MICROSAR CAN Network Management

Technical Reference

Nm_Asr4NmCan

Version 9.00.00

Authors	Markus Schuster, Patrick Kleemann
Status	Released

Document Information

History

Author	Date	Version	Remarks
Markus Schuster	2012-08-08	1.00.00	ESCAN00058396 Creation
Markus Drescher	2013-05-10	1.01.00	ESCAN00063144 Updated Architecture Overview ESCAN00064972 Support of Variant Post-Build-Loadable ESCAN00065301 Improved send behavior descriptions in chapter 3.6.3 and Immediate Nm Transmission feature descriptions in chapters 3.15 and 3.16 ESCAN00065574 Extended chapter 3.13 ESCAN00067271 Merged chapter 'AUTOSAR Standard Compliance' with chapter 3, removed 'Compiler Abstraction and Memory Mapping' chapter, various improvements ESCAN00067277 Adapted chapter 5.3.1.1 ESCAN00067278 Replaced Nm_PrepareBusSleep by Nm_PrepareBusSleepMode
Markus Drescher	2013-10-01	2.00.00	ESCAN00067700 Added Runtime Measurement Support to 'Features Beyond the AUTOSAR Standard' ESCAN00070810 Updated Architecture Overview
Markus Drescher	2014-02-24	2.01.00	ESCAN00072375 Adapted condition for usage of CanSM_TxTimeoutException in chapter 5.4 ESCAN00073874 Updated Architecture Overview
Markus Schuster	2014-05-12	3.00.00	ESCAN00075248 Add description of dependency of Bus Load Reduction and Partial Networking feature on the same channel in chapter 3.6.4
Markus Schuster	2014-10-09	4.00.00	ESCAN00076763 Added description in chapter 1, 3.1.2 and 5.3.1.1. Removed chapter 5.2 'Type Definitions' ESCAN00078817 Added description in chapter 3.1.1
Markus Schuster	2015-06-03	5.00.00	ESCAN00082408 Updated Table 3-1 and Table 3-3, Table 3-7
Markus Schuster	2016-03-02	6.00.00	ESCAN00086897 Adapted chapter 3.4 ESCAN00087953 Adapted chapter 3.8 ESCAN00087415 Adapted chapter 3.1.1

Markus Schuster	2016-05-09	6.01.00	ESCAN00089821 Adapted chapter 3.1 ESCAN00090105 Adapted chapter 3.18.3 ESCAN00090927 Added chapter 3.5.2.1
Markus Schuster	2016-11-17	6.02.00	FEATC-58 Adapted chapter 3.11
Patrick Kleemann	2018-01-15	7.00.00	STORYC-3565 Added chapter 3.6.3.1 ESCAN00098007 Modified chapter 3.15 ESCAN00097257 Added CanNm_CancelTransmit service
Patrick Kleemann	2018-02-06	7.01.00	STORYC-3912 Added information about channel based parameters STORYC-4227 Modified chapter 3.6.3.1
Patrick Kleemann	2018-02-26	7.02.00	STORYC-4228 Modified chapter 3.13
Patrick Kleemann	2018-04-04	7.02.01	ESCAN00098922 Modified chapter 3.5
Patrick Kleemann	2018-07-25	7.02.02	ESCAN00100169 Modified chapter 5.3.2.8.1
Patrick Kleemann	2018-09-28	8.00.00	STORYC-6795 Modified chapter 3.6.2 and 3.18 STORYC-7160 Modified chapter 3.1.1 STORYC-7163 Modified chapter 3.7 ESCAN00101460 Modified chapter 4.5 ESCAN00101395 Modified chapter 3.15
Patrick Kleemann	2019-03-22	9.00.00	STORYC-8039 Removed restriction regarding PN byte range in chapter 3.18, Modified chapter 3.18.2 Added chapter 3.18.6 ESCAN00102594 Modified chapter 4.5

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_SRS_NetworkManagement.pdf	3.0.0
[2]	AUTOSAR	AUTOSAR_SWS_CANInterface.pdf	5.0.0
[3]	AUTOSAR	AUTOSAR_SWS_CANNetworkManagement.pdf	3.3.0
[4]	AUTOSAR	AUTOSAR_SWS_DiagnosticEventManager.pdf	4.2.0
[5]	AUTOSAR	AUTOSAR_SWS_DevelopmentErrorTracer.pdf	3.2.0
[6]	AUTOSAR	AUTOSAR_TR_BSWModuleList.pdf	1.6.0
[7]	AUTOSAR	AUTOSAR_SWS_RTE.pdf	3.2.0
[8]	Vector	Technical Reference MICROSAR PDU Router	See delivery

Table 1-1 Reference Documents

Scope of the Document

This technical reference describes the specific use of the CAN Network Management basic software.

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

Illustrations

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
1.00.00	Adaption to AUTOSAR Release 4
1.02.00	Support Variant Post-Build-Loadable
2.00.00	Added Runtime Measurement Support
3.00.00	Support Variant Post-Build-Selectable

Table 1-1 Component history

2. Introduction

This document describes the functionality, API and configuration of the AUTOSAR BSW module CANNM as specified in [1] and [3]. Also, the integration of the Network Management into the AUTOSAR stack is covered by this document.

The FlexRay Network Management, LIN Network Management and the UDP Network Management are not covered by this document.

Please note that in this document the term Application is not used strictly for the user software but also for any higher software layer, like e.g. the Communication Manager (ComM). Therefore, Application refers to any of the software components using the CAN NM.

For further information please also refer to the AUTOSAR SWS specifications, referenced at the beginning of this document in Table: 'Reference Documents'.

Supported AUTOSAR Release*:	4	
Supported Configuration Variants:	pre-compile, post-build-loadable, post-build-selectable	
Vendor ID:	CANNM_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	CANNM_MODULE_ID	31 decimal (According to ref.[6])

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

2.1 Naming Conventions

The names of the service functions provided by the NM Interface and CAN NM always start with a prefix that denominates the module where the service is located. E.g. a service that starts with '`CanNm_`' is implemented within the CAN NM.

Naming conventions	
`Nm_`	Services of NM Interface.
`CanNm_`	Services of CAN NM.
`Det_`	Services of Development Error Tracer.
`Dem_`	Services of Diagnostic Event Manager.

Table 2-1 Naming Conventions

Nodes which are configured to be passive will be also referred to as passive nodes. Accordingly, nodes that are not passive will be termed as active nodes.

2.2 Architecture Overview

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

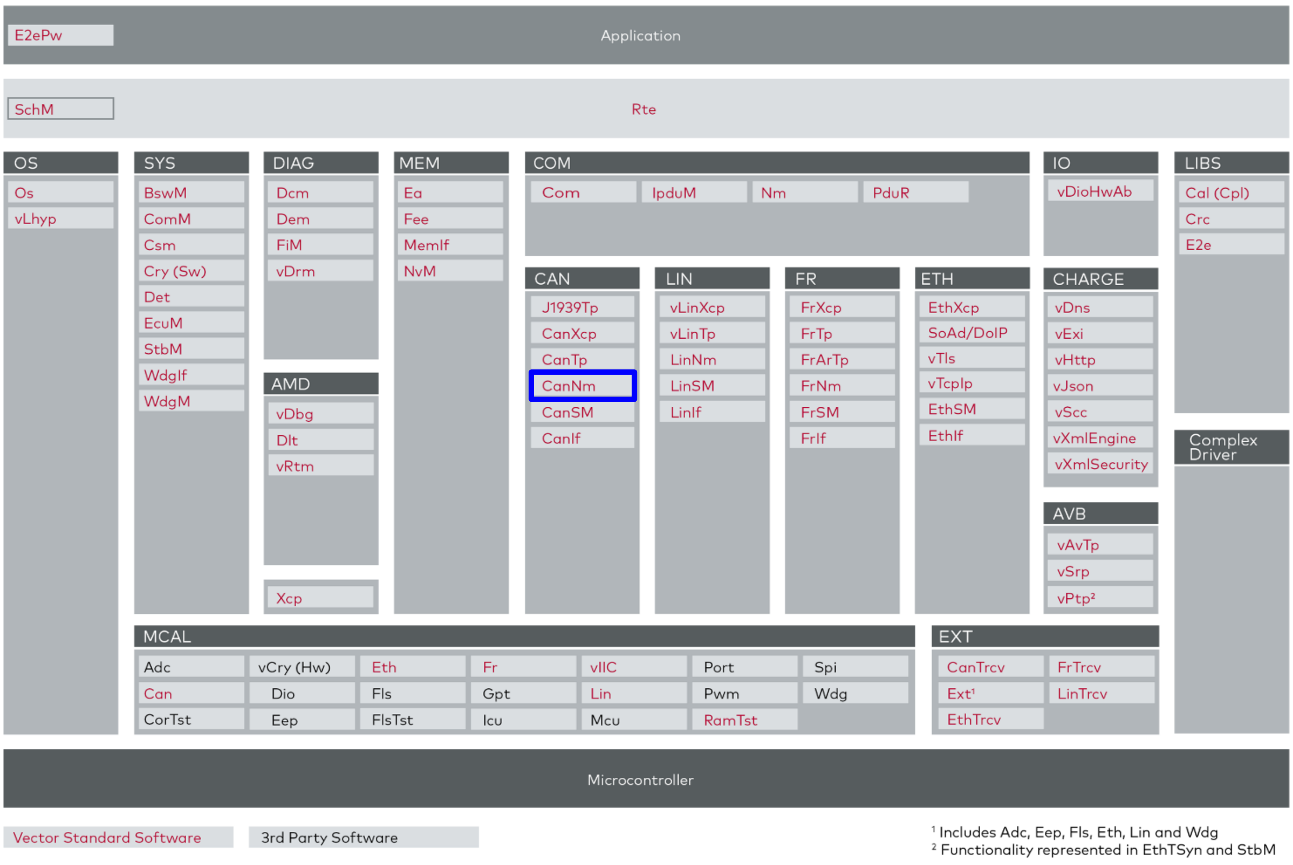

Figure 2-1 AUTOSAR 4.x Architecture Overview

2.2.1 Architecture of AUTOSAR Network Management

In the current AUTOSAR Release the standard AUTOSAR Network Management may consist of up to five modules:

> NM Interface[1]

> CAN NM

> FlexRay NM[1]

> LIN NM[1]

> UDP NM[1]

The NM Interface schedules function calls from the application to the respective module for each channel, e.g. for a CAN channel the corresponding CAN NM function will be called. CAN NM exclusively interacts with the NM Interface.

The communication bus specific functionality is incorporated in the corresponding bus-specific NM. The CAN-specific part implements the network management algorithm and is

[1] Not covered by this document.

responsible for the transmission of NM messages on the communication bus by interacting with the AUTOSAR CAN Interface.

The next figure shows the interfaces to adjacent modules of the CAN NM. These interfaces are described in chapter 5 ‘API Description’.

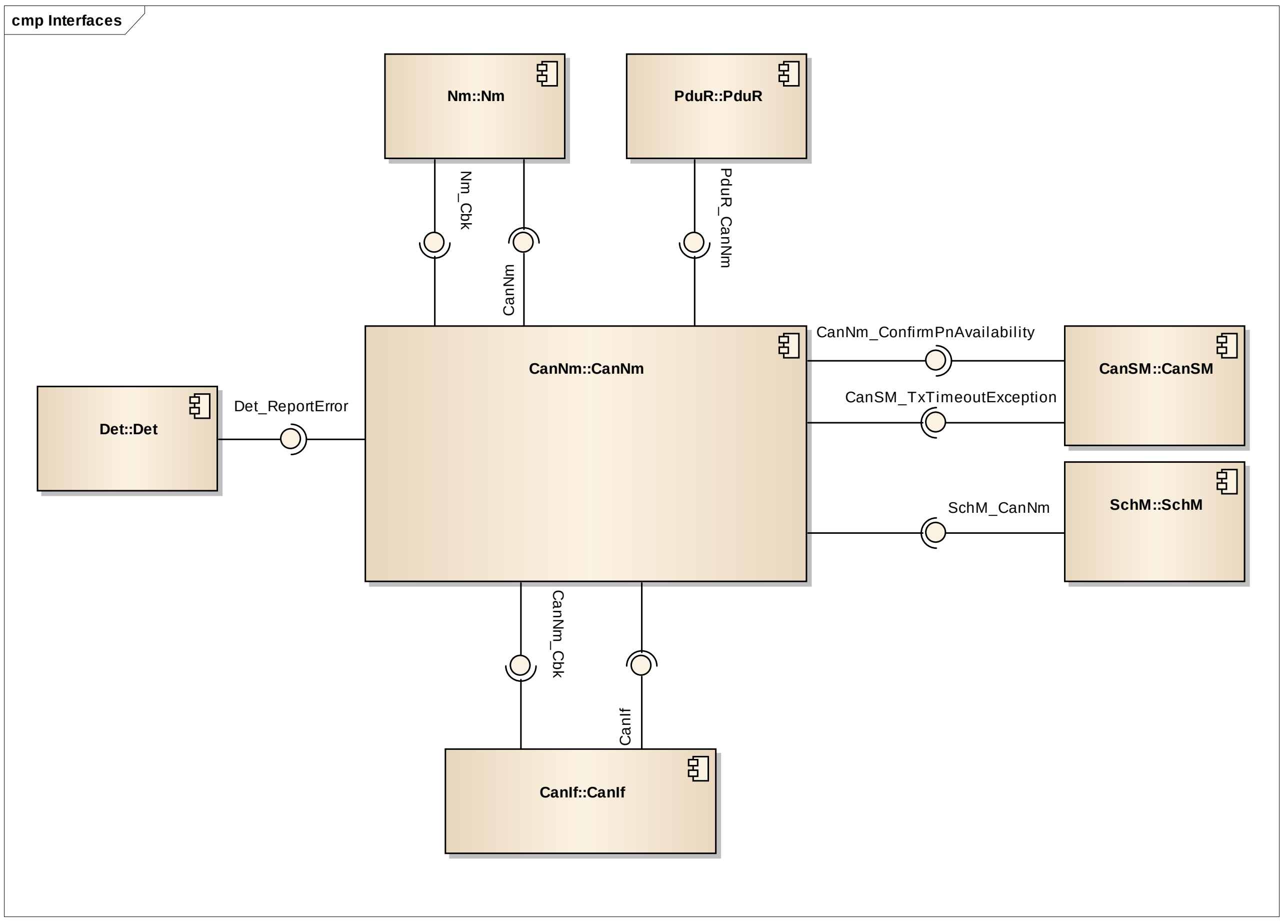

Figure 2-2 Interfaces to adjacent modules of the CANNM

Applications do not access the services of the BSW modules directly. They use the service ports provided by the BSW modules via the RTE. Since the CAN NM has no service ports, the CAN NM cannot be accessed via RTE by the application.

3. Functional Description

3.1 Features

The Network Management is a network comprehensive protocol that provides services for the organization of the network. It is a decentralized and direct network management. That means that every ECU transmits a special network management message, which is reserved for the network management only.

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

The AUTOSAR standard functionality is specified in [3], 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

Vector Informatik provides further CanNm 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 [3] are supported:

Supported AUTOSAR Standard Conform Features
Controlled transition of all ECU's to bus-sleep mode and vice versa.
User Data Handling
Node Detection
Remote Sleep Indication
Coordinator Synchronization Support
Bus Synchronization
Bus Load Reduction
Immediate Tx Confirmation
Passive Mode Support
Immediate Restart
Pdu Rx Indication
State Change Indication
Repeat Message Indication
Com User Data Support
Active Wake-up Bit
Immediate Nm Transmissions
Car Wake-up
Partial Networking
Retry First Message Request

Supported AUTOSAR Standard Conform Features
Post-Build Loadable
MICROSAR Identity Manager using Post-Build Selectable

Table 3-1 Supported AUTOSAR standard conform features

3.1.1 Deviations Against AUTOSAR

The following features specified in [3] are not supported:

Category	Description	ASR Version
Functional	Communication Scheduling ch. 7.6: The CanNmMsgCycleOffset is not applied when all NM messages have been transmitted with CanNmImmediateNmCycleTime.[CANNM335]	4.0.3
Functional	Initialization ch. 7.4. A call of CanNm_PassiveStartUp() in PrepareBusSleep leads to a transition to Repeat Message state. [CANNM147]	4.0.3
Functional/Config	Error Notification ch. 7.16. CANNM_E_NETWORK_TIMEOUT is only reported if configuration switch CANNM_DISABLE_TX_ERROR_REPORT is enabled[CANNM193][CANNM194]	>4.0.3
Functional	Debugging Concept ch. 7.18.3 Debugging is supported in MICROSAR, but not as described in this chapter.[CANNM287]-[CANNM290]	4.0.3
Functional	The Transmission Timeout handling is not executed if a second NM message transmit request is issued before the first one is acknowledged. (Refer to ESCAN00100877)	4.0.3

Table 3-2 Not supported AUTOSAR standard conform features

3.1.1.1 RAM Initialization

If RAM is not implicitly initialized at start-up, the function `CanNm_InitMemory` must be called.

3.1.1.2 Additional Configuration Dependencies

Following additional dependencies between configuration parameters are added to avoid bad configurations:

> Com Control Enabled must be disabled for passive nodes.

> Node Detection Enabled must be disabled for passive nodes.

3.1.1.3 Variant Post-Build

Instead of the Configuration Variant Post-Build, the Variant Post-Build-Loadable is supported.

3.1.2 Additions/ Extensions

The following extensions of the CAN NM software specifications ([3]) are available within the Network Management embedded software components. If required, the extensions must be enabled during configuration.

Features Provided Beyond the AUTOSAR Standard
Single Channel Optimization
Memory Initialization
Disable Transmission Error Reporting
Calling CanNm_PassiveStartUp in Prepare Bus Sleep
Additional Development Error Codes
Variable DLC Support
Changeability of Additional Parameters During the Post-Build Phase
Runtime Measurement Support

Table 3-3 Features provided beyond the AUTOSAR standard

Note
Some additional non-AUTOSAR features are only available if they are explicitly ordered by the customer.

3.1.2.1 Single Channel Optimization

For single channel systems, it is possible to optimize the source code for saving precious resources (ROM, RAM and CPU load). This optimization is only possible when source code is available.

Please note that single channel optimization can only be enabled in pre-compile configurations.

3.1.2.2 Memory Initialization

AUTOSAR expects the startup code to automatically initialize RAM. Not every startup code of embedded targets reinitializes all variables correctly it is possible that the state of a variable may not be initialized, as expected. To avoid this problem the Vector AUTOSAR NM provides additional functions to initialize the relevant variables of the CAN NM.

Refer also to chapter 5.3.1.3 ‘CanNm_InitMemory’.

3.1.2.3 Disable Transmission Error Reporting

The error reporting for the following transmission errors can be disabled:

> CANNM_E_DEV_NETWORK_TIMEOUT

3.1.2.4 Calling CanNm_PassiveStartUp in Prepare Bus Sleep

Calling CanNm_PassiveStartUp in Prepare Bus Sleep Mode has the same effects as if it was called in Bus Sleep Mode. This has been done to support the Synchronous Wake-up Feature in ComM.

3.1.2.5 Additional Development Error Codes

There are additional Development Error Codes provided as Vector extension. Refer to chapter 3.12.1.1 for details.

3.1.2.6 Variable DLC Support

CanNm supports multiple DLCs for CAN NM messages. Refer to chapter 3.6.5 for details.

3.1.2.7 Changeability of Additional Parameters During the Post-Build Phase

In the Variant Post-Build-Loadable, the configuration parameters 'Node Id', 'Rx Pdu Ref', 'Tx User Data Pdu Ref', 'Pn Filter Mask Byte Index' and 'Pn Filter Mask Byte Value' are also changeable during the post-build phase as addition to the post-build-changeable parameters according to [3].

3.1.3 Limitations

3.1.3.1 Ranges of Timers

The range for the following timer is limited concerning the specified range:

> Remote Sleep Indication Timer: 0.001..65.535 s (if remote sleep indication is enabled)

All timers should be multiples of the main functions cycle time.

3.1.3.2 Effects of CanNm_DisableCommunication

If `CanNm_DisableCommunication` was called, CanNm_NetworkRelease has the same effects as if `CanNm_DisableCommunication` was not called (contradicts to CANNM294 of [3]). This was done to provide a more robust implementation.

3.1.3.3 CANNM_E_NET_START_IND Development Error

The `CANNM_E_NET_START_IND` development error code (CANNM336 of [3]) is not reported to the Det if an NM message has been received in Bus Sleep Mode.

The following provides a detailed description of the functional scope.

3.2 Network Management Mechanism

As described above the AUTOSAR NM is a decentralized direct network management. This means that every network node has the same functionality and performs state and operation mode changes self-sufficient depending on the internal state and whether network management messages are still received.

The network management mechanism is quite simple:

> Every network node transmits its NM messages only if it needs to communicate with other network nodes. Normally bus-communication is required if clamp15 (ignition is turned on) is set or during follow-up.

> If there is no more network node in the whole network that need to communicate with other network nodes, any node transmits no more NM messages.

> Each network node performs a transition to bus-sleep mode a certain time after the last NM message has been transmitted by any node. Therefore, all nodes will go to bus-sleep mode together.

> If any network node requires bus-communication at any time it can wake up the whole network by transmitting NM messages.

Caution
The transmission of application messages, e.g. transmitted by the Com, does not stop immediately after the last NM message has been transmitted.

FAQ
The application is in charge of the decision whether the bus communication is required or not.

The following figure shows the state diagram of the CAN NM. The events are calls of CAN NM functions by the application or data link layer or the timeout of internal timers.

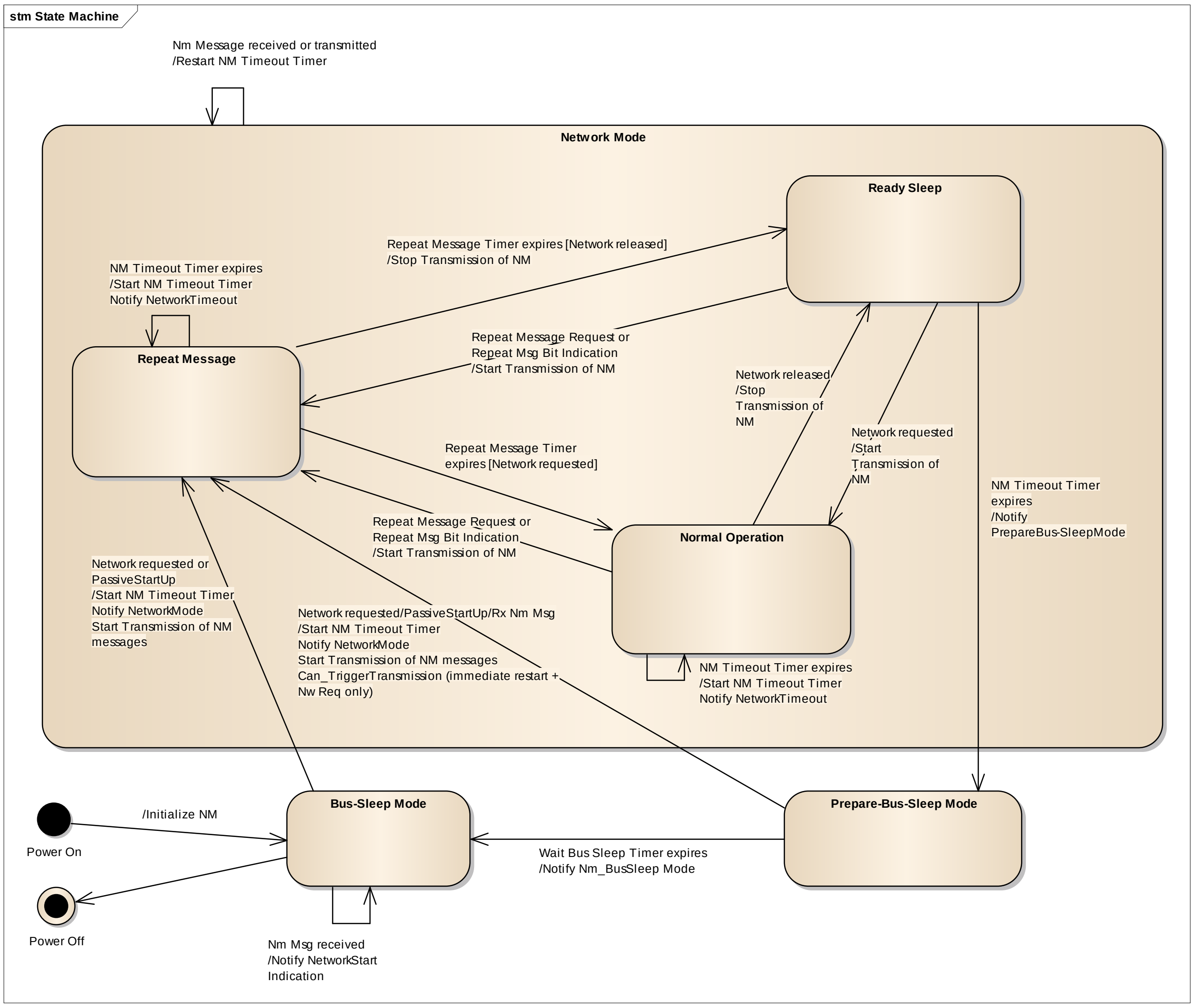
stm State Machine
Nm Message received or transmitted /Restart NM Timeout Timer
Network Mode
Ready Sleep
Repeat Message Timer expires [Network released] /Stop Transmission of NM
NM Timeout Timer expires /Start NM Timeout Timer Notify NetworkTimeout
Repeat Message
Repeat Message Request or Repeat Msg Bit Indication /Start Transmission of NM
Network released /Stop Transmission of NM
Network requested /Start Transmission of NM
Repeat Message Timer expires [Network requested]
NM Timeout Timer expires /Notify PrepareBus-SleepMode
Repeat Message Request or Repeat Msg Bit Indication /Start Transmission of NM
Normal Operation
Network requested or PassiveStartUp /Start NM Timeout Timer Notify NetworkMode Start Transmission of NM messages
Network requested/PassiveStartUp/Rx Nm Msg /Start NM Timeout Timer Notify NetworkMode Start Transmission of NM messages Can_TriggerTransmission (immediate restart + Nw Req only)
NM Timeout Timer expires /Start NM Timeout Timer Notify NetworkTimeout
/Initialize NM
Power On
Bus-Sleep Mode
Prepare-Bus-Sleep Mode
Wait Bus Sleep Timer expires /Notify Nm_BusSleep Mode
Power Off
Nm Msg received /Notify NetworkStart Indication

Figure 3-1 State Diagram of CAN NM from SWS CAN NM [3]

3.3 Initialization

Before the CAN NM can be used it must be initialized by the application. The initialization must be carried out before any other functionality of the CAN NM is executed. It shall take place after initialization of the CAN Interface and prior to initialization of the NM Interface.

Also refer to chapter 5.3.1.1 ‘CanNm_Init: Initialization of CAN NM’.

Caution
The CAN NM assumes that some variables are initialized with zero at start-up. If the embedded target does not initialize RAM within the start-up code the function ‘CanNm_InitMemory’ must be called during start-up and before the initialization is performed. Refer also to chapter 3.1.2.2 ‘Memory Initialization’.

Note
In an AUTOSAR environment where the ECU Manager Fixed is used, the initialization is performed within the ECU Manager. If the ECU Manager Flex is used, the initialization is usually carried out by the BswM.

3.4 Passive Mode

Nodes in passive mode cannot transmit NM messages and therefore they do not actively participate in the network. Due to that, passive nodes cannot request the network.

This mode can be used for nodes that do not need to keep the bus awake to save resources.

By setting 'Repeat Message Time' to a value equal to 0, the Repeat Message state is skipped. The state does not make sense for passive nodes, since the node is only able to receive NM messages, not to send any. Usually, there is another node that sends NM messages in Repeat Message so there is no need for 'Repeat Message Time' being greater than 0 for passive nodes.

Nevertheless, if 'Repeat Message Time' is configured to a value greater than 0 and ‘Timeout Time’ is greater than ‘Repeat Message Time’ and if 'Passive Mode Enabled' is turned ON, the transition from Repeat Message to Prepare Bus Sleep only depends on the reception of NM messages. If there is no recently received NM message, the transition to Prepare Bus Sleep occurs after Timeout Time has elapsed.

In case ‘Repeat Message Time’ is greater than ‘Timeout Time’ and no NM message is received a DET error will occur at least once when the Timeout Timer elapses within Repeat Message State. The transition to Prepare Bus Sleep occurs ‘Timeout Time’ after the last DET error call.

3.5 Operation Modes and States

The AUTOSAR NM consists of three operation modes:

> Network Mode

- Prepare Bus-Sleep Mode
- Bus-Sleep Mode

The NM Interface is notified about changes of the operation mode by calling the following functions:

Entering Bus-Sleep Mode:
`Nm_BusSleepMode()` (5.4)

Entering Network Mode:
`Nm_NetworkMode()` (5.4)

Leaving Network Mode:
`Nm_PrepareBusSleepMode()` (5.4)

Information about the current state and the current mode is provided by the service call `CanNm_GetState` (chapter 5.3.2.2).

The CAN NM notifies changes of the current state to the NM Interface by calling the optional function

`Nm_StateChangeNotification()` (5.4)

Note
The function Nm_StateChangeNotification() is only called if the previous and the current state differ.

3.5.1 Network Mode

The Network Mode comprises three states:

- Repeat Message
- Normal Operation
- Ready Sleep

This is the mode in that the ECU is ‘online’ and participates in the network. The participation in the network is active or passive depending on the state:

- Active participation: a node keeps the network awake (Repeat message State and Normal Operation State).
- Passive participation: a node is ready for sleep (Ready Sleep State) and any other node keeps the network alive.

The application is notified about entering the Network Mode by a call of the function:

`Nm_NetworkMode``()` (5.4)

The NM Interface notifies leaving the Network Mode to the application by a call of the function:

`Nm_PrepareBusSleepMode``()` (5.4)

Info
The Com is active during Network Mode. It is started upon entry and stopped upon exit of Network Mode. I.e. application messages are transmitted and received within Network Mode!

3.5.1.1 Repeat Message State

The Repeat Message State is entered:

> If a NM message has been received in Prepare Bus-Sleep Mode.

> If the network has been requested by a call of `CanNm_NetworkRequest()` in Bus-Sleep or Prepare Bus-Sleep Mode.

> If the network is woken up from Bus-Sleep Mode or from Prepare Bus-Sleep Mode by a call of `CanNm_PassiveStartUp()`.

> If any network node (including itself) has requested node detection in Ready Sleep or Normal Operation State.

In Repeat Message State, the NM messages are transmitted cyclically regardless whether bus load reduction is enabled or disabled.

The Repeat Message State is left after a certain customizable time.

Depending on the bus-communication need of the application Normal Operation State or Ready Sleep State is entered upon exit of Repeat Message State.

3.5.1.2 Normal Operation State

The network management stays in Normal Operation State until the bus-communication is released. The local bus-communication request of the application is distributed in the network by the transmission of NM messages.

3.5.1.3 Ready Sleep State

The network management stays in Ready Sleep State as long as the application does not request bus-communication and the application of any other node still requests bus-communication (by transmitting NM messages).

A certain customizable time after the last network node has released bus-communication a transition to Prepare Bus-Sleep Mode is performed (i.e. Network Mode is left).

3.5.2 Prepare Bus-Sleep Mode

The transmission of application messages is stopped when entering Prepare Bus-Sleep Mode. The bus activity is calmed down (pending message are still transmitted) in this mode and finally there is no more activity on the bus.

After the ‘wait bus sleep time’ the drop out of Prepare Bus-Sleep Mode to Bus-Sleep Mode the NM Interface is notified by the service call:

`Nm_BusSleepMode()` (5.4)

Caution
When entering Bus-Sleep Mode the physical bus interface must be put into sleep mode.

On CAN channels the transceiver and the CAN-Controller must be put in sleep mode. This information is forwarded by this callback to the ComM.

Note
If both NmOsek and CanNm are used on the same channel, CanNm is aware of the prolonged shutdown of the NmOsek in case of a Limphome condition if the Wait Bus Sleep Extensions feature is turned ON. For details see the following chapter 3.5.2.1.

The Prepare Bus-Sleep Mode is left to Network Mode upon successful reception of a NM message or if the network has been requested by a call of `CanNm_NetworkRequest()` or if the network has been woken up by a call of `CanNm_PassiveStartUp()`.

3.5.2.1 Wait Bus Sleep Extensions

If both NmOsek and CanNm are coordinated on the same channel, the internal state of NmOsek influences the shutdown behavior of the CanNm.

> NmOsek transitions from state NmNormal to NmWaitBusSleep

In this case the CanNm applies its normal shutdown time by using its own "wait bus sleep time".

> NmOsek transitions from state NmLimpHome to NmWaitBusSleep

In this case the CanNm applies a longer shutdown time by using "TErrorWaitBusSleep" configured in NmOsek.

Note
This feature is automatically enabled when NmOsek and CanNm are configured on the same channel and "Wait Bus Sleep Extensions" feature is enabled in NmOsek.

3.5.3 Bus-Sleep Mode

All network nodes perform a transition to bus-sleep mode at almost the same time, if no NM message is lost and there hasn't been a wake-up by any node.

The bus-sleep mode is left (wake-up) by a call of

`CanNm_PassiveStartUp()` (5.3.2.3)

or if the network has been requested by a call of

`CanNm_NetworkRequest()` (5.3.2.4.1)

In both cases Repeat Message State will be entered (see Chapter 3.5.1.1 'Repeat Message State').

If a NM message is received in Bus-Sleep Mode the service

`Nm_NetworkStartIndication()` (5.4)

is called by CAN NM.

3.5.4 Wake-up Registration

The network management needs to know whether the application requires bus communication. Per default the network management does not actively participate in the network. The active participation in the network is requested by the service

`CanNm_NetworkRequest()` (5.3.2.4.1)

Calling this function in Bus-Sleep Mode starts the network and leads to a transition to Repeat Message State (see Chapter 3.5.3 'Bus-Sleep Mode').

If bus communication is not required anymore it can be released with the service

`CanNm_NetworkRelease()` (5.3.2.4.2)

Caution
When the communication control service is used the bus-communication shall not be released as long as the NM message transmission ability is disabled.

Note that a bus-communication request is handled within the next task. Nevertheless, it is ensured that a communication request always leads to start-up even if the communication is released before the next task is executed. Within Network Mode a fast toggling (i.e. without task execution in between) of the communication status does not lead to any action.

3.5.5 User Data Handling

The user data for the NM message transmitted next on the bus can be set by the service:

`CanNm_SetUserData()` (5.3.2.5.1)

The service

`CanNm_GetUserData()` (5.3.2.5.2)

allows reading the user data of the last received message on the bus.

As the NM PDU layout is completely configurable, the user data placement depends on the given configuration.
The PDU layout and the content of the user data itself are OEM specific and therefore provided by the OEM.

Note that for setting of NM user data a second possibility can be configured. Refer to chapter 3.13 ‘Com User Data Support’ for more information. If the feature ‘Com User Data Support’ is used the API `CanNm_SetUserData()` is not available.

3.6 Network Management Message Transmission and Reception

3.6.1 AUTOSAR CAN Interface

The network management requests the transmission of NM messages by calling the service `CanIf_Transmit` [2]. The application must take care of the user data. For details refer to chapter 3.5.5 ‘User Data Handling’.

The successful transmission of every network management message is confirmed by the CAN Interface with the service

`CanNm_TxConfirmation()` (5.5.1.1)

The CAN Interface indicates the reception of NM message by calling the service

`CanNm_RxIndication()` (5.5.1.2)

3.6.2 PDU Message Layout

The default PDU Message Layout is described in the following table:

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte n	User data n - 2							
Byte n-1	User data n - 3							
…	…							
Byte 7	User data 5							
Byte 6	User data 4							
Byte 5	User data 3							
Byte 4	User data 2							
Byte 3	User data 1							
Byte 2	User data 0							
Byte 1	Control Bit Vector							
Byte 0	Source Node Identifier							

Table 3-4 PDU NM Message Layout

The number of User Data Bytes as well as the positions of the Control Bit Vector and Source Node Identifier can be configured arbitrarily but depend on the availability of the corresponding features (User Data Support / Node Detection Enabled / Node Identifier).

Note
The maximum Pdu length is configurable to 64 bytes.

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 1 (default)	Reserved	Cluster Request Information Bit[2]	Reserved	Active Wake-up Bit	NM Coordinator Sleep Ready	Reserved	Reserved	Repeat Message Request

Table 3-5 Control Bit Vector

The Repeat Message Request Bit is only used if the 'Node Detection' feature is active. Refer to chapter 3.7 for more information.

The NM Coordinator Sleep Ready Bit is only used if the 'Coordinator Synchronization Support' is active. See chapter 3.11 for more details.

The Active Wake-up Bit is used for the 'Active Wake-up Handling' (chapter 3.14).

The Cluster Request Information Bit is used for 'Partial Networking' (chapter 3.18).

All bits inside the Control Bit Vector are optionally used and depend on the setting of these features. If the feature is not used, the bit value is 0.

3.6.3 Message Transmissions

A standard sequence for NM message transmissions when Repeat Message is entered is depicted in Figure 3-2.

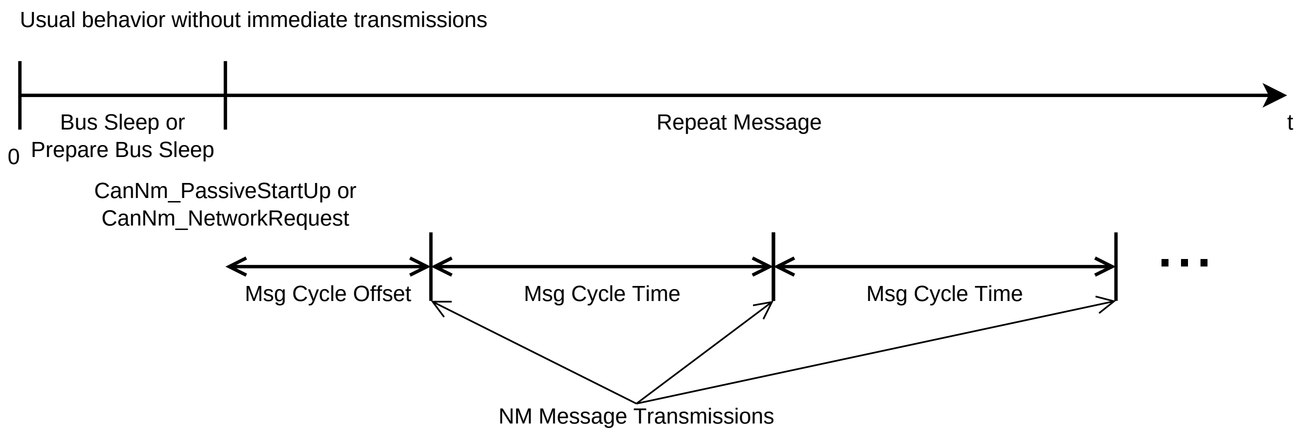

Figure 3-2 Usual Behavior of NM Transmissions when Repeat Message is entered

The first NM message is transmitted when 'Msg Cycle Offset' ms has been elapsed after Repeat Message has been entered. The next NM message will be transmitted after Msg Cycle Time has been elapsed. The configuration settings that influence this behavior are:

[2] This bit is also called 'Partial Network Information Bit'

- ‘Msg Cycle Offset’: time before the transmission of the first NM message
- ‘Msg Cycle Time’: time between each message transmission
- ‘Immediate Restart Enabled’: if enabled, an additional NM message will be sent immediately upon an active request (CanNm_NetworkRequest was called) from Prepare Bus Sleep to Repeat Message (see also chapter 3.16) in case ‘Msg Cycle Offset’ is greater than zero. Immediate Restart can be enabled per channel.
- ‘Immediate Nm Transmissions’: if this setting is greater than zero, an immediate NM message will be sent when Repeat Message is entered due to an active request. The interval between NM messages will be different for the next (‘Immediate Nm Transmissions’ - 1) NM messages to be sent. Refer to chapter 3.15 for more details.
- ‘Repeat Message Time’: this setting determines for how long CanNm shall keep the Repeat Message state. If the node has been requested passively, the next state will be Ready Sleep.

Note that the NM message is sent as long as the NM state is ‘Repeat Message’ or ‘Normal Operation’. For details about these states, see also chapter 3.5.1.

Note
The lower layer (e.g. CAN Interface) may reject the send request if Network Mode has just been entered.

CanNm usually does not retry to issue the send request of the NM message. There are features, which may enable retries in certain conditions:

If ‘Immediate Nm Transmissions’ are greater than zero, the rejected send request is not considered as ‘Immediate Transmission’ (see chapter 3.15).

3.6.3.1 Retry First Message Request

This feature can be enabled per channel. If the first transmission after transition from Bus Sleep or Prepare Bus Sleep to Repeat Message State is not accepted by the lower layers (e.g. CanIf), the message request shall be repeated in the next main function cycle until the first transmission request is accepted.

If the feature Immediate Restart is enabled (refer to ch. 3.16) and the first (Immediate Restart) message could not be transmitted, the Retry First Message Request mechanism is executed until a message is accepted. The next Nm message will be transmitted after the configured offset. Afterwards the regular message cycle is used (refer to Figure 3-3).

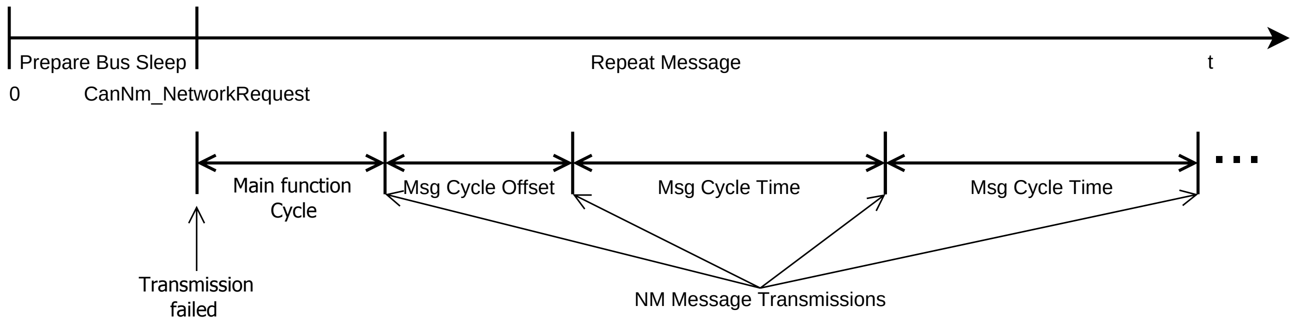

Figure 3-3 Retry First Message Request with Immediate Restart enabled

3.6.4 Bus Load Reduction

The bus load reduction is started automatically if enabled and when Normal Operation state is entered. When Normal Operation state is left, bus load reduction algorithm is stopped. For more information refer to chapter 7.7 of [3].

Note
Bus Load Reduction cannot be used on the channel if Partial Networking is used on the same channel and vice versa. However, setups like Bus Load Reduction is active on the one channel and Partial Networking is active on the other channel is allowed.

3.6.5 Support for RX PDUs with Different Lengths

The CAN NM supports messages with different lengths (DLCs). This support can be enabled by disabling the ‘CanIf Range Config DLC Check’ setting in the module configuration.

Note
The setting is enabled, if a macro definition of CANNM_CANIF_RANGE_CONFIG_DLC_CHECK can be found in CanNm_Cfg.h.

In case the ‘CanIf Range Config DLC Check’ setting is disabled, it is assumed that the CanIf module accepts NM messages with different DLCs. The minimum DLC for NM messages may be configured in the CanIf module, messages with a DLC less than the configured value for the DLC check will be discarded. Messages with the same DLC or greater DLC will be received and forwarded to CanNm.

However, the maximum number of bytes that is evaluated from the received message is equal to the ‘Pdu Length’ setting of the corresponding channel. For messages with a length *n* smaller than ‘Pdu Length’, the bytes *n* … (‘Pdu Length’ - 1) are considered as being zero.

Examples:

The length of a received message is 8, ‘Pdu Length’ is configured to 6. In this case, the last two user data bytes are not further processed by CanNm (e.g. CanNm_GetUserData does not return data for these two bytes).

The length of a received message is 4, ‘Pdu Length’ is configured to 6. In this case, bytes 4 and 5 are considered as being zero (e.g. CanNm_GetUserData returns 0 for these bytes).

The minimum required number of bytes for a received NM message that should be processed by CanNm may be configured by using the CanIf DLC Check feature [2].

If the ‘CanIf Range Config DLC Check’ setting is enabled, either the CanIf DLC feature must be enabled to accept only NM messages with a DLC greater than or equal to the ‘Pdu Length’ setting or there must not be any ECU that sends NM messages with a DLC less than the ‘Pdu Length’ setting. Otherwise, the behavior of the CanNm will be arbitrary.

3.7 Node Detection

To detect which nodes are currently present within the network, this mechanism can be used. If a network node requests node detection, the requesting node performs a transition to Repeat Message State and sets the Repeat Message Bit within the NM PDU. Upon reception of the Repeat Message Bit all network nodes perform a transition to Repeat Message State. This allows the requesting node to collect all source node identifiers from active nodes.

The local source node identifier can be retrieved by the service

`CanNm_GetLocalNodeIdentifier()` (5.3.2.6.3)

The source node identifier from the last received message can be retrieved by the service

`CanNm_GetNodeIdentifier()` (5.3.2.6.2)

Note
The Node Id and Node Detection can be enabled per channel. Furthermore, Post Build Selectable is supported.

The transition to Repeat Message after a received Repeat Message bit is only done if the current state equals Normal Operation or Ready Sleep.

3.8 NM PDU Receive Indication

The NM Interface is notified about the reception of an NM message by the optional function

`Nm_PduRxIndication()` (5.4)

In case more than one BusNm is configured on the same channel, a BusNm specific reception notification is called.

`Nm_CanNm_PduRxIndication()` (5.4)

The CAN NM notifies the callback directly to the NM Interface in context of the function `CanNm_RxIndication` (see chapter 5.5.1.2).

3.9 Communication Control

To support ISO 14229 Communication Control Service $28 the network management has a message transmission control status, which allows disabling the transmission of NM messages while bus-communication is requested. Therefore, the function

`CanNm_DisableCommunication()` (5.3.2.11.1) can be called. The transmission of NM messages will be stopped within the next CAN NM main function call.

The NM PDU transmission ability is enabled again by the service

`CanNm_EnableCommunication()` (5.3.2.11.2)

Caution
An ECU shall not shut down if the NM PDU transmission ability is disabled.

Note
Communication Control can be enabled per channel. Furthermore, Post Build Selectable is supported.

3.10 Gateway Functionality

3.10.1 Remote Sleep Indication and Cancellation

To synchronize networks, it might be necessary to get an indication whether no more network nodes require bus-communication. This is the so-called ‘Remote Sleep Indication’. The start of the remote sleep indication is indicated by

`Nm_RemoteSleepIndication()` (5.4)

If any NM message is received during Normal Operation State or Ready Sleep State after the remote sleep indication the service ‘Remote Sleep Cancellation’ is called:

`Nm_RemoteSleepCancellation()` (5.4)

It is also possible to retrieve the current remote sleep state by calling the service:

`CanNm_CheckRemoteSleepIndication()` (5.3.2.8.1)

Remote sleep indication can only be checked in Ready Sleep state and Normal Operation state.

3.10.2 Bus Synchronization

To synchronize networks for a synchronized shutdown it might be necessary to transmit an asynchronous NM message to reset the network timers. This can be done by calling the service:

`CanNm_RequestBusSynchronization()` (5.3.2.7.1)

However, this service shall only be called within Network Mode.

3.11 Coordinator Synchronization Support

For supporting the NM Interface Coordinator Synchronization with more than one coordinator connected to the same channel it is necessary to provide one additional bit in the CBV. Therefore the service call

`CanNm_SetSleepReadyBit()` (5.3.2.12.1)

allows to set and clear the Nm Coordinator Sleep Ready Bit (see chapter 3.6.2 for further information). Each call of this API triggers the immediate transmission of an NM message can be sent according to the current state in the CanNm State Machine (see Figure 3-1) to propagate the change of the Sleep Ready Bit as soon as possible.

Caution
The 'Coordinator Synchronization Support' requires the Control Bit Vector.

Therefore, this feature must be enabled if the Coordination Synchronization Support is used.

3.12 Error Handling

3.12.1 Development Error Detection

By default, development errors are reported to the DET using the service `Det_ReportError()` as specified in [5], if development error reporting is enabled (i.e. pre-compile parameter `CANNM_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 reported CANNM ID is 31.

The reported service IDs identify the services which are described in 5.3 and 5.5. The following table presents the service IDs and the related services:

Service ID	Service
0x00	CanNm_Init
0x01	CanNm_PassiveStartUp
0x02	CanNm_NetworkRequest
0x03	CanNm_NetworkRelease
0x04	CanNm_SetUserData
0x05	CanNm_GetUserData
0x06	CanNm_GetNodeIdentifier
0x07	CanNm_GetLocalNodeIdentifier
0x08	CanNm_RepeatMessageRequest
0x0A	CanNm_GetPduData

Service ID	Service
0x0B	CanNm_GetState
0x0C	CanNm_DisableCommunication
0x0D	CanNm_EnableCommunication
0x13	CanNm_MainFunction
0x14	CanNm_Transmit
0x16	CanNm_ConfirmPnAvailability
0x17	CanNm_SetSleepReadyBit
0x40	CanNm_TxConfirmation
0x42	CanNm_RxIndication
0xC0	CanNm_RequestBusSynchronization
0xD0	CanNm_CheckRemoteSleepIndication
0xF1	CanNm_GetVersionInfo
0xF2	CanNm_CancelTransmit

Table 3-6 Service IDs

3.12.1.1 Det_ReportError

Development errors are reported by the service

`Det_ReportError()` (5.4)

Please refer to the documentation of the development error tracer [5] for further information and a detailed description of the API. The module Id, API Ids and error Ids can be found within the software components' header file.

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

Error	Code	Description
0x01	CANNM_E_NO_INIT	API service used without module initialization.
0x02	CANNM_E_INVALID_CHANNEL	API service used with wrong channel handle.
0x03	CANNM_E_INVALID_PDUID	API service used with wrong PDU ID
0x04	CANNM_E_NET_START_IND	Reception of NM Message in Bus Sleep Mode
0x05	CANNM_E_INIT_FAILED	CAN NM initialization has failed.
0x11	CANNM_E_NETWORK_TIMEOUT	NM-Timeout Timer has abnormally expired outside of the Ready Sleep State.
0x12	CANNM_E_PARAM_POINTER[3]	Null pointer has been passed as an argument.
0x20	CANNM_E_RXINDICATION_DLC_ERROR[4]	DLC of received NM message does not match with configured PDU Length.
0x21	CANNM_E_PDUR_TRIGGERTX_ERROR[4]	Call of function PduR_TriggerTransmit failed.
0x22	CANNM_E_ALREADY_INITIALIZED	CAN NM initialization done more than once.

[3] Error does not apply to the function `CanNm_Init`.
[4] Vector extension

Error	Code	Description
0x33	CANNM_E_INVALID_GENDATA	Invalid write access due to wrong configuration data.

Table 3-7 Errors reported to DET

3.12.1.2 Parameter Checking

AUTOSAR requires that API functions check the validity of their parameters. The checks in Table 3-8 are internal parameter checks of the API functions. These checks are for development error reporting. The error reporting of `CANNM_E_NETWORK_TIMEOUT` can be en-/disabled separately by the configuration switch ‘Disable Tx Err Report’. The Parameter `CANNM_DEV_ERROR_DETECT` dis-/ enables the call of Det_ReportError() for all checks globally.

The following table shows which parameter checks are performed on which services:

Service \ Check	CANNM_E_NO_INIT	CANNM_E_INVALID_CHANNEL	CANNM_E_NULL_POINTER	CANNM_E_INVALID_PDUID	CANNM_E_NET_START_IND	CANNM_E_INIT_FAILED	CANNM_E_NETWORK_TIMEOUT	CANNM_E_RXINDICATION_DLC_ERROR	CANNM_E_PDUR_TRIGGERTX_ERROR
CanNm_Init						■ [5]			
CanNm_PassiveStartUp	■	■ [6]							
CanNm_NetworkRequest	■	■ [6,7]							
CanNm_NetworkRelease	■	■ [6,7]							
CanNm_SetUserData	■ [7]	■ [6,7]	■ [7]						
CanNm_GetUserData	■	■ [6,7]	■						
CanNm_GetPduData	■	■ [6,7]	■						
CanNm_RepeatMessageRequest	■ [5,7]	■ [6,7]							
CanNm_GetNodeIdentifier	■	■ [6,7]	■						
CanNm_GetLocalNodeIdentifier	■	■ [6,7]	■						
CanNm_RequestBusSynchronization	■ [7]	■ [6,7]							
CanNm_CheckRemoteSleepIndication	■ [7]	■ [6,7]	■ [7]						
CanNm_GetState	■	■ [6,7]	■						
CanNm_GetVersionInfo			■ [5]						

[5] Only checked if CANNM_DEV_ERROR_DETECT is STD_ON
[6] Only checked if CANNM_OPTIMIZE_CHANNEL_ENABLED is not defined (‘Api Optimization’ is OFF)

Check / Service	CANNM_E_NO_INIT	CANNM_E_INVALID_CHANNEL	CANNM_E_NULL_POINTER	CANNM_E_INVALID_PDUID	CANNM_E_NET_START_IND	CANNM_E_INIT_FAILED	CANNM_E_NETWORK_TIMEOUT	CANNM_E_RXINDICATION_DLC_ERROR	CANNM_E_PDUR_TRIGGERTX_ERROR
CanNm_Transmit	■[7]		■[7]	■[5,7]					
CanNm_CancelTransmit	■								
CanNm_EnableCommunication	■[7]	■[6,7,7]							
CanNm_DisableCommunication	■[7]	■[6,7]							
CanNm_ConfirmPnAvailability	■	■[6,7]							
CanNm_SetSleepReadyBit	■	■[6,7]							
CanNm_RxIndication	■		■	■[5]				■[5]	
CanNm_MainFunction	■	■[6,7]					■[5]		■[5]
CanNm_TxConfirmation	■[7]			■[5,7]					

Table 3-8 Development Error Reporting: Assignment of checks to services

3.12.2 Production Code Error Reporting

By default, production code related errors are reported to the DEM using the service `Dem_ReportErrorStatus()` as specified in [4], if production error reporting is enabled.

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 errors reported to DEM are described in the following table:

Error Code	Description
N/A	Currently no DEM errors are specified

Table 3-9 Errors reported to DEM

3.13 Com User Data Support

The CAN NM supports the possibility to write the NM user data via Com signals. Therefore, the signals must be provided within an additional I-PDU in the configuration. The CAN NM updates its transmission buffer each time before sending a NM message with the current data. Therefore, it calls the function:

`PduR_CanNmTriggerTransmit()` (5.4)

[7] Only checked if CANNM_PASSIVE_MODE_ENABLED is STD_OFF

When the NM message has been successfully transmitted the confirmation is forwarded to PduR by calling the function:

`PduR_CanNmTxConfirmation()` (5.4)

Depending on the signal and I-PDU configuration a signal change can lead to a request for an immediate NM message transmission by calling the function

`CanNm_Transmit()` (5.3.2.9.1)

The CAN NM then transmits the changed data in the next main function when the transmission of NM messages is allowed. Afterwards the message cycle timer is restarted, i.e. the cyclic message transmission raster changes.

The spontaneous transmission through CanNm_Transmit is allowed in the NM states Repeat Message and Normal Operation if and only if

> 'Pn Enabled' is ON and 'Pn Handle Multiple Network Requests' is OFF AND/OR

> 'Car Wake Up Rx Enabled' is ON AND/OR.

> 'Allow Triggered Msg Transmissions' is ON.

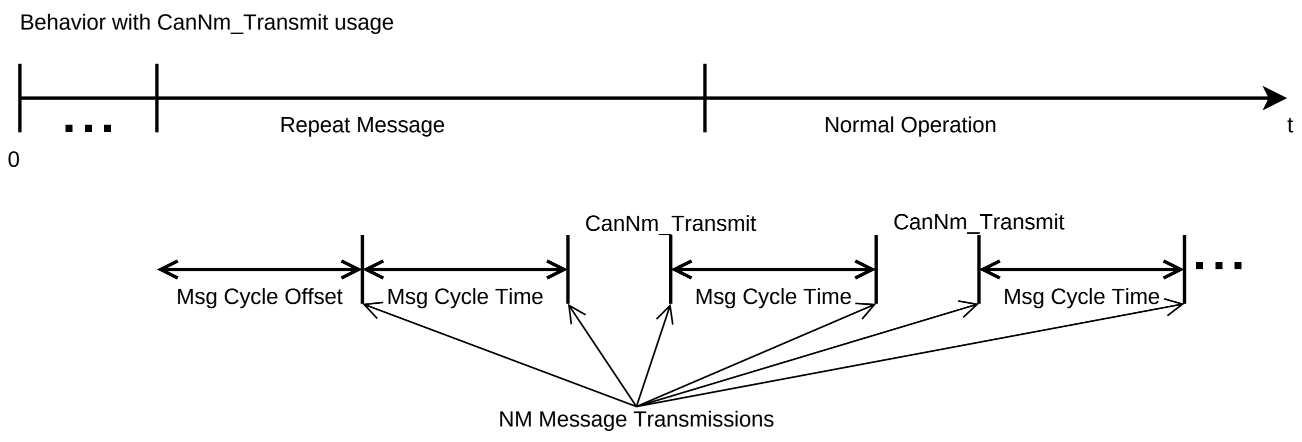

Figure 3-4 Immediate Transmission due to Signal Change inside User Data I-PDU

The following chapters describe more detailed the configuration preconditions of this feature.

Note that some additional configuration for this feature must be done in the PDU Router. Refer to [10] for details.

3.13.1 Configuration Preconditions in an AUTOSAR ECU Configuration

For using the feature 'Com User Data Support' some additional configuration content within the AUTOSAR system description / ECU Extract is necessary. The following table provides an overview of the items that must be added to the system description.

Configuration Element	Description
Signal I-PDU	For each NM message one signal I-PDU must be configured. An appropriate signal mapping to the I-Signals must be defined here. I-PDUs are defined in the ECU-specific part.
I-Signal	Multiple system signals can be defined for each NM message. At least one signal is required. I-Signals are defined in the ECU-specific

Configuration Element	Description
	part and refer to a system signal.
System Signal	For each I-Signal a corresponding system signal is necessary which defines length, data type and init value.
I-PDU Port	For each I-PDU an I-PDU port with the communication direction ‘OUT’ is required.
Signal Port	For each signal, a signal port with the communication direction ‘OUT’ is required.
I-PDU Triggering	For each I-PDU an I-PDU triggering is required that references to the corresponding I-PDU port and the signal I-PDU.
Signal Triggering	For each I-Signal a signal triggering is required that references to the corresponding signal port and I-Signal.

Table 3-10 Configuration Precondition Overview for AUTOSAR ECU Configurations

Additionally, a reference from the NM PDU to the related I-PDU with the signals must be established by adding ‘ISignalToIPduMappings’ to the NM PDU. The following example demonstrates how this should be done:

```
<NM-PDU>
   <SHORT-NAME>NM_PDU</SHORT-NAME>
   <LENGTH>8</LENGTH>
   <I-SIGNAL-TO-I-PDU-MAPPINGS>
       <I-SIGNAL-TO-I-PDU-MAPPING>
           <SHORT-NAME>NM_USR_DT </SHORT-NAME>
           <I-SIGNAL-REF DEST="I-
SIGNAL">/ISignal/NM_USR_DT_SIGNAL</I-SIGNAL-REF>
           <PACKING-BYTE-ORDER>MOST-SIGNIFICANT-BYTE-LAST</PACKING-
BYTE-ORDER>
           <START-POSITION>32</START-POSITION>
       </I-SIGNAL-TO-I-PDU-MAPPING>
   </I-SIGNAL-TO-I-PDU-MAPPINGS>
</NM-PDU>
```

3.14 Active Wake-up Handling

The mode change from Bus-Sleep Mode or Prepare Bus-Sleep Mode to Network Mode triggered by CanNm_NetworkRequest() is specified as “Active Wake-up”. Upon an Active Wake-up the CAN NM sets the active wake-up bit within the Control Bit Vector at bit position 4.

This feature is optional and must be configured.

3.15 Immediate Nm Transmissions

If an Active Wake-up occurs the CAN NM transmits the first NM message immediately (the NM message offset time is ignored) when entering Repeat Message State. For the next NM messages CAN NM uses a faster NM message cycle time. Afterwards it uses the normal NM message cycle time. This behavior is illustrated in Figure 3-5.

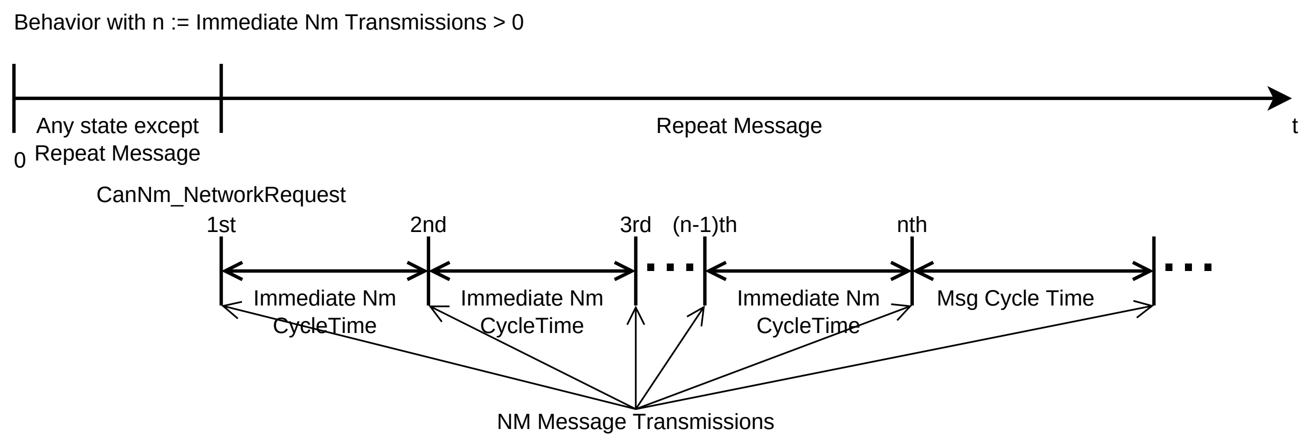

Figure 3-5 Immediate Nm Transmissions

The number of 'Immediate Nm Transmissions' is the number that is configured for this parameter. As it can be seen in Figure 3-5, after the first Immediate Nm Transmission the interval between the NM messages is 'Immediate Nm CycleTime' for *(n-1)* times. Then, the usual interval 'Msg Cycle Time' is used again.

Note that "Any state except Repeat Message" in Figure 3-5 refers to 'Bus Sleep' and 'Prepare Bus Sleep'. If the setting 'Pn Handle Multiple Network Requests' is ON, it also refers to 'Ready Sleep' and 'Normal Operation'.

This feature is optional and must be enabled in the configuration. The number of messages that are transmitted faster ('Immediate Nm Transmissions') and the fast message cycle time ('Immediate Nm Cycle Time') can also be configured.

Note
This feature should not be confused with the possibility for an immediate transmission if the 'Com User Data Support' feature is on (chapter 3.13) and should also not be confused with the 'Immediate Restart Enabled' feature described in the following chapter.

Note
If the send request of an 'immediate transmission' is rejected by the lower layer (e.g. CanIf), the rejected send request is not considered as 'immediate transmission'. That means that the counter that counts the number of 'immediate transmissions' *ImmediateNmMsgCount* is not decremented. The transmission is retried in the next main function cycle.

Example: Let 'Immediate Nm Transmissions' := 2. The initial counter value of *ImmediateNmMsgCount* is 2.

1. When Repeat Message has just been entered, the first transmission request $TReq_A$ is rejected. *ImmediateNmMsgCount* is not decremented.
2. CanNm waits until the next main function cycle of CanNm (first interval t_{int1st}).
3. CanNm sends the NM message successfully. *ImmediateNmMsgCount* is decremented to 1.
4. CanNm waits 'Immediate Msg CycleTime' (second interval t_{int2nd}).
5. CanNm sends the next NM message successfully.
6. The counter is decremented to 0.
7. Then 'Msg Cycle Time' is waited until the next NM message is sent because *ImmediateNmMsgCount* is already 0.

If the first NM transmission request $TReq_A$ was successful (step 1), the second interval time t_{int2nd} would be 'Msg Cycle Time' instead of 'Immediate Msg CycleTime'.

3.16 Immediate Restart Enabled

This feature enables the possibility to send an additional NM message upon the transition from Prepare Bus Sleep to Repeat Message if and only if the network is requested actively (e.g. there is a request for Full Communication in ComM).

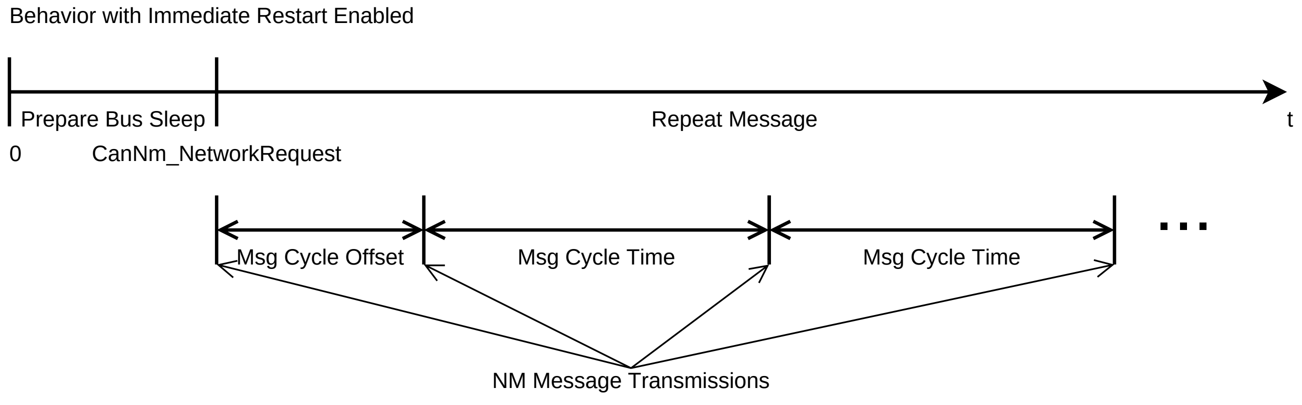

Figure 3-6 Behavior for NM Transmissions if Immediate Restart Enabled is ON

This behavior is depicted in Figure 3-6. Note that the only difference to the standard transmission behavior (i.e. 'Immediate Restart Enabled' would be OFF, for the behavior see also chapter 3.6.3) is the additional NM message right after Repeat Message has been entered.

Note
The additional NM message is only sent if the 'Msg Cycle Offset' setting is greater than 0 and if 'Immediate Nm Transmissions' = 0 (refer to chapter 3.15 for details).

Immediate Restart can be enabled per channel. Furthermore, Post Build Selectable is supported.

3.17 Car Wake-up

Every ECU shall be able to wake up all other ECUs of the car. This wake-up request information is contained in the NM message user data of an ECU. The central gateway ECU evaluates the Car Wake-up request information and wakes up all connected communication channels.

This feature is optional and must be configured.

3.17.1 Rx-Path

If the CAN NM receives a NM message it evaluates the user data content. If the Car Wake-up Bit is set and the Node ID passes a filter (if Node ID filter is enabled) the CAN NM notifies the NM via the following callback function:

`Nm_CarWakeUpIndication()` (5.4)

3.17.2 Tx-Path

For the transmission of the Car Wake-up Bit it must be set at the corresponding location within the NM user data. If the feature 'Com User Data Support' is used and the corresponding signal and I-PDU are configured for directly transmitting a changed signal the information is sent immediately. Refer also to chapter 3.13 'Com User Data Support'.

Info
It is recommended to use the feature 'Com User Data Support' for the transmission path.

3.18 Partial Networking

To reduce the power consumption of ECUs it shall be possible to switch off the communication stack during active bus communication. To control the shutdown and wake-up of such ECUs the CAN NM provides an additional algorithm. The NM message user data contains the information which partial networks (PN) are requested. This information is evaluated by the CAN NM and provided to the upper layer in an aggregated form by updating the content of additional I-PDUs in the Com.

Algorithm details are described in the following sub-chapters.

This feature and all its sub-features are optional and must be configured.

3.18.1 Availability of Partial Network Request Information

To distinguish between NM messages containing PN cluster request information (CRI) and NM messages without CRI a special bit in the control bit vector (bit 6) is used. Only if this bit is set the NM message contains PN information and will be processed by the algorithm.

3.18.2 Transmission of the CRI Bit in the NM User Data

The CAN NM sets the CRI Bit at bit position 6 in the Control Bit Vector to 1 for each channel if the Partial Networking feature is enabled on the corresponding channel.

For additional CRI bit handling refer to chapter 3.18.6.

3.18.3 Filter Algorithm for Received NM Messages

NM messages that are not relevant for an ECU with PN must be dropped. Therefore, the content of received NM messages is evaluated after the filter algorithm described in this section has been activated. Otherwise the usual way of receiving messages is being used. The filter is disabled after the initialization of the CAN NM module. The message reception filter is being activated after a call of CanNm_ConfirmPnAvailability (refer to chapter 5.5.2.1 'CanNm_ConfirmPnAvailability: Notification for Activating the PN Filter Functionality' for further details). The filter is disabled in case the channel is started again, by either an internal or external event and 'CanNm_ConfirmPnAvailability' was not called.

The filter algorithm works as follows:

If the CRI bit is cleared the NM message is not relevant for the ECU.

If the CRI bit is set the CAN NM evaluates the CRI content of the NM message. The location and the length of the CRI in the NM user data can be configured. Each bit within the CRI content represents one cluster. The corresponding cluster is being requested if and only if the bit that belongs to the cluster is set. Because not every cluster is relevant for the ECU a configurable PN filter mask is applied to the CRI content. Irrelevant cluster requests can be ignored by setting the corresponding bit in the filter to 0. If at least one bit within the received PN information matches with a bit in the PN filter mask the NM message is relevant for the ECU, otherwise the NM message is not relevant for the ECU.

If a NM message is not relevant and the configuration parameter 'All Nm Messages Keep Awake' is true the standard NM message reception handling is done, otherwise the NM message is ignored.

If a NM message is relevant the CAN NM performs the standard NM message reception and additionally the filtered PN content of this message is used for the further PN algorithm.

3.18.4 Aggregation of Requested Partial Networks

The CAN NM aggregates requested PN information by two slightly different algorithms. First the external (received) and internal (sent) PN requests are aggregated over all networks (channels) to a combined state called External Internal Requests Aggregated

(EIRA). Second only the external (received) PN requests are aggregated for each network to the so called External Requests Aggregated (ERA) state. Both algorithms can be activated independently in the configuration.

For the EIRA algorithm every received or sent NM message on any network is evaluated and the relevant PN information (according to the PN filter mask and the CRI bit) is combined to one aggregated state. Therefore, this state contains the information which partial networks are active on the whole ECU.

The ERA algorithm performs the evaluation of the received NM messages and storage of the relevant PN information (according to the PN filter mask and the CRI bit) per network. Therefore, the ERA state contains for each network the information which partial networks are requested by other ECUs and must be active due to external needs.

Whenever a cluster is requested the first time (i.e. a bit is set the first time within this PN information) the new request is stored and a timer is started. When the request is repeated before the timer elapses the timer is restarted. When the timer elapses, the request is deleted.

Any change (storing or deleting a request) within the EIRA or ERA leads to an update of the content of the EIRA or ERA I-PDU in the Com. Therefore, the following function is called with the corresponding EIRA or ERA PDU handle:

`PduR_CanNmRxIndication()` (5.4)

Note that one ERA I-PDU exists for each network.

3.18.5 Spontaneous Sending of NM Messages

When a new PN is internally requested the corresponding bit in the NM message user data will be set. This request must be immediately visible on the bus by sending the updated user data content as fast as possible. Therefore, two mechanisms can be used.

3.18.5.1 Using Com Transmission on Change Mechanism

When the NM user data is set via Com the signals can be configured for immediate transmission on change. This would lead to one additional NM message transmission whenever the content of the signal changes. Refer also to chapter 3.13 ‘Com User Data Support’.

To enable this behavior, the setting ‘Pn Handle Multiple Network Requests’ has to be turned OFF.

3.18.5.2 Using NM Request and Immediate Nm Transmission

When CAN NM is in Network Mode and the upper layer requests network again by calling the function ‘CanNm_NetworkRequest’ (see chapter 5.3.2.4.1‘CanNm_NetworkRequest: Request the Network’ for details) the CAN NM performs a state transition to Repeat Message. This leads to an immediate transmission of the NM message followed by several transmissions with a faster cycle time.

Caution
Note that the feature ‘Immediate Nm Transmission’ (refer to chapter 3.15 ‘Immediate Nm Transmissions’) must be enabled when using this mechanism for spontaneous sending of NM messages.

Note that this mechanism will only be active if PN feature is enabled.

To enable this behavior, the setting ‘Pn Handle Multiple Network Requests’ has to be turned ON.

3.18.6 Transmission of CRI bit without Partial Networking

It is possible to transmit an NM message with only the CRI bit set in the CBV, but Partial Networking feature disabled. This is done by enabling the switch ‘Cri Bit Always Enabled’ in the DaVinci Configurator.

4. Integration

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

4.1 Scope of Delivery

The delivery of the CANNM 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
`CanNm.c`	■		Source code of CAN NM. The user **must not** change this file!
`CanNm.h`	■	■	API of CAN NM. The user **must not** change this file!
`CanNm_Cbk.h`	■	■	API of CAN NM callback functions. The user **must not** change this file!

Table 4-1 Static files

Do not edit manually
The static files listed above must not be edited by the user!

4.1.2 Dynamic Files

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

File Name	Description
`CanNm_Cfg.c`	Pre-compile variant configuration source file. The user **must not** change this file!
`CanNm_Cfg.h`	Configuration header file for CAN NM. The user **must not** change this file!
`CanNm_Lcfg.c`	Link-time variant Configuration source file. The user **must not** change this file!
`CanNm_Pbcfg.c`	Post-build variant Configuration source file. The user **must not** change this file!

Table 4-2 Generated files

Do not edit manually
The dynamic files listed above must not be edited by the user! They should be generated with the configuration tool to guarantee valid parameters.

4.2 Include Structure

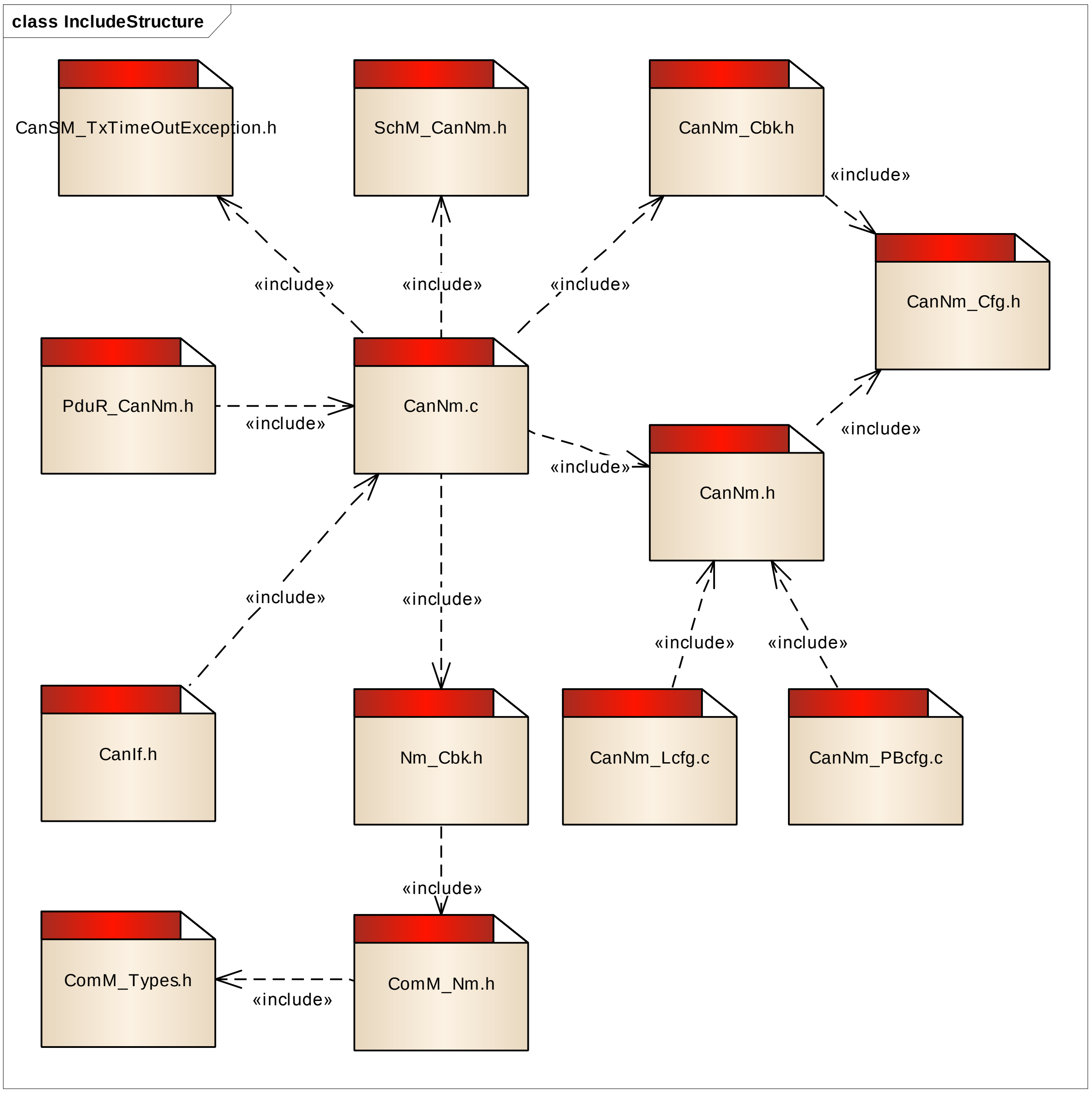

Figure 4-1 Include structure

4.3 Main Functions

The CAN NM contains one main function that must be called cyclically on task level. The default timing value is 10 milliseconds. The call cycle time value for the main function must be set in the configuration settings (Setting: 'Main Function Period').

Note
In an AUTOSAR environment where the BSW Scheduler (SchM) is used the main functions are called by the SchM and must not be called by the application.

4.4 Critical Sections

Critical Sections are handled by the RTE [7]. They are automatically configured by the DaVinci Configurator. User interaction is only necessary by updating the internal behavior using the solving action in DaVinci Configurator. It is signaled as a Warning in the Validation tab.

The CAN NM calls the following function when entering a critical section:

SchM_Enter_CanNm_CANNM_EXCLUSIVE_AREA_i() (5.4)

When the critical section is left the following function is called by the CAN NM:

SchM_Enter_CanNm_CANNM_EXCLUSIVE_AREA_i() (5.4)

Details which section needs what kind of interrupt lock are provided in chapter 4.5 'Critical Section Codes'.

4.5 Critical Section Codes

The CAN NM provides several critical section codes which must lead to corresponding interrupt locks, described in the following table:

Critical Section Define	Interrupt Lock
`CANNM_EXCLUSIVE_AREA_0`	No interruption by any interrupt is allowed. Therefore, this section must always lock global interrupts.
`CANNM_EXCLUSIVE_AREA_1`	No interruption of CanNm_MainFunction or CanNm_RequestBusSynchronization by CanNm_SetUserData or CanNm_SetSleepReadyBit allowed. This means that global interrupts must be used for this section only if CanNm_MainFunction or CanNm_RequestBusSynchronization can be interrupted by one of the following functions: > CanNm_SetUserData > CanNm_SetSleepReadyBit Otherwise no interrupt locks are necessary.

`CANNM_EXCLUSIVE_AREA_2`	No interruption of CanNm_SetUserData by CanNm_MainFunction or CanNm_RequestBusSynchronization allowed. This means that global interrupts must be locked if CanNm_SetUserData can be interrupted by the following functions: > CanNm_MainFunction > CanNm_RequestBusSynchronization Otherwise no interrupt locks are necessary.
`CANNM_EXCLUSIVE_AREA_3`	No interruption of CanNm_SetSleepReadyBit by CanNm_MainFunction or CanNm_RequestBusSynchronization allowed. This means that global interrupts must be locked if CanNm_SetSleepReadyBit can be interrupted by the following functions: > CanNm_MainFunction > CanNm_RequestBusSynchronization Otherwise no interrupt locks are necessary.
`CANNM_EXCLUSIVE_AREA_4`	No interruption of CanNm_RxIndication by CanNm_GetUserData or CanNm_GetPduData allowed. This means that global interrupts must be locked if CanNm_RxIndication can be interrupted by the following functions: > CanNm_GetUserData > CanNm_GetPduData Otherwise no interrupt locks are necessary.
`CANNM_EXCLUSIVE_AREA_5`	No interruption of CanNm_GetUserData or CanNm_GetPduData by CanNm_RxIndication allowed. This means that global interrupts must be locked if CanNm_GetUserData or CanNm_GetPduData can be interrupted by the following functions: > CanNm_RxIndication Otherwise no interrupt locks are necessary.
`CANNM_EXCLUSIVE_AREA_6`	No interruption of CanNm_MainFunction by CanNm_RequestBusSynchronization allowed. This means that global interrupts must be locked if CanNm_MainFunction can be interrupted by the following function: > CanNm_RequestBusSynchronization Otherwise no interrupt locks are necessary. Note: This critical section can be left empty if CanNm_MainFunction and Nm_MainFunction have the same task.

Table 4-3 Critical Section Codes

5. API Description

For an interfaces overview please see Figure 2-2.

5.1 Data Types

The software module CAN NM uses the standard AUTOSAR data types that are defined within `Std_Types.h` and the platform specific data types that are defined within `Platform_Types.h` and the Communication Stack Types defined within `ComStack_Types.h`. Furthermore, the standard AUTOSAR NM Stack Types defined within `NmStack_Types.h` are used.

CAN NM also uses the Communication Stack Types defined within `ComStack_Types.h`.

5.2 Global Constants

5.2.1 AUTOSAR Specification Version

The version of AUTOSAR specification on which the appropriate implementation is based on is provided by three BCD coded defines:

Name	Type	Description
`CANNM_AR_RELEASE_MAJOR_VERSION`	BCD	Contains the major specification version number.
`CANNM_AR_RELEASE_MINOR_VERSION`	BCD	Contains the minor specification version number.
`CANNM_AR_RELEASE_REVISION_VERSION`	BCD	Contains the patch level specification version number.

Table 5-1 Specification Version API Data

5.2.2 Component Versions

The source code versions of CAN NM are provided by three BCD coded macros (and additionally as constants):

Name	Type	Description
`CANNM_SW_MAJOR_VERSION (CanNm_MainVersion)`	BCD	Contains the major component version number.
`CANNM_SW_MINOR_VERSION (CanNm_SubVersion)`	BCD	Contains the minor component version number.
`CANNM_SW_PATCH_VERSION (CanNm_ReleaseVersion)`	BCD	Contains the patch level component version number.

Table 5-2 Component Version API Data

These constants are declared as external and can be read by the application at any time.

5.2.3 Vendor and Module ID

CanNm provides the vendor identifier according to AUTOSAR as defined in the next table.

Name	Type	Description	Value
`CANNM_VENDOR_ID`	-	Vendor ID according to AUTOSAR.	30
`CANNM_MODULE_ID`	-	Module ID according to AUTOSAR.	31

Table 5-3 Vendor/Module ID

5.3 Services Provided by CANNM

5.3.1 Administrative Functions

5.3.1.1 CanNm_Init: Initialization of CAN NM

Prototype	
`void CanNm_Init (const CanNm_ConfigType * const cannmConfigPtr)`	
Parameter	
`cannmConfigPtr`	Configuration structure for initializing the module
Return code	
–	-
Functional Description	
Initialization of the CAN Network Management (CANNM041) and its internal state machine. By default, the NM starts in the Bus-Sleep Mode.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > The function CanNm_Init() must be called before any other CanNm service function is called (except CanNm_InitMemory()). > During the execution of the function CanNm_Init(), it must be ensured that the execution is not interrupted by any other function of the CanNm module. This can for instance be accomplished by > global interrupt locks OR > CAN interrupt locks. > In Variant post-build-selectable and post-build-loadable a valid configuration pointer must be passed in the CanNm_Init function call. > This function is non-reentrant. > This function is synchronous.	
Expected Caller Context	
> Task level	

Table 5-4 CanNm_Init

5.3.1.2 CanNm_MainFunction: Main Function for All Channel Instances

Prototype	
`void CanNm_MainFunction (void)`	
Parameter	
–	-
Return code	
–	-

Functional Description
Main function of the CanNm which processes the NM algorithm. This function is responsible to handle all CanNm instances. (CANNM234).
Particularities and Limitations
> Service ID: see table 'Service IDs' > This function is non-reentrant. > This function is synchronous. > This function is called by SchM.
Expected Caller Context
> Task level

Table 5-5 CanNm_MainFunction

5.3.1.3 CanNm_InitMemory: Memory Initialization

Prototype	
`void CanNm_InitMemory (void)`	
Parameter	
–	-
Return code	
–	-
Functional Description	
Initialize Memory, so that expected start values are set.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is non-reentrant. > This function is synchronous. > This function is called by the Application.	
Expected Caller Context	
> System startup	

Table 5-6 CanNm_InitMemory

5.3.2 Service Functions

5.3.2.1 CanNm_GetVersionInfo: Version Information API

Prototype	
`void CanNm_GetVersionInfo (Std_VersionInfoType *versioninfo)`	
Parameter	
`versioninfo`	Pointer to where to store the version information of this module

Return code	
–	-
Functional Description	
CanNm_GetVersionInfo() returns version information, vendor ID and AUTOSAR module ID of the component. The versions are BCD-coded.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is reentrant. > This function is synchronous. > This function is available if `CANNM_VERSION_INFO_API` is `STD_ON`	
Expected Caller Context	
> Task level	

Table 5-7 CanNm_GetVersionInfo

5.3.2.2 CanNm_GetState: Get the State of the Network Management

Prototype

```
Std_ReturnType CanNm_GetState (const NetworkHandleType  nmChannelHandle,
                               Nm_StateType *  const    nmStatePtr,
                               Nm_ModeType *   const    nmModePtr)
```

Parameter	
nmChannelHandle	Index of the network channel
nmStatePtr	Pointer where the state of the Network Management shall be copied to
nmModePtr	Pointer where the mode of the Network Management shall be copied to
Return code	
Std_ReturnType	E_OK - No error E_NOT_OK - Getting the NM state has failed
Functional Description	
Return current state and mode of the network management (CANNM223).	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is reentrant. > This function is synchronous. > This function is called by NM Interface.	
Expected Caller Context	
> Task and interrupt level	

Table 5-8 CanNm_GetState

5.3.2.3 CanNm_PassiveStartUp: Wake up the Network Management

Prototype	
`Std_ReturnType` **`CanNm_PassiveStartUp`** `(const NetworkHandleType nmChannelHandle)`	
Parameter	
`nmChannelHandle`	Index of the network channel
Return code	
`Std_ReturnType`	E_OK - No error E_NOT_OK - Start of network management has failed
Functional Description	
Starts the NM from the Bus Sleep Mode and triggers a transition to the Network Mode (Repeat Message State) (CANNM211). This service has no effect if the current state is not equal to Bus Sleep Mode or Prepare Bus Sleep Mode. In that case E_NOT_OK is returned.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is reentrant. > This function is asynchronous. > This function is called by NM Interface.	
Expected Caller Context	
> Task and interrupt level	

Table 5-9 CanNm_PassiveStartUp

5.3.2.4 Wake-up Registration

5.3.2.4.1 CanNm_NetworkRequest: Request the Network

Prototype	
`Std_ReturnType` **`CanNm_NetworkRequest`** `(const NetworkHandleType nmChannelHandle)`	
Parameter	
`nmChannelHandle`	Index of the network channel
Return code	
`Std_ReturnType`	E_OK - No error E_NOT_OK - Requesting bus-communication has failed
Functional Description	
Request the network, since ECU needs to communicate on the bus (CANNM213).	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is reentrant. > This function is asynchronous. > This function is called by NM Interface.	

Expected Caller Context
> Task and interrupt level

Table 5-10 CanNm_NetworkRequest

5.3.2.4.2 CanNm_NetworkRelease: Release the Network

Prototype	
`Std_ReturnType` **`CanNm_NetworkRelease`** `(const NetworkHandleType nmChannelHandle)`	
Parameter	
`nmChannelHandle`	Index of the network channel
Return code	
`Std_ReturnType`	E_OK - No error E_NOT_OK - Releasing bus-communication has failed
Functional Description	
Release the network, since ECU doesn't have to communicate on the bus (CANNM214).	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is reentrant. > This function is asynchronous. > This function is called by NM Interface.	
Expected Caller Context	
> Task and interrupt level	

Table 5-11 CanNm_NetworkRelease

5.3.2.5 User Data Handling

5.3.2.5.1 CanNm_SetUserData: Set User Data

Prototype	
`Std_ReturnType` **`CanNm_SetUserData`** `( const NetworkHandleType nmChannelHandle, const uint8 * const nmUserDataPtr)`	
Parameter	
`nmChannelHandle`	Index of the network channel
`nmUserDataPtr`	Pointer to User data for the next transmitted NM message shall be copied from
Return code	
`Std_ReturnType`	E_OK - No error E_NOT_OK - Setting of user data has failed
Functional Description	
Set user data for NM messages transmitted next on the bus (CANNM217).	

Particularities and Limitations
> Service ID: see table 'Service IDs' > This function is non-reentrant. > This function is synchronous. > This function is called from NM Interface.
Expected Caller Context
> Task and interrupt level

Table 5-12 CanNm_SetUserData

5.3.2.5.2 CanNm_GetUserData: Get User Data

Prototype	
`Std_ReturnType` **`CanNm_GetUserData`** `( const NetworkHandleType nmChannelHandle, uint8 * const nmUserDataPtr)`	
Parameter	
`nmChannelHandle`	Index of the network channel
`nmUserDataPtr`	Pointer where user data out of the last received NM message shall be copied to
Return code	
`Std_ReturnType`	E_OK - No error E_NOT_OK - Getting of user data has failed
Functional Description	
Get user data out of the last NM messages received from the bus (CANNM218).	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is reentrant. > This function is synchronous. > This function is called from NM Interface.	
Expected Caller Context	
> Task and interrupt level	

Table 5-13 CanNm_GetUserData

5.3.2.5.3 CanNm_GetPduData: Get NM PDU Data

Prototype	
`Std_ReturnType` **`CanNm_GetPduData`** `( const NetworkHandleType nmChannelHandle, uint8 * const nmPduDataPtr)`	
Parameter	
`nmChannelHandle`	Index of the network channel
`nmPduDataPtr`	Pointer where PDU Data out of the most recently received NM message shall be copied to

Return code	
`Std_ReturnType`	E_OK - No error E_NOT_OK - Getting the PDU data has failed
Functional Description	
Get the whole PDU data out of the last NM message received from the bus (CANNM138).	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is reentrant. > This function is asynchronous. > This function is called from NM Interface.	
Expected Caller Context	
> Task and interrupt level	

Table 5-14 CanNm_GetPduData

5.3.2.6 Node Detection

5.3.2.6.1 CanNm_RepeatMessageRequest: Set Repeat Message Request Bit

Prototype	
`Std_ReturnType` **`CanNm_RepeatMessageRequest`** `(const NetworkHandleType nmChannelHandle)`	
Parameter	
`nmChannelHandle`	Index of the network channel
Return code	
`Std_ReturnType`	E_OK - No error E_NOT_OK - Repeat Message Request has failed
Functional Description	
Request state change to Repeat Message State (CANNM221).	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is reentrant. > This function is asynchronous. > This function is called from NM Interface	
Expected Caller Context	
> Task and interrupt level	

Table 5-15 CanNm_RepeatMessageRequest

5.3.2.6.2 CanNm_GetNodeIdentifier: Get Node Identifier

Prototype	
`Std_ReturnType` **`CanNm_GetNodeIdentifier`** `( const NetworkHandleType nmChannelHandle, uint8 * const nmNodeIdPtr)`	
Parameter	
`nmChannelHandle`	Index of the network channel
`nmNodeIdPtr`	Pointer where node identifier from the last received NM message shall be copied to
Return code	
`Std_ReturnType`	E_OK - No error E_NOT_OK - Getting of node identifier has failed
Functional Description	
Get node identifier of the last received NM message (CANNM219).	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is reentrant. > This function is synchronous. > This function is called from NM Interface	
Expected Caller Context	
> Task and interrupt level	

Table 5-16 CanNm_GetNodeIdentifier

5.3.2.6.3 CanNm_GetLocalNodeIdentifier: Get Local Node Identifier

Prototype	
`Std_ReturnType` **`CanNm_GetLocalNodeIdentifier`** `( const NetworkHandleType nmChannelHandle, uint8 * const nmNodeIdPtr)`	
Parameter	
`nmChannelHandle`	Index of the network channel
`nmNodeIdPtr`	Pointer where node identifier of the local node shall be copied to
Return code	
`Std_ReturnType`	E_OK - No error E_NOT_OK - Getting of local node identifier has failed
Functional Description	
Get node identifier configured for the local node (CANNM220).	

Particularities and Limitations
> Service ID: see table 'Service IDs' > This function is reentrant. > This function is synchronous. > This function is called from NM Interface
Expected Caller Context
> Task and interrupt level

Table 5-17 CanNm_GetLocalNodeIdentifier

5.3.2.7 Bus Synchronization

5.3.2.7.1 CanNm_RequestBusSynchronization: Synchronization of Networks

Prototype	
`Std_ReturnType` **`CanNm_RequestBusSynchronization`** `( const NetworkHandleType nmChannelHandle)`	
Parameter	
`nmChannelHandle`	Index of the network channel
Return code	
`Std_ReturnType`	E_OK - No error E_NOT_OK - Requesting bus synchronization has failed
Functional Description	
Request bus synchronization (CANNM226) (Transmission of an asynchronous NM message to support coordination of coupled networks).	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is non-reentrant. > This function is synchronous. > This function is called from NM Interface	
Expected Caller Context	
> Task level	

Table 5-18 CanNm_RequestBusSynchronization

5.3.2.8 Remote Sleep Indication

5.3.2.8.1 CanNm_CheckRemoteSleepIndication: Check for Remote Sleep Indication

Prototype
`Std_ReturnType` **`CanNm_CheckRemoteSleepIndication`** `( const NetworkHandleType nmChannelHandle, boolean * const nmRemoteSleepIndPtr)`

Parameter	
`nmChannelHandle`	Index of the network channel
`nmRemoteSleepIndPtr`	Pointer where current remote sleep state is copied to.
Return code	
`Std_ReturnType`	E_OK - No error E_NOT_OK - Checking remote sleep indication has failed
Functional Description	
Check if remote sleep was indicated or not (CANNM227).	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is reentrant. > This function is synchronous. > This function is called from NM Interface	
Expected Caller Context	
> Task and interrupt level	

Table 5-19 CanNm_CheckRemoteSleepIndication

5.3.2.9 NM Message Transmission Request

5.3.2.9.1 CanNm_Transmit: Spontaneous NM Message Transmission

Prototype	
`Std_ReturnType` **`CanNm_Transmit`** `(PduIdType CanNmTxPduId, const PduInfoType *PduInfoPtr)`	
Parameter	
`CanNmTxPduId`	L-PDU handle of CAN L-PDU to be transmitted. This handle specifies the corresponding CAN LPDU ID and implicitly the CAN Driver instance as well as the corresponding CAN controller device.
`PduInfoPtr`	Pointer to a structure with CAN L-PDU related data: DLC and pointer to CAN L-SDU buffer.
Return code	
`Std_ReturnType`	E_OK - No error E_NOT_OK - transmit request has not been accepted due to wrong state
Functional Description	
This function is used by the PduR to trigger a spontaneous transmission of an NM message with the provided NM User Data (CANM331).	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is reentrant. > This function is synchronous. > This function is called from PduR	

Expected Caller Context
> Task and interrupt level

Table 5-20 CanNm_Transmit

5.3.2.10 NM Message Transmission Cancellation

5.3.2.10.1 CanNm_CancelTransmit: Empty function

Prototype	
`Std_ReturnType` **`CanNm_CancelTransmit`** `( PduIdType CanNmTxSduId )`	
Parameter	
`CanNmTxSduId`	Parameter is not used.
Return code	
`Std_ReturnType`	E_OK - No error E_NOT_OK - component was not initialized
Functional Description	
This function has no functional purpose. This function is required for compatibility reasons.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is reentrant. > This function is synchronous. > This function is called from PduR	
Expected Caller Context	
> Task and interrupt level	

Table 5-21 CanNm_CancelTransmit

5.3.2.11 Communication Control Service

5.3.2.11.1 CanNm_DisableCommunication: Disable NM Message Transmission

Prototype	
`Std_ReturnType` **`CanNm_DisableCommunication`** `(const NetworkHandleType nmChannelHandle)`	
Parameter	
`nmChannelHandle`	Index of the network channel
Return code	
`Std_ReturnType`	E_OK - No error E_NOT_OK - Disable NM Message transmission control status has failed
Functional Description	
Disable NM message transmission control status (CANNM215).	

Particularities and Limitations
> Service ID: see table 'Service IDs' > This function is reentrant. > This function is asynchronous. > This function is called from NM Interface
Expected Caller Context
> Task and interrupt level

Table 5-22 CanNm_DisableCommunication

5.3.2.11.2 CanNm_EnableCommunication: Enabled NM Message Transmission

Prototype	
`Std_ReturnType` **`CanNm_EnableCommunication`** `( const NetworkHandleType nmChannelHandle)`	
Parameter	
`nmChannelHandle`	Index of the network channel
Return code	
`Std_ReturnType`	E_OK - No error E_NOT_OK - Enabling NM Message transmission control status has failed
Functional Description	
Enable NM message transmission control status (CANNM216).	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is reentrant. > This function is asynchronous. > This function is called from NM Interface	
Expected Caller Context	
> Task and interrupt level	

Table 5-23 CanNm_EnableCommunication

5.3.2.12 Coordinator Synchronization Support

5.3.2.12.1 CanNm_SetSleepReadyBit: Set Sleep Ready Bit in the CBV

Prototype	
`Std_ReturnType` **`CanNm_SetSleepReadyBit`** `( const NetworkHandleType nmChannelHandle, const boolean nmSleepReadyBit)`	
Parameter	
`nmChannelHandle`	Index of the network channel
`nmSleepReadyBit`	Value written to Ready Sleep Bit in CBV

Return code		
`Std_ReturnType`	E_OK E_NOT_OK	- No error - Writing of Sleep Ready Bit has failed
Functional Description		
Set the NM Coordinator Sleep Ready bit in the Control Bit Vector (CANNM338).		
Particularities and Limitations		
> Service ID: see table 'Service IDs' > This function is non-reentrant. > This function is synchronous. > This function is called from NM Interface		
Expected Caller Context		
> Task level		

Table 5-24 CanNm_SetSleepReadyBit

5.4 Services Used by CANNM

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

Component	API
CanIf	CanIf_Transmit[8]
CanSM	CanSM_TxTimeoutException[9]
DET	Det_ReportError[10]
NM	Nm_CarWakeUpIndication[11] Nm_BusSleepMode Nm_CoordReadyToSleepIndication[12] Nm_CoordReadyToSleepCancellation[12] Nm_NetworkMode Nm_NetworkStartIndication Nm_PduRxIndication[13] Nm_CanNm_PduRxIndication[14] Nm_PrepareBusSleepMode Nm_RemoteSleepCancellation[15] Nm_RemoteSleepIndication[15]

[8] Service only used if the feature ‘Passive Mode’ is disabled
[9] Service only used if ‘Immediate Tx Conf Enabled’ is disabled and ‘Pn Enabled’ is enabled and if CanSM provides this function
[10] Service only used if the feature ‘Dev Error Detect’ is enabled
[11] Service only used if the feature ‘Car Wake Up Rx Enabled’ is enabled.
[12] Service only used if the feature ‘Coordinator Sync Support’ is enabled.
[13] Service only used if the feature ‘Pdu Rx Indication Enabled’ is enabled.
[14] Service only used if the feature ‘Bus Nm Specific Pdu Rx Indication Enabled‘ is enabled in NmIf.

Component	API
	Nm_RepeatMessageIndication[16] Nm_StateChangeNotification[17] Nm_TxTimeoutException[8,18]
PduR	PduR_CanNmTriggerTransmit[8,19] PduR_CanNmTxConfirmation[8,19,20] PduR_CanNmRxIndication[21]
SchM	SchM_Enter_CanNm_CANNM_EXCLUSIVE_AREA_i SchM_Exit_CanNm_CANNM_EXCLUSIVE_AREA_i for i=0,…,5

Table 5-25 Services used by the CANNM

5.5 Callback Functions

5.5.1 Callback Functions from CAN Interface

5.5.1.1 CanNm_TxConfirmation: NM Message Confirmation Function

Prototype	
`void CanNm_TxConfirmation (PduIdType TxPduId)`	
Parameter	
`TxPduId`	ID of CAN NM PDU that has been transmitted
Return code	
–	-
Functional Description	
This function is called by the CAN Interface after a CAN NM PDU has been successfully transmitted (CANNM228).	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is reentrant. > This function is asynchronous. > This function is called from data link layer	
Expected Caller Context	
> Task and interrupt level	

Table 5-26 CanNm_TxConfirmation

[15] Service only used if the feature ‘Remote Sleep Ind Enabled’ is enabled.
[16] Service only used if the feature ‘Repeat Msg Ind Enabled’ is enabled.
[17] Service only used if the feature ‘State Change Ind Enabled’ is enabled.
[18] Service only used if the feature ‘Immediate Tx Conf Enabled’ is enabled.
[19] Service only used if the feature ‘Com User Data Support’ is enabled.
[20] Service only used if the feature ‘Immediate Txconf Enabled’ is disabled.
[21] Service only used if the features ‘Pn Eira Calc Enabled’ or ‘Pn Era Calc Enabled’ is enabled.

5.5.1.2 CanNm_RxIndication: NM Message Indication

Prototype	
`void` **`CanNm_RxIndication`** `(PduIdType RxPduId, const PduInfoType *PduInfoPtr)`	
Parameter	
`RxPduId`	ID of CAN NM PDU that has been received
`PduInfoPtr`	Pointer to a PduInfoType containing the received CAN NM SDU and its length
Return code	
–	-
Functional Description	
This function is called by the CAN Interface after a CAN L-PDU has been received (CANNM231).	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is non-reentrant. > This function is synchronous. > This function is called from data link layer	
Expected Caller Context	
> Task and interrupt level	

Table 5-27 CanNm_RxIndication

5.5.2 Callback Function from CAN State Manager

5.5.2.1 CanNm_ConfirmPnAvailability: Notification for Activating the PN Filter Functionality

Prototype	
`void` **`CanNm_ConfirmPnAvailability`** `(const NetworkHandleType nmChannelHandle)`	
Parameter	
`nmChannelHandle`	Index of the network channel
Return code	
–	-
Functional Description	
Enables the PN filter functionality on the indicated NM channel (CANM344).	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is reentrant. > This function is synchronous. > This function is called by CanSM	
Expected Caller Context	
> Task and interrupt level	

Table 5-28 CanNm_ConfirmPnAvailability

6. Glossary and Abbreviations

6.1 Glossary

Term	Description
Confirmation	Notification by the data link layer on asynchronous successful transmission of a CAN message
Identifier	Identifies a CAN message
Indication	Notification by the data link layer on asynchronous reception of a CAN message
Message	One or more signals are assigned to each message.
Signal	Signals describe the significance of the individual data segments within a message. Typically, bits, bytes or words are used for data segments but individual bit combinations are also possible. In the CAN database, each data segment is assigned a symbolic name, a value range, a conversion formula and a physical unit, as well as a list of receiving nodes.

Table 6-1 Glossary

6.2 Abbreviations

Abbreviation	Description
API	**A**pplication **P**rogramming **I**nterface
AUTOSAR	**Aut**omotive **O**pen **S**ystem **Ar**chitecture
BswM	**B**asic **S**oftware **M**ode **Manager**
CAN	**C**ontroller **A**rea **N**etwork
CanIf	**Can** **I**nterface
CCL	**C**ommunication **C**ontrol **L**ayer
ComM	**Com**munication **M**anager
CRI	Partial Network **C**luster **R**equest **I**nformation
DET	**D**evelopment **E**rror **T**racer
DEM	**D**iagnostic **E**vent **M**anager
DLC	**D**ata **L**ength **C**ode (Number of data bytes of a CAN message)
DLL	**D**ata **l**ink **l**ayer
ECU	**E**lectronic **C**ontrol **U**nit
EIRA	**E**xternal **I**nternal **R**equests **A**ggregated
ERA	**E**xternal **R**equests **A**ggregated
FIBEX	**Fi**eld **B**us **Ex**change
ID	**Id**entifier (of a CAN message)
IL	**I**nteraction **L**ayer

I-PDU	**I**nteraction Layer **P**rotocol **D**ata **U**nit
ISR	**I**nterrupt **S**ervice **R**outine
LIN	**L**ocal **I**nterconnect **N**etwork
MISRA	**M**otor **I**ndustry **S**oftware **R**eliability **A**ssociation
NM	**N**etwork **M**anagement
PDU	**P**rotocol **D**ata **U**nit
PN	**P**artial **N**etwork / **P**artial **N**etworking
RAM	**R**andom **A**ccess **M**emory
RI	**R**eference **I**mplementation (Reference Implementation of the CAN-Driver High Level part)
ROM	**R**ead **O**nly **M**emory
SchM	**Sch**edule **M**anager (BSW Scheduler)
SRS	**S**ystem **R**equirements **S**pecification (used for AUTOSAR documents)
SWS	**S**oftware **S**pecification (used for AUTOSAR documents)
UDP	**U**ser **D**atagram **P**rotocol

Table 6-2 Abbreviations

7. Contact

Visit our website for more information on

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

www.vector.com