MICROSAR CAN State Manager

Technical Reference

Version 2.9.0

Authors	Mark A. Fingerle
Status	Released

Document Information

History

Author	Date	Version	Remarks
Mark A. Fingerle	2012-08-08	2.0.0	Creation from scratch
Mark A. Fingerle	2012-10-23	2.1.0	ESCAN00062053 Interface to provide internal bus-off recovery level 3.8, Table 3-4, 5.1, 5.2.13 ESCAN00062050 Instruction order for transition to no communication Figure 3-3 Figure 3-5 Update state machine pictures Figure 3-1, Figure 3-2, Figure 3-4
Mark A. Fingerle	2013-05-03	2.2.0	ESCAN00065274 Trigger CanIf PduMode wake up filter in PN use case 6.2.6 Remove chapter “4.2 Include Structure” and “4.3 Compiler Abstraction and Memory Mapping”
Mark A. Fingerle	2013-06-13	2.3.0	ESCAN00068036 SetEcuPassive 0, 5.2.14, 6.2.7 ESCAN00068039 PreventBusSleepAtStartUp 3.13, 5.2.15, 6.2.8
Mark A. Fingerle	2013-08-13	2.4.0	ESCAN00069109 3.11 Baud Rate Adaption ESCAN00068797 3.14 BusOff Recovery Notifications
Mark A. Fingerle	2014-10-13	2.5.0	ESCAN00076768 Post-Build Selectable (Identity Manager) support 6.2.9 ESCAN00076224 Add APIs to Assist EcuM Wakeup Validation 3.15, 5.2.11, 5.2.12 ESCAN00079340 Description BCD-coded return-value of GetVersionInfo() AUTOSAR deviation 6.1
Mark A. Fingerle	2015-11-13	2.6.0	ESCAN00086062 3.10 Swift Tx Timeout Exception
Mark A. Fingerle	2016-01-13	2.7.0	ESCAN00088643 Extended RAM Check 5.2.2, 5.2.16, 5.2.17, 5.4.8, 5.4.9, 5.5.1, 5.5.2, 5.5.3, 5.5.4
Mark A. Fingerle	2016-05-13	2.8.0	ESCAN00090185 Wakeup validation fail (Start/Stop wakeup sources); Wakeup validation must not be used with asynchronous Trcv (SPI) 5.2.11 5.2.12 ESCAN00090829 Improve description how to redirect "Error Reporting APIs" 3.17.1, 3.17.2
Mark A. Fingerle	2016-08-01	2.9.0	ESCAN00091303 6.2.13 Expanded Tx Timeout Exception Handling

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	Specification of CAN State Manager	2.2.0
[2]	AUTOSAR	Specification of Development Error Tracer	3.2.0
[3]	AUTOSAR	Specification of Diagnostics Event Manager	4.2.0
[4]	AUTOSAR	List of Basic Software Modules	1.6.0
[5]	AUTOSAR	Specification of CAN Interface	5.0.0
[6]	AUTOSAR	Specification of Communication Manager	4.0.0
[7]	AUTOSAR	Specification of Basic Software Mode Manager	1.2.0

Scope of the Document

This technical reference describes the general use of the CAN State Manager basis software. All aspects which are CAN controller specific are described in the technical reference of the CAN Interface, which is also part of the delivery.

Caution
We have configured the programs in accordance with your specifications in the questionnaire. Whereas the programs do support other configurations than the one specified in your questionnaire, Vector's release of the programs delivered to your company is expressly restricted to the configuration you have specified in the questionnaire.

Contents

Illustration List

Tables

1 Component History

The component history gives an overview over the important milestones that are supported in the different versions of the component.

Component Version	New Features
2.0.0	Creation according to AUTOSAR 4.0.3
5.1.0	Extended RAM Check

Table 1-1 Component history

2 Introduction

This document describes the functionality, API and configuration of the AUTOSAR BSW module CanSM as specified in [1].

Supported AUTOSAR Release*:	4	
Supported Configuration Variants:	pre-compile, Post-Build Selectable	
Vendor ID:	CANSM_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	CANSM_MODULE_ID	140 decimal (according to ref. [4])

* For the precise AUTOSAR Release 4.x please see the release specific documentation.

The CAN State Manager (CanSM) realizes a software layer between the Communication Manager (ComM) and the CAN Interface (CanIf). The CanSM handles the startup and shutdown of the communication of a CAN network. The CAN State Manager maps the CAN State Manager states to the states of the ComM and causes the necessary actions to change the CAN State Manager state to those requested by the ComM. The main function of the CAN State Manager is called cyclically by the Schedule Manager (SchM).

2.1 Architecture Overview

The following figure shows where the CanSM is located in the AUTOSAR architecture.

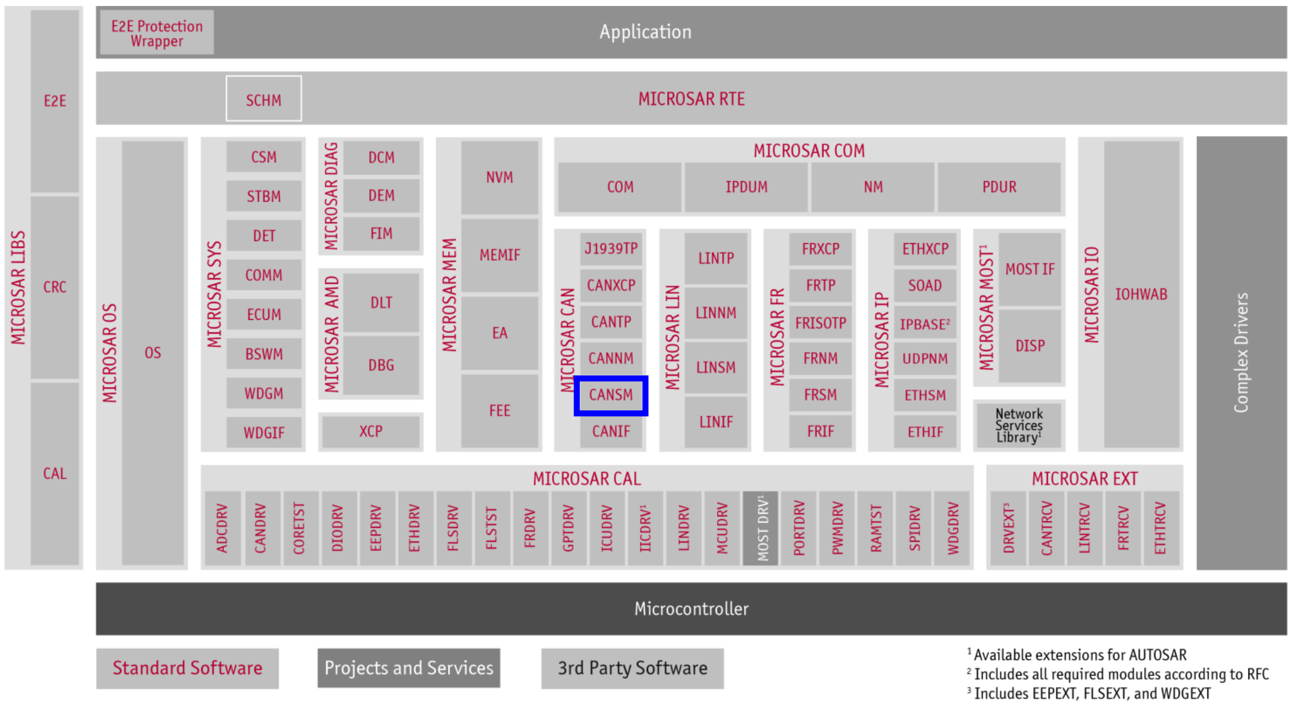

Figure 2-1 AUTOSAR architecture

The next figure shows the interfaces to adjacent modules of the CanSM. These interfaces are described in chapter 5.

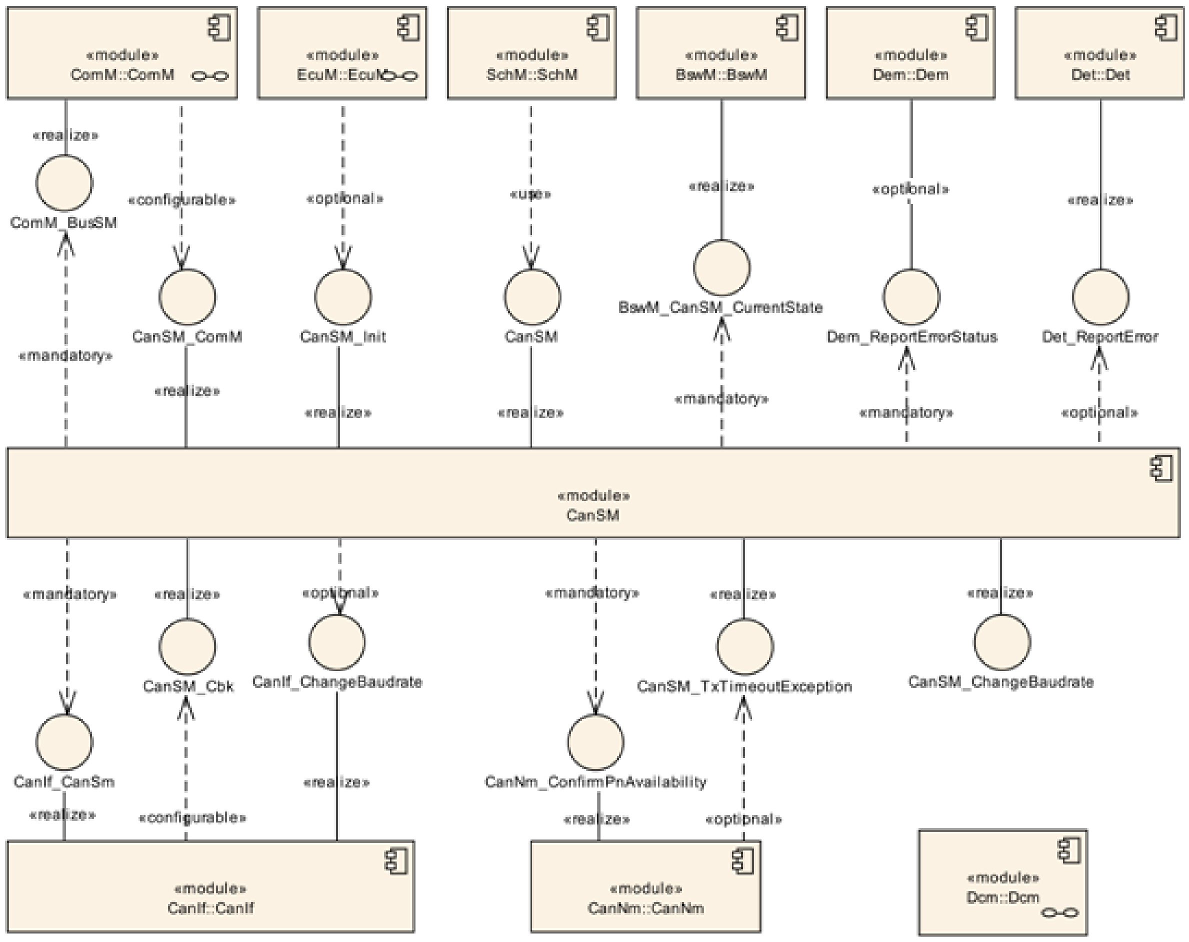

Figure 2-2 Interfaces to adjacent modules of the CanSM

Applications do not access the services of the BSW modules directly.

3 Functional Description

3.1 Features

The features listed in the following tables cover the complete functionality specified for the CanSM.

The AUTOSAR standard functionality is specified in [1], the corresponding features are listed in the tables

> Table 3-1 Supported AUTOSAR standard conform features

> Table 3-2 Not supported AUTOSAR standard conform features

For further information of not supported features see also chapter 6.

Vector Informatik provides further CanSM functionality beyond the AUTOSAR standard. The corresponding features are listed in the table

> Table 3-3 Features provided beyond the AUTOSAR standard

The following features specified in [1] are supported:

Supported AUTOSAR Standard Conform Features
Translation of network communication mode requests
Output of current network communication modes (Polling and Callback)
Control of peripherals (CAN Transceivers, CAN Controllers)
Control of PDU mode
Handle the network mode via a separate state machine per network
Bus error management: Bus-off recovery via a separate state machine per network
Change Baud Rate handling
Tx Timeout Exception handling
Error classification, detection and notification
Enable and disable development and production error detection

Table 3-1 Supported AUTOSAR standard conform features

The following features specified in [1] are not supported:

Category	Description	ASR Version
Functional	Several controllers per network.	4.0.3
Config	Change networks and controllers via Post-build configuration.	4.0.3
Config	Configuration variants “link-time”.	4.0.3

Table 3-2 Not supported AUTOSAR standard conform features

The following features are provided beyond the AUTOSAR standard:

Features Provided Beyond The AUTOSAR Standard
Deactivate the DEM at pre-compile time, like DET.
Changes of the communication mode are possible even during a pending mode indication.
Handling of bus-off events which occurs after `CANSM_FULL_COMMUNNICATION` has been left.
Interface to provide internal bus-off recovery level; CanSM_CheckBorLevel()
PduMode wake up filter in PN use case
Execute transition from SILENT to FULL within RequestComMode
ECU active/passive mode functionality
Prevent the bus sleep state of the CanIf, CanDrv and CanTrcv at CanSM initialization for the given CAN network handle
MICROSAR Identity Manager using Post-Build Selectable
Extended RAM Check

Table 3-3 Features provided beyond the AUTOSAR standard

3.2 Initialization

Some embedded targets do not initialize RAM to zero during start-up. Therefore some variables have to be initialized explicitly if they need a specific value before the initialization function `CanSM_Init` is called. This is done by the function `CanSM_InitMemory`. The function initializes the CanSM variables and sets the state to ‘not initialized’. The function has to be called before the initialization function `CanSM_Init`.

After that, the initialization `CanSM_Init` has to be triggered and the CAN State Manager will set the internal used variables to their start values to ensure a deterministic behavior of the state machines.

Info
The CanSM initializes the CAN channel into the state NO COMMUNICATION. This means, the CAN modules (CanIf, CanDrv and CanTrcv) are set into the corresponding state for NO COMMUNCIATION (bus sleep). During this transition, detected wake-up reasons, inside the CAN modules, are cleared.
This leads to the behavior that wake-up events, which are triggered by the CAN bus, cannot be detected and/or validated during the initialization phase.
If the detection/validation of the wake up information is necessary for the ECU then the CanSM API CanSM_PreventBusSleepAtStartUp() can be used to prevent the bus sleep mode at start up for the above listed CAN modules.

3.3 State Machine

The CanSM functionality cannot be used before the API function `CanSM_Init` has been called. If the `CanSM_Init` function is executed successfully the CanSM starts the transition to the state `CANSM_NO_COMMUNNICATION`.

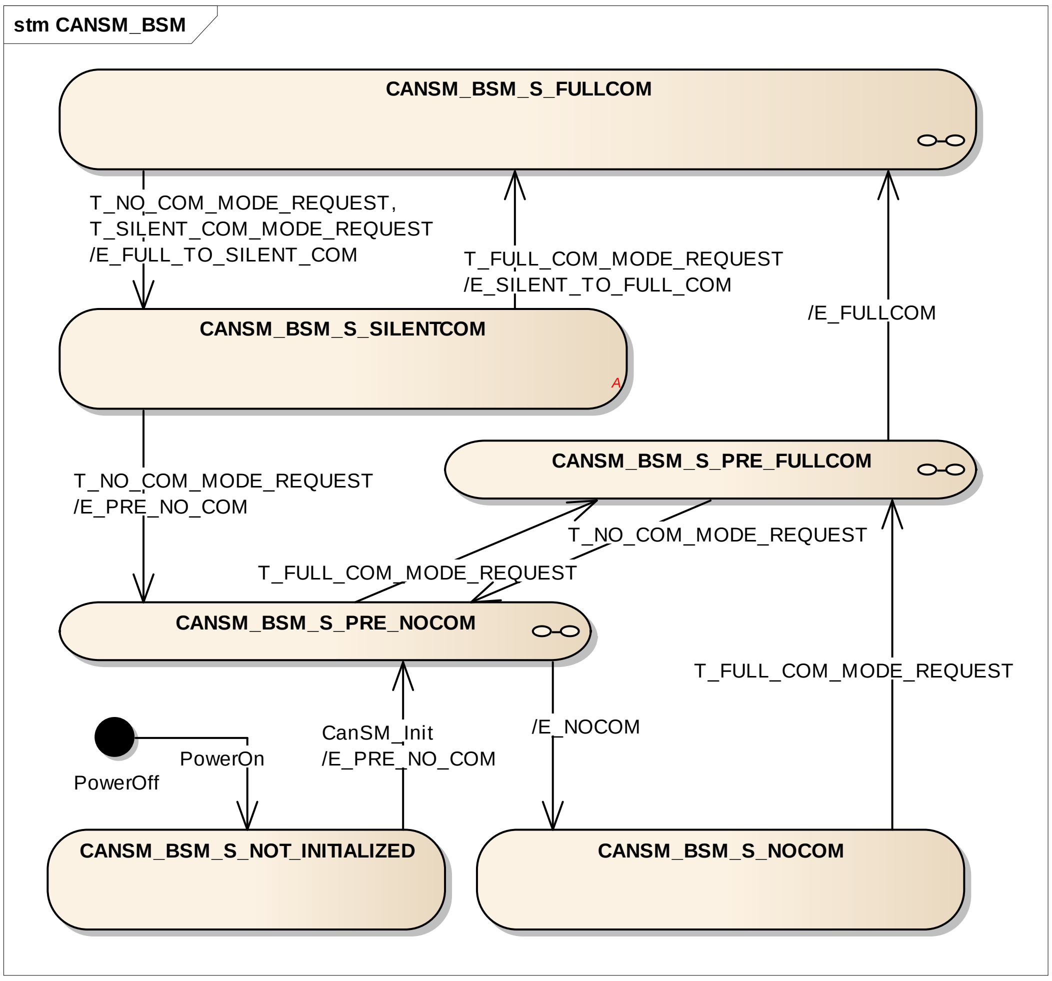

Figure 3-1 CanSM state machine

3.3.1 Mode Request Indication and Repetition

If the CanSM triggers the transceiver or the controller, the CanSM waits for the corresponding indication that the requested mode is reached. If the function call returns E_NOT_OK and the corresponding indication is missing, the CanSM repeats the request in the next main cycle. The CanSM repeats a controller/transceiver mode request also if the correct mode indication is not received within the `CanSMModeRequestRepetitionTime`.

Each repetition is counted and if the amount exceeds the value `CanSMModeRequestRepetitionMax`, the counter is reset, the Det will be informed with `E_MODE_REQUEST_TIMEOUT` and the CanSM performs the transition to `CANSM_NO_COMMUNICATION`. The repetition counter is also reset if the desired final state is reached.

3.3.2 Communication Mode Request Change (During Pending Mode Indication or Running Bus-Off Recovery)

If the state machine reachs a sub state and a changed mode request is present, the state machine changes immediately the “current direction” to reach the desired communication mode. The CanSM ensures that the controller and transceiver are set to the corresponding mode. Therefore the CanSM performs always the whole sub-state machine, so if e.g. a startup is skipped by a NoCom request the CanSM changes the controller mode, too, even if it has not been changed and it is still `STOPPED`.

Exception:

`COMM_SILENT_COMMUNICATION` request can NOT be requested if

- The transition (from SILENT or after initialization) to `CANSM_NO_COMMUNNICATION` has been started
- The CanSM is in state `CANSM_SILENT_COMMUNNICATION`
- The CanSM is in state `CANSM_NO_COMMUNNICATION`

3.3.3 CANSM_NO_COMMUNICATION to CANSM_FULL_COMMUNICATION

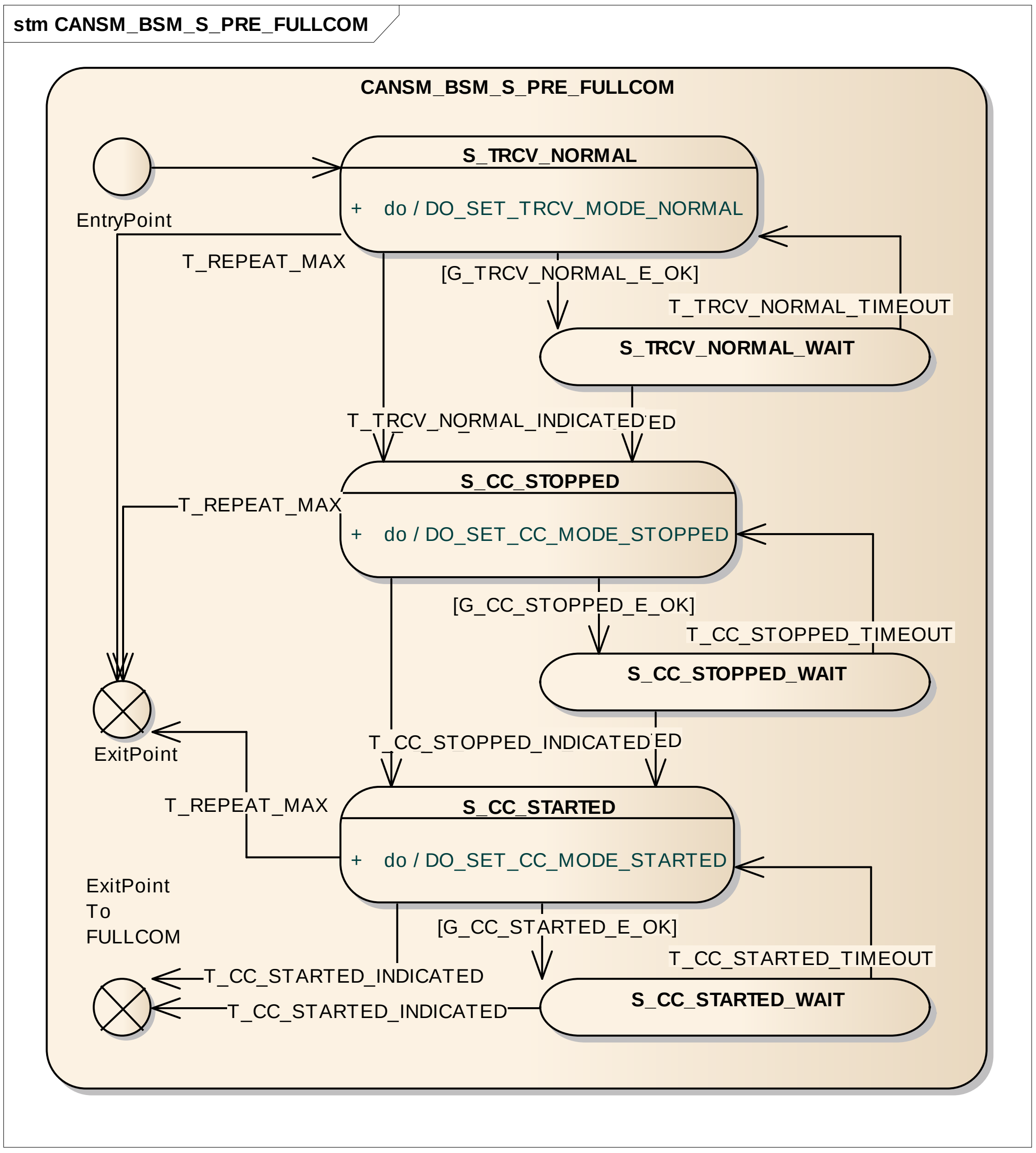

Figure 3-2 Sub state transition to CANSM_FULL_COMMUNICATION

In this state there is no communication on the CAN channel. When full communication is requested the CanSM sets the transceiver mode to `NORMAL` and the controller mode to

`STARTED` (via `STOPPED`). In case of a successful transition the CanSM sets the Rx and Tx Pdu Mode to `ONLINE`, informs the ComM (see [6]) and the BswM (see [7]) about the new communication state and starts the “ensure timer”. If the `CanSMBorTimeTxEnsured` lapse without a bus-off indication the CanSM informs the Dem that no bus-off is present. Alternatively to the “ensure timer” the CanSM may poll the TxState to decide that no bus-off is present if `CanSMBorTxConfirmationPolling` is activated.

Caution
This chapter describes only the normal shutdown. In case a partial network is activated the CanSM performs an alternative sequence which is described in chapter 3.9.

3.3.4 CANSM_FULL_COMMUNICATION to CANSM_SILENT_COMMUNICATION

As long as full communication is requested the CanSM stays in this state, otherwise the CanSM switches to silent mode and stops the Tx PDU mode. In case of a successful transition the CanSM notifies the ComM and BswM about the `CANSM_SILENT_COMMUNICATION` communication state.

3.3.5 CANSM_SILENT_COMMUNICATION

The state represents the prepare bus sleep phase of the network. The node is still able to receive CAN messages but does not transmit them.

3.3.6 CANSM_SILENT_COMMUNICATION to CANSM_FULL_COMMUNICATION

According to the requested communication mode the CanSM switches back to `CANSM_FULL_COMMUNNICATION`, starts the Tx PDU mode and notifies the ComM and BswM about the new communication state.

3.3.7 Transition to CANSM_NO_COMMUNICATION

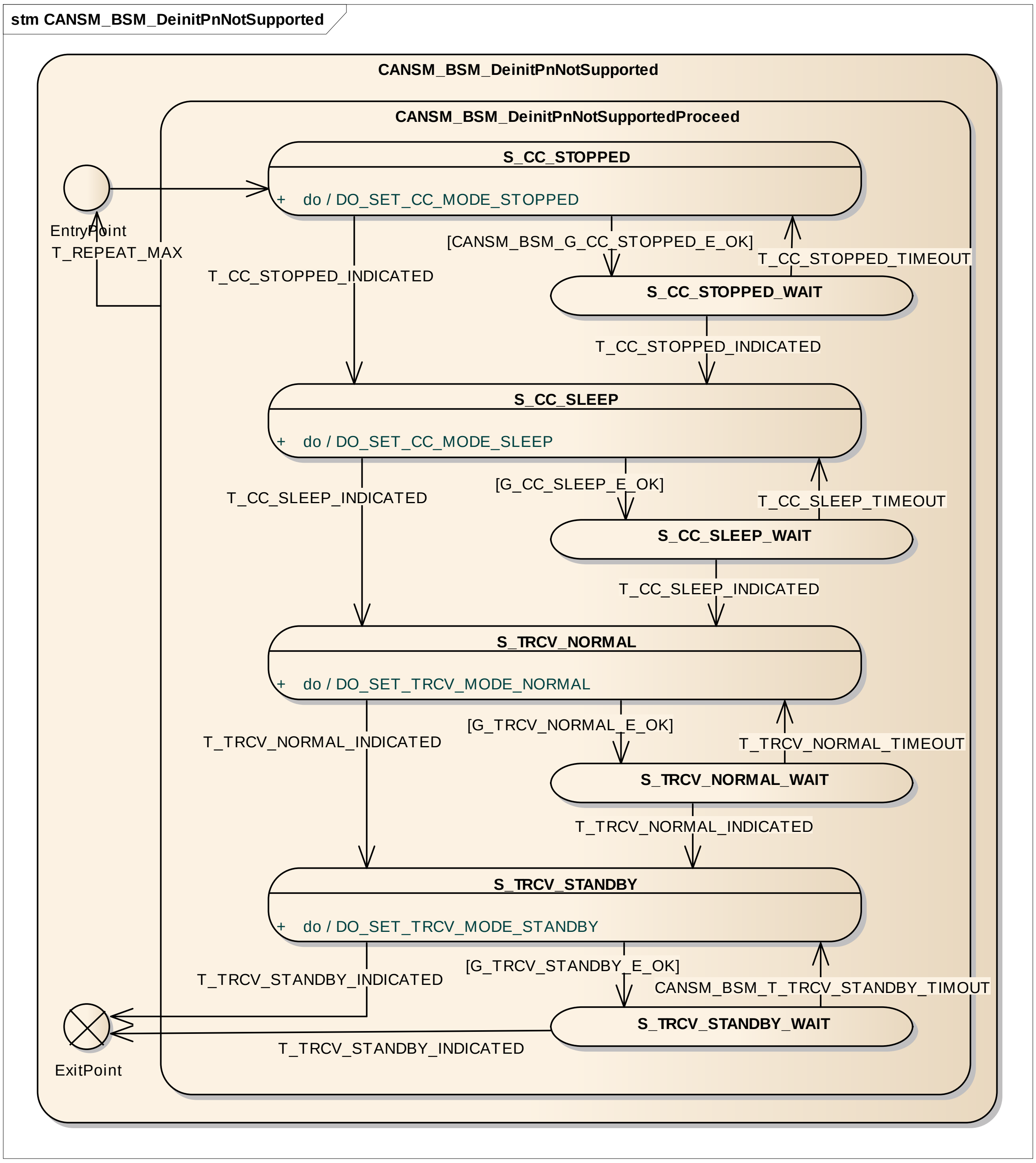

Figure 3-3 Sub state transition to CANSM_NO_COMMUNICATION

The CanSM informs the BswM about the communication `CANSM_NO_COMMUNICATION` immediately if the transition shutdown process has been started. According to the requested communication mode the CanSM switches to `CANSM_NO_COMMUNICATION`. Then the CanSM sets the controller to `SLEEP` (via `STOPPED`) and the transceiver to `STANDBY` (via `NORMAL`). In case of a successful transition the CanSM informs the ComM about the new communication state (if this transition is executed in the call context of

CanSM_Init the ComM and BswM functions are not called because these modules will be initialized after the CanSM).

3.4 Bus-Off Recovery

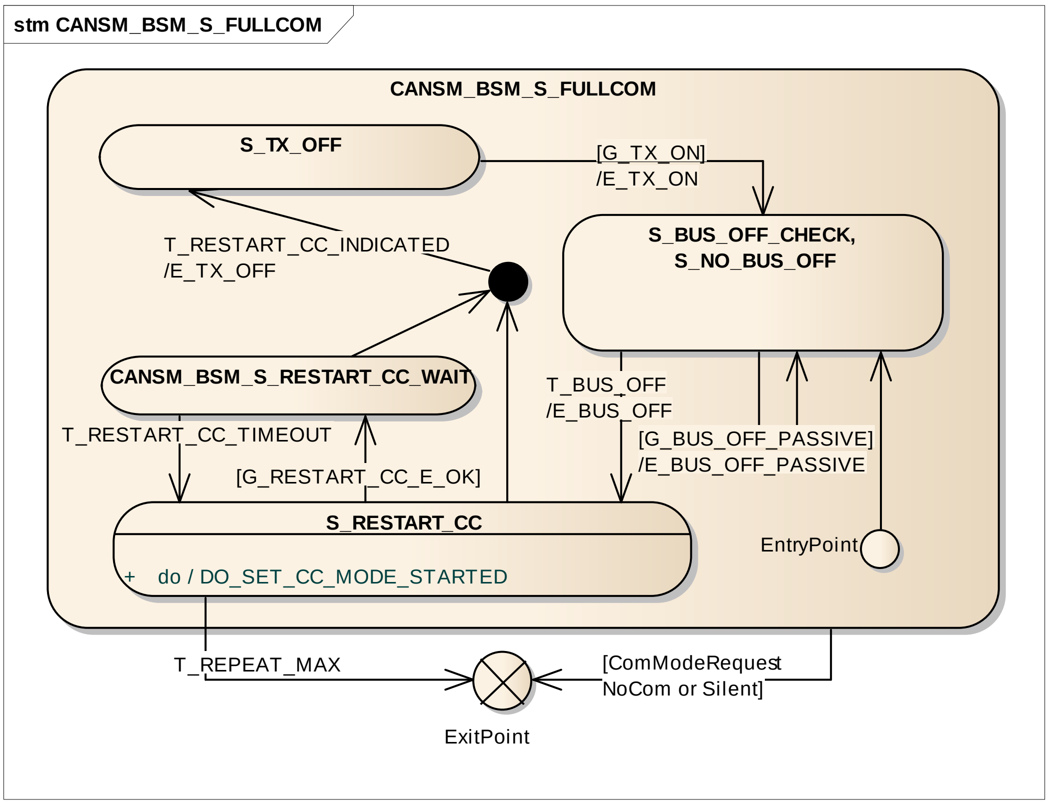

Figure 3-4 CanSM sub-state bus-off recovery

In case bus-off is indicated the CanSM informs the Dem (E_BUSOFF and EVENT_STATUS_PREFAILED), the ComM (SILENT) and BswM (BUSOFF). In the next step the CanSM restarts the controller to STARTED mode. If the according mode indication is received the CanSM sets the Rx Pdu Mode to ONLINE and Tx Pdu Mode to OFFLINE and starts the bus-off timer. If the CanSMBorTimeL1 (or CanSMBorTimeL2 if the bus-off count is equal or greater than CanSMBorCounterL1ToL2) elapse CanSM reactivates the Tx path of the channel again, informs the ComM (FULL) and BswM (FULL) and starts the “ensure timer”. If the CanSMBorTimeTxEnsured timer has elapsed without a bus-off indication the CanSM informs the Dem, otherwise the next bus-off recovery sequence is started. The “ensure timer” can also substituted by polling the TxState if CanSMBorTxConfirmationPolling is activated as mentioned above.

Note
The indicated Dem event does not instantly lead to a DTC due to the EventStatus pre-failed. The mechanism to qualify the event as failed has to be configured within the DEM [3].

3.5 Main Function

The CanSM has one main function which has to be called cyclically by the SchM. The main function triggers a state transition in case of a received mode indication or if a timer elapses.

3.6 Communication Modes

The ComM collects the communication requests from the SWC and from the network. Accordingly the ComM calculates the needed communication mode and requests this from the CAN State Manager via the function `CanSM_RequestComMode`.

3.7 Communication Mode Polling

The ComM is informed about every mode change by the CAN State Manager via the callback function `ComM_BusSM_ModeIndication`.

Additional the ComM may request the communication mode which is currently active by calling the API function `CanSM_GetCurrentComMode`. The CAN State Manager will deliver the communication mode to the pointer passed as a function parameter.

3.8 Bus-off Level Polling

The current bus-off level can be determinate by calling the API function `CanSM_CheckBorLevel`. The CanSM will deliver the bus-off level (`CANSM_BOR_NONE`, `CANSM_BOR_LEVEL1` or `CANSM_BOR_LEVEL2`) to the pointer passed as a function parameter.

3.9 Partial Networking

If Partial Networking for a CAN channel is activated the CAN transceiver can only be woken up by a specified CAN Message. Also the Network Management will ignore NM messages which do not belong to the Partial Network and the CanSM will perform an alternative shutdown sequence.

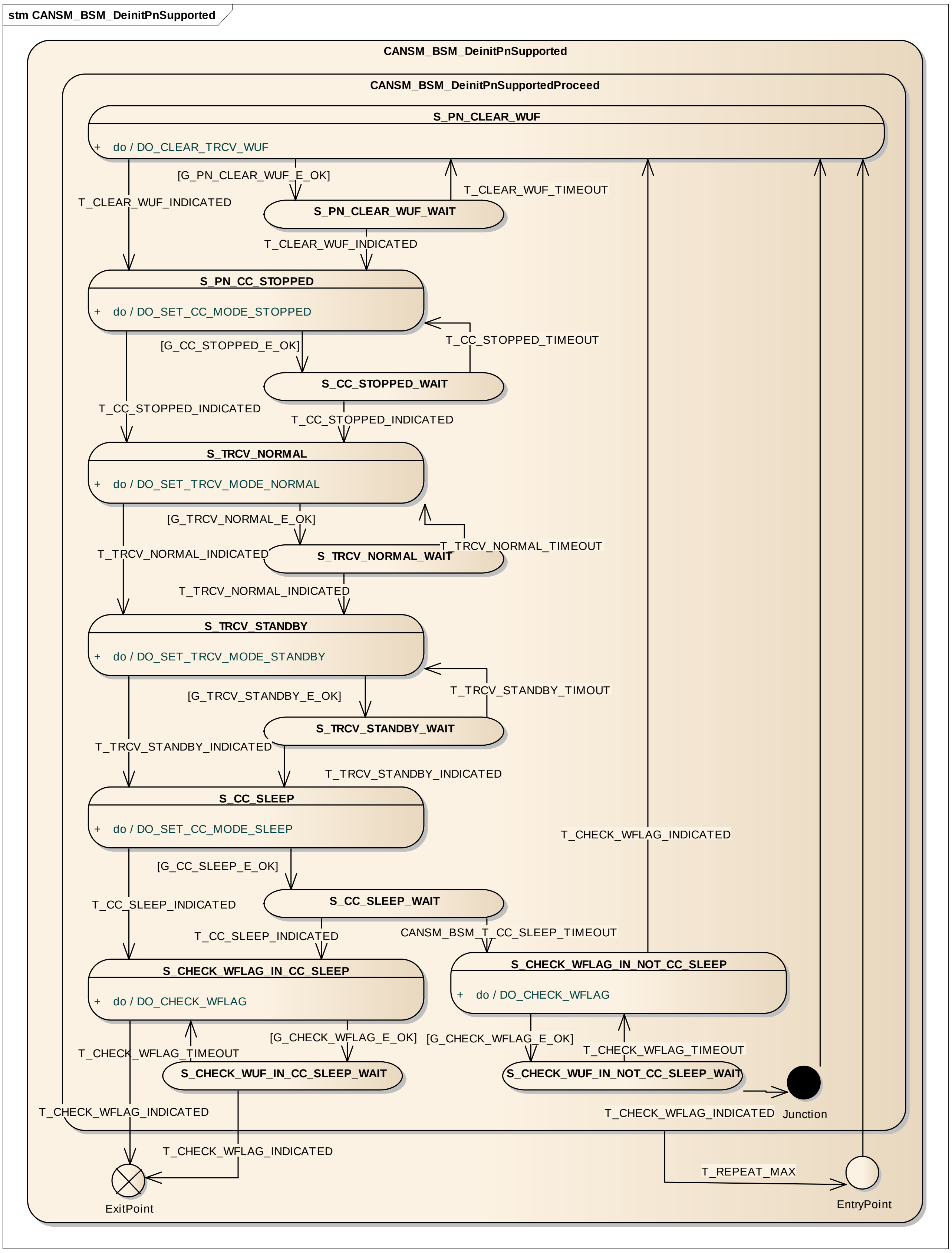

Figure 3-5 Sub state Partial Network transition to CANSM_NO_COMMUNICATION

If the feature has been enabled globally (at pre-compile time) and on the desired channel, the CanSM first resets the current available wake-up information in the transceiver, before the transceiver is set to `STANDBY` and the controller to `SLEEP`. If this is done, the CanSM triggers the function `CanIf_CheckTrcvWakeFlag` to handle a wake-up which might have occurred during the shutdown. If an API call does not deliver the expected reaction it will called again as described in chapter 3.3, subchapter "Mode Request Indication and Repetition". But the absence of the controller `STOPPED` indication has an exceptional nature and does not lead to a repetition. Instead of the repetition the `CheckTrcvWakeFlag` will be triggered and the whole shutdown sequence will be repeated from start after the `CanSM_CheckTransceiverWakeFlagIndication` has been received.

3.10 Tx Timeout Exception

If the CanSM gets the `CanSM_TxTimeoutException` notification the CanSM performs the transition to `CANSM_NO_COMMUNICATION`, except bus-off is active. In this case the `CanSM_TxTimeoutException` notification will be ignored because it is quite likely a "false report" due to the TxOffline phase and the communication will work again after that and if not, the "Tx Timeout Exception" will be indicated by the CanNm again anyway.

If a "Tx Timeout Exception" handling is running any incoming communication mode request will be postponed until `CANSM_NO_COMMUNICATION` has been reached. After that the transition to `CANSM_FULL_COMMUNICATION` will be started if the last requested communication mode was `COMM_FULL_COMMUNICATION` or `COMM_SILENT_COMMUNICATION`.

In addition the CanSM provides an abbreviated recovery mechanism. If the feature `CanSMSwiftTxTimeoutRecovery` is activated, only the conroller is set to `STOPPED` and back to `STARTED`, instead of executing the entire shutdown and start up sequence. If it was not successful to set the controller back to `STARTED` within the first try the CanSM indicates `COMM_SILENT_COMMUNICATION` to the ComM and `CANSM_BSWM_NO_COMMUNICATION` to the BswM and executes the stanard repetition mechanism to reach the needed controller mode.

3.11 Baud Rate Adaption

The adaption of the baud rate is started by calling the function `CanSM_SetBaudrate` (or `CanSM_ChangeBaudrate)`. A Baud Rate Change is only possible if the communication state is `COMM_FULL_COMMUNICATION` and no bus-off is present (validated by "Tx ensured time" or "Tx Confirmation").

When the Baud Rate Change has been accepted the CanSM informs the BswM (CHANGE_BAUDRATE), set the PDU mode to `OFFLINE` and the controller mode to `STOPPED`. After the controller mode `STOPPED` is reached the CanSM informs the ComM (NoCom) and lead the driver to set the new baud rate. Then the controller mode will be set back to `STARTED`. After the controller mode `STARTED` is reached the CanSM set the PDU mode to `ONLINE`.

Note
The feature is intended to be used by the Dcm module.

Caution
The `CanSM_ChangeBaudrate` API is deprecated. So it is recommended to use the `CanSM_SetBaudrate` API instead.

`CanSM_SetBaudrate` API and `CanSM_ChangeBaudrate` API cannot be provided simultaneously.

If `CanSM_ChangeBaudrate` API is used nevertheless the desired baud rate has to be validated via the function `CanSM_CheckBaudrate` before the function `CanSM_ChangeBaudrate` will be called.

3.12 ECU Passive Mode

After the initialization of the CanSM the ECU mode is active per default. The ECU mode is the same for each CAN channel.

The CanSM can be instructed to handle the passive or active mode, globally for all channels via the API CanSM_SetEcuPassive(). The mode stays until a new request is issued or a (re-)initialization of the CanSM happens.

In passive mode the CanSM sets the Tx PDU mode to OFFLINE_ACTIVE instead to ONLINE (3.3.6, 3.3.3). If the ECU mode switches from passive to active the CanSM switches the Tx PDU modes which are in OFFLINE_ACTIVE to ONLINE.

During a bus-off recovery phase the modification of the Tx PDU mode is postponed until the bus-off recovery phase has been finished (Ch 3.4, Figure 3-4 E_TX_ON).

3.13 PreventBusSleepAtStartUp

If the feature is enabled within the configuration tool the function CanSM_PreventBusSleepAtStartUp() becomes available. The function, if called before the initialization, causes the CanSM to skip the initial transition of the according CAN channel. Usually the CanSM sets the controller to sleep mode and the transceiver to standby during the initialization.

Note
The CanSM expects that a FULL_COMMUNICATION request follows after the function has been used and so the CanSM performs no further actions.

Caution
If CanSM_PreventBusSleepAtStartUp() is used the CanDrv and CanTrcv stay in their initial state and so usually no CAN wake-ups are possible.

3.14 BusOff Recovery Notifications Extension of Tx Offline Duration

The feature gives the application the possibility to react on an active bus-off. If the feature is activated the CanSM triggers the "bus-off begin indication function" immediately, each time the CanSM is informed about a bus-off. The second parameter of the function can be used to extend the "bus-off recovery time" `(TxOffline)` (from 0 up to 153ms which is the maximum value needed by the J1939Nm).

When the CanSM enters the state `S_BUS_OFF_CHECK`, the Tx path is restarted. The communication should work again and the CanSM informs the application via the "bus-off end indication function". The according channel can be identified via the network handle, which is the first parameter of both functions.

The name of the indication functions can be set within the configuration tool. If the indication function is not needed delete the function name (empty string) or delete the parameter. Both functions can be (de)activated separately.

If J1939Nm is used, both the begin (`J1939Nm_GetBusOffDelay`) and end (`J1939Nm_BusOffEnd`) indications are required.

3.15 Wake-up Validation Assistance

3.16 Start/Stop Wake-up Sources

With the new APIs (5.2.11, 5.2.12) the CanSM can be used, to start and stop the wake-up sources, to enable the wake-up validation. Thus it can be avoided that the EcuM callout starts the wake-up sources while the CanSM performs the transition to no communication or the EcuM callout stops the wake-up sources while the CanSM performs the transition to full communication.

3.16.1 Normal Behavior

Usually the CanSM is informed about the start of the wake-up validation sequence (via 5.2.11) within the state `CANSM_NO_COMMUNICATION`. In this case the CanSM sets the CAN controller to STARTED and the CAN transceiver to NORMAL. If the validation is successful it will be finished by a full communication request, then the Pdu mode is set to ONLINE and the ComM and the BswM are notified with the corresponding full communication indication.

The validation is also finished if the wake-up has not been determined as valid within the specified validation time. Then the CanSM is informed by the according API (5.2.12) and the CanSM switches the controller back to SLEEP and the CAN transceiver to STANDBY. If a validation sequence is started while the CanSM performs a transition to `CANSM_NO_COMMUNICATION`, the current transition to `CANSM_NO_COMMUNICATION` will be canceled.

3.16.2 Usage

To use the wake-up validation assistance of the CanSM, remove the "set controller mode" and "set transceiver mode" functions from the EcuM wake-up sources callouts, call `CanSM_StartWakeupSources` instead within the EcuM callout `EcuM_StartWakeupSources` and the `CanSM_StopWakeupSources` within the EcuM callout `EcuM_StopWakeupSources`. Pay also attention to 4.2.

3.16.3 Exceptional Behavior

The change of the CAN HW mode could be disturbed and is not possible within the HW loop timeout. Especially the change of the controller mode may fail due to message reception, dominant voltage level or electromagnetic interference.

If any transceiver or controller mode change returns E_NOT_OK any further actions will be omitted and the CanSM will return E_NOT_OK too; except if the set controller mode to SLEEP is answered with E_NOT_OK. In this case CanSM triggers a new wake-up by the EcuM, which will start a new wake-up validation sequence. So no further exceptional actions are necessary and the CanSM `StopWakeupSources` returns E_OK.

In case the CanSM returns an E_NOT_OK the CanTrcv/CanDrv are in "undefined" state so it is most likely not possible to react on any event on the CAN bus respectively no Rx, no Tx or no wake-up is possible which can lead to the effects described in the following chapter.

3.16.4 Potential Effect

3.16.4.1 Start of the Wakeup Sources Fail

Because of the disturbance during the mode change the CAN HW (controller and/or transceiver) might be in an undefined state and is probably not able to react on incoming messages. Messages on the bus are lost, until a new wake-up is possible, after the validation timeout elapses and a successful call of `StopWakeupSources`.

3.16.4.2 Stop of the Wakeup Sources Fail

Because of the disturbance during the mode change the CAN HW (controller and/or transceiver) might be in an undefined state. Probably the CAN wake-up will not work and the ECU is not able to react on Rx messages on the affected CAN bus.

Caution
The EcuM may perform a state change to stop/sleep in the same `EcuM_MainFunction()` cycle where `EcuM_StopWakeupSources()` is called. So it is possible that the ECU stays in low power mode and cannot be woken up again (internal/external wake-up or wake-up by CAN).

3.16.5 Countermeasures

> A short disturbance can probably be resolved by calling Start-/StopWakeupSources() within the current call context again.

> As a second approach the return value of StartWakeupSources could be ignored. As a result the validation time elapses, the wake-up sources are stopped and a new wake-up interrupt triggers the validation again, if the CAN communication is still running. As a drawback, the ECU cannot participate in the CAN communication during this period and therefore is not recommended for time critical systems.

> Furthermore, the validation procedure can be bypassed altogether. Instead of calling `CanIf_CheckValidation()`, the wake-up can be validated "manually" by calling `EcuM_ValidateWakeupEvent()` directly. As a result, normal CAN communication is started on the channel.

Note: This may also lead to a wake-up of other ECUs on the affected CAN channel, due to the electromagnetic interference, because of inhibited wake-up validation.

> If the `StopWakeupSources()` fails the validation sequence could be restarted again “manually” via `EcuM_SetWakeupEvent()` call. The ECU can react faster to a potential running CAN communication, under the assumption that the `StartWakeupSources()` will be executed successfully. Alternatively it is possible to initiate an ECU reset. The whole CAN stack becomes reinitialized by the BSW modules from scratch.

Note
The appropriate solution depends highly on the type of the ECU and on the requirements which have to be fulfilled by the ECU.

Caution
If any one of the functions `CanSM_StartWakeupSources()` 5.2.11 or `CanSM_StopWakeupSources` 5.2.12 returns a failure (i.e. returns `E_NOT_OK`) the application has to perform an ECU dependent error handling.

Note
Wakeup validation does not work with asynchronous hardware e.g. partial network transceiver.

3.17 Error Handling

3.17.1 Development Error Reporting

By default, development errors are reported to the DET using the service `Det_ReportError()` as specified in [2], if development error reporting is enabled (i.e. pre-compile parameter `CANSM_DEV_ERROR_DETECT == STD_ON`).

If another module is used for development error reporting, the function prototype for reporting the error can be configured by the integrator, but must have the same signature as the service `Det_ReportError()`. The redirection of the function name has to be done via “User Config File”.

The reported CanSM ID is 140.

The reported service IDs identify the services which are described in 5.2. The following table presents the service IDs and the related services:

Service ID	Service
`0x00`	`CanSM_Init`
`0x01`	`CanSM_GetVersionInfo`
`0x02`	`CanSM_RequestComMode`
`0x03`	`CanSM_GetCurrentComMode`
`0x04`	`CanSM_ControllerBusOff`
`0x05`	`CanSM_MainFunction`
`0x06`	`CanSM_ConfirmPnAvailability`
`0x07`	`CanSM_ControllerModeIndication`
`0x08`	`CanSM_ClearTrcvWufFlagIndication`
`0x09`	`CanSM_TransceiverModeIndication`
`0x0A`	`CanSM_CheckTransceiverWakeFlagIndication`
`0x0B`	`CanSM_TxTimeoutException`
`0x0C`	`CanSM_CheckBaudrate`
`0x0E`	`CanSM_ChangeBaudrate`
`0x0D`	`CanSM_SetBaudrate`
`0x0F`	`CanSM_CheckBorLevel`
`0x40`	`CanSM_PreventBusSleepAtStartUp`

Service ID	Service
`0x20u`	`CanSM_StartWakeupSources`
`0x21u`	`CanSM_StopWakeupSources`

Table 3-4 Service IDs

The errors reported to DET are described in the following table:

Error Code		Description
`0x01`	`CANSM_E_UNINIT`	API service used without having called the initialization function.
`0x02`	`CANSM_E_PARAM_POINTER`	API service called with invalid pointer in parameter list
`0x03`	`CANSM_E_INVALID_NETWORK_HANDLE`	API service called with wrong network handle parameter, which is not configured in the CanSM configuration.
`0x04`	`CANSM_E_PARAM_CONTROLLER`	API service called with wrong controller index.
`0x05`	`CANSM_E_PARAM_TRANSCEIVER`	API service called with wrong transceiver index.
`0x06`	`CANSM_E_BUSOFF_RECOVERY_ACTIVE`	API network mode request called during not finished bus-off recovery
`0x07`	`CANSM_E_WAIT_MODE_INDICATION`	API network mode request called during pending indication
`0x08`	`CANSM_E_INVALID_COMM_REQUEST`	API network mode request called with invalid communication mode request e.g. SILENT requested in state NoCom.
`0x09`	`CANSM_E_PARAM_INVALID_BAUDRATE`	API change baud rate called with invalid baud rate i.e. the requested baud rate is not equal to the remembered, valid baud rate of the last CanSM_CheckBaudrate call.
`0x0A`	`CANSM_E_MODE_REQUEST_TIMEOUT`	API set transceiver/controller mode request for a network failed more often as allowed by configuration.
`0x0B`	`CANSM_E_INITIALIZED`	API service used after the initialization function.

Table 3-5 Errors reported to DET

3.17.2 Production Code Error Reporting

By default, production code related errors are reported to the DEM using the service `Dem_ReportErrorStatus()` as specified in [3], if production error reporting is enabled (i.e. pre-compile parameter `CANSM_PROD_ERROR_DETECT == STD_ON`).

If another module is used for production code error reporting, the function prototype for reporting the error can be configured by the integrator, but must have the same signature as the service `Dem_ReportErrorStatus()`. The redirection of the function name has to be done via “User Config File”.

The errors reported to DEM are described in the following table:

Error Code	Description
`CANSM_E_BUSOFF_NETWORK_<X>`	The CAN State Manager reports to the DEM a network specific bus-off event each time the bus-off could be recovered or the bus-off could not be recovered within the specified tries.

Table 3-6 Errors reported to DEM

4 Integration

This chapter gives necessary information for the integration of the MICROSAR CanSM into an application environment of an ECU.

4.1 Scope of Delivery

The delivery of the CanSM contains the files which are described in the chapters 4.1.1 and 4.1.2:

4.1.1 Static Files

File Name	Source Code Delivery	Object Code Delivery	Description
CanSM.c	■		This is the source file of the CanSM. It contains the implementation of the main functionality (not available if libraries are delivered).
CanSM.h	■	■	This is the main header file of the CAN State Manager which provides the "defines", function prototypes and types of the CAN State Manager.
CanSM_BswM.h	■	■	This header exports the `CanSM_BswMCurrentStateType`, which is dedicated to the BswM module.
CanSM_Cbk.h	■	■	This is the callback header file that declares the notification functions which inform the CanSM about the transceiver or controller changes.
CanSM_ComM.h	■	■	This is a header file of the CAN State Manager which is the specific interface for the ComM to the services of the CAN State Manager.
CanSM_Dcm.h	■	■	This header exports the `Set`/`Check`/`ChangeBaudrate` interfaces, which are dedicated to the Dcm module.
CanSM_EcuM.h	■	■	This header exports the `Init`/`InitMemory` interfaces, which are used to (pre)initialize the CAN state manager.
CanSM_TxTimeoutException.h	■	■	The header provide the callback function CanSM_TxTimeoutException as optional interface (if PN is active) to the CanNm.

Table 4-1 Static files

4.1.2 Dynamic Files

The dynamic files are generated by the configuration tool DaVinci Configurator.

File Name	Description
CanSM_Cfg.h	Configuration header file which is generated. It contains pre-compile switches, which enable/disable features, type definitions and constant values.
CanSM_Lcfg.c	Configuration source file. It contains configuration parameter which may be changed at link time.

File Name	Description
CanSM_PBcfg.c	Configuration source file. It contains for example timer variable values or channel configuration parameter. It contains configuration parameter which may be changed after link time.

Table 4-2 Generated files

4.2 Critical Sections

Critical sections are handled by the BSW Scheduler. The intention of the following critical sections is to block the interrupt of CanSM functions (with a higher priority).

- The `CANSM_EXCLUSIVE_AREA_1` has to be used if it is possible that the function `CanSM_MainFunction()` may be interrupted by any of the functions
  - `CanSM_RequestComMode()`
  - `CanSM_ControllerBusOff()`
  - `CanSM_TxTimeoutException()`
  - `CanSM_SetEcuPassive()`
  - `CanSM_StopWakeupSources()`
  - `CanSM_StartWakeupSources().`
- The `CANSM_EXCLUSIVE_AREA_2` has to be used if it is possible that the function `CanSM_RequestComMode()` may be interrupted by any of the functions
  - `CanSM_MainFunction()`
  - `CanSM_ControllerModeIndication()`
  - `CanSM_TransceiverModeIndication()`
  - `CanSM_ClearTrcvWufFlagIndication()`
  - `CanSM_CheckTransceiverWakeFlagIndication()`
  - `CanSM_TxTimeoutException()`
  - `CanSM_SetEcuPassive()`
  - `CanSM_StopWakeupSources()`
  - `CanSM_StartWakeupSources().`
- The `CANSM_EXCLUSIVE_AREA_3` has to be used if it is possible that the function `CanSM_ControllerBusOff()` may be interrupted by any of the functions
  - `CanSM_RequestComMode()`
  - `CanSM_ControllerBusOff()`
  - `CanSM_TxTimeoutException().`

The intention of the following critical sections is to avoid a change of the CAN controller or transceiver mode during shutdown of the CAN communication when the CanSM performs the transition to from Silent Communication to No Communication.

> The `CANSM_EXCLUSIVE_AREA_4` has to be used if it is possible that one of functions `CanSM_MainFunction()` or `CanSM_RequestComMode()` may be interrupted by a CAN event.

   1. By CAN Wake Up Interrupt
   2. By CAN Wake Up Polling
   3. By CAN Bus-Off (Can error)

> The `CANSM_EXCLUSIVE_AREA_5` has to be used if it is possible that one of the functions `CanSM_SetEcuPassive()` or `CanSM_StartWakeupSources()` or `CanSM_StopWakeupSources()` may be interrupted by any of the functions

  > `CanSM_RequestComMode()`

  > `CanSM_MainFunction()`.

> Or it is possible that the function `CanSM_ControllerModeIndication()` may be interrupted by the function

  > `CanSM_SetEcuPassive()`.

5 API Description

For an interfaces overview please see Figure 2-2.

5.1 Type Definitions

The types defined by the CanSM are described in this chapter.

Type Name	C-Type	Description	Value Range
`CanSM_BswMCurrentStateType`	`uint8`	CAN specific communication modes / states notified to the BswM module.	`CANSM_BSWM_NO_COMMUNICATION`
			`CANSM_BSWM_SILENT_COMMUNICATION`
			`CANSM_BSWM_FULL_COMMUNICATION`
			`CANSM_BSWM_BUS_OFF`
			`CANSM_BSWM_CHANGE_BAUDRATE`
`CanSM_ChannelConfigPtrType`	`pointer`	Pointer to the structure which contains the configuration data of a CAN channel.	
`CanSM_ChannelConfigType`	`struct`	Structure which contains the configuration data of a CAN channel.	
`CanSM_ConfigType`	`struct`	Structure which contains the global configuration data.	
`CanSM_ChannelVarRecordType`	`struct`	Structure contains the variable values of a specific CAN channel.	
`CanSM_BorStateType`	`uint8`	Can specific bus-off level.	`CANSM_BOR_NONE`
			`CANSM_BOR_LEVEL1`
			`CANSM_BOR_LEVEL2`

Table 5-1 Type definitions

5.2 Services Provided by CanSM

5.2.1 CanSM_InitMemory

Prototype	
void **CanSM_InitMemory**(void)	
Parameter	
-	-
Return code	
-	-

Functional Description
This function initializes the CanSM memory and sets the variable `CanSM_IsInitialized` to `FALSE`
Particularities and Limitations
■ Service ID: see table 'Service IDs' ■ Called once at start-up before the initialization function.
Expected Caller Context
■ Function is called once before CanSM_Init

Table 5-2 CanSM_InitMemory

5.2.2 CanSM_PreInit

Prototype	
`void CanSM_PreInit (const CanSM_ConfigType *const ConfigPtr)`	
Parameter	
`ConfigPtr` [in]	Pointer to configuration structure
Return code	
void	none
Functional Description	
Initializes the configuration data component.	
Particularities and Limitations	
CanSM_InitMemory has been called if CANSM_PREVENT_BUSSLEEP_AT_STARTUP is activated unless CanSM_EnableSetBusSleep[] is initialized by start up code. The API is only needed in case of extended RAM check. Otherwise use CanSM_Init without CanSM_PreInit. Configuration Variant(s): CANSM_EXTENDED_RAM_CHECK	
Call context	
> TASK > This function is Reentrant	

Table 5-3 CanSM_PreInit

5.2.3 CanSM_Init

Prototype	
`void CanSM_Init( const CanSM_ConfigType* const ConfigPtr )`	
Parameter	
`ConfigPtr`	Pointer to the configuration structure that shall be used for the post-build parameters.
Return code	
-	-

Functional Description	
Service for CAN State Manager initialization.	
Particularities and Limitations	
■ Service ID: see table 'Service IDs' ■ Non Reentrant	
Expected Caller Context	
■ Called once after startup	

Table 5-4 CanSM_Init

5.2.4 CanSM_MainFunction

Prototype	
`void` **`CanSM_MainFunction`**`( void )`	
Parameter	
-	-
Return code	
-	-
Functional Description	
The main function of the CanSM executes asynchron transitions of each network, which is configured for the CanSM.	
Particularities and Limitations	
■ Service ID: see table 'Service IDs' ■ CanSM has to be initialized. Function has to be called cyclically. The cycle time is set in the configuration tool. ■ Non Reentrant	
Expected Caller Context	
■ Cyclic on task level	

Table 5-5 CanSM_MainFunction

5.2.5 CanSM_RequestComMode

Prototype	
`Std_ReturnType` **`CanSM_RequestComMode`**`( NetworkHandleType NetworkHandle, ComM_ModeType CanSM_RequestedComMMode )`	
Parameter	
`NetworkHandle`	The communication network number belonging to the request.
`CanSM_RequestedComMMode`	New desired value of the communication mode.
Return code	
`ReturnType`	Returns whether function parameter are valid or not.

Functional Description
The function stores the requested communication mode for the network handle and executes the corresponding network mode state machine.
Particularities and Limitations
■ Service ID: see table 'Service IDs' ■ CanSM has to be initialized. ■ Reentrant for different CAN networks, not reentrant for the same CAN network
Expected Caller Context
■ Function can be called in task and interrupt context.

Table 5-6 CanSM_RequestComMode

5.2.6 CanSM_GetCurrentComMode

Prototype	
`Std_ReturnType` **`CanSM_GetCurrentComMode`** `( NetworkHandleType NetworkHandle, ComM_ModeType* CanSM_ComMModePtr )`	
Parameter	
`NetworkHandle`	Index of the network channel.
`CanSM_ComMModePtr`	Pointer where the communication mode information is copied to.
Return code	
`ReturnType`	Returns whether function parameter are valid or not.
Functional Description	
This service delivers the current communication mode of a CAN network.	
Particularities and Limitations	
■ Service ID: see table 'Service IDs' ■ CanSM has to be initialized.	
Expected Caller Context	
■ Function can be called in task and interrupt context.	

Table 5-7 CanSM_GetCurrentComMode

5.2.7 CanSM_GetVersionInfo

Prototype	
`void` **`CanSM_GetVersionInfo`** `( Std_VersionInfoType * VersionInfo )`	
Parameter	
`VersionInfo`	Pointer, where to store the version data of the CanSM.
Return code	
-	-

Functional Description
This service returns the version information of this module. The version information includes: - Module Id - Vendor Id - Vendor specific version numbers (The versions are BCD-coded).
Particularities and Limitations
■ Service ID: see table 'Service IDs' ■ The function is only available if enabled at compile time (CANSM_VERSION_INFO_API = STD_ON)
Expected Caller Context
■ Function can be called in task and interrupt context.

Table 5-8 CanSM_GetVersionInfo

5.2.8 CanSM_CheckBaudrate

Prototype	
`Std_ReturnType` **`CanSM_CheckBaudrate`**`( NetworkHandleType CanSM_NetworkHandle, uint16 CanSM_Baudrate )`	
Parameter	
`CanSM_NetworkHandle`	The communication network number belonging to the request.
`CanSM_Baudrate`	New desired baud rate.
Return code	
`ReturnType`	E_OK: Baudrate supported by all configured CAN controllers of the network E_NOT_OK: Baudrate not supported / invalid network
Functional Description	
This service check, if a certain baud rate is supported by the configured CAN controller of a CAN network.	
Particularities and Limitations	
■ Service ID: see table 'Service IDs' ■ CanSM has to be initialized. ■ Reentrant for different CAN networks, not reentrant for the same CAN network ■ Please note that this API is deprecated and is kept only for backward compatibility reasons (Substituted by CanSM_SetBaudrate).	
Expected Caller Context	
■ Function can be called in task and interrupt context.	

Table 5-9 CanSM_CheckBaudrate

5.2.9 CanSM_ChangeBaudrate

Prototype
`Std_ReturnType` **`CanSM_ChangeBaudrate`**`( NetworkHandleType CanSM_NetworkHandle, uin16 CanSM_Baudrate )`

Parameter	
`CanSM_NetworkHandle`	The communication network number belonging to the request.
`CanSM_Baudrate`	New desired baud rate.
Return code	
`ReturnType`	Returns whether function parameter are valid or not.
Functional Description	
This service starts a process to change the baud rate for the configured CAN controllers of a certain CAN network	
Particularities and Limitations	
■ Service ID: see table 'Service IDs' ■ CanSM has to be initialized. ■ CanSM_CheckBaudrate has to be called first successfully. ■ Reentrant for different CAN networks, not reentrant for the same CAN network ■ Please note that this API is deprecated and is kept only for backward compatibility reasons (Substituted by CanSM_SetBaudrate).	
Expected Caller Context	
■ Function can be called in task and interrupt context.	

Table 5-10 CanSM_ChangeBaudrate

5.2.10 CanSM_SetBaudrate

Prototype	
`Std_ReturnType` **`CanSM_SetBaudrate`**`( NetworkHandleType CanSM_NetworkHandle, uint16 BaudRateConfigID )`	
Parameter	
`CanSM_NetworkHandle`	The communication network number belonging to the request.
`BaudRateConfigID`	References a baud rate configuration by ID (see CanControllerBaudRateConfigID)
Return code	
`ReturnType`	E_OK: Service request accepted, setting of (new) baud rate started E_NOT_OK: Service request not accepted
Functional Description	
This service starts a process to change the baud rate for the configured CAN controller of a CAN network.	
Particularities and Limitations	
■ Service ID: see table 'Service IDs' ■ CanSM has to be initialized ■ Reentrant for different CAN networks, not reentrant for the same CAN network	
Expected Caller Context	
■ Function can be called in task and interrupt context.	

Table 5-11 CanSM_SetBaudrate

5.2.11 CanSM_StartWakeupSources

Prototype	
`Std_ReturnType CanSM_StartWakeupSources( NetworkHandleType CanSM_NetworkHandle )`	
Parameter	
`NetworkHandle`	Network handle of the wake-up source which should be started
Return code	
`E_OK`	The CanSM has set the CanTrcv and CanDrv in the required states
`E_NOT_OK`	It was not possible to set the CanTrcv or CanDrv to the required state to perform the wake-up validation, e.g. because of dominant level on Rx pin. At this point the CanTrcv or CanDrv are in an “undefined” state. The CanSM itself does not execute any retry. The application has to perform an ECU dependent error handling.
Functional Description	
This function notifies the CanSM module that the EcuM has received a wake-up event which has to be validated.	
Particularities and Limitations	
■ CanSM has to be initialized. ■ Reentrant for different CAN networks ■ Transceiver which work asynchronous must not be used (i.e. Partial network Trcv, SPI Trcv, Trcv within SBC)	
Expected Caller Context	
■ Function can be called in task context.	

Table 5-12 CanSM_StartWakeupSources

5.2.12 CanSM_StopWakeupSources

Prototype	
`Std_ReturnType CanSM_StopWakeupSources( NetworkHandleType CanSM_NetworkHandle, EcuM_WakeupSourceType WakeupSource )`	
Parameter	
`NetworkHandle`	The communication network number belonging to the request.
`WakeupSource`	The wake-up source handle of the CAN channel which should be stopped
Return code	
`E_OK`	The CanSM has set the CanTrcv and CanDrv in the required states or started a new wakeup.
`E_NOT_OK`	It was not possible to set the CanTrcv or CanDrv to the required state, e.g. because of dominant level on Rx pin. At this point the CanTrcv or CanDrv are in an “undefined” state. The CanSM itself does not execute any retry. The application has to perform an ECU dependent error handling.
Functional Description	
This function notifies the CanSM module that the wake-up has not been determined as valid within the specified validation time	

Particularities and Limitations
■ CanSM has to be initialized. ■ Reentrant for different CAN networks ■ Transceiver which work asynchronous must not be used (i.e. Partial network Trcv, SPI Trcv, Trcv within SBC)
Expected Caller Context
■ Function can be called in task and interrupt context.

Table 5-13 CanSM_StopWakeupSources

5.2.13 CanSM_CheckBorLevel

Prototype	
`Std_ReturnType` **`CanSM_CheckBorLevel`**`( const NetworkHandleType NetworkHandle, const CanSM_BorStateType* CanSM_BorStatePtr)`	
Parameter	
`NetworkHandle`	Index of the network channel.
`CanSM_BorStatePtr`	Pointer to target variable, which shall be used for the output of the bus-off recovery level.
Return code	
`ReturnType`	E_OK: API request accepted E_NOT_OK: API request rejected
Functional Description	
This service delivers the current bus-off level of a CAN network.	
Particularities and Limitations	
■ Service ID: see table 'Service IDs' ■ CanSM has to be initialized.	
Expected Caller Context	
■ Function can be called in task and interrupt context.	

Table 5-14 CanSM_CheckBorLevel

5.2.14 CanSM_SetEcuPassive

Prototype	
`void CanSM_SetEcuPassive( boolean CanSM_EcuPassiveMode )`	
Parameter	
`CanSM_EcuPassiveMode`	Boolean parameter which switches the ECU mode between active and passive mode
Return code	
-	-

Functional Description
The function stores the requested ECU mode until it's modified by the next call of this function. In passive mode the CanSM sets the Tx PDU mode to OFFLINE_ACTIVE instead to ONLINE.
Particularities and Limitations
■ CanSM has to be initialized. ■ Non Reentrant
Expected Caller Context
■ Function can be called in task and interrupt context.

Table 5-15 CanSM_SetEcuPassive

5.2.15 CanSM_PreventBusSleepAtStartUp

Prototype	
`Std_ReturnType CanSM_PreventBusSleepAtStartUp( NetworkHandleType CanSM_NetworkHandle )`	
Parameter	
`CanSM_NetworkHandle`	communication network handle
Return code	
`Std_ReturnType`	Returns whether the network handle is valid and if the function has been called before or after the initialization.
Functional Description	
The function can be used to prevent the bus sleep state of the CanIf, CanDrv and CanTrcv at start up for the given CAN network handle. The CanIf, CanDrv and CanTrcv leaves in the corresponding module initialization state.	
Particularities and Limitations	
■ Called at start-up before the CanSM initialization function ■ The function must not be used with PostBuildSelecabel configuarions	
Expected Caller Context	
■ Function has to be called before CanSM_Init	

Table 5-16 CanSM_PreventBusSleepAtStartUp

5.2.16 CanSM_RamCheckStatus

Prototype	
`Std_ReturnType` **`CanSM_RamCheckStatus`** `(NetworkHandleType CanSM_NetworkHandle)`	
Parameter	
CanSM_NetworkHandle [in]	Network handle
Return code	
Std_ReturnType	CANSM_APPL_RAMCHECK_ENABLE Everything is E_OK CANSM_APPL_RAMCHECK_DISABLE Communication shall be disabled CANSM_APPL_RAMCHECK_ENABLE_REPEAT Communication shall be

	enabled and the RAM check repeated CANSM_APPL_RAMCHECK_DISABLE_REPEAT Communication shall be disabled and the RAM check repeated E_NOT_OK wrong Parameter
Functional Description	
Reports the RAM check status to the ComM.	
Particularities and Limitations	
Reports the last RAM check status Configuration Variant(s): CANSM_EXTENDED_RAM_CHECK	
Call context	
> ANY > This function is Synchronous > This function is Reentrant	

Table 5-17 CanSM_RamCheckStatus

5.2.17 CanSM_RamCheckEnableMailbox

Prototype	
`void` **`CanSM_RamCheckEnableMailbox`** `(NetworkHandleType Network, Can_HwHandleType MailBox)`	
Parameter	
`Network` [in]	network handle
`MailBox` [in]	HW mail box identifier
Return code	
void	none
Functional Description	
Forwards enable mail box.	
Particularities and Limitations	
If a mail box shall be enabled the information from the application is passed through to the CanDrv via CanIf. Configuration Variant(s): CANSM_EXTENDED_RAM_CHECK	
Call context	
> ANY > This function is Synchronous > This function is Reentrant	

Table 5-18 CanSM_RamCheckEnableMailbox

5.3 Services Used by CanSM

In the following table services provided by other components, which are used by the CanSM are listed. For details about prototype and functionality refer to the documentation of the providing component.

Component	API
Application	Appl_CanSM_RamCheckCorruptController
Application	Appl_CanSM_RamCheckCorruptMailbox
Application	Appl_CanSM_RamCheckFinished
Application	Appl_CanSM_RamCheckStart
BswM	BswM_CanSM_CurrentState
CanIf	CanIf_SetControllerMode
CanIf	CanIf_SetTrcvMode
CanIf	CanIf_ChangeBaudrate
CanIf	CanIf_SetPduMode
CanIf	CanIf_CheckTrcvWakeFlag
CanIf	CanIf_ClearTrcvWufFlag
CanIf	CanIf_GetTxConfirmationState
CanIf	CanIf_RamCheckEnableController
CanIf	CanIf_RamCheckEnableMailbox
CanIf	CanIf_RamCheckExecute
CanNm	CanNm_ConfirmPnAvailability
DEM	Dem_ReportErrorStatus
DET	Det_ReportError
ComM	ComM_BusSM_ModeIndication
SchM	SchM_Enter_CanSM_CANSM_EXCLUSIVE_AREA_i for i=1,2,3,4,5
SchM	SchM_Exit_CanSM_CANSM_EXCLUSIVE_AREA_i for i=1,2,3,4,5

Table 5-19 Services used by the CanSM

5.4 Callback Functions

This chapter describes the callback functions that are implemented by the CanSM and can be invoked by other modules. The prototypes of the callback functions are provided in the header file `CanSM_Cbk.h` by the CanSM.

5.4.1 CanSM_ControllerBusOff

Prototype	
`void` **`CanSM_ControllerBusOff`** `( uint8 CanSM_ControllerId )`	
Parameter	
`CanSM_ControllerId`	Index of the CAN controller, which detected a bus-off event
Return code	
-	-

Functional Description
The CanSM is notified about a bus-off event on a certain CAN controller with this callback function. The CanSM uses this information to execute the bus-off recovery for the corresponding controller.
Particularities and Limitations
■ CanSM has to be initialized.
Expected Caller Context
■ Function can be called in task and interrupt context.

Table 5-20 CanSM_ControllerBusOff

5.4.2 CanSM_ControllerModeIndication

Prototype	
`void` **`CanSM_ControllerModeIndication`**`(uint8 CanSM_ControllerId, CanIf_ControllerModeType CanSM_ControllerMode )`	
Parameter	
`CanSM_ControllerId`	Index of the CAN controller, which detected a bus-off event
`CanSM_ControllerMode`	Notified CAN controller mode
Return code	
-	-
Functional Description	
This callback shall notify the CanSM module about a CAN controller mode change.	
Particularities and Limitations	
■ Service ID: see table 'Service IDs' ■ CanSM has to be initialized.	
Expected Caller Context	
■ Function can be called in task and interrupt context.	

Table 5-21 CanSM_ControllerModeIndication

5.4.3 CanSM_TransceiverModeIndication

Prototype	
`void` **`CanSM_TransceiverModeIndication`**`( uint8 CanSM_TransceiverId, CanIf_TrcvModeType CanSM_TransceiverMode )`	
Parameter	
`CanSM_TransceiverId`	Index of the CAN controller, which detected a bus-off event
`CanSM_TransceiverMode`	Notified CAN transceiver mode
Return code	
-	-
Functional Description	
This callback shall notify the CanSM module about a CAN transceiver mode change.	

Particularities and Limitations
■ Service ID: see table 'Service IDs' ■ CanSM has to be initialized.
Expected Caller Context
■ Function can be called in task and interrupt context.

Table 5-22 CanSM_TransceiverModeIndication

5.4.4 CanSM_ClearTrcvWufFlagIndication

Prototype	
`void` **`CanSM_ClearTrcvWufFlagIndication`** `( uint8 CanSM_TransceiverId )`	
Parameter	
`CanSM_TransceiverId`	The transceiver ID number belonging to the request.
Return code	
–	-
Functional Description	
This call-back function indicates the CanIf_ClearTrcvWufFlag API process end for the notified CAN Transceiver.	
Particularities and Limitations	
■ Service ID: see table 'Service IDs' ■ CanSM has to be initialized. ■ Reentrant for different CAN transceivers	
Expected Caller Context	
■ Function can be called in task and interrupt context.	

Table 5-23 CanSM_ClearTrcvWufFlagIndication

5.4.5 CanSM_CheckTransceiverWakeFlagIndication

Prototype	
`void` **`CanSM_CheckTransceiverWakeFlagIndication`** `( uint8 CanSM_TransceiverId )`	
Parameter	
`CanSM_TransceiverId`	The transceiver ID number belonging to the request.
Return code	
–	-
Functional Description	
This call-back function indicates the CheckTransceiverWakeFlag API process end for the notified CAN Transceiver.	

Particularities and Limitations
■ Service ID: see table 'Service IDs' ■ CanSM has to be initialized. ■ Reentrant for different CAN transceivers
Expected Caller Context
■ Function can be called in task and interrupt context.

Table 5-24 CanSM_CheckTransceiverWakeFlagIndication

5.4.6 CanSM_ConfirmPnAvailability

Prototype	
`void` **`CanSM_ConfirmPnAvailability`** `( uint8 CanSM_TransceiverId )`	
Parameter	
`CanSM_TransceiverId`	The transceiver ID number belonging to the request.
Return code	
–	-
Functional Description	
This call-back function indicates that the transceiver is running in PN communication mode. In this case the CanNm will be informed by calling CanNm_ConfirmPnAvailability.	
Particularities and Limitations	
■ Service ID: see table 'Service IDs' ■ CanSM has to be initialized. ■ Reentrant for different CAN transceivers	
Expected Caller Context	
■ Function can be called in task and interrupt context.	

Table 5-25 CanSM_ConfirmPnAvailability

5.4.7 CanSM_TxTimeoutException

Prototype	
`void` **`CanSM_TxTimeoutException`** `( NetworkHandleType CanSM_NetworkHandle )`	
Parameter	
`CanSM_NetworkHandle`	The communication network number belonging to the request.
Return code	
–	-
Functional Description	
This function notifies the CanSM module that the Com has detected a Tx timeout exception, which shall be recovered by the CanSM module by a re-initialization of the CAN controller.	

Particularities and Limitations
■ Service ID: see table 'Service IDs' ■ CanSM has to be initialized. ■ Reentrant for different CAN networks
Expected Caller Context
■ Function can be called in task and interrupt context.

Table 5-26 CanSM_TxTimeoutException

5.4.8 CanSM_RamCheckCorruptMailbox

Prototype	
`void` **`CanSM_RamCheckCorruptMailbox`** `(uint8 CanSM_ControllerId, Can_HwHandleType MailBox)`	
Parameter	
CanSM_ControllerId [in]	CAN controller index
`MailBox` [in]	Mail box identifier
Return code	
void	none
Functional Description	
Handles the indication of a RAM check error.	
Particularities and Limitations	
Gets information about RAM check errors. Forwards the information to the application and evaluates HW register failures Configuration Variant(s): -	
Call context	
> ANY > This function is Reentrant	

Table 5-27 CanSM_RamCheckCorruptMailbox

5.4.9 CanSM_RamCheckCorruptController

Prototype	
`void` **`CanSM_RamCheckCorruptController`** `(uint8 CanSM_ControllerId)`	
Parameter	
CanSM_ControllerId [in]	CAN controller index
Return code	
void	none
Functional Description	
Handles the indication of a RAM check error.	
Particularities and Limitations	
Gets information about RAM check errors. Forwards the information to the application and evaluates HW	

register failures Configuration Variant(s): -
Call context
> ANY > This function is Reentrant

Table 5-28 CanSM_RamCheckCorruptController

5.5 Callout Functions

5.5.1 Appl_CanSM_RamCheckStart

Prototype	
`void **Appl_CanSM_RamCheckStart** (NetworkHandleType CanSM_NetworkHandle)`	
Parameter	
CanSM_NetworkHandle [in]	network handle
Return code	
void	none
Functional Description	
Indicates the start of the RAM check.	
Particularities and Limitations	
Indicates the start of the RAM check. Configuration Variant(s): CANSM_EXTENDED_RAM_CHECK	
Call context	
> ANY > This function is Synchronous > This function is Reentrant	

Table 5-29 Appl_CanSM_RamCheckStart

5.5.2 Appl_CanSM_RamCheckCorruptController

Prototype	
`void **Appl_CanSM_RamCheckCorruptController** (NetworkHandleType CanSM_NetworkHandle)`	
Parameter	
CanSM_NetworkHandle [in]	network handle
Return code	
void	none
Functional Description	
Forwards register RAM failures.	
Particularities and Limitations	
If register RAM failures occurs the information from the CanDrv is passed through the Application.	

Configuration Variant(s): CANSM_EXTENDED_RAM_CHECK
Call context
> ANY > This function is Synchronous > This function is Reentrant

Table 5-30 Appl_CanSM_RamCheckCorruptController

5.5.3 Appl_CanSM_RamCheckCorruptMailbox

Prototype	
`void` **`Appl_CanSM_RamCheckCorruptMailbox`** `(NetworkHandleType CanSM_NetworkHandle, Can_HwHandleType MailBox)`	
Parameter	
CanSM_NetworkHandle [in]	Network handle
Can_HwHandleType [in]	HW mail box identifier
Return code	
void	none
Functional Description	
Forwards message box RAM failures.	
Particularities and Limitations	
If a message box RAM failure occurs the information from the CanDrv is passed through the Application. Configuration Variant(s): CANSM_EXTENDED_RAM_CHECK	
Call context	
> ANY > This function is Synchronous > This function is Reentrant	

Table 5-31 Appl_CanSM_RamCheckCorruptMailbox

5.5.4 Appl_CanSM_RamCheckFinished

Prototype	
`Std_ReturnType` **`Appl_CanSM_RamCheckFinished`** `(NetworkHandleType CanSM_NetworkHandle)`	
Parameter	
CanSM_NetworkHandle [in]	Network handle
Return code	
Std_ReturnType	CANSM_APPL_RAMCHECK_ENABLE Everything is E_OK CANSM_APPL_RAMCHECK_DISABLE Communication shall be disabled CANSM_APPL_RAMCHECK_ENABLE_REPEAT Communication shall be enabled and the RAM check repeated CANSM_APPL_RAMCHECK_DISABLE_REPEAT Communication shall be disabled and the RAM check repeated

Functional Description
Indicates the end of the RAM check.
Particularities and Limitations
The CanDrv has finished the extended RAM check. All potential errors have been reported. The Application has to specify further actions via return value. Configuration Variant(s): CANSM_EXTENDED_RAM_CHECK
Call context
> ANY > This function is Synchronous > This function is Reentrant

Table 5-32 Appl_CanSM_RamCheckFinished

6 AUTOSAR Standard Compliance

6.1 Deviations

6.1.1 Communication mode requests are acceped if possible

The module accepts the communication mode requests even if there is a pending mode indication. E.g. the CanSM is in state S_CC_STARTED_WAIT (3.3.3) and gets a NO_COMMUNICATION request the deinitialization (3.3.7) becomes started.

Det_ReportError with the ErrorId parameter CANSM_E_WAIT_MODE_INDICATION is not used.

6.2 Additions/ Extensions

6.2.1 API CanSM_InitMemory()

This service function was added to be called at "Power On" or after reset to set the global CanSM state. Afterwards the CanSM can be initialized correctly.

6.2.2 No Mode Notification During CanSM_Init

The ComM_BusSM_ModeIndication and BswM_CanSM_CurrentState are not called during the transition from `CANSM_INIT` to `CANSM_NO_COMMUNNICATION` because the ComM and BswM become initialized after the CanSM.

6.2.3 Configuration Options

It's possible to (de)activate the DEM at pre-compile time, like DET.

6.2.4 Additional Bus-Off Recovery in State Silent

If bus-off occurs outside the state FULL_COMMUNICATION, the CanSM handles bus-off and sets the CAN controller mode to STARTED once.

6.2.5 API CanSM_CheckBorLevel()

This service function delivers the current bus-off level of a CAN network.

6.2.6 Partial Network Wake Up Filter

For the partial network use case it has to be ensured that that the first message on the bus is a wake up message. Therefore the CanSM triggers the PDU Mode `CANIF_SET_ONLINE_WAKF` instead `CANIF_SET_ONLINE`. The CanSM feature is automatically active if the feature is active in the CanIf.

6.2.7 ECU Passive Mode

The passive mode deactivates the Tx part during full communication. The ECU listens "passively" on all CAN busses.

6.2.8 PreventBusSleepAtStartUp

The additional API CanSM_PreventBusSleepAtStartUp() allows to skip the initial transition for the selected channel(s).

6.2.9 Post-Build Selectable (Identity Manager)

The code generator and the static code supports post build selectable configuration.

6.2.10 APIs to Assist EcuM Wakeup Validation

The APIs can be used to ensure that the CAN HW is started/online during running wakeup Validation (chapters 3.15, 3.17.1, 4.2, 5.2.11, 5.2.12).

6.2.11 Swift or Large Tx Timeout Exception handling

The CanSM provides two different versions of Tx Timeout Exception handling. The desired one can be configured. The new swift version sets the controller to stopped and back to started instead executing the whole shut down sequence to NoCom.

6.2.12 Extended RAM Check

The CanSM triggers the DrvCan to execute CanSelfDiag (Extended RAM Check).

6.2.13 Expanded Tx Timeout Exception Handling

The CanSM provides the option to configure a callout function which is called at the end of the timeout exception handling. If a valid function name is configured the CanSM activates the "expanded" time out exception handling. The "expanded" time out exception handling is equal to the CanSMSwiftTxTimeoutRecovery followed by the configured end indication. In addition the CanSM executes the handling also if the Tx timeout exception is indicated in the states "SILENTCOM" or "BUS_OFF_CHECK".

6.3 Limitations

6.3.1 Controllers

The CanSM supports only one controller per channel.

6.3.2 Configuration Class

Only VARIANT-PRE-COMPILE is supported.

7 Glossary and Abbreviations

7.1 Glossary

Term	Description
DaVinci Configurator	Generation tool for MICROSAR components

Table 7-1 Glossary

7.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
BswM	Basic Software Mode Manager
CAN	Controller Area Network
CanDrv	CAN Driver
CanIf	CAN Interface
CanNm	CAN Network Management
CanSM	CAN State Manager
CanTrcv	CAN Transceiver
Cbk	Call-back / call-out notification (functions)
Cfg	Configuration
ComM	Communication Manager
DEM, Dem	Diagnostic Event Manager
DET, Det	Development Error Tracer
DTC	Diagnostic Trouble Code
ECU	Electronic Control Unit
EcuM	ECU State Manager
HIS	Hersteller Initiative Software
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
PDU	Protocol Data Unit
PN	Partial Networking
RAM	Random Access Memory
SBC	System Basis Chip
SchM	BSW Scheduler
SPI	Serial Peripheral Interface
SWC	Software Component

Table 7-2 Abbreviations

8 Contact

Visit our website for more information on

> News
> Products
> Demo software
> Support
> Training data
> Addresses

www.vector.com