 .. 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	0.0 .. 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	0.0 .. 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	**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	**CANSM242 :**		
Name	CanSMRxPduInit {CANSM_RX_PDU_INIT}		
Description	This parameter shall configure the initialize parameter for the configured RX PDU group, when the CanSM module references the API Com_IpduGroupStart in the transition from no communication to full communication.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Local		

SWS Item	**CANSM243 :**		
Name	CanSMTxPduInit {CANSM_TX_PDU_INIT}		
Description	This parameter shall configure the initialize parameter for the configured TX PDU group, when the CanSM module references the API Com_IpduGroupStart in the transition from no communication to full communication.		
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	**CANSM138 :**		
Name	CanSMRxPduGroupId {CANSM_RX_PDU_GROUP_ID}		
Description	ID of the RX PDU Group assigned to the configured network handle. Reference to one of the PDU group Ids of the COM configuration.		
Multiplicity	1		
Type	Reference to ComIPduGroup		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: Local dependency: Com		

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 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		

SWS Item	**CANSM139 :**		
Name	CanSMTxPduGroupId {CANSM_TX_PDU_GROUP_ID}		
Description	ID of the TX PDU Group assigned to the configured network handle. Reference to one of the PDU group Ids of the COM configuration.		
Multiplicity	1		
Type	Reference to ComIPduGroup		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: Local dependency: Com		

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

10.2.4 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 CanController		
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: Can		

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 [17] Figure 4.1 and Figure 7.1).

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

11 Changes to Release 2.1

The CanSM is a new module in the AUTOSAR BSW module stack. The whole content of this SWS is new in reference to release 2.1.

11.1 Deleted SWS Items

SWS Item	*Rationale*

11.2 Replaced SWS Items

SWS Item of Release 1	*replaced by SWS Item*	*Rationale*

11.3 Changed SWS Items

SWS Item	*Rationale*

11.4 Added SWS Items

SWS Item	*Rationale*