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

Document Version	2.2.0
Document Status	Final
Part of Release	4.0
Revision	3

Document Change History

Date	Version	Changed by	Change Description
24.11.2011	2.2.0	AUTOSAR Administration	• Added new handling to support partial networking • Changed handling for bus deinitialisation according to AR3.x behaviour • New API and handling to change the baudrate of a CAN network • Changed handling for bus-off recovery and related production error report • Comprehensive revision of all state machine diagrams and SWS-ID-items • Changed classification of production errors and development errors • Solve conflicts of SWS-ID items with the conformance test specification
21.10.2010	2.1.0	AUTOSAR Administration	• Configurable Bus-Off revovery with CAN TX confirmation instead of time based recovery • Control of PDU channel modes completely shifted from CanIf to CanSM module
30.11.2009	2.0.0	AUTOSAR Administration	• VMM/AMM Concept related changes (PDU group control shifted to BswM) • Asynchronous handling of CAN network mode transitions (consideration of CAN Transceiver and CAN controller mode notifications) • Solution of Document Improvement issues reported by TO (e. g. split up of non atomic software requirements, textual requirements instead of only a state diagram)

Document Change History			
Date	**Version**	**Changed by**	**Change Description**
			• Legal disclaimer revised
23.06.2008	1.0.1	AUTOSAR Administration	Legal disclaimer revised
13.11.2007	1.0.0	AUTOSAR Administration	Initial Release

Advice for users

AUTOSAR specifications 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 specifications for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such specifications, 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 a member of the Communication Service Layer. It interacts with the Communication Hardware Abstraction Layer and the System Service Layer.

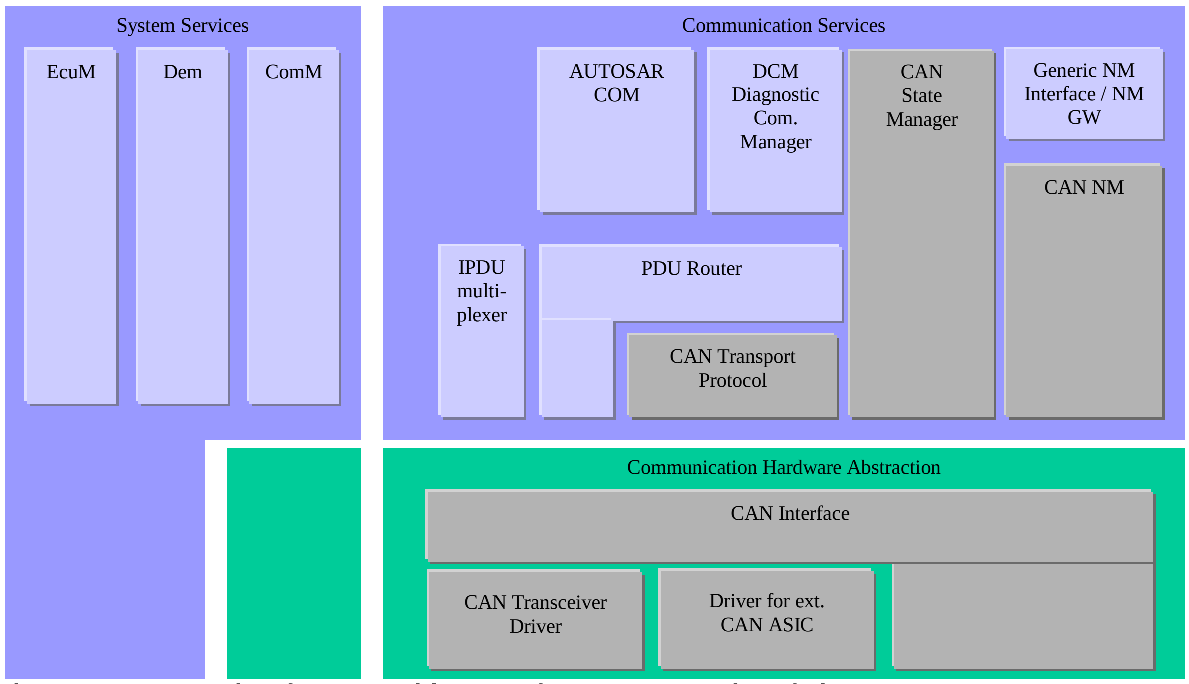

Figure 1-1: Layered Software Architecture from CanSM point of view

2 Acronyms and abbreviations

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

3 Related documentation

3.1 Input documents

[1] List of Basic Software Modules
AUTOSAR_TR_BSWModuleList.pdf

[2] Layered Software Architecture
AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf

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

[4] Specification of ECU Configuration
AUTOSAR_TPS_ECUConfiguration.pdf

[5] Specification of Standard Types
AUTOSAR_SWS_StandardTypes.pdf

[6] Specification of Communication Stack Types
AUTOSAR_SWS_CommunicationStackTypes.pdf

[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_CANTransceiverDriver.pdf

[10] Specification of Communication Manager
AUTOSAR_SWS_COMManager.pdf

[11] Specification of ECU State Manager

AUTOSAR_SWS_ECUStateManager.pdf

[12] Specification of Diagnostics Event Manager
AUTOSAR_SWS_DiagnosticEventManager.pdf

[13] Specification of CAN Interface
AUTOSAR_SWS_CANInterface.pdf

[14] Specification of BSW Scheduler
AUTOSAR_SWS_BSW_Scheduler.pdf

[15] Specification of Development Error Tracer
AUTOSAR_SWS_DevelopmentErrorTracer.pdf

[18] Specification of Basic Software Mode Manager
AUTOSAR_SWS_BSWModeManager.pdf

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

[20] Specification of Diagnostic Communication Manager,
AUTOSAR_SWS_DiagnosticCommunicationManager.pdf

3.2 Related standards and norms

None

4 Constraints and assumptions

4.1 Limitations

The CanSM module can be used for CAN communication only. Its task is to operate with the CanIf module 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 module can be used for all domain applications whenever the CAN protocol is used.

5 Dependencies to other modules

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

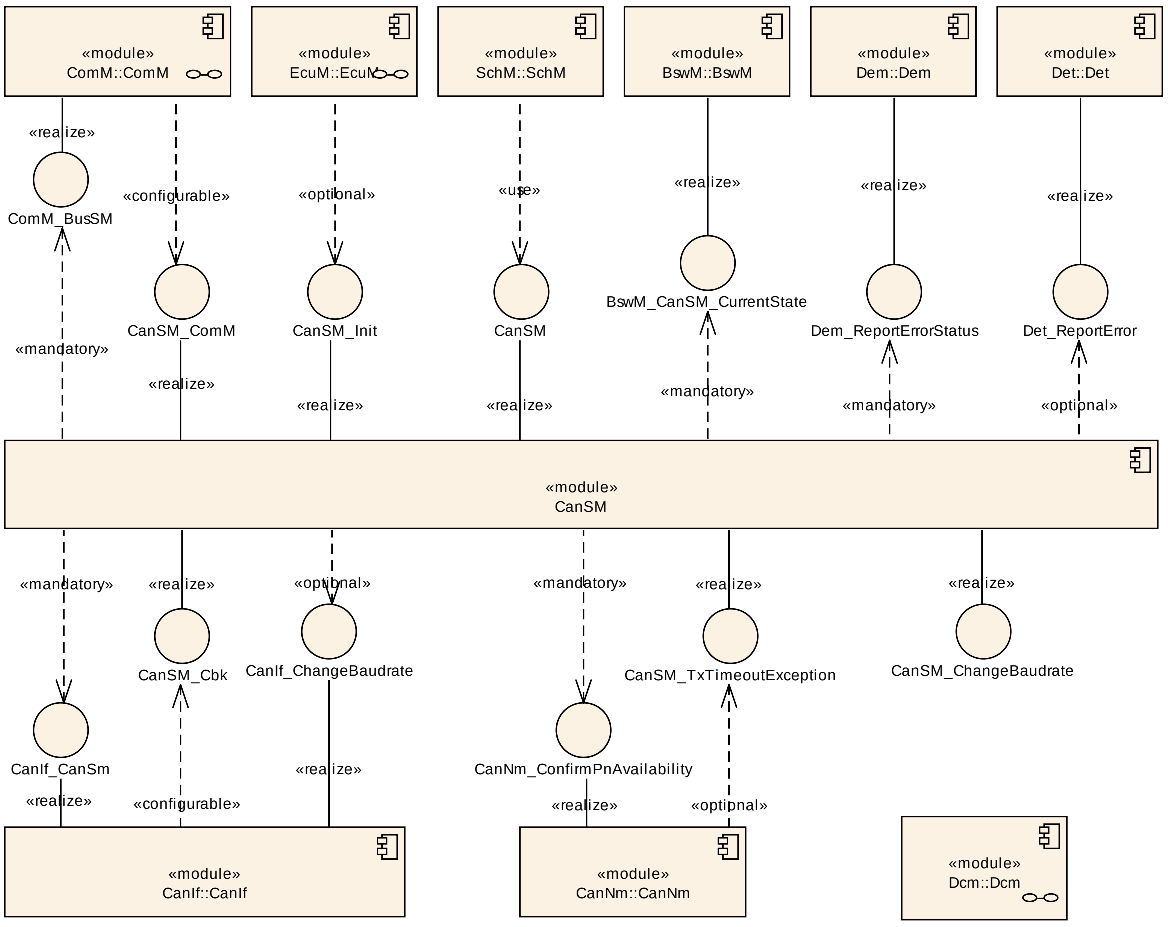

Figure 5-1: Module dependencies of the CanSM module

5.1 ECU State Manager (EcuM)

The EcuM module initializes the CanSM module (refer to [11] for a detailed specification of this module).

5.2 BSW Scheduler (SchM)

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

5.3 Communication Manager (ComM)

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

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

5.4 CAN Interface (CanIf)

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

The CanIf module notifies the CanSM module about peripheral events.

5.5 Diagnostic Event Manager (DEM)

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

5.6 Basic Software Mode Manager (BswM)

The CanSM need to notify bus specific mode changes to the BswM module (refer to [18] for a 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 and shall handle notified CanNm timeout exceptions in case of partial networking (ref. to [19] for a detailed specification of this module).

5.8 Diagnostic Communication Manager (Dcm)

The CanSM module provides an API, which can be used by the Dcm module to request a baud rate change of a CAN network (ref. to [20] for a detailed specification of this module).

5.9 Development Error Tracer (DET)

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

5.10 File structure

5.10.1 Code file structure

This specification does not define the code file structure completely. Nevertheless, the code-file structure shall include the following files:

[CANSM361] ⌈The CanSM module shall provide a file CanSM_Lcfg.c that contains all link time configurable parameters of the module.⌋()

[CANSM362] ⌈The CanSM module shall provide a file CanSM_PBcfg.c that contains all post build time configurable parameters of the module.⌋()

5.10.2 Header file structure

[CANSM008] ⌈The header file CanSM.h shall export CanSM module specific types and the API of the CanSM module, 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

[CANSM174] ⌈The header file CanSM.h shall include the header file ComM.h.⌋()

Rationale: Some APIs of the CanSM use type definitions of the ComM module.

[CANSM253] ⌈The header file CanSM_EcuM.h shall export the init function of the CanSM.⌋()

Rationale: The header file CanSM_EcuM.h is used for the integration of the CanSM module into the EcuM module.

[CANSM009] ⌈The header file CanSM_ComM.h shall export the CanSM module's API dedicated to the ComM module.⌋()

[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 5.10.1 above) and shall contain pre-compile parameters, which are not declared as "const" parameter, but as defines.⌋(BSW00344, BSW0404, BSW00345, BSW00381, BSW00412)

[CANSM011] ⌈The header file CanSM_Cbk.h shall declare the callback notification functions of the CanSM module.⌋()

[CANSM013] ⌈The CanSM module (CanSM.c) shall reference its header file CanSM.h.⌋()

Rationale: -to make its type definitions available

[CANSM254] ⌈The CanSM module (CanSM.c) shall reference its header file CanSM_Cfg.h.⌋()

Rationale: -to make its configuration parameters available

[CANSM014]⌈ The CanSM module (CanSM.c) shall include the header file Dem.h.⌋()

Rationale: The functions declared in Dem.h are used to report production errors.

[CANSM015] ⌈The CanSM module (CanSM.c) shall include the header file Det.h.⌋(BSW171)

Rationale: The functions declared in Det.h are used to report development errors.

[CANSM016] ⌈The CanSM module (CanSM.c) shall include the header file MemMap.h.⌋(BSW00436)

Rationale: MemMap.h makes it possible to map the code and the data of the CanSM module into specific memory sections.

[CANSM017] ⌈The CanSM module (CanSM.c) shall include the header file CanIf.h.⌋()

Rationale: The API of the CanIf module is needed for peripheral control.

[CANSM191] ⌈The CanSM module (CanSM.c) shall include the header file ComM_BusSM.h.⌋()

Rationale: The file ComM_BusSM.h provides the API of the ComM module, which is exclusively intended for the bus state managers.

[CANSM347] ⌈The header file CanSM_BswM.h shall export the interfaces, which are dedicated to the BswM module.⌋()

[CANSM348] ⌈The CanSM module (CanSM.c) shall include the header file CanSM_BswM.h.⌋()

[CANSM547] ⌈The header file CanSM_Dcm.h shall export the interfaces, which are dedicated to the Dcm module.⌋()

[CANSM548] ⌈The CanSM module (CanSM.c) shall include the interface `CanNm_ConfirmPnAvailability` (CanNm_ConfirmPnAvailability.h) of the CanNm module.⌋()

[CANSM549] ⌈The header file `CanSM_TxTimeoutException.h` shall provide the callback function `CanSM_TxTimeoutException` as optional interface to the CanNm module.⌋()

5.10.3 Version check

[CANSM025] ⌈The CanSM module shall perform Inter Module Checks to avoid integration of incompatible files. The imported included files shall be checked by preprocessing directives.⌋(BSW167, BSW004)

The following version numbers shall be verified:
- <MODULENAME>_AR_RELEASE_MAJOR_VERSION
- <MODULENAME>_AR_RELEASE_MINOR_VERSION

Where <MODULENAME> is the module short name of the other (external) modules which provide header files included by the CanSM module.

If the values are not identical to the expected values, an error shall be reported.

6 Requirements traceability

Requirement	Satisfied by
-	CANSM363
-	CANSM530
-	CANSM511
-	CANSM414
-	CANSM573
-	CANSM441
-	CANSM013
-	CANSM443
-	CANSM532
-	CANSM499
-	CANSM017
-	CANSM453
-	CANSM014
-	CANSM496
-	CANSM533
-	CANSM521
-	CANSM557
-	CANSM372
-	CANSM463
-	CANSM418
-	CANSM347
-	CANSM374
-	CANSM009
-	CANSM513
-	CANSM410
-	CANSM375
-	CANSM483
-	CANSM187
-	CANSM510
-	CANSM431
-	CANSM536
-	CANSM254
-	CANSM266
-	CANSM563
-	CANSM538
-	CANSM365
-	CANSM432
-	CANSM400

-	CANSM566
-	CANSM235
-	CANSM501
-	CANSM461
-	CANSM412
-	CANSM451
-	CANSM468
-	CANSM437
-	CANSM490
-	CANSM189
-	CANSM371
-	CANSM411
-	CANSM072
-	CANSM543
-	CANSM348
-	CANSM436
-	CANSM558
-	CANSM433
-	CANSM561
-	CANSM398
-	CANSM480
-	CANSM401
-	CANSM278
-	CANSM449
-	CANSM447
-	CANSM479
-	CANSM403
-	CANSM419
-	CANSM243
-	CANSM008
-	CANSM535
-	CANSM182
-	CANSM183
-	CANSM450
-	CANSM184
-	CANSM528
-	CANSM011
-	CANSM569
-	CANSM465
-	CANSM284
-	CANSM547
-	CANSM456

-	CANSM460
-	CANSM504
-	CANSM452
-	CANSM523
-	CANSM188
-	CANSM469
-	CANSM471
-	CANSM455
-	CANSM534
-	CANSM470
-	CANSM505
-	CANSM489
-	CANSM442
-	CANSM556
-	CANSM420
-	CANSM462
-	CANSM430
-	CANSM459
-	CANSM562
-	CANSM516
-	CANSM366
-	CANSM574
-	CANSM425
-	CANSM500
-	CANSM548
-	CANSM429
-	CANSM492
-	CANSM555
-	CANSM550
-	CANSM506
-	CANSM477
-	CANSM427
-	CANSM484
-	CANSM428
-	CANSM309
-	CANSM370
-	CANSM417
-	CANSM367
-	CANSM503
-	CANSM560
-	CANSM554
-	CANSM397

-	CANSM572
-	CANSM438
-	CANSM423
-	CANSM475
-	CANSM540
-	CANSM512
-	CANSM487
-	CANSM509
-	CANSM527
-	CANSM508
-	CANSM525
-	CANSM244
-	CANSM517
-	CANSM445
-	CANSM518
-	CANSM413
-	CANSM402
-	CANSM458
-	CANSM444
-	CANSM377
-	CANSM426
-	CANSM539
-	CANSM549
-	CANSM399
-	CANSM473
-	CANSM537
-	CANSM474
-	CANSM415
-	CANSM434
-	CANSM464
-	CANSM435
-	CANSM497
-	CANSM493
-	CANSM488
-	CANSM529
-	CANSM396
-	CANSM238
-	CANSM186
-	CANSM466
-	CANSM502
-	CANSM507
-	CANSM491

-	CANSM364
-	CANSM514
-	CANSM310
-	CANSM167
-	CANSM485
-	CANSM369
-	CANSM174
-	CANSM472
-	CANSM253
-	CANSM571
-	CANSM467
-	CANSM448
-	CANSM069
-	CANSM368
-	CANSM457
-	CANSM362
-	CANSM494
-	CANSM191
-	CANSM360
-	CANSM190
-	CANSM567
-	CANSM282
-	CANSM440
-	CANSM515
-	CANSM421
-	CANSM495
-	CANSM524
-	CANSM526
-	CANSM486
-	CANSM546
-	CANSM476
-	CANSM446
-	CANSM454
-	CANSM478
-	CANSM531
-	CANSM541
-	CANSM439
-	CANSM395
-	CANSM361
-	CANSM416
-	CANSM542
-	CANSM376

-	CANSM568
BSW003	CANSM024
BSW00308	CANSM999
BSW00309	CANSM999
BSW00314	CANSM999
BSW00323	CANSM071
BSW00326	CANSM999
BSW00333	CANSM064
BSW00336	CANSM999
BSW00338	CANSM028
BSW00339	CANSM074
BSW00341	CANSM999
BSW00344	CANSM010
BSW00345	CANSM010
BSW00347	CANSM999
BSW00353	CANSM999
BSW00358	CANSM023
BSW00359	CANSM064
BSW00360	CANSM999
BSW00361	CANSM999
BSW00375	CANSM999
BSW00376	CANSM065
BSW00377	CANSM999
BSW00381	CANSM010
BSW00386	CANSM071, CANSM028
BSW00395	CANSM999
BSW004	CANSM025
BSW00404	CANSM023
BSW00405	CANSM023
BSW00406	CANSM023, CANSM179
BSW00407	CANSM024
BSW00412	CANSM010
BSW00414	CANSM023
BSW00416	CANSM999
BSW00417	CANSM999
BSW00422	CANSM522, CANSM498
BSW00423	CANSM999
BSW00425	CANSM065
BSW00426	CANSM999
BSW00427	CANSM999
BSW00428	CANSM999
BSW00429	CANSM999

BSW00431	CANSM999
BSW00432	CANSM999
BSW00433	CANSM999
BSW00434	CANSM999
BSW00435	CANSM999
BSW00436	CANSM016
BSW00437	CANSM999
BSW00439	CANSM999
BSW00440	CANSM999
BSW005	CANSM999
BSW01142	CANSM062, CANSM063
BSW01144	CANSM424
BSW01146	CANSM064
BSW0404	CANSM010
BSW0405	CANSM023
BSW0424	CANSM065
BSW09080	CANSM062, CANSM063
BSW09081	CANSM062
BSW09083	CANSM062
BSW09084	CANSM063
BSW101	CANSM023
BSW161	CANSM999
BSW162	CANSM999
BSW167	CANSM025
BSW168	CANSM999
BSW170	CANSM999
BSW171	CANSM015
ref.toCANSM419	CANSM385

According to [3] (General BSW Requirements):

Requirement	***Satisfied by***
[BSW00344] Reference to link-time configuration	Chapter 5.10, CANSM010
[BSW0404] Reference to post build time configuration	Chapter 5.10, CANSM010
[BSW0405] Reference to multiple configuration sets	CANSM023, chapter 8.2.1
[BSW00345] Pre-compile time configuration	Chapter 5.10, CANSM010, CANSM123_Conf, CANSM126_Conf, CANSM127_Conf
[BSW159] Tool based configuration	Changed to not applicable during SW improvement (CANSM155 deleted)
[BSW167] Static configuration checking	CANSM025

[BSW171] Configurability of optional functionality	CANSM015, CANSM133_Conf
[BSW170] Data for reconfiguration of SW-components	Not applicable (requirement on SWC-module)
[BSW00380] Separate C-Files for configuration parameters	Chapter 5.10
[BSW00419] Separate C-Files for pre-compile time configuration parameters	Chapter 5.10
[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	CANSM161_Conf, CANSM137_Conf, CANSM141_Conf
[BSW00384] List dependencies to other modules	Chapter 5
[BSW00387] Specify the configuration class of callback function	Chapter 8.3.6
[BSW00388] Introduce containers	CANSM123_Conf, CANSM126_Conf, CANSM127_Conf
[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 range	Chapter 10.2
[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	Chapter 10.2.1
[BSW00438] Post Build Configuration Data Structure	chapter (TODO)
[BSW00402] Published information	Chapter 10.3
[BSW00375] Notification of wake-up reason	Not applicable (no wake up interrupt)
[BSW101] Initialization interface	CANSM023

[BSW00416] Sequence of Initialization	Not applicable (CanSM module cannot influence the sequence for initialization)
[BSW00406] Check module initialization	CANSM023 CANSM179
[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 functions	Not applicable (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 deinitialization function)
[BSW00337] Classification of errors	Chapter 7.3
[BSW00338] Detection and Reporting of development errors	Chapter 7.4, CANSM028
[BSW00369] Do not return development error codes via API	Chapter 7.7
[BSW00339] Reporting of production relevant errors and exceptions	CANSM074
[BSW00422] Pre–de–bouncing of production relevant error status	CANSM498, CANSM520, CANSM522
[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 7.3
[BSW00385] List possible error notifications	chapter 7.3
[BSW00386] Configuration for detecting an error	Chapter 7.4, 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 5.10.2
[BSW164] Implementation of interrupt service routines	Chapter 7.7
[BSW00325] Runtime of interrupt service routines	Chapter 7.7
[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	Changed to not applicable during SW improvement (CANSM155 deleted)
[BSW007] HIS MISRA C	Chapter 7.7
[BSW00300] Module naming convention	Chapter 7.7
[BSW00413] Accessing instances of BSW modules	Chapter 7.7
[BSW00347] Naming separation of different instances of BSW drivers	Not applicable (not in scope of this spec)
[BSW00305] Self–defined data types naming convention	Chapter 8.2
[BSW00307] Global variables naming convention	Chapter 7.7
[BSW00310] API naming convention	Chapter 8.3
[BSW00373] Main processing function naming convention	Chapter 8.5.1
[BSW00327] Error values naming convention	Chapter 7.3
[BSW00335] Status values naming convention	Chapter 8.2
[BSW00350] Development error detection keyword	Chapter 7.4
[BSW00408] Configuration parameter naming convention	Chapter 10.2
[BSW00410] Compiler switches shall have defined values	Chapter 10.2

[BSW00411] Get version info keyword	Chapter 8.3.2 Chapter 10.2
[BSW00346] Basic set of module files	Chapter 5.10
[BSW158] Separation of configuration from implementation	Chapter 5.10
[BSW00314] Separation of interrupt frames and service routines	Not applicable (not in scope of this spec)
[BSW00370] Separation of callback interface from API	Chapter 5.10
[BSW00435] Header File Structure for the Basic Software Scheduler	Not applicable (not in scope of this spec)
[BSW00436] Module Header File Structure for the Basic Software Memory Mapping	CANSM016
[BSW00348] Standard type header	Chapter 5.10
[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 5.10
[BSW00302] Limit exported information	Chapter 5.10
[BSW00328] Avoid duplication of code	Chapter 7.7
[BSW00312] Shared code shall be reentrant	Chapter 7.7
[BSW006] Platform independency	Chapter 7.7
[BSW00357] Standard API return type [	Chapter 8.3
[BSW00377] Module specific API return types	Not applicable (not used)
[BSW00304] AUTOSAR integer data types	Chapter 7.7
[BSW00355] Do not redefine AUTOSAR integer data types	Chapter 7.7
[BSW00378] AUTOSAR boolean type	Chapter 7.7
[BSW00306] Avoid direct use of compiler and platform specific keywords [	Chapter 7.7
[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 8.3
[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	Chapter 7.7
[BSW00330] Usage of macros / inline functions instead of functions	Chapter 7.7
[BSW00331] Separation of error and status values	Chapter 7.3, Chapter 8.2,
[BSW009] Module User Documentation	Chapter 7.7
[BSW00401] Documentation of multiple instances of configuration parameters	Chapter 10.2
[BSW172] Compatibility and documentation of scheduling strategy	Chapter 7.7
[BSW010] Memory resource documentation	Chapter 7.7
[BSW00333] Documentation of callback function context	CANSM064
[BSW00374] Module vendor identification	CANSM125
[BSW00379] Module identification	CANSM125
[BSW003] Version identification	CANSM125, CANSM024
[BSW00318] Format of module version numbers	CANSM125
[BSW00321] Enumeration of module version numbers	Chapter 7.7
[BSW00341] Microcontroller compatibility documentation	Not applicable (not in scope of this spec)
[BSW00334] Provision of XML file	Chapter 7.7
[BSW00439] Declaration of interrupt handlers and ISRs	Not applicable (CanSM not part of MCAL)
[BSW00405] Reference to multiple configuration sets	CANSM023
[BSW00440] Function prototype for callback functions of AUTOSAR Services	Not applicable (not in scope of this spec)
[BSW00441] Enumeration literals and #define naming convention	Chapter of CanSM_StateType
[BSW00404] Reference to post build time configuration	CANSM023

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	CANSM126_Conf
BSW01142] Control flow abstraction of CAN networks	CANSM062, CANSM063, chapter 7.2
[BSW01143] BusOff recovery time	CANSM128_Conf, CANSM129_Conf
[BSW01144] Power-On Initialization	CANSM424

[BSW01145] Management of CAN devices	chapter 7.2
[BSW01146] Bus-off recovery and error handling	Figure 7-6 CANSM064, CANSM070_Conf, CANSM343

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 upper level requirements for the ComM. These requirements are listed in following table:

Requirement	***Satisfied by***
[BSW09080] Physical channel independency	CANSM062, CANSM063, CANSM126_Conf
[BSW09081] API for requesting communication	CANSM062
[BSW09083] Support of different communication modes	CANSM062
[BSW09084] API for querying the current communication mode	CANSM063
[BSW09085] Indication of communication mode changes	chapter 7, chapter 8.6.1

7 Functional specification

This chapter specifies the different functions of the CanSM module in the AUTOSAR BSW architecture.

An ECU can have different communication networks. Each network has to be identified with an unique network handle. The ComM module 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 CanSM module.

The CanSM module is responsible for the control flow abstraction of CAN networks:

It changes the communication modes of the configured CAN networks depending on the mode requests from the ComM module.

Therefore the CanSM module uses the API of the CanIf module. The CanIf module is responsible for the control flow abstraction of the configured CAN Controllers and CAN Transceivers (the data flow abstraction of the CanIf module is not relevant for the CanSM module). Any change of the CAN Controller modes and CAN Transceiver modes will be notified by the CanIf module to the CanSM module. Depending on this notifications and state of the CAN network state machine, which the CanSM module shall implement for each configured CAN network, the CanSM module notifies the ComM and the BswM (ref. to chapter 7.2 for details).

7.1 General requirements

[CANSM266] ⌈The CanSM module shall store the latest notified current network mode with `ComM_BusSM_ModeIndication` (chapter 8.6.1) for each configured CAN network internally (ref. to CANSM126_Conf).⌋()

[CANSM284] ⌈The internally stored network modes of the CanSM module can have the values `COMM_NO_COMMUNICATION`, `COMM_SILENT_COMMUNICATION`, `COMM_FULL_COMMUNICATION`.⌋()

[CANSM428] ⌈All effects of the CanSM state machine `CANSM_BSM` (ref. to Figure 7-1), shall be operated in the context of the CanSM main function (ref. to CANSM065).⌋()

[CANSM278] ⌈If the CanSM state machine `CANSM_BSM` (ref. to Figure 7-1) is in the state `CANSM_BSM_S_NOT_INITIALIZED`, it shall deny network mode requests from the ComM module (ref. to CANSM062).⌋()

[CANSM385] ⌈If the CanSM module state machine was triggered with `T_REPEAT_MAX` (ref. to CANSM463, CANSM480, CANSM495, CANSM523, CANSM536), the CanSM module shall call the function Det_ReportError with the ErrorId parameter CANSM_E_MODE_REQUEST_TIMEOUT (ref. to chapter 7.3).⌋

[CANSM422] ⌈If the CanIf module notifies PN availability for a configured CAN Transceiver to the CanSM module with the callback function CanSM_ConfirmPnAvailability (ref. to CANSM419), 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.⌋()

[CANSM375] ⌈The CanSM module shall deny any network mode request, if the time since the last detected bus-off is lower than CanSMBorTimeL1 (ref. to CANSM128_Conf) and the bus-off counter is lower than CanSMBorCounterL1ToL2 (ref. to CANSM131_Conf).⌋()

Rationale: Block communication mode requests during bus-off recovery

[CANSM376] ⌈The CanSM module shall deny any network mode request, if the time since the last detected bus-off is lower than CanSMBorTimeL2 and the bus-off counter is greater or equal than CanSMBorCounterL1ToL2 (ref. to CANSM131_Conf).⌋()

Rationale: Block communication mode requests during bus-off recovery

[CANSM560] ⌈If no CanSMTransceiverId (ref. to CANSM137_Conf) is configured for a CAN Network, then the CanSM module shall bypass all specified CanIf_SetTrcvMode (e. g. CANSM446) calls for the CAN Network and proceed in the different state transitions as if it has got the supposed CanSM_TransceiverModeIndication already (e. g. CANSM448).⌋()

[CANSM567] ⌈If the CanSM module is requested to provide the information, if a certain baudrate is supported by a configured CAN network (ref. to CANSM126_Conf) with CanSM_CheckBaudrate (ref. to CANSM501), then the CanSM module shall reference the CanIf API function CanIf_CheckBaudrate (ref. to chapter 8.6.2) for all configured CAN controllers of the CAN network and notify, that the baud rate is supported by the CAN network, if all CanIf_CheckBaudrate calls have returned E_OK. ⌋()

[CANSM568] ⌈If the CanSM module is requested to provide the information, if a certain baud rate is supported by a configured CAN network (ref. to CANSM126_Conf) with CanSM_CheckBaudrate (ref. to CANSM501), then the CanSM module shall reference the CanIf API function CanIf_CheckBaudrate (ref. to chapter 8.6.2) for all configured CAN controllers of the CAN network and notify, that the baud rate is not supported, if not all `CanIf_CheckBaudrate` calls have returned `E_OK`. ⌋()

[CANSM572] ⌈The CanSM module shall remember for each configured CAN network the checked baud rate and the notified result of the last `CanSM_CheckBaudrate` call (ref. to CANSM567 and CANSM568). ⌋()

Rationale: This is necessary to decide, if the following `CanSM_ChangeBaudrate` (ref. to CANSM561) call is valid.

7.2 State machine for each CAN network

The following diagram specifies the behavioral state machine of the CanSM module, which shall be implemented for each configured CAN network (ref. to CANSM126_Conf).

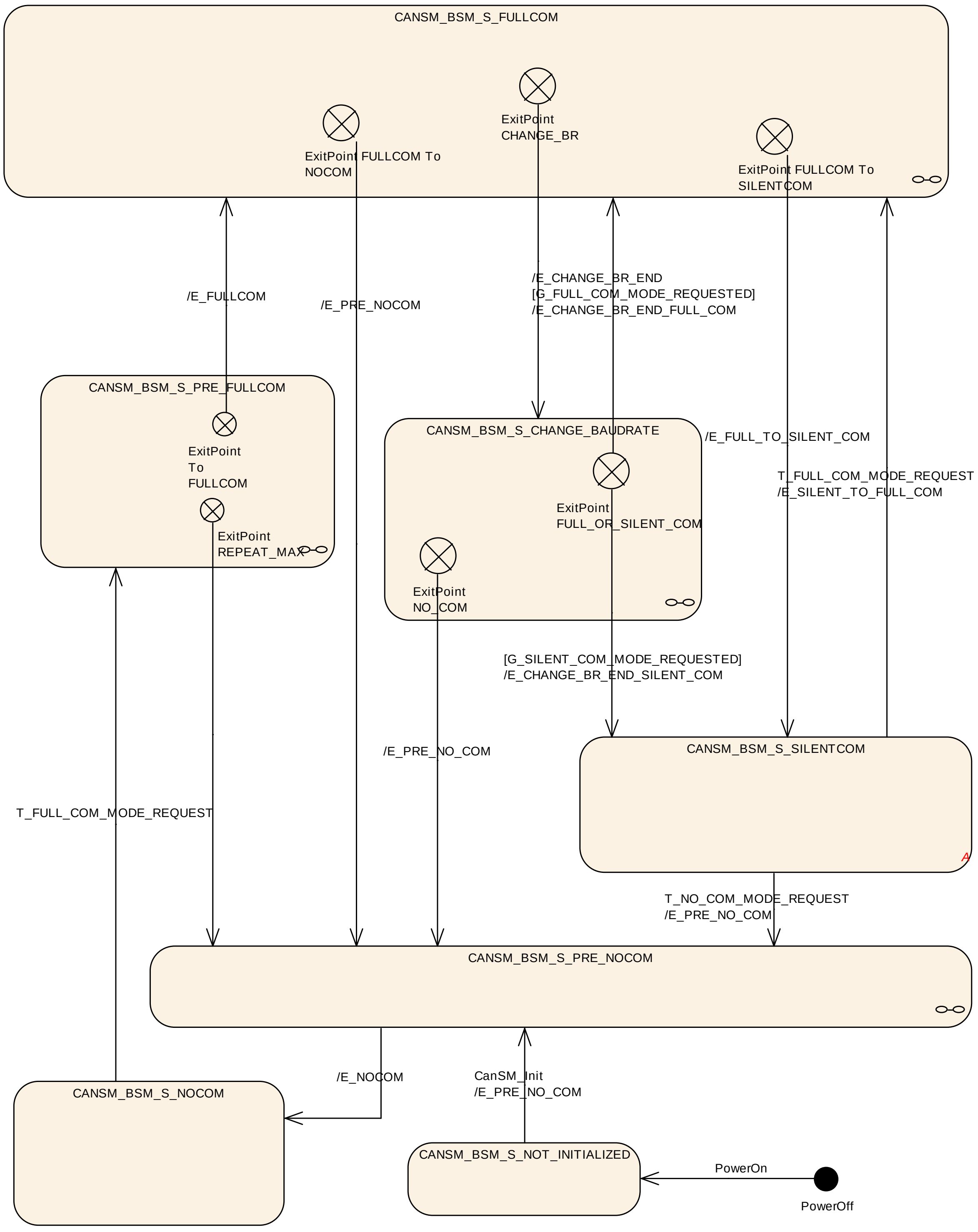

Figure 7-1: CANSM_BSM, state machine diagram for one CAN network

7.2.1 Trigger: PowerOn

[CANSM424]⌈After PowerOn the CanSM state machines (ref. to Figure 7-1) shall be in the state `CANSM_BSM_NOT_INITIALIZED.`⌋(BSW01144)

7.2.2 Trigger: CanSM_Init

[CANSM423]⌈If the CanSM module is requested with the function `CanSM_Init` (ref. to chapter 8.3.1), this shall trigger the CanSM state machines (ref. to Figure 7-1) for all configured CAN Networks (ref. to CANSM126_Conf) with the trigger `CanSM_Init.`⌋()

7.2.3 Trigger: T_FULL_COM_MODE_REQUEST

[CANSM425]⌈The API request `CanSM_RequestComMode` (ref. to CANSM062) with the parameter `ComM_Mode` equal to `COMM_FULL_COMMUNICATION` shall trigger the state machine with `T_FULL_COM_MODE_REQUEST`, if the function parameter `network` matches the configuration parameter `CANSM_NETWORK_HANDLE` (ref. to CANSM161_Conf).⌋()

7.2.4 Trigger: T_NO_COM_MODE_REQUEST

[CANSM426]⌈The API request `CanSM_RequestComMode` (ref. to CANSM062) with the parameter `ComM_Mode` equal to `COMM_NO_COMMUNICATION` shall trigger the state machine with `T_NO_COM_MODE_REQUEST`, if the function parameter `network` matches the configuration parameter `CANSM_NETWORK_HANDLE` (ref. to CANSM161_Conf).⌋()

7.2.5 Guarding condition: G_FULL_COM_MODE_REQUESTED

[CANSM427]⌈The guarding condition `G_FULL_COM_MODE_REQUESTED` of the CanSM_BSM state machine (ref. to Figure 7-1) shall evaluate, if the latest accepted communication mode request with `CanSM_RequestComMode` (ref. to CANSM062) for the respective network handle of the state machine has been with the parameter `ComM_Mode` equal to `COMM_FULL_COMMUNICATION.`⌋()

7.2.6 Guarding condition: G_SILENT_COM_MODE_REQUESTED

[CANSM429]⌈The guarding condition `G_SILENT_COM_MODE_REQUESTED` of the CanSM_BSM state machine (ref. to Figure 7-1) shall evaluate, if the latest accepted communication mode request with `CanSM_RequestComMode` (ref. to CANSM062) for the respective network handle of the state machine has been with the parameter ComM_Mode equal to COMM_SILENT_COMMUNICATION.⌋()

7.2.7 Effect: E_PRE_NOCOM

[CANSM431] ⌈The effect E_PRE_NOCOM of the CanSM_BSM state machine (ref. to Figure 7-1) shall call for the corresponding CAN network the API BswM_CanSM_CurrentState with the parameters Network := CanSMComMNetworkHandleRef and CurrentState := CANSM_BSWM_NO_COMMUNICATION.⌋()

7.2.8 Effect: E_NOCOM

[CANSM430] ⌈The effect E_NOCOM of the CanSM_BSM state machine (ref. to Figure 7-1) shall change the internally stored network mode (ref. to CANSM266) of the addressed CAN network to COMM_NO_COMMUNICATION and shall call the API ComM_BusSM_ModeIndication with the parameters Channel := CanSMComMNetworkHandleRef (ref. to CANSM161_Conf) and ComMode := COMM_NO_COMMUNICATION.⌋()

7.2.9 Effect: E_FULL_COM

[CANSM435] ⌈The effect E_FULL_COM of the CanSM_BSM state machine (ref. to Figure 7-1) shall call at 1st place for the corresponding CAN network the API BswM_CanSM_CurrentState with the parameters Network := CanSMComMNetworkHandleRef and CurrentState := CANSM_BSWM_FULL_COMMUNICATION.⌋()

[CANSM539] ⌈The effect E_FULL_COM of the CanSM_BSM state machine (ref. to Figure 7-1) shall call at 2nd place for each configured CAN controller of the CAN network the API CanIf_SetPduMode with the parameters ControllerId := CanSMControllerId (ref. to CANSM141_Conf) and PduModeRequest := CANIF_SET_ONLINE.⌋()

[CANSM540] ⌈The effect E_FULL_COM of the CanSM_BSM state machine (ref. to Figure 7-1) shall call at 3rd place for the corresponding CAN network the API ComM_BusSM_ModeIndication with the parameters Channel := CanSMComMNetworkHandleRef (ref. to CANSM161_Conf) and ComMode := COMM_FULL_COMMUNICATION.⌋()

7.2.10 Effect: E_FULL_TO_SILENT_COM

[CANSM434] ⌈The effect E_FULL_TO_SILENT_COM of the CanSM_BSM state machine (ref. to Figure 7-1) shall call at 1st place for the corresponding CAN network the API BswM_CanSM_CurrentState with the parameters Network := CanSMComMNetworkHandleRef and CurrentState := CANSM_BSWM_SILENT_COMMUNICATION.⌋()

[CANSM541] ⌈The effect E_FULL_TO_SILENT_COM of the CanSM_BSM state machine (ref. to Figure 7-1) shall call at 2nd place for each configured CAN controller of the CAN network the API CanIf_SetPduMode with the parameters ControllerId := CanSMControllerId (ref. to CANSM141_Conf) and PduModeRequest := CANIF_SET_ONLINE⌋()

[CANSM537] ⌈The effect E_FULL_TO_SILENT_COM of the CanSM_BSM state machine (ref. to Figure 7-1) shall call at 3rd place for each configured CAN controller of the CAN network the API CanIf_SetPduMode with the parameters ControllerId := CanSMControllerId (ref. to CANSM141_Conf) and PduModeRequest := CANIF_SET_TX_OFFLINE.⌋()

[CANSM538] ⌈The effect E_FULL_TO_SILENT_COM of the CanSM_BSM state machine (ref. to Figure 7-1) shall call at 4th place for the corresponding CAN network the API ComM_BusSM_ModeIndication with the parameters Channel := CanSMComMNetworkHandleRef (ref. to CANSM161_Conf) and ComMode := COMM_SILENT_COMMUNICATION.⌋()

7.2.11 Effect: E_BR_END_FULL_COM

[CANSM432] ⌈The effect E_BR_END_FULL_COM of the CanSM_BSM state machine (ref. to Figure 7-1) shall be the same as E_FULLCOM (ref. to CANSM435).⌋()

7.2.12 Effect: E_BR_END_SILENT_COM

[CANSM433] ⌈The effect E_BR_END_SILENT_COM of the CanSM_BSM state machine (ref. to Figure 7-1) shall be the same as E_FULL_TO_SILENT_COM (ref. to CANSM434).⌋()

7.2.13 Effect: E_SILENT_TO_FULL_COM

[CANSM550] ⌈The effect E_SILENT_TO_FULL_COM of the CanSM_BSM state machine (ref. to Figure 7-1) shall be the same as E_FULLCOM (ref. to CANSM435).⌋()

7.2.14 Sub state machine: CANSM_BSM_S_PRE_NOCOM

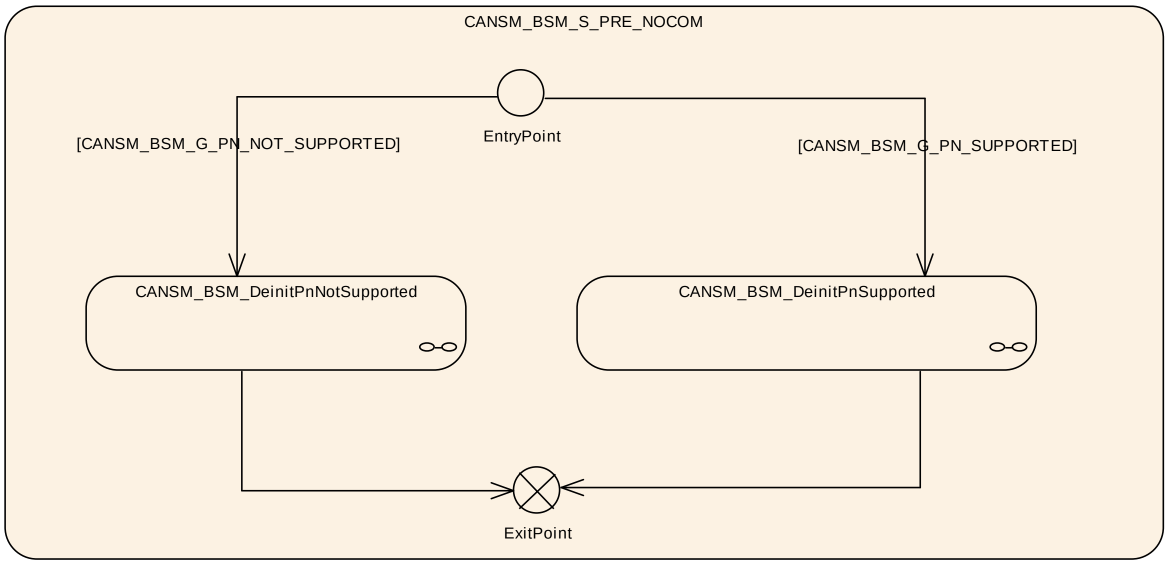
CANSM_BSM_S_PRE_NOCOM
EntryPoint
[CANSM_BSM_G_PN_NOT_SUPPORTED]
[CANSM_BSM_G_PN_SUPPORTED]
CANSM_BSM_DeinitPnNotSupported
CANSM_BSM_DeinitPnSupported
ExitPoint

Figure 7-2: CANSM_BSM_S_PRE_NOCOM, sub state machine of CANSM_BSM

7.2.14.1 Guarding condition: CANSM_BSM_G_PN_NOT_SUPPORTED

[CANSM436] ⌈The guarding condition `CANSM_BSM_G_PN_NOT_SUPPORTED` of the sub state machine `CANSM_BSM_S_PRE_NO_COM` (ref. to Figure 7-2) shall evaluate, if the configuration parameter `CanTrcvHwPnSupport` (ref. to [9], CanTrcv160_Conf) is `FALSE`, which is available via the reference `CanSMTransceiverId` (ref. to CANSM137_Conf) or if no `CanSMTransceiverId` is configured at all.⌋()

7.2.14.2 Guarding condition: CANSM_BSM_G_PN_SUPPORTED

[CANSM437] ⌈The guarding condition `CANSM_BSM_G_PN_SUPPORTED` of the sub state machine `CANSM_BSM_S_PRE_NO_COM` (ref. to Figure 7-2) shall evaluate, if a `CanSMTransceiverId` (ref. to CANSM137_Conf) is configured and if the configuration parameter `CanTrcvHwPnSupport` (ref. to [9], CanTrcv160_Conf) is `TRUE`, which is available via the reference `CanSMTransceiverId` (ref. to CANSM137_Conf).⌋()

7.2.14.3 Sub state machine for deinitialization with partial network support

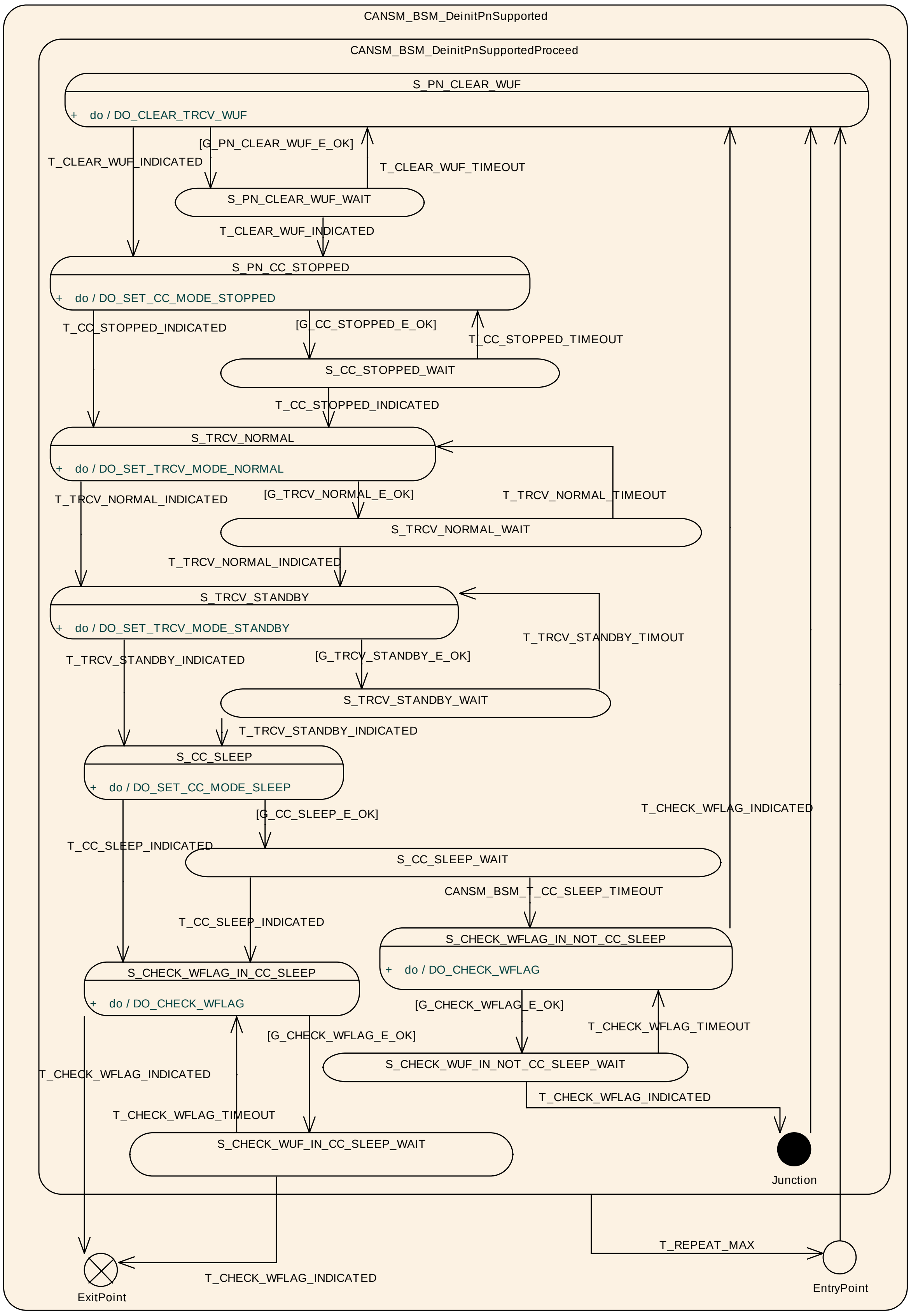

Figure 7-3: CANSM_BSM_DeinitPnSupported, sub state machine of CANSM_BSM_S_PRE_NOCOM

7.2.14.3.1 State operation to do in: S_PN_CLEAR_WUF

[CANSM438] ⌈As long the sub state machine CANSM_BSM_DeinitPnSupported (ref. to Figure 7-3) is in the state S_PN_CLEAR_WUF, the CanSM module operate the do action DO_CLEAR_TRCV_WUF and therefore repeat the API request CanIf_ClrTrcvWufFlag (ref. to chapter 8.6.1) and use the configured Transceiver (ref. to CANSM137_Conf) as API function parameter.⌋()

7.2.14.3.2 Guarding condition: G_PN_CLEAR_WUF_E_OK

[CANSM439] ⌈The guarding condition G_PN_CLEAR_WUF_E_OK of the sub state machine CANSM_BSM_DeinitPnSupported (ref. to Figure 7-3) shall be passed, if the API call of CANSM438 has returned E_OK.⌋()

7.2.14.3.3 Trigger: T_CLEAR_WUF_INDICATED

[CANSM440] ⌈The callback function CanSM_ClearTrcvWufFlagIndication (ref. to CANSM413) shall trigger the sub state machine CANSM_BSM_DeinitPnSupported (ref. to Figure 7-3) of the CAN network with T_CLEAR_WUF_INDICATED, if the function parameter Transceiver of CanSM_ClearTrcvWufFlagIndication matches to the configured CAN Transceiver (ref. to CANSM137_Conf) of the CAN network.⌋()

7.2.14.3.4 Trigger: T_CLEAR_WUF_TIMEOUT

[CANSM443] ⌈After a timeout of CANSM_MODEREQ_REPEAT_TIME (ref. to CANSM336_Conf) for the callback function CanSM_ClearTrcvWufFlagIndication (ref. to CANSM440), this condition shall trigger the sub state machine CANSM_BSM_DeinitPnSupported (ref. to Figure 7-3) of the respective network with T_CLEAR_WUF_TIMEOUT.⌋()

7.2.14.3.5 State operation to do in: S_PN_CC_STOPPED

[CANSM441] ⌈As long the sub state machine CANSM_BSM_DeinitPnSupported (ref. to Figure 7-3) is in the state S_PN_CC_STOPPED, the CanSM module shall operate the do action DO_SET_CC_MODE_STOPPED and therefore repeat for all configured CAN controllers of the CAN network (ref. to CANSM141_Conf) the API request CanIf_SetControllerMode (ref. to chapter 8.6.1) with ControllerMode equal to CANIF_CS_STOPPED.⌋()

7.2.14.3.6 Guarding condition: G_CC_STOPPED_E_OK

[CANSM442] ⌈The guarding condition G_CC_STOPPED_E_OK of the sub state machine CANSM_BSM_DeinitPnSupported (ref. to Figure 7-3) shall be passed, if all API calls of CANSM441 have returned E_OK.⌋()

7.2.14.3.7 Trigger: T_CC_STOPPED_INDICATED

[CANSM444] ⌈If CanSM module has got all mode indications (ref. to CANSM396) for the configured CAN controllers of the CAN network (ref. to CANSM141_Conf) after the respective requests to stop the CAN controllers of the CAN network (ref. to CANSM442), this shall trigger the sub state machine CANSM_BSM_DeinitPnSupported (ref. to Figure 7-3) of the CAN network with T_CC_STOPPED_INDICATED.⌋()

7.2.14.3.8 Trigger: T_CC_STOPPED_TIMEOUT

[CANSM445] ⌈After a timeout of CANSM_MODEREQ_REPEAT_TIME (ref. to CANSM336_Conf) for all supposed controller stopped mode indications (ref. to CANSM444), this condition shall trigger the sub state machine CANSM_BSM_DeinitPnSupported (ref. to Figure 7-3) of the respective network with T_CC_STOPPED_TIMEOUT.⌋()

7.2.14.3.9 State operation to do in: S_TRCV_NORMAL

[CANSM446] ⌈As long the sub state machine CANSM_BSM_DeinitPnSupported (ref. to Figure 7-3) is in the state S_TRCV_NORMAL, the CanSM module shall operate the do action DO_SET_TRCV_MODE_NORMAL and therefore repeat for the configured CAN Transceiver of the CAN network (ref. to CANSM137_Conf) the API request CanIf_SetTrcvMode (ref. to chapter 8.6.1) with TransceiverMode equal to CANTRCV_TRCVMODE_NORMAL.⌋()

7.2.14.3.10 Guarding condition: G_TRCV_NORMAL_E_OK

[CANSM447] ⌈The guarding condition G_TRCV_NORMAL_E_OK of the sub state machine CANSM_BSM_DeinitPnSupported (ref. to Figure 7-3) shall be passed, if the API call of CANSM446 has returned E_OK.⌋()

7.2.14.3.11 Trigger: T_TRCV_NORMAL_INDICATED

[CANSM448] ⌈If CanSM module has got the CANTRCV_TRCVMODE_NORMAL mode indication (ref. to CANSM399) for the configured CAN Transceiver of the CAN network (ref. to CANSM137_Conf) after the respective request (ref. to CANSM446), this shall trigger the sub state machine CANSM_BSM_DeinitPnSupported (ref. to Figure 7-3) of the CAN network with T_TRCV_NORMAL_INDICATED.⌋()

7.2.14.3.12 Trigger: T_TRCV_NORMAL_TIMEOUT

[CANSM449] ⌈After a timeout of CANSM_MODEREQ_REPEAT_TIME (ref. to CANSM336_Conf) for the supposed transceiver normal indication (ref. to CANSM448), this condition shall trigger the sub state machine CANSM_BSM_DeinitPnSupported (ref. to Figure 7-3) of the respective network with T_TRCV_NORMAL_TIMEOUT.⌋()

7.2.14.3.13 State operation to do in: S_TRCV_STANDBY

[CANSM450] ⌈As long the sub state machine CANSM_BSM_DeinitPnSupported (ref. to Figure 7-3) is in the state S_TRCV_STANDBY, the CanSM module shall operate the do action DO_SET_TRCV_STANDBY and therefore repeat for the configured CAN Transceiver of the CAN network (ref. to CANSM137_Conf) the API request CanIf_SetTrcvMode (ref. to chapter 8.6.1) with TransceiverMode equal to CANTRCV_TRCVMODE_STANDBY.⌋()

7.2.14.3.14 Guarding condition: G_TRCV_STANDBY_E_OK

[CANSM451] ⌈The guarding condition G_TRCV_STANDBY_E_OK of the sub state machine CANSM_BSM_DeinitPnSupported (ref. to Figure 7-3) shall be passed, if the API call of CANSM450 has returned E_OK.⌋()

7.2.14.3.15 Trigger: T_TRCV_STANDBY_INDICATED

[CANSM452] ⌈If the CanSM module has got the CANTRCV_TRCVMODE_STANDBY mode indication (ref. to CANSM399) for the configured CAN Transceiver of the CAN network (ref. to CANSM137_Conf) after the respective request (ref. to CANSM450), this shall trigger the sub state machine CANSM_BSM_DeinitPnSupported (ref. to Figure 7-3) of the CAN network with T_TRCV_STANDBY_INDICATED.⌋()

7.2.14.3.16 Trigger: T_TRCV_STANDBY_TIMEOUT

[CANSM454] ⌈After a timeout of CANSM_MODEREQ_REPEAT_TIME (ref. to CANSM336_Conf) for the supposed transceiver standby indication (ref. to CANSM452), this condition shall trigger the sub state machine CANSM_BSM_DeinitPnSupported (ref. to Figure 7-3) of the respective network with T_TRCV_STANDBY_TIMEOUT.⌋()

7.2.14.3.17 State operation to do in: S_CC_SLEEP

[CANSM453] ⌈As long the sub state machine CANSM_BSM_DeinitPnSupported (ref. to Figure 7-3) is in the state S_CC_SLEEP, the CanSM module shall operate the do action DO_SET_CC_MODE_SLEEP and therefore repeat for all configured CAN controllers of the CAN network (ref. to CANSM141_Conf) the API request CanIf_SetControllerMode (ref. to chapter 8.6.1) with ControllerMode equal to CANIF_CS_SLEEP.⌋()

7.2.14.3.18 Guarding condition: G_CC_SLEEP_E_OK

[CANSM455] ⌈The guarding condition G_CC_SLEEP_E_OK of the sub state machine CANSM_BSM_DeinitPnSupported (ref. to Figure 7-3) shall be passed, if all API calls of CANSM453 have returned E_OK.⌋()

7.2.14.3.19 Trigger: T_CC_SLEEP_INDICATED

[CANSM456] ⌈If CanSM module has got all mode indications (ref. to CANSM396) for the configured CAN controllers of the CAN network (ref. to CANSM141_Conf) after the respective requests to set the CAN controllers of the CAN network to sleep mode (ref. to CANSM453), this shall trigger the sub state machine CANSM_BSM_DeinitPnSupported (ref. to Figure 7-3) of the CAN network with T_CC_SLEEP_INDICATED.⌋()

7.2.14.3.20 Trigger: CANSM_BSM_T_CC_SLEEP_TIMEOUT

[CANSM457] ⌈After a timeout of CANSM_MODEREQ_REPEAT_TIME (ref. to CANSM336_Conf) for all supposed controller sleep mode indications (ref. to CANSM456), this condition shall trigger the sub state machine CANSM_BSM_DeinitPnSupported (ref. to Figure 7-3) of the respective network with CANSM_BSM_T_CC_SLEEP_TIMEOUT.⌋()

7.2.14.3.21 State operation to do in: S_CHECK_WFLAG_IN_CC_SLEEP

[CANSM458] ⌈As long the sub state machine CANSM_BSM_DeinitPnSupported (ref. to Figure 7-3) is in the state S_CHECK_WFLAG_IN_CC_SLEEP, the CanSM module operate the do action DO_CHECK_WFLAG and therefore repeat the API request CanIf_CheckTrcvWakeFlag (ref. to chapter 8.6.1) and use the configured CAN Transceiver of the related Network (ref. to CANSM137_Conf) as Transceiver parameter.⌋()

7.2.14.3.22 Guarding condition: G_CHECK_WFLAG_E_OK

[CANSM459] ⌈The guarding condition G_CHECK_WFLAG_E_OK of the sub state machine CANSM_BSM_DeinitPnSupported (ref. to Figure 7-3) shall be passed, if the API call of CANSM458 or CANSM462 has returned E_OK.⌋()

7.2.14.3.23 Trigger: T_CHECK_WFLAG_INDICATED

[CANSM460] ⌈The callback function CanSM_CheckTransceiverWakeFlag-Indication (ref. to CANSM416) shall trigger the sub state machine CANSM_BSM_DeinitPnSupported (ref. to Figure 7-3) of the CAN network with T_CHECK_WFLAG_INDICATED, if the function parameter Transceiver of CanSM_CheckTransceiverWakeFlagIndication matches to the configured CAN Transceiver (ref. to CANSM137_Conf) of the CAN network.⌋()

7.2.14.3.24 Trigger: T_CHECK_WFLAG_TIMEOUT

[CANSM461] ⌈After a timeout of CANSM_MODEREQ_REPEAT_TIME (ref. to CANSM336_Conf) for the callback function CanSM_CheckTransceiver-WakeFlagIndication (ref. to CANSM460), this condition shall trigger the sub state machine CANSM_BSM_DeinitPnSupported (ref. to Figure 7-3) of the respective network with T_CHECK_WFLAG_TIMEOUT.⌋()

7.2.14.3.25 State operation to do in: S_CHECK_WFLAG_IN_NOT_CC_SLEEP

[CANSM462] ⌈As long the sub state machine CANSM_BSM_DeinitPnSupported (ref. to Figure 7-3) is in the state S_CHECK_WFLAG_IN_NOT_CC_SLEEP, the CanSM module operate the do action DO_CHECK_WFLAG and therefore repeat the API request CanIf_CheckTrcvWakeFlag (ref. to chapter 8.6.1) and use the configured CAN Transceiver of the related Network (ref. to CANSM137_Conf) as Transceiver parameter.⌋()

7.2.14.3.26 Trigger: T_REPEAT_MAX

[CANSM463] ⌈If the sub state machine CANSM_BSM_DeinitPnSupported (ref. to Figure 7-3) has repeated any of the CanIf API calls (ref. to CANSM438, CANSM441, CANSM446, CANSM450, CANSM453, CANSM458, CANSM462) more often than configured (ref. to CANSM335_Conf) without getting the return value E_OK and without getting the supposed mode indication callbacks (ref. to CANSM444, CANSM448, CANSM452, CANSM456, CANSM460), this shall trigger the sub state machine CANSM_BSM_DeinitPnSupported with T_REPEAT_MAX.⌋()

7.2.14.4 Sub state machine for deinitialization without partial network support

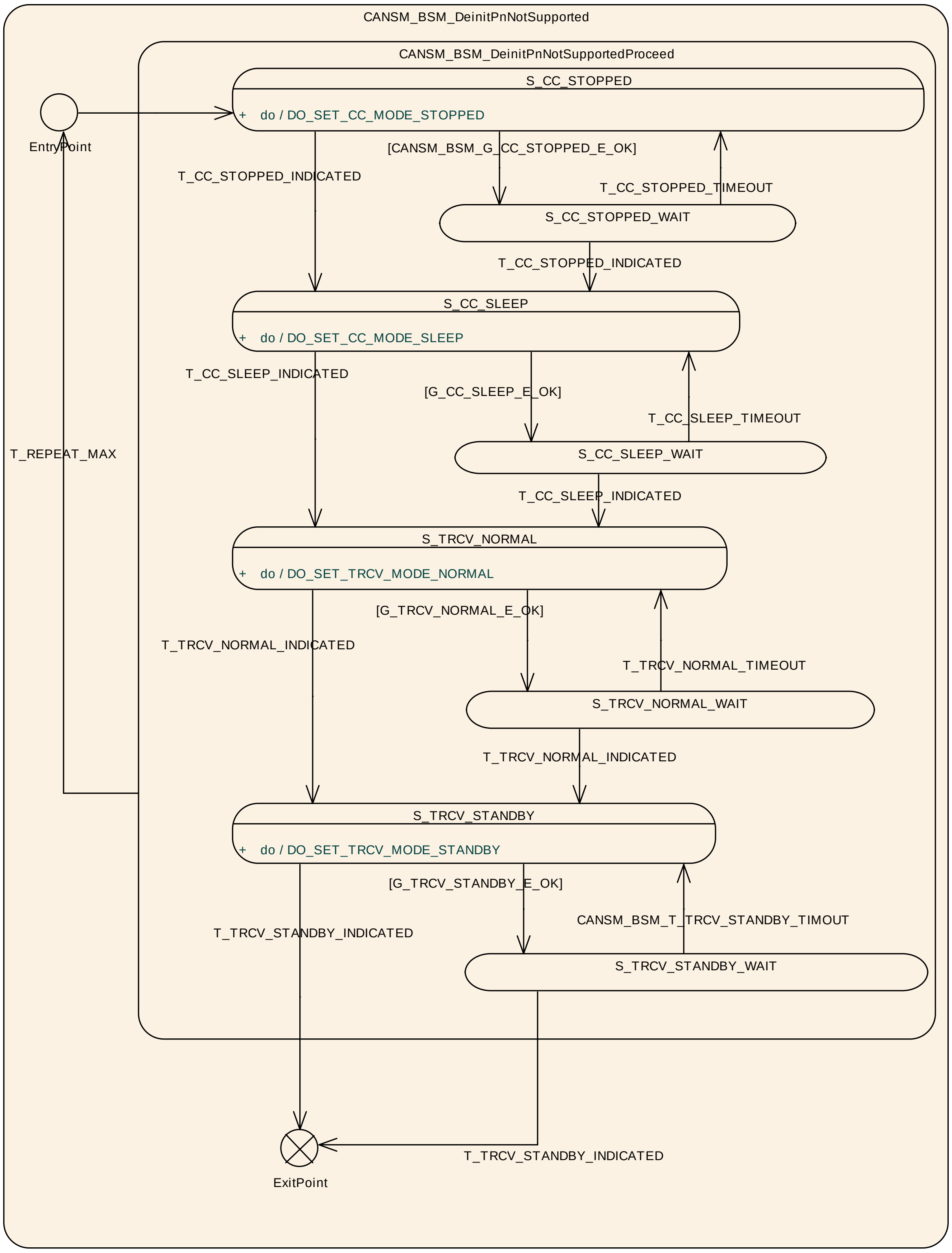

Figure 7-4: CANSM_BSM_DeinitPnNotSupported, sub state machine of CANSM_BSM_S_PRE_NOCOM

7.2.14.4.1 State operation to do in: S_CC_STOPPED

[CANSM464] ⌈As long the sub state machine CANSM_BSM_DeinitPnNotSupported (ref. to Figure 7-4) is in the state S_CC_STOPPED, the CanSM module shall operate the do action DO_SET_CC_MODE_STOPPED and therefore repeat for all configured CAN controllers of the CAN network (ref. to CANSM141_Conf) the API request CanIf_SetControllerMode (ref. to chapter 8.6.1) with ControllerMode equal to CANIF_CS_STOPPED.⌋()

7.2.14.4.2 Guarding condition: CANSM_BSM_G_CC_STOPPED_OK

[CANSM465] ⌈The guarding condition CANSM_BSM_G_CC_STOPPED_OK of the sub state machine CANSM_BSM_DeinitPnNotSupported (ref. to Figure 7-4) shall be passed, if all API calls of CANSM464 have returned E_OK.⌋()

7.2.14.4.3 Trigger: T_CC_STOPPED_INDICATED

[CANSM466] ⌈If CanSM module has got all mode indications (ref. to CANSM396) for the configured CAN controllers of the CAN network (ref. to CANSM141_Conf) after the respective requests to stop the CAN controllers of the CAN network (ref. to CANSM464), this shall trigger the sub state machine CANSM_BSM_DeinitPnNotSupported (ref. to Figure 7-4) of the CAN network with T_CC_STOPPED_INDICATED.⌋()

7.2.14.4.4 Trigger: T_CC_STOPPED_TIMEOUT

[CANSM467] ⌈After a timeout of CANSM_MODEREQ_REPEAT_TIME (ref. to CANSM336_Conf) for all supposed controller stopped mode indications (ref. to CANSM466), this condition shall trigger the sub state machine CANSM_BSM_DeinitPnNotSupported (ref. to Figure 7-4) of the respective network with T_CC_STOPPED_TIMEOUT.⌋()

7.2.14.4.5 State operation to do in: S_CC_SLEEP

[CANSM468] ⌈As long the sub state machine CANSM_BSM_DeinitPnNotSupported (ref. to Figure 7-4) is in the state S_CC_SLEEP, the CanSM module shall operate the do action DO_SET_CC_MODE_SLEEP and therefore repeat for all configured CAN controllers of the CAN network (ref. to CANSM141_Conf) the API request CanIf_SetControllerMode (ref. to chapter 8.6.1) with ControllerMode equal to CANIF_CS_SLEEP.⌋()

7.2.14.4.6 Guarding condition: G_CC_SLEEP_E_OK

[CANSM469] ⌈The guarding condition `G_CC_SLEEP_E_OK` of the sub state machine `CANSM_BSM_DeinitPnNotSupported` (ref. to Figure 7-4) shall be passed, if all API calls of CANSM468 have returned `E_OK`.⌋()

7.2.14.4.7 Trigger: T_CC_SLEEP_INDICATED

[CANSM470] ⌈If CanSM module has got all mode indications (ref. to CANSM396) for the configured CAN controllers of the CAN network (ref. to CANSM141_Conf) after the respective requests to set the CAN controllers of the CAN network to sleep mode (ref. to CANSM468), this shall trigger the sub state machine `CANSM_BSM_DeinitPnNotSupported` (ref. to Figure 7-4) of the CAN network with `T_CC_SLEEP_INDICATED`.⌋()

7.2.14.4.8 Trigger: T_CC_SLEEP_TIMEOUT

[CANSM471] ⌈After a timeout of `CANSM_MODEREQ_REPEAT_TIME` (ref. to CANSM336_Conf) for all supposed controller sleep mode indications (ref. to CANSM470), this condition shall trigger the sub state machine `CANSM_BSM_DeinitPnNotSupported` (ref. to Figure 7-4) of the respective network with `T_CC_SLEEP_TIMEOUT`.⌋()

7.2.14.4.9 State operation to do in: S_TRCV_NORMAL

[CANSM472] ⌈If for the CAN network a CAN Transceiver is configured (ref. to CANSM137_Conf), then as long the sub state machine `CANSM_BSM_DeinitPnNotSupported` (ref. to Figure 7-4) is in the state `S_TRCV_NORMAL`, the CanSM module shall operate the do action `DO_SET_TRCV_MODE_NORMAL` and therefore repeat for the configured CAN Transceiver of the CAN network (ref. to CANSM137_Conf) the API request `CanIf_SetTrcvMode` (ref. to chapter 8.6.1) with `TransceiverMode` equal to `CANTRCV_TRCVMODE_NORMAL`.⌋()

7.2.14.4.10 Guarding condition: G_TRCV_NORMAL_E_OK

[CANSM473] ⌈The guarding condition `G_TRCV_NORMAL_E_OK` of the sub state machine `CANSM_BSM_DeinitPnNotSupported` (ref. to Figure 7-4) shall be passed, if the API call of CANSM472 has returned `E_OK`.⌋()

7.2.14.4.11 Trigger: T_TRCV_NORMAL_INDICATED

[CANSM474] ⌈If CanSM module has got the `CANTRCV_TRCVMODE_NORMAL` mode indication (ref. to CANSM399) for the configured CAN Transceiver of the CAN network (ref. to CANSM137_Conf) after the respective request (ref. to CANSM472), this shall trigger the sub state machine CANSM_BSM_DeinitPnNotSupported (ref. to Figure 7-4) of the CAN network with T_TRCV_NORMAL_INDICATED.⌋()

[CANSM556] ⌈If no CAN Transceiver is configured for the CAN network, then this shall trigger the sub state machine CANSM_BSM_DeinitPnNotSupported (ref. to Figure 7-4) of the CAN network in the state S_TRCV_NORMAL with T_TRCV_NORMAL_INDICATED.⌋()

7.2.14.4.12 Trigger: T_TRCV_NORMAL_TIMEOUT

[CANSM475] ⌈After a timeout of CANSM_MODEREQ_REPEAT_TIME (ref. to CANSM336_Conf) for the supposed transceiver normal indication (ref. to CANSM474), this condition shall trigger the sub state machine CANSM_BSM_DeinitPnNotSupported (ref. to Figure 7-4) of the respective network with T_TRCV_NORMAL_TIMEOUT.⌋()

7.2.14.4.13 State operation to do in: S_TRCV_STANDBY

[CANSM476] ⌈If for the CAN network a CAN Transceiver is configured (ref. to CANSM137_Conf), then as long the sub state machine CANSM_BSM_DeinitPnNotSupported (ref. to Figure 7-4) is in the state S_TRCV_STANDBY, the CanSM module shall operate the do action DO_SET_TRCV_MODE_STANDBY and therefore repeat for the configured CAN Transceiver of the CAN network (ref. to CANSM137_Conf) the API request CanIf_SetTrcvMode (ref. to chapter 8.6.1) with TransceiverMode equal to CANTRCV_TRCVMODE_STANDBY.⌋()

7.2.14.4.14 Guarding condition: G_TRCV_STANDBY_E_OK

[CANSM477] ⌈The guarding condition G_TRCV_STANDBY_E_OK of the sub state machine CANSM_BSM_DeinitPnNotSupported (ref. to Figure 7-4) shall be passed, if the API call of CANSM476 has returned E_OK.⌋()

7.2.14.4.15 Trigger: T_TRCV_STANDBY_INDICATED

[CANSM478] ⌈If CanSM module has got the CANTRCV_TRCVMODE_STANDBY mode indication (ref. to CANSM399) for the configured CAN Transceiver of the CAN network (ref. to CANSM137_Conf) after the respective request (ref. to CANSM476), this shall trigger the sub state machine CANSM_BSM_DeinitPnNotSupported (ref. to Figure 7-4) of the CAN network with T_TRCV_STANDBY_INDICATED.⌋()

[CANSM557] ⌈If no CAN Transceiver is configured for the CAN network (ref. to CANSM137_Conf), then this shall trigger the sub state machine

CANSM_BSM_DeinitPnNotSupported (ref. to Figure 7-4) of the CAN network in the state S_TRCV_STANDBY with T_TRCV_STANDBY_INDICATED.⌋()

7.2.14.4.16 Trigger: CANSM_BSM_T_TRCV_STANDBY_TIMEOUT

[CANSM479] ⌈After a timeout of CANSM_MODEREQ_REPEAT_TIME (ref. to CANSM336_Conf) for the supposed transceiver standby indication (ref. to CANSM478), this condition shall trigger the sub state machine CANSM_BSM_DeinitPnNotSupported (ref. to Figure 7-4) of the respective network with CANSM_BSM_T_TRCV_STANDBY_TIMEOUT.⌋()

7.2.14.4.17 Trigger: T_REPEAT_MAX

[CANSM480] ⌈If the sub state machine CANSM_BSM_DeinitPnNotSupported (ref. to Figure 7-4) has repeated any of the CanIf API calls (ref. to CANSM464, CANSM468, CANSM472, CANSM476) more often than configured (ref. to CANSM335_Conf) without getting the return value E_OK and without getting the supposed mode indication callbacks (ref. to CANSM466, CANSM470, CANSM474, CANSM478), this shall trigger the sub state machine CANSM_BSM_DeinitPnNotSupported with T_REPEAT_MAX.⌋()

7.2.15 Sub state machine to prepare full communication

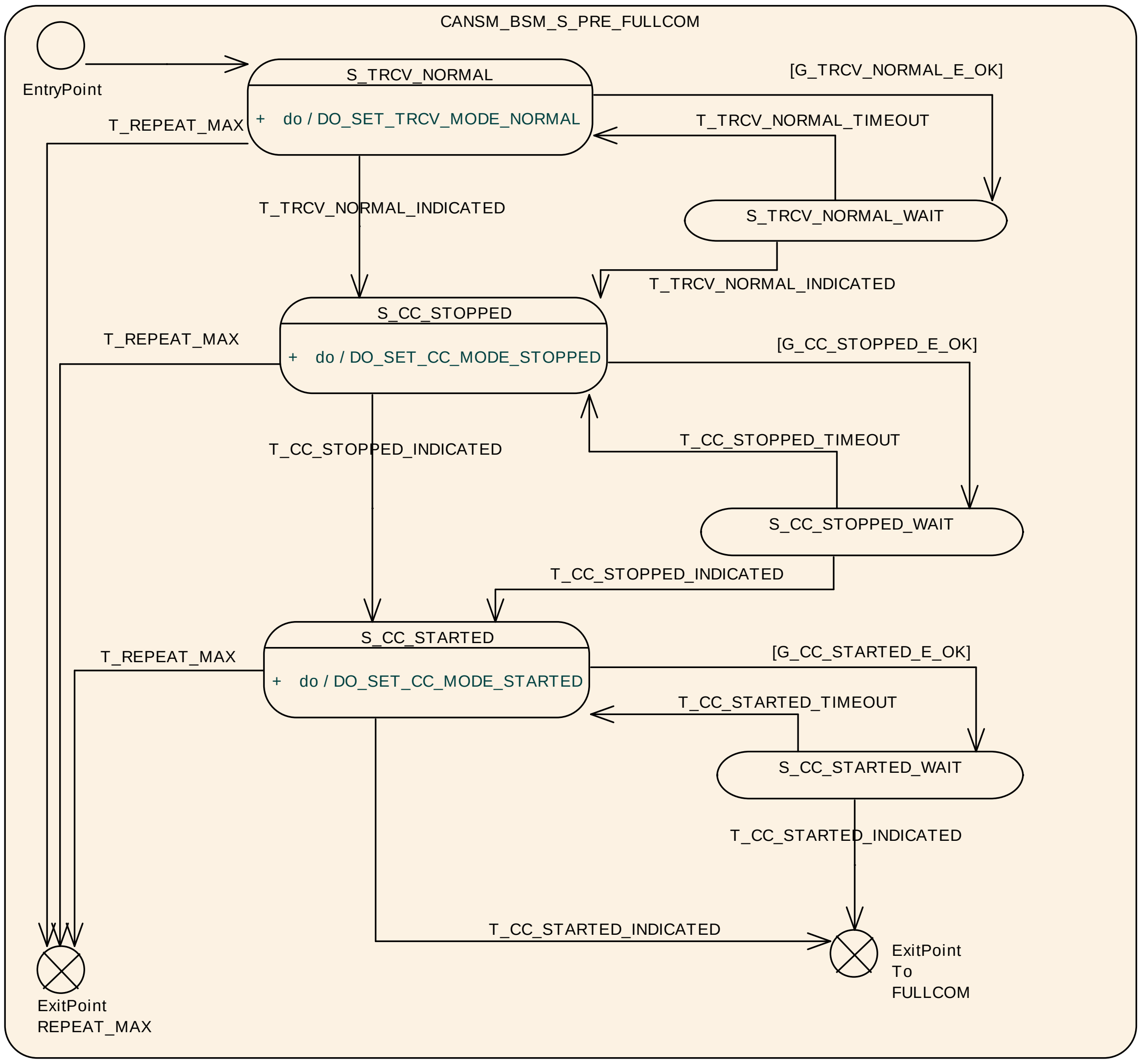

Figure 7-5: CANSM_BSM_S_PRE_FULLCOM, sub state machine of CANSM_BSM

7.2.15.1 State operation to do in: S_TRCV_NORMAL

[CANSM483] ⌈If for the CAN network a CAN Transceiver is configured (ref. to CANSM137_Conf), then as long the sub state machine `CANSM_BSM_S_PRE_FULLCOM` (ref. to Figure 7-5) is in the state `S_TRCV_NORMAL`, the CanSM module shall operate the do action `DO_SET_TRCV_MODE_NORMAL` and therefore repeat for the configured CAN Transceiver of the CAN network (ref. to CANSM137_Conf) the API request `CanIf_SetTrcvMode` (ref. to chapter 8.6.1) with `TransceiverMode` equal to `CANTRCV_TRCVMODE_NORMAL`.⌋()

7.2.15.2 Guarding condition: G_TRCV_NORMAL_E_OK

[CANSM484] ⌈The guarding condition G_TRCV_NORMAL_E_OK of the sub state machine CANSM_BSM_S_PRE_FULLCOM (ref. to Figure 7-5) shall be passed, if the API call of CANSM483 has returned E_OK.⌋()

7.2.15.3 Trigger: T_TRCV_NORMAL_INDICATED

[CANSM485] ⌈If CanSM module has got the CANTRCV_TRCVMODE_NORMAL mode indication (ref. to CANSM399) for the configured CAN Transceiver of the CAN network (ref. to CANSM137_Conf) after the respective request (ref. to CANSM483), this shall trigger the sub state machine CANSM_BSM_S_PRE_FULLCOM (ref. to Figure 7-5) of the CAN network with T_TRCV_NORMAL_INDICATED.⌋()

[CANSM558] ⌈If no CAN Transceiver is configured for the CAN network (ref. to CANSM137_Conf), then this shall trigger the sub state machine CANSM_BSM_S_PRE_FULLCOM (ref. to Figure 7-5) of the CAN network in the state S_TRCV_NORMAL with T_TRCV_NORMAL_INDICATED.⌋()

7.2.15.4 Trigger: T_TRCV_NORMAL_TIMEOUT

[CANSM486] ⌈After a timeout of CANSM_MODEREQ_REPEAT_TIME (ref. to CANSM336_Conf) for the supposed transceiver normal indication (ref. to CANSM485), this condition shall trigger the sub state machine CANSM_BSM_S_PRE_FULLCOM (ref. to Figure 7-5) of the respective network with T_TRCV_NORMAL_TIMEOUT.⌋()

7.2.15.5 State operation to do in: S_CC_STOPPED

[CANSM487] ⌈As long the sub state machine CANSM_BSM_S_PRE_FULLCOM (ref. to Figure 7-5) is in the state S_CC_STOPPED, the CanSM module shall operate the do action DO_SET_CC_MODE_STOPPED and therefore repeat for all configured CAN controllers of the CAN network (ref. to CANSM141_Conf) the API request CanIf_SetControllerMode (ref. to chapter 8.6.1) with ControllerMode equal to CANIF_CS_STOPPED.⌋()

7.2.15.6 Guarding condition: G_CC_STOPPED_OK

[CANSM488] ⌈The guarding condition G_CC_STOPPED_OK of the sub state machine CANSM_BSM_S_PRE_FULLCOM (ref. to Figure 7-5) shall be passed, if all API calls of CANSM487 have returned E_OK.⌋()

7.2.15.7 Trigger: T_CC_STOPPED_INDICATED

[CANSM489] ⌈If CanSM module has got all mode indications (ref. to CANSM396) for the configured CAN controllers of the CAN network (ref. to CANSM141_Conf) after the respective requests to stop the CAN controllers of the CAN network (ref. to CANSM487), this shall trigger the sub state machine `CANSM_BSM_S_PRE_FULLCOM` (ref. to Figure 7-5) of the CAN network with `T_CC_STOPPED_INDICATED`.⌋()

7.2.15.8 Trigger: T_CC_STOPPED_TIMEOUT

[CANSM490] ⌈After a timeout of `CANSM_MODEREQ_REPEAT_TIME` (ref. to CANSM336_Conf) for all supposed controller stopped mode indications (ref. to CANSM489), this condition shall trigger the sub state machine `CANSM_BSM_S_PRE_FULLCOM` (ref. to Figure 7-5) of the respective network with `T_CC_STOPPED_TIMEOUT`.⌋()

7.2.15.9 State operation to do in: S_CC_STARTED

[CANSM491] ⌈As long the sub state machine `CANSM_BSM_S_PRE_FULLCOM` (ref. to Figure 7-5) is in the state `S_CC_STARTED`, the CanSM module shall operate the do action `DO_SET_CC_MODE_STARTED` and therefore repeat for all configured CAN controllers of the CAN network (ref. to CANSM141_Conf) the API request `CanIf_SetControllerMode` (ref. to chapter 8.6.1) with `ControllerMode` equal to `CANIF_CS_STARTED`.⌋()

7.2.15.10 Guarding condition: G_CC_STARTED_OK

[CANSM492] ⌈The guarding condition `G_CC_STARTED_OK` of the sub state machine `CANSM_BSM_S_PRE_FULLCOM` (ref. to Figure 7-5) shall be passed, if all API calls of CANSM491 have returned `E_OK`.⌋()

7.2.15.11 Trigger: T_CC_STARTED_INDICATED

[CANSM493] ⌈If CanSM module has got all mode indications (ref. to CANSM396) for the configured CAN controllers of the CAN network (ref. to CANSM141_Conf) after the respective requests to start the CAN controllers of the CAN network (ref. to CANSM491), this shall trigger the sub state machine `CANSM_BSM_S_PRE_FULLCOM` (ref. to Figure 7-5) of the CAN network with `T_CC_STARTED_INDICATED`.⌋()

7.2.15.12 Trigger: T_CC_STARTED_TIMEOUT

[CANSM494] ⌈After a timeout of `CANSM_MODEREQ_REPEAT_TIME` (ref. to CANSM336_Conf) for all supposed controller started mode indications (ref. to CANSM493), this condition shall trigger the sub state machine

CANSM_BSM_S_PRE_FULLCOM (ref. to Figure 7-5) of the respective network with T_CC_STARTED_TIMEOUT.⌋()

7.2.15.13 Trigger: T_REPEAT_MAX

[CANSM495] ⌈If the sub state machine CANSM_BSM_S_PRE_FULLCOM (ref. to Figure 7-5) has repeated any of the CanIf API calls (ref. to CANSM483, CANSM487, CANSM491) more often than configured (ref. to CANSM335_Conf) without getting the return value E_OK and without getting the supposed mode indication callbacks (ref. to CANSM485, CANSM489, CANSM493), this shall trigger the sub state machine CANSM_BSM_S_PRE_FULLCOM with T_REPEAT_MAX.⌋()

7.2.16 Sub state machine for requested full communication mode

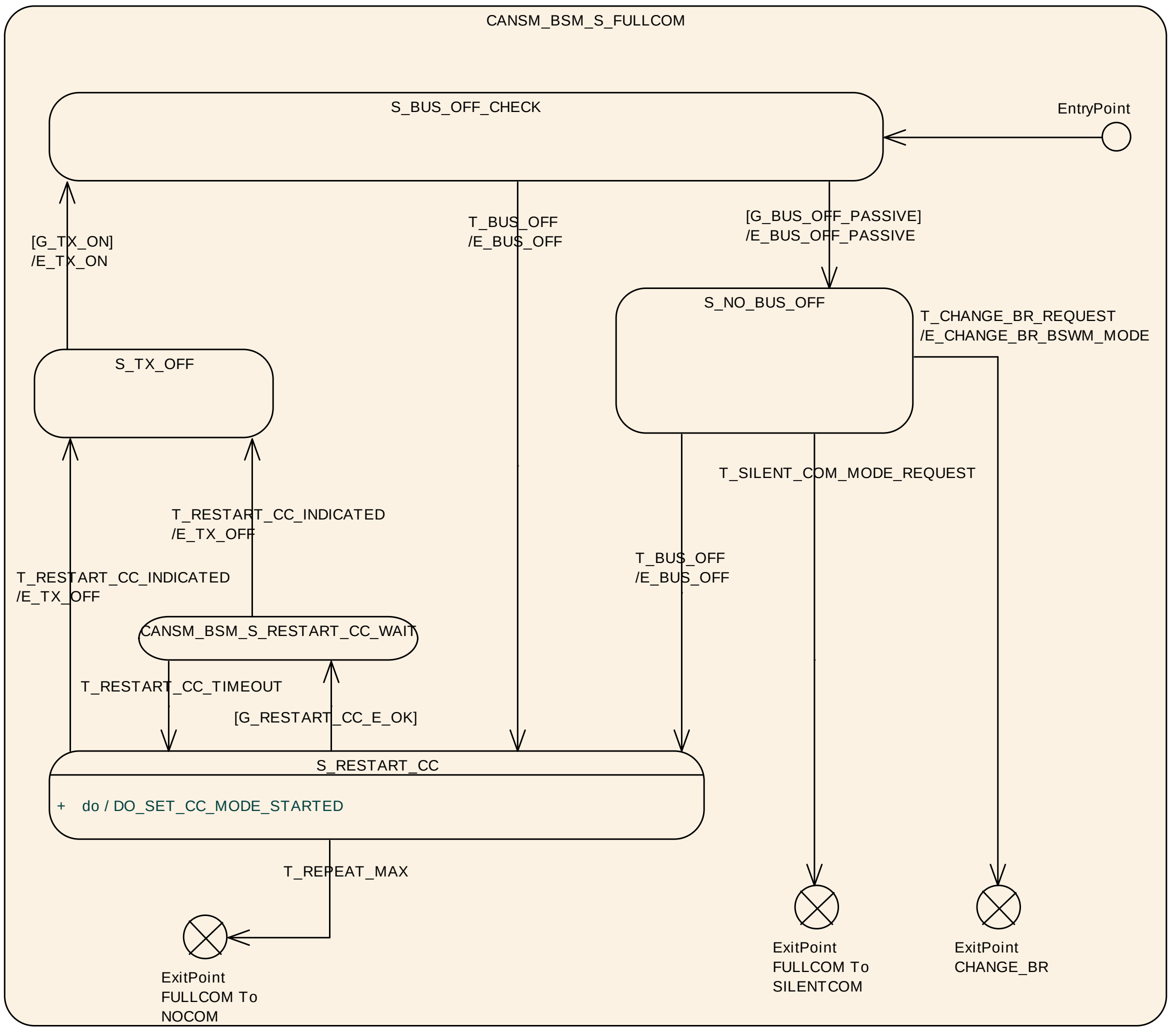

Figure 7-6: CANSM_BSM_S_FULLCOM, sub state machine of CANSM_BSM

7.2.16.1 Guarding condition: G_BUS_OFF_PASSIVE

[CANSM496] ⌈The guarding condition G_BUS_OFF_PASSIVE of the sub state machine CANSM_BSM_S_FULLCOM (ref. to Figure 7-6) shall be passed, if CANSM_BOR_TX_CONFIRMATION_POLLING is disabled (ref. to CANSM339_Conf) and the time duration since the effect E_TX_ON is greater or equal the configuration parameter CANSM_BOR_TIME_TX_ENSURED (ref. to CANSM130_Conf).⌋()

[CANSM497] ⌈The guarding condition G_BUS_OFF_PASSIVE of the sub state machine CANSM_BSM_S_FULLCOM (ref. to Figure 7-6) shall be passed, if CANSM_BOR_TX_CONFIRMATION_POLLING is enabled (ref. to CANSM339_Conf) and the API CanIf_GetTxConfirmationState (ref. to chapter 8.6.1) returns CANIF_TX_RX_NOTIFICATION for all configured CAN controllers of the CAN network (ref. to CANSM141_Conf).⌋()

7.2.16.2 Effect: E_BUS_OFF_PASSIVE

[CANSM498] ⌈The effect E_BUS_OFF_PASSIVE of the sub state machine CANSM_BSM_S_FULLCOM (ref. to Figure 7-6) shall invocate Dem_ReportErrorStatus (ref. to chapter 8.6.1) with the parameters EventId := CANSM_E_BUS_OFF (ref. to CANSM070_Conf) and EventStatus := DEM_EVENT_STATUS_PASSED.⌋(BSW00422)

7.2.16.3 Trigger: T_SILENT_COM_MODE_REQUEST

[CANSM499] ⌈The API request CanSM_RequestComMode (ref. to CANSM062) with the parameter ComM_Mode equal to COMM_SILENT_COMMUNICATION shall trigger the sub state machine CANSM_BSM_S_FULLCOM (ref. to Figure 7-6) with T_SILENT_COM_MODE_REQUEST, which corresponds to the function parameter network and the configuration parameter CANSM_NETWORK_HANDLE (ref. to CANSM161_Conf).⌋()

Rationale: Regular use case for the transition of the CanNm Network mode to the CanNm Prepare Bus-Sleep mode .

[CANSM554] ⌈The API request CanSM_RequestComMode (ref. to CANSM062) with the parameter ComM_Mode equal to COMM_NO_COMMUNICATION shall trigger the sub state machine CANSM_BSM_S_FULLCOM (ref. to Figure 7-6) with T_SILENT_COM_MODE_REQUEST, which corresponds to the function parameter network and the configuration parameter CANSM_NETWORK_HANDLE (ref. to CANSM161_Conf).⌋()

Remark: Depending on the ComM configuration, the ComM module will request COMM_SILENT_COMMUNICATION first and then COMM_NO_COMMUNICATION or

COMM_NO_COMMUNICATION directly (ComMNmVariant=LIGHT)".

7.2.16.4 Trigger: T_CHANGE_BR_REQUEST

[CANSM507] ⌈The API function CanSM_ChangeBaudrate (ref. to CANSM501) shall trigger the sub state machine CANSM_BSM_S_FULLCOM (ref. to Figure 7-6) for the requested CAN network with T_CHANGE_BR_REQUEST, if the CanSM module has accepted the CanSM_ChangeBaudrate request with return of E_OK.⌋()

7.2.16.5 Effect: E_CHANGE_BR_BSWM_MODE

[CANSM528] ⌈The effect E_CHANGE_BR_BSWM_MODE of the sub state machine CANSM_BSM_S_FULLCOM (ref. to Figure 7-6) shall call for the corresponding CAN network the API BswM_CanSM_CurrentState with the parameters Network := CanSMComMNetworkHandleRef and CurrentState := CANSM_BSWM_CHANGE_BAUDRATE.⌋()

7.2.16.6 Trigger: T_BUS_OFF

[CANSM500] ⌈The callback function CanSM_ControllerBusOff (ref. to CANSM064) shall trigger the sub state machine CANSM_BSM_S_FULLCOM (ref. to Figure 7-6) for the CAN network with T_BUS_OFF, if one of its configured CAN controllers matches to the function parameter ControllerId of the callback function CanSM_ControllerBusOff.⌋()

7.2.16.7 Effect: E_BUS_OFF

[CANSM508] ⌈The effect E_BUS_OFF of the sub state machine CANSM_BSM_S_FULLCOM (ref. to Figure 7-6) shall call at 1st place for the corresponding CAN network the API BswM_CanSM_CurrentState with the parameters Network := CanSMComMNetworkHandleRef and CurrentState := CANSM_BSWM_BUS_OFF.⌋()

[CANSM521] ⌈The effect E_BUS_OFF of the sub state machine CANSM_BSM_S_FULLCOM (ref. to Figure 7-6) shall call at 2nd place for the corresponding CAN network the API ComM_BusSM_ModeIndication with the parameters Channel := CanSMComMNetworkHandleRef (ref. to CANSM161_Conf) and ComMode := COMM_SILENT_COMMUNICATION.⌋()

[CANSM522] ⌈The effect E_BUS_OFF of the sub state machine CANSM_BSM_S_FULLCOM (ref. to Figure 7-6) shall invocate Dem_ReportErrorStatus (ref. to chapter 8.6.1) with the parameters EventId :=

CANSM_E_BUS_OFF (ref. to CANSM070_Conf) and EventStatus := DEM_EVENT_STATUS_PRE_FAILED.⌋(BSW00422)

7.2.16.8 State operation to do in: S_RESTART_CC

[CANSM509] ⌈As long the sub state machine CANSM_BSM_S_FULLCOM (ref. to Figure 7-6) is in the state S_RESTART_CC, the CanSM module shall operate the do action DO_SET_CC_MODE_STARTED and therefore repeat for all configured CAN controllers of the CAN network (ref. to CANSM141_Conf) the API request CanIf_SetControllerMode (ref. to chapter 8.6.1) with ControllerMode equal to CANIF_CS_STARTED.⌋()

7.2.16.9 Guarding condition: G_RESTART_CC_OK

[CANSM510] ⌈The guarding condition G_RESTART_CC_OK of the sub state machine CANSM_BSM_S_FULLCOM (ref. to Figure 7-6) shall be passed, if all API calls of CANSM509 have returned E_OK.⌋()

7.2.16.10 Trigger: T_RESTART_CC_INDICATED

[CANSM511] ⌈If CanSM module has got all mode indications (ref. to CANSM396) for the configured CAN controllers of the CAN network (ref. to CANSM141_Conf) after the respective requests to start the CAN controllers of the CAN network (ref. to CANSM509), this shall trigger the sub state CANSM_BSM_S_FULLCOM (ref. to Figure 7-6) of the CAN network with T_RESTART_CC_INDICATED.⌋()

7.2.16.11 Trigger: T_RESTART_CC_TIMEOUT

[CANSM512] ⌈After a timeout of CANSM_MODEREQ_REPEAT_TIME (ref. to CANSM336_Conf) for all supposed controller started mode indications (ref. to CANSM511), this condition shall trigger the sub state machine CANSM_BSM_S_FULLCOM (ref. to Figure 7-6) of the respective network with T_RESTART_CC_TIMEOUT.⌋()

7.2.16.12 Effect: E_TX_OFF

[CANSM513] ⌈The effect E_TX_OFF of the sub state machine CANSM_BSM_S_FULLCOM (ref. to Figure 7-6) shall call for the configured CAN controllers of the CAN network (ref. to CANSM141_Conf) the API function CanIf_SetPduMode (ref. to chapter 8.6.1) with the parameters ControllerId := CanSMControllerId (ref. to CANSM141_Conf) and PduModeRequest := CANIF_SET_TX_OFFLINE.⌋()

7.2.16.13 Guarding condition: G_TX_ON

[CANSM514] ⌈The guarding condition G_TX_ON of the sub state machine CANSM_BSM_S_FULLCOM (ref. to Figure 7-6) shall be passed after a time duration of CanSMBorTimeL1 (ref. to CANSM128_Conf), if the count of bus-off recovery retries with E_BUS_OFF without passing the guarding condition G_BUS_OFF_PASSIVE is lower than CanSMBorCounterL1ToL2 (ref. to CANSM131_Conf).⌋()

[CANSM515] ⌈The guarding condition G_TX_ON of the sub state machine CANSM_BSM_S_FULLCOM (ref. to Figure 7-6) shall be passed after a time duration of CanSMBorTimeL2 (ref. to CANSM129_Conf), if the count of bus-off recovery retries with E_BUS_OFF without passing the guarding condition G_BUS_OFF_PASSIVE is greater than or equal to CanSMBorCounterL1ToL2 (ref. to CANSM131_Conf).⌋()

7.2.16.14 Effect: E_TX_ON

[CANSM516] ⌈The effect E_TX_ON of the sub state machine CANSM_BSM_S_FULLCOM (ref. to Figure 7-6) shall call at 1st place for the configured CAN controllers of the CAN network (ref. to CANSM141_Conf) the API function CanIf_SetPduMode (ref. to chapter 8.6.1) with the parameters ControllerId := CanSMControllerId (ref. to CANSM141_Conf) and PduModeRequest := CANIF_SET_ONLINE.⌋()

[CANSM517] ⌈The effect E_TX_ON of the sub state machine CANSM_BSM_S_FULLCOM (ref. to Figure 7-6) shall call at 2nd place for the corresponding CAN network the API BswM_CanSM_CurrentState with the parameters Network := CanSMComMNetworkHandleRef and CurrentState := CANSM_BSWM_FULL_COMMUNICATION.⌋()

[CANSM518] ⌈The effect E_TX_ON of the sub state machine CANSM_BSM_S_FULLCOM (ref. to Figure 7-6) shall call at 3rd place the API ComM_BusSM_ModeIndication with the parameters Channel := CanSMComMNetworkHandleRef (ref. to CANSM161_Conf) and ComMode := COMM_FULL_COMMUNICATION.⌋()

7.2.16.15 Trigger: T_REPEAT_MAX

[CANSM523] ⌈If the sub state machine CANSM_BSM_S_FULLCOM (ref. to Figure 7-6) has repeated the CanIf API to restart the CAN controller(s) of the CAN network (ref. to CANSM509) more often than configured (ref. to CANSM335_Conf) without getting the return value E_OK and without getting the supposed mode indication (ref. to CANSM511), this shall trigger the sub state machine CANSM_BSM_S_FULLCOM with T_REPEAT_MAX.⌋()

7.2.17 Sub state machine to operate a requested baud rate change

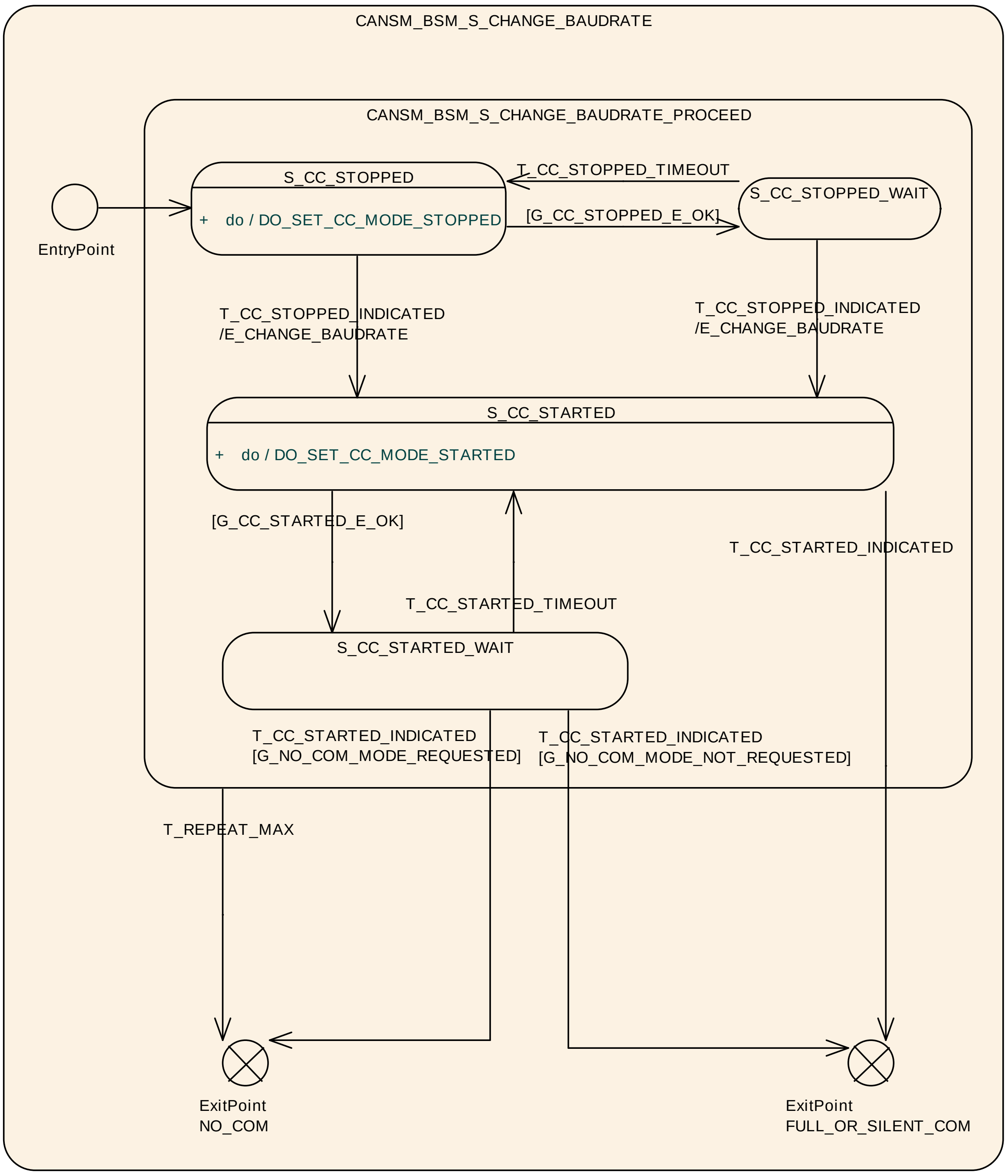

Figure 7-7: CANSM_BSM_S_CHANGE_BAUDRATE, sub state machine of CANSM_BSM

7.2.17.1 State operation to do in: S_CC_STOPPED

[CANSM524] ⌈As long the sub state machine `CANSM_BSM_S_CHANGE_BAUDRATE` (ref. to Figure 7-7) is in the state `S_CC_STOPPED`, the CanSM module shall operate the do action `DO_SET_CC_MODE_STOPPED` and therefore repeat for all configured CAN controllers of the CAN network (ref. to CANSM141_Conf) the API request

CanIf_SetControllerMode (ref. to chapter 8.6.1) with ControllerMode equal to CANIF_CS_STOPPED.⌋()

7.2.17.2 Guarding condition: G_CC_STOPPED_OK

[CANSM525] ⌈The guarding condition G_CC_STOPPED_OK of the sub state machine CANSM_BSM_S_CHANGE_BAUDRATE (ref. to Figure 7-7) shall be passed, if all API calls of CANSM524 have returned E_OK.⌋()

7.2.17.3 Trigger: T_CC_STOPPED_INDICATED

[CANSM526] ⌈If CanSM module has got all mode indications (ref. to CANSM396) for the configured CAN controllers of the CAN network (ref. to CANSM141_Conf) after the respective requests to stop the CAN controllers of the CAN network (ref. to CANSM524), this shall trigger the sub state machine CANSM_BSM_S_CHANGE_BAUDRATE (ref. to Figure 7-7) of the CAN network with T_CC_STOPPED_INDICATED.⌋()

7.2.17.4 Trigger: T_CC_STOPPED_TIMEOUT

[CANSM527] ⌈After a timeout of CANSM_MODEREQ_REPEAT_TIME (ref. to CANSM336_Conf) for all supposed controller stopped mode indications (ref. to CANSM526), this condition shall trigger the sub state machine CANSM_BSM_S_CHANGE_BAUDRATE (ref. to Figure 7-7) of the respective network with T_CC_STOPPED_TIMEOUT.⌋()

7.2.17.5 Effect: E_CHANGE_BAUDRATE

[CANSM529] ⌈The effect E_CHANGE_BAUDRATE of the sub state machine CANSM_BSM_S_CHANGE_BAUDRATE (ref. to Figure 7-7) shall call at 1st place for the corresponding CAN network the API ComM_BusSM_ModeIndication with the parameters Channel := CanSMComMNetworkHandleRef (ref. to CANSM161_Conf) and ComMode := COMM_NO_COMMUNICATION.⌋()

[CANSM531] ⌈The effect E_CHANGE_BAUDRATE of the sub state machine CANSM_BSM_S_CHANGE_BAUDRATE (ref. to Figure 7-7) shall call at 2nd place for all configured CAN controllers of the CAN network (ref. to CANSM141_Conf) the API request CanIf_ChangeBaudrate (ref. to chapter 8.6.2) with the respective ControllerId parameter and shall use as baudrate parameter the checked and remembered baud rate (ref. to CANSM572 and CANSM503).⌋()

7.2.17.6 State operation to do in: S_CC_STARTED

[CANSM532] ⌈As long the sub state machine CANSM_BSM_S_CHANGE_BAUDRATE (ref. to Figure 7-7) is in the state S_CC_STARTED, the CanSM module shall operate the do action DO_SET_CC_MODE_STARTED and therefore repeat for all configured CAN controllers of the CAN network (ref. to CANSM141_Conf) the API request CanIf_SetControllerMode (ref. to chapter 8.6.1) with ControllerMode equal to CANIF_CS_STARTED.⌋()

7.2.17.7 Guarding condition: G_CC_STARTED_OK

[CANSM533] ⌈The guarding condition G_CC_STARTED_OK of the sub state machine CANSM_BSM_S_CHANGE_BAUDRATE (ref. to Figure 7-7) shall be passed, if all API calls of CANSM532 have returned E_OK.⌋()

7.2.17.8 Trigger: T_CC_STARTED_INDICATED

[CANSM534] ⌈If CanSM module has got all mode indications (ref. to CANSM396) for the configured CAN controllers of the CAN network (ref. to CANSM141_Conf) after the respective requests to start the CAN controllers of the CAN network (ref. to CANSM532), this shall trigger the sub state machine CANSM_BSM_S_CHANGE_BAUDRATE (ref. to Figure 7-7) of the CAN network with T_CC_STARTED_INDICATED.⌋()

7.2.17.9 Trigger: T_CC_STARTED_TIMEOUT

[CANSM535] ⌈After a timeout of CANSM_MODEREQ_REPEAT_TIME (ref. to CANSM336_Conf) for all supposed controller started mode indications (ref. toCANSM534), this condition shall trigger the sub state machine CANSM_BSM_S_CHANGE_BAUDRATE (ref. to Figure 7-7) of the respective network with T_CC_STARTED_TIMEOUT.⌋()

7.2.17.10 Trigger: T_REPEAT_MAX

[CANSM536] ⌈If the sub state machine CANSM_BSM_S_CHANGE_BAUDRATE (ref. to Figure 7-7) has repeated the referenced CanIf APIs (ref. to CANSM524, CANSM532) for the CAN controllers of the corresponding CAN network more often than configured (ref. to CANSM335_Conf) without getting the return value E_OK and without getting the supposed mode indications (ref. to CANSM526, CANSM534), this shall trigger the sub state machine CANSM_BSM_S_CHANGE_BAUDRATE with T_REPEAT_MAX.⌋()

7.2.17.11 Guarding condition: G_NO_COM_MODE_REQUESTED

[CANSM542] ⌈The sub state machine `CANSM_BSM_S_CHANGE_BAUDRATE` (ref. to Figure 7-7) shall pass the guarding condition `G_NO_COM_MODE_REQUESTED`, if the latest accepted communication mode request with `CanSM_RequestComMode` (ref. to CANSM062) for the respective network handle of the state machine has been with the parameter `ComM_Mode` equal to `COMM_NO_COMMUNICATION`.⌋()

7.2.17.12 Guarding condition: G_NO_COM_MODE_NOT_REQUESTED

[CANSM543] ⌈The sub state machine `CANSM_BSM_S_CHANGE_BAUDRATE` (ref. to Figure 7-7) shall pass the guarding condition `G_NO_COM_MODE_NOT_REQUESTED`, if the latest accepted communication mode request with `CanSM_RequestComMode` (ref. to CANSM062) for the respective network handle of the state machine has been with the parameter `ComM_Mode` equal to `COMM_SILENT_COMMUNICATION` or `COMM_FULL_COMMUNICATION`.⌋()

7.3 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.⌋()

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
Network mode request during not finished bus-off recovery	Development	`CANSM_E_BUSOFF_RECOVERY_ACTIVE`	0x06
Network mode request during pending indication	Development	`CANSM_E_WAIT_MODE_INDICATION`	0x07
Invalid communication mode request	Development	`CANSM_E_INVALID_COMM_REQUEST`	0x08
Invalid BaudrateConfig for at least one of the CAN Controllers of the	Development	`CANSM_E_PARAM_INVALID_BAUDRATE`	0x09

requested CAN Network			
Mode request for a network failed more often as allowed by configuration	Development	`CANSM_E_MODE_REQUEST_TIMEOUT`	`0x0A`
The bus-off recovery state machine of a CAN network has detected a certain amount of sequential bus-offs without successful recovery	Production	`CANSM_E_BUS_OFF` (ref. to `CANSM070_Conf`)	`Assigned by DEM`

7.4 Error detection

[CANSM363] ⌈The detection of development errors shall be configurable as ON / OFF.⌋()

[CANSM364] ⌈The detection of development errors shall be configurable at pre-compile time.⌋()

[CANSM365] ⌈The switch `CanSMDevErrorDetect` (ref. to CANSM133_Conf) 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 7.3 and chapter 8.⌋(BSW00323, BSW00386)

[CANSM072] ⌈The detection of production code errors cannot be switched off.⌋()

Remark: Refer to CANSM498, CANSM522 for the detailed description of the production errors “bus-off” and “mode request timeout”.

7.5 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).⌋(BSW00338, BSW00386)

[CANSM074]⌈ Production errors shall be reported to the Diagnostic Event Manager.⌋(BSW00339)

Remark: For the configuration of the DEM module it has to be considered, that the bus-off events and CAN-controller-timeout events are already debounced by the CanSM module itself internally. The detailed description for the event status determination of those production errors can be found in to CANSM498, CANSM520, CANSM522.

7.6 Interface for AUTOSAR debug and trace

The following requirements shall be considered to export debug information from the CanSM module :

[CANSM310] ⌈The CanSM module shall define every variable as global, which is designated to be accessed by AUTOSAR debugging.⌋()

Rationale: Make debug information visible

[CANSM309] ⌈The type definitions of the debug-able variables of the CanSM module shall be exported by the standard module header file CanSM.h.⌋()

Rationale: To allow the debugging tool chain to calculate the size of elements by C-"sizeof" and to (optionally) decode the structure elements.

7.7 Non-functional design rules

The CanSM shall cover the software module design requirements of the SRS General [3].

8 API specification

8.1 Imported types

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

[CANSM243]

Module	***Imported Type***
CanIf	CanIf_ControllerModeType
	CanIf_NotifStatusType
	CanIf_PduSetModeType
Can_GeneralTypes	CanTrcv_TrcvModeType
ComM	ComM_ModeType
ComStack_Types	NetworkHandleType
Dem	Dem_EventIdType
	Dem_EventStatusType
Std_Types	Std_ReturnType
	Std_VersionInfoType

8.2 Type definitions

The following tables contain the type definitions of the CanSM module.

8.2.1 CanSM_StateType

Name:	`CanSM_StateType`	
Type:	`Enumeration`	
Range:	`CANSM_INITED`	--
	`CANSM_UNINITED`	--
Description:	Defines the values of the internal states of the CanSM module	

8.2.2 CanSM_ConfigType

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.		

8.2.3 CanSM_BswMCurrentStateType

Name:	`CanSM_BswMCurrentStateType`	
Type:	`Enumeration`	
Range:	`CANSM_BSWM_NO_COMMUNICATION`	--
	`CANSM_BSWM_SILENT_COMMUNICATION`	--
	`CANSM_BSWM_FULL_COMMUNICATION`	--
	`CANSM_BSWM_BUS_OFF`	--
	`CANSM_BSWM_CHANGE_BAUDRATE`	--
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 module.

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	

⌋(BSW0405, BSW101, BSW00406, BSW00358, BSW00414, BSW00405, BSW00404)

[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`.⌋(BSW00406)

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:	Reentrant
Parameters (in):	None
Parameters	None

(inout):		
Parameters (out):	VersionInfo	Pointer to where to store the version information of this module.
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)	

⌋(BSW00407, BSW003)

[CANSM366] ⌈If the source code for caller and callee of `CanSM_GetVersionInfo` is available this function should be realized as a macro. The macro should be defined in the header file CanSM.h.⌋()

[CANSM244] ⌈The function `CanSM_GetVersionInfo` shall return the version information of this module. The version information includes:

- Module Id
- Vendor Id
- Vendor specific version numbers (BSW00407).⌋()

[CANSM367] ⌈The function `CanSM_GetVersionInfo` shall be configurable On/Off by the configuration parameter: `CANSM_VERSION_INFO_API` (ref. to CANSM311_Conf).⌋()

[CANSM368] ⌈The function `CanSM_GetVersionInfo` shall be pre compile time configurable by the configuration parameter: `CANSM_VERSION_INFO_API` (ref. to CANSM311_Conf).⌋()

[CANSM374] ⌈The function `CanSM_GetVersionInfo` shall report the development error `CANSM_E_PARAM_POINTER` to the DET, if the user of this function hands over a NULL-pointer as VersionInfo.⌋()

8.3.3 CanSM_RequestComMode

[CANSM062] ⌈

Service name:	CanSM_RequestComMode	
Syntax:	`Std_ReturnType CanSM_RequestComMode( NetworkHandleType network, ComM_ModeType ComM_Mode )`	
Service ID[hex]:	0x02	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant (only for different network handles)	
Parameters (in):	network	Handle of destined communication network for request
	ComM_Mode	Requested communication mode
Parameters	None	

(inout):		
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.	

⌋(BSW01142, BSW09080, BSW09081, BSW09083)

Remark: Please refer to [10] for a detailed description of the communication modes.

[CANSM369] ⌈The function `CanSM_RequestComMode` shall accept its request, if the `NetworkHandle` parameter of the request is a handle contained in the configuration of the CanSM module (ref. to CANSM161_Conf).⌋()

[CANSM370] ⌈The function `CanSM_RequestComMode` shall deny its request, if the `NetworkHandle` parameter of the request is not a handle contained in the configuration of the CanSM module (ref. to CANSM161_Conf).⌋()

[CANSM555] ⌈The CanSM module shall deny the API request `CanSM_RequestComMode`, if the initial transition for the requested CAN network is not finished yet after the `CanSM_Init` request (ref. to CANSM423, CANSM430).⌋()

[CANSM183] ⌈The function `CanSM_RequestComMode` shall call the function `Det_ReportError` with `ErrorId` parameter `CANSM_E_INVALID_NETWORK_HANDLE`, if it does not accept the network handle of the request.⌋()

[CANSM402] ⌈The function `CanSM_RequestComMode` shall deny its request, if the current network mode is `COMM_NO_COMMUNICATION` and the user of this function requests `COMM_SILENT_COMMUNICATION`.⌋()

Rationale: The only use case for silent communication is to prepare bus sleep on CAN. Therefore a transition from no communication to silent communication is invalid.

[CANSM403] ⌈If the function `CanSM_RequestComMode` denies its request, because of an invalid requested transistion, it shall invocate the function `Det_ReportError` with `ErrorId` parameter `CANSM_E_INVALID_COMM_REQUEST`.⌋()

[CANSM182] ⌈If the function `CanSM_RequestComMode` accepts the request, the request shall be considered by the CanSM state machine (ref. to CANSM427, CANSM429, CANSM499, CANSM542 and CANSM543).⌋()

[CANSM184] ⌈If the CanSM module is not initialized, when the function `CanSM_RequestComMode` is called, then this function shall call the function `Det_ReportError` with `ErrorId` parameter `CANSM_E_UNINIT`.⌋()

[CANSM395] ⌈If the CanSM module has to deny the request `CanSM_RequestComMode`, because of a pending mode indication (ref. to CANSM388), then this function shall call the function `Det_ReportError` with the `ErrorId` parameter `CANSM_E_WAIT_MODE_INDICATION` (ref. to chapter 7.3).⌋()

8.3.4 CanSM_GetCurrentComMode

[CANSM063] ⌈

Service name:	CanSM_GetCurrentComMode	
Syntax:	`Std_ReturnType CanSM_GetCurrentComMode( NetworkHandleType network, ComM_ModeType* ComM_ModePtr )`	
Service ID[hex]:	0x03	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	network	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.	

⌋(BSW01142, BSW09080, BSW09084)

[CANSM282] ⌈The CanSM module shall return `E_NOT_OK` for the API request `CanSM_GetCurrentComMode`, if the initial transition for the requested CAN network with `E_NOCOM` (ref. to CANSM430) is not finished yet.⌋()

[CANSM371] ⌈The function `CanSM_GetCurrentComMode` shall accept its request, if the `NetworkHandle` parameter of the request is a handle contained in the configuration of the CanSM module (ref. to CANSM161_Conf).⌋()

[CANSM372] ⌈The function `CanSM_GetCurrentComMode` shall deny its request, if the `NetworkHandle` parameter of the request is not a handle contained in the configuration of the CanSM module (ref. to CANSM161_Conf).⌋()

[CANSM187] ⌈The function `CanSM_GetCurrentComMode` shall call the function `Det_ReportError` with `ErrorId` parameter `CANSM_E_INVALID_NETWORK_HANDLE`, if it does not accept the network handle of the request.⌋()

[CANSM186] ⌈The function `CanSM_GetCurrentComMode` shall put out the current communication mode for the network handle (ref. to CANSM266) to the designated pointer of type `ComM_ModeType`, if it accepts the request.⌋()

[CANSM188] ⌈If the CanSM module is not initialized, when the function `CanSM_GetCurrentComMode` is called, then this function shall call the function `Det_ReportError` with `ErrorId` parameter `CANSM_E_UNINIT`.⌋()

[CANSM360] ⌈The function `CanSM_GetCurrentComMode` shall report the development error `CANSM_E_PARAM_POINTER` to the DET, if the user of this function hands over a NULL-pointer as `ComM_ModePtr`.⌋()

8.3.5 CanSM_CheckBaudrate

[CANSM501] ⌈

Service name:	CanSM_CheckBaudrate	
Syntax:	`Std_ReturnType CanSM_CheckBaudrate( NetworkHandleType network, const uint16 Baudrate )`	
Service ID[hex]:	0x0c	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	network	Handle of the addressed CAN network to check if a baudrate is supported
	Baudrate	Baudrate to check in kbps
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Baudrate supported by all configured CAN controllers of the network E_NOT_OK: Baudrate not supported / invalid network
Description:	This service shall check, if a certain baudrate is supported by the configured CAN controllers of a certain CAN network.	

⌋()

CANSM564: ⌈The CanSM module shall provide the API function `CanSM_CheckBaudrate`, if the `CanSmChangeBaudrateApi` parameter (ref. to CANSM342_Conf) is configured with the value `TRUE`. ⌋()

CANSM565: ⌈The CanSM module shall not provide the API function `CanSM_CheckBaudrate`, if the `CanSmChangeBaudrateApi` parameter (ref. to CANSM342_Conf) is configured with the value `FALSE.` ⌋()

[CANSM562] ⌈The CanSM module shall deny the `CanSM_CheckBaudrate` API request, if the `NetworkHandle` parameter does not match to the configured Network handles of the CanSM module (ref. to CANSM161_Conf).⌋()

[CANSM571] ⌈The function `CanSM_CheckBaudrate` shall call the function `Det_ReportError` with `ErrorId` parameter `CANSM_E_INVALID_NETWORK_HANDLE` (ref. to chapter 7.3), if it does not accept the network handle of the request.⌋()

[CANSM563] ⌈If the `NetworkHandle` parameter in the `CanSM_CheckBaudrate` request matches to one of the configured Network handles (ref. to CANSM161_Conf) and the requested baud rate is supported (ref. to CANSM567), then the function shall return `E_OK`. ⌋()

[**CANSM566**] ⌈If the `NetworkHandle` parameter in the `CanSM_CheckBaudrate` request matches to one of the configured Network handles (ref. to CANSM161_Conf) and the requested baud rate is not supported (ref. to CANSM568), then the function shall return `E_NOT_OK`. ⌋()

8.3.6 CanSM_ChangeBaudrate

[CANSM561] ⌈

Service name:	CanSM_ChangeBaudrate	
Syntax:	`Std_ReturnType CanSM_ChangeBaudrate(NetworkHandleType network, const uint16 Baudrate)`	
Service ID[hex]:	0x0e	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant	
Parameters (in):	network	Handle of the addressed CAN network for the baudrate change
	Baudrate	Requested Baudrate in kbps

Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Service request accepted E_NOT_OK: Service request not accepted
Description:	This service shall start an asynchronous process to change the baudrate for the configured CAN controllers of a certain CAN network	

[CANSM569] ⌈The CanSM module shall provide the API function `CanSM_ChangeBaudrate`, if the `CanSmChangeBaudrateApi` parameter (ref. to CANSM342_Conf) is configured with the value `TRUE`. ⌋()

[CANSM570] The CanSM module shall not provide the API function `CanSM_ChangeBaudrate`, if the `CanSmChangeBaudrateApi` parameter (ref. to CANSM342_Conf) is configured with the value `FALSE.` ⌋()

[CANSM502] ⌈The CanSM module shall deny the `CanSM_ChangeBaudrate` API request, if the `NetworkHandle` parameter does not match to the configured Network handles of the CanSM module (ref. to CANSM161_Conf).⌋()

[CANSM504] ⌈The function `CanSM_ChangeBaudrate` shall call the function `Det_ReportError` with `ErrorId` parameter `CANSM_E_INVALID_NETWORK_HANDLE` (ref. to chapter 7.3), if it does not accept the network handle of the request.⌋()

[CANSM505] ⌈The function `CanSM_ChangeBaudrate` shall deny its request, if the requested CAN network is not in the communication mode `COMM_FULL_COMMUNICATION`.⌋()

[CANSM530] ⌈The CanSM module shall deny the `CanSM_ChangeBaudrate` API request, if the CanSM module is not initialized.⌋()

[CANSM506] ⌈If the function `CanSM_ChangeBaudrate` is called and the CanSM module is not initialized, then this function shall call the function `Det_ReportError` with `ErrorId` parameter `CANSM_E_UNINIT` (ref. to chapter 7.3).⌋()

[CANSM573] ⌈If the requested baud rate is not equal to the remembered baud rate of the last `CanSM_CheckBaudrate` call (ref. to CANSM572) for the corresponding CAN network or if the remembered result of the last `CanSM_CheckBaudrate` call for the corresponding CAN network has been E_NOT_OK, then the CanSM_ChangeBaudrate call shall return E_NOT_OK. ⌋()

[CANSM574] ⌈If the requested baud rate is not equal to the remembered baud rate of the last CanSM_CheckBaudrate call (ref. to CANSM572) for the corresponding CAN network or if the remembered result of the last CanSM_CheckBaudrate call for the corresponding CAN network has been E_NOT_OK, then the CanSM_ChangeBaudrate call the function Det_ReportError with ErrorId parameter CANSM_E_PARAM_INVALID_BAUDRATE (ref. to chapter 7.3). ⌋()

[CANSM503] ⌈If no condition is present to deny the CanSM_ChangeBaudrate request according to CANSM502, CANSM505, CANSM530 and CANSM573, then the CanSM module shall return E_OK and start the asynchronous process to change the baud rate of the CAN network's CAN Controllers to the checked and requested baud rate (ref. to CANSM507).⌋()

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 ControllerId)	
Service ID[hex]:	0x04	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant (only for different CanControllers)	
Parameters (in):	ControllerId	CAN controller, which detected a bus-off event
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	This callback function notifies the CanSM about a bus-off event on a certain CAN controller, which needs to be considered with the specified bus-off recovery handling for the impacted CAN network.	

⌋(BSW00359, BSW00333, BSW01146)

[CANSM189] ⌈If the function `CanSM_ControllerBusOff` gets a `Controller`, which is not configured as `CanSMControllerId` in the configuration of the CanSM module, it shall call the function `Det_ReportError` with `ErrorId` parameter `CANSM_E_PARAM_CONTROLLER`.⌋()

[CANSM190] ⌈If the CanSM module is not initialized, when the function `CanSM_ControllerBusOff` is called, then the function `CanSM_ControllerBusOff` shall call the function `Det_ReportError` with `ErrorId` parameter `CANSM_E_UNINIT`.⌋()

[CANSM377] ⌈If the CanSM module has to deny the request `CanSM_RequestComMode`, because of a not finished bus-off recovery (ref. to CANSM375 and CANSM376), then this function shall call the function `Det_ReportError` with the `ErrorId` parameter `CANSM_E_BUSOFF_RECOVERY_ACTIVE` (ref. to chapter 7.3).⌋()

[CANSM235] ⌈If the CanSM module is initialized and the input parameter `Controller` is one of the CAN controllers configured with the parameter `CanSMControllerId`, this bus-off event shall be considered by the CAN Network state machine (ref. to CANSM500).⌋()

Additional 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_ControllerModeIndication

[CANSM396] ⌈

Service name:	CanSM_ControllerModeIndication	
Syntax:	`void CanSM_ControllerModeIndication( uint8 ControllerId, CanIf_ControllerModeType ControllerMode )`	
Service ID[hex]:	0x07	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant (only for different CAN controllers)	
Parameters (in):	ControllerId	CAN controller, whose mode has changed
	ControllerMode	Notified CAN controller mode
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	This callback shall notify the CanSM module about a CAN controller mode	

	change.

⌋()

[CANSM397] ⌈If the function `CanSM_ControllerModeIndication` gets a `ControllerId`, which is not configured as `CanSMControllerId` in the configuration of the CanSM module, it shall call the function `Det_ReportError` with `ErrorId` parameter `CANSM_E_PARAM_CONTROLLER`.⌋()

[CANSM398] ⌈If the CanSM module is not initialized, when the function `CanSM_ControllerModeIndication` is called, then the function `CanSM_ControllerModeIndication` shall call the function `Det_ReportError` with `ErrorId` parameter `CANSM_E_UNINIT`.⌋()

8.4.3 CanSM_TransceiverModeIndication

[CANSM399] ⌈

Service name:	CanSM_TransceiverModeIndication	
Syntax:	`void CanSM_TransceiverModeIndication( uint8 TransceiverId, CanTrcv_TrcvModeType TransceiverMode )`	
Service ID[hex]:	0x09	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant for different CAN Transceivers	
Parameters (in):	TransceiverId	CAN transceiver, whose mode has changed
	TransceiverMode	Notified CAN transceiver mode
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	This callback shall notify the CanSM module about a CAN transceiver mode change.	

⌋()

[CANSM400] ⌈If the function `CanSM_TransceiverModeIndication` gets a `TransceiverId`, which is not configured as `CanSMTransceiverId` in the configuration of the CanSM module, it shall call the function `Det_ReportError` with `ErrorId` parameter `CANSM_E_PARAM_TRANSCEIVER`.⌋()

[CANSM401] ⌈If the CanSM module is not initialized, when the function `CanSM_TransceiverModeIndication` is called, then the function `CanSM_TransceiverModeIndication` shall call the function `Det_ReportError` with `ErrorId` parameter `CANSM_E_UNINIT`.⌋()

8.4.4 CanSM_TxTimeoutException

[CANSM410] ⌈

Service name:	CanSM_TxTimeoutException	
Syntax:	`void CanSM_TxTimeoutException( NetworkHandleType Channel )`	
Service ID[hex]:	0x0b	
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.	

⌋()

[CANSM411] ⌈The function `CanSM_TxTimeoutException` shall report `CANSM_E_UNINIT` to the DET, if the CanSM is not initialized yet.⌋()

[CANSM412] ⌈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.

8.4.5 CanSM_ClearTrcvWufFlagIndication

[CANSM413] ⌈

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.	

⌋()

[CANSM414] ⌈The function `CanSM_ClearTrcvWufFlagIndication` shall report `CANSM_E_UNINIT` to the DET, if the CanSM is not initialized yet.⌋()

[CANSM415] ⌈If the function `CanSM_ClearTrcvWufFlagIndication` gets a `TransceiverId`, which is not configured (ref. to CANSM137_Conf) 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_CheckTransceiverWakeFlagIndication

[CANSM416] ⌈

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.	

⌋()

[CANSM417] ⌈The function `CanSM_CheckTransceiverWakeFlagIndication` shall report `CANSM_E_UNINIT` to the DET, if the CanSM module is not initialized yet.⌋()

[CANSM418] ⌈If the function `CanSM_CheckTransceiverWakeFlagIndication` gets a `TransceiverId`, which is not configured (ref. to CANSM137_Conf) in the configuration of the CanSM module, it shall call the function `Det_ReportError` with `ErrorId` parameter `CANSM_E_PARAM_TRANSCEIVER`.⌋()

8.4.7 CanSM_ConfirmPnAvailability

[CANSM419] ⌈

Service name:	CanSM_ConfirmPnAvailability	
Syntax:	`void CanSM_ConfirmPnAvailability( uint8 TransceiverId )`	
Service ID[hex]:	0x06	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	TransceiverId	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.	

⌋()

[CANSM546] ⌈The function `CanSM_ConfirmPnAvailability` shall notify the CanNm module (ref. to CANSM422), if it is called with a configured Transceiver as input parameter (ref. to CANSM137_Conf).⌋()

[CANSM420] ⌈
The function `CanSM_ConfirmPnAvailability` shall report `CANSM_E_UNINIT` to the DET, if the CanSM module is not initialized yet.⌋()

[CANSM421] ⌈
If the function `CanSM_ConfirmPnAvailability` gets a `TransceiverId`, which is not configured (ref. to CANSM137_Conf) 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

These functions are directly called by Basic Software Scheduler. 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( void )`
Service ID[hex]:	0x05
Timing:	FIXED_CYCLIC
Description:	Scheduled function of the CanSM

⌋(BSW0424, BSW00425, BSW00376)

[CANSM167] ⌈The main function of the CanSM module shall operate the effects of the CanSM state machine (ref. to chapter 7.2), which the CanSM module shall implement for each configured CAN Network.⌋()

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	This service calls the corresponding CAN Driver service for changing of the CAN controller mode.
CanIf_SetPduMode	This service sets the requested mode at the L-PDUs of a predefined logical PDU channel.
CanIf_SetTrcvMode	This service changes the operation mode of the tansceiver TransceiverId, via calling the corresponding CAN Transceiver Driver service.
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 and BswM.
Dem_ReportErrorStatus	Queues the reported events from the BSW modules (API is only used by BSW modules). The interface has an asynchronous behavior,

	because the processing of the event is done within the Dem main function.

8.6.2 Optional Interfaces

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

API function	***Description***
CanIf_ChangeBaudrate	This service shall change the baudrate of the CAN controller.
CanIf_CheckBaudrate	This service shall check, if a certain CAN controller supports a requested baudrate
Det_ReportError	Service to report development errors.

8.6.3 Configurable Interfaces

In this chapter all interfaces are listed where the target functions could be configured. The target function is usually a callback function. The names of these kind of interfaces is not fixed because they are configurable.

There are no configurable interfaces for the CanSM module.

9 Sequence diagrams

All interactions of the CanSM module with the depending modules CanIf, ComM, BswM, Dem and CanNm and Dcm are specified in the state machine diagrams (ref. to Figure 7-1- Figure 7-7). Therefore the CanSM SWS provides only some exemplary sequences for the use case to operate the DCM request to change the baud rate. This also includes the sequences to start and to stop the CAN controller(s) of a CAN network.

Remark: 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).

9.1 Sequence for baud rate change request from the DCM module

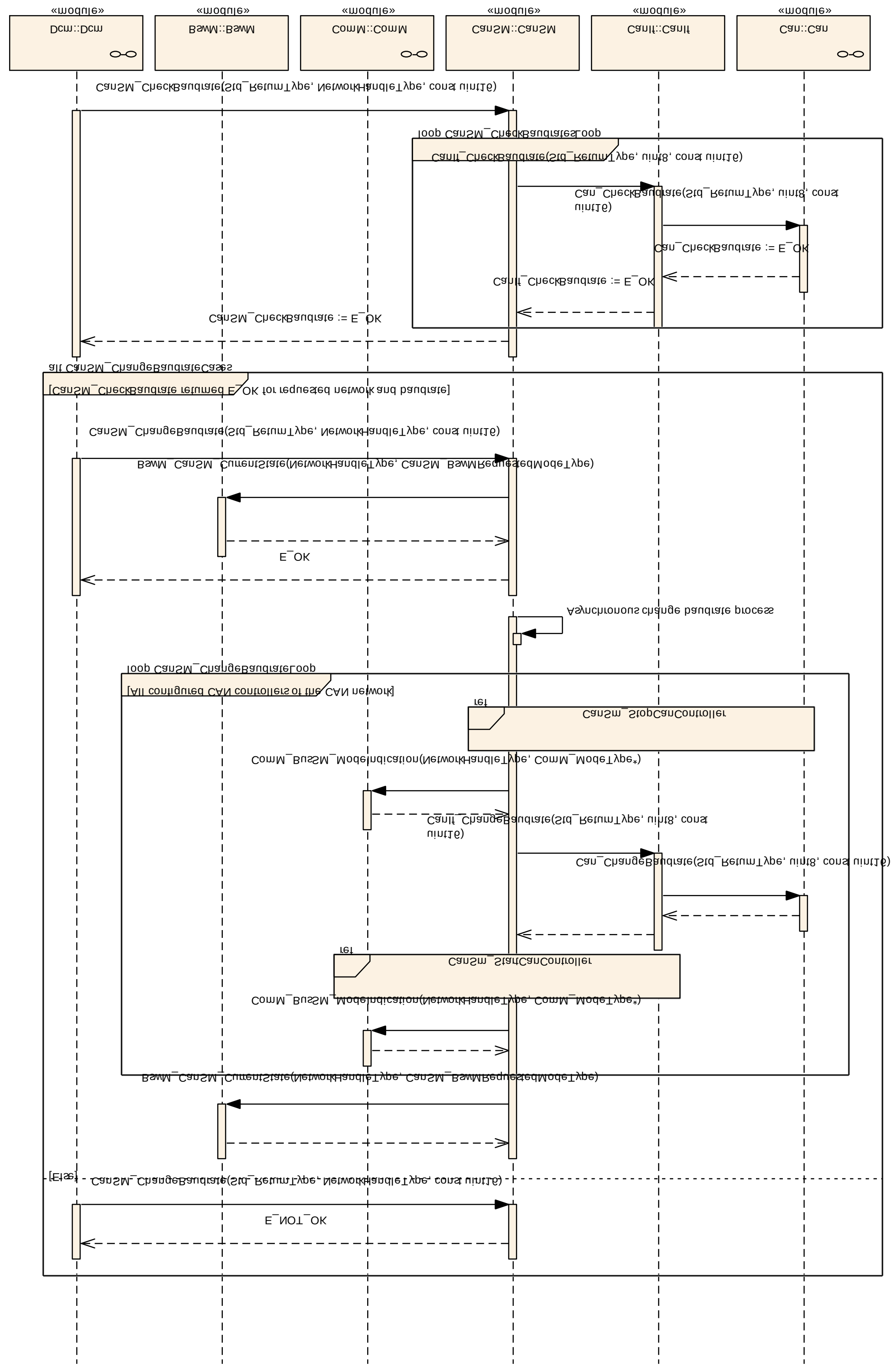
«module» Dcm::Dcm
«module» BswM::BswM
«module» ComM::ComM
«module» CanSM::CanSM
«module» CanIf::CanIf
«module» Can::Can
CanSM_CheckBaudrate(Std_ReturnType, NetworkHandleType, const uint16)
loop CanSM_CheckBaudratesLoop
CanIf_CheckBaudrate(Std_ReturnType, uint8, const uint16)
Can_CheckBaudrate(Std_ReturnType, uint8, const uint16)
Can_CheckBaudrate := E_OK
CanIf_CheckBaudrate := E_OK
CanSM_CheckBaudrate := E_OK
alt CanSM_ChangeBaudrateCases
[CanSM_CheckBaudrate returned E_OK for requested network and baudrate]
CanSM_ChangeBaudrate(Std_ReturnType, NetworkHandleType, const uint16)
BswM_CanSM_CurrentState(NetworkHandleType, CanSM_BswMRequestedModeType)
E_OK
Asynchronous change baudrate process
loop CanSM_ChangeBaudrateLoop
[All configured CAN controllers of the CAN network]
ref
CanSm_StopCanController
ComM_BusSM_ModeIndication(NetworkHandleType, ComM_ModeType*)
CanIf_ChangeBaudrate(Std_ReturnType, uint8, const uint16)
Can_ChangeBaudrate(Std_ReturnType, uint8, const uint16)
ref
CanSm_StartCanController
ComM_BusSM_ModeIndication(NetworkHandleType, ComM_ModeType*)
BswM_CanSM_CurrentState(NetworkHandleType, CanSM_BswMRequestedModeType)
[Else]
CanSM_ChangeBaudrate(Std_ReturnType, NetworkHandleType, const uint16)
E_NOT_OK

figure 9-1: Sequence for baud rate change request from the DCM module

9.2 Sequence diagram CanSm_StartCanController

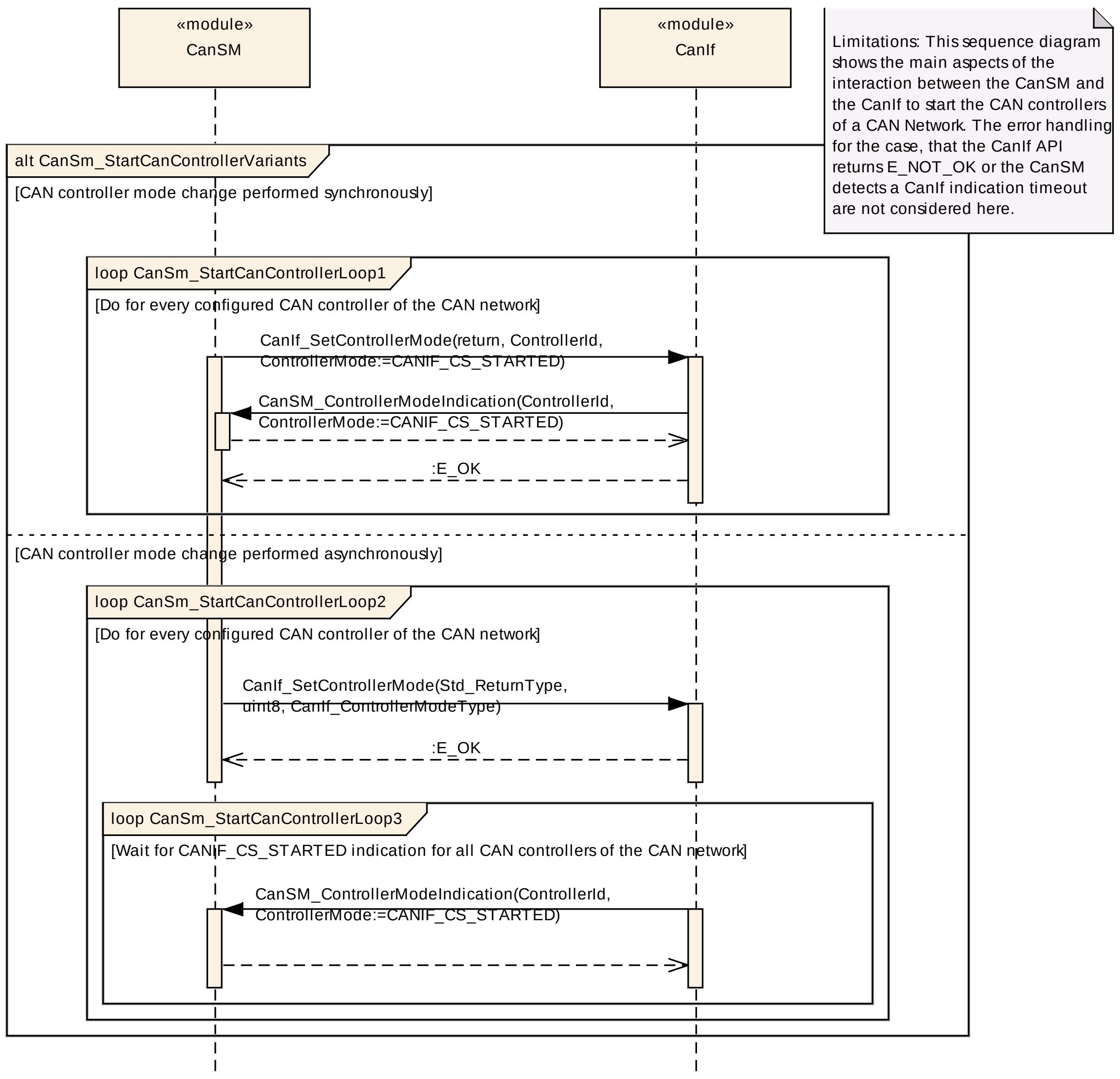
«module»
CanSM
«module»
CanIf
Limitations: This sequence diagram shows the main aspects of the interaction between the CanSM and the CanIf to start the CAN controllers of a CAN Network. The error handling for the case, that the CanIf API returns E_NOT_OK or the CanSM detects a CanIf indication timeout are not considered here.
alt CanSm_StartCanControllerVariants
[CAN controller mode change performed synchronously]
loop CanSm_StartCanControllerLoop1
[Do for every configured CAN controller of the CAN network]
CanIf_SetControllerMode(return, ControllerId, ControllerMode:=CANIF_CS_STARTED)
CanSM_ControllerModeIndication(ControllerId, ControllerMode:=CANIF_CS_STARTED)
:E_OK
[CAN controller mode change performed asynchronously]
loop CanSm_StartCanControllerLoop2
[Do for every configured CAN controller of the CAN network]
CanIf_SetControllerMode(Std_ReturnType, uint8, CanIf_ControllerModeType)
:E_OK
loop CanSm_StartCanControllerLoop3
[Wait for CANIF_CS_STARTED indication for all CAN controllers of the CAN network]
CanSM_ControllerModeIndication(ControllerId, ControllerMode:=CANIF_CS_STARTED)

figure 9-2: Sequence diagram CanSm_StartCanController

9.3 Sequence diagram CanSm_StopCanController

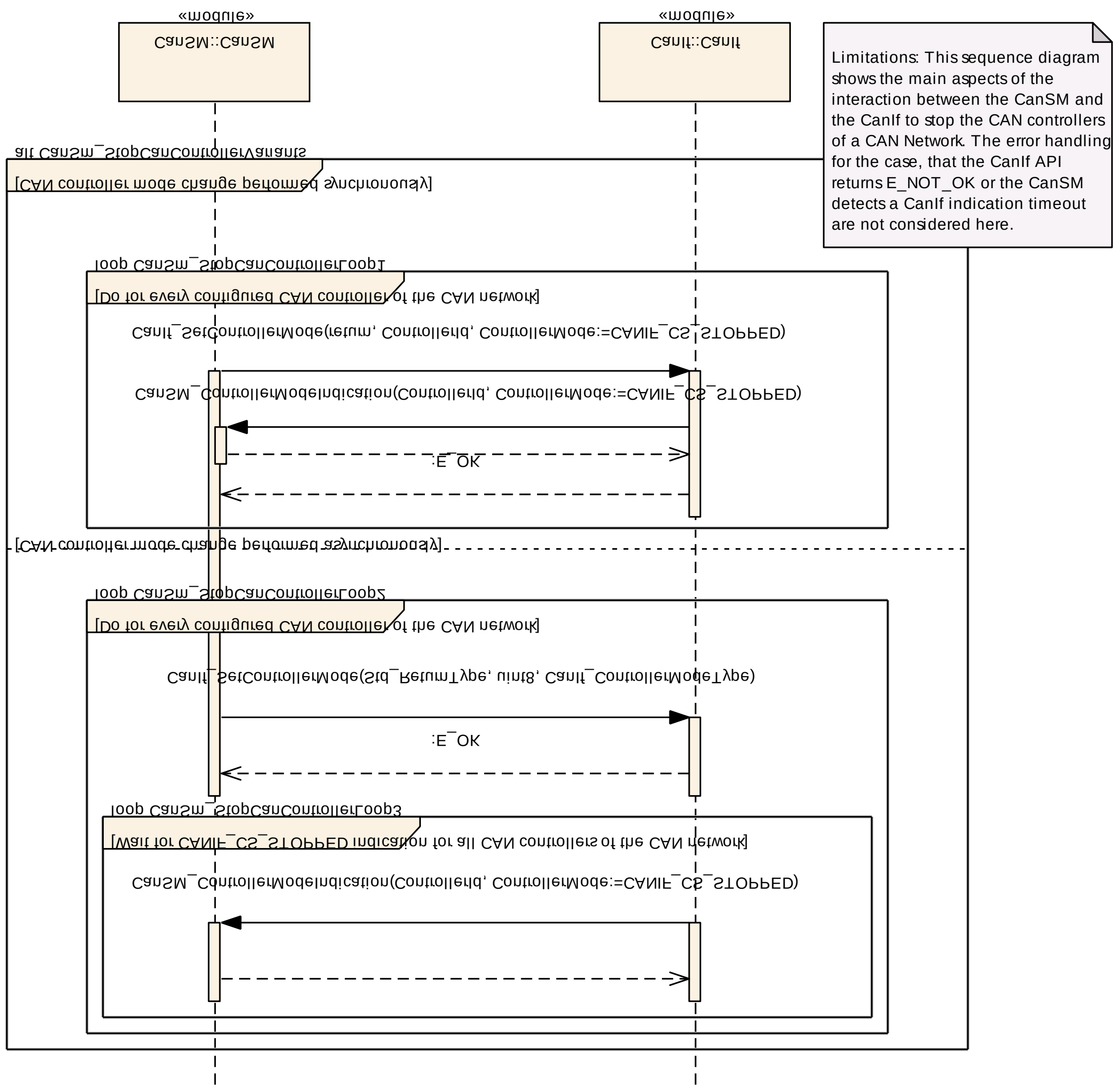
«module»
CanSM::CanSM
«module»
CanIf::CanIf
Limitations: This sequence diagram shows the main aspects of the interaction between the CanSM and the CanIf to stop the CAN controllers of a CAN Network. The error handling for the case, that the CanIf API returns E_NOT_OK or the CanSM detects a CanIf indication timeout are not considered here.
alt CanSm_StopCanControllerVariants
[CAN controller mode change performed synchronously]
loop CanSm_StopCanControllerLoop1
[Do for every configured CAN controller of the CAN network]
CanIf_SetControllerMode(return, ControllerId, ControllerMode:=CANIF_CS_STOPPED)
CanSM_ControllerModeIndication(ControllerId, ControllerMode:=CANIF_CS_STOPPED)
:E_OK
[CAN controller mode change performed asynchronously]
loop CanSm_StopCanControllerLoop2
[Do for every configured CAN controller of the CAN network]
CanIf_SetControllerMode(Std_ReturnType, uint8, CanIf_ControllerModeType)
:E_OK
loop CanSm_StopCanControllerLoop3
[Wait for CANIF_CS_STOPPED indication for all CAN controllers of the CAN network]
CanSM_ControllerModeIndication(ControllerId, ControllerMode:=CANIF_CS_STOPPED)

figure 9-3: Sequence diagram CanSm_StopCanController

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.

10.2 Containers and configuration parameters

The following chapters summarize all configuration parameters of the CanSM module. The detailed meanings of the parameters describe chapter 7 and chapter 8.

10.2.1 Variants

[CANSM250] ⌈VARIANT-PRE-COMPILE: Only pre-compile parameters⌋()

[CANSM251] ⌈VARIANT-LINK-TIME: Mix of pre-compile and link time parameters⌋()

[CANSM252] ⌈VARIANT-POST-BUILD: Mix of pre compile-, link time and post build time parameters⌋()

10.2.2 CanSM

Module Name	*CanSM*
Module Description	Configuration of the CanSM module

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CanSMConfiguration	1	This container contains the global parameters of the CanSM and sub containers, which are for the CAN network specific configuration.

CanSMGeneral	1	Container for general pre-compile parameters of the CanSM module

10.2.3 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	EcucBooleanParamDef		
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_TIME_PERIOD}		
Description	This parameter defines the cycle time of the function CanSM_MainFunction in seconds		
Multiplicity	1		
Type	EcucFloatParamDef		
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	**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	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Local		

SWS Item	**CANSM342_Conf :**
Name	CanSmChangeBaudrateApi {CANSM_CHANGE_BAUDRATE_API}
Description	The support of the Can_ChangeBaudrate API is optional. If this parameter is set to true the Can_ChangeBaudrate API shall be supported. Otherwise the API is not supported.

Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

No Included Containers

10.2.4 CanSMConfiguration

SWS Item	**CANSM123_Conf :**
Container Name	CanSMConfiguration [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	**CANSM335_Conf :**		
Name	CanSMModeRequestRepetitionMax {CANSM_MODEREQ_MAX}		
Description	Specifies the maximal amount of mode request repetitions without a respective mode indication from the CanIf module until the CanSM module reports a development error to the DET and tries to go back to no communication.		
Multiplicity	1		
Type	EcucIntegerParamDef		
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	**CANSM336_Conf :**		
Name	CanSMModeRequestRepetitionTime {CANSM_MODEREQ_REPEAT_TIME}		
Description	Specifies in which time duration the CanSM module shall repeat mode change requests by using the API of the CanIf module.		
Multiplicity	1		
Type	EcucFloatParamDef		
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		

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

10.2.5 CanSMManagerNetwork

SWS Item	**CANSM126_Conf :**	
Container Name	CanSMManagerNetwork	
Description	This container contains the CAN network specific parameters of each CAN network	
Configuration Parameters		

SWS Item	**CANSM131_Conf :**		
Name	CanSMBorCounterL1ToL2 {CANSM_BOR_COUNTER_L1_TO_L2}		
Description	This threshold defines the count of bus-offs until the bus-off recovery switches from level 1 (short recovery time) to level 2 (long recovery time).		
Multiplicity	1		
Type	EcucIntegerParamDef		
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_Conf :**		
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	EcucFloatParamDef		
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_Conf :**		
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	EcucFloatParamDef		
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_Conf :**
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 (e. g. time period of the fastest cyclic transmitted PDU of

	the COM module / ComTxModeTimePeriodFactor).		
Multiplicity	1		
Type	EcucFloatParamDef		
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_Conf :**		
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	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Local		

SWS Item	**CANSM161_Conf :**		
Name	CanSMComMNetworkHandleRef {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_Conf :**		
Name	CanSMTransceiverId {CANSM_TRANSCEIVER_ID}		
Description	ID of the CAN transceiver assigned to the configured network handle. Reference to one of the transceivers managed by the CanIf module.		
Multiplicity	0..1		
Type	Reference to [CanIfTrcvCfg]		
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		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CanSMController	1..*	This container contains the controller IDs assigned to a CAN network.
CanSMDemEventParameterRefs	0..1	Container for the references to DemEventParameter elements which shall be invoked using the API

		Dem_ReportErrorStatus API in case the corresponding error occurs. The EventId is taken from the referenced DemEventParameter's DemEventId value. The standardized errors are provided in the container and can be extended by vendor specific error references.

10.2.6 CanSMDemEventParameterRefs

SWS Item	**CANSM127_Conf :**
Container Name	CanSMDemEventParameterRefs
Description	Container for the references to DemEventParameter elements which shall be invoked using the API Dem_ReportErrorStatus API in case the corresponding error occurs. The EventId is taken from the referenced DemEventParameter's DemEventId value. The standardized errors are provided in the container and can be extended by vendor specific error references.
Configuration Parameters	

SWS Item	**CANSM070_Conf :**		
Name	CANSM_E_BUS_OFF {CANSM_E_BUS_OFF}		
Description	Reference to configured DEM event to report bus off errors for this CAN network.		
Multiplicity	0..1		
Type	Reference to [DemEventParameter]		
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: Dem		

No Included Containers

10.2.7 CanSMController

SWS Item	**CANSM338_Conf :**
Container Name	CanSMController
Description	This container contains the controller IDs assigned to a CAN network.
Configuration Parameters	

SWS Item	**CANSM141_Conf :**		
Name	CanSMControllerId {CANSM_CONTROLLER_ID}		
Description	Unique handle to identify one certain CAN controller. Reference to one of the CAN controllers managed by the CanIf module.		
Multiplicity	1		
Type	Reference to [CanIfCtrlCfg]		
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

[CANSM559]「The standardized common published parameters as required by BSW00402 in the General Requirements on Basic Software Modules [3] shall be published within the header file of this module and need to be provided in the BSW Module Description. The according module abbreviation can be found in the List of Basic Software Modules [1].」()

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

11 Changes between AR3.0 and AR4.0 rev001

11.1 Deleted SWS Items

SWS Item	*Rationale*
CANSM061	Cleared to solve issues of improvement process
CANSM211	Cleared to solve issues of improvement process
CANSM212	Cleared to solve issues of improvement process
CANSM214	Cleared to solve issues of improvement process
CANSM215	Cleared to solve issues of improvement process
CANSM216	Cleared to solve issues of improvement process
CANSM169	Cleared to solve issues of improvement process
CANSM236	Cleared to solve issues of improvement process
CANSM155	Requirement ID from standard text removed
CANSM172	
CANSM262	
CANSM263	
CANSM267	
CANSM274	
CANSM277	
CANSM291	
CANSM296	
CANSM300	
CANSM175	
CANSM138	
CANSM139	
CANSM198	
CANSM077	
CANSM076	
CANSM078	
CANSM079	
CANSM290	Work on clarification
CANSM293	Avoid mix of use cases for peripheral requests and mode transitions between communication mode requests from ComM and bus-off handling
CANSM037	Cleared after improvement process
CANSM007	Cleared to consider latest SWS template
CANSM242	Cleared to consider latest SWS template
CANSM249	Deleted to solve
CANSM027	Deleted to solve
CANSM180	Deleted to solve
CANSM181	Deleted to solve
CANSM185	Deleted to solve
CANSM303	
CANSM315	): Bus-off recovery handling sequence misinterpretation
CANSM316_Conf CANSM270 CANSM317 CANSM318 CANSM320 CANSM319 CANSM321 CANSM323 CANSM325 CANSM326 CANSM357 CANSM322	Deleted during changes to solve [Can][CanIf][CanSm][CanTrcv] Full COM Request have to be asynchronous but is specified synchronous

CANSM328 CANSM329 CANSM330 CANSM358 CANSM331 CANSM359 CANSM332 CANSM333 CANSM356 CANSM344 CANSM345 CANSM346 CANSM343	

11.2 Replaced SWS Items

SWS Item	***replaced by SWS Item***	***Rationale***
CANSM057	CANSM287, CANSM288	Made requirement atomic
CANSM122	CANSM250, CANSM251, CANSM252	One requirement per variant

11.3 Changed SWS Items

SWS Item	*Change*	*Rationale*
CANSM027	Linefeed removed within standard requirement	
CANSM074	Changed to prescribed standard text	
CANSM128	Removed to CanStateManagerNetworks Container in ECUCParameterDefinitions of Meta Model and new generation of artifacts	
CANSM129	Removed to CanStateManagerNetworks Container in ECUCParameterDefinitions of Meta Model and new generation of artifacts	
CANSM024	CanSM_GetVersionInfo corrected in BSW Model and new generation of artifacts	
CANSM123	Multiplicity changed in MM and new generation of artifacts	
CANSM397	Refer to new requirements CANSM340-CANSM342	Avoid mix of use cases for peripheral requests and mode transitions between communication mode requests from ComM and bus-off handling
CANSM070	Specified as configuration parameter of the CanSM, which references as diagnostic event parameter from the DEM	Optimize configuration
CANSM334	Specified as configuration parameter of the CanSM, which references as diagnostic event parameter from the DEM	Optimize configuration
CANSM250	Variant 1 renamed into VARIANT-PRE-COMPILE	
CANSM251	Variant 2 renamed into VARIANT-LINK-TIME	
CANSM252	Variant 3 renamed into VARIANT-POST-BUILD	
CANSM161_Conf	CanSMNetworkHandle => CanSMComMNetworkHandleRef	
CANSM174	“The CanSM module (CanSM.c) shall include the header file ComM.h” changed into “the header file CanSM.h shall include the header file ComM.h”	
CANSM289	CANSM_CS_STARTED replaced with CANIF_CS_STARED	Typo
CANSM265	Instruction order changed (first callback, then internal state change)	
CANSM349	Instruction order changed (first callback, then internal state change)	
CANSM256	Instruction order changed (first callback, then internal state change)	
CANSM261	Instruction order changed (first callback, then internal state change)	
CANSM276	Instruction order changed (first callback, then internal state change)	
CANSM281	Instruction order changed (first callback, then internal state change)	
CANSM353	Instruction order changed (first callback, then internal state change)	
CANSM315	Typo: therfore => therefore	
CANSM279	Typo (missing <)>)	
CANSM235	Formulation (regard => handle)	
CANSM346	Formulation (consider => handle)	

CAMSM353	Typo: Full replaced with silent; wrong requirement references corrected	
CANSM349	BswM_CanSM_RequestMode changed into BswM_CanSM_CurrentState	
CANSM350	BswM_CanSM_RequestMode changed into BswM_CanSM_CurrentState	
CANSM351	BswM_CanSM_RequestMode changed into BswM_CanSM_CurrentState	
CANSM352	BswM_CanSM_RequestMode changed into BswM_CanSM_CurrentState	
CANSM281	BswM_CanSM_RequestMode changed into BswM_CanSM_CurrentState	
CANSM354	BswM_CanSM_RequestMode changed into BswM_CanSM_CurrentState	
CANSM355	BswM_CanSM_RequestMode changed into BswM_CanSM_CurrentState	
CANSM356	BswM_CanSM_RequestMode changed into BswM_CanSM_CurrentState	
CANSM357	BswM_CanSM_RequestMode changed into BswM_CanSM_CurrentState	
CANSM358	BswM_CanSM_RequestMode changed into BswM_CanSM_CurrentState	
CANSM359	BswM_CanSM_RequestMode changed into BswM_CanSM_CurrentState	
CANSM265 CANSM349 CANSM256 CANSM264 CANSM350 CANSM261 CANSM271 CANSM351 CANSM272 CANSM275 CANSM352 CANSM276 CANSM280 CANSM281 CANSM353	First change internal state, then issue callbacks	
CANSM284	CANSM_UNINITED removed	
CANSM255 CANSM268	Removed reference to chapter 7.1	Reference to same chapter while in same chapter
CANSM025	Version check corrected	[Csm] Version Check requirement needs correction
CANSM125	CANSM_MODULE_ID specified as uint16	
CANSM257 CANSM258 CANSM259 CANSM270	CanIf_SetTransceiverMode replaced with CanIf_SetTrcvMode	[Csm] Mismatch in the API name of the CanIf module
CANSM257 CANSM258	Typos	
CANSM167	Main function shall only implement the parts of the bus-off recovery state machine, which depend on time	Changed in scope of the document improvement process by TO
CANSM259 CANSM260 CANSM264 CANSM269 CANSM270 CANSM271	CanSMNetworkHandle replaced with CanSMComMNetworkHandleRef where necessary	[Csm] Wrong container name used to describe the requirements

CANSM273 CANSM275 CANSM279 CANSM280 CANSM281		
CANSM257 CANSM259 CANSM270 CANSM258 CANSM260 CANSM269 CANSM273 CANSM279 CANSM289 CANSM337 CANSM340 CANSM323 CANSM328 CANSM329	Solved inconsistency between API parameters specified in BSW UML and referenced parameters in the CANSM requirements	[CanSm] Mismatch in the API argument name of the CanIf module
CANSM062	Synchronous API	CANSM: CanSM_RequestComMode sync vs. async
CANSM336_Conf	Parameter name changed into: CanSMModeRequestRepetitionTime /CANSM_MODEREQ_REPEAT_TIME Description revised	[Can][CanIf][CanSm][CanTrcv] Full COM Request have to be asynchronous but is specified synchronous
CANSM335_Conf	Parameter name changed into: CanSMModeRequestRepetitionMax / CANSM_MODEREQ_MAX Description revised	
CANSM334_Conf	Parameter name changed into: CANSM_E_MODE_REQUEST_TIMEOUT Description revised	
CANSM257	Relation to requirement CANSM379	
CANSM258	Relation to requirement CANSM382	
CANSM265	Relation to requirement CANSM383	
CANSM349	Formulation changed to consider each network separately	
CANSM256	Formulation changed to consider each network separately	
CANSM260	Relation to requirement CANSM390	
CANSM264	Relation to requirement CANSM260	
CANSM350	Relation to requirement CANSM264	
CANSM261	Relation to requirement CANSM350	
CANSM269	Relation to requirement CANSM393	
CANSM271	Relation to requirement CANSM269	
CANSM351	Relation to requirement CANSM271	
CANSM337	Relation to requirement CANSM289	
CANSM339	Relation to requirement CANSM337	
CANSM338	Relation to requirement CANSM339	
CANSM354	Relation to requirement CANSM338	
CANSM340	Removed term of network mode request, because obsolete (requests are blocked during recovery)	
CANSM341	Relation to requirement CANSM342	
CANSM342	Relation to requirement CANSM340	
CANSM355	Relation to requirement CANSM341	

CANSM295	Relation to requirement CANSM294	
CANSM297	1.) Removed term of network mode request, because obsolete (requests are blocked during recovery) 2.) Added relation to CANSM355	
CANSM299	Relation to requirement CANSM298	
CANSM288	Bus-off counter to 0 from no to full communication instead of after PowerOn	
CANSM279	Also go to silent, if no communication is requested	
CANSM280	Relation to requirement CANSM279	
CANSM281	Relation to requirement CANSM280	
CANSM353	Relation to requirement CANSM281	
CANSM256 CANSM350 CANSM272 CANSM276 CANSM353 CANSM354 CANSM355	“Channel_Id” changed into “Network” “CanSM_Requested_Mode” changed into “CurrentState”	
CANSM295 CANSM299	paramters” corrected into “parameters”	
CANSM069	New development errors CANSM_E_WAIT_MODE_INDICATION, CANSM_E_INVALID_COMM_REQUEST	
CANSM069	New development error: CANSM_E_BUSOFF_RECOVERY_ACTIVE,	[CanSm] V1.1.36; C7.2; State transition description does not care about error conditions

11.4 Added SWS Items

SWS Item	***Rationale***
CANSM242	Requirement for file structure
CANSM243	Requirement for imported type
CANSM244	Standard requirement for `CanSM_GetVersionInfo`
CANSM249	Missing ID for standard requirement
CANSM310	
CANSM309	
CANSM306	
CANSM307	
CANSM308	
CANSM374	

CANSM312	
CANSM315	
CANSM337	Avoid mix of use cases for peripheral requests and mode transitions between communication mode requests from ComM and bus-off handling
CANSM338	Avoid mix of use cases for peripheral requests and mode transitions between communication mode requests from ComM and bus-off handling
CANSM339	Avoid mix of use cases for peripheral requests and mode transitions between communication mode requests from ComM and bus-off handling
CANSM340	Avoid mix of use cases for peripheral requests and mode transitions between communication mode requests from ComM and bus-off handling
CANSM341	Avoid mix of use cases for peripheral requests and mode transitions between communication mode requests from ComM and bus-off handling
CANSM342	Avoid mix of use cases for peripheral requests and mode transitions between communication mode requests from ComM and bus-off handling
CANSM343	New recovery for can controller timeouts events
CANSM344	New recovery for can controller timeouts events
CANSM345	New recovery for can controller timeouts events
CANSM346	New recovery for can controller timeouts events
CANSM317-CANSM333	New recovery for can controller timeouts events
CANSM347-CANSM359	
CANSM360	NULL pointer exception for the function CanSM_GetCurrentComMode
CANSM361	Added to consider latest SWS template and to solve
CANSM362	Added to consider latest SWS template and to solve
CANSM363	
CANSM364	
CANSM365	
CANSM366	
CANSM367	
CANSM368	
CANSM369	
CANSM370	
CANSM371	
CANSM372	
CANSM375 CANSM376 CANSM377	Solution of [CanSm] V1.1.36; C7.2; State transition description does not care about error conditions
CANSM378 CANSM381 CANSM386 CANSM387 CANSM388 CANSM390 CANSM385 CANSM391 CANSM392 CANSM393 CANSM394 CANSM395 CANSM396 CANSM397 CANSM398 CANSM399 CANSM400 CANSM401 CANSM402 CANSM403	Added during changes to solve [Can][CanIf][CanSm][CanTrcv] Full COM Request have to be asynchronous but is specified synchronous
CAN001_PI	Rework of Published Information

12 Changes between AUTOSAR R4.0 rev001 and rev002

12.1 Deleted SWS Items

SWS Item	***Rationale***
CANSM285	[CanSm] Internal network mode cannot be initialized to CANSM_UNINITED
CANSM022	[CanSm] export of main function

12.2 Replaced SWS Items

SWS Item	***replaced by SWS Item***	***Rationale***

12.3 Changed SWS Items

SWS Item	***Change***	***Rationale***
CANSM349 CANSM256 CANSM350 CANSM261 CANSM351 CANSM272 CANSM352 CANSM276 CANSM281 CANSM353 CANSM354 CANSM338 CANSM341 CANSM355	CanSMNetworkHandle replaced with CanSMComMNetworkHandleRef	”Wrong container name used to describe the requirements”
CANSM070_Conf CANSM334_Conf	Multiplicity changed from 1 to 0…1	[CanSm] Multiplicity of CanSmDemEventParameterRefs
CANSM314_Conf CANSM123_Conf CANSM126_Conf CANSM127_Conf CANSM338_Conf	Following parameters and containers renamed: - CanStateManagerConfiguration to CanSMConfiguration - CanStateManagerController to CanSMController - CanStateManagerGeneral to CanSMGeneral - CanStateManagerNetwork to CanSMManagerNetwork - CanSmDemEventParameterRefs to CanSMDemEventParameterRefs	[CanSm] Ecuc Parameter naming in CanSm
CANSM025	Requirement changed according to changed BSW004	[mult] SRS_General: BSW004
CANSM292 CANSM294 CANSM298	Evaluation of the new configuration parameter CANSM_BOR_TX_CONFIRMATION_POLLING (CANSM339_Conf)	[CanSm][CanIf] Bus-Off recovery optimization
CANSM257 CANSM392	CANTRCV_STANDBY replaced with CANIF_TRCV_MODE_STANDBY	[CanSm] Mismatch in the enumeration values of the CanIf module: Update of generated

		artifacts
CANSM259 CANSM391 CANSM378	CANTRCV_NORMAL replaced with CANIF_TRCV_MODE_NORMAL	[CanSm] Mismatch in the enumeration values of the CanIf module: Update of generated artifacts
CANSM337_Conf	Changed description field	[diverse] Clarify harmonized channel ID in COM-Stack
CANSM399 CANSM243	Type of function parameter TransceiverMode changed to CanTrcv_TrcvModeType	[CanTrcv][LinTrcv][LinIf][LinSM] APIs to be removed from State Diagram

12.4 Added SWS Items

SWS Item	***Rationale***
CANSM339_Conf	[CanSm][CanIf] Bus-Off recovery optimization
CANSM404	[CanSm][CanIf] Bus-Off recovery optimization
CANSM405	[CanSm][CanIf] Bus-Off recovery optimization
CANSM406	[CanSm][CanIf] Bus-Off recovery optimization
CANSM407	[CanSm] PDU channel modes of CanIf not correctly served
CANSM408	[CanSm] PDU channel modes of CanIf not correctly served
CANSM409	[CanSm] PDU channel modes of CanIf not correctly served

13 Changes between AUTOSAR R4.0 rev002 and rev003

13.1 Deleted SWS Items

SWS Item	***Rationale***
CANSM349	Solution for implementation [CanSm] Contradiction between CanSM and ComM for call of ComM_BusSM_ModeIndication()”
CANSM337_Conf	Implementation Task [CanSm][EthSM][FrSM][LinSM][BswM] Local network index of Bus SM modules
CANSM334_Conf	Completion of Production error concept in Com Stack)
CANSM132_Conf	Solution of [CanSm] Bus off recovery time independent of error detection time)

13.2 Replaced SWS Items

SWS Item	***replaced by SWS Item***	***Rationale***
CANSM255	CANSM424 –	1.) Implementation [CanSm] Instruction order for
CANSM268	CANSM427	transition to no communication
CANSM378		
CANSM257	CANSM429 –	2.) Implementation
CANSM381	CANSM480	[Dcm][BswM][CanSm][CanIf][Can] change of
CANSM407		baudrate within UDS service linkcontrol
CANSM258	CANSM483 –	
CANSM265	CANSM500	3.) Implementation [CanSm] Instruction order of
CANSM256		Entering NoCom
CANSM259	CANSM507 –	
CANSM390	CANSM529	4.) Implementation
CANSM260		[CanTrcv][CanIf][CanSm][CanNm] Handling if PN
CANSM409	CANSM531 –	functionality is disabled in the Trcv
CANSM264	CANSM543	
CANSM350		5.) Implementation [CanSM][CanNm] Partial
CANSM261	CANSM550	Networking – Error handling for missing ACK
CANSM394		(WUF retransmission)
CANSM391	CANSM554	
CANSM392		
CANSM393		
CANSM408		
CANSM269		
CANSM271		
CANSM351		
CANSM272		
CANSM273		
CANSM275		
CANSM352		
CANSM276		
CANSM279		
CANSM280		
CANSM281		
CANSM353		
CANSM286		
CANSM302		
CANSM287		
CANSM288		
CANSM289		

CANSM337 CANSM339 CANSM354 CANSM338 CANSM292 CANSM404 CANSM340 CANSM342 CANSM341 CANSM355 CANSM294 CANSM405 CANSM295 CANSM297 CANSM298 CANSM406 CANSM299 CANSM301 CANSM386 CANSM387 CANSM388		

13.3 Changed SWS Items

SWS Item	*Change*	*Rationale*
CANSM256	Dependency to removed CANSM349 replaced with dependency to existing CANSM265	Solution for implementation [CanSm] Contradiction between CanSM and ComM for call of ComM_BusSM_ModeIndication()”
CANSM130_Conf	Dependency to CANSM_BOR_TX_CONFIRMATION_POLLING added for CanSMBorTimeTxEnsured	Implementation [CanSm][CanIf] Bus-Off recovery optimization)
CANSM266	Clarification: The CanSM module shall store the latest notified current network mode to the ComM for each configured CAN network internally (ref. to CANSM126_Conf).	1.) Implementation [CanSm] Instruction order for transition to no communication 2.) Implementation [CanSm] Instruction order of Entering NoCom
CANSM186	Reference to CANSM266 for clarification	1.) Implementation [CanSm] Instruction order for transition to no communication 2.) Implementation [CanSm] Instruction order of Entering NoCom
CANSM282	Reference to CANSM430 for clarification	1.) Implementation [CanSm] Instruction order for transition to no communication 2.) Implementation [CanSm] Instruction order of Entering

		NoCom
CANSM182	Reference to CANSM427, CANSM429, CANSM499, CANSM524 and CANSM543 for clarification	1.) Implementation [CanSm] Instruction order for transition to no communication 2.) Implementation [CanSm] Instruction order of Entering NoCom
CANSM235	Reference to CANSM500 for clarification	1.) Implementation [CanSm] Instruction order for transition to no communication 2.) Implementation [CanSm] Instruction order of Entering NoCom
CANSM167	Main function not only for bus-off recovery, but for all effects of the CanSM state machine	Asynchronous interaction behavior of the CanIf API and the CanSM state machine: e. g. for Implementation [CanSm] Instruction order for transition to no communication
CANSM385	Part removed with specifies transition to "no communication"	e. .g: Implementation [CanSm] Instruction order for transition to no communication
CANSM137_Conf	CanSMTransceiverId multiplicity changed to 0…1 CanSMTransceiverId (CANSM137_Conf) references CanIfTrcvCfg instead of CanTrcvChannel now	1.) Solution of CanSm: Controller of CanIf -> ControllerId) 2.) Solution [CanSM]: Multiplicity of configuration parameter CanSMTransceiverId)
CANSM141_Conf	CanSMControllerId (CANSM141_Conf) references CanIfCtrlCfg instead of CanController now	Solution CanSm: Controller of CanIf -> ControllerId)
CANSM385 CANSM072 CANSM074	CANSM_E_MODE_REQUEST_TIMEOUT changed into DET error	Solution Completion of Production error concept in Com Stack)
CANSM335_Conf	Description of CANSM335_Conf changed: … „reports a development error to the DET" …	Solution for reopened [CanSm] Bus off recovery time independent of error detection time

13.4 Added SWS Items

SWS Item	***Rationale***
CANSM501 CANSM502 CANSM503 CANSM504 CANSM505	Implementation [Dcm][BswM][CanSm][CanIf][Can] change of baudrate within UDS service linkcontrol:

CANSM506 CANSM530 CANSM544 CANSM545 CANSM547 CANSM551 CANSM552 CANSM553	
CANSM413 CANSM414 CANSM415 CANSM416 CANSM417 CANSM418	Implementation [CanSm] Instruction order of Entering NoCom
CANSM419 CANSM420 CANSM421 CANSM422 CANSM546 CANSM548	Implementation [CanTrcv][CanIf][CanSm][CanNm] Handling if PN functionality is disabled in the Trcv
CANSM410 CANSM411 CANSM412 CANSM549	Implementation [CanSM][CanNm] Partial Networking - Error handling for missing ACK (WUF retransmission)
CANSM555	Solution of [CanSM] Clarification required for CanSM_RequestComMode
CANSM556, CANSM557, CANSM558	Solution of [CanSM]: Multiplicity of configuration parameter CanSMTransceiverId)
CANSM560	Solution [CanSm] Modification required on handling CanTrcv)
CANSM561-CANSM574	[Dcm][BswM][CanSm][CanIf][Can] change of baudrate within UDS service linkcontrol)

14 Not applicable requirements

[CANSM999]「 These requirements are not applicable to this specification. 」(BSW170, BSW00375, BSW00395, BSW00416, BSW00437, BSW168, BSW00423, BSW00426, BSW00427, BSW00428, BSW00429, BSW00431, BSW00432, BSW00433, BSW00434, BSW00336, BSW00417, BSW161, BSW162, BSW005, BSW00326, BSW00347, BSW00314, BSW00435, BSW00353, BSW00361, BSW00377, BSW00308, BSW00309, BSW00360, BSW00341, BSW00439, BSW00440)