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

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

Document Change History

Date	**Release**	**Changed by**	**Description**
2023-11-23	R23-11	AUTOSAR Release Management	• Support for selective WakeUp via CAN-Controller • Clarification of "Available via: Configurable" • Added SWS IDs for "mandatory interfaces" & "optional interfaces • Editorial changes
2022-11-24	R22-11	AUTOSAR Release Management	• ComTxModeTimePeriodFactor replaced with ComTxModeTimePeriod
2021-11-25	R21-11	AUTOSAR Release Management	• Note added for CanSM_TransceiverModeIndication() • Communication mode notification to ComM after initialization clarified • Clean-up in CANSM_BSM regarding REPEAT_MAX / No Never-Give-Up Strategy
2020-11-30	R20-11	AUTOSAR Release Management	• Pretended Networking removed • Editorial changes

▽

△

2019-11-28	R19-11	AUTOSAR Release Management	• Fixed Change_Baudrate-Statemachine for NoCom • Added GetPduMode-Interface to list • Inconsistent behavior due to REPEAT_MAX / No Never-Give-Up Strategy fixed • Changed Document Status from Final to published
2018-30-31	4.4.0	AUTOSAR Release Management	• Reclassification of some errors • Editorial changes
2017-12-08	4.3.1	AUTOSAR Release Management	• Moved CANSM_E_MODE_REQUEST_TIMEOUT to Runtime Error
2016-11-30	4.3.0	AUTOSAR Release Management	• Provide DeInit-API • ECU passive mode clarified and fixed • Editorial changes
2015-07-31	4.2.2	AUTOSAR Release Management	• Development Error Tracer replaced with Default Error Tracer • Bus-off recovery time dependencies specified more precisely • Optional interface to check and to change baudrate removed
2014-10-31	4.2.1	AUTOSAR Release Management	• API for ECU passive mode activation • Baudrate change without reinitialisation, if possible • Interface handling to CanIf module improved • Interface handling to ComM module improved

▽

△

2014-03-31	4.1.3	AUTOSAR Release Management	• Introduction of random delays • Re-Request of ComMode • WakeupValidation to avoid race conditions • Adapt Bus Off Recovery and NM state synchronization
2013-10-31	4.1.2	AUTOSAR Release Management	• Dependency to DCM module removed • Mileading timing row removed in CanSM_MainFunction • Editorial changes • Removed chapter(s) on change documentation
2013-03-15	4.1.1	AUTOSAR Administration	• Support Pretended Networking mode handling • Changed concept to setup baudrate • Initialization Sequence between ComM and CanSM • Do not send WUF as First Message on the Bus after BusOff • CanSm_TxTimeoutExeption in case of BusOff
2011-12-22	4.0.3	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

▽

△

2009-12-18	4.0.1	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
2010-02-02	3.1.4	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) • Legal disclaimer revised
2008-08-13	3.1.1	AUTOSAR Administration	• Legal disclaimer revised
2007-12-21	3.0.1	AUTOSAR Administration	• Initial Release

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.

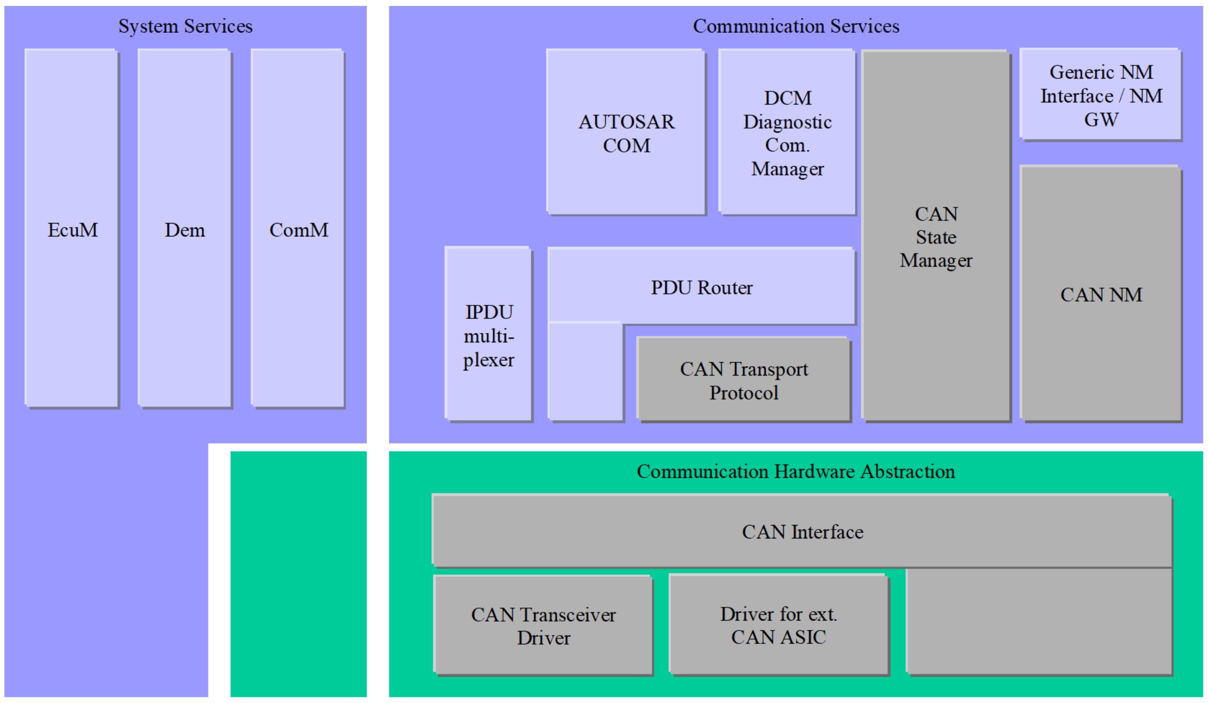

Figure 1.1: Layered Software Architecture from CanSM point of view

2 Acronyms and Abbreviations

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

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	Default 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

Table 2.1: Acronyms and abbreviations used in the scope of this Document

3 Related documentation

3.1 Input documents & related standards and norms

[1] Glossary
AUTOSAR_FO_TR_Glossary

[2] General Specification of Basic Software Modules
AUTOSAR_CP_SWS_BSWGeneral

[3] Specification of ECU State Manager
AUTOSAR_CP_SWS_ECUStateManager

[4] Specification of RTE Software
AUTOSAR_CP_SWS_RTE

[5] Specification of Communication Manager
AUTOSAR_CP_SWS_COMManager

[6] Specification of CAN Interface
AUTOSAR_CP_SWS_CANInterface

[7] Specification of Diagnostic Event Manager
AUTOSAR_CP_SWS_DiagnosticEventManager

[8] Specification of Basic Software Mode Manager
AUTOSAR_CP_SWS_BSWModeManager

[9] Specification of CAN Network Management
AUTOSAR_CP_SWS_CANNetworkManagement

[10] Specification of Default Error Tracer
AUTOSAR_CP_SWS_DefaultErrorTracer

[11] Specification of CAN Transceiver Driver
AUTOSAR_CP_SWS_CANTransceiverDriver

[12] General Requirements on Basic Software Modules
AUTOSAR_CP_SRS_BSWGeneral

3.2 Related specification

AUTOSAR provides a General Specification on Basic Software modules [2, SWS BSW General], which is also valid for CAN State Manager.

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

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 or 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.

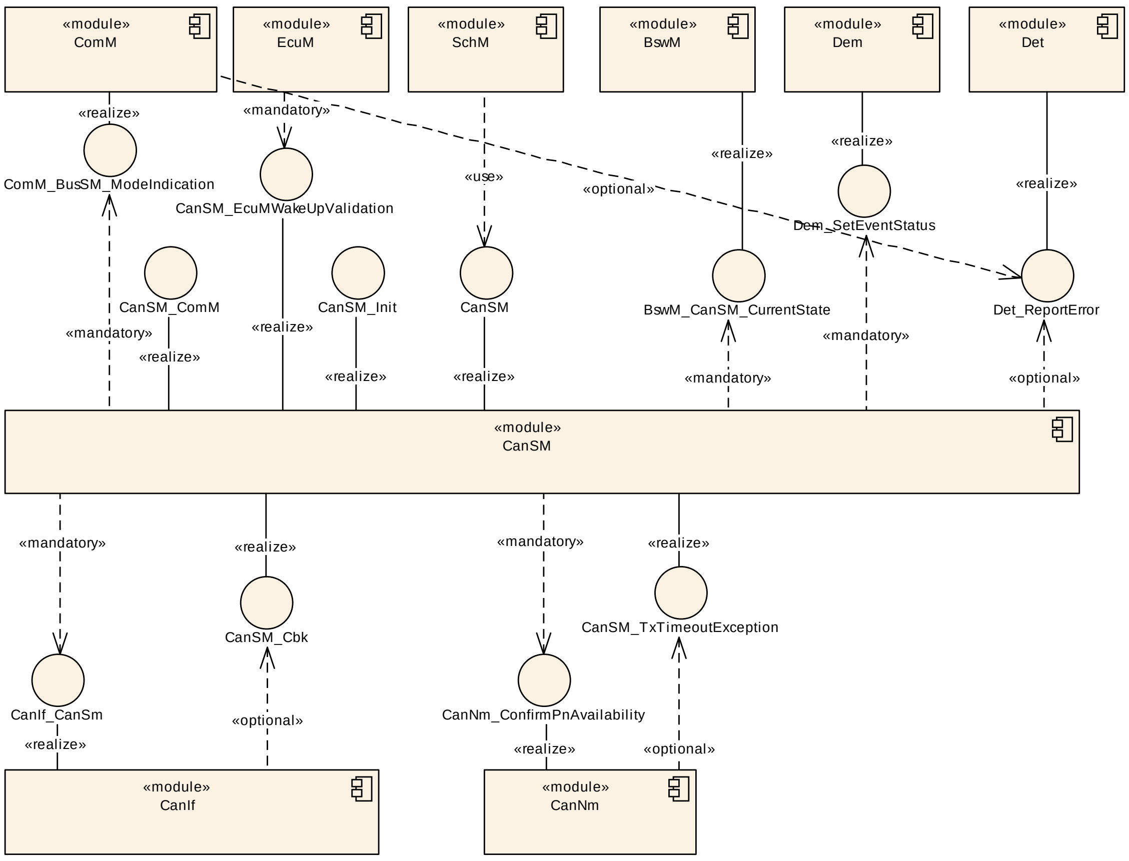

Figure 5.1: Module dependencies of the CanSM module

5.1 ECU State Manager (EcuM)

The EcuM module initializes the CanSM module and interacts with the CanSM module for the CAN wakeup validation (refer to [3, Specification of ECU State Manager] for a detailed specification of this module).

5.2 BSW Scheduler (SchM, part of RTE)

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 [4, Specification of RTE Software] 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 [5, Specification of Communication Manager] 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 [6, Specification of CAN Interface] 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 [7, Specification of Diagnostic Event Manager] 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 [8, Specification of Basic Software Mode Manager] 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 (refer to [9, Specification of CAN Network Management] for a detailed specification of this module).

5.8 Default Error Tracer (DET)

The CanSM module reports development and runtime errors to the DET module.
Development Errors are only reported if development error handling is switched on by

configuration (refer to [10, Specification of Default Error Tracer] for a detailed specification of this module).

5.9 File structure

5.9.1 Code file structure

For details refer to the chapter 5.1.6 "Code file structure" in [2, SWS BSW General].

5.9.2 Header file structure

[SWS_CanSM_00008] ⌈The header file `CanSM.h` shall export CanSM module specific types and the APIs `CanSM_GetVersionInfo` and `CanSM_Init`.⌋*(SRS_BSW_-00447)*

5.9.3 Version check

For details refer to the chapter 5.1.8 "Version Check" in [2, SWS BSW General].

6 Requirements Tracing

The following tables reference the requirements specified in <CITATIONS_OF_CONTRIBUTED_DOCUMENTS> and links to the fulfillment of these. Please note that if column "Satisfied by" is empty for a specific requirement this means that this requirement is not fulfilled by this document.

Requirement	Description	Satisfied by
[SRS_BSW_00003]	All software modules shall provide version and identification information	[SWS_CanSM_00024] [SWS_CanSM_00374]
[SRS_BSW_00004]	All Basic SW Modules shall perform a pre-processor check of the versions of all imported include files	[SWS_CanSM_00652]
[SRS_BSW_00005]	Modules of the μC Abstraction Layer (MCAL) may not have hard coded horizontal interfaces	[SWS_CanSM_00652]
[SRS_BSW_00006]	The source code of software modules above the μC Abstraction Layer (MCAL) shall not be processor and compiler dependent.	[SWS_CanSM_00652]
[SRS_BSW_00007]	All Basic SW Modules written in C language shall conform to the MISRA C 2012 Standard.	[SWS_CanSM_00652]
[SRS_BSW_00009]	All Basic SW Modules shall be documented according to a common standard.	[SWS_CanSM_00652]
[SRS_BSW_00010]	The memory consumption of all Basic SW Modules shall be documented for a defined configuration for all supported platforms.	[SWS_CanSM_00652]
[SRS_BSW_00101]	The Basic Software Module shall be able to initialize variables and hardware in a separate initialization function	[SWS_CanSM_00023]
[SRS_BSW_00159]	All modules of the AUTOSAR Basic Software shall support a tool based configuration	[SWS_CanSM_00652]
[SRS_BSW_00160]	Configuration files of AUTOSAR Basic SW module shall be readable for human beings	[SWS_CanSM_00652]
[SRS_BSW_00161]	The AUTOSAR Basic Software shall provide a microcontroller abstraction layer which provides a standardized interface to higher software layers	[SWS_CanSM_00652]
[SRS_BSW_00162]	The AUTOSAR Basic Software shall provide a hardware abstraction layer	[SWS_CanSM_00652]
[SRS_BSW_00164]	The Implementation of interrupt service routines shall be done by the Operating System, complex drivers or modules	[SWS_CanSM_00652]
[SRS_BSW_00167]	All AUTOSAR Basic Software Modules shall provide configuration rules and constraints to enable plausibility checks	[SWS_CanSM_00652]
[SRS_BSW_00168]	SW components shall be tested by a function defined in a common API in the Basis-SW	[SWS_CanSM_00652]

▽

△

Requirement	Description	Satisfied by
[SRS_BSW_00170]	The AUTOSAR SW Components shall provide information about their dependency from faults, signal qualities, driver demands	[SWS_CanSM_00652]
[SRS_BSW_00172]	The scheduling strategy that is built inside the Basic Software Modules shall be compatible with the strategy used in the system	[SWS_CanSM_00652]
[SRS_BSW_00300]	All AUTOSAR Basic Software Modules shall be identified by an unambiguous name	[SWS_CanSM_00652]
[SRS_BSW_00301]	All AUTOSAR Basic Software Modules shall only import the necessary information	[SWS_CanSM_00652]
[SRS_BSW_00302]	All AUTOSAR Basic Software Modules shall only export information needed by other modules	[SWS_CanSM_00652]
[SRS_BSW_00305]	Data types naming convention	[SWS_CanSM_00652]
[SRS_BSW_00306]	AUTOSAR Basic Software Modules shall be compiler and platform independent	[SWS_CanSM_00652]
[SRS_BSW_00307]	Global variables naming convention	[SWS_CanSM_00652]
[SRS_BSW_00308]	AUTOSAR Basic Software Modules shall not define global data in their header files, but in the C file	[SWS_CanSM_00652]
[SRS_BSW_00309]	All AUTOSAR Basic Software Modules shall indicate all global data with read-only purposes by explicitly assigning the const keyword	[SWS_CanSM_00652]
[SRS_BSW_00310]	API naming convention	[SWS_CanSM_00652]
[SRS_BSW_00312]	Shared code shall be reentrant	[SWS_CanSM_00652]
[SRS_BSW_00314]	All internal driver modules shall separate the interrupt frame definition from the service routine	[SWS_CanSM_00652]
[SRS_BSW_00318]	Each AUTOSAR Basic Software Module file shall provide version numbers in the header file	[SWS_CanSM_00652]
[SRS_BSW_00321]	The version numbers of AUTOSAR Basic Software Modules shall be enumerated according specific rules	[SWS_CanSM_00652]
[SRS_BSW_00323]	All AUTOSAR Basic Software Modules shall check passed API parameters for validity	[SWS_CanSM_00652]
[SRS_BSW_00325]	The runtime of interrupt service routines and functions that are running in interrupt context shall be kept short	[SWS_CanSM_00652]
[SRS_BSW_00327]	Error values naming convention	[SWS_CanSM_00652]
[SRS_BSW_00328]	All AUTOSAR Basic Software Modules shall avoid the duplication of code	[SWS_CanSM_00652]
[SRS_BSW_00330]	It shall be allowed to use macros instead of functions where source code is used and runtime is critical	[SWS_CanSM_00652]

▽

△

Requirement	Description	Satisfied by
[SRS_BSW_00331]	All Basic Software Modules shall strictly separate error and status information	[SWS_CanSM_00652]
[SRS_BSW_00333]	For each callback function it shall be specified if it is called from interrupt context or not	[SWS_CanSM_00064] [SWS_CanSM_00189] [SWS_CanSM_00190] [SWS_CanSM_00235]
[SRS_BSW_00334]	All Basic Software Modules shall provide an XML file that contains the meta data	[SWS_CanSM_00652]
[SRS_BSW_00335]	Status values naming convention	[SWS_CanSM_00652]
[SRS_BSW_00336]	Basic SW module shall be able to shutdown	[SWS_CanSM_00652] [SWS_CanSM_91001]
[SRS_BSW_00337]	Classification of development errors	[SWS_CanSM_00654]
[SRS_BSW_00339]	Reporting of production relevant error status	[SWS_CanSM_00652]
[SRS_BSW_00341]	Module documentation shall contains all needed informations	[SWS_CanSM_00652]
[SRS_BSW_00342]	It shall be possible to create an AUTOSAR ECU out of modules provided as source code and modules provided as object code, even mixed	[SWS_CanSM_00652]
[SRS_BSW_00343]	The unit of time for specification and configuration of Basic SW modules shall be preferably in physical time unit	[SWS_CanSM_00652]
[SRS_BSW_00346]	All AUTOSAR Basic Software Modules shall provide at least a basic set of module files	[SWS_CanSM_00652]
[SRS_BSW_00347]	A Naming seperation of different instances of BSW drivers shall be in place	[SWS_CanSM_00652]
[SRS_BSW_00348]	All AUTOSAR standard types and constants shall be placed and organized in a standard type header file	[SWS_CanSM_00652]
[SRS_BSW_00350]	All AUTOSAR Basic Software Modules shall allow the enabling/ disabling of detection and reporting of development errors.	[SWS_CanSM_00652]
[SRS_BSW_00353]	All integer type definitions of target and compiler specific scope shall be placed and organized in a single type header	[SWS_CanSM_00652]
[SRS_BSW_00357]	For success/failure of an API call a standard return type shall be defined	[SWS_CanSM_00652]
[SRS_BSW_00358]	The return type of init() functions implemented by AUTOSAR Basic Software Modules shall be void	[SWS_CanSM_00023]
[SRS_BSW_00359]	All AUTOSAR Basic Software Modules callback functions shall avoid return types other than void if possible	[SWS_CanSM_00064] [SWS_CanSM_00189] [SWS_CanSM_00190] [SWS_CanSM_00235]
[SRS_BSW_00360]	AUTOSAR Basic Software Modules callback functions are allowed to have parameters	[SWS_CanSM_00652] [SWS_CanSM_00652]

▽

△

Requirement	Description	Satisfied by
[SRS_BSW_00369]	All AUTOSAR Basic Software Modules shall not return specific development error codes via the API	[SWS_CanSM_00652] [SWS_CanSM_00660]
[SRS_BSW_00373]	The main processing function of each AUTOSAR Basic Software Module shall be named according the defined convention	[SWS_CanSM_00652]
[SRS_BSW_00374]	All Basic Software Modules shall provide a readable module vendor identification	[SWS_CanSM_00652]
[SRS_BSW_00375]	Basic Software Modules shall report wake-up reasons	[SWS_CanSM_00652]
[SRS_BSW_00377]	A Basic Software Module can return a module specific types	[SWS_CanSM_00652]
[SRS_BSW_00378]	AUTOSAR shall provide a boolean type	[SWS_CanSM_00652]
[SRS_BSW_00379]	All software modules shall provide a module identifier in the header file and in the module XML description file.	[SWS_CanSM_00652]
[SRS_BSW_00380]	Configuration parameters being stored in memory shall be placed into separate c-files	[SWS_CanSM_00652]
[SRS_BSW_00383]	The Basic Software Module specifications shall specify which other configuration files from other modules they use at least in the description	[SWS_CanSM_00652]
[SRS_BSW_00384]	The Basic Software Module specifications shall specify at least in the description which other modules they require	[SWS_CanSM_00652]
[SRS_BSW_00385]	List possible error notifications	[SWS_CanSM_00652]
[SRS_BSW_00386]	The BSW shall specify the configuration and conditions for detecting an error	[SWS_CanSM_00652]
[SRS_BSW_00388]	Containers shall be used to group configuration parameters that are defined for the same object	[SWS_CanSM_00652]
[SRS_BSW_00389]	Containers shall have names	[SWS_CanSM_00652]
[SRS_BSW_00390]	Parameter content shall be unique within the module	[SWS_CanSM_00652]
[SRS_BSW_00392]	Parameters shall have a type	[SWS_CanSM_00652]
[SRS_BSW_00393]	Parameters shall have a range	[SWS_CanSM_00652]
[SRS_BSW_00394]	The Basic Software Module specifications shall specify the scope of the configuration parameters	[SWS_CanSM_00652]
[SRS_BSW_00395]	The Basic Software Module specifications shall list all configuration parameter dependencies	[SWS_CanSM_00652]

▽

△

Requirement	Description	Satisfied by
[SRS_BSW_00396]	The Basic Software Module specifications shall specify the supported configuration classes for changing values and multiplicities for each parameter/container	[SWS_CanSM_00652]
[SRS_BSW_00397]	The configuration parameters in pre-compile time are fixed before compilation starts	[SWS_CanSM_00652]
[SRS_BSW_00398]	The link-time configuration is achieved on object code basis in the stage after compiling and before linking	[SWS_CanSM_00652]
[SRS_BSW_00399]	Parameter-sets shall be located in a separate segment and shall be loaded after the code	[SWS_CanSM_00652]
[SRS_BSW_00400]	Parameter shall be selected from multiple sets of parameters after code has been loaded and started	[SWS_CanSM_00023] [SWS_CanSM_00597] [SWS_CanSM_00652]
[SRS_BSW_00401]	Documentation of multiple instances of configuration parameters shall be available	[SWS_CanSM_00652]
[SRS_BSW_00402]	Each module shall provide version information	[SWS_CanSM_00652]
[SRS_BSW_00404]	BSW Modules shall support post-build configuration	[SWS_CanSM_00023]
[SRS_BSW_00405]	BSW Modules shall support multiple configuration sets	[SWS_CanSM_00023]
[SRS_BSW_00406]	A static status variable denoting if a BSW module is initialized shall be initialized with value 0 before any APIs of the BSW module is called	[SWS_CanSM_00023] [SWS_CanSM_00184]
[SRS_BSW_00407]	Each BSW module shall provide a function to read out the version information of a dedicated module implementation	[SWS_CanSM_00024] [SWS_CanSM_00374]
[SRS_BSW_00408]	All AUTOSAR Basic Software Modules configuration parameters shall be named according to a specific naming rule	[SWS_CanSM_00652]
[SRS_BSW_00409]	All production code error ID symbols are defined by the Dem module and shall be retrieved by the other BSW modules from Dem configuration	[SWS_CanSM_00652]
[SRS_BSW_00410]	Compiler switches shall have defined values	[SWS_CanSM_00652]
[SRS_BSW_00411]	All AUTOSAR Basic Software Modules shall apply a naming rule for enabling/disabling the existence of the API	[SWS_CanSM_00652]
[SRS_BSW_00413]	An index-based accessing of the instances of BSW modules shall be done	[SWS_CanSM_00652]
[SRS_BSW_00414]	Init functions shall have a pointer to a configuration structure as single parameter	[SWS_CanSM_00023]

▽

△

Requirement	Description	Satisfied by
[SRS_BSW_00415]	Interfaces which are provided exclusively for one module shall be separated into a dedicated header file	[SWS_CanSM_00652]
[SRS_BSW_00416]	The sequence of modules to be initialized shall be configurable	[SWS_CanSM_00652]
[SRS_BSW_00417]	Software which is not part of the SW-C shall report error events only after the Dem is fully operational.	[SWS_CanSM_00652]
[SRS_BSW_00419]	If a pre-compile time configuration parameter is implemented as `const` it should be placed into a separate c-file	[SWS_CanSM_00652]
[SRS_BSW_00422]	Pre-de-bouncing of error status information is done within the Dem	[SWS_CanSM_00498] [SWS_CanSM_00522] [SWS_CanSM_00605] [SWS_CanSM_00652]
[SRS_BSW_00423]	BSW modules with AUTOSAR interfaces shall be describable with the means of the SW-C Template	[SWS_CanSM_00652]
[SRS_BSW_00424]	BSW module main processing functions shall not be allowed to enter a wait state	[SWS_CanSM_00065] [SWS_CanSM_00167]
[SRS_BSW_00425]	The BSW module description template shall provide means to model the defined trigger conditions of schedulable objects	[SWS_CanSM_00065] [SWS_CanSM_00167]
[SRS_BSW_00426]	BSW Modules shall ensure data consistency of data which is shared between BSW modules	[SWS_CanSM_00652]
[SRS_BSW_00427]	ISR functions shall be defined and documented in the BSW module description template	[SWS_CanSM_00652]
[SRS_BSW_00428]	A BSW module shall state if its main processing function(s) has to be executed in a specific order or sequence	[SWS_CanSM_00652]
[SRS_BSW_00429]	Access to OS is restricted	[SWS_CanSM_00652]
[SRS_BSW_00432]	Modules should have separate main processing functions for read/receive and write/transmit data path	[SWS_CanSM_00652]
[SRS_BSW_00433]	Main processing functions are only allowed to be called from task bodies provided by the BSW Scheduler	[SWS_CanSM_00652]
[SRS_BSW_00437]	Memory mapping shall provide the possibility to define RAM segments which are not to be initialized during startup	[SWS_CanSM_00652]
[SRS_BSW_00438]	Configuration data shall be defined in a structure	[SWS_CanSM_00023] [SWS_CanSM_00597] [SWS_CanSM_00652]
[SRS_BSW_00439]	Enable BSW modules to handle interrupts	[SWS_CanSM_00652]
[SRS_BSW_00440]	The callback function invocation by the BSW module shall follow the signature provided by RTE to invoke servers via `Rte_Call` API	[SWS_CanSM_00652]
[SRS_BSW_00441]	Naming convention for type, macro and function	[SWS_CanSM_00652]

▽

△

Requirement	Description	Satisfied by
[SRS_BSW_00447]	Standardizing Include file structure of BSW Modules Implementing Autosar Service	[SWS_CanSM_00008]
[SRS_BSW_00448]	Module SWS shall not contain requirements from other modules	[SWS_CanSM_00652]
[SRS_BSW_00449]	BSW Service APIs used by Autosar Application Software shall return a Std_ReturnType	[SWS_CanSM_00652]
[SRS_BSW_00450]	A Main function of a un-initialized module shall return immediately	[SWS_CanSM_00652]
[SRS_BSW_00451]	Hardware registers shall be protected if concurrent access to these registers occur	[SWS_CanSM_00652]
[SRS_BSW_00452]	Classification of runtime errors	[SWS_CanSM_00652]
[SRS_BSW_00453]	BSW Modules shall be harmonized	[SWS_CanSM_00652]
[SRS_BSW_00454]	An alternative interface without a parameter of category DATA_ REFERENCE shall be available.	[SWS_CanSM_00652]
[SRS_BSW_00456]	A Header file shall be defined in order to harmonize BSW Modules	[SWS_CanSM_00652]
[SRS_BSW_00457]	Callback functions of Application software components shall be invoked by the Basis SW	[SWS_CanSM_00652]
[SRS_BSW_00458]	Classification of production errors	[SWS_CanSM_00652]
[SRS_BSW_00459]	It shall be possible to concurrently execute a service offered by a BSW module in different partitions	[SWS_CanSM_00652]
[SRS_BSW_00460]	Reentrancy Levels	[SWS_CanSM_00652]
[SRS_BSW_00461]	Modules called by generic modules shall satisfy all interfaces requested by the generic module	[SWS_CanSM_00652]
[SRS_BSW_00462]	All Standardized Autosar Interfaces shall have unique requirement Id / number	[SWS_CanSM_00652]
[SRS_BSW_00463]	Naming convention of callout prototypes	[SWS_CanSM_00652]
[SRS_BSW_00465]	It shall not be allowed to name any two files so that they only differ by the cases of their letters	[SWS_CanSM_00652]
[SRS_BSW_00466]	Classification of extended production errors	[SWS_CanSM_00652] [SWS_CanSM_00664]
[SRS_BSW_00467]	The init / deinit services shall only be called by BswM or EcuM	[SWS_CanSM_00652]
[SRS_BSW_00469]	Fault detection and healing of production errors and extended production errors	[SWS_CanSM_00652]
[SRS_BSW_00470]	Execution frequency of production error detection	[SWS_CanSM_00652]
[SRS_BSW_00471]	Do not cause dead-locks on detection of production errors - the ability to heal from previously detected production errors	[SWS_CanSM_00652]
[SRS_BSW_00472]	Avoid detection of two production errors with the same root cause.	[SWS_CanSM_00652]

▽

△

Requirement	Description	Satisfied by
[SRS_Can_01001]	The CAN Interface implementation and interface shall be independent from underlying CAN Controller and CAN Transceiver	[SWS_CanSM_00652]
[SRS_Can_01002]	The CAN Interface shall be responsible for the dispatching of the received PDUs	[SWS_CanSM_00652]
[SRS_Can_01003]	The appropriate higher communication stack shall be notified by the CAN Interface about an occurred reception	[SWS_CanSM_00652]
[SRS_Can_01004]	Software filtering shall be implemented by the CAN Interface	[SWS_CanSM_00652]
[SRS_Can_01005]	The CAN Interface shall perform a check for correct DLC of received PDUs	[SWS_CanSM_00652]
[SRS_Can_01006]	The CAN Interface shall provide a service to enable/disable L-PDU reception per CAN Controller	[SWS_CanSM_00652]
[SRS_Can_01007]	The CAN Interface shall dispatch the transmission request by an upper layer module to the desired CAN controller	[SWS_CanSM_00652]
[SRS_Can_01008]	The CAN Interface shall provide a transmission request service	[SWS_CanSM_00652]
[SRS_Can_01009]	The CAN Interface shall provide a transmission confirmation dispatcher	[SWS_CanSM_00652]
[SRS_Can_01011]	The CAN Interface shall provide a transmit buffer	[SWS_CanSM_00652]
[SRS_Can_01013]	The CAN Interface shall provide a Tx-L-PDU enable/disable service per CAN Controller	[SWS_CanSM_00652]
[SRS_Can_01014]	The CAN State Manager shall offer a network configuration independent interface for upper layers	[SWS_CanSM_00652]
[SRS_Can_01015]	The CAN Interface configuration shall be able to import information from CAN communication matrix.	[SWS_CanSM_00652]
[SRS_Can_01016]	The CAN Interface shall have an interface to the static configuration information of the CAN Driver	[SWS_CanSM_00652]
[SRS_Can_01018]	The CAN Interface shall have an interface to the static configuration information of the CAN Driver	[SWS_CanSM_00652]
[SRS_Can_01020]	The TX-Buffer shall be statically configurable	[SWS_CanSM_00652]
[SRS_Can_01021]	CAN The CAN Interface shall implement an interface for initialization	[SWS_CanSM_00652]
[SRS_Can_01022]	The CAN Interface shall support the selection of configuration sets	[SWS_CanSM_00652]
[SRS_Can_01023]	The CAN Interface shall be initialized in a defined way.	[SWS_CanSM_00652]
[SRS_Can_01027]	The CAN Interface shall provide a service to change the CAN Controller mode.	[SWS_CanSM_00652]

▽

△

Requirement	Description	Satisfied by
[SRS_Can_01028]	The CAN Interface shall provide a service to query the CAN controller state	[SWS_CanSM_00652]
[SRS_Can_01029]	The CAN Interface shall report bus-off state of a device to an upper layer	[SWS_CanSM_00652]
[SRS_Can_01032]	The CAN Interface shall report a wake-up notification to the ECU State Manager	[SWS_CanSM_00652]
[SRS_Can_01033]	The CAN Driver shall fulfill the general requirements for Basic Software Modules as specified in AUTOSAR_SRS_SPAL	[SWS_CanSM_00652]
[SRS_Can_01034]	The CAN Driver shall offer a Hardware independent interface.	[SWS_CanSM_00652]
[SRS_Can_01035]	The CAN Driver shall support multiple CAN controllers of the same CAN hardware unit	[SWS_CanSM_00652]
[SRS_Can_01036]	The Can Driver shall support Standard Identifier and Extended Identifier	[SWS_CanSM_00652]
[SRS_Can_01037]	The CAN driver shall allow the static configuration of the hardware reception filter	[SWS_CanSM_00652]
[SRS_Can_01038]	The bit timing of each CAN Controller shall be configurable	[SWS_CanSM_00652]
[SRS_Can_01039]	Hardware Object Handles shall be provided for the CAN Interface in the static configuration file.	[SWS_CanSM_00652]
[SRS_Can_01041]	The CAN Driver shall implement an interface for initialization	[SWS_CanSM_00652]
[SRS_Can_01042]	The CAN Driver shall support dynamic selection of configuration sets	[SWS_CanSM_00652]
[SRS_Can_01043]	The CAN Driver shall provide a service to enable/disable interrupts of the CAN Controller.	[SWS_CanSM_00652]
[SRS_Can_01045]	The CAN Driver shall offer a reception indication service.	[SWS_CanSM_00652]
[SRS_Can_01049]	The CAN Driver shall provide a dynamic transmission request service	[SWS_CanSM_00652]
[SRS_Can_01051]	The CAN Driver shall provide a transmission confirmation service	[SWS_CanSM_00652]
[SRS_Can_01053]	The CAN Driver shall provide a service to change the CAN controller mode.	[SWS_CanSM_00652]
[SRS_Can_01054]	The CAN Driver shall provide a notification for controller wake-up events	[SWS_CanSM_00652]
[SRS_Can_01055]	CAN Driver shall provide a notification for bus-off state	[SWS_CanSM_00652]
[SRS_Can_01058]	shall be configurable whether Multiplex Transmission is used	[SWS_CanSM_00652]
[SRS_Can_01059]	The CAN Driver shall guarantee data consistency of received L-PDUs	[SWS_CanSM_00652]

▽

△

Requirement	Description	Satisfied by
[SRS_Can_01060]	The CAN driver shall not recover from bus-off automatically	[SWS_CanSM_00652]
[SRS_Can_01061]	The CAN Interface shall provide dynamic TX Handles	[SWS_CanSM_00652]
[SRS_Can_01062]	Each event for each CAN Controller shall be configurable to be detected by polling or by an interrupt	[SWS_CanSM_00652]
[SRS_Can_01065]	The AUTOSAR CAN Transport Layer shall be based on ISO 15765-2 and 15765-4 specifications	[SWS_CanSM_00652]
[SRS_Can_01066]	The AUTOSAR CAN Transport Layer shall be statically configurable to support either single or multiple connections in an optimizing way	[SWS_CanSM_00652]
[SRS_Can_01068]	The CAN Transport Layer shall identify each N-SDU with a unique identifier.	[SWS_CanSM_00652]
[SRS_Can_01069]	CAN address information and N-SDU identifier mapping	[SWS_CanSM_00652]
[SRS_Can_01071]	The CAN Transport Layer shall identify each N-PDU (also called L-SDU) with a unique identifier	[SWS_CanSM_00652]
[SRS_Can_01073]	The CAN Transport Layer shall be statically configured to pad unused bytes of PDU	[SWS_CanSM_00652]
[SRS_Can_01074]	The Transport connection properties shall be statically configured	[SWS_CanSM_00652]
[SRS_Can_01075]	The CAN Transport Layer shall implement an interface for initialization	[SWS_CanSM_00652]
[SRS_Can_01076]	The CAN Transport Layer services shall not be operational before initializing the module	[SWS_CanSM_00652]
[SRS_Can_01078]	The AUTOSAR CAN Transport Layer shall support the ISO 15765-2 addressing formats	[SWS_CanSM_00652]
[SRS_Can_01079]	The CAN Transport Layer shall be compliant with the CAN Interface module notifications	[SWS_CanSM_00652]
[SRS_Can_01081]	The value of CAN Transport protocol timeouts shall be statically configurable for each connection	[SWS_CanSM_00652]
[SRS_Can_01082]	Error handling	[SWS_CanSM_00652]
[SRS_Can_01086]	Data padding value of unused bytes	[SWS_CanSM_00652]
[SRS_Can_01090]	The bus transceiver driver package shall offer configuration parameters that are needed to configure the driver for a given bus and the supported notifications	[SWS_CanSM_00652]
[SRS_Can_01091]	The CAN bus transceiver driver shall support the configuration for more than one bus	[SWS_CanSM_00652]

▽

△

Requirement	Description	Satisfied by
[SRS_Can_01095]	The bus transceiver driver shall support the compile time configuration of one notification to an upper layer for change notification for "wakeup by bus" events	[SWS_CanSM_00652]
[SRS_Can_01096]	The bus transceiver driver shall provide an API to initialize the driver internally	[SWS_CanSM_00652]
[SRS_Can_01097]	CAN Bus Transceiver driver API shall be synchronous	[SWS_CanSM_00652]
[SRS_Can_01098]	The bus transceiver driver shall support an API to send the addressed transceiver into its Standby mode	[SWS_CanSM_00652]
[SRS_Can_01099]	The bus transceiver driver shall support an API to send the addressed transceiver into its Sleep mode	[SWS_CanSM_00652]
[SRS_Can_01100]	The bus transceiver driver shall support an API to send the addressed transceiver into its Normal mode	[SWS_CanSM_00652]
[SRS_Can_01101]	The bus transceiver driver shall support an API to read out the current operation mode of the transceiver of a specified bus within the ECU	[SWS_CanSM_00652]
[SRS_Can_01103]	The bus transceiver driver shall support an API to read out the reason of the last wakeup of a specified bus within the ECU	[SWS_CanSM_00652]
[SRS_Can_01107]	The CAN Transceiver Driver shall support the situation where a wakeup by bus occurs during the same time the transition to standby/sleep is in progress	[SWS_CanSM_00652]
[SRS_Can_01108]	The bus transceiver driver shall support the AUTOSAR ECU state manager in a way that a safe system startup and shutdown is possible	[SWS_CanSM_00652]
[SRS_Can_01109]	The bus transceiver driver shall check the control communication to the transceiver and the reaction of the transceiver for correctness	[SWS_CanSM_00652]
[SRS_Can_01110]	CAN Bus Transceiver driver shall handle the transceiver specific timing requirements internally	[SWS_CanSM_00652]
[SRS_Can_01111]	The CAN Transport Layer shall be the interface layer between PDU Router and CAN Interface for CAN messages needing transport protocol functionalities	[SWS_CanSM_00652]
[SRS_Can_01112]	The CAN Transport Layer interface shall be independent of its internal communication configuration	[SWS_CanSM_00652]
[SRS_Can_01114]	Data Consistency of L-PDUs to transmit shall be guaranteed	[SWS_CanSM_00652]
[SRS_Can_01115]	The bus transceiver driver shall support an API to enable and disable the wakeup notification for each bus separately	[SWS_CanSM_00652]

▽

△

Requirement	Description	Satisfied by
[SRS_Can_01116]	The AUTOSAR CAN Transport Layer shall be able to manage both normal and extended modes in parallel	[SWS_CanSM_00652]
[SRS_Can_01121]	CAN Interface shall be the interface layer between the underlying CAN Driver(s) and CAN transceiver Driver(s) and Upper Layers	[SWS_CanSM_00652]
[SRS_Can_01122]	The CAN driver shall support the situation where a wakeup by bus occurs during the same time the transition to standby/sleep is in progress	[SWS_CanSM_00652]
[SRS_Can_01125]	The CAN stack shall ensure not to lose messages in receive direction	[SWS_CanSM_00652]
[SRS_Can_01126]	The CAN stack shall be able to produce 100% bus load	[SWS_CanSM_00652]
[SRS_Can_01129]	The CAN Interface module shall provide a procedural interface to read out data of single CAN messages by upper layers (Polling mechanism)	[SWS_CanSM_00652]
[SRS_Can_01130]	Receive Status Interface of CAN Interface	[SWS_CanSM_00652]
[SRS_Can_01131]	The CAN Interface module shall provide the possibility to have polling and callback notification mechanism in parallel	[SWS_CanSM_00652]
[SRS_Can_01132]	The CAN driver shall be able to detect notification events message object specific by CAN-Interrupt and polling	[SWS_CanSM_00652]
[SRS_Can_01134]	The CAN Driver shall support multiplexed transmission	[SWS_CanSM_00652]
[SRS_Can_01135]	It shall be possible to configure one or several TX Hardware Objects	[SWS_CanSM_00652]
[SRS_Can_01136]	The CAN Interface module shall provide a service to check for validation of a CAN wake-up event	[SWS_CanSM_00652]
[SRS_Can_01138]	The CAN Bus Transceiver Driver shall provide one callback function for lower layer ICU Driver for wake up by bus events	[SWS_CanSM_00652]
[SRS_Can_01139]	The CAN Interface and Driver shall offer a CAN Controller specific interface for initialization	[SWS_CanSM_00652]
[SRS_Can_01140]	The CAN Interface shall support both Standard (11bit) and Extended (29bit) Identifiers	[SWS_CanSM_00652]
[SRS_Can_01141]	The CAN Interface shall support both Standard (11bit) and Extended (29bit) Identifiers at same time on one network	[SWS_CanSM_00652]

▽

△

Requirement	Description	Satisfied by
[SRS_Can_01142]	The CAN State Manager shall offer a network abstract API to upper layer	[SWS_CanSM_00062] [SWS_CanSM_00065] [SWS_CanSM_00167] [SWS_CanSM_00182] [SWS_CanSM_00183] [SWS_CanSM_00186] [SWS_CanSM_00187] [SWS_CanSM_00188] [SWS_CanSM_00266] [SWS_CanSM_00278] [SWS_CanSM_00282] [SWS_CanSM_00284] [SWS_CanSM_00360] [SWS_CanSM_00369] [SWS_CanSM_00370] [SWS_CanSM_00371] [SWS_CanSM_00372] [SWS_CanSM_00385] [SWS_CanSM_00399] [SWS_CanSM_00410] [SWS_CanSM_00422] [SWS_CanSM_00423] [SWS_CanSM_00425] [SWS_CanSM_00426] [SWS_CanSM_00427] [SWS_CanSM_00428] [SWS_CanSM_00429] [SWS_CanSM_00430] [SWS_CanSM_00431] [SWS_CanSM_00432] [SWS_CanSM_00433] [SWS_CanSM_00434] [SWS_CanSM_00436] [SWS_CanSM_00437] [SWS_CanSM_00438] [SWS_CanSM_00439] [SWS_CanSM_00440] [SWS_CanSM_00441] [SWS_CanSM_00442] [SWS_CanSM_00443] [SWS_CanSM_00444] [SWS_CanSM_00445] [SWS_CanSM_00446] [SWS_CanSM_00447] [SWS_CanSM_00448] [SWS_CanSM_00449] [SWS_CanSM_00450] [SWS_CanSM_00451] [SWS_CanSM_00452] [SWS_CanSM_00453] [SWS_CanSM_00454] [SWS_CanSM_00455] [SWS_CanSM_00456] [SWS_CanSM_00457] [SWS_CanSM_00458] [SWS_CanSM_00459] [SWS_CanSM_00460] [SWS_CanSM_00461] [SWS_CanSM_00462] [SWS_CanSM_00464] [SWS_CanSM_00465] [SWS_CanSM_00466] [SWS_CanSM_00467] [SWS_CanSM_00468] [SWS_CanSM_00469] [SWS_CanSM_00470] [SWS_CanSM_00471] [SWS_CanSM_00472] [SWS_CanSM_00473] [SWS_CanSM_00474] [SWS_CanSM_00475] [SWS_CanSM_00476] [SWS_CanSM_00477] [SWS_CanSM_00478] [SWS_CanSM_00479] [SWS_CanSM_00483] [SWS_CanSM_00484] [SWS_CanSM_00485] [SWS_CanSM_00486] [SWS_CanSM_00487] [SWS_CanSM_00488] [SWS_CanSM_00489] [SWS_CanSM_00490] [SWS_CanSM_00491] [SWS_CanSM_00492] [SWS_CanSM_00493] [SWS_CanSM_00494] [SWS_CanSM_00496] [SWS_CanSM_00497] [SWS_CanSM_00499] [SWS_CanSM_00500] [SWS_CanSM_00502] [SWS_CanSM_00503] [SWS_CanSM_00504] [SWS_CanSM_00505] [SWS_CanSM_00506] [SWS_CanSM_00507] [SWS_CanSM_00508] [SWS_CanSM_00509] [SWS_CanSM_00510] [SWS_CanSM_00511] [SWS_CanSM_00512] [SWS_CanSM_00514] [SWS_CanSM_00515] [SWS_CanSM_00517] [SWS_CanSM_00518] [SWS_CanSM_00521] [SWS_CanSM_00524] [SWS_CanSM_00525] [SWS_CanSM_00526] [SWS_CanSM_00527] [SWS_CanSM_00528] [SWS_CanSM_00529] [SWS_CanSM_00530] [SWS_CanSM_00531] [SWS_CanSM_00532] [SWS_CanSM_00533] [SWS_CanSM_00534] [SWS_CanSM_00535] [SWS_CanSM_00538] [SWS_CanSM_00540] [SWS_CanSM_00541] [SWS_CanSM_00542] [SWS_CanSM_00543] [SWS_CanSM_00550] [SWS_CanSM_00555] ▽

▽

△

Requirement	Description	Satisfied by
		△ [SWS_CanSM_00556] [SWS_CanSM_00557] [SWS_CanSM_00558] [SWS_CanSM_00561] [SWS_CanSM_00569] [SWS_CanSM_00570] [SWS_CanSM_00576] [SWS_CanSM_00577] [SWS_CanSM_00578] [SWS_CanSM_00579] [SWS_CanSM_00580] [SWS_CanSM_00581] [SWS_CanSM_00582] [SWS_CanSM_00584] [SWS_CanSM_00600] [SWS_CanSM_00602] [SWS_CanSM_00603] [SWS_CanSM_00604] [SWS_CanSM_00607] [SWS_CanSM_00608] [SWS_CanSM_00623] [SWS_CanSM_00624] [SWS_CanSM_00625] [SWS_CanSM_00626] [SWS_CanSM_00627] [SWS_CanSM_00628] [SWS_CanSM_00629] [SWS_CanSM_00630] [SWS_CanSM_00631] [SWS_CanSM_00632] [SWS_CanSM_00633] [SWS_CanSM_00634] [SWS_CanSM_00635] [SWS_CanSM_00636] [SWS_CanSM_00639] [SWS_CanSM_00641] [SWS_CanSM_00642] [SWS_CanSM_00651] [SWS_CanSM_00653] [SWS_CanSM_00667]
[SRS_Can_01143]	The CAN State Manager shall support a configurable BusOff recovery time	[SWS_CanSM_00652]
[SRS_Can_01144]	The CAN State Manager shall implement an interface for initialization.	[SWS_CanSM_00600] [SWS_CanSM_00602] [SWS_CanSM_00603] [SWS_CanSM_00604] [SWS_CanSM_00606] [SWS_CanSM_00637]
[SRS_Can_01145]	The CAN State Manager shall control the assigned CAN Devices	[SWS_CanSM_00062] [SWS_CanSM_00065] [SWS_CanSM_00167] [SWS_CanSM_00182] [SWS_CanSM_00183] [SWS_CanSM_00369] [SWS_CanSM_00370] [SWS_CanSM_00396] [SWS_CanSM_00397] [SWS_CanSM_00398] [SWS_CanSM_00399] [SWS_CanSM_00400] [SWS_CanSM_00401] [SWS_CanSM_00410] [SWS_CanSM_00411] [SWS_CanSM_00412] [SWS_CanSM_00413] [SWS_CanSM_00414] [SWS_CanSM_00415] [SWS_CanSM_00416] [SWS_CanSM_00417] [SWS_CanSM_00418] [SWS_CanSM_00419] [SWS_CanSM_00420] [SWS_CanSM_00421] [SWS_CanSM_00423] [SWS_CanSM_00425] [SWS_CanSM_00426] [SWS_CanSM_00427] [SWS_CanSM_00428] [SWS_CanSM_00429] [SWS_CanSM_00430] [SWS_CanSM_00431] [SWS_CanSM_00432] [SWS_CanSM_00433] [SWS_CanSM_00434] [SWS_CanSM_00436] [SWS_CanSM_00437] [SWS_CanSM_00438] [SWS_CanSM_00439] [SWS_CanSM_00440] [SWS_CanSM_00441] [SWS_CanSM_00442] [SWS_CanSM_00443] [SWS_CanSM_00444] [SWS_CanSM_00445] [SWS_CanSM_00446] [SWS_CanSM_00447] [SWS_CanSM_00448] [SWS_CanSM_00449] [SWS_CanSM_00450] [SWS_CanSM_00451] [SWS_CanSM_00452] [SWS_CanSM_00453] [SWS_CanSM_00454] [SWS_CanSM_00455] [SWS_CanSM_00456] [SWS_CanSM_00457] [SWS_CanSM_00458] [SWS_CanSM_00459] [SWS_CanSM_00460] [SWS_CanSM_00461] [SWS_CanSM_00462] [SWS_CanSM_00464] [SWS_CanSM_00465] [SWS_CanSM_00466] [SWS_CanSM_00467] [SWS_CanSM_00468] ▽

▽

△

Requirement	Description	Satisfied by
		△ [SWS_CanSM_00469] [SWS_CanSM_00470] [SWS_CanSM_00471] [SWS_CanSM_00472] [SWS_CanSM_00473] [SWS_CanSM_00474] [SWS_CanSM_00475] [SWS_CanSM_00476] [SWS_CanSM_00477] [SWS_CanSM_00478] [SWS_CanSM_00479] [SWS_CanSM_00483] [SWS_CanSM_00484] [SWS_CanSM_00485] [SWS_CanSM_00486] [SWS_CanSM_00487] [SWS_CanSM_00488] [SWS_CanSM_00489] [SWS_CanSM_00490] [SWS_CanSM_00491] [SWS_CanSM_00492] [SWS_CanSM_00493] [SWS_CanSM_00494] [SWS_CanSM_00496] [SWS_CanSM_00497] [SWS_CanSM_00499] [SWS_CanSM_00500] [SWS_CanSM_00507] [SWS_CanSM_00508] [SWS_CanSM_00509] [SWS_CanSM_00510] [SWS_CanSM_00511] [SWS_CanSM_00512] [SWS_CanSM_00514] [SWS_CanSM_00515] [SWS_CanSM_00517] [SWS_CanSM_00518] [SWS_CanSM_00521] [SWS_CanSM_00524] [SWS_CanSM_00525] [SWS_CanSM_00526] [SWS_CanSM_00527] [SWS_CanSM_00528] [SWS_CanSM_00529] [SWS_CanSM_00531] [SWS_CanSM_00532] [SWS_CanSM_00533] [SWS_CanSM_00534] [SWS_CanSM_00535] [SWS_CanSM_00538] [SWS_CanSM_00540] [SWS_CanSM_00541] [SWS_CanSM_00542] [SWS_CanSM_00543] [SWS_CanSM_00546] [SWS_CanSM_00550] [SWS_CanSM_00555] [SWS_CanSM_00556] [SWS_CanSM_00557] [SWS_CanSM_00558] [SWS_CanSM_00560] [SWS_CanSM_00576] [SWS_CanSM_00577] [SWS_CanSM_00578] [SWS_CanSM_00579] [SWS_CanSM_00580] [SWS_CanSM_00581] [SWS_CanSM_00582] [SWS_CanSM_00584] [SWS_CanSM_00600] [SWS_CanSM_00602] [SWS_CanSM_00603] [SWS_CanSM_00604] [SWS_CanSM_00607] [SWS_CanSM_00608] [SWS_CanSM_00609] [SWS_CanSM_00610] [SWS_CanSM_00611] [SWS_CanSM_00612] [SWS_CanSM_00613] [SWS_CanSM_00616] [SWS_CanSM_00617] [SWS_CanSM_00618] [SWS_CanSM_00619] [SWS_CanSM_00620] [SWS_CanSM_00621] [SWS_CanSM_00622] [SWS_CanSM_00623] [SWS_CanSM_00624] [SWS_CanSM_00625] [SWS_CanSM_00626] [SWS_CanSM_00627] [SWS_CanSM_00628] [SWS_CanSM_00629] [SWS_CanSM_00630] [SWS_CanSM_00631] [SWS_CanSM_00632] [SWS_CanSM_00633] [SWS_CanSM_00634] [SWS_CanSM_00636] [SWS_CanSM_00638] [SWS_CanSM_00639] [SWS_CanSM_00641] [SWS_CanSM_00642] [SWS_CanSM_00651] [SWS_CanSM_00653] [SWS_CanSM_00668] [SWS_CanSM_00669] [SWS_CanSM_00670] [SWS_CanSM_91004]
[SRS_Can_01146]	The CAN State Manager shall contain a CAN BusOff recovery algorithm for each used CAN Controller	[SWS_CanSM_00600] [SWS_CanSM_00602] [SWS_CanSM_00603] [SWS_CanSM_00604] [SWS_CanSM_00606] [SWS_CanSM_00637]
[SRS_Can_01147]	The CAN Driver shall not support remote frames	[SWS_CanSM_00652]

▽

△

Requirement	Description	Satisfied by
[SRS_Can_01148]	The AUTOSAR CAN Transport Layer shall provide a service to enable dynamic setting of protocol parameters	[SWS_CanSM_00652]
[SRS_Can_01149]	The CAN Transport Layer shall support full-duplex communication for TP channels	[SWS_CanSM_00652]
[SRS_Can_01151]	The CAN Interface shall provide a service to check for a CAN Wake-up event.	[SWS_CanSM_00652]
[SRS_Can_01153]	The Tx-Filter shall ensure, that the first message which is sent on the bus is a Wakeup Frame (WUF) in the case of partial networking	[SWS_CanSM_00652]
[SRS_Can_01154]	The bus transceiver driver package shall offer configuration parameters that are required to configure the driver for partial networking	[SWS_CanSM_00652]
[SRS_Can_01155]	The bus transceiver driver shall support the selection of configuration sets	[SWS_CanSM_00652]
[SRS_Can_01156]	The bus transceiver driver shall support wake up events by a Remote Wake-up Pattern (RWUP) or Remote Wake-up Frame (RWUF) if partial networking is supported by the tranceiver hardware	[SWS_CanSM_00652]
[SRS_Can_01157]	The bus transceiver driver shall provide an API for clearing the WUF bit in the tranceiver hardware	[SWS_CanSM_00652]
[SRS_Can_01158]	The CAN stack shall provide a TX offline active mode for ECU passive mode	[SWS_CanSM_00435] [SWS_CanSM_00516] [SWS_CanSM_00539] [SWS_CanSM_00644] [SWS_CanSM_00645] [SWS_CanSM_00646] [SWS_CanSM_00647] [SWS_CanSM_00648] [SWS_CanSM_00649] [SWS_CanSM_00650] [SWS_CanSM_00656]
[SRS_Can_01159]	The CAN Interface shall provide dynamic RX Handles	[SWS_CanSM_00652]
[SRS_Can_01160]	Padding of bytes due to discrete CAN FD DLC	[SWS_CanSM_00652]
[SRS_Can_01161]	The CAN Driver shall not support remote frames	[SWS_CanSM_00652]
[SRS_Can_01162]	CAN Interface shall support classic CAN and CAN FD frames	[SWS_CanSM_00652]
[SRS_Can_01163]	The AUTOSAR CAN Transport Layer shall support classic CAN and CAN FD communication as specified by ISO 15765-2	[SWS_CanSM_00652]
[SRS_Can_01164]	The CAN State Manager shall implement an interface for de-initialization.	[SWS_CanSM_00658] [SWS_CanSM_91001]
[SRS_ModeMgm_-00049]	The Communication Manager shall initiate the wake-up and keep awake physical channels	[SWS_CanSM_00652]
[SRS_ModeMgm_-09001]	The number and names of main states and the transitions between main states shall be standardized.	[SWS_CanSM_00652]

▽

△

Requirement	Description	Satisfied by
[SRS_ModeMgm_-09009]	The ECU State Manager shall provide the ability to execute external, statically-configured code at each transition between ECU states	[SWS_CanSM_00652]
[SRS_ModeMgm_-09017]	The ECU State Manager shall provide an API to query the current ECU state	[SWS_CanSM_00652]
[SRS_ModeMgm_-09028]	The Watchdog Manager shall support multiple watchdog instances	[SWS_CanSM_00652]
[SRS_ModeMgm_-09071]	It shall be possible to limit communication modes independently for each physical channel	[SWS_CanSM_00652]
[SRS_ModeMgm_-09072]	ECU shutdown shall be forced	[SWS_CanSM_00652]
[SRS_ModeMgm_-09078]	The Communication Manager shall coordinate multiple communication requests	[SWS_CanSM_00652]
[SRS_ModeMgm_-09080]	Each physical channel shall be controlled by an independent communication mode	[SWS_CanSM_00652]
[SRS_ModeMgm_-09081]	The Communication Manager shall provide an API allowing collecting communication requests	[SWS_CanSM_00652]
[SRS_ModeMgm_-09083]	The Communication Manager shall support two communication modes for each physical channel	[SWS_CanSM_00652]
[SRS_ModeMgm_-09084]	The Communication Manager shall provide an API which allows application to query the current communication mode	[SWS_CanSM_00063] [SWS_CanSM_00652]
[SRS_ModeMgm_-09085]	The Communication Manager shall provide an indication of communication mode changes	[SWS_CanSM_00652]
[SRS_ModeMgm_-09087]	The Minimum duration of communication request after wakeup shall be configurable	[SWS_CanSM_00652]
[SRS_ModeMgm_-09089]	The Communication Manager shall be able to prevent waking up physical channels	[SWS_CanSM_00652]
[SRS_ModeMgm_-09090]	Relationship between users and physical channels shall be configurable at pre compile time	[SWS_CanSM_00652]
[SRS_ModeMgm_-09097]	The ECU State Manager module shall start a timeout after receiving a wake-up indication	[SWS_CanSM_00652]
[SRS_ModeMgm_-09098]	Storing the wake-up reasons shall be available	[SWS_CanSM_00652]
[SRS_ModeMgm_-09100]	Selection of wake-up sources shall be configurable	[SWS_CanSM_00652]
[SRS_ModeMgm_-09101]	An API to query the reset reason shall be provided	[SWS_CanSM_00652]
[SRS_ModeMgm_-09102]	API for selecting the sleep mode shall be provided	[SWS_CanSM_00652]
[SRS_ModeMgm_-09104]	ECU State Manager shall take over control after OS shutdown	[SWS_CanSM_00652]

▽

△

Requirement	Description	Satisfied by
[SRS_ModeMgm_-09106]	The list of entities supervised by the Watchdog Manager shall be configurable at pre-compile time	[SWS_CanSM_00652]
[SRS_ModeMgm_-09107]	The Watchdog Manager shall provide an initialization service	[SWS_CanSM_00652]
[SRS_ModeMgm_-09109]	It shall be possible to prohibit the disabling of watchdog	[SWS_CanSM_00652]
[SRS_ModeMgm_-09110]	The watchdog Manager shall provide a service interface, to select a mode of the Watchdog Manager	[SWS_CanSM_00652]
[SRS_ModeMgm_-09112]	The Watchdog Manager shall cyclically check the periodicity of the supervised entities	[SWS_CanSM_00652]
[SRS_ModeMgm_-09113]	Initialization of Basic Software modules shall be done	[SWS_CanSM_00652]
[SRS_ModeMgm_-09114]	Starting/invoking the shutdown process shall be provided	[SWS_CanSM_00652]
[SRS_ModeMgm_-09115]	The ECU State Manager shall include a mechanism to evaluate the condition to stay in the RUN state	[SWS_CanSM_00652]
[SRS_ModeMgm_-09116]	Requesting and releasing the RUN state shall be provided	[SWS_CanSM_00652]
[SRS_ModeMgm_-09118]	The ECU State Manager shall provide a mechanism to enter a step by step decreasing power mode	[SWS_CanSM_00652]
[SRS_ModeMgm_-09119]	Several sleep modes shall be available	[SWS_CanSM_00652]
[SRS_ModeMgm_-09120]	Configuration of initialization process of Basic Software modules shall be available	[SWS_CanSM_00652]
[SRS_ModeMgm_-09122]	Configuration of users of the ECU State Manager	[SWS_CanSM_00652]
[SRS_ModeMgm_-09125]	The Watchdog Manager shall provide a service allowing the Update temporal program flow monitoring	[SWS_CanSM_00652]
[SRS_ModeMgm_-09126]	An API for querying the wake-up reason shall be provided	[SWS_CanSM_00652]
[SRS_ModeMgm_-09127]	The ECU State Manager shall de-initialize Basic Software modules where appropriate during the shutdown process	[SWS_CanSM_00652]
[SRS_ModeMgm_-09128]	Several shutdown targets shall be supported	[SWS_CanSM_00652]
[SRS_ModeMgm_-09132]	It shall be possible to assign Network Management to physical channels	[SWS_CanSM_00652]
[SRS_ModeMgm_-09133]	It shall be possible to assign physical channels to the Communication Manager	[SWS_CanSM_00652]
[SRS_ModeMgm_-09136]	The ECU State Manager shall be the receiver of all wake-up events	[SWS_CanSM_00652]
[SRS_ModeMgm_-09141]	The Communication Manager shall be able to configure the physical channel wake-up prevention	[SWS_CanSM_00652]

▽

△

Requirement	Description	Satisfied by
[SRS_ModeMgm_-09143]	The Watchdog Manager shall set the triggering condition during inactive monitoring	[SWS_CanSM_00652]
[SRS_ModeMgm_-09145]	Wake-sleep operation shall be supported	[SWS_CanSM_00652]
[SRS_ModeMgm_-09146]	Configuration of time triggered increased inoperation shall be provided	[SWS_CanSM_00652]
[SRS_ModeMgm_-09147]	Configuration of de-initialization process of Basic Software modules shall be provided	[SWS_CanSM_00652]
[SRS_ModeMgm_-09149]	The Communication Manager shall provide an API for querying the requested communication mode	[SWS_CanSM_00652]
[SRS_ModeMgm_-09155]	The Communication Manager shall provide a counter for inhibited communication requests	[SWS_CanSM_00652]
[SRS_ModeMgm_-09156]	It shall be provided an API to retrieve the number of inhibited "Full Communication" mode requests	[SWS_CanSM_00652]
[SRS_ModeMgm_-09157]	It shall be possible to revoke a communication mode limitation, independently for each physical channel	[SWS_CanSM_00652]
[SRS_ModeMgm_-09158]	The Watchdog Manager shall support Post build time and mode dependent selectable configuration sets for the Watchdog Manager	[SWS_CanSM_00652]
[SRS_ModeMgm_-09159]	The Watchdog Manager shall report failure of temporal or program flow monitoring to DEM	[SWS_CanSM_00652]
[SRS_ModeMgm_-09160]	The Watchdog Manager shall provide the indication of failed temporal monitoring	[SWS_CanSM_00652]
[SRS_ModeMgm_-09161]	The Watchdog Manager shall reset the triggering condition in the Watchdog Driver in Case of temporal failure	[SWS_CanSM_00652]
[SRS_ModeMgm_-09162]	The Watchdog Manager shall be able to notify the software of an upcoming watchdog reset	[SWS_CanSM_00652]
[SRS_ModeMgm_-09163]	It shall be possible to configure a delay before provoking a watchdog reset	[SWS_CanSM_00652]
[SRS_ModeMgm_-09164]	Shutdown synchronization for SW-Components shall be supported	[SWS_CanSM_00652]
[SRS_ModeMgm_-09165]	The ECU State Manager shall provide services to request and release the POST-RUN state	[SWS_CanSM_00652]
[SRS_ModeMgm_-09166]	The ECU State Manager shall evaluate the condition to stay in the POST-RUN state	[SWS_CanSM_00652]
[SRS_ModeMgm_-09168]	The Communication Manager shall support users that are connected to no physical channel	[SWS_CanSM_00652]
[SRS_ModeMgm_-09169]	The Watchdog Manager shall be able to immediately reset the MCU	[SWS_CanSM_00652]

▽

△

Requirement	Description	Satisfied by
[SRS_ModeMgm_-09172]	It shall be possible to evaluate the current communication mode	[SWS_CanSM_00652]
[SRS_ModeMgm_-09173]	A Run State shall have a minimum duration	[SWS_CanSM_00652]
[SRS_ModeMgm_-09174]	The BSW Mode Manager shall support the 'disable normal Communication'	[SWS_CanSM_00652]
[SRS_ModeMgm_-09175]	A configurable Set of Mode dependent enabled and concomitant disabled IPDU groups shall be supported	[SWS_CanSM_00652]
[SRS_ModeMgm_-09176]	Configurable Sets of Mode dependent enabled I-PDU Groups shall be supported	[SWS_CanSM_00652]
[SRS_ModeMgm_-09177]	The rules of the mode arbitration shall be pre-compile and post-build configurable	[SWS_CanSM_00652]
[SRS_ModeMgm_-09178]	The lists of mode transition specific actions shall be pre-compile and post-build configurable	[SWS_CanSM_00652]
[SRS_ModeMgm_-09179]	The BSW Mode Manager shall provide an Interface to allow Mode Requests of SW-C's	[SWS_CanSM_00652]
[SRS_ModeMgm_-09180]	The BSW Mode Manager shall evaluate the current mode requests	[SWS_CanSM_00652]
[SRS_ModeMgm_-09182]	The BSW Mode Manager shall propagate a performed mode change to all local SW-Cs	[SWS_CanSM_00652]
[SRS_ModeMgm_-09183]	Configurable Mode Activation initiated Reset of Signals to Initial Values shall be supported	[SWS_CanSM_00652]
[SRS_ModeMgm_-09184]	The mode manager shall be able to use a COM interface to activate, respectively deactivate, I-PDU groups	[SWS_CanSM_00652]
[SRS_ModeMgm_-09185]	A persistent Alarm Clock used by local SW-Cs shall be provided	[SWS_CanSM_00652]
[SRS_ModeMgm_-09186]	Alarm Clock shall be active while the ECU is powered	[SWS_CanSM_00652]
[SRS_ModeMgm_-09187]	In Case of wakeup, all the alarm clock shall be canceled	[SWS_CanSM_00652]
[SRS_ModeMgm_-09188]	In Case of startup, all the alarm clock shall be canceled	[SWS_CanSM_00652]
[SRS_ModeMgm_-09189]	Consecutive requests shall honor the earliest expiring alarm only	[SWS_CanSM_00652]
[SRS_ModeMgm_-09190]	The alarm clock service shall allow setting an alarm relative to the current time using a time resolution of seconds	[SWS_CanSM_00652]
[SRS_ModeMgm_-09194]	The alarm clock service shall allow setting the clock	[SWS_CanSM_00652]
[SRS_ModeMgm_-09199]	The alarm clock service shall allow setting an alarm absolute by using an absolute time with a resolution of seconds	[SWS_CanSM_00652]
[SRS_ModeMgm_-09207]	ComM shall allow for additional bus specific state managers	[SWS_CanSM_00652]

▽

△

Requirement	Description	Satisfied by
[SRS_ModeMgm_-09220]	It shall be possible to configure all the transition relations	[SWS_CanSM_00652]
[SRS_ModeMgm_-09221]	The Watchdog Manager shall check the correct sequence of code execution in supervised entities	[SWS_CanSM_00652]
[SRS_ModeMgm_-09222]	The Watchdog Manager shall provide a service allowing the Update logical program flow monitoring	[SWS_CanSM_00652]
[SRS_ModeMgm_-09223]	The Watchdog Manager shall support Post build time and mode dependent selectable configuration of transition relations	[SWS_CanSM_00652]
[SRS_ModeMgm_-09225]	The Watchdog Manager shall provide the indication of failed logical monitoring	[SWS_CanSM_00652]
[SRS_ModeMgm_-09226]	The Watchdog Manager shall reset reset the triggering condition in the Watchdog Driver in Case of logical program flow violation	[SWS_CanSM_00652]
[SRS_ModeMgm_-09228]	The BSW Mode Manager shall provide an Interface to allow Mode Requests of BSW Modules	[SWS_CanSM_00652]
[SRS_ModeMgm_-09229]	The mode manager shall be able to make generic, configured callouts of void functions to other BSW modules	[SWS_CanSM_00652]
[SRS_ModeMgm_-09230]	All actions shall only be performed on mode change	[SWS_CanSM_00652]
[SRS_ModeMgm_-09231]	The Watchdog Manager shall periodically set the triggering condition in the Watchdog Driver as long as the monitoring has not failed	[SWS_CanSM_00652]
[SRS_ModeMgm_-09232]	The Watchdog Manager shall provide a service to cause a watchdog reset	[SWS_CanSM_00652]
[SRS_ModeMgm_-09234]	The EcuM shall handle the initialization of Basic Software modules	[SWS_CanSM_00652]
[SRS_ModeMgm_-09235]	The ECU State Manager shall offer two targets for shutting down the ECU	[SWS_CanSM_00652]
[SRS_ModeMgm_-09236]	There shall be one instance of the function EcuM_Init that distinguishes between the different cores	[SWS_CanSM_00652]
[SRS_ModeMgm_-09237]	RTE_Start shall be called on each core.	[SWS_CanSM_00652]
[SRS_ModeMgm_-09238]	State changes shall be ECU global	[SWS_CanSM_00652]
[SRS_ModeMgm_-09239]	To shutdown, ShutdownAllCores shall be called on the master core after synchronizing all cores	[SWS_CanSM_00652]
[SRS_ModeMgm_-09240]	ComM shall notify BswM of any PNC communication state change	[SWS_CanSM_00652]
[SRS_ModeMgm_-09241]	BswM shall be able to request communication modes for existing CommUsers	[SWS_CanSM_00652]

▽

△

Requirement	Description	Satisfied by
[SRS_ModeMgm_-09243]	The Communication Manager shall be able to handle the Partial Networks on Flexray, CAN and Ethernet	[SWS_CanSM_00652]
[SRS_ModeMgm_-09244]	The number of supported PNCs shall be configurable strictly at pre-compile time	[SWS_CanSM_00652]
[SRS_ModeMgm_-09245]	Enabling or disabling the Partial Network Cluster management in Com M shall be post-build selectable.	[SWS_CanSM_00652]
[SRS_ModeMgm_-09246]	The communication manager shall arbitrate and coordinate requests from users on physical channel and users on PNCs	[SWS_CanSM_00652]
[SRS_ModeMgm_-09247]	For each configured PNC an independent state machine shall be instantiated	[SWS_CanSM_00652]
[SRS_ModeMgm_-09248]	it shall be possible to distinguish between internal and external PNC activation requests	[SWS_CanSM_00652]
[SRS_ModeMgm_-09249]	PNC gateway and coordination functionality	[SWS_CanSM_00652]
[SRS_ModeMgm_-09250]	PNC activation requests shall be exchanged with the Network Management via a PNC bit vector	[SWS_CanSM_00652]
[SRS_ModeMgm_-09251]	PNC communication state shall be forwarded to the BswM	[SWS_CanSM_00598] [SWS_CanSM_00652]
[SRS_ModeMgm_-09253]	The BswM shall be able to set the halt mode for each single CPU Core independently	[SWS_CanSM_00652]
[SRS_ModeMgm_-09254]	Validation and handling of a wakeup event shall be done locally	[SWS_CanSM_00652]
[SRS_ModeMgm_-09255]		[SWS_CanSM_00652]
[SRS_ModeMgm_-09256]	PNC Gateway Functionality shall consider systems with more than one gateways connected to the same network	[SWS_CanSM_00652]
[SRS_ModeMgm_-09270]	The ECU State Manager shall provide a service for the selection of the shutdown target	[SWS_CanSM_00652]
[SRS_ModeMgm_-09271]	The ECU State Manager shall provide a service for the retrieval of the current shutdown target	[SWS_CanSM_00652]
[SRS_ModeMgm_-09272]	The ECU State Manager shall provide a service for the retrieval of the last sleep targets	[SWS_CanSM_00652]
[SRS_ModeMgm_-09274]	The ECU State Manager shall provide a service for the retrieval of the selected reset modality	[SWS_CanSM_00652]
[SRS_ModeMgm_-09275]	The ECU State Manager shall provide a service for querying the time of previous resets	[SWS_CanSM_00652]
[SRS_ModeMgm_-09276]	The ECU State Manager shall provide a service allowing the selection of the boot target	[SWS_CanSM_00652]

▽

△

Requirement	Description	Satisfied by
[SRS_ModeMgm_-09277]	The ECU State Manager shall provide an alarm clock service which shall allow the retrieval of clock values	[SWS_CanSM_00652]

Table 6.1: RequirementsTracing

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).

Note:

CanSM module will not notify ComM about its communication mode after initialization, unless a communication mode has explicitly been requested by ComM.

7.1 General requirements

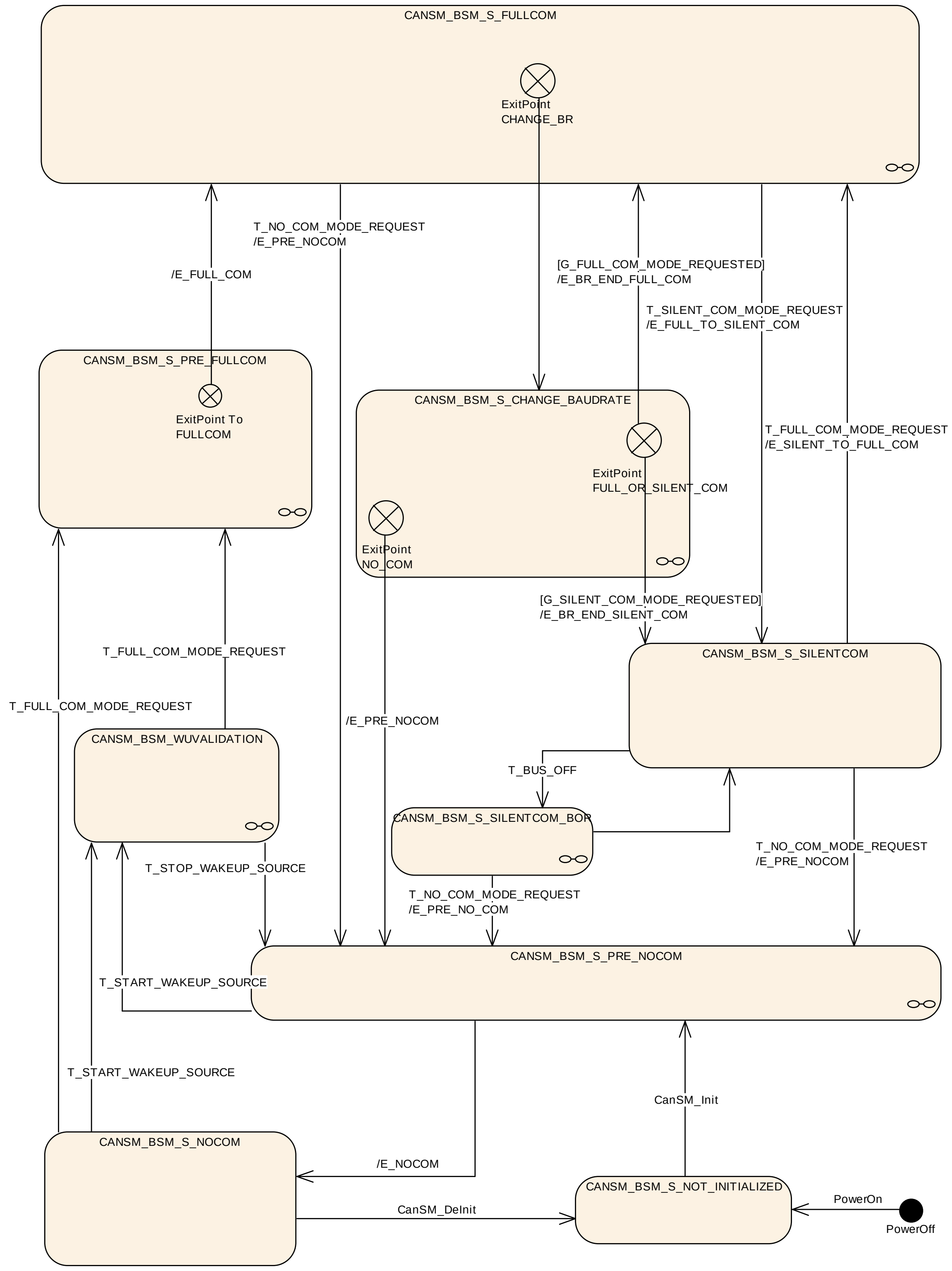
CANSM_BSM_S_FULLCOM
ExitPoint CHANGE_BR
T_NO_COM_MODE_REQUEST /E_PRE_NOCOM
/E_FULL_COM
[G_FULL_COM_MODE_REQUESTED] /E_BR_END_FULL_COM
T_SILENT_COM_MODE_REQUEST /E_FULL_TO_SILENT_COM
CANSM_BSM_S_PRE_FULLCOM
ExitPoint To FULLCOM
CANSM_BSM_S_CHANGE_BAUDRATE
ExitPoint FULL_OR_SILENT_COM
ExitPoint NO_COM
T_FULL_COM_MODE_REQUEST /E_SILENT_TO_FULL_COM
[G_SILENT_COM_MODE_REQUESTED] /E_BR_END_SILENT_COM
CANSM_BSM_S_SILENTCOM
T_FULL_COM_MODE_REQUEST
T_FULL_COM_MODE_REQUEST
/E_PRE_NOCOM
CANSM_BSM_WUVALIDATION
T_BUS_OFF
CANSM_BSM_S_SILENTCOM_BOR
T_NO_COM_MODE_REQUEST /E_PRE_NOCOM
T_STOP_WAKEUP_SOURCE
T_NO_COM_MODE_REQUEST /E_PRE_NO_COM
CANSM_BSM_S_PRE_NOCOM
T_START_WAKEUP_SOURCE
T_START_WAKEUP_SOURCE
CanSM_Init
CANSM_BSM_S_NOCOM
/E_NOCOM
CANSM_BSM_S_NOT_INITIALIZED
PowerOn
PowerOff
CanSM_DeInit

Figure 7.1: CANSM_BSM, state machine diagram for one CAN network

[SWS_CanSM_00266] ⌈The CanSM module shall store the current network mode for each configured CAN network internally (ref. to [ECUC_CanSM_00126]).⌋*(SRS_-Can_01142)*

[SWS_CanSM_00284] ⌈The internally stored network modes of the CanSM module can have the values `COMM_NO_COMMUNICATION`, `COMM_SILENT_COMMUNICATION`, `COMM_FULL_COMMUNICATION`.⌋*(SRS_Can_01142)*

[SWS_CanSM_00428] ⌈All effects of the CanSM state machine `CANSM_BSM` shall be operated in the context of the CanSM main function (ref. to [SWS_CanSM_00065]).⌋*(SRS_Can_01142, SRS_Can_01145)*

[SWS_CanSM_00278] ⌈If the CanSM state machine `CANSM_BSM` is in the state `CANSM_BSM_S_NOT_INITIALIZED`, it shall deny network mode requests from the ComM module (ref. to [SWS_CanSM_00062]).⌋*(SRS_Can_01142)*

[SWS_CanSM_00385] ⌈If CanSM has repeated one of the CanIf API calls `CanIf_-SetControllerMode` (ref. to [SWS_CanSM_91002]), `CanIf_SetTrcvMode` (ref. to [SWS_CanSM_91002]), `CanIf_ClearTrcvWufFlag` (ref. [SWS_CanSM_91002]) or `CanIf_CheckTrcvWakeFlag` (ref. [SWS_CanSM_91002]) more often than `CanSMModeRequestRepetitionMax` (ref. to [ECUC_CanSM_00335]) without getting the return value `E_OK` or without getting the corresponding mode indication callbacks `CanSM_ControllerModeIndication`, `CanSM_Transceiver-ModeIndication`, `CanSM_ClearTrcvWufFlagIndication` or `CanSM_Check-TransceiverWakeFlagIndication`, CanSM shall call the function `Det_Re-portRuntimeError` (ref. to [SWS_CanSM_91002]) with `ErrorId` parameter `CANSM_E_MODE_REQUEST_TIMEOUT`.⌋*(SRS_Can_01142)*

[SWS_CanSM_00422] ⌈If the CanIf module notifies PN availability for a configured CAN Transceiver to the CanSM module with the callback function `CanSM_ConfirmP-nAvailability` (ref. to [SWS_CanSM_00419]), then the CanSM module shall call the API `CanNm_ConfirmPnAvailability` (ref. to [SWS_CanSM_91002]) with the related CAN network as `channel` to confirm the PN availability to the CanNm module.⌋*(SRS_Can_01142)*

[SWS_CanSM_00667]{DRAFT} ⌈If the CanIf module notifies PN availability for a configured CAN Controller to the CanSM module with the callback function `CanSM_ConfirmCtrlPnAvailability` (ref. to [SWS_CanSM_91004]), then the CanSM module shall call the API `CanNm_ConfirmPnAvailability` (ref. to [SWS_CanSM_91002]) with the related CAN network as `channel` to confirm the PN availability to the CanNm module.⌋*(SRS_Can_01142)*

[SWS_CanSM_00560] ⌈If no `CanSMTransceiverId` (ref. to [ECUC_CanSM_00137]) is configured for a CAN Network, then the CanSM module shall bypass all specified `CanIf_SetTrcvMode` (ref. to [SWS_CanSM_91002]) (e.g. [SWS_CanSM_00446]) calls for the CAN Network and proceed in the different state transitions as if it has got the supposed `CanSM_TransceiverModeIndication` already (e.g. [SWS_CanSM_00448]).⌋*(SRS_Can_01145)*

[SWS_CanSM_00635] ⌈The CanSM module shall store for each configured CAN network (ref. to [ECUC_CanSM_00126]) the latest communication mode request, which has been accepted by returning `E_OK` in the API request `CanSM_RequestComMode` (ref. to [SWS_CanSM_00062], [SWS_CanSM_00182]) and use it as trigger for the state machine of the related CAN network, [SWS_CanSM_00427], [SWS_CanSM_00429], [SWS_CanSM_00499], [SWS_CanSM_00542], [SWS_CanSM_00543], [SWS_CanSM_00425], [SWS_CanSM_00426]).⌋*(SRS_Can_01142)*

[SWS_CanSM_00638] ⌈The CanSM module shall store after every successful CAN controller mode change (ref. to [SWS_CanSM_00396]) or bus-off conditioned change to `CAN_CS_STOPPED` (ref. to [SWS_CanSM_00064]), the changed mode internally for each CAN controller.⌋*(SRS_Can_01145)*

7.2 State machine for each CAN network

The diagram (ref. to Figure 7.1) specifies the behavioral state machine of the CanSM module, which shall be implemented for each configured CAN network (ref. to [ECUC_CanSM_00126])

7.2.1 Trigger: PowerOn

[SWS_CanSM_00424] ⌈After PowerOn the CanSM state machines shall be in the state `CANSM_BSM_NOT_INITIALIZED`.⌋*()*

7.2.2 Trigger: CanSM_Init

[SWS_CanSM_00423] ⌈If the CanSM module is requested with the function `CanSM_Init`, this shall trigger the CanSM state machines for all configured CAN Networks (ref. to [ECUC_CanSM_00126]) with the trigger `CanSM_Init`.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.3 Trigger: CanSM_DeInit

[SWS_CanSM_00658] ⌈If the CanSM module is requested with the function `CanSM_DeInit`, this shall trigger the CanSM state machines for all configured CAN Networks (ref. to [ECUC_CanSM_00126]) with the trigger `CanSM_DeInit`.⌋*(SRS_Can_01164)*

Note: Caller of the `CanSM_DeInit` function has to ensure all CAN networks are in the state `CANSM_NO_COMMUNICATION`

7.2.4 Trigger: T_START_WAKEUP_SOURCE

[SWS_CanSM_00607] ⌈If the API request `CanSM_StartWakeupSource` (ref. to [SWS_CanSM_00609]) returns `E_OK` (ref. to [SWS_CanSM_00616]), it shall trigger the state machine with `T_START_WAKEUP_SOURCE`.⌋*(SRS_Can_01142, SRS_Can_-01145)*

7.2.5 Trigger: T_STOP_WAKEUP_SOURCE

[SWS_CanSM_00608] ⌈If the API request `CanSM_StopWakeupSource` (ref. to [SWS_CanSM_00610]) returns `E_OK` (ref. to [SWS_CanSM_00622]), it shall trigger the state machine with `T_STOP_WAKEUP_SOURCE`.⌋*(SRS_Can_01142, SRS_Can_-01145)*

7.2.6 Trigger: T_FULL_COM_MODE_REQUEST

[SWS_CanSM_00425] ⌈The API request `CanSM_RequestComMode` (ref. to [SWS_CanSM_00635]) 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 `CanSMComMNetworkHandleRef` (ref. to [ECUC_CanSM_00161]).⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.7 Trigger: T_SILENT_COM_MODE_REQUEST

[SWS_CanSM_00499] ⌈The API request `CanSM_RequestComMode` (ref. to [SWS_CanSM_00635]) with the parameter `ComM_Mode` equal to `COMM_SILENT_COMMUNICATION` shall trigger the sub state machine `CANSM_BSM_S_FULLCOM` with `T_SILENT_COM_MODE_REQUEST`, which corresponds to the function parameter network and the configuration parameter `CanSM-ComMNetworkHandleRef` (ref. to [ECUC_CanSM_00161]).⌋*(SRS_Can_01145, SRS_Can_01142)*

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

7.2.8 Trigger: T_NO_COM_MODE_REQUEST

[SWS_CanSM_00426] ⌈The API request `CanSM_RequestComMode` (ref. to [SWS_CanSM_00635]) 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 CanSMComMNetworkHandleRef) (ref. to [ECUC_CanSM_00161]).⌋ *(SRS_Can_01142, SRS_Can_01145)*

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)".

7.2.9 Trigger: T_BUS_OFF

[SWS_CanSM_00606] ⌈The callback function CanSM_ControllerBusOff (ref. to [SWS_CanSM_00064]) shall trigger the state machine CANSM_BSM 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.⌋ *(SRS_Can_01144, SRS_Can_01146)*

7.2.10 Guarding condition: G_FULL_COM_MODE_REQUESTED

[SWS_CanSM_00427] ⌈The guarding condition G_FULL_COM_MODE_REQUESTED of the CanSM_BSM state machine shall evaluate, if the latest accepted communication mode request with CanSM_RequestComMode (ref. to [SWS_CanSM_00635]) for the respective network handle of the state machine has been with the parameter ComM_Mode equal to COMM_FULL_COMMUNICATION.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.11 Guarding condition: G_SILENT_COM_MODE_REQUESTED

[SWS_CanSM_00429] ⌈The guarding condition G_SILENT_COM_MODE_REQUESTED of the CanSM_BSM state machine shall evaluate, if the latest accepted communication mode request with CanSM_RequestComMode (ref. to [SWS_CanSM_00635]) for the respective network handle of the state machine has been with the parameter ComM_Mode equal to COMM_SILENT_COMMUNICATION.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.12 Effect: E_PRE_NOCOM

[SWS_CanSM_00431] ⌈The effect E_PRE_NOCOM of the CanSM_BSM state machine shall call for the corresponding CAN network the API BswM_CanSM_CurrentState (ref. to [SWS_CanSM_91002]) with the parameters Network := CanSMComMNetworkHandleRef and CurrentState := CANSM_BSWM_NO_COMMUNICATION.⌋ *(SRS_Can_01142, SRS_Can_01145)*

7.2.13 Effect: E_NOCOM

[SWS_CanSM_00430] ⌈The effect E_NOCOM of the CanSM_BSM state machine shall change the internally stored network mode (ref. to [SWS_CanSM_00266]) of the addressed CAN network to COMM_NO_COMMUNICATION.⌋*(SRS_Can_01142, SRS_Can_01145)*

[SWS_CanSM_00651] ⌈If a communication mode request for the network is present already (ref. to [SWS_CanSM_00635]) and the stored communication mode request is COMM_NO_COMMUNICATION, then the effect E_NOCOM of the CanSM_BSM state machine shall call the API ComM_BusSM_ModeIndication (ref. to [SWS_CanSM_91002]) with the parameters Channel := CanSMComMNetworkHandleRef (ref. to [ECUC_CanSM_00161]) and ComMode := COMM_NO_COMMUNICATION.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.14 Effect: E_FULL_COM

[SWS_CanSM_00539] ⌈If ECU passive is FALSE (ref. to [SWS_CanSM_00646]), then the effect E_FULL_COM of the CanSM_BSM state machine shall call at 1st place for each configured CAN controller of the CAN network the API CanIf_SetPduMode (ref. to [SWS_CanSM_91002]) with the parameters ControllerId := CanSMControllerId (ref. to [ECUC_CanSM_00141]) and PduModeRequest := CANIF_ONLINE.⌋*(SRS_Can_01158)*

[SWS_CanSM_00647] ⌈If ECU passive is TRUE (ref. to [SWS_CanSM_00646]), then the effect E_FULL_COM of the CanSM_BSM state machine shall call at 1st place for each configured CAN controller of the CAN network the API CanIf_SetPduMode (ref. to [SWS_CanSM_91002]) with the parameters ControllerId := CanSMControllerId (ref. to [ECUC_CanSM_00141]) and PduModeRequest := CANIF_TX_OFFLINE_ACTIVE.⌋*(SRS_Can_01158)*

[SWS_CanSM_00435] ⌈After considering [SWS_CanSM_00539] and [SWS_CanSM_00647] in context of the effect E_FULL_COM of the CanSM_BSM state machine, the CanSM module shall call the API ComM_BusSM_ModeIndication (ref. to [SWS_CanSM_91002]) for the corresponding CAN network with the parameters Channel := CanSMComMNetworkHandleRef (ref. to [ECUC_CanSM_00161]) and ComMode := COMM_FULL_COMMUNICATION.⌋*(SRS_Can_01158)*

[SWS_CanSM_00540] ⌈After considering [SWS_CanSM_00435] in context of the effect E_FULL_COM of the CanSM_BSM state machine, the CanSM module shall call the API BswM_CanSM_CurrentState (ref. to [SWS_CanSM_91002]) for the corresponding CAN network with the parameters Network := CanSMComMNetworkHandleRef and CurrentState := CANSM_BSWM_FULL_COMMUNICATION.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.15 Effect: E_FULL_TO_SILENT_COM

[SWS_CanSM_00434] ⌈The effect E_FULL_TO_SILENT_COM of the CanSM_BSM state machine shall call at 1st place for the corresponding CAN network the API BswM_CanSM_CurrentState (ref. to [SWS_CanSM_91002]) with the parameters Network := CanSMComMNetworkHandleRef and CurrentState := CANSM_BSWM_SILENT_COMMUNICATION.⌋*(SRS_Can_01142, SRS_Can_01145)*

[SWS_CanSM_00541] ⌈The effect E_FULL_TO_SILENT_COM of the CanSM_BSM state machine shall call at 2nd place for each configured CAN controller of the CAN network the API CanIf_SetPduMode (ref. to [SWS_CanSM_91002]) with the parameters ControllerId := CanSMControllerId (ref. to [ECUC_CanSM_00141]) and PduModeRequest := CANIF_TX_OFFLINE.⌋*(SRS_Can_01142, SRS_Can_01145)*

[SWS_CanSM_00538] ⌈The effect E_FULL_TO_SILENT_COM of the CanSM_BSM state machine shall call at 3th place for the corresponding CAN network the API ComM_BusSM_ModeIndication (ref. to [SWS_CanSM_91002]) with the parameters Channel := CanSMComMNetworkHandleRef (ref. to [ECUC_CanSM_00161]) and ComMode := COMM_SILENT_COMMUNICATION.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.16 Effect: E_BR_END_FULL_COM

[SWS_CanSM_00432] ⌈The effect E_BR_END_FULL_COM of the CanSM_BSM state machine shall be the same as E_FULL_COM.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.17 Effect: E_BR_END_SILENT_COM

[SWS_CanSM_00433] ⌈The effect E_BR_END_SILENT_COM of the CanSM_BSM state machine shall be the same as E_FULL_TO_SILENT_COM.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.18 Effect: E_SILENT_TO_FULL_COM

[SWS_CanSM_00550] ⌈The effect E_SILENT_TO_FULL_COM of the CanSM_BSM state machine shall be the same as E_FULL_COM.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.19 Sub state machine CANSM_BSM_WUVALIDATION

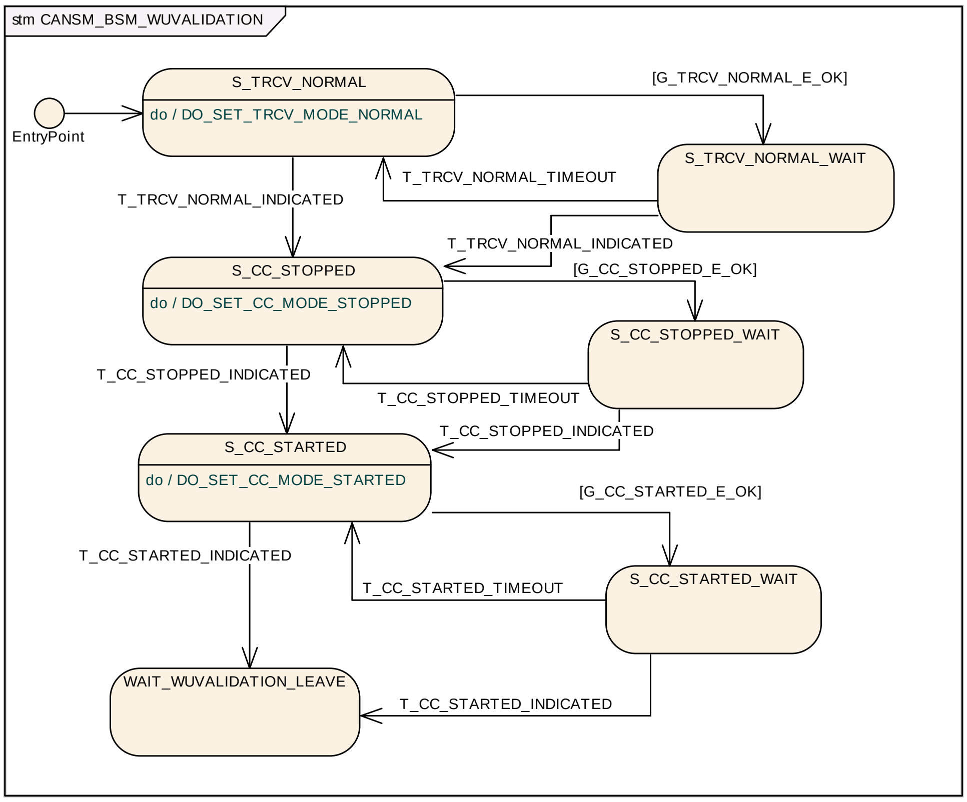

Figure 7.2: CANSM_BSM_WUVALIDATION, sub state machine of CANSM_BSM

7.2.19.1 State operation to do in: S_TRCV_NORMAL

[SWS_CanSM_00623] ⌈If for the CAN network a CAN Transceiver is configured (ref. to [ECUC_CanSM_00137]), then as long the sub state machine CANSM_BSM_WUVALIDATION 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 [ECUC_CanSM_00137]) the API request CanIf_SetTrcvMode (ref. to [SWS_CanSM_91002]) with TransceiverMode equal to CANTRCV_TRCVMODE_NORMAL.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.19.2 Guarding condition G_TRCV_NORMAL_E_OK

[SWS_CanSM_00624] ⌈The guarding condition G_TRCV_NORMAL_E_OK of the sub state machine CANSM_BSM_WUVALIDATION shall be passed, if the API call of [SWS_CanSM_00483] has returned E_OK.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.19.3 Trigger: T_TRCV_NORMAL_INDICATED

[SWS_CanSM_00625] ⌈If CanSM module has got the CANTRCV_TRCVMODE_NORMAL mode indication (ref. to [SWS_CanSM_00399]) for the configured CAN Transceiver of the CAN network (ref. to [ECUC_CanSM_00137]) after the respective request (ref. to [SWS_CanSM_00623]), this shall trigger the sub state machine machine CANSM_BSM_WUVALIDATION of the CAN network with T_TRCV_NORMAL_INDICATED.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.19.4 Trigger: T_TRCV_NORMAL_TIMEOUT

[SWS_CanSM_00626] ⌈After a timeout of CANSM_MODEREQ_REPEAT_TIME (ref. to [ECUC_CanSM_00336]) for the supposed transceiver normal indication (ref. to [SWS_CanSM_00625]), this condition shall trigger the sub state machine CANSM_BSM_WUVALIDATION of the respective network with T_TRCV_NORMAL_TIMEOUT.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.19.5 State operation to do in: S_CC_STOPPED

[SWS_CanSM_00627] ⌈As long the sub state machine CANSM_BSM_WUVALIDATION 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 [ECUC_CanSM_00141]) the API request CanIf_SetControllerMode (ref. to [SWS_CanSM_91002]) with ControllerMode equal to CAN_CS_STOPPED, if the current CAN controller mode (ref. to [SWS_CanSM_00638]) is different.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.19.6 Guarding condition: G_CC_STOPPED_OK

[SWS_CanSM_00628] ⌈The guarding condition G_CC_STOPPED_OK of the sub state machine CANSM_BSM_WUVALIDATION shall be passed, if all API calls of [SWS_CanSM_00627] have returned E_OK.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.19.7 Trigger: T_CC_STOPPED_INDICATED

[SWS_CanSM_00629] ⌈If the CanSM module has got all mode indications (ref. to [SWS_CanSM_00396]) for the configured CAN controllers of the CAN network (ref. to [ECUC_CanSM_00141]) after the respective requests to stop the CAN controllers of the CAN network (ref. to [SWS_CanSM_00627]), this shall trigger the sub state machine CANSM_BSM_WUVALIDATION of the CAN network with T_CC_STOPPED_INDICATED.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.19.8 Trigger: T_CC_STOPPED_TIMEOUT

[SWS_CanSM_00630] ⌈After a timeout of CANSM_MODEREQ_REPEAT_TIME (ref. to [ECUC_CanSM_00336]) for all supposed controller stopped mode indications (ref. to [SWS_CanSM_00629]), this condition shall trigger the sub state machine CANSM_BSM_WUVALIDATION of the respective network with T_CC_STOPPED_TIMEOUT.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.19.9 State operation to do in: S_CC_STARTED

[SWS_CanSM_00631] ⌈As long the sub state machine CANSM_BSM_WUVALIDATION 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 [ECUC_CanSM_00141]) the API request CanIf_SetControllerMode with ControllerMode equal to CAN_CS_STARTED, if the current CAN controller mode (ref. to [SWS_CanSM_00638]) is different.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.19.10 Guarding condition: G_CC_STARTED_E_OK

[SWS_CanSM_00632] ⌈The guarding condition G_CC_STARTED_OK of the sub state machine CANSM_BSM_WUVALIDATION shall be passed, if all API calls of [SWS_CanSM_00631] have returned E_OK.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.19.11 Trigger: T_CC_STARTED_INDICATED

[SWS_CanSM_00633] ⌈If CanSM module has got all mode indications (ref. to [SWS_CanSM_00396]) for the configured CAN controllers of the CAN network (ref. to [ECUC_CanSM_00141]) after the respective requests to start the CAN controllers of the CAN network (ref. to [SWS_CanSM_00631]), this shall trigger the sub state machine CANSM_BSM_WUVALIDATION of the CAN network with T_CC_STARTED_INDICATED.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.19.12 Trigger: T_CC_STARTED_TIMEOUT

[SWS_CanSM_00634] ⌈After a timeout of CANSM_MODEREQ_REPEAT_TIME (ref. to [ECUC_CanSM_00336]) for all supposed controller started mode indications (ref. to[SWS_CanSM_00633]), this condition shall trigger the sub state machine CANSM_BSM_WUVALIDATION of the respective network with T_CC_STARTED_TIMEOUT.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.20 Sub state machine: CANSM_BSM_S_PRE_NOCOM

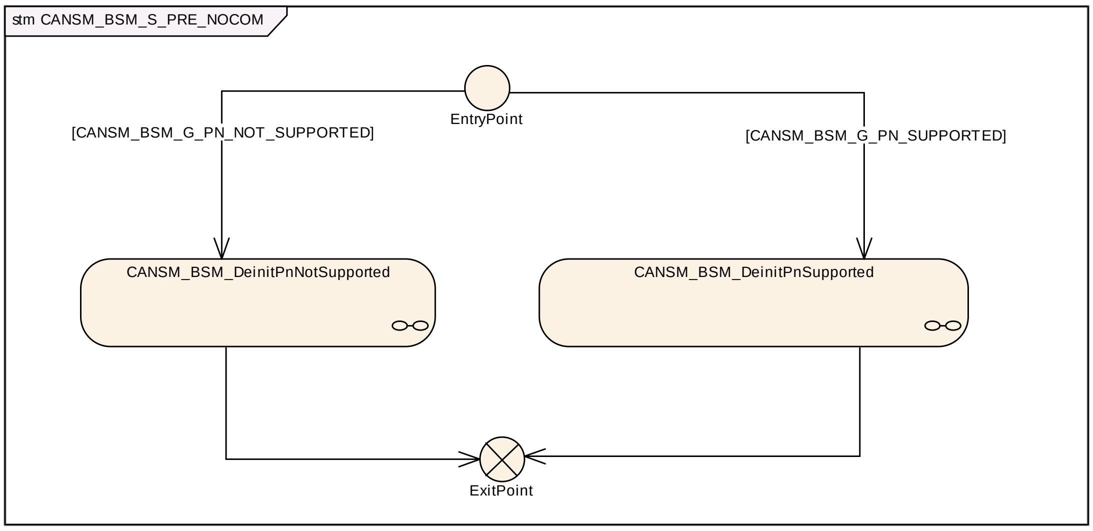
stm CANSM_BSM_S_PRE_NOCOM
EntryPoint
[CANSM_BSM_G_PN_NOT_SUPPORTED]
[CANSM_BSM_G_PN_SUPPORTED]
CANSM_BSM_DeinitPnNotSupported
CANSM_BSM_DeinitPnSupported
ExitPoint

Figure 7.3: CANSM_BSM_S_PRE_NOCOM, sub state machine of CANSM_BSM

7.2.20.1 Guarding condition: CANSM_BSM_G_PN_NOT_SUPPORTED

[SWS_CanSM_00436] ⌈The guarding condition `CANSM_BSM_G_PN_NOT_SUPPORTED` of the sub state machine `CANSM_BSM_S_PRE_NO_COM` shall evaluate, if the configuration parameter `CanTrcvPnEnabled` (ref. to [11, ECUC_CanTrcv_00172]) is `FALSE`, which is available via the reference `CanSMTransceiverId` (ref. to [ECUC_CanSM_00137]) or if no `CanSMTransceiverId` is configured at all.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.20.2 Guarding condition: CANSM_BSM_G_PN_SUPPORTED

[SWS_CanSM_00437] ⌈The guarding condition `CANSM_BSM_G_PN_SUPPORTED` of the sub state machine `CANSM_BSM_S_PRE_NO_COM` shall evaluate, if a `CanSMTransceiverId` (ref. to [ECUC_CanSM_00137]) is configured and if the configuration parameter `CanTrcvPnEnabled` (ref. to [11, ECUC_CanTrcv_00172]) is TRUE, which is available via the reference `CanSMTransceiverId` (ref. to [ECUC_CanSM_00137]).⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.20.3 Sub state machine: CANSM_BSM_DeinitPnSupported

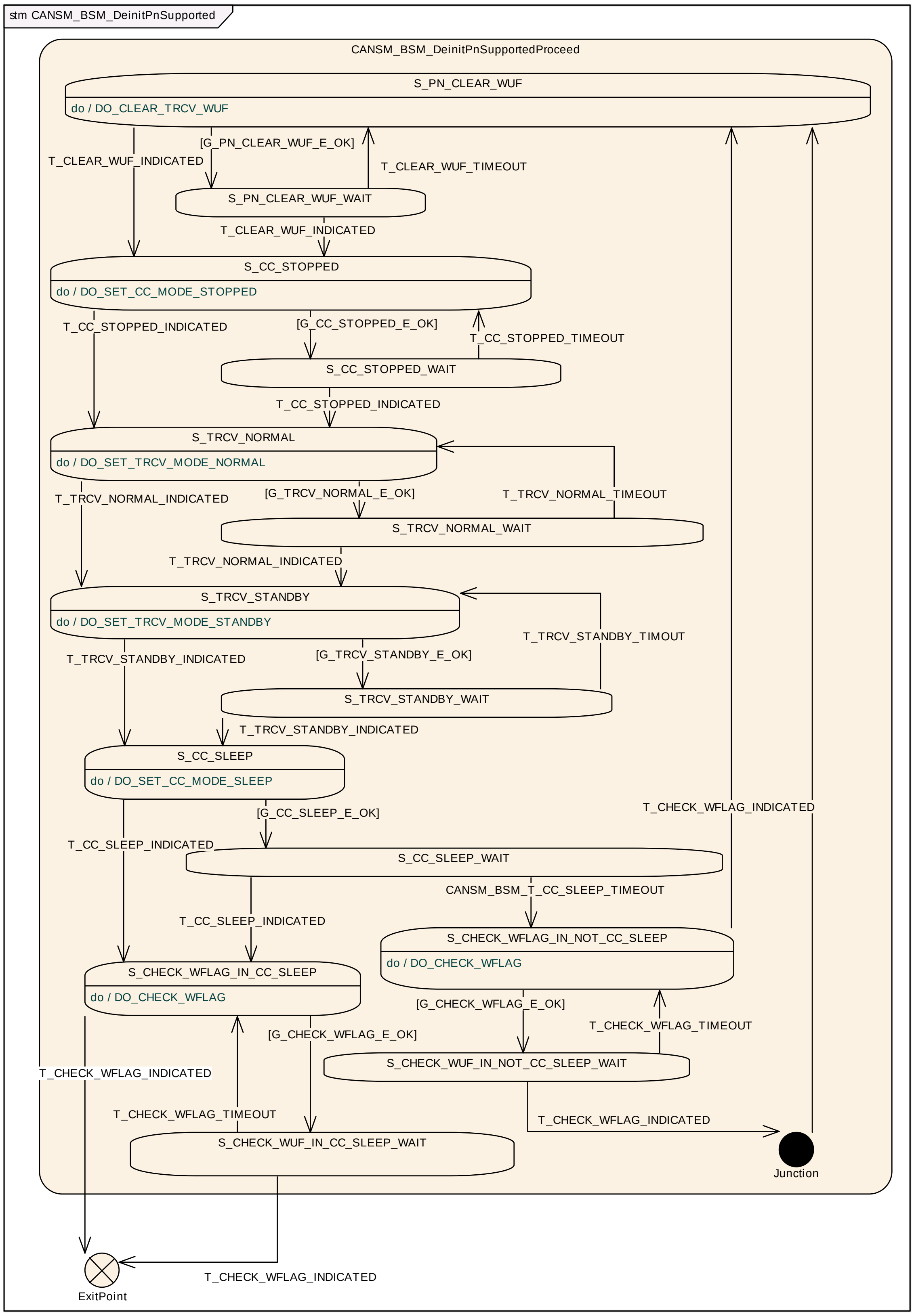

Figure 7.4: CANSM_BSM_DeinitPnSupported, sub state machine of CANSM_BSM_S_PRE_NOCOM

7.2.20.3.1 State operation to do in: S_PN_CLEAR_WUF

[SWS_CanSM_00438] ⌈As long the sub state machine CANSM_BSM_DeinitPnSupported 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_ClearTrcvWufFlag and use the configured Transceiver (ref. to [ECUC_CanSM_00137]) as API function parameter.⌋*(SRS_Can_01142, SRS_Can_-01145)*

7.2.20.3.2 Guarding condition: G_PN_CLEAR_WUF_E_OK

[SWS_CanSM_00439] ⌈The guarding condition G_PN_CLEAR_WUF_E_OK of the sub state machine CANSM_BSM_DeinitPnSupported shall be passed, if the API call of [SWS_CanSM_00438] has returned E_OK.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.20.3.3 Trigger: T_CLEAR_WUF_INDICATED

[SWS_CanSM_00440] ⌈The callback function CanSM_ClearTrcvWuf-FlagIndication (ref. to [SWS_CanSM_00413]) shall trigger the sub state machine CANSM_BSM_DeinitPnSupported 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 [ECUC_CanSM_00137]) of the CAN network.⌋*(SRS_Can_01142, SRS_Can_-01145)*

7.2.20.3.4 Trigger: T_CLEAR_WUF_TIMEOUT

[SWS_CanSM_00443] ⌈After a timeout of CANSM_MODEREQ_REPEAT_TIME (ref. to [ECUC_CanSM_00336]) for the callback function CanSM_ClearTrcvWufFlagIndi-cation (ref. to [SWS_CanSM_00440]), this condition shall trigger the sub state machine CANSM_BSM_DeinitPnSupported of the respective network with T_CLEAR_WUF_TIMEOUT.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.20.3.5 State operation to do in: S_CC_STOPPED

[SWS_CanSM_00441] ⌈As long the sub state machine CANSM_BSM_DeinitPnSupported 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 [ECUC_CanSM_00141]) the API request CanIf_SetControllerMode with ControllerMode equal to CAN_CS_STOPPED, if the current CAN controller mode (ref. to [SWS_CanSM_00638]) is different.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.20.3.6 Guarding condition: G_CC_STOPPED_E_OK

[SWS_CanSM_00442] ⌈The guarding condition `G_CC_STOPPED_E_OK` of the sub state machine `CANSM_BSM_DeinitPnSupported` shall be passed, if all API calls of [SWS_CanSM_00441] have returned `E_OK`.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.20.3.7 Trigger: T_CC_STOPPED_INDICATED

[SWS_CanSM_00444] ⌈If CanSM module has got all mode indications (ref. to [SWS_CanSM_00396]) for the configured CAN controllers of the CAN network (ref. to [ECUC_CanSM_00141]) after the respective requests to stop the CAN controllers of the CAN network (ref. to [SWS_CanSM_00442]), this shall trigger the sub state machine `CANSM_BSM_DeinitPnSupported` of the CAN network with `T_CC_STOPPED_INDICATED`.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.20.3.8 Trigger: T_CC_STOPPED_TIMEOUT

[SWS_CanSM_00445] ⌈After a timeout of `CANSM_MODEREQ_REPEAT_TIME` (ref. to [ECUC_CanSM_00336]) for all supposed controller stopped mode indications (ref. to [SWS_CanSM_00444]), this condition shall trigger the sub state machine `CANSM_BSM_DeinitPnSupported` of the respective network with `T_CC_STOPPED_TIMEOUT`.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.20.3.9 State operation to do in: S_TRCV_NORMAL

[SWS_CanSM_00446] ⌈As long the sub state machine `CANSM_BSM_DeinitPnSupported` 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 [ECUC_CanSM_00137]) the API request `CanIf_SetTrcvMode` (ref. to [SWS_CanSM_91002]) with `TransceiverMode` equal to `CANTRCV_TRCVMODE_NORMAL`.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.20.3.10 Guarding condition: G_TRCV_NORMAL_E_OK

[SWS_CanSM_00447] ⌈The guarding condition `G_TRCV_NORMAL_E_OK` of the sub state machine `CANSM_BSM_DeinitPnSupported` shall be passed, if the API call of [SWS_CanSM_00446] has returned `E_OK`.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.20.3.11 Trigger: T_TRCV_NORMAL_INDICATED

[SWS_CanSM_00448] ⌈If CanSM module has got the `CANTRCV_TRCVMODE_NORMAL` mode indication (ref. to [SWS_CanSM_00399]) for the configured CAN Transceiver of the CAN network (ref. to [ECUC_CanSM_00137]) after the respective request (ref. to [SWS_CanSM_00446]), this shall trigger the sub state machine `CANSM_BSM_DeinitPnSupported` of the CAN network with `T_TRCV_NORMAL_INDICATED`.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.20.3.12 Trigger: T_TRCV_NORMAL_TIMEOUT

[SWS_CanSM_00449] ⌈After a timeout of `CANSM_MODEREQ_REPEAT_TIME` (ref. to [ECUC_CanSM_00336]) for the supposed transceiver normal indication (ref. to [SWS_CanSM_00448]), this condition shall trigger the sub state machine `CANSM_BSM_DeinitPnSupported` of the respective network with `T_TRCV_NORMAL_TIMEOUT`.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.20.3.13 State operation to do in: S_TRCV_STANDBY

[SWS_CanSM_00450] ⌈As long the sub state machine `CANSM_BSM_DeinitPnSupported` 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 [ECUC_CanSM_00137]) the API request `CanIf_SetTrcvMode` (ref. to [SWS_CanSM_91002]) with `TransceiverMode` equal to `CANTRCV_TRCVMODE_STANDBY`.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.20.3.14 Guarding condition: G_TRCV_STANDBY_E_OK

[SWS_CanSM_00451] ⌈The guarding condition `G_TRCV_STANDBY_E_OK` of the sub state machine `CANSM_BSM_DeinitPnSupported` shall be passed, if the API call of [SWS_CanSM_00450] has returned `E_OK`.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.20.3.15 Trigger: T_TRCV_STANDBY_INDICATED

[SWS_CanSM_00452] ⌈If the CanSM module has got the `CANTRCV_TRCVMODE_STANDBY` mode indication (ref. to [SWS_CanSM_00399]) for the configured CAN Transceiver of the CAN network (ref. to [ECUC_CanSM_00137]) after the respective request (ref. to [SWS_CanSM_00450]), this shall trigger the sub state machine `CANSM_BSM_DeinitPnSupported` of the CAN network with `T_TRCV_STANDBY_INDICATED`.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.20.3.16 Trigger: T_TRCV_STANDBY_TIMEOUT

[SWS_CanSM_00454] ⌈After a timeout of CANSM_MODEREQ_REPEAT_TIME (ref. to [ECUC_CanSM_00336]) for the supposed transceiver standby indication (ref. to [SWS_CanSM_00452]), this condition shall trigger the sub state machine CANSM_BSM_DeinitPnSupported of the respective network with T_TRCV_STANDBY_TIMEOUT.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.20.3.17 State operation to do in: S_CC_SLEEP

[SWS_CanSM_00453] ⌈As long the sub state machine CANSM_BSM_DeinitPnSupported 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 [ECUC_CanSM_00141]) the API request CanIf_SetControllerMode (ref. to [SWS_CanSM_91002]) with ControllerMode equal to CAN_CS_SLEEP, if the current CAN controller mode (ref. to [SWS_CanSM_00638]) is different.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.20.3.18 Guarding condition: G_CC_SLEEP_E_OK

[SWS_CanSM_00455] ⌈The guarding condition G_CC_SLEEP_E_OK of the sub state machine CANSM_BSM_DeinitPnSupported shall be passed, if all API calls of [SWS_CanSM_00453] have returned E_OK.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.20.3.19 Trigger: T_CC_SLEEP_INDICATED

[SWS_CanSM_00456] ⌈If CanSM module has got all mode indications (ref. to [SWS_CanSM_00396]) for the configured CAN controllers of the CAN network (ref. to [ECUC_CanSM_00141]) after the respective requests to set the CAN controllers of the CAN network to sleep mode (ref. to [SWS_CanSM_00453]), this shall trigger the sub state machine CANSM_BSM_DeinitPnSupported of the CAN network with T_CC_SLEEP_INDICATED.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.20.3.20 Trigger: CANSM_BSM_T_CC_SLEEP_TIMEOUT

[SWS_CanSM_00457] ⌈After a timeout of CANSM_MODEREQ_REPEAT_TIME (ref. to [ECUC_CanSM_00336]) for all supposed controller sleep mode indications (ref. to [SWS_CanSM_00456]), this condition shall trigger the sub state machine CANSM_BSM_DeinitPnSupported (ref. to Figure 7-4Figure 7-4) of the respective network with CANSM_BSM_T_CC_SLEEP_TIMEOUT.⌋*(SRS_Can_01142, SRS_Can_-01145)*

7.2.20.3.21 State operation to do in: S_CHECK_WFLAG_IN_CC_SLEEP

[SWS_CanSM_00458] ⌈As long the sub state machine CANSM_BSM_DeinitPnSupported 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. [SWS_CanSM_91002]) and use the configured CAN Transceiver of the related Network (ref. to [ECUC_CanSM_00137]) as Transceiver parameter.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.20.3.22 Guarding condition: G_CHECK_WFLAG_E_OK

[SWS_CanSM_00459] ⌈The guarding condition G_CHECK_WFLAG_E_OK of the sub state machine CANSM_BSM_DeinitPnSupported shall be passed, if the API call of [SWS_CanSM_00458] or [SWS_CanSM_00462] has returned E_OK.⌋*(SRS_Can_-01142, SRS_Can_01145)*

7.2.20.3.23 Trigger: T_CHECK_WFLAG_INDICATED

[SWS_CanSM_00460] ⌈The callback function CanSM_CheckTransceiver-WakeFlagIndication (ref. to [SWS_CanSM_00416]) shall trigger the sub state machine CANSM_BSM_DeinitPnSupported 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 [ECUC_CanSM_00137]) of the CAN network.⌋*(SRS_Can_-01142, SRS_Can_01145)*

7.2.20.3.24 Trigger: T_CHECK_WFLAG_TIMEOUT

[SWS_CanSM_00461] ⌈After a timeout of CANSM_MODEREQ_REPEAT_TIME (ref. to [ECUC_CanSM_00336]) for the callback function CanSM_CheckTransceiverWake-FlagIndication (ref. to [SWS_CanSM_00460]), this condition shall trigger the sub state machine CANSM_BSM_DeinitPnSupported of the respective network with T_CHECK_WFLAG_TIMEOUT.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.20.3.25 State operation to do in: S_CHECK_WFLAG_IN_NOT_CC_SLEEP

[SWS_CanSM_00462] ⌈As long the sub state machine CANSM_BSM_DeinitPnSupported 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_CheckTrcvWake-Flag (ref. [SWS_CanSM_91002]) and use the configured CAN Transceiver of the related Network (ref. to [ECUC_CanSM_00137]) as `Transceiver` parameter.⌋ *(SRS_Can_01142, SRS_Can_01145)*

7.2.20.4 Sub state machine: CANSM_BSM_DeinitPnNotSupported

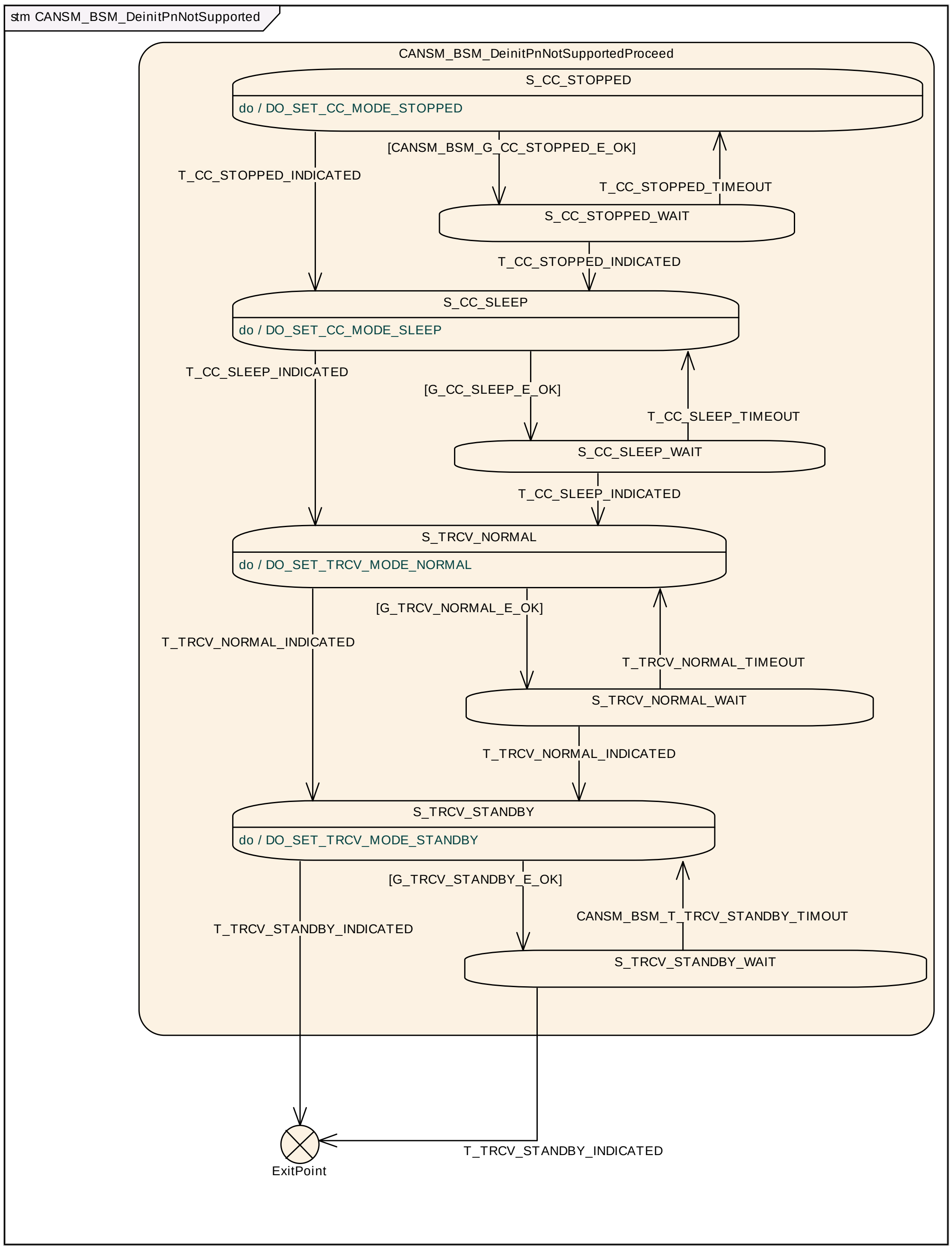

Figure 7.5: CANSM_BSM_DeinitPnNotSupported, sub state machine of CANSM_BSM_S_PRE_NOCOM

7.2.20.4.1 State operation to do in: S_CC_STOPPED

[SWS_CanSM_00464] ⌈As long the sub state machine CANSM_BSM_DeinitPnNotSupported 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 [ECUC_CanSM_00141]) the API request CanIf_SetControllerMode (ref. to [SWS_CanSM_91002]) with ControllerMode equal to CAN_CS_STOPPED, if the current CAN controller mode (ref. to [SWS_CanSM_00638]) is different.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.20.4.2 Guarding condition: CANSM_BSM_G_CC_STOPPED_OK

[SWS_CanSM_00465] ⌈The guarding condition CANSM_BSM__CC_STOPPED_OK of the sub state machine CANSM_BSM_DeinitPnNotSupported shall be passed, if all API calls of [SWS_CanSM_00464] have returned E_OK.⌋*(SRS_Can_01142, SRS_-Can_01145)*

7.2.20.4.3 Trigger: T_CC_STOPPED_INDICATED

[SWS_CanSM_00466] ⌈If CanSM module has got all mode indications (ref. to [SWS_CanSM_00396]) for the configured CAN controllers of the CAN network (ref. to [ECUC_CanSM_00141]) after the respective requests to stop the CAN controllers of the CAN network (ref. to [SWS_CanSM_00464]), this shall trigger the sub state machine CANSM_BSM_DeinitPnNotSupported of the CAN network with T_CC_STOPPED_INDICATED.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.20.4.4 Trigger: T_CC_STOPPED_TIMEOUT

[SWS_CanSM_00467] ⌈After a timeout of CANSM_MODEREQ_REPEAT_TIME (ref. to [ECUC_CanSM_00336]) for all supposed controller stopped mode indications (ref. to [SWS_CanSM_00466]), this condition shall trigger the sub state machine CANSM_BSM_DeinitPnNotSupported of the respective network with T_CC_STOPPED_TIMEOUT.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.20.4.5 State operation to do in: S_CC_SLEEP

[SWS_CanSM_00468] ⌈As long the sub state machine CANSM_BSM_DeinitPnNotSupported 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 [ECUC_CanSM_00141]) the API request CanIf_SetControllerMode (ref. to [SWS_CanSM_91002]) with

ControllerMode equal to CAN_CS_SLEEP, if the current CAN controller mode (ref. to [SWS_CanSM_00638]) is different.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.20.4.6 Guarding condition: G_CC_SLEEP_E_OK

[SWS_CanSM_00469] ⌈The guarding condition G_CC_SLEEP_E_OK of the sub state machine CANSM_BSM_DeinitPnNotSupported shall be passed, if all API calls of [SWS_CanSM_00468] have returned E_OK.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.20.4.7 Trigger: T_CC_SLEEP_INDICATED

[SWS_CanSM_00470] ⌈If CanSM module has got all mode indications (ref. to [SWS_CanSM_00396]) for the configured CAN controllers of the CAN network (ref. to [ECUC_CanSM_00141]) after the respective requests to set the CAN controllers of the CAN network to sleep mode (ref. to [SWS_CanSM_00468]), this shall trigger the sub state machine CANSM_BSM_DeinitPnNotSupported of the CAN network with T_CC_SLEEP_INDICATED.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.20.4.8 Trigger: T_CC_SLEEP_TIMEOUT

[SWS_CanSM_00471] ⌈After a timeout of CANSM_MODEREQ_REPEAT_TIME (ref. to [ECUC_CanSM_00336]) for all supposed controller sleep mode indications (ref. to [SWS_CanSM_00470]), this condition shall trigger the sub state machine CANSM_BSM_DeinitPnNotSupported of the respective network with T_CC_SLEEP_TIMEOUT.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.20.4.9 State operation to do in: S_TRCV_NORMAL

[SWS_CanSM_00472] ⌈If for the CAN network a CAN Transceiver is configured (ref. to [ECUC_CanSM_00137]), then as long the sub state machine CANSM_BSM_DeinitPnNotSupported 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 [ECUC_CanSM_00137]) the API request CanIf_SetTrcv-Mode (ref. to [SWS_CanSM_91002]) with TransceiverMode equal to CANTRCV_TRCVMODE_NORMAL.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.20.4.10 Guarding condition: G_TRCV_NORMAL_E_OK

[SWS_CanSM_00473] ⌈The guarding condition G_TRCV_NORMAL_E_OK of the sub state machine CANSM_BSM_DeinitPnNotSupported shall be passed, if the API call of [SWS_CanSM_00472] has returned E_OK.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.20.4.11 Trigger: T_TRCV_NORMAL_INDICATED

[SWS_CanSM_00474] ⌈If CanSM module has got the CANTRCV_TRCVMODE_NORMAL mode indication (ref. to [SWS_CanSM_00399]) for the configured CAN Transceiver of the CAN network (ref. to [ECUC_CanSM_00137]) after the respective request (ref. to [SWS_CanSM_00472]), this shall trigger the sub state machine CANSM_BSM_DeinitPnNotSupported of the CAN network with T_TRCV_NORMAL_INDICATED.⌋*(SRS_Can_01142, SRS_Can_01145)*

[SWS_CanSM_00556] ⌈If no CAN Transceiver is configured for the CAN network, then this shall trigger the sub state machine CANSM_BSM_DeinitPnNotSupported of the CAN network in the state S_TRCV_NORMAL with T_TRCV_NORMAL_INDICATED.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.20.4.12 Trigger: T_TRCV_NORMAL_TIMEOUT

[SWS_CanSM_00475] ⌈After a timeout of CANSM_MODEREQ_REPEAT_TIME (ref. to [ECUC_CanSM_00336]) for the supposed transceiver normal indication (ref. to [SWS_CanSM_00474]), this condition shall trigger the sub state machine CANSM_BSM_DeinitPnNotSupported of the respective network with T_TRCV_NORMAL_TIMEOUT.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.20.4.13 State operation to do in: S_TRCV_STANDBY

[SWS_CanSM_00476] ⌈If for the CAN network a CAN Transceiver is configured (ref. to [ECUC_CanSM_00137]), then as long the sub state machine CANSM_BSM_DeinitPnNotSupported 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 [ECUC_CanSM_00137]) the API request CanIf_SetTrcvMode (ref. to [SWS_CanSM_91002]) with TransceiverMode equal to CANTRCV_TRCVMODE_STANDBY.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.20.4.14 Guarding condition: G_TRCV_STANDBY_E_OK

[SWS_CanSM_00477] ⌈The guarding condition G_TRCV_STANDBY_E_OK of the sub state machine CANSM_BSM_DeinitPnNotSupported shall be passed, if the API call of [SWS_CanSM_00476] has returned E_OK.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.20.4.15 Trigger: T_TRCV_STANDBY_INDICATED

[SWS_CanSM_00478] ⌈If CanSM module has got the CANTRCV_TRCVMODE_STANDBY mode indication (ref. to [SWS_CanSM_00399]) for the configured CAN Transceiver of the CAN network (ref. to [ECUC_CanSM_00137]) after the respective request (ref. to [SWS_CanSM_00476]), this shall trigger the sub state machine CANSM_BSM_DeinitPnNotSupported of the CAN network with T_TRCV_STANDBY_INDICATED.⌋*(SRS_Can_01142, SRS_Can_01145)*

[SWS_CanSM_00557] ⌈If no CAN Transceiver is configured for the CAN network (ref. to [ECUC_CanSM_00137]), then this shall trigger the sub state machine CANSM_BSM_DeinitPnNotSupported of the CAN network in the state S_TRCV_STANDBY with T_TRCV_STANDBY_INDICATED.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.20.4.16 Trigger: CANSM_BSM_T_TRCV_STANDBY_TIMEOUT

[SWS_CanSM_00479] ⌈After a timeout of CANSM_MODEREQ_REPEAT_TIME (ref. to [ECUC_CanSM_00336]) for the supposed transceiver standby indication (ref. to [SWS_CanSM_00478]), this condition shall trigger the sub state machine CANSM_BSM_DeinitPnNotSupported of the respective network with CANSM_BSM_T_TRCV_STANDBY_TIMEOUT.⌋*(SRS_Can_01142, SRS_Can_01145)*

7.2.21 Sub state machine: CANSM_BSM_S_SILENTCOM_BOR

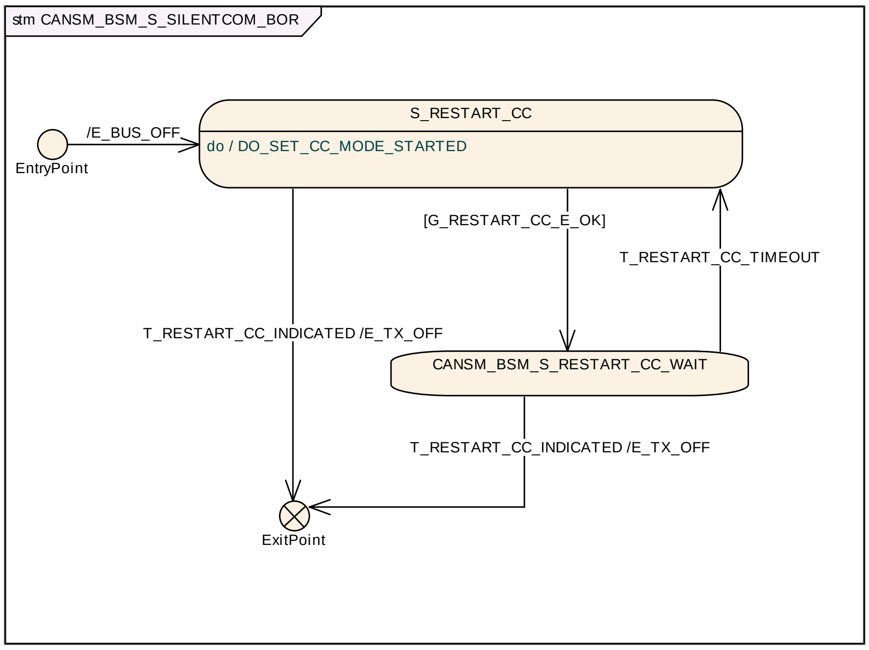
stm CANSM_BSM_S_SILENTCOM_BOR
EntryPoint
/E_BUS_OFF
S_RESTART_CC
do / DO_SET_CC_MODE_STARTED
[G_RESTART_CC_E_OK]
T_RESTART_CC_TIMEOUT
T_RESTART_CC_INDICATED /E_TX_OFF
CANSM_BSM_S_RESTART_CC_WAIT
T_RESTART_CC_INDICATED /E_TX_OFF
ExitPoint

Figure 7.6: CANSM_BSM_S_SILENTCOM_BOR, sub state machine of CANSM_BSM

7.2.21.1 Effect: E_BUS_OFF

[SWS_CanSM_00605] ⌈The effect `E_BUS_OFF` of the sub state machine `CANSM_BSM_S_FULLCOM` `CANSM_BSM_S_SILENTCOM_BOR` shall invocate `Dem_SetEventStatus` (ref. to [SWS_CanSM_91002]) with the parameters `EventId` := `CANSM_E_BUS_OFF` (ref. to [ECUC_CanSM_00070]) and `EventStatus` := `DEM_EVENT_STATUS_PRE_FAILED`.⌋*(SRS_BSW_00422)*

7.2.21.2 State operation: S_RESTART_CC

[SWS_CanSM_00604] ⌈As long the sub state machine `CANSM_BSM_S_SILENTCOM_BOR` 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 [ECUC_CanSM_00141]) the API request `CanIf_SetControllerMode` (ref. to [SWS_CanSM_91002]) with `ControllerMode` equal to `CAN_CS_STARTED`, if the current CAN controller mode (ref. to [SWS_CanSM_00638]) is different.⌋*(SRS_Can_01142, SRS_Can_01145, SRS_Can_01144, SRS_Can_01146)*

7.2.21.3 G_RESTART_CC_E_OK

[SWS_CanSM_00603] ⌈The guarding condition G_RESTART_CC_OK of the sub state machine CANSM_BSM_S_SILENTCOM_BOR shall be passed, if all API calls of [SWS_CanSM_00604] have returned E_OK.⌋*(SRS_Can_01142, SRS_Can_01145, SRS_Can_01144, SRS_Can_01146)*

7.2.21.4 Trigger: T_RESTART_CC_INDICATED

[SWS_CanSM_00600] ⌈If CanSM module has got all mode indications (ref. to [SWS_CanSM_00396]) for the configured CAN controllers of the CAN network (ref. to [ECUC_CanSM_00141]) after the respective requests to start the CAN controllers of the CAN network (ref. to [SWS_CanSM_00604]), this shall trigger the sub state CANSM_BSM_S_SILENTCOM_BOR of the CAN network with T_RESTART_CC_INDICATED.⌋*(SRS_Can_01142, SRS_Can_01145, SRS_Can_01144, SRS_Can_01146)*

7.2.21.5 T_RESTART_CC_TIMEOUT

[SWS_CanSM_00602] ⌈After a timeout of CANSM_MODEREQ_REPEAT_TIME (ref. to [ECUC_CanSM_00336]) for all supposed controller started mode indications (ref. to [SWS_CanSM_00600]), this condition shall trigger the sub state machine CANSM_BSM_S_SILENTCOM_BOR of the respective network with T_RESTART_CC_TIMEOUT.⌋*(SRS_Can_01142, SRS_Can_01145, SRS_Can_01144, SRS_Can_01146)*

7.2.21.6 Effect: E_TX_OFF

The effect E_TX_OFF shall do nothing (default PDU mode after restart of CAN controller is already TX OFF, ref. to CanIf SWS).

7.2.22 Sub state machine: CANSM_BSM_S_PRE_FULLCOM

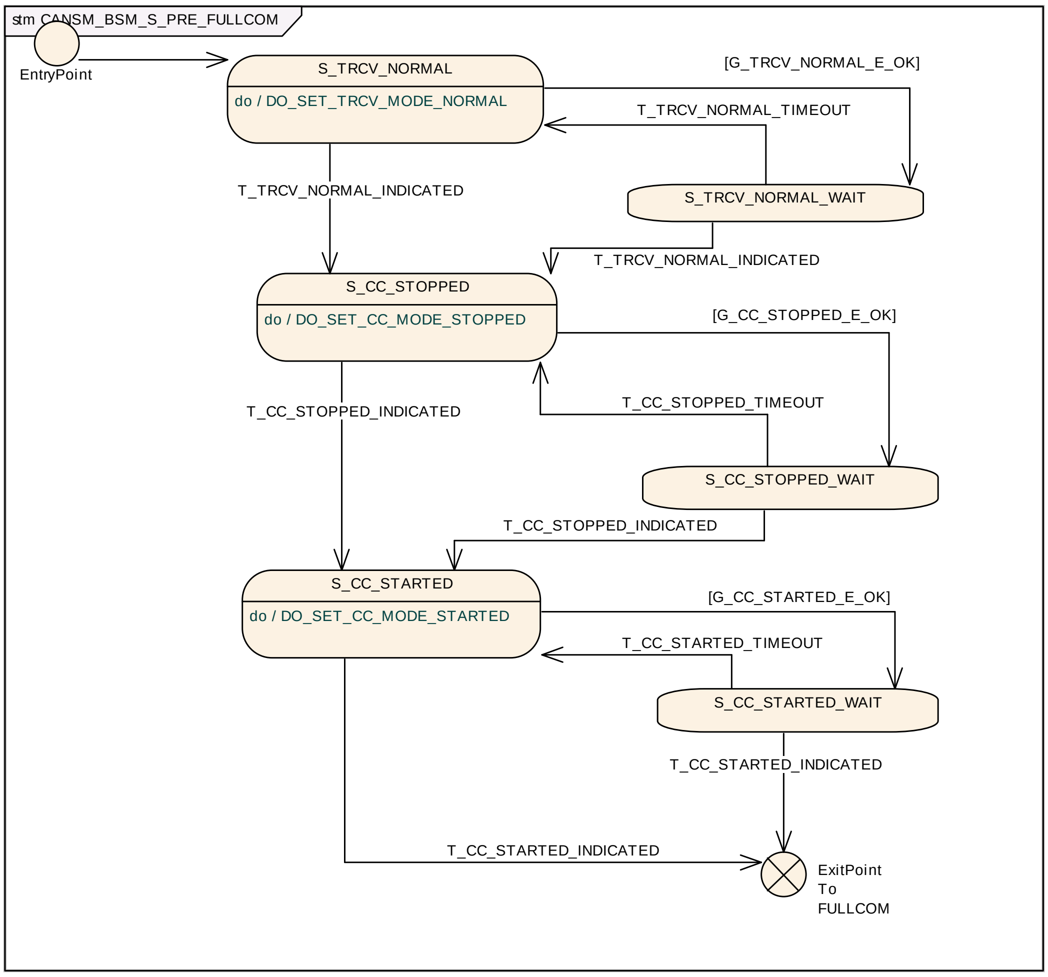

Figure 7.7: CANSM_BSM_S_PRE_FULLCOM, sub state machine of CANSM_BSM

7.2.22.1 State operation to do in: S_TRCV_NORMAL

[SWS_CanSM_00483] ⌈If for the CAN network a CAN Transceiver is configured (ref. to [ECUC_CanSM_00137]), then as long the sub state machine `CANSM_BSM_S_PRE_FULLCOM` 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 [ECUC_CanSM_00137]) the API request `CanIf_SetTrcvMode` (ref. to [SWS_CanSM_91002]) with `TransceiverMode` equal to `CANTRCV_TRCVMODE_NORMAL`.⌋*(SRS_Can_01145, SRS_Can_01142)*

7.2.22.2 Guarding condition: G_TRCV_NORMAL_E_OK

[SWS_CanSM_00484] ⌈The guarding condition G_TRCV_NORMAL_E_OK of the sub state machine CANSM_BSM_S_PRE_FULLCOM shall be passed, if the API call of [SWS_CanSM_00483] has returned E_OK.⌋*(SRS_Can_01145, SRS_Can_01142)*

7.2.22.3 Trigger: T_TRCV_NORMAL_INDICATED

[SWS_CanSM_00485] ⌈If CanSM module has got the CANTRCV_TRCVMODE_NORMAL mode indication (ref. to [SWS_CanSM_00399]) for the configured CAN Transceiver of the CAN network (ref. to [ECUC_CanSM_00137]) after the respective request (ref. to [SWS_CanSM_00483]), this shall trigger the sub state machine CANSM_BSM_S_PRE_FULLCOM of the CAN network with T_TRCV_NORMAL_INDICATED.⌋*(SRS_Can_01145, SRS_Can_01142)*

[SWS_CanSM_00558] ⌈If no CAN Transceiver is configured for the CAN network (ref. to [ECUC_CanSM_00137]), then this shall trigger the sub state machine CANSM_BSM_S_PRE_FULLCOM of the CAN network in the state S_TRCV_NORMAL with T_TRCV_NORMAL_INDICATED.⌋*(SRS_Can_01145, SRS_Can_01142)*

7.2.22.4 Trigger: T_TRCV_NORMAL_TIMEOUT

[SWS_CanSM_00486] ⌈After a timeout of CANSM_MODEREQ_REPEAT_TIME (ref. to [ECUC_CanSM_00336]) for the supposed transceiver normal indication (ref. to [SWS_CanSM_00485]), this condition shall trigger the sub state machine CANSM_BSM_S_PRE_FULLCOM of the respective network with T_TRCV_NORMAL_TIMEOUT.⌋*(SRS_Can_01145, SRS_Can_01142)*

7.2.22.5 State operation to do in: S_CC_STOPPED

[SWS_CanSM_00487] ⌈As long the sub state machine CANSM_BSM_S_PRE_FULLCOM 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 [ECUC_CanSM_00141]) the API request CanIf_SetControllerMode (ref. to [SWS_CanSM_91002]) with ControllerMode equal to CAN_CS_STOPPED, if the current CAN controller mode (ref. to [SWS_CanSM_00638]) is different.⌋*(SRS_Can_01145, SRS_Can_01142)*

7.2.22.6 Guarding condition: G_CC_STOPPED_OK

[SWS_CanSM_00488] ⌈The guarding condition G_CC_STOPPED_OK of the sub state machine CANSM_BSM_S_PRE_FULLCOM shall be passed, if all API calls of [SWS_CanSM_00487] have returned E_OK.⌋*(SRS_Can_01145, SRS_Can_01142)*

7.2.22.7 Trigger: T_CC_STOPPED_INDICATED

[SWS_CanSM_00489] ⌈If CanSM module has got all mode indications (ref. to [SWS_CanSM_00396]) for the configured CAN controllers of the CAN network (ref. to [ECUC_CanSM_00141]) after the respective requests to stop the CAN controllers of the CAN network (ref. to [SWS_CanSM_00487]), this shall trigger the sub state machine CANSM_BSM_S_PRE_FULLCOM of the CAN network with T_CC_STOPPED_INDICATED.⌋*(SRS_Can_01145, SRS_Can_01142)*

7.2.22.8 Trigger: T_CC_STOPPED_TIMEOUT

[SWS_CanSM_00490] ⌈After a timeout of CANSM_MODEREQ_REPEAT_TIME (ref. to [ECUC_CanSM_00336]) for all supposed controller stopped mode indications (ref. to [SWS_CanSM_00489]), this condition shall trigger the sub state machine CANSM_BSM_S_PRE_FULLCOM of the respective network with T_CC_STOPPED_TIMEOUT.⌋*(SRS_Can_01145, SRS_Can_01142)*

7.2.22.9 State operation to do in: S_CC_STARTED

[SWS_CanSM_00491] ⌈As long the sub state machine CANSM_BSM_S_PRE_FULLCOM 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 [ECUC_CanSM_00141]) the API request CanIf_SetControllerMode (ref. to [SWS_CanSM_91002]) with ControllerMode equal to CAN_CS_STARTED, if the current CAN controller mode (ref. to [SWS_CanSM_00638]) is different.⌋*(SRS_Can_01145, SRS_Can_01142)*

7.2.22.10 Guarding condition: G_CC_STARTED_OK

[SWS_CanSM_00492] ⌈The guarding condition G_CC_STARTED_OK of the sub state machine CANSM_BSM_S_PRE_FULLCOM shall be passed, if all API calls of [SWS_CanSM_00491] have returned E_OK.⌋*(SRS_Can_01145, SRS_Can_01142)*

7.2.22.11 Trigger: T_CC_STARTED_INDICATED

[SWS_CanSM_00493] ⌈If CanSM module has got all mode indications (ref. to [SWS_CanSM_00396]) for the configured CAN controllers of the CAN network (ref. to [ECUC_CanSM_00141]) after the respective requests to start the CAN controllers of the CAN network (ref. to [SWS_CanSM_00491]), this shall trigger the sub state machine CANSM_BSM_S_PRE_FULLCOM of the CAN network with T_CC_STARTED_INDICATED.⌋*(SRS_Can_01145, SRS_Can_01142)*

7.2.22.12 Trigger: T_CC_STARTED_TIMEOUT

[SWS_CanSM_00494] ⌈After a timeout of CANSM_MODEREQ_REPEAT_TIME (ref. to [ECUC_CanSM_00336]) for all supposed controller started mode indications (ref. to [SWS_CanSM_00493]), this condition shall trigger the sub state machine CANSM_BSM_S_PRE_FULLCOM of the respective network with T_CC_STARTED_TIMEOUT.⌋*(SRS_Can_01145, SRS_Can_01142)*

7.2.23 Sub state machine CANSM_BSM_S_FULLCOM

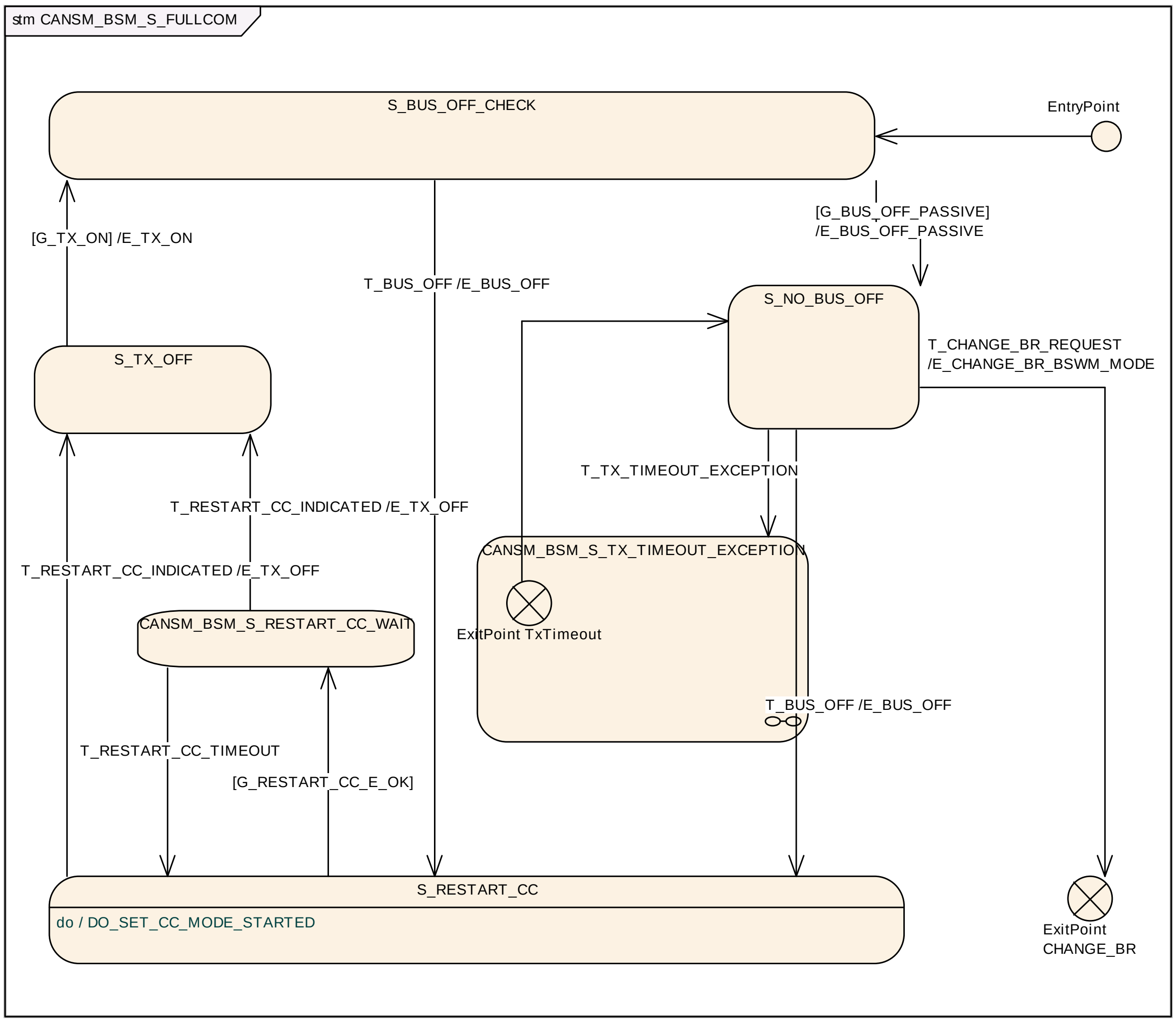

Figure 7.8: CANSM_BSM_S_FULLCOM, sub state machine of CANSM_BSM

7.2.23.1 Guarding condition: G_BUS_OFF_PASSIVE

[SWS_CanSM_00496] ⌈The guarding condition G_BUS_OFF_PASSIVE of the sub state machine CANSM_BSM_S_FULLCOM shall be passed, if CANSM_BOR_TX_CONFIRMATION_POLLING is disabled (ref. to [ECUC_CanSM_00339]) and the time duration since the effect E_TX_ON is greater or equal the configuration parameter CANSM_BOR_TIME_TX_ENSURED (ref. to [ECUC_CanSM_00130]).⌋*(SRS_Can_01145, SRS_Can_01142)*

[SWS_CanSM_00497] ⌈The guarding condition G_BUS_OFF_PASSIVE of the sub state machine CANSM_BSM_S_FULLCOM shall be passed, if CANSM_BOR_TX_CONFIRMATION_POLLING is enabled (ref. to [ECUC_CanSM_00339]) and the API CanIf_GetTxConfirmationState (ref. to [SWS_CanSM_91002]) returns CANIF_TX_RX_NOTIFICATION for all configured

CAN controllers of the CAN network (ref. to [ECUC_CanSM_00141]).⌋*(SRS_Can_-01145, SRS_Can_01142)*

7.2.23.2 Effect: E_BUS_OFF_PASSIVE

[SWS_CanSM_00498] ⌈The effect `E_BUS_OFF_PASSIVE` of the sub state machine `CANSM_BSM_S_FULLCOM` shall invocate `Dem_SetEventStatus` (ref. to [SWS_CanSM_91002]) with the parameters `EventId` := `CANSM_E_BUS_OFF` (ref. to [ECUC_CanSM_00070]) and `EventStatus` := `DEM_EVENT_STATUS_PASSED`.⌋*(SRS_BSW_00422)*

7.2.23.3 Trigger: T_CHANGE_BR_REQUEST

[SWS_CanSM_00507] ⌈If no condition is present to deny the `CanSM_SetBaudrate` request (ref. to [SWS_CanSM_00503]), this shall trigger the state machine `CANSM_BSM_S_FULLCOM` and respectively the parent state machine `CanSM_BSM` with `T_CHANGE_BR_REQUEST` (causes either a direct baud rate change if possible via `CanIf_SetBaudrate` (ref. to [SWS_CanSM_91003])) or the start of the required asynchronous process to do that⌋*(SRS_Can_01145, SRS_Can_01142)*

7.2.23.4 Effect: E_CHANGE_BR_BSWM_MODE

[SWS_CanSM_00528] ⌈The effect `E_CHANGE_BR_BSWM_MODE` of the sub state machine `CANSM_BSM_S_FULLCOM` shall call for the corresponding CAN network the API `BswM_CanSM_CurrentState` (ref. to [SWS_CanSM_91002]) with the parameters `Network` := `CanSMComMNetworkHandleRef` and `CurrentState` := `CANSM_BSWM_CHANGE_BAUDRATE`.⌋*(SRS_Can_01145, SRS_Can_01142)*

7.2.23.5 Trigger: T_BUS_OFF

[SWS_CanSM_00500] ⌈The callback function `CanSM_ControllerBusOff` (ref. to [SWS_CanSM_00064]) shall trigger the sub state machine `CANSM_BSM_S_FULLCOM` 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`.⌋*(SRS_Can_01145, SRS_Can_01142)*

[SWS_CanSM_00653] ⌈If more than one CAN controller belongs to one CAN network and for one of its controllers a bus-off is indicated with `CanSM_ControllerBusOff`, then the CanSM shall stop in context of the effect `E_BUS_OFF` the other CAN contoller(s) of the CAN network, too.⌋*(SRS_Can_01145, SRS_Can_01142)*

7.2.23.6 Effect: E_BUS_OFF

[SWS_CanSM_00508] ⌈The effect E_BUS_OFF of the sub state machine CANSM_BSM_S_FULLCOM shall call at 1st place for the corresponding CAN network the API BswM_CanSM_CurrentState (ref. to [SWS_CanSM_91002]) with the parameters Network := CanSMComMNetworkHandleRef and CurrentState := CANSM_BSWM_BUS_OFF.⌋*(SRS_Can_01145, SRS_Can_01142)*

[SWS_CanSM_00521] ⌈The effect E_BUS_OFF of the sub state machine CANSM_BSM_S_FULLCOM shall call at 2nd place for the corresponding CAN network the API ComM_BusSM_ModeIndication (ref. to [SWS_CanSM_91002]) with the parameters Channel := CanSMComMNetworkHandleRef (ref. to [ECUC_CanSM_00161]) and ComMode := COMM_SILENT_COMMUNICATION.⌋*(SRS_Can_01145, SRS_Can_-01142)*

[SWS_CanSM_00522] ⌈The effect E_BUS_OFF of the sub state machine CANSM_BSM_S_FULLCOM shall invocate Dem_SetEventStatus (ref. to [SWS_CanSM_91002]) with the parameters EventId := CANSM_E_BUS_OFF (ref. to [ECUC_CanSM_00070]) and EventStatus := DEM_EVENT_STATUS_PRE_FAILED.⌋*(SRS_BSW_00422)*

7.2.23.7 State operation to do in: S_RESTART_CC

[SWS_CanSM_00509] ⌈As long the sub state machine CANSM_BSM_S_FULLCOM 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 [ECUC_CanSM_00141]) the API request CanIf_Set-ControllerMode (ref. to [SWS_CanSM_91002]) with ControllerMode equal to CAN_CS_STARTED, if the current CAN controller mode (ref. to [SWS_CanSM_00638]) is different.⌋*(SRS_Can_01145, SRS_Can_01142)*

7.2.23.8 Guarding condition: G_RESTART_CC_OK

[SWS_CanSM_00510] ⌈The guarding condition G_RESTART_CC_OK of the sub state machine CANSM_BSM_S_FULLCOM shall be passed, if all API calls of [SWS_CanSM_00509] have returned E_OK.⌋*(SRS_Can_01145, SRS_Can_01142)*

7.2.23.9 Trigger: T_RESTART_CC_INDICATED

[SWS_CanSM_00511] ⌈If CanSM module has got all mode indications (ref. to [SWS_CanSM_00396]) for the configured CAN controllers of the CAN network (ref. to [ECUC_CanSM_00141]) after the respective requests to start the CAN controllers of the CAN network (ref. to [SWS_CanSM_00509]), this shall trigger the sub state

CANSM_BSM_S_FULLCOM of the CAN network with T_RESTART_CC_INDICATED.⌋ *(SRS_Can_01145, SRS_Can_01142)*

7.2.23.10 Trigger: T_RESTART_CC_TIMEOUT

[SWS_CanSM_00512] ⌈After a timeout of CANSM_MODEREQ_REPEAT_TIME (ref. to [ECUC_CanSM_00336]) for all supposed controller started mode indications (ref. to [SWS_CanSM_00511]), this condition shall trigger the sub state machine CANSM_BSM_S_FULLCOM of the respective network with T_RESTART_CC_TIMEOUT.⌋ *(SRS_Can_01145, SRS_Can_01142)*

7.2.23.11 Effect: E_TX_OFF

The effect E_TX_OFF shall do nothing.

7.2.23.12 Guarding condition: G_TX_ON

[SWS_CanSM_00514] ⌈If CanSMEnableBusOffDelay is FALSE, then guarding condition G_TX_ON of the sub state machine CANSM_BSM_S_FULLCOM shall be passed after a time duration of CanSMBorTimeL1 (ref. to [ECUC_CanSM_00128]) related to the last T_BUS_OFF, 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 [ECUC_CanSM_00131]).⌋ *(SRS_Can_01145, SRS_Can_01142)*

[SWS_CanSM_00515] ⌈If CanSMEnableBusOffDelay is FALSE, then the guarding condition G_TX_ON of the sub state machine CANSM_BSM_S_FULLCOM shall be passed after a time duration of CanSMBorTimeL2 (ref. to [ECUC_CanSM_00129]) related to the last T_BUS_OFF, 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 [ECUC_CanSM_00131]).⌋ *(SRS_Can_01145, SRS_Can_01142)*

[SWS_CanSM_00636] ⌈If CanSMEnableBusOffDelay is TRUE, then the guarding conditions of [SWS_CanSM_00514] and [SWS_CanSM_00515] shall be passed after the specified time duration in each case plus the additional random delay value, which shall be requested after the bus-off event with the configured call out function <User_GetBusOffDelay> (API name defined by CanSMGetBusOffDelayFunction).⌋ *(SRS_Can_01145, SRS_Can_01142)*

7.2.23.13 Effect: E_TX_ON

[SWS_CanSM_00516] ⌈If ECU passive is FALSE (ref. to [SWS_CanSM_00646]), then the effect E_TX_ON of the sub state machine CANSM_BSM_S_FULLCOM shall call at 1st place for the configured CAN controllers of the CAN network (ref. to [ECUC_CanSM_00141]) the API function CanIf_SetPduMode (ref. to [SWS_CanSM_91002]) with the parameters ControllerId := CanSMControllerId (ref. to [ECUC_CanSM_00141]) and PduModeRequest := CANIF_ONLINE.⌋ *(SRS_Can_01158)*

[SWS_CanSM_00648] ⌈If ECU passive is TRUE (ref. to [SWS_CanSM_00646]), then the effect E_TX_ON of the sub state machine CANSM_BSM_S_FULLCOM shall call at 1st place for the configured CAN controllers of the CAN network (ref. to [ECUC_CanSM_00141]) the API function CanIf_SetPduMode (ref. to [SWS_CanSM_91002]) with the parameters ControllerId := CanSMControllerId (ref. to [ECUC_CanSM_00141]) and PduModeRequest := CANIF_TX_OFFLINE_ACTIVE.⌋ *(SRS_Can_01158)*

[SWS_CanSM_00517] ⌈The effect E_TX_ON of the sub state machine CANSM_BSM_S_FULLCOM shall call at 2nd place for the corresponding CAN network the API BswM_CanSM_CurrentState (ref. to [SWS_CanSM_91002]) with the parameters Network := CanSMComMNetworkHandleRef and CurrentState := CANSM_BSWM_FULL_COMMUNICATION.⌋ *(SRS_Can_01145, SRS_Can_01142)*

[SWS_CanSM_00518] ⌈The effect E_TX_ON of the sub state machine CANSM_BSM_S_FULLCOM shall call at 3rd place the API ComM_BusSM_ModeIndication (ref. to [SWS_CanSM_91002]) with the parameters Channel := CanSMComMNetworkHandleRef (ref. to [ECUC_CanSM_00161]) and ComMode := COMM_FULL_COMMUNICATION.⌋ *(SRS_Can_01145, SRS_Can_01142)*

7.2.23.14 Trigger: T_TX_TIMEOUT_EXCEPTION

[SWS_CanSM_00584] ⌈The callback function CanSM_TxTimeoutException (ref. to [SWS_CanSM_00410]) shall trigger the sub state machine CANSM_BSM_S_FULLCOM with T_TX_TIMEOUT_EXCEPTION.⌋ *(SRS_Can_01145, SRS_Can_01142)*

7.2.23.15 Notes

In the state S_NO_BUS_OFF no state operation is required for the CanSM module.

7.2.23.16 Sub state machine: CANSM_BSM_S_TX_TIMEOUT_EXCEPTION

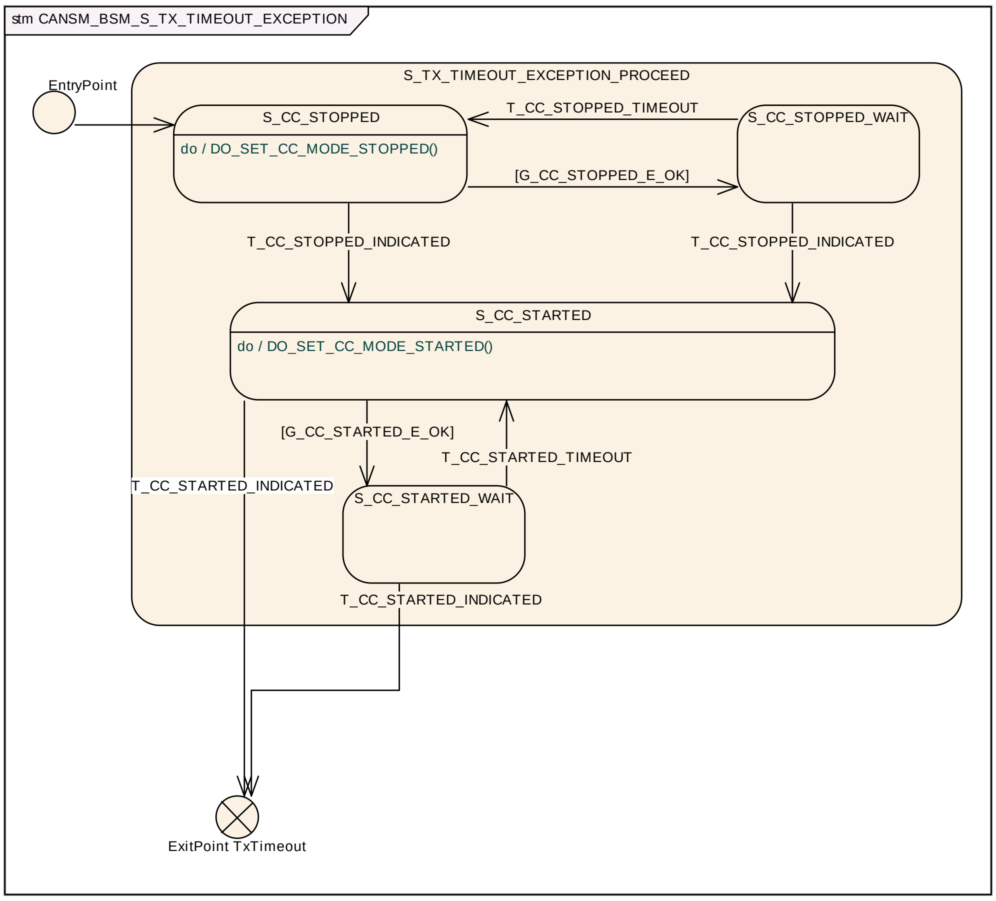

Figure 7.9: CANSM_BSM_S_TX_TIMEOUT_EXCEPTION, sub state machine of CANSM_BSM_S_FULLCOM

7.2.23.16.1 Trigger: T_CC_STOPPED_TIMEOUT

[SWS_CanSM_00576] ⌈After a timeout of CANSM_MODEREQ_REPEAT_TIME (ref. to [ECUC_CanSM_00336]) for all supposed controller stopped mode indications (ref. to [SWS_CanSM_00579]), this condition shall trigger the sub state machine CANSM_BSM_S_TX_TIMEOUT_EXCEPTION of the respective network with T_CC_STOPPED_TIMEOUT.⌋*(SRS_Can_01145, SRS_Can_01142)*

7.2.23.16.2 Guarding condition: G_CC_STOPPED_E_OK

[SWS_CanSM_00577] ⌈The guarding condition G_CC_STOPPED_E_OK of the sub state machine CANSM_BSM_S_TX_TIMEOUT_EXCEPTION shall be passed, if all API calls of [SWS_CanSM_00578] have returned E_OK.⌋*(SRS_Can_01145, SRS_Can_-01142)*

7.2.23.16.3 State operation: DO_SET_CC_MODE_STOPPED ()

[SWS_CanSM_00578] ⌈As long the sub state machine CANSM_BSM_S_TX_TIMEOUT_EXCEPTION 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 [ECUC_CanSM_00141]) the API request CanIf_SetControllerMode (ref. to [SWS_CanSM_91002]) with ControllerMode equal to CAN_CS_STOPPED, if the current CAN controller mode (ref. to [SWS_CanSM_00638]) is different.⌋*(SRS_Can_-01145, SRS_Can_01142)*

7.2.23.16.4 Trigger: T_CC_STOPPED_INDICATED

[SWS_CanSM_00579] ⌈If CanSM module has got all mode indications (ref. to [SWS_CanSM_00396]) for the configured CAN controllers of the CAN network (ref. to [ECUC_CanSM_00141]) after the respective requests to stop the CAN controllers of the CAN network (ref. to [SWS_CanSM_00524]), this shall trigger the sub state machine CANSM_BSM_S_TX_TIMEOUT_EXCEPTION of the CAN network with T_CC_STOPPED_INDICATED.⌋*(SRS_Can_01145, SRS_Can_01142)*

7.2.23.16.5 Trigger: T_CC_STARTED_INDICATED

[SWS_CanSM_00580] ⌈If CanSM module has got all mode indications (ref. to [SWS_CanSM_00396]) for the configured CAN controllers of the CAN network (ref. to [ECUC_CanSM_00141]) after the respective requests to start the CAN controllers of the CAN network (ref. to [SWS_CanSM_00582]), this shall trigger the sub state machine CANSM_BSM_S_TX_TIMEOUT_EXCEPTION of the CAN network with T_CC_STARTED_INDICATED.⌋*(SRS_Can_01145, SRS_Can_01142)*

7.2.23.16.6 Guarding condition: G_CC_STARTED_E_OK

[SWS_CanSM_00581] ⌈The guarding condition G_CC_STARTED_E_OK of the sub state machine CANSM_BSM_S_TX_TIMEOUT_EXCEPTION shall be passed, if all API calls of [SWS_CanSM_00582] have returned E_OK.⌋*(SRS_Can_01145, SRS_Can_-01142)*

7.2.23.16.7 State operation: DO_SET_CC_MODE_STARTED

[SWS_CanSM_00582] ⌈As long the sub state machine CANSM_BSM_S_TX_TIMEOUT_EXCEPTION 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

[ECUC_CanSM_00141]) the API request `CanIf_SetControllerMode` (ref. to [SWS_CanSM_91002]) with `ControllerMode` equal to `CAN_CS_STARTED`, if the current CAN controller mode (ref. to [SWS_CanSM_00638]) is different.⌋*(SRS_Can_-01145, SRS_Can_01142)*

7.2.23.16.8 ExitPoint: TxTimeout

[SWS_CanSM_00655] ⌈If the sub state machine `CANSM_BSM_S_TX_TIMEOUT_EXCEPTION` is triggered with `T_CC_STARTED_INDICATED`, the API `CanIf_SetPduMode` (ref. to [SWS_CanSM_91002]) shall be called with `CANIF_ONLINE`.⌋*()*

7.2.24 Sub state machine: CANSM_BSM_S_CHANGE_BAUDRATE

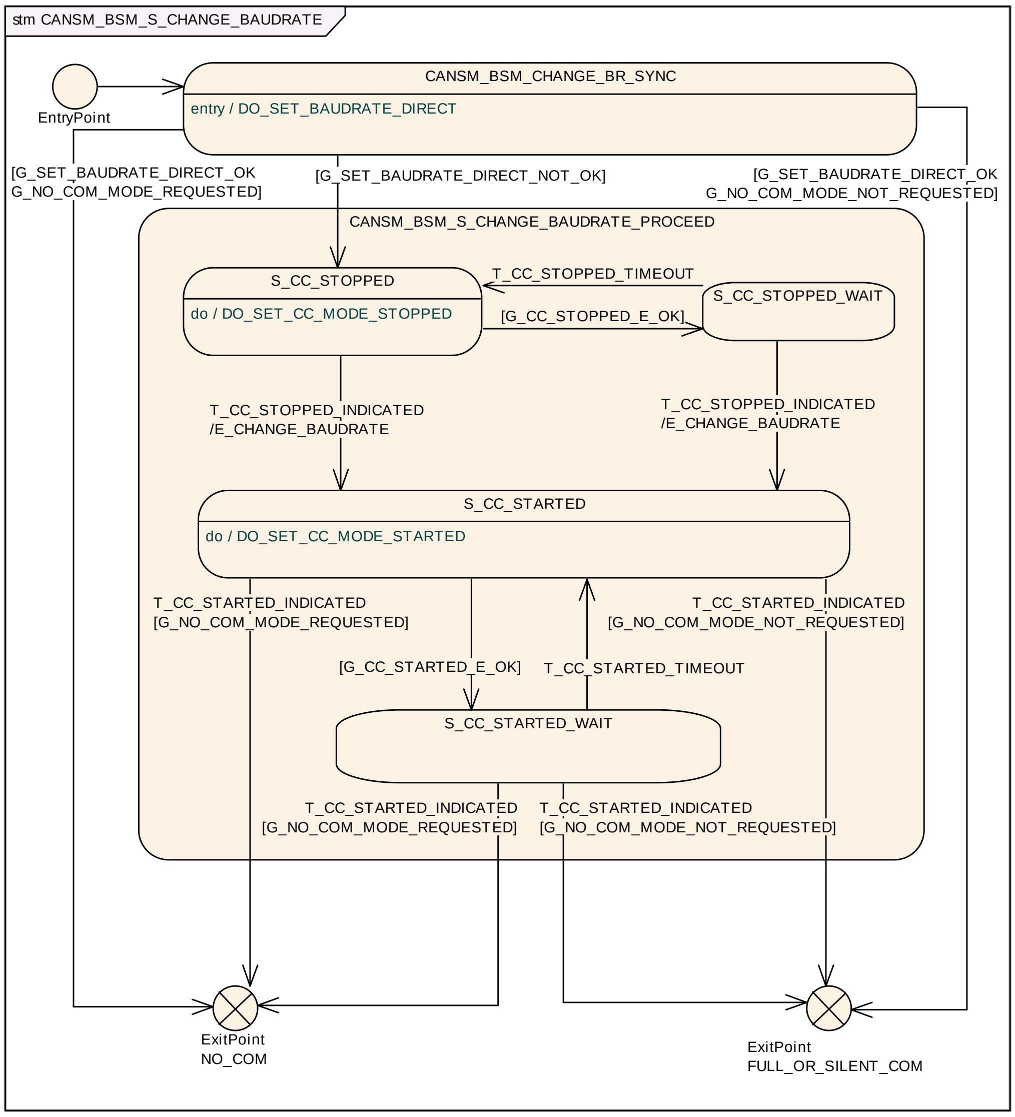
stm CANSM_BSM_S_CHANGE_BAUDRATE
EntryPoint
CANSM_BSM_CHANGE_BR_SYNC
entry / DO_SET_BAUDRATE_DIRECT
[G_SET_BAUDRATE_DIRECT_OK G_NO_COM_MODE_REQUESTED]
[G_SET_BAUDRATE_DIRECT_NOT_OK]
[G_SET_BAUDRATE_DIRECT_OK G_NO_COM_MODE_NOT_REQUESTED]
CANSM_BSM_S_CHANGE_BAUDRATE_PROCEED
S_CC_STOPPED
do / DO_SET_CC_MODE_STOPPED
T_CC_STOPPED_TIMEOUT
[G_CC_STOPPED_E_OK]
S_CC_STOPPED_WAIT
T_CC_STOPPED_INDICATED /E_CHANGE_BAUDRATE
T_CC_STOPPED_INDICATED /E_CHANGE_BAUDRATE
S_CC_STARTED
do / DO_SET_CC_MODE_STARTED
T_CC_STARTED_INDICATED [G_NO_COM_MODE_REQUESTED]
T_CC_STARTED_INDICATED [G_NO_COM_MODE_NOT_REQUESTED]
[G_CC_STARTED_E_OK]
T_CC_STARTED_TIMEOUT
S_CC_STARTED_WAIT
T_CC_STARTED_INDICATED [G_NO_COM_MODE_REQUESTED]
T_CC_STARTED_INDICATED [G_NO_COM_MODE_NOT_REQUESTED]
ExitPoint NO_COM
ExitPoint FULL_OR_SILENT_COM

Figure 7.10: CANSM_BSM_S_CHANGE_BAUDRATE, sub state machine of CANSM_BSM

7.2.24.1 State operation to do on entry: DO_SET_BAUDRATE_DIRECT

[SWS_CanSM_00639] ⌈The state operation DO_SET_BAUDRATE_DIRECT shall call the API request CanIf_SetBaudrate (ref. to [SWS_CanSM_91003])) for all configured CAN controllers of the CAN network (ref. to [ECUC_CanSM_00141] with the respective ControllerId parameter. It shall use as BaudRateConfigID parameter the respective function parameter BaudRateConfigID from the call CanSM_SetBaudrate.⌋*(SRS_Can_01145, SRS_Can_01142)*

7.2.24.2 Guarding condition: G_SET_BAUDRATE_DIRECT_OK

[SWS_CanSM_00641] ⌈If all CanIf_SetBaudrate (ref. to [SWS_CanSM_91003])) (ref. to [SWS_CanSM_00639]) requests returned with E_OK, the guarding condition G_SET_BAUDRATE_DIRECT_OK shall be passed.⌋*(SRS_Can_01145, SRS_Can_01142)*

7.2.24.3 Guarding conditions: G_SET_BAUDRATE_DIRECT_NOT_OK

[SWS_CanSM_00642] ⌈If any of the CanIf_SetBaudrate (ref. to [SWS_CanSM_91003])) (ref. to [SWS_CanSM_00639]) requests did return with E_NOT_OK, the guarding condition G_SET_BAUDRATE_NOT_OK of the state CANSM_BSM_CHANGE_BR_SYNC shall be passed.⌋*(SRS_Can_01145, SRS_Can_01142)*

7.2.24.4 State operation to do in: S_CC_STOPPED

[SWS_CanSM_00524] ⌈As long the sub state machine CANSM_BSM_S_CHANGE_BAUDRATE 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 [ECUC_CanSM_00141]) the API request CanIf_SetControllerMode (ref. to [SWS_CanSM_91002]) with ControllerMode equal to CAN_CS_STOPPED, if the current CAN controller mode (ref. to [SWS_CanSM_00638]) is different.⌋*(SRS_Can_01145, SRS_Can_01142)*

7.2.24.5 Guarding condition: G_CC_STOPPED_OK

[SWS_CanSM_00525] ⌈The guarding condition G_CC_STOPPED_OK of the sub state machine CANSM_BSM_S_CHANGE_BAUDRATE shall be passed, if all API calls of [SWS_CanSM_00524] have returned E_OK.⌋*(SRS_Can_01145, SRS_Can_01142)*

7.2.24.6 Trigger: T_CC_STOPPED_INDICATED

[SWS_CanSM_00526] ⌈If CanSM module has got all mode indications (ref. to [SWS_CanSM_00396]) for the configured CAN controllers of the CAN network (ref. to [ECUC_CanSM_00141]) after the respective requests to stop the CAN controllers of the CAN network (ref. to [SWS_CanSM_00524]), this shall trigger the sub state machine `CANSM_BSM_S_CHANGE_BAUDRATE` of the CAN network with `T_CC_STOPPED_INDICATED`.⌋*(SRS_Can_01145, SRS_Can_01142)*

7.2.24.7 Trigger: T_CC_STOPPED_TIMEOUT

[SWS_CanSM_00527] ⌈After a timeout of `CANSM_MODEREQ_REPEAT_TIME` (ref. to [ECUC_CanSM_00336]) for all supposed controller stopped mode indications (ref. to [SWS_CanSM_00526]), this condition shall trigger the sub state machine `CANSM_BSM_S_CHANGE_BAUDRATE` of the respective network with `T_CC_STOPPED_TIMEOUT`.⌋*(SRS_Can_01145, SRS_Can_01142)*

7.2.24.8 Effect: E_CHANGE_BAUDRATE

[SWS_CanSM_00529] ⌈The effect `E_CHANGE_BAUDRATE` of the sub state machine `CANSM_BSM_S_CHANGE_BAUDRATE` shall call at 1st place for the corresponding CAN network the API `ComM_BusSM_ModeIndication` (ref. to [SWS_CanSM_91002]) with the parameters `Channel` := `CanSMComMNetworkHandleRef` (ref. to [ECUC_CanSM_00161]) and `ComMode` := `COMM_NO_COMMUNICATION`.⌋*(SRS_Can_-01145, SRS_Can_01142)*

[SWS_CanSM_00531] ⌈The effect `E_CHANGE_BAUDRATE` of the sub state machine `CANSM_BSM_S_CHANGE_BAUDRATE` shall call at 2nd place for all configured CAN controllers of the CAN network (ref. to [ECUC_CanSM_00141]) the API request `CanIf_SetBaudrate` (ref. to [SWS_CanSM_91003])) with the respective `ControllerId` parameter and shall use as `BaudRateConfigID` parameter the remembered `BaudRateConfigID` from the call `CanSM_SetBaudrate`.⌋*(SRS_Can_01145, SRS_-Can_01142)*

7.2.24.9 State operation to do in: S_CC_STARTED

[SWS_CanSM_00532] ⌈As long the sub state machine `CANSM_BSM_S_CHANGE_BAUDRATE` 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 [ECUC_CanSM_00141]) the API request `CanIf_SetControllerMode` (ref. to [SWS_CanSM_91002]) with `ControllerMode` equal to `CAN_CS_STARTED`, if the current CAN controller mode (ref. to [SWS_CanSM_00638]) is different.⌋*(SRS_Can_01145, SRS_Can_01142)*

7.2.24.10 Guarding condition: G_CC_STARTED_OK

[SWS_CanSM_00533] ⌈The guarding condition G_CC_STARTED_OK of the sub state machine CANSM_BSM_S_CHANGE_BAUDRATE shall be passed, if all API calls of [SWS_CanSM_00532] have returned E_OK.⌋*(SRS_Can_01145, SRS_Can_01142)*

7.2.24.11 Trigger: T_CC_STARTED_INDICATED

[SWS_CanSM_00534] ⌈If CanSM module has got all mode indications (ref. to [SWS_CanSM_00396]) for the configured CAN controllers of the CAN network (ref. to [ECUC_CanSM_00141]) after the respective requests to start the CAN controllers of the CAN network (ref. to [SWS_CanSM_00532]), this shall trigger the sub state machine CANSM_BSM_S_CHANGE_BAUDRATE of the CAN network with T_CC_STARTED_INDICATED.⌋*(SRS_Can_01145, SRS_Can_01142)*

7.2.24.12 Trigger: T_CC_STARTED_TIMEOUT

[SWS_CanSM_00535] ⌈After a timeout of CANSM_MODEREQ_REPEAT_TIME (ref. to [ECUC_CanSM_00336]) for all supposed controller started mode indications (ref. to[SWS_CanSM_00534]), this condition shall trigger the sub state machine CANSM_BSM_S_CHANGE_BAUDRATE of the respective network with T_CC_STARTED_TIMEOUT.⌋*(SRS_Can_01145, SRS_Can_01142)*

7.2.24.13 Guarding condition: G_NO_COM_MODE_REQUESTED

[SWS_CanSM_00542] ⌈The sub state machine CANSM_BSM_S_CHANGE_BAUDRATE shall pass the guarding condition G_NO_COM_MODE_REQUESTED, if the latest accepted communication mode request with CanSM_RequestComMode (ref. to [SWS_CanSM_00635]) for the respective network handle of the state machine has been with the parameter ComM_Mode equal to COMM_NO_COMMUNICATION.⌋*(SRS_Can_01145, SRS_Can_01142)*

7.2.24.14 Guarding condition: G_NO_COM_MODE_NOT_REQUESTED

[SWS_CanSM_00543] ⌈The sub state machine CANSM_BSM_S_CHANGE_BAUDRATE shall pass the guarding condition G_NO_COM_MODE_NOT_REQUESTED, if the latest accepted communication mode request with CanSM_RequestComMode (ref. to [SWS_CanSM_00635]) 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.⌋*(SRS_Can_01145, SRS_Can_01142)*

7.3 Error Classification

Section "Error Handling" of the document [2] "General Specification of Basic Software Modules" describes the error handling of the Basic Software in detail. Above all, it constitutes a classification scheme consisting of five error types which may occur in BSW modules.

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

7.3.1 Development Errors

[SWS_CanSM_00654] Definiton of development errors in module CanSM ⌈

Type of error	*Related error code*	*Error value*
API service used without module initialization	CANSM_E_UNINIT	0x01
API service called with wrong pointer	CANSM_E_PARAM_POINTER	0x02
API service called with wrong parameter	CANSM_E_INVALID_NETWORK_HANDLE	0x03
API service called with wrong parameter	CANSM_E_PARAM_CONTROLLER	0x04
API service called with wrong parameter	CANSM_E_PARAM_TRANSCEIVER	0x05
DeInit API service called when not all CAN networks are in state CANSM_NO_ COMMUNICATION	CANSM_E_NOT_IN_NO_COM	0x0B

⌋*(SRS_BSW_00337)*

7.3.2 Runtime Errors

[SWS_CanSM_00664] Definiton of runtime errors in module CanSM ⌈

Type of error	*Related error code*	*Error value*
Mode request for a network failed more often than allowed by configuration	CANSM_E_MODE_REQUEST_TIMEOUT	0x0A

⌋*(SRS_BSW_00466)*

7.3.3 Transient Faults

There are no transient faults.

7.3.4 Production Errors

There are no production errors.

7.3.5 Extended Production Errors

There are no extended production errors.

7.3.5.1 CANSM_E_BUS_OFF

[SWS_CanSM_00666] ⌈

Error Name:	CANSM_E_BUS_OFF (ref. to ECUC_CanSM_00070)	
Short Description:	Bus-off detection	
Long Description:	The bus-off recovery state machine of a CAN network has detected a certain amount of sequential bus-offs without successful recovery	
Recommended DTC:	Assigned by DEM	
Detection Criteria:	Fail	PRE_FAILED when CanSM_ControllerBusOff is called (T_BUS_OFF/E_BUS_OFF), debouncing to be defined by OEM in DEM
	Pass	After successful transmission of a CAN frame (G_BUS_OFF_PASSIVE/E_BUS_OFF_PASSIVE)
Secondary Parameters:	None	
Time Required:	PRE_FAILED immediately (in error interrupt context), FAILED depending on debounce configuration of DEM	
Monitor Frequency	Continuous	
MIL illumniation:	Assigned by DEM	

⌋*()*

7.4 ECU online active / passive mode

[SWS_CanSM_00646] ⌈The CanSM module shall store the state of the requested ECU passive mode (ref. to [SWS_CanSM_00644]).⌋*(SRS_Can_01158)*

[SWS_CanSM_00649] ⌈When `CanSM_SetEcuPassive` is called with `CanSM_Passive=true;` (ref. to [SWS_CanSM_00644]), then the CanSM shall change all PDU modes of the configured CAN controllers, which are `CANIF_ONLINE` at the moment to `CANIF_TX_OFFLINE_ACTIVE` by calling the API `CanIf_SetPduMode` (ref. to [SWS_CanSM_91002]) with the parameters `ControllerId` := `CanSMControllerId` (ref. to [ECUC_CanSM_00141]) and `PduModeRequest` := `CANIF_TX_OFFLINE_ACTIVE`.⌋*(SRS_Can_01158)*

[SWS_CanSM_00650] ⌈If CanSM_SetEcuPassive called with CanSM_Passive= false; (ref. to [SWS_CanSM_00644]), then the CanSM shall change all PDU modes of the configured CAN controllers, which are CANIF_TX_OFFLINE_ACTIVE at the moment to CANIF_ONLINE by calling the API CanIf_SetPduMode (ref. to [SWS_CanSM_91002]) with the parameters ControllerId := CanSMControllerId (ref. to [ECUC_CanSM_00141]) and PduModeRequest := CANIF_ONLINE.⌋ *(SRS_Can_01158)*

[SWS_CanSM_00656] ⌈If the CanSM module needs informations about the actual PduMode, the CanSM shall call the API CanIf_GetPduMode (ref. to [SWS_CanSM_91002]) to get the current Pdu Mode of the CanIf.⌋ *(SRS_Can_01158)*

7.5 Non-functional design rules

The CanSM shall cover the software module design requirements of the [12, General Requirements on Basic Software Modules].

8 API specification

8.1 Imported types

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

[SWS_CanSM_00243] Definition of imported datatypes of module CanSM ⌈

Module	***Header File***	***Imported Type***
Can	Can_GeneralTypes.h	Can_ControllerStateType
CanIf	CanIf.h	CanIf_NotifStatusType
	CanIf.h	CanIf_PduModeType
CanTrcv	Can_GeneralTypes.h	CanTrcv_TrcvModeType
ComM	Rte_ComM_Type.h	ComM_ModeType
ComStack_Types	ComStack_Types.h	NetworkHandleType
Dem	Rte_Dem_Type.h	Dem_EventIdType
	Rte_Dem_Type.h	Dem_EventStatusType
Std	Std_Types.h	Std_ReturnType
	Std_Types.h	Std_VersionInfoType

⌋*()*

8.2 Type definitions

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

8.2.1 CanSM_ConfigType

[SWS_CanSM_00597] Definition of datatype CanSM_ConfigType ⌈

Name	CanSM_ConfigType	
Kind	Structure	
Elements	–	
	Type	–
	Comment	–
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.	
Available via	CanSM.h	

⌋*(SRS_BSW_00400, SRS_BSW_00438)*

8.2.2 CanSM_BswMCurrentStateType

[SWS_CanSM_00598] Definition of datatype CanSM_BswMCurrentStateType ⌈

Name	CanSM_BswMCurrentStateType		
Kind	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		
Available via	CanSM.h		

⌋*(SRS_ModeMgm_09251)*

8.3 Function definitions

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

8.3.1 CanSM_Init

[SWS_CanSM_00023] Definition of API function CanSM_Init ⌈

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 Can SM
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	This service initializes the CanSM module	
Available via	CanSM.h	

⌋*(SRS_BSW_00405, SRS_BSW_00101, SRS_BSW_00406, SRS_BSW_00358, SRS_BSW_00414, SRS_BSW_00404, SRS_BSW_00400, SRS_BSW_00438)*

8.3.2 CanSM_DeInit

[SWS_CanSM_91001] Definition of API function CanSM_DeInit ⌈

Service Name	CanSM_DeInit	
Syntax	`void CanSM_DeInit ( void )`	
Service ID [hex]	0x14	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	This service de-initializes the CanSM module.	
Available via	CanSM.h	

⌋*(SRS_Can_01164, SRS_BSW_00336)* Note: General behavior and constraints on de-initialization functions are specified by [SWS_BSW_00152], [SWS_BSW_00072], [SWS_BSW_00232], [SWS_BSW_00233].

Caveat: Caller of the `CanSM_DeInit` function has to ensure all CAN networks are in the state `CANSM_NO_COMMUNICATION`.

[SWS_CanSM_00660] ⌈If development error detection for the CanSM module is enabled: The function `CanSM_DeInit` shall raise the error `CANSM_E_NOT_IN_NO_COM` if not all CAN networks are in state `CANSM_NO_COMMUNICATION`.⌋*(SRS_BSW_00369)*

8.3.3 CanSM_RequestComMode

[SWS_CanSM_00062] Definition of API function CanSM_RequestComMode ⌈

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

⌋*(SRS_Can_01145, SRS_Can_01142)* Remark: Please refer to [5, Specification of Communication Manager] for a detailed description of the communication modes.

[SWS_CanSM_00369] ⌈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 [ECUC_CanSM_00161]).⌋*(SRS_Can_01145, SRS_Can_01142)*

[SWS_CanSM_00370] ⌈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 [ECUC_CanSM_00161]).⌋*(SRS_Can_01145, SRS_Can_01142)*

[SWS_CanSM_00555] ⌈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 [SWS_CanSM_00423], [SWS_CanSM_00430]).⌋*(SRS_Can_01145, SRS_Can_01142)*

[SWS_CanSM_00183] ⌈The function `CanSM_RequestComMode` shall call the function `Det_ReportError` (ref. to [SWS_CanSM_91003]) with `ErrorId` parameter `CANSM_E_INVALID_NETWORK_HANDLE`, if it does not accept the network handle of the request.⌋*(SRS_Can_01145, SRS_Can_01142)*

[SWS_CanSM_00182] ⌈If the function `CanSM_RequestComMode` accepts the request, the request shall be considered by the CanSM state machine (ref. to [SWS_CanSM_00635]).⌋*(SRS_Can_01145, SRS_Can_01142)*

[SWS_CanSM_00184] ⌈If the CanSM module is not initialized, when the function `CanSM_RequestComMode` is called, then this function shall call the function `Det_ReportError` (ref. to [SWS_CanSM_91003]) with `ErrorId` parameter `CANSM_E_UNINIT`.⌋*(SRS_BSW_00406)*

8.3.4 CanSM_GetCurrentComMode

[SWS_CanSM_00063] Definition of API function CanSM_GetCurrentComMode ⌈

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.	
Available via	CanSM.h	

⌋*(SRS_ModeMgm_09084)*

[SWS_CanSM_00282] ⌈The CanSM module shall return E_NOT_OK for the API request CanSM_GetCurrentComMode until the call of the provided API CanSM_Init (ref. to [SWS_CanSM_00023]).⌋*(SRS_Can_01142)*

[SWS_CanSM_00371] ⌈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 [ECUC_CanSM_00161]).⌋*(SRS_Can_-01142)*

[SWS_CanSM_00372] ⌈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 [ECUC_CanSM_00161]).⌋*(SRS_Can_-01142)*

[SWS_CanSM_00187] ⌈The function CanSM_GetCurrentComMode shall call the function Det_ReportError (ref. to [SWS_CanSM_91003]) with ErrorId parameter CANSM_E_INVALID_NETWORK_HANDLE, if it does not accept the network handle of the request.⌋*(SRS_Can_01142)*

[SWS_CanSM_00186] ⌈The function CanSM_GetCurrentComMode shall put out the current communication mode for the network handle (ref. to [SWS_CanSM_00266]) to the designated pointer of type ComM_ModeType, if it accepts the request.⌋*(SRS_-Can_01142)*

[SWS_CanSM_00188] ⌈If the CanSM module is not initialized (ref. to [SWS_CanSM_00282]), when the function CanSM_GetCurrentComMode is called, then this function shall call the function Det_ReportError (ref. to [SWS_CanSM_91003]) with ErrorId parameter CANSM_E_UNINIT.⌋*(SRS_-Can_01142)*

[SWS_CanSM_00360] ⌈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.⌋*(SRS_Can_01142)*

8.3.5 CanSM_StartWakeupSource

[SWS_CanSM_00609] Definition of API function CanSM_StartWakeupSource ⌈

Service Name	CanSM_StartWakeupSource	

Syntax	`Std_ReturnType CanSM_StartWakeupSource (`
	`  NetworkHandleType network`
	`)`	

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

⌋*(SRS_Can_01145)*

[SWS_CanSM_00611] ⌈The API function `CanSM_StartWakeupSource` shall return `E_NOT_OK`, if the CanSM module is not initialized yet with `CanSM_Init` (ref. to [SWS_CanSM_00023]).⌋*(SRS_Can_01145)*

[SWS_CanSM_00617] ⌈The function `CanSM_StartWakeupSource` shall call the function `Det_ReportError` (ref. to [SWS_CanSM_91003]) with `ErrorId` parameter `CANSM_E_UNINIT`, if the CanSM module is not initialized yet with `CanSM_Init` (ref. to [SWS_CanSM_00023]).⌋*(SRS_Can_01145)*

[SWS_CanSM_00612] ⌈The function `CanSM_StartWakeupSource` shall return `E_NOT_OK`, if the CanSM module is initialized and the network parameter of the request is not a handle contained in the configuration of the CanSM module (ref. to [ECUC_CanSM_00161]).⌋*(SRS_Can_01145)*

[SWS_CanSM_00613] ⌈The function `CanSM_StartWakeupSource` shall call the function `Det_ReportError` (ref. to [SWS_CanSM_91003]) with `ErrorId` parameter `CANSM_E_INVALID_NETWORK_HANDLE`, if the CanSM module is initialized and the requested handle is invalid concerning the CanSM configuration (ref. to [ECUC_CanSM_00161]).⌋*(SRS_Can_01145)*

[SWS_CanSM_00616] ⌈The function `CanSM_StartWakeupSource` shall return `E_OK` and it shall be considered as trigger (ref. to [SWS_CanSM_00607]) for the state machine of the related network, if the CanSM module is initialized and the requested handle is valid concerning the CanSM configuration (ref. to [ECUC_CanSM_00161]).⌋ *(SRS_Can_01145)*

8.3.6 CanSM_StopWakeupSource

[SWS_CanSM_00610] Definition of API function CanSM_StopWakeupSource ⌈

Service Name	CanSM_StopWakeupSource	

Syntax	`Std_ReturnType CanSM_StopWakeupSource (`
	`  NetworkHandleType network`
	`)`	

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

⌋*(SRS_Can_01145)*

[SWS_CanSM_00618] ⌈The API function `CanSM_StopWakeupSource` shall return `E_NOT_OK`, if the CanSM module is not initialized yet with `CanSM_Init` (ref. to [SWS_CanSM_00023]).⌋*(SRS_Can_01145)*

[SWS_CanSM_00619] ⌈The function `CanSM_StopWakeupSource` shall call the function `Det_ReportError` (ref. to [SWS_CanSM_91003]) with `ErrorId` parameter `CANSM_E_UNINIT`, if the CanSM module is not initialized yet with `CanSM_Init` (ref. to [SWS_CanSM_00023]).⌋*(SRS_Can_01145)*

[SWS_CanSM_00620] ⌈The function `CanSM_StopWakeupSource` shall return `E_NOT_OK`, if the CanSM module is initialized and the network parameter of the request is not a handle contained in the configuration of the CanSM module (ref. to [ECUC_CanSM_00161]).⌋*(SRS_Can_01145)*

[SWS_CanSM_00621] ⌈The function `CanSM_StopWakeupSource` shall call the function `Det_ReportError` (ref. to [SWS_CanSM_91003]) with `ErrorId` parameter `CANSM_E_INVALID_NETWORK_HANDLE`, if the CanSM module is initialized and the requested handle is invalid concerning the CanSM configuration (ref. to [ECUC_CanSM_00161]).⌋*(SRS_Can_01145)*

[SWS_CanSM_00622] ⌈The function `CanSM_StopWakeupSource` shall return `E_OK` and it shall be considered as trigger (ref. to [SWS_CanSM_00608]) for the state machine of the related network, if the CanSM module is initialized and the requested handle is valid concerning the CanSM configuration (ref. to [ECUC_CanSM_00161]).⌋*(SRS_Can_01145)*

8.3.7 Optional

8.3.7.1 CanSM_GetVersionInfo

[SWS_CanSM_00024] Definition of API function CanSM_GetVersionInfo ⌈

Service Name	CanSM_GetVersionInfo	

Syntax	`void CanSM_GetVersionInfo (`
	`  Std_VersionInfoType* VersionInfo`
	`)`	

Service ID [hex]	0x01	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
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)	
Available via	CanSM.h	

⌋*(SRS_BSW_00407, SRS_BSW_00003)*

[SWS_CanSM_00374] ⌈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`.⌋*(SRS_BSW_00407, SRS_BSW_00003)*

8.3.7.2 CanSM_SetBaudrate

[SWS_CanSM_00561] Definition of API function CanSM_SetBaudrate ⌈

Service Name	CanSM_SetBaudrate	

Syntax	`Std_ReturnType CanSM_SetBaudrate (`
	`  NetworkHandleType Network,`
	`  uint16 BaudRateConfigID`
	`)`	

Service ID [hex]	0x0d	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different Networks. Non reentrant for the same Network.	
Parameters (in)	Network	Handle of the addressed CAN network for the baud rate change
	BaudRateConfigID	references a baud rate configuration by ID (see CanController BaudRateConfigID)
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: Service request accepted, setting of (new) baud rate started `E_NOT_OK`: Service request not accepted

▽

△

Description	This service shall start an asynchronous process to change the baud rate for the configured CAN controllers of a certain CAN network. Depending on necessary baud rate modifications the controllers might have to reset.
Available via	CanSM.h

⌋*(SRS_Can_01142)*

[SWS_CanSM_00569] ⌈The CanSM module shall provide the API function `CanSM_SetBaudrate`, if the `CanSMSetBaudrateApi` parameter is configured with the value `TRUE`.⌋*(SRS_Can_01142)*

[SWS_CanSM_00570] ⌈The CanSM module shall not provide the API function `CanSM_SetBaudrate`, if the `CanSMSetBaudrateApi` is configured with the value `FALSE`.⌋*(SRS_Can_01142)*

[SWS_CanSM_00502] ⌈The CanSM module shall deny the `CanSM_SetBaudrate` API request, if the `NetworkHandle` parameter does not match to the configured Network handles of the CanSM module (ref. to [ECUC_CanSM_00161]).⌋*(SRS_Can_01142)*

[SWS_CanSM_00504] ⌈The function `CanSM_SetBaudrate` shall call the function `Det_ReportError` (ref. to [SWS_CanSM_91003]) with `ErrorId` parameter `CANSM_E_INVALID_NETWORK_HANDLE`, if it does not accept the network handle of the request.⌋*(SRS_Can_01142)*

[SWS_CanSM_00505] ⌈The function `CanSM_SetBaudrate` shall deny its request, if the requested CAN network is not in the communication mode `COMM_FULL_COMMUNICATION`.⌋*(SRS_Can_01142)*

[SWS_CanSM_00530] ⌈The CanSM module shall deny the `CanSM_SetBaudrate` API request, if the CanSM module is not initialized.⌋*(SRS_Can_01142)*

[SWS_CanSM_00506] ⌈If the function `CanSM_SetBaudrate` is called and the CanSM module is not initialized, then this function shall call the function `Det_ReportError` (ref. to [SWS_CanSM_91003]) with `ErrorId` parameter `CANSM_E_UNINIT`.⌋*(SRS_Can_01142)*

[SWS_CanSM_00503] ⌈IIf no condition is present to deny the `CanSM_SetBaudrate` request according to [SWS_CanSM_00502] and [SWS_CanSM_00505], [SWS_CanSM_00530], then the CanSM module shall return `E_OK` and operate the process for the requested baud rate change as specified with [SWS_CanSM_00507].⌋*(SRS_Can_01142)*

8.3.7.3 CanSM_SetEcuPassive

[SWS_CanSM_00644] Definition of API function CanSM_SetEcuPassive ⌈

Service Name	CanSM_SetEcuPassive	
Syntax	`Std_ReturnType CanSM_SetEcuPassive ( boolean CanSM_Passive )`	
Service ID [hex]	0x13	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	CanSM_Passive	TRUE: set all CanSM channels to passive, i.e. receive only FALSE: set all CanSM channels back to non-passive
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: Request accepted `E_NOT_OK`: Request not accepted
Description	This function can be used to set all CanSM channels of the ECU to a receive only mode.	
Available via	CanSM.h	

⌋*(SRS_Can_01158)*

[SWS_CanSM_00645] ⌈The CanSM module shall provide the API function `CanSM_SetEcuPassive`, if the `CanSMTxOfflineActiveSupport` parameter is configured with the value TRUE.⌋*(SRS_Can_01158)*

8.4 Call-back notifications

This is a list of functions provided for other modules.

8.4.1 CanSM_ControllerBusOff

[SWS_CanSM_00064] Definition of callback function CanSM_ControllerBusOff ⌈

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.
Available via	CanSM_CanIf.h

⌋*(SRS_BSW_00359, SRS_BSW_00333)*

[SWS_CanSM_00189] ⌈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 (ref. to [SWS_CanSM_91003]) with ErrorId parameter CANSM_E_PARAM_CONTROLLER.⌋*(SRS_BSW_00359, SRS_BSW_00333)*

[SWS_CanSM_00190] ⌈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 (ref. to [SWS_CanSM_91003]) with ErrorId parameter CANSM_E_UNINIT.⌋*(SRS_BSW_00359, SRS_BSW_00333)*

[SWS_CanSM_00235] ⌈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 [SWS_CanSM_00500]).⌋*(SRS_BSW_00359, SRS_BSW_00333)*

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

[SWS_CanSM_00396] Definition of callback function CanSM_ControllerModeIndication ⌈

Service Name	CanSM_ControllerModeIndication	
Syntax	`void CanSM_ControllerModeIndication ( uint8 ControllerId, Can_ControllerStateType 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.
Available via	CanSM_CanIf.h

⌋(*SRS_Can_01145*)

[SWS_CanSM_00397] ⌈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` (ref. to [SWS_CanSM_91003]) with `ErrorId` parameter `CANSM_E_PARAM_CONTROLLER`.⌋ (*SRS_Can_01145*)

[SWS_CanSM_00398] ⌈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` (ref. to [SWS_CanSM_91003]) with `ErrorId` parameter `CANSM_E_UNINIT`.⌋(*SRS_Can_01145*)

8.4.3 CanSM_TransceiverModeIndication

[SWS_CanSM_00399] Definition of callback function CanSM_TransceiverMode Indication ⌈

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.	
Available via	CanSM_CanIf.h	

⌋(*SRS_Can_01145*, *SRS_Can_01142*) Note: `CANTRCV_TRCVMODE_SLEEP` state can be requested to Can_Trcv module only by integration code and not by CanSM module. Hence when `CanSM_TransceiverModeIndication`() is invoked for `CANTRCV_TRCVMODE_SLEEP`, CanSM module should ignore this request.

[SWS_CanSM_00400] ⌈If the function `CanSM_TransceiverModeIndication` gets a `TransceiverId`, which is not configured as CanSM`TransceiverId` in the configuration of the CanSM module, it shall call the function `Det_ReportError` (ref. to [SWS_CanSM_91003]) with `ErrorId` parameter `CANSM_E_PARAM_TRANSCEIVER`.⌋ (*SRS_Can_01145*)

[SWS_CanSM_00401] ⌈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` (ref. to [SWS_CanSM_91003]) with `ErrorId` parameter `CANSM_E_UNINIT`.⌋*(SRS_Can_01145)*

8.4.4 CanSM_TxTimeoutException

[SWS_CanSM_00410] Definition of callback function CanSM_TxTimeoutException ⌈

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 within the respective network state machine of the CanSM module.	
Available via	CanSM_CanIf.h	

⌋*(SRS_Can_01142, SRS_Can_01145)*

[SWS_CanSM_00411] ⌈The function `CanSM_TxTimeoutException` shall report `CANSM_E_UNINIT` to the DET, if the CanSM module is not initialized yet.⌋*(SRS_Can_01145)*

[SWS_CanSM_00412] ⌈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.⌋*(SRS_Can_01145)*

Remarks: Reentrancy is necessary for different Channels.

8.4.5 CanSM_ClearTrcvWufFlagIndication

[SWS_CanSM_00413] Definition of callback function CanSM_ClearTrcvWufFlagIndication ⌈

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.	
Available via	CanSM_CanIf.h	

⌋*(SRS_Can_01145)*

[SWS_CanSM_00414] ⌈The function `CanSM_ClearTrcvWufFlagIndication` shall report `CANSM_E_UNINIT` to the DET, if the CanSM module is not initialized yet.⌋*(SRS_Can_01145)*

[SWS_CanSM_00415] ⌈If the function `CanSM_ClearTrcvWufFlagIndication` gets a `TransceiverId`, which is not configured (ref. to [ECUC_CanSM_00137]) in the configuration of the CanSM module, it shall call the function `Det_ReportError` (ref. to [SWS_CanSM_91003]) with `ErrorId` parameter `CANSM_E_PARAM_TRANSCEIVER`.⌋*(SRS_Can_01145)*

8.4.6 CanSM_CheckTransceiverWakeFlagIndication

[SWS_CanSM_00416] Definition of callback function CanSM_CheckTransceiverWakeFlagIndication ⌈

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 CanIf_CheckTrcvWakeFlag API process end for the notified CAN Transceiver.
Available via	CanSM_CanIf.h

⌋*(SRS_Can_01145)*

[SWS_CanSM_00417] ⌈The function `CanSM_CheckTransceiverWakeFlagIndication` shall report `CANSM_E_UNINIT` to the DET, if the CanSM module is not initialized yet.⌋*(SRS_Can_01145)*

[SWS_CanSM_00418] ⌈If the function `CanSM_CheckTransceiverWakeFlagIndication` gets a `TransceiverId`, which is not configured (ref. to [ECUC_CanSM_00137]) in the configuration of the CanSM module, it shall call the function `Det_ReportError` (ref. to [SWS_CanSM_91003]) with `ErrorId` parameter `CANSM_E_PARAM_TRANSCEIVER`.⌋*(SRS_Can_01145)*

8.4.7 CanSM_ConfirmPnAvailability

[SWS_CanSM_00419] Definition of callback function CanSM_ConfirmPnAvailability ⌈

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.	
Available via	CanSM_CanIf.h	

⌋*(SRS_Can_01145)*

[SWS_CanSM_00546] ⌈The function `CanSM_ConfirmPnAvailability` shall notify the Can_Nm module (ref. to [SWS_CanSM_00422]), if it is called with a configured Transceiver as input parameter (ref. to [ECUC_CanSM_00137]).⌋*(SRS_Can_01145)*

[SWS_CanSM_00420] ⌈The function `CanSM_ConfirmPnAvailability` shall report `CANSM_E_UNINIT` to the DET, if the CanSM module is not initialized yet.⌋*(SRS_Can_-01145)*

[SWS_CanSM_00421] ⌈If the function `CanSM_ConfirmPnAvailability` gets a `TransceiverId`, which is not configured (ref. to [ECUC_CanSM_00137]) in the configuration of the CanSM module, it shall call the function `Det_ReportError` (ref. to

[SWS_CanSM_91003]) with ErrorId parameter CANSM_E_PARAM_TRANSCEIVER.⌋ *(SRS_Can_01145)*

8.4.8 CanSM_ConfirmCtrlPnAvailability

[SWS_CanSM_91004]{DRAFT} **Definition of callback function CanSM_ConfirmCtrlPnAvailability** ⌈

Service Name	CanSM_ConfirmCtrlPnAvailability (draft)	
Syntax	`void CanSM_ConfirmCtrlPnAvailability ( uint8 ControllerId )`	
Service ID [hex]	0x15	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ControllerId	CAN controller, which was checked for PN availability
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	This callback function indicates that the controller is running in PN communication mode. **Tags:** atp.Status=draft	
Available via	CanSM_CanIf.h	

⌋*(SRS_Can_01145)*

[SWS_CanSM_00668]{DRAFT} ⌈The function CanSM_ConfirmCtrlPnAvailability shall notify the CanNm module (ref. to [SWS_CanSM_00667]), if it is called with a configured Controller as input parameter (ref. to [ECUC_CanSM_00141]).⌋*(SRS_Can_01145)*

[SWS_CanSM_00669]{DRAFT} ⌈The function CanSM_ConfirmCtrlPnAvailability shall report CANSM_E_UNINIT to the DET, if the CanSM module is not initialized yet.⌋*(SRS_Can_01145)*

[SWS_CanSM_00670]{DRAFT} ⌈If the function CanSM_ConfirmCtrlPnAvailability gets a ControllerId, which is not configured (ref. to [ECUC_CanSM_00141]) in the configuration of the CanSM module, it shall call the function Det_ReportError (ref. to [SWS_CanSM_91003]) with ErrorId parameter CANSM_E_PARAM_CONTROLLER.⌋*(SRS_Can_01145)*

8.5 Scheduled functions

For details refer to the chapter 8.5 "Scheduled functions" in SWS_BSWGeneral.

8.5.1 CanSM_MainFunction

[SWS_CanSM_00065] Definition of scheduled function CanSM_MainFunction ⌈

Service Name	CanSM_MainFunction
Syntax	`void CanSM_MainFunction ( void )`
Service ID [hex]	0x05
Description	Scheduled function of the CanSM
Available via	SchM_CanSM.h

⌋*(SRS_BSW_00424, SRS_BSW_00425, SRS_Can_01145, SRS_Can_01142)*

[SWS_CanSM_00167] ⌈The main function of the CanSM module shall operate the effects of the CanSM state machine, which the CanSM module shall implement for each configured CAN Network.⌋*(SRS_BSW_00424, SRS_BSW_00425, SRS_Can_-01145, SRS_Can_01142)*

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.

[SWS_CanSM_91002] Definition of mandatory interfaces in module CanSM ⌈

API Function	Header File	Description
BswM_CanSM_CurrentState	BswM_CanSM.h	Function called by CanSM to indicate its current state.
CanIf_CheckTrcvWakeFlag	CanIf.h	Requests the CanIf module to check the Wake flag of the designated CAN transceiver.
CanIf_ClearTrcvWufFlag	CanIf.h	Requests the CanIf module to clear the WUF flag of the designated CAN transceiver.
CanIf_GetPduMode	CanIf.h	This service reports the current mode of a requested PDU channel.
CanIf_GetTxConfirmationState	CanIf.h	This service reports, if any TX confirmation has been done for the whole CAN controller since the last CAN controller start.
CanIf_SetControllerMode	CanIf.h	This service calls the corresponding CAN Driver service for changing of the CAN controller mode.
CanIf_SetPduMode	CanIf.h	This service sets the requested mode at the L-PDUs of a predefined logical PDU channel.

▽

△

API Function	Header File	Description
CanIf_SetTrcvMode	CanIf.h	This service changes the operation mode of the tansceiver TransceiverId, via calling the corresponding CAN Transceiver Driver service.
CanNm_ConfirmPnAvailability	CanNm.h	Enables the PN filter functionality on the indicated NM channel. Availability: The API is only available if CanNmGlobalPnSupport is TRUE.
ComM_BusSM_ModeIndication	ComM.h	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_SetEventStatus	Dem.h	Called by SW-Cs or BSW modules to report monitor status information to the Dem. BSW modules calling Dem_SetEventStatus can safely ignore the return value. This API will be available only if ({Dem/Dem ConfigSet/DemEventParameter/DemEvent ReportingType} == STANDARD_REPORTING)
Det_ReportRuntimeError	Det.h	Service to report runtime errors. If a callout has been configured then this callout shall be called.

⌋*()*

8.6.1.1 Remark: Usage of CanIf_SetPduMode

Although the CanIf module provides more requestable PDU modes, the CanSM module only uses the parameters CANIF_ONLINE, CANIF_TX_OFFLINE_ACTIVE and CANIF_TX_OFFLINE for the call of the API CanIf_SetPduMode.

The CANIF_OFFLINE mode is assumed automatically by CanIf and needs not to be set by CanSM.

8.6.2 Optional Interfaces

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

[SWS_CanSM_91003] Definition of optional interfaces in module CanSM ⌈

API Function	Header File	Description
CanIf_SetBaudrate	CanIf.h	This service shall set the baud rate configuration of the CAN controller. Depending on necessary baud rate modifications the controller might have to reset.
Det_ReportError	Det.h	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.

8.6.3.1 <User_GetBusOffDelay>

[SWS_CanSM_00637] Definition of configurable interface <User_GetBusOffDelay> ⌈

Service Name	<User_GetBusOffDelay>	

Syntax	`void <User_GetBusOffDelay> (`
	`  NetworkHandleType network,`
	`  uint8* delayCyclesPtr`
	`)`	

Sync/Async	Synchronous	
Reentrancy	Reentrant for different networks	
Parameters (in)	network	CAN network where a BusOff occurred.
Parameters (inout)	None	
Parameters (out)	delayCyclesPtr	Number of CanSM base cycles to wait additionally to L1/L2 after a BusOff occurred.
Return value	None	
Description	This callout function returns the number of CanSM base cycles to wait additionally to L1/L2 after a BusOff occurred.	
Available via	Configuration parameter CanSM/CanSMGeneral/CanSMGetBusOffDelayHeader	

⌋*(SRS_Can_01144, SRS_Can_01146)*

9 Sequence diagrams

All interactions of the CanSM module with the depending modules CanIf, ComM, BswM, Dem and CanNm are specified in the state machine diagrams (ref. to Figure 7-1-Figure 7-10). Therefore the CanSM SWS provides only some exemplary sequences for the use case 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 [11, Specification of CAN Transceiver Driver].

9.1 Sequence diagram CanSm_StartCanController

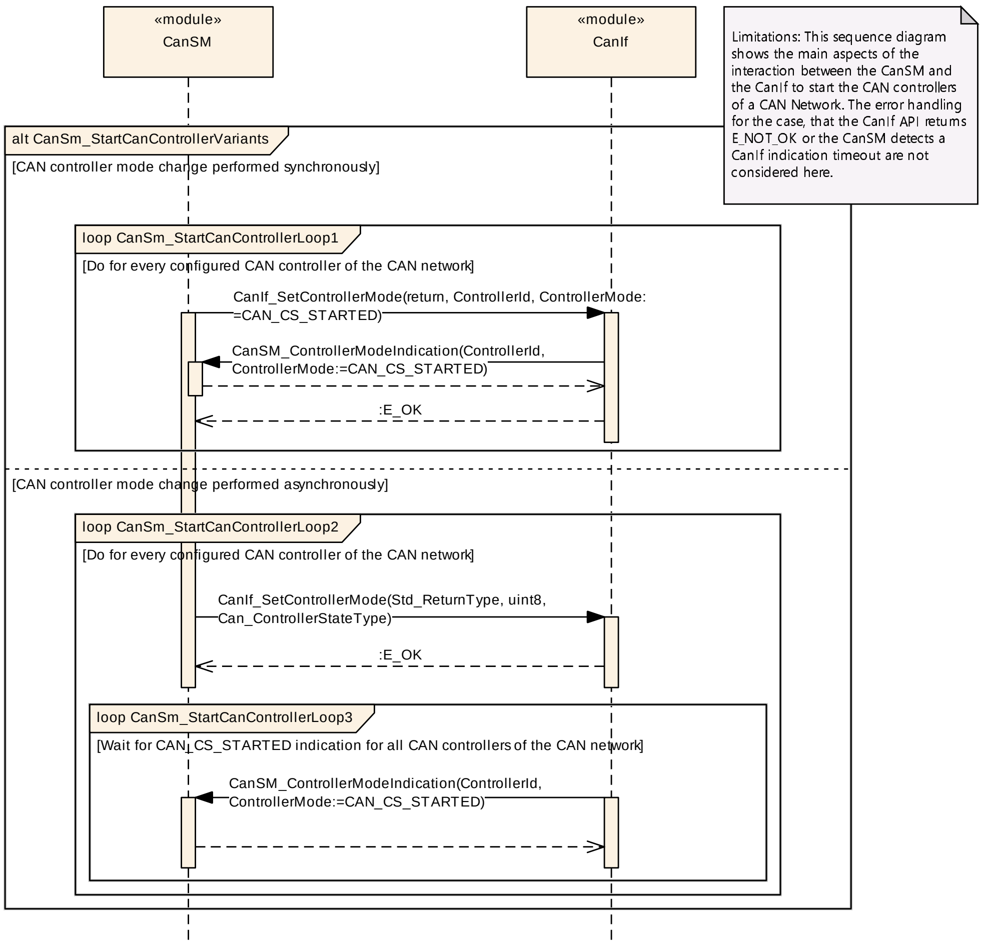
«module»
CanSM
«module»
CanIf
Limitations: This sequence diagram shows the main aspects of the interaction between the CanSM and the CanIf to start the CAN controllers of a CAN Network. The error handling for the case, that the CanIf API returns E_NOT_OK or the CanSM detects a CanIf indication timeout 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:=CAN_CS_STARTED)
CanSM_ControllerModeIndication(ControllerId, ControllerMode:=CAN_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, Can_ControllerStateType)
:E_OK
loop CanSm_StartCanControllerLoop3
[Wait for CAN_CS_STARTED indication for all CAN controllers of the CAN network]
CanSM_ControllerModeIndication(ControllerId, ControllerMode:=CAN_CS_STARTED)

Figure 9.1: CanSm_StartCanController

9.2 Sequence diagram CanSm_StopCanController

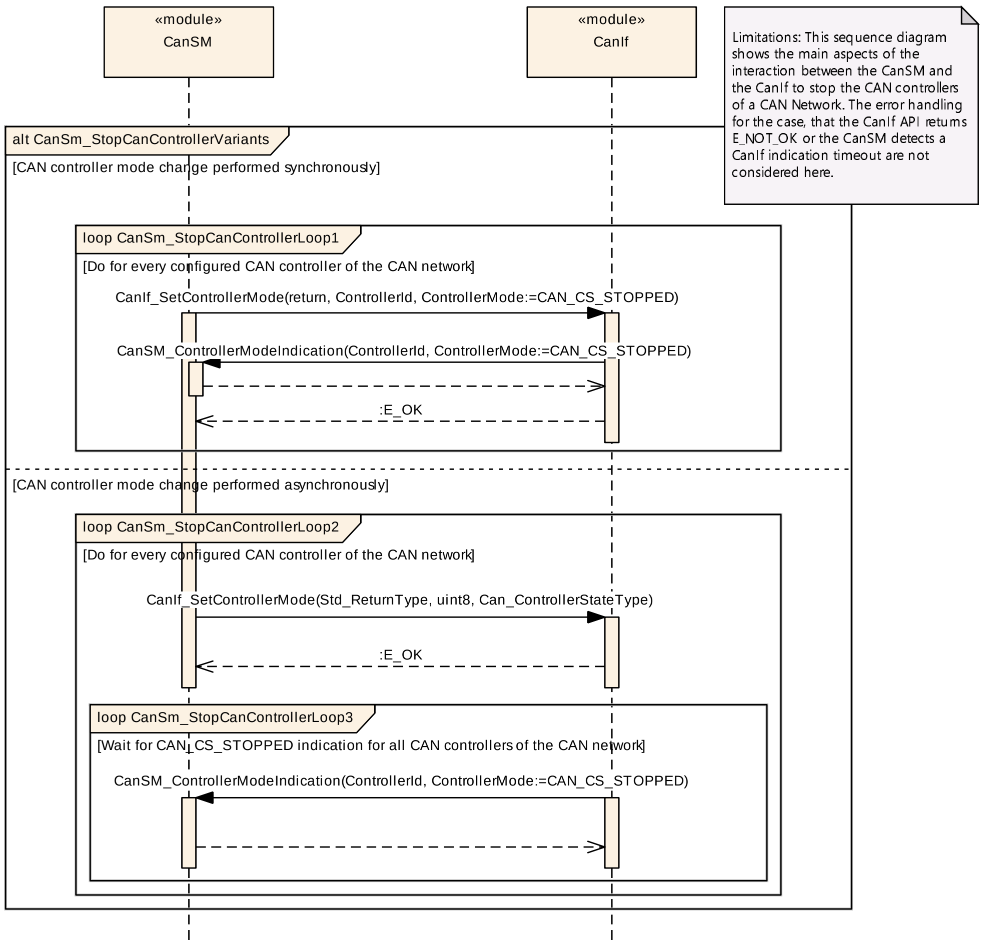
«module»
CanSM
«module»
CanIf
Limitations: This sequence diagram shows the main aspects of the interaction between the CanSM and the CanIf to stop the CAN controllers of a CAN Network. The error handling for the case, that the CanIf API returns E_NOT_OK or the CanSM detects a CanIf indication timeout 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:=CAN_CS_STOPPED)
CanSM_ControllerModeIndication(ControllerId, ControllerMode:=CAN_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, Can_ControllerStateType)
:E_OK
loop CanSm_StopCanControllerLoop3
[Wait for CAN_CS_STOPPED indication for all CAN controllers of the CAN network]
CanSM_ControllerModeIndication(ControllerId, ControllerMode:=CAN_CS_STOPPED)

Figure 9.2: 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

For details refer to the chapter 10.1 “Introduction to configuration specification” in SWS_BSWGeneral.

10.2 Containers and configuration parameters

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

10.2.1 CanSM

SWS Item	**[ECUC_CanSM_00351]**
Module Name	CanSM
Description	Configuration of the CanSM module
Post-Build Variant Support	true
Supported Config Variants	VARIANT-LINK-TIME, VARIANT-POST-BUILD, VARIANT-PRE-COMPILE

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
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

10.2.2 CanSMConfiguration

SWS Item	**[ECUC_CanSM_00123]**
Container Name	CanSMConfiguration
Parent Container	CanSM
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	**[ECUC_CanSM_00335]**		
Parameter Name	CanSMModeRequestRepetitionMax		
Parent Container	CanSMConfiguration		
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	–		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**[ECUC_CanSM_00336]**		
Parameter Name	CanSMModeRequestRepetitionTime		
Parent Container	CanSMConfiguration		
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	–		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

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

10.2.3 CanSMGeneral

SWS Item	**[ECUC_CanSM_00314]**		
Container Name	CanSMGeneral		
Parent Container	CanSM		
Description	Container for general pre-compile parameters of the CanSM module		
Configuration Parameters			

SWS Item	**[ECUC_CanSM_00133]**		
Parameter Name	CanSMDevErrorDetect		
Parent Container	CanSMGeneral		
Description	Switches the development error detection and notification on or off. • true: detection and notification is enabled. • false: detection and notification is disabled.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	**[ECUC_CanSM_00347]**		
Parameter Name	CanSMGetBusOffDelayFunction		
Parent Container	CanSMGeneral		
Description	This parameter configures the name of the <User_GetBusOffDelay> callout function, which is used by CanSM to acquire an additional L1/L2 delay time. This function is only called for channels where CanSMEnableBusOffDelay is enabled.		
Multiplicity	0..1		
Type	EcucFunctionNameDef		
Default value	–		
Regular Expression	–		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	**[ECUC_CanSM_00348]**
Parameter Name	CanSMGetBusOffDelayHeader
Parent Container	CanSMGeneral
Description	This parameter configures the header file containing the prototype of the <User_Get BusOffDelay> callout function.

▽

△

Multiplicity	0..1		
Type	EcucStringParamDef		
Default value	–		
Regular Expression	–		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	**[ECUC_CanSM_00312]**		
Parameter Name	CanSMMainFunctionTimePeriod		
Parent Container	CanSMGeneral		
Description	This parameter defines the cycle time of the function CanSM_MainFunction in seconds		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	]0 .. INF[		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	**[ECUC_CanSM_00344]**		
Parameter Name	CanSMPncSupport		
Parent Container	CanSMGeneral		
Description	Enables or disables support of partial networking. False: Partial Networking is disabled True: Partial Networking is enabled		
Multiplicity	0..1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local dependency: This parameter shall be available only if ComMPncSupport is enabled in ComM		

SWS Item	**[ECUC_CanSM_00343]**		
Parameter Name	CanSMSetBaudrateApi		
Parent Container	CanSMGeneral		
Description	The support of the Can_SetBaudrate API is optional. If this parameter is set to true the Can_SetBaudrate API shall be supported. Otherwise the API is not supported.		
Multiplicity	0..1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

SWS Item	**[ECUC_CanSM_00349]**		
Parameter Name	CanSMTxOfflineActiveSupport		
Parent Container	CanSMGeneral		
Description	Determines whether the ECU passive feature is supported by CanSM. True: Enabled False: Disabled		
Multiplicity	0..1		
Type	EcucBooleanParamDef		
Default value	–		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local dependency: CanIfTxOfflineActiveSupport		

SWS Item	**[ECUC_CanSM_00311]**		
Parameter Name	CanSMVersionInfoApi		
Parent Container	CanSMGeneral		
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	false		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	

▽

△

	Post-build time	–	
Scope / Dependency	scope: local		

No Included Containers

10.2.4 CanSMManagerNetwork

SWS Item	**[ECUC_CanSM_00338]**
Container Name	CanSMController
Parent Container	CanSMManagerNetwork
Description	This container contains the controller IDs assigned to a CAN network.
Configuration Parameters	

SWS Item	**[ECUC_CanSM_00141]**		
Parameter Name	CanSMControllerId		
Parent Container	CanSMController		
Description	Unique handle to identify one certain CAN controller. Reference to one of the CAN controllers managed by the CanIf module.		
Multiplicity	1		
Type	Symbolic name reference to CanIfCtrlCfg		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local dependency: CanIf		

No Included Containers

SWS Item	**[ECUC_CanSM_00126]**
Container Name	CanSMManagerNetwork
Parent Container	CanSMConfiguration
Description	This container contains the CAN network specific parameters of each CAN network
Configuration Parameters	

SWS Item	**[ECUC_CanSM_00131]**	
Parameter Name	CanSMBorCounterL1ToL2	
Parent Container	CanSMManagerNetwork	
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	–	

▽

△

Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**[ECUC_CanSM_00128]**		
Parameter Name	CanSMBorTimeL1		
Parent Container	CanSMManagerNetwork		
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	–		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**[ECUC_CanSM_00129]**		
Parameter Name	CanSMBorTimeL2		
Parent Container	CanSMManagerNetwork		
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	–		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**[ECUC_CanSM_00130]**	
Parameter Name	CanSMBorTimeTxEnsured	
Parent Container	CanSMManagerNetwork	
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, ComTxModeTimePeriod).	
Multiplicity	1	
Type	EcucFloatParamDef	
Range	[0 .. 65.535]	

▽

△

Default value	–		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local dependency: CANSM_BOR_TX_CONFIRMATION_POLLING disabled		

SWS Item	**[ECUC_CanSM_00339]**		
Parameter Name	CanSMBorTxConfirmationPolling		
Parent Container	CanSMManagerNetwork		
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 CanSMBorTimeTx Ensured parameter for this decision.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	**[ECUC_CanSM_00346]**		
Parameter Name	CanSMEnableBusOffDelay		
Parent Container	CanSMManagerNetwork		
Description	This parameter defines if the <User_GetBusOffDelay> shall be called for this network.		
Multiplicity	0..1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	**[ECUC_CanSM_00161]**
Parameter Name	CanSMComMNetworkHandleRef
Parent Container	CanSMManagerNetwork
Description	Unique handle to identify one certain CAN network. Reference to one of the network handles configured for the ComM.
Multiplicity	1
Type	Symbolic name reference to ComMChannel

▽

△

Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local dependency: ComM		

SWS Item	**[ECUC_CanSM_00137]**		
Parameter Name	CanSMTransceiverId		
Parent Container	CanSMManagerNetwork		
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	Symbolic name reference to CanIfTrcvCfg		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local dependency: 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_SetEventStatus in case the corresponding error occurs. The EventId is taken from the referenced DemEventParameter's DemEventId symbolic value. The standardized errors are provided in this container and can be extended by vendor-specific error references.

10.2.5 CanSMDemEventParameterRefs

SWS Item	**[ECUC_CanSM_00127]**
Container Name	CanSMDemEventParameterRefs
Parent Container	CanSMManagerNetwork
Description	Container for the references to DemEventParameter elements which shall be invoked using the API Dem_SetEventStatus in case the corresponding error occurs. The Event Id is taken from the referenced DemEventParameter's DemEventId symbolic value. The standardized errors are provided in this container and can be extended by vendor-specific error references.
Configuration Parameters	

SWS Item	**[ECUC_CanSM_00070]**		
Parameter Name	CANSM_E_BUS_OFF		
Parent Container	CanSMDemEventParameterRefs		
Description	Reference to configured DEM event to report bus off errors for this CAN network.		
Multiplicity	0..1		
Type	Symbolic name reference to DemEventParameter		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local dependency: Dem		

SWS Item	**[ECUC_CanSM_00352]**		
Parameter Name	CANSM_E_MODE_REQUEST_TIMEOUT		
Parent Container	CanSMDemEventParameterRefs		
Description	Reference to configured DEM event to report bus off errors for this CAN network.		
Multiplicity	0..1		
Type	Symbolic name reference to DemEventParameter		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local dependency: Dem		

No Included Containers

10.3 Published Information

For details refer to the chapter 10.3 “Published Information” in SWS_BSWGeneral.

A Not applicable requirements

[SWS_CanSM_00652] ⌈The following requirements are not applicable to this specification, because they are either general BSW requirements, which apply to all BSW modules and not only especially to the CanSM module or they are not applicable at all.⌋*(SRS_BSW_00170, SRS_BSW_00375, SRS_BSW_00395, SRS_BSW_00416, SRS_BSW_00437, SRS_BSW_00168, SRS_BSW_00423, SRS_BSW_00426, SRS_BSW_00427, SRS_BSW_00428, SRS_BSW_00429, SRS_BSW_00432, SRS_BSW_00433, SRS_BSW_00336, SRS_BSW_00417, SRS_BSW_00161, SRS_BSW_00162, SRS_BSW_00005, SRS_BSW_00347, SRS_BSW_00314, SRS_BSW_00353, SRS_BSW_00377, SRS_BSW_00308, SRS_BSW_00309, SRS_BSW_00360, SRS_BSW_00341, SRS_BSW_00439, SRS_BSW_00440, SRS_BSW_00004, SRS_BSW_00006, SRS_BSW_00007, SRS_BSW_00009, SRS_BSW_00010, SRS_BSW_00159, SRS_BSW_00160, SRS_BSW_00164, SRS_BSW_00167, SRS_BSW_00172, SRS_BSW_00300, SRS_BSW_00301, SRS_BSW_00302, SRS_BSW_00305, SRS_BSW_00306, SRS_BSW_00307, SRS_BSW_00310, SRS_BSW_00312, SRS_BSW_00318, SRS_BSW_00321, SRS_BSW_00323, SRS_BSW_00325, SRS_BSW_00327, SRS_BSW_00328, SRS_BSW_00330, SRS_BSW_00331, SRS_BSW_00334, SRS_BSW_00335, SRS_BSW_00339, SRS_BSW_00342, SRS_BSW_00343, SRS_BSW_00346, SRS_BSW_00348, SRS_BSW_00350, SRS_BSW_00357, SRS_BSW_00360, SRS_BSW_00369, SRS_BSW_00373, SRS_BSW_00374, SRS_BSW_00378, SRS_BSW_00379, SRS_BSW_00380, SRS_BSW_00383, SRS_BSW_00384, SRS_BSW_00385, SRS_BSW_00386, SRS_BSW_00388, SRS_BSW_00389, SRS_BSW_00390, SRS_BSW_00392, SRS_BSW_00393, SRS_BSW_00394, SRS_BSW_00396, SRS_BSW_00397, SRS_BSW_00398, SRS_BSW_00399, SRS_BSW_00400, SRS_BSW_00401, SRS_BSW_00402, SRS_BSW_00408, SRS_BSW_00409, SRS_BSW_00410, SRS_BSW_00411, SRS_BSW_00413, SRS_BSW_00415, SRS_BSW_00419, SRS_BSW_00422, SRS_BSW_00438, SRS_BSW_00441, SRS_BSW_00448, SRS_BSW_00449, SRS_BSW_00450, SRS_BSW_00451, SRS_BSW_00452, SRS_BSW_00453, SRS_BSW_00454, SRS_BSW_00456, SRS_BSW_00457, SRS_BSW_00458, SRS_BSW_00459, SRS_BSW_00460, SRS_BSW_00461, SRS_BSW_00462, SRS_BSW_00463, SRS_BSW_00465, SRS_BSW_00466, SRS_BSW_00467, SRS_BSW_00469, SRS_BSW_00470, SRS_BSW_00471, SRS_BSW_00472, SRS_Can_01001, SRS_Can_01002, SRS_Can_01003, SRS_Can_01004, SRS_Can_01005, SRS_Can_01006, SRS_Can_01007, SRS_Can_01008, SRS_Can_01009, SRS_Can_01011, SRS_Can_01013, SRS_Can_01014, SRS_Can_01015, SRS_Can_01016, SRS_Can_01018, SRS_Can_01020, SRS_Can_01021, SRS_Can_01022, SRS_Can_01023, SRS_Can_01027, SRS_Can_01028, SRS_Can_01029, SRS_Can_01032, SRS_Can_01033, SRS_Can_01034, SRS_Can_01035, SRS_Can_01036, SRS_Can_01037, SRS_Can_01038, SRS_Can_01039, SRS_Can_01041, SRS_Can_01042, SRS_Can_01043, SRS_Can_01045, SRS_Can_01049, SRS_Can_01051, SRS_Can_01053, SRS_Can_01054, SRS_Can_01055, SRS_Can_01058, SRS_Can_01059, SRS_Can_01060, SRS_Can_01061, SRS_Can_01062, SRS_Can_01065, SRS_Can_01066, SRS_Can_01068, SRS_Can_01069, SRS_Can_-*

01071, SRS_Can_01073, SRS_Can_01074, SRS_Can_01075, SRS_Can_01076, SRS_Can_01078, SRS_Can_01079, SRS_Can_01081, SRS_Can_01082, SRS_-Can_01086, SRS_Can_01090, SRS_Can_01091, SRS_Can_01095, SRS_Can_-01096, SRS_Can_01097, SRS_Can_01098, SRS_Can_01099, SRS_Can_01100, SRS_Can_01101, SRS_Can_01103, SRS_Can_01107, SRS_Can_01108, SRS_-Can_01109, SRS_Can_01110, SRS_Can_01111, SRS_Can_01112, SRS_Can_-01114, SRS_Can_01115, SRS_Can_01116, SRS_Can_01121, SRS_Can_01122, SRS_Can_01125, SRS_Can_01126, SRS_Can_01129, SRS_Can_01130, SRS_-Can_01131, SRS_Can_01132, SRS_Can_01134, SRS_Can_01135, SRS_Can_-01136, SRS_Can_01138, SRS_Can_01139, SRS_Can_01140, SRS_Can_01141, SRS_Can_01143, SRS_Can_01147, SRS_Can_01148, SRS_Can_01149, SRS_-Can_01151, SRS_Can_01153, SRS_Can_01154, SRS_Can_01155, SRS_Can_-01156, SRS_Can_01157, SRS_Can_01159, SRS_Can_01160, SRS_Can_01161, SRS_Can_01162, SRS_Can_01163, SRS_ModeMgm_00049, SRS_ModeMgm_-09001, SRS_ModeMgm_09009, SRS_ModeMgm_09017, SRS_ModeMgm_-09028, SRS_ModeMgm_09071, SRS_ModeMgm_09072, SRS_ModeMgm_-09078, SRS_ModeMgm_09080, SRS_ModeMgm_09081, SRS_ModeMgm_-09083, SRS_ModeMgm_09084, SRS_ModeMgm_09085, SRS_ModeMgm_-09087, SRS_ModeMgm_09089, SRS_ModeMgm_09090, SRS_ModeMgm_-09097, SRS_ModeMgm_09098, SRS_ModeMgm_09100, SRS_ModeMgm_-09101, SRS_ModeMgm_09102, SRS_ModeMgm_09104, SRS_ModeMgm_-09106, SRS_ModeMgm_09107, SRS_ModeMgm_09109, SRS_ModeMgm_-09110, SRS_ModeMgm_09112, SRS_ModeMgm_09113, SRS_ModeMgm_-09114, SRS_ModeMgm_09115, SRS_ModeMgm_09116, SRS_ModeMgm_-09118, SRS_ModeMgm_09119, SRS_ModeMgm_09120, SRS_ModeMgm_-09122, SRS_ModeMgm_09125, SRS_ModeMgm_09126, SRS_ModeMgm_-09127, SRS_ModeMgm_09128, SRS_ModeMgm_09132, SRS_ModeMgm_-09133, SRS_ModeMgm_09136, SRS_ModeMgm_09141, SRS_ModeMgm_-09143, SRS_ModeMgm_09145, SRS_ModeMgm_09146, SRS_ModeMgm_-09147, SRS_ModeMgm_09149, SRS_ModeMgm_09155, SRS_ModeMgm_-09156, SRS_ModeMgm_09157, SRS_ModeMgm_09158, SRS_ModeMgm_-09159, SRS_ModeMgm_09160, SRS_ModeMgm_09161, SRS_ModeMgm_-09162, SRS_ModeMgm_09163, SRS_ModeMgm_09164, SRS_ModeMgm_-09165, SRS_ModeMgm_09166, SRS_ModeMgm_09168, SRS_ModeMgm_-09169, SRS_ModeMgm_09172, SRS_ModeMgm_09173, SRS_ModeMgm_-09174, SRS_ModeMgm_09175, SRS_ModeMgm_09176, SRS_ModeMgm_-09177, SRS_ModeMgm_09178, SRS_ModeMgm_09179, SRS_ModeMgm_-09180, SRS_ModeMgm_09182, SRS_ModeMgm_09183, SRS_ModeMgm_-09184, SRS_ModeMgm_09185, SRS_ModeMgm_09186, SRS_ModeMgm_-09187, SRS_ModeMgm_09188, SRS_ModeMgm_09189, SRS_ModeMgm_-09190, SRS_ModeMgm_09194, SRS_ModeMgm_09199, SRS_ModeMgm_-09207, SRS_ModeMgm_09220, SRS_ModeMgm_09221, SRS_ModeMgm_-09222, SRS_ModeMgm_09223, SRS_ModeMgm_09225, SRS_ModeMgm_-09226, SRS_ModeMgm_09228, SRS_ModeMgm_09229, SRS_ModeMgm_-09230, SRS_ModeMgm_09231, SRS_ModeMgm_09232, SRS_ModeMgm_-

09234, *SRS_ModeMgm_09235*, *SRS_ModeMgm_09236*, *SRS_ModeMgm_-09237*, *SRS_ModeMgm_09238*, *SRS_ModeMgm_09239*, *SRS_ModeMgm_-09240*, *SRS_ModeMgm_09241*, *SRS_ModeMgm_09243*, *SRS_ModeMgm_-09244*, *SRS_ModeMgm_09245*, *SRS_ModeMgm_09246*, *SRS_ModeMgm_-09247*, *SRS_ModeMgm_09248*, *SRS_ModeMgm_09249*, *SRS_ModeMgm_-09250*, *SRS_ModeMgm_09251*, *SRS_ModeMgm_09253*, *SRS_ModeMgm_-09254*, *SRS_ModeMgm_09255*, *SRS_ModeMgm_09256*, *SRS_ModeMgm_09270*, *SRS_ModeMgm_09271*, *SRS_ModeMgm_09272*, *SRS_ModeMgm_09274*, *SRS_-ModeMgm_09275*, *SRS_ModeMgm_09276*, *SRS_ModeMgm_09277*)