VECTOR

MICROSAR Classic Network Management Interface

MICROSAR Classic Network Management Interface

Technical Reference

Version 14.00.00

Authors	visgle, vismdr, vispps,smarcelin
Status	Released

Document Information

History

Author	Date	Version	Remarks
visoh	2011-03-11	1.00.00	ESCAN00048893: Initial Version of Nm AUTOSAR Release 4
visrpp	2012-07-20	2.00.00	ESCAN00058346: Updated to ASR 4.0.3
vismdr	2013-05-11	2.01.00	ESCAN00063146: Updated Figure 1-1 ESCAN00067284: Added chapter ‘Component History’ (deleted), merged Chapter ‘AUTOSAR Standard Compliance’ with chapter 2.1, removed chapter ‘Compiler Abstraction and Memory Mapping’, various improvements ESCAN00067285: Rewritten chapter 2.9
vismdr	2013-10-01	3.00.00	ESCAN00068794: Added J1939Nm Support ESCAN00068989: Adapted conditions for availability of Nm_PrepareBusSleepMode ESCAN00070593: Added Runtime Measurement Support as ‘Feature Beyond the AUTOSAR Standard’ ESCAN00070804: Updated Figure 1-1
vismdr	2014-02-14	3.01.00	ESCAN00071769: Updated chapters ‘Component History’ (deleted), 2.1, 4.2, 4.4, 4.6, added chapter 2.10 ESCAN00073703: Updated Figure 1-1 ESCAN00073704: Updated chapter 2.1.3 ESCAN00073705: Updated chapter 2.12.1 ESCAN00073707: Added chapter 3.3.2 ESCAN00073709: Updated chapter 4.1
vismdr	2014-04-17	4.00.00	ESCAN00074299: Added chapter 2.1.2.8 ESCAN00075103: Updated chapter 2 ESCAN00075105: Updated Figure 1-1 ESCAN00075012: Updated Figure 2-1, added chapter 2.1.1.2 ESCAN00075311: Updated Figure 2-1 ESCAN00075118: Updated chapters 2.1.2, 2.4.4 ESCAN00075812: Added chapter 2.3.1
vismdr	2014-10-07	5.00.00	ESCAN00076764: Updated chapters 1, 2.1 ESCAN00078802: Updated chapter 4.2.15 ESCAN00078803: Updated Figure 1-1
vismdr	2015-03-23	6.00.00	ESCAN00081207 Updated Table 2-7, updated Table 4-1
visgle	2015-07-21	7.00.00	ESCAN00080959: Updated chapter

			2.1.2.1 ESCAN00083545: Added chapters: 2.1.2.9, 4.4.8 and 4.6.1.4. ESCAN00083555: Updated chapters.4.4.2, 4.4.3, 4.4.4, 4.4.5 and 4.4.6 ESCAN00084339: Updated chapter 2.12.1
visgle	2015-12-16	8.00.00	ESCAN00084773 Updated chapter 4.6.1, 2.1.2.3, 2.9.2, 3.2 ESCAN00085986: Updated chapter 4.2.16 ESCAN00087098: Updated chapter 2.1.1
vismdr	2016-03-08	9.00.00	ESCAN00088776: Updated Figure 1-1 ESCAN00088777: Update to new CI layout ESCAN00088778: Extended chapter 2.10.1
visgle	2016-07-04	10.00.00	ESCAN00089481 Extended chapters: 2.1.2.6, 2.4.2 and 2.4.6.
vispps	2021-03-31	13.00.00	NMM-1127: Deleted chapter ‘Component History’. Updated chapters 1.1, 2.1.2, 2.8, 2.11, 2.12.1, 3.1.2, 3.2, 4.2.1, 4.2.7 and 4.3. Added chapters 2.7, 4.2.18, 4.2.19 and 4.2.20.
vispps	2021-09-13	13.01.00	NMM-1414: Added chapters 3.3.3, 3.3.4 and 3.4.
vispps	2022-01-14	13.02.00	NMM-1843: Added chapters 4.2.21 and 4.4.16. Updated chapters 2.1.2, 2.12.1 and 4.5.
vispps	2022-08-04	13.03.00	Product name updated to MICROSAR Classic
smarcelin	2023-09-25	14.00.00	No changes in this file

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	Specification of Network Management Interface	V3.0.0
[2]	AUTOSAR	Specification of Development Error Tracer	V3.2.0
[3]	AUTOSAR	Specification of Diagnostic Event Manager	V4.2.0
[4]	AUTOSAR	List of Basic Software Modules	V1.6.0
[5]	AUTOSAR	Specification of RTE	V3.2.0
[6]	Vector	TechnicalReference MICROSAR Classic Can Network Management	-

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 Introduction

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

Supported AUTOSAR Release*:	4	
Supported Configuration Variants:	pre-compile, post-build-selectable	
Vendor ID:	NM_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	NM_MODULE_ID	29 decimal (according to ref. [4])

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

The Nm module acts as an adaptation layer between the AUTOSAR BSW module ComM and the AUTOSAR BSW bus-specific network management modules (BusNm), e.g. CanNm. Therefore a call of the ComM on a network is forwarded to the corresponding BusNm on this network. Callback functions from a BusNm are forwarded to the ComM. This functionality is also called ‘basic functionality’.

Beside the standard BusNm modules defined by AUTOSAR, the Nm module can also support generic lower layer modules to allow the integration of OEM specific or legacy NM components, e.g. OSEK NM (NmOsek). For this support it is required that the lower layer modules implements the requirements for a generic BusNm.

Optionally the Nm module provides a coordination algorithm to perform a synchronous shutdown handling of several connected networks and/or multiple BusNms on one channel.

1.1 Architecture Overview

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

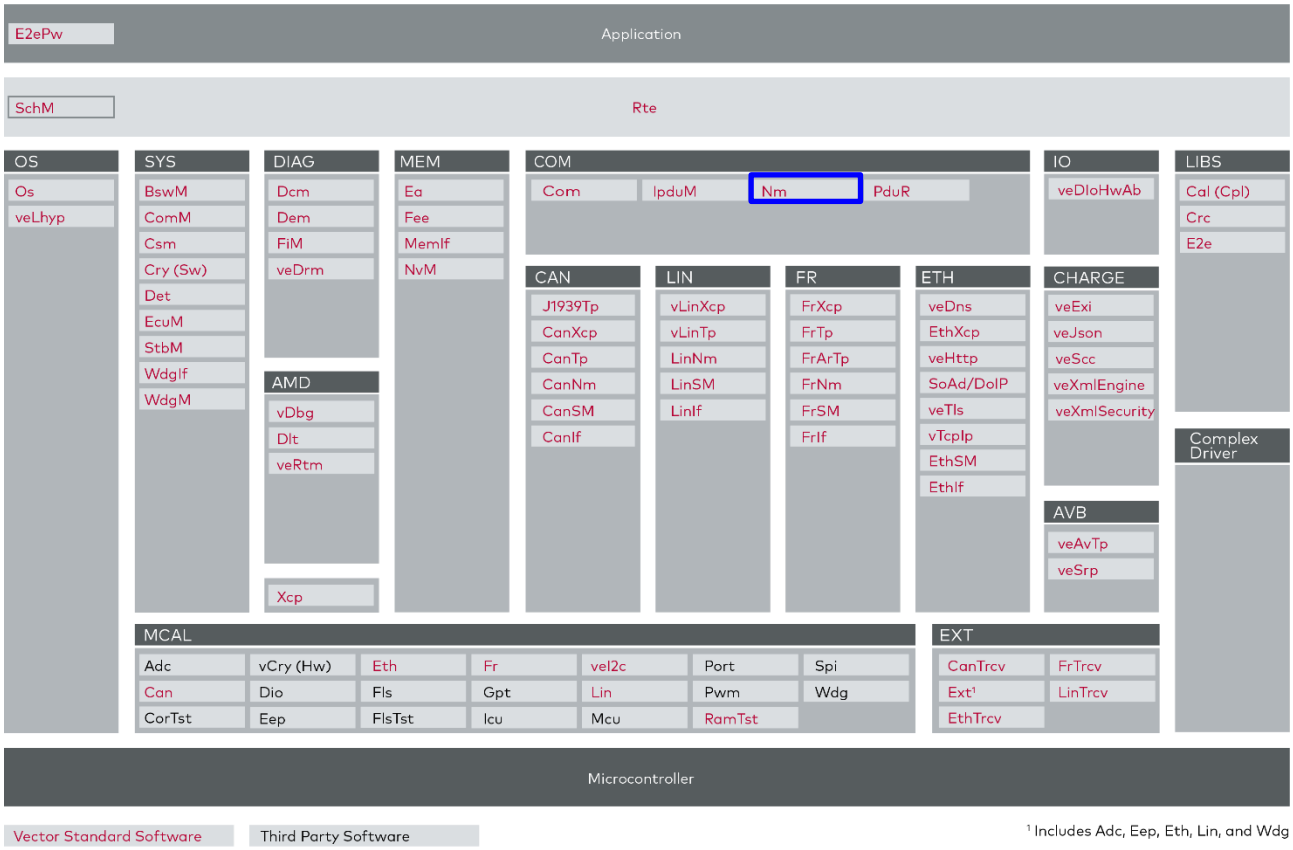

Figure 1-1 AUTOSAR 4.x Architecture Overview

The next figure shows the interfaces to adjacent modules of the Nm. These interfaces are described in chapter 4.

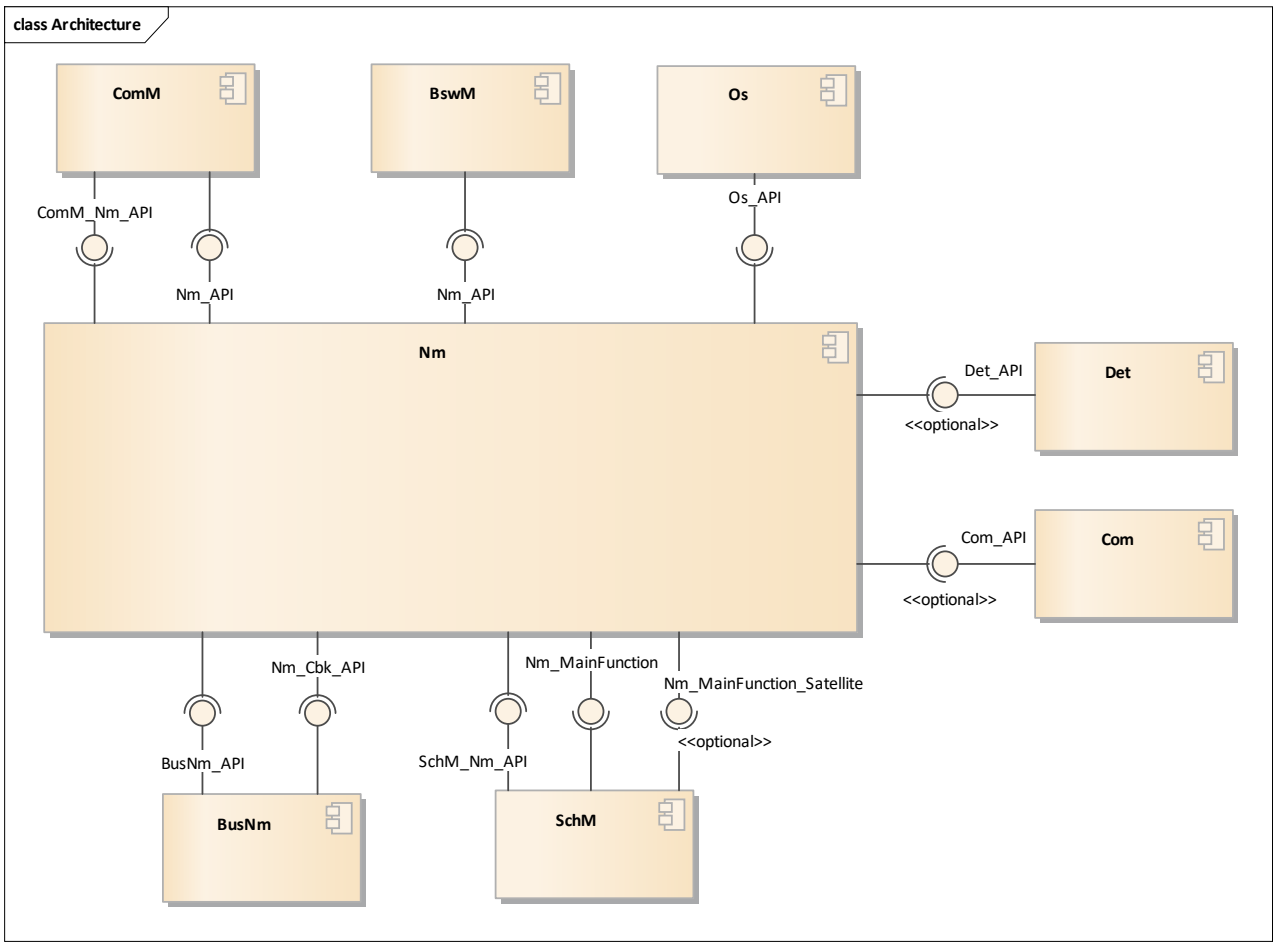

Figure 1-2 Interfaces to adjacent modules of the Nm

2 Functional Description

2.1 Features

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

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

> Table 2-1 Supported AUTOSAR standard conform features

> Table 2-2 Not supported AUTOSAR standard conform features

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

Table 2-3 Features provided beyond the AUTOSAR standard

The following features specified in [1] are supported:

Supported AUTOSAR Standard Conform Features
Basic Functionality
Support of Generic BusNm Modules
Coordinator Functionality
State Report
MICROSAR Classic Identity Manager using Post-Build Selectable

Table 2-1 Supported AUTOSAR standard conform features

2.1.1 Deviations Against AUTOSAR 4.0.3

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

Category	Description	ASR Version
-	-	4.0.3

Table 2-2 Not supported AUTOSAR standard conform features

2.1.1.1 Set Sleep Ready Bit

According to [1], Sleep Ready Bit will be set before waiting for Synchronization Points. In this implementation, Sleep Ready Bit will be set after a Synchronization Point appears.

2.1.1.2 Nm_NetworkStartIndication as trigger for Coordinated Shutdown Abortion

The function Nm_NetworkStartIndication is not a trigger for aborting a shutdown in the NM Coordinator algorithm despite the requirements in [1]. The NM deviates against this requirement, because it has been removed in newer versions of [1].

2.1.2 Additions/ Extensions

The following features are provided beyond the AUTOSAR standard:

Features Provided Beyond The AUTOSAR Standard
Additional DET Error Codes
Synchronization Timeout
Configurable Notification Functions
Macro Layer Optimization
Memory Initialization
Support for J1939Nm
Runtime Measurement Support
Automatic Calculation of Shutdown Delay Timer
Callback Nm_CoordReadyToSleepCancellation
Multiple BusNms on One Channel
Wake-up by Nm Coordinator
Passive Mode as Global Setting
BusNm Specific Pdu Rx Indication support
Multi-Partition support
Synchronized PNC Shutdown support

Table 2-3 Features provided beyond the AUTOSAR standard

2.1.2.1 Additional DET Error Codes

The following error code is reported additionally to the errors defined in [1]:

- NM_E_SYNCHRONIZATION_TIMEOUT: `Nm_SynchronizationPoint` was not called within the configured synchronization timeout time.
- NM_E_INVALID_STATE: An invalid/unexpected state transition has been passed to `Nm_StateChangeNotification` (only available if the optimization for only one BusNm on a channel is OFF) (e.g. transition from Normal Operation to Ready Sleep if all BusNms are currently in Bus Sleep).
- NM_E_SAME_STATES: The same states have been passed to `Nm_StateChangeNotification.`
- NM_E_FUNCTION_PTR_IS_NULL: The pointer that has been passed in order to call a function is equals to NULL. (E.g. BusNm function in the generated function table).

- NM_E_INVALID_PARTITION: An API service was called in the wrong partition context, e.g. `Nm_PostInit` was called from another partition context than the Nm master partition or `Nm_EnableCommunication` was called from a partition where no Nm instance is assigned to.
- NM_E_NO_PREINIT: `Nm_PreInit` was not called prior to `Nm_Init`.
- NM_E_BUSNM_E_NOT_OK: <Bus>Nm has returned E_NOT_OK when the Nm forwarded an API service on a satellite partition. (E.g. `Nm_EnableCommunication` needs to be forwarded on a satellite partition and the corresponding <Bus>Nm returns E_NOT_OK due to the call of `<Bus>Nm_EnableCommunication`).
- NM_E_NO_POSTINIT: API service of the Nm is used without prior module initialization of all Nm instances, i.e. `Nm_Init` needs to be called for every instance followed by `Nm_PostInit`.
- NM_E_ALREADY_INITIALIZED: Multiple initializations of a partition context, i.e. `Nm_Init` was already called for this individual Nm instance.
- NM_E_INVALID_GENDATA: Invalid generated data was detected.

2.1.2.2 Synchronization Timeout

If Nm Synchronizing Network is enabled, coordinator algorithm waits for a suitable point in time to continue shutdown of corresponding networks. If such a point is never reached, coordination algorithm will wait forever. To prevent this, a timeout for this point can be defined by setting the ‘Synchronization Timeout Time’.

2.1.2.3 Configurable Notification Functions

Some of the callback functions provided by the Nm may be needed by upper layers. Therefore the Nm was extended to provide a configurable notification interface where those callbacks can be configured to be forwarded to another module. Therefore a name has to be entered into the corresponding configuration parameter.

The following table shows which Nm callbacks can be forwarded and the corresponding configuration parameter where a function name has to be entered.

Nm Callback	Configuration Parameter
Nm_StateChangeNotification	State Change Indication Callback
Nm_RemoteSleepIndication	Remote Sleep Indication Callback
Nm_RemoteSleepCancellation	Remote Sleep Cancellation Callback
Nm_PduRxIndication	PDU Receive Indication Callback
Nm_RepeatMessageIndication	Repeat Message Indication Callback
Nm_TxTimeoutException	Transmission Timeout Error Callback
Nm_<Specific Standard BusNm>_PduRxIndication	Standard Bus Nm Pdu Rx Indication
Nm_<Specific Generic BusNm>_PduRxIndication	Generic Bus Nm Pdu Rx Indication

Table 2-4 Configurable Notification Function Mapping

Note that the prototypes for the forwarded functions must be provided by the module that wants to implement those notifications. Therefore header files containing the prototype definitions can be entered in the configuration.

The API prototype for these functions are described in chapter 4.6.1 ‘Notifications’.

2.1.2.4 Macro Layer Optimization

When having only one type of BusNm the Nm can be configured to be realized completely as a macro layer to save resources (ROM, RAM and run-time).

For further information refer to chapter 2.6 ‘Macro Layer Optimization’.

2.1.2.5 Memory Initialization

Not every start-up code of embedded targets and neither CANoe provide initialized RAM. It thus may happen that the state of a variable that needs initialized RAM may not be set to the expected initial value. Therefore an explicit initialization of such variables has to be provided at start-up by calling the additional function `Nm_InitMemory`.

For more information refer to chapter 2.8 ‘Initialization’.

2.1.2.6 Automatic Calculation of Shutdown Delay Timer

The shutdown delay timer is determined automatically. Therefore, the shutdown time of the corresponding BusNm is calculated and cannot be set by the user. The computation formula for each type is listed in the following table.

NM Type	Shutdown Time
CAN Nm	CanNmWaitBusSleepTime[1] + CanNmTimeoutTime[1]
FlexRay Nm	((FrNmReadySleepCnt[2] + 2) * FrNmRepetitionCycle[3] * FrCycleTime[4])
Udp Nm	UdpNmWaitBusSleepTime[1] + UdpNmTimeoutTime[1]
Lin Nm	0
Generic Nm	Value of GenericBusNmShutdownTime[1]

Table 2-5 BusNm Shutdown Time Calculation

If BusNm is a Generic Nm, Generic Bus Nm Shutdown Time is used. The shutdown time is subtracted from the Global Coordinator Time and the result is used as the shutdown delay timer.

2.1.2.7 Callback Nm_CoordReadyToSleepCancellation

When the NM Coordinator Sleep Ready Bit in the Control Bit Vector is set, the corresponding BusNm sets an indication and coordination algorithm assumes that the corresponding network is ready to sleep. But when the Sleep Ready Bit is not set anymore, and therefore the network is not ready to sleep anymore, there is no indication/cancellation according to [1]. For this reason, this callback is introduced.

---

[1] In the 'Wait Bus Sleep Extensions' feature is activated and in presence of NmOsek BusNm, the NM coordination algorithm permits two different shut-down times, depending on the NmOsek state (Normal or LimpHome), this times are calculated during run time. Refer to chapter 2.4.6 for more information.
In all cases the timing value given in s.
[2] Ready Sleep Counter is given in number of Repetition Cycles.
[3] Repetition Cycle is given in number of FlexRay Cycles
[4] FlexRay Cycle Time (duration of one FlexRay cycle) given in ms.

For further information refer to chapter 4.4.15 ‘Nm_CoordReadyToSleepCancellation’.

2.1.2.8 Passive Mode as Global Setting

The setting ‘Passive Mode Enabled’ can either be configured for each NM channel (note: the BusNm setting is globally) or globally for all channels.

The possibility to configure this setting for each channel exists due to the requirements in [1]; however newer versions of [1] have moved the ‘Passive Mode Enabled’ setting to a global configuration container so that this setting is applied for the whole ECU.

The NM module supports both possibilities, but the parameter may either only exist in every channel configuration container or exist in the global container.

2.1.2.9 BusNm Specific Pdu Rx Indication Support

The setting ‘Bus Nm Specific Pdu Rx Indication Enabled’ allows the generation of a BusNm specific callback that shall be called by the BusNm upon reception of the RxNmPdu. This function is called to notify the reception of a NmPdu in order to distinguish between each BusNm on the same channel (Multiple BusNms on a Channel, chapter 2.10) by using different identifiers for each BusNm.

Any function can be configured that shall be called on NM PDU reception.

Please note that this feature is relevant for rare cases.

Example

In a setup with one channel equipped with CanNm and NmOsek on the same channel, a distinction of which BusNm has received a NM PDU cannot be made by the NmPduReceiveIndCallback, since the provided Network Handle is the same.

Therefore, for a distinction, define the parameter ‘Standard Bus Nm Pdu Rx Indication’ to a certain value for CanNm and the ‘Generic Bus Nm Pdu Rx Indication’ for the NmOsek to yet another certain value.

Note

Use case not needed if, for example:

In a setup with two channels, one of them equipped with CanNm, the other one equipped with FrNm, there is no distinction required between the BusNms, because the Network Handle parameter is different for each channel.

Therefore it suffices to use ‘Pdu Rx Indication Enabled’ with ‘Pdu Receive Ind Callback’. (see also chapter 4.4.9 and 4.6.1.3).

Caution
'Bus Nm Specific Pdu Rx Indication Support' cannot be turned on if 'Standard Bus Type' is not equal to NM_BUSNM_CANNM. This functionality works only for CAN channels.

It is not necessary to configure the function if there is only one BusNm on the channel. The 'Pdu Receive Ind Callback' (See chapter 4.4.9) can be used as an alternative for this purpose.

2.1.2.9.1 Macro Layer interaction with BusNm Specific Pdu Rx Indication

It is possible to use 'Bus Nm Specific Pdu Rx Indication' with the Macro Layer optimization active.

The BusNm should call Nm_<BusNm>_PduRxIndication which is a macro definition in Nm_Cfg.h, therefore, at most one upper layer BusNm Specific Pdu Rx Indication is allowed if 'Macro Layer Enabled' is turned ON.

2.1.3 Limitations

2.1.3.1 Multiple BusNms on One Channel

There are several restrictions for multiple BusNms on one channel:

The BusNms on one channel are either coordinated completely actively or passively on one node.

Multiple BusNms on one channel can only be used on CAN channels.

The NM Coordinator needs to be activated if at least one channel exists that contains more than one BusNm. The channel can however be inside a coordination cluster that contains multiple channels or can form a coordination cluster on its own. This implies that, if the channel is coordinated actively by the node, the node is the last one that withdraws its communication request.

Furthermore, it is required that the BusNms either call the required functions for an active coordination (Nm_RemoteSleepIndication, Nm_RemoteSleepCancellation) / passive coordination (Nm_CoordReadyToSleepIndication, Nm_CoordReadyToSleepCancellation) or that the BusNm is a Channel Sleep Master, i.e. other nodes cannot keep the bus awake while the own node is ready to sleep.

Passive Mode can only have the same value for all BusNms on a channel.

The feature 'State Report Enabled' only reports the aggregated state (numerically highest state is considered as current state) of all BusNms on a channel to a Com signal.

If 'Synchronized Network' is enabled, it is only waited for one function call of Nm_SynchronizationPoint by one of the BusNms until the Shutdown Delay Timers are started.

Many service functions aggregate data retrieved from the BusNms in a manner that is not useful for the application (see also chapter 2.10.4).

2.2 Basic Functionality

The Nm module is a bus-independent adaptation layer between the ComM module and the bus-specific network management modules (e.g. CanNm, FrNm, LinNm, UdpNm or J1939Nm). Therefore it forwards function calls from the ComM module to the corresponding bus-specific network management and vice versa.

Further details can be found in [1].

The API description can be found in chapter 4 ‘API Description’.

2.3 Support of Generic BusNm Modules

Beside the bus-specific network management modules defined by AUTOSAR the Nm module is able to support further OEM-specific or legacy Nm modules if they fulfill the requirements for generic BusNm modules. This means that such modules have to provide the same APIs as the standard BusNm modules but with an own prefix. Also these modules have to take care about the Nm module configuration.

2.3.1 Creating a Generic BusNm or a Generic BusNm Wrapper

If a Generic BusNm needs to be created or a legacy Nm module needs to be wrapped by Generic BusNm interfaces, the following (not necessarily complete) list of aspects has to be considered:

- The Generic BusNm interfaces that are called by the Nm module have to be provided by the <GenericBusNmPrefix>.h. The <GenericBusNmPrefix> has to be configured in the configuration tool in the Nm module. This parameter determines the header file name as well as the prefix of the APIs called by the Nm module.
- At least the interface of Nm concerning the current mode (Nm_NetworkMode / Nm_BusSleepMode and optionally Nm_PrepareBusSleepMode has to be used).
- If the Generic BusNm sends or receives messages, the corresponding Bus Interface module (e.g. CanIf) has to be configured to forward the RxIndication / TxConfirmation towards the Generic BusNm and to have the possibility to use <BusIf>_Transmit.

The last aspect is not covered by this document, because this document describes only the aspects of the NM Interface module.

2.3.1.1 Providing the Interfaces that are called by the Nm module

The <GenericBusNmPrefix>.h needs to provide the following function declarations:

- <GenericBusNmPrefix>_PassiveStartUp
- <GenericBusNmPrefix>_NetworkRequest[5]
- <GenericBusNmPrefix>_NetworkRelease[5]
- <GenericBusNmPrefix>_EnableCommunication[6]
- <GenericBusNmPrefix>_DisableCommunication[6]
- <GenericBusNmPrefix>_SetUserData[7]

[5] Function is only required if ‘Passive Mode Enabled’ is OFF in the Nm channel/module configuration settings
[6] Function is only required if ‘Com Control Enabled’ is ON in the Nm module configuration settings

- <GenericBusNmPrefix>_GetUserData[8]
- <GenericBusNmPrefix>_GetPduData[9]
- <GenericBusNmPrefix>_RepeatMessageRequest[10]
- <GenericBusNmPrefix>_GetNodeIdentifier[11]
- <GenericBusNmPrefix>_GetLocalNodeIdentifier[11]
- <GenericBusNmPrefix>_CheckRemoteSleepIndication[12]
- <GenericBusNmPrefix>_GetState
- <GenericBusNmPrefix>_RequestBusSynchronization[13]
- <GenericBusNmPrefix>_SetSleepReadyBit[14]

It is recommended to copy the function prototypes from Nm.h and replace the compiler abstraction and memory mapping parts to the sections/keywords of NM to <GENERICBUSNMPREFIX>. Since there are no prototypes of Nm_RequestBusSynchronization and Nm_SetSleepReadyBit, an example header for the exemplary Generic Bus Nm ‘SpecialBusNm’ is provided for these function prototypes as follows. Note that Compiler_Cfg.h has to be extended by the SPECIALBUSNM_CODE keyword and MemMap.h has to be adapted by the SPECIALBUSNM_START_SEC_CODE / SPECIALBUSNM_STOP_SEC_CODE section begin/end parts.

[7] Function is only required if ‘User Data Enabled’ is ON, ‘Passive Mode Enabled’ is OFF, ‘Com User Data Support’ is OFF in the Nm module configuration settings
[8] Function is only required if ‘User Data Enabled’ is ON in the Nm module configuration settings
[9] Function is only required if ‘Node Id Enabled’ is ON or ‘User Data Enabled’ is ON in the Nm module configuration settings
[10] Function is only required if ‘Node Detection Enabled’ is ON in the Nm module configuration settings
[11] Function is only required if ‘Node Id Enabled’ is ON in the Nm module configuration settings
[12] Function is only required if ‘Remote Sleep Ind Enabled’ is ON in the Nm module configuration settings
[13] Function is only required if ‘Bus Synchronization Enabled’ is ON in the Nm module configuration settings
[14] Function is only required if ‘Passive Mode Enabled’ is OFF in the Nm channel/module configuration settings and if ‘Coordinator Sync Support’ is ON in the Nm module configuration settings

Example
If a Generic Bus Nm called ‘SpecialBusNm’ needs to be implemented, a header file called ‘SpecialBusNm.h’ also needs to be provided. Note that this header does not contain all prototypes, only those that cannot be derived from the prototypes of Nm.h.

```
#if !defined(SPECIALBUSNM_H)
#define SPECIALBUSNM_H

#include "Nm_Cfg.h"

#define SPECIALBUSNM_START_SEC_CODE
#include "MemMap.h"

/* Insert other function prototypes like
 * SpecialNm_PassiveStartUp here...
 */

#if (NM_BUS_SYNCHRONIZATION_ENABLED == STD_ON)
extern FUNC(Std_ReturnType, SPECIALBUSNM_CODE)
SpecialBusNm_RequestBusSynchronization
( CONST( NetworkHandleType, AUTOMATIC ) nmNetworkHandle );
#endif

#if (NM_PASSIVE_MODE_ENABLED == STD_OFF) && \
    (NM_COORDINATOR_SYNC_SUPPORT == STD_ON)
extern FUNC(Std_ReturnType, SPECIALBUSNM_CODE)
GenericBusNm_SetSleepReadyBit
(CONST(NetworkHandleType, AUTOMATIC) nmNetworkHandle,
CONST(boolean, AUTOMATIC) nmSleepReadyBit);
#endif

#define SPECIALBUSNM_STOP_SEC_CODE
#include "MemMap.h"

#endif
```

2.3.1.2 Implementing the functions called by Nm

As next step, implement these functions in a C file of the application that includes <GenericBusNm>.h. As a minimum requirement,

- <GenericBusNm>_PassiveStartUp
- <GenericBusNm>_NetworkRequest
- <GenericBusNm>_NetworkRelease

> <GenericBusNm>_GetState

should be implemented.

It is useful to use a variable of the Nm_StateType that holds the current state towards the NM Interface.

When the <GenericBusNm> is asleep, <GenericBusNm>_PassiveStartUp and/or <GenericBusNm>_NetworkRequest have been called, this should lead to a call of Nm_NetworkMode (not necessarily in the context of these function, may be delayed). When <GenericBusNm> is not asleep (i.e. Nm_NetworkMode has been called), Nm_BusSleepMode may only be called if there is no network request (i.e. <GenericBusNm>_NetworkRelease has been called after <GenericBusNm>_NetworkRequest or only <GenericBusNm>_PassiveStartUp led to the call of Nm_NetworkMode).

The usage of

> Nm_BusSleepMode

> Nm_NetworkMode

is mandatory for a Generic BusNm. The usage of the other callback functions (see chapter 4.4) is optional.

If a legacy module is wrapped, the <GenericBusNm>_PassiveStartUp, <GenericBusNm>_NetworkRequest, <GenericBusNm>_NetworkRelease functions (and eventually other <GenericBusNm> functions called by Nm) probably need to call a function of the legacy module. Vice versa, if the legacy module wants to notify a higher module, these callbacks need to be implemented by <GenericBusNm> and to be forwarded to the Nm callbacks (e.g. Nm_BusSleepMode, Nm_NetworkMode).

2.4 Coordinator Functionality

The coordinator functionality can be used to shut down two or more networks synchronously. The coordination algorithm keeps all networks of a coordinator awake as long as at least one network is not ready to sleep. When all networks are ready to sleep, synchronous shutdown will start.

The coordinator can also be used to coordinate the usage of multiple BusNms on one network.

2.4.1 Coordinated Networks

An ECU can have multiple, independent coordinators as long as every coordinator has at least two networks (or at least two BusNms one network). Not every network of an ECU must be part of a coordinator.

A network, which shall be part of a coordinator, must have a configured ‘Coordinator Cluster Index’. This index identifies the coordinator which is associated to the network.

Furthermore, a coordinated network can either be an active or a passive one. On an actively coordinated network, the current ECU is the last ECU which releases the Network. As long as any other ECU requests the network, the current ECU requests the network, too.

Caution
A network should only have one active coordinator!

If there were two or more active coordinators on one network, no ECU would enter sleep because the coordinators keep awake each other.

Additionally, an actively coordinated network is kept awake as long as any other network of the same coordinator is requested.

If a coordinated network is not an active one, it is automatically a passively coordinated network. Passively coordinated networks are kept awake when a local request exists or as long as at least one actively coordinated network of the same coordinator is requested.

2.4.2 Shutdown Algorithm

The coordinator algorithm checks the communication status of all networks belonging to the same coordinator. Communication is required as long as a local or a remote request is present. A local request means, that `Nm_NetworkRequest()` was called and `Nm_NetworkRelease()` was not called yet. For an actively coordinated network a remote request is assumed as long as no call of `Nm_RemoteSleepIndication()` indicates that network is ready to sleep. `Nm_RemoteSleepCancellation()` restores a remote request. `Nm_CoordReadyToSleepIndication()` and `Nm_CoordReady-ToSleepCancellation()` are the counterpart of passively coordinated networks.

When no communication request for actively coordinated networks is present, shutdown algorithm starts. Therefore, `BusNm_NetworkRelease()` will be called on passively coordinated networks if there is no local communication request present anymore. As soon as an actively coordinator on a remote ECU determines that no other ECU requests the network, it will locally initiate its shutdown algorithm. Hence, remaining ECUs do not have a remote communication request on its passively coordinated channels and can continue the shutdown procedure.

If one of the networks belonging to the coordinator is awake and is a synchronizing network, the shutdown algorithm waits for a suitable point in time to continue the shutdown procedure. This point is indicated by `Nm_SynchronizationPoint()`.

If no synchronizing network is available or the synchronization point has occurred, a network-specific delay time starts for each actively coordinated network. The timing is predetermined and depends on the Global Coordination Delay Time and the network-specific shutdown time (refer to chapter 2.1.2.6 ‘Automatic Calculation of Shutdown Delay Timer’ for more information). In case the network has NmOsek and all NmOsek members are in “Normal State” the shutdown delay time is calculated differently. (refer to chapter 2.4.6 ‘Wait Bus Sleep Extension’) Additionally, the Sleep Ready Bit is set by calling the corresponding `BusNm_SetSleepReadyBit()` function.

When a timer of a network expires, this network will be released by calling the corresponding `BusNm_RequestBusSynchronization()` and `BusNm_Network-Release()` function.

Finally, when all timers are expired and every network has notified Bus Sleep, the coordinator is shut down.

If the coordinator detects any need for communication during the shutdown procedure, the algorithm ensures that all not sleeping networks are restarted again. Additionally, on actively coordinated networks, the Sleep Ready Bit will be set back. For networks which are already asleep `ComM_Nm_RestartIndication()` will be called.

Caution
When a new request on a network occurs and coordinator is already shut down, neither a restart nor an indication will be invoked on networks belonging to the same coordinator.

2.4.3 State Machine of Coordinator

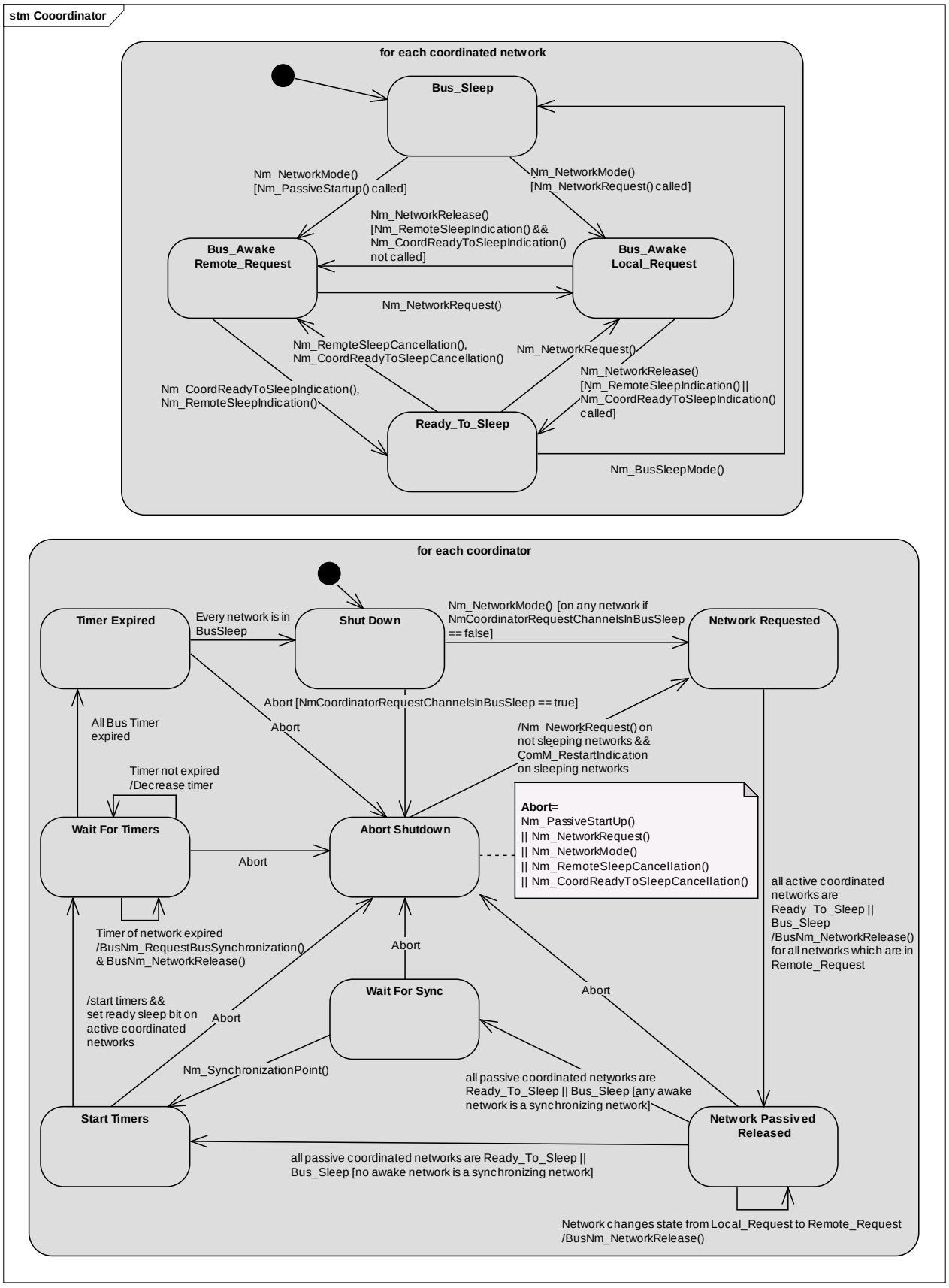

Figure 2-1 State Machine of Coordinator

2.4.4 Wake-up

The wake-up of networks is outside the scope of the Nm according to AUTOSAR, independent of coordinator support being enabled or not. Further details can be found in [1].

Note
ComM can be configured to automatically start all networks when one network is requested (Synchronous Wake-up).

However, the NM Coordinator can also be used to indicate to ComM via ComM_Nm_RestartIndication that an asleep network should be requested in the coordinator state ‘Shut Down’ (see Figure 2-1). This requires the configuration setting ‘Coordinator Request Channels In Bus Sleep’ to be turned ON.

Thus an alternative way of waking up networks synchronously can be realized: if one of the networks that is inside a coordinator cluster is woken up by bus or is requested by ComM while the NM Coordinator is in the ‘Shut Down’ state, all networks are woken up in the coordinator cluster. So this feature permits the NM coordinator to wake up only the channels in one NM coordinator cluster, not every communication channel like ‘Synchronous Wake-up’ of ComM does.

If the configuration setting ‘Coordinator Request Channels In Bus Sleep’ is not ON, this behavior is disabled.

2.4.5 Sleep Master

If a coordinated network is a Sleep Master, the current ECU can absolutely decide when to shut down this network. Therefore, the coordinator assumes that this bus is always ready to sleep when no local request exists and does not evaluate the remote sleep indication status.

2.4.6 Wait Bus Sleep Extensions

Within NM Coordination, the shutdown time of each BusNm is considered as a static time and therefore generated into constant arrays.

However, the feature 'Wait Bus Sleep Extensions' permits NmOsek to have two different shutdown times. Thus, the NM Coordination algorithm needs to decide during run-time which of the two shutdown times will be applied by NmOsek depending on the status of NmOsek (either Normal or LimpHome).

When at least one of the NmOsek BusNms is in 'LimpHome' status, the regular shut down time is used.

On the other hand, when the NmOsek BusNms are in 'Normal' status on every channel, a shorter shut down timer is used. The timer calculation is realized by Nm and it depends on the configured *NmGenericBusNmShutdownTime*.

2.4.6.1 CanNm and NmOsek on the same channel

CanNm behaves differently regarding its shutdown time if NmOsek is located on the same channel and if Wait Bus Sleep Extensions are turned ON [6]. This is also considered by the Nm code generator for the shutdown delay timer calculation.

2.5 State Report

The feature ‘State Report’ writes state change notifications about a network as bit-coded values in a configured Com signal for every network which has enabled the feature.

The following table provides all state changes (from previous state to current state) and the corresponding signal values that are set by the NM Interface.

Previous State	Current State	Signal Value
Bus Sleep Mode[15]	Repeat Message	1
Prepare Bus Sleep Mode[16]	Repeat Message	2
Repeat Message	Normal Operation	4
Ready Sleep	Normal Operation	8
Ready Sleep	Repeat Message	16
Normal Operation	Repeat Message	32

Table 2-6 Nm State Change Signal Values

2.6 Macro Layer Optimization

The Nm module implementation can be optionally configured to be only a macro layer if only one BusNm type is used. All function calls beside `Nm_GetVersionInfo()` are then realized as preprocessor defines that forward the calls directly to calls of the corresponding BusNm module. Also all callback functions from the BusNm module are redefined directly to callbacks to the upper layer (e.g. ComM).

2.7 Multi-Partition Handling

In case of Multi-Partition, the Nm module consists of multiple instances which are assigned to different partitions. These instances are divided into one master partition and one or multiple satellite partitions.

The master partition is defined as the partition where the Nm module is assigned to. This partition provides the main function and handles the data synchronization with the satellite partitions.

The satellite partitions are defined as the partitions where at least one Nm channel is assigned to. Each satellite partition provides its own satellite main function where the forwarding mechanism of services towards the corresponding BusNms and ComM is performed.

[15] As FlexRay NM does not perform a transition directly from Bus Sleep Mode to Repeat Message State the value is set in the transition from Synchronize Mode to Repeat Message State.
[16] This transition is not available for FlexRay NM.

2.8 Initialization

Before calling any other functionality of the Nm module the module has to be initialized. The initialization is realized by calling the intialization functions `Nm_PreInit`, `Nm_Init` and `Nm_PostInit` successively. The initialization call shall take place after initializing the BusNm modules and prior to the initialization of the ComM module.

For API details refer to chapters 4.2.1 ‘Nm_Init’, 4.2.18 ‘Nm_PreInit’ and 4.2.19 ‘Nm_PostInit’.

Note

The pre-initialization function `Nm_PreInit` must only be called once and must be called before `Nm_Init` is called on any partition.

In case of Multi-Partition, each instance of the Nm module must be initialized by calling `Nm_Init` on each partition where an Nm instance is located.

The post-initialization function `Nm_PostInit` must only be called after all Nm instances are initialized, i.e. `Nm_Init` was called on every Nm partition. The `Nm_PostInit` must be called on the Nm master partition.

The Nm module assumes that some variables are initialized with certain values at start-up, if the feature ‘Macro Layer Optimization’ is disabled and ‘Coordinator Support’ is enabled. As not all embedded targets support the initialization of RAM within the start-up code the Nm module provides the function `Nm_InitMemory`. This function has to be called during start-up and before `Nm_PreInit` is called. Refer also to chapter 2.1.2.5 ‘Memory Initialization’.

For API details refer to chapter 4.2.17 ‘Nm_InitMemory’.

2.9 Provision of the NM State

The NM State can be determined either by using the function `Nm_GetState()` (see also chapter 4.2.14) or by activating the feature ‘State Change Ind Enabled’. Both of these features are implemented according to [1].

Note that the NM state may also be optionally reported into Com signals using the ‘State Report Enabled’ feature (refer to chapter 2.5 for details).

2.9.1 Determining the NM State Using Nm_GetState

If `Nm_GetState()` (see also chapter 4.2.14) has been called, the current NM state and the current NM mode of the bus-specific Nm (e.g. CanNm, FrNm) associated with the system channel index `nmChannelHandle` are written to the passed pointer variables. Possible state and mode values that are returned into these variables can be seen in the definition of `Nm_StateType` / `Nm_ModeType` in NmStack_Types.h.

2.9.2 Using the ‘State Change Ind Enabled’ feature

The ‘State Change Ind Enabled’ feature enables a callback that is called each time the NM state has been changed. To use this feature, activate the ‘State Change Ind Enabled’ setting in configuration. Furthermore, enter the name of the function that shall be called in

case of a NM state change into the ‘State Change Ind Callback’ field and provide the file name of the header file that contains the function prototype of this function as one of the ‘Callbacks Prototype Header’ setting.

Note that the function prototype of the function given in ‘State Change Ind Callback’ has to be the same as (except the function name) `Nm_StateChangeNotification` (refer to chapter 4.4.10 for details).

2.10 Multiple BusNms on One Channel

The Nm module provides the possibility to support multiple BusNms on one channel (e.g. CanNm and J1939Nm, see Figure 2-2 Left). That means that there can be several NM algorithms running on one network and they can be coordinated by one node (see Figure 2-2 Right).

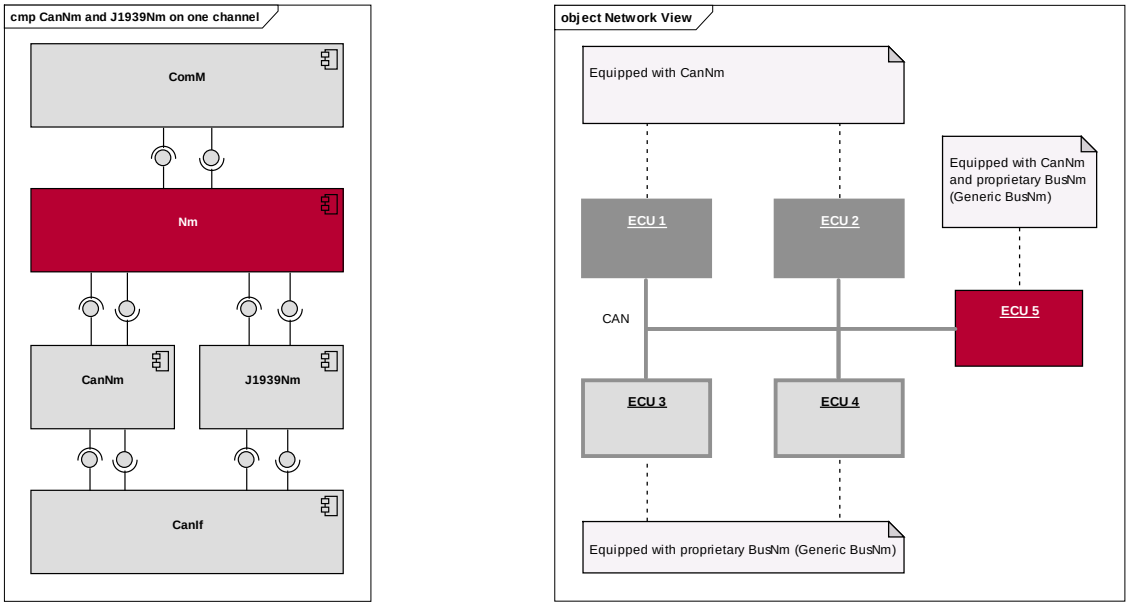

Figure 2-2 Left: Architectural View in AUTOSAR. Right: Network Topology with multiple BusNms on one network

In the bus topology example (Figure 2-2 Right), two ECUs are equipped with CanNm (ECU 1, ECU 2) and two other ECUs (ECU 3, ECU 4) are equipped with a proprietary BusNm. ECU 5 is equipped with both BusNms (CanNm and the proprietary one).

To realize the functionality of ECU 5, this feature can be used. It requires the NM Coordinator to be used.

There should be only one node that has multiple BusNms on the network (ECU 5 in this example). If there is more than one node, only one of the nodes that have multiple BusNms is allowed to be coordinated actively (thus the other nodes with multiple BusNms need to be coordinated passively, Figure 2-3). This use case is not recommended.

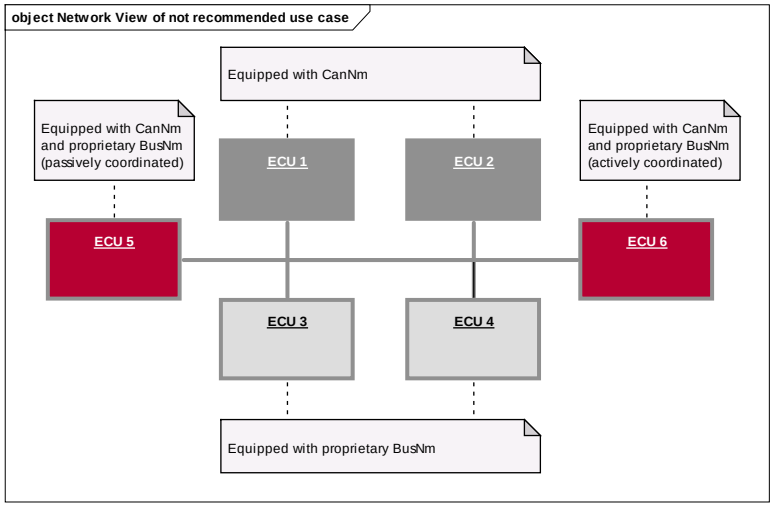

Figure 2-3 Not recommended use case having more than one node with multiple BusNms on the network

Even worse would be a setup with three different types of BusNms (e.g. CanNm, proprietary BusNm Type 1, and proprietary BusNm Type 2) without having one node that provides all three types of BusNms. Instead there could be two nodes having two BusNms of a certain type, for instance one having the two first types and the other having the last two types (Figure 2-4). Such setups are not supported!

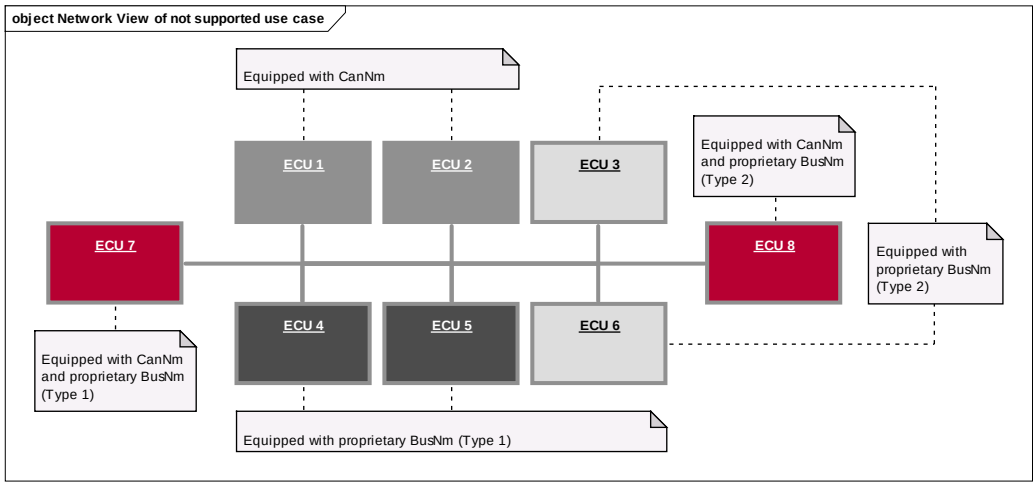

Figure 2-4 Unsupported use case having more than one node with multiple BusNms

Caution
Currently, multiple BusNms are only supported on CAN channels.

The following sections explain the basic principles how information is exchanged between the BusNms, Nm and the upper layers.

2.10.1 Notification of Mode Changes in the BusNms

The BusNms notify the Nm module about changes in their modes (Bus Sleep Mode, Prepare Bus Sleep Mode, Network Mode).

This mode is notified towards ComM. If multiple BusNms are configured on one channel, the modes need to be aggregated before ComM is notified. In other words, the ‘highest’ mode of all BusNms on a channel needs to be determined before ComM is notified about mode changes. The set of modes of all BusNms on a channel is an unordered one (the order does not matter to Nm).

The mode requests (Nm_PassiveStartUp, Nm_NetworkRequest, Nm_NetworkRelease) are propagated towards the BusNm by the NM Coordinator (refer to chapter 2.4 for details).

Figure 2-5 provides an example of two BusNms on one channel. The encoding of the sub-states ‘00’, ‘01’, ‘02’, ‘11’, ‘12’, ‘22’ represents the current modes of the underlying BusNms, where ‘0’ indicates ‘Bus Sleep Mode’, ‘1’ refers to ‘Prepare Bus Sleep Mode’ and ‘2’ stands for ‘Network Mode’.

Note that the super-states ‘Bus Sleep Mode’, ‘Prepare Bus Sleep Mode’ and ‘Bus Sleep Mode’ indicate the mode that is forwarded as an aggregated mode towards ComM.

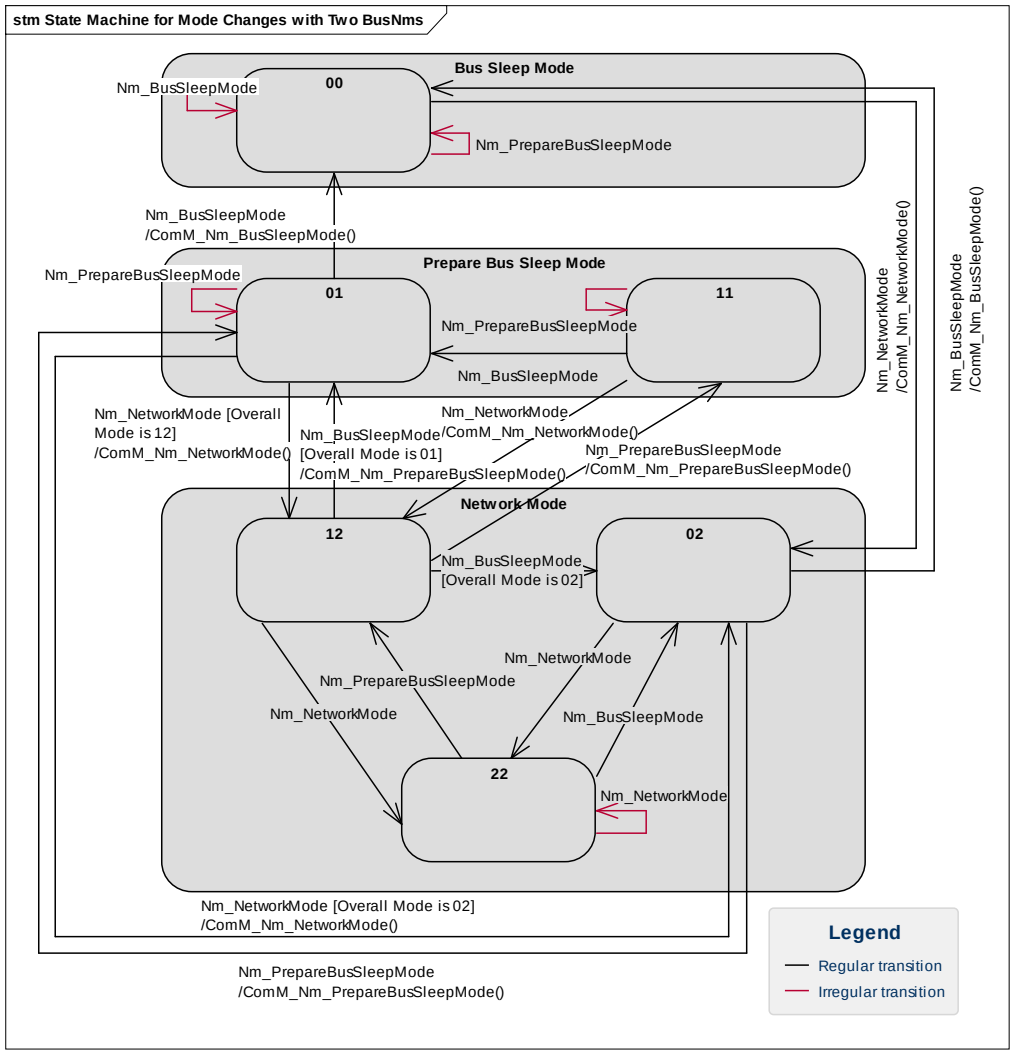

Figure 2-5 Mode Changes with two BusNms on one channel

The states ‘01’ of ‘Prepare Bus Sleep Mode’ and ’12’ of ‘Network Mode’ need to recalculate the Overall Mode of all BusNms for certain triggers (01: Nm_NetworkMode, 12: Nm_BusSleepMode). This is because these triggers are ambiguous inside these states (e.g. possible target states from ‘01’ for the trigger Nm_NetworkMode are ‘02’ and ‘12’, because either the one BusNm or the other may have changed to Network Mode).

In ambiguous cases, BusNm_GetState is called for each BusNm. This requires each BusNm to provide the correct new mode in the context of the Nm_BusSleepMode, Nm_PrepareBusSleepMode, Nm_NetworkMode calls by the BusNm.

Figure 2-5 also contains transitions annotated as ‘irregular transitions’. These are transitions that shall normally not occur because they are not intended by AUTOSAR. Example: Nm_PrepareBusSleepMode() shall not be called in Bus Sleep Mode. However, if it was called, it would not lead to any problems.

Caution
Take care that BusNm_GetState returns the correct new mode in the context of the Nm_BusSleepMode, Nm_PrepareBusSleepMode, Nm_NetworkMode calls by the BusNm when implementing a Generic BusNm.

Note
The ‘Synchronize Mode’ is not explicitly notified through a callback function and is therefore considered as ‘Bus Sleep Mode’ concerning the aggregation.

2.10.2 State Change Notifications

If the ‘State Change Ind Enabled’ feature is used (see chapter 2.9.2), the state change notifications are aggregated over all BusNms on a channel. Only the numerically highest state is forwarded to the ‘State Change Ind Callback’ function (e.g. implemented by BswM). In other words, the current overall state of *n* BusNms is $\max_{i=1,\ldots,n}\{state_i\}$.

Since the previous state is also provided by the call of Nm_StateChangeNotification, there are no ambiguities and errors in state changes can also be detected and reported to Det (see chapter 2.12.1).

An example for two BusNms on one channel is provided in Table 2-7.

Current State of BusNm 1 / Current State of BusNm 2	0: NM_STATE_UNINIT	1: NM_STATE_BUS_SLEEP	2: NM_STATE_PREPARE_BUS_SLEEP	3: NM_STATE_READY_SLEEP	4: NM_STATE_NORMAL_OPERATION	5: NM_STATE_REPEAT_MESSAGE	6: NM_STATE_SYNCHRONIZE	7: NM_STATE_OFFLINE	8: NM_STATE_CHECK_WAKEUP	9: NM_STATE_WAIT_STARTUP	10: NM_STATE_WAIT_NETWORK_GW_MSG_ACTIVE	11: NM_STATE_WAIT_NETWORK_GW_AND_EVENT_MSG_ACTIVE	12: NM_STATE_BUS_OFF
0: NM_STATE_UNINIT	0	1	2	3	4	5	6	7	8	9	10	11	12
1: NM_STATE_BUS_SLEEP	1	1	2	3	4	5	6	7	8	9	10	11	12
2: NM_STATE_PREPARE_BUS_SLEEP	2	2	2	3	4	5	6	7	8	9	10	11	12
3: NM_STATE_READY_SLEEP	3	3	3	3	4	5	6	7	8	9	10	11	12
4: NM_STATE_NORMAL_OPERATION	4	4	4	4	4	5	6	7	8	9	10	11	12
5: NM_STATE_REPEAT_MESSAGE	5	5	5	5	5	5	6	7	8	9	10	11	12
6: NM_STATE_SYNCHRONIZE	6	6	6	6	6	6	6	7	8	9	10	11	12
7: NM_STATE_OFFLINE	7	7	7	7	7	7	7	7	8	9	10	11	12
8: NM_STATE_CHECK_WAKEUP	8	8	8	8	8	8	8	8	8	9	10	11	12
9: NM_STATE_WAIT_STARTUP	9	9	9	9	9	9	9	9	9	9	10	11	12
10: NM_STATE_WAIT_NETWORK_GW_MSG_ACTIVE	10	10	10	10	10	10	10	10	10	10	10	11	12
11: NM_STATE_WAIT_NETWORK_GW_AND_EVENT_MSG_ACTIVE	11	11	11	11	11	11	11	11	11	11	11	11	12
12: NM_STATE_BUS_OFF	12	12	12	12	12	12	12	12	12	12	12	12	12

Table 2-7 Overall State of two BusNms on one channel

Note that the initial state of each BusNm is ‘Bus Sleep’ (1).

2.10.3 Remote Sleep Indication Statuses

The Remote Sleep status for actively coordinated channels and the Coordinator Ready to Sleep status for passively coordinated channels is aggregated over all BusNms on a channel.

The ‘Remote Sleep Ind Callback’ / ‘Remote Sleep Cancel Callback’ notifications that may be implemented by an upper layer are thus also augmented by an aggregation of the Remote Sleep Indication overall state of all BusNms.

The initial Remote Sleep state is ‘Number of BusNms that are Channel Sleep Masters’. Currently, J1939Nm is always considered as a Channel Sleep Master (see chapter2.4.5) and each Generic BusNm can be configured individually as Channel Sleep Master.

The ‘Remote Sleep Ind Callback’ function is called when all BusNms that are not ‘Channel Sleep Masters’ have indicated the readiness to sleep of the other nodes (call of Nm_RemoteSleepIndication on actively coordinated channels / Nm_CoordReadyToSleepIndication on passively coordinated channels).

If ‘Remote Sleep Ind Callback’ has already been called and one BusNm has detected that at least one remote node is not ready to sleep (call of Nm_RemoteSleepCancellation on actively coordinated channels / Nm_CoordReadyToSleepCancellation passively coordinated channels), the ‘Remote Sleep Cancel Callback’ function is called.

A call of Nm_NetworkMode by the first BusNm that enters Network Mode resets the Remote sleep to its initial value (‘Number of BusNms that are Channel Sleep Masters’).

An example state machine of the remote sleep callbacks is illustrated in Figure 2-6.

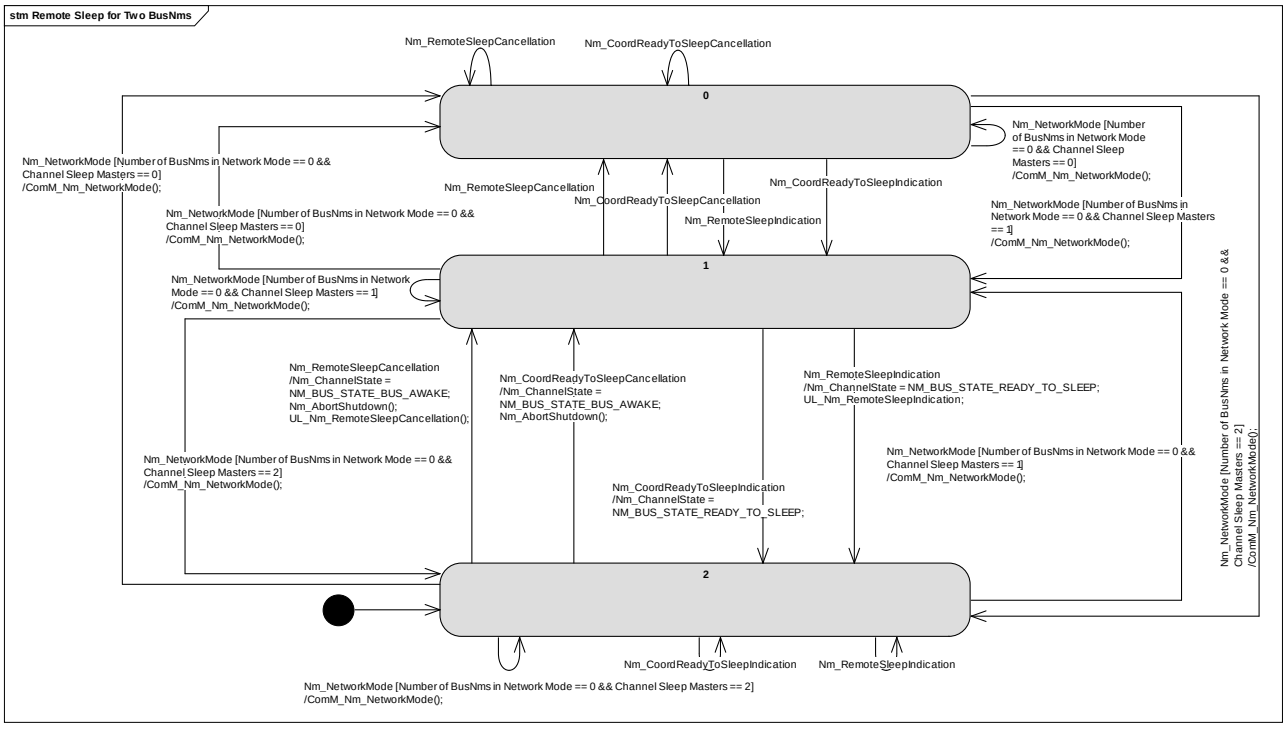

Figure 2-6 State Machine of Remote Sleep callbacks for two BusNms on one channel

As it can be seen, the total number of states is ‘Number of BusNms on the channel’ + 1.

2.10.4 Other Aggregated Information and Caveats

If the following service functions are used by the application, the data that is retrieved from the BusNms is aggregated overall BusNms. It is therefore recommended to call the corresponding BusNm function directly if multiple BusNms are used on one channel. Refer to the detailed service description chapters for details.

Nm_GetUserData (chapter 4.2.8)

Nm_GetPduData (chapter 4.2.9)

Nm_GetNodeIdentifier (chapter 4.2.11)

Nm_GetLocalNodeIdentifier (chapter 4.2.12)

Nm_CheckRemoteSleepIndication (chapter 4.2.13)

Nm_GetState (chapter 4.2.14)

The service function Nm_SetUserData (chapter 4.2.7) propagates the provided user data to all BusNms on the channel. Therefore the pointer nmUserDataPtr has to point to a buffer that is large enough to fit into the user data bytes of the BusNm that has greatest number of user data bytes. Even though E_NOT_OK may be returned from Nm_SetUserData, the user data has been accepted by the BusNms that support the service BusNm_SetUserData. No DET Error is thrown by Nm itself (but maybe thrown by the BusNm, depends on the BusNm).

Also consider the following caveats:

Caution

- The function Nm_PduRxIndication (and thus the user callback UL_Nm_PduRxIndication) can be called by any BusNm on the channel and it cannot be determined during run-time which BusNm has called the function. Refer to chapter 4.6.1.3 for details.
- The functions Nm_<BusNm>_PduRxIndication (and thus the configured user callbacks) can be called by any BusNm on the channel and they can determine which BusNm has called the function by using different identifiers for each BusNm.
- The function Nm_StateChangeNotification (and thus the callback UL_Nm_StateChangeNotification) can be called by any BusNm on the channel and the numerically highest state is reported as current state towards the UL_Nm_StateChangeNotification callback and the Com Signal if 'State Report Enabled' is used. Refer to chapter 4.6.1.5 for details.
- The function Nm_RepeatMessageIndication (and thus the callback UL_Nm_RepeatMessageIndication) can be called by any BusNm on the channel and thus it cannot be determined which of the BusNms has entered Repeat Message. The repeat message request is not forwarded to the other BusNms on the channel by the Nm. Refer to chapter 4.6.1.6 for details.
- The function Nm_TxTimeoutException and Nm_CarWakeUpIndication (and thus the callbacks UL_Nm_TxTimeoutException, UL_Nm_CarWakeUpIndication) can be called by any BusNm on the channel and thus it cannot be determined which of the BusNms has called it.
- The return values of the service functions are aggregated (i.e. if one BusNm call returns E_NOT_OK, E_NOT_OK is returned by the corresponding Nm function).
- The data behind the pointer(s) in Nm_GetNodeIdentifier, Nm_GetLocalNodeIdentifier might have been manipulated although E_NOT_OK is returned.

2.11 Main Functions

The Nm module implementation provides one master main function (main function running on the Nm master partition). When the NM Coordinator is enabled this main function has to be called cyclically on task level. The default cycle time is 10 milliseconds. The value has to be set in the component configuration.

For API details refer to chapter 4.2.16 'Nm_MainFunction'.

In case of Multi-Partition, the Nm module implementation additionally provides one satellite main function for each partition where the Nm is assigned to (the satellite main function does also exist for the Nm master partition). These main functions have to be called cyclically on task level. The cycle time of the satellite main functions is the same as for the Nm master main function.

Note
In case of Multi-Partition, the satellite main functions are only provided if one of the following features is turned ON:

- ‘Coordinator Support’
- ‘Node Detection’
- ‘Communication Control’
- ‘User Data’ is used in combination with `Nm_SetUserData()`.

2.12 Error Handling

2.12.1 Development Error Reporting

Reporting of development errors is done by the service

```
Std_ReturnType Det_ReportError (
      uint16 ModuleId, uint8 InstanceId,
      uint8 ApiId, uint8 ErrorId )
```
(4.3)

Please refer to the documentation of the development error tracer [2] for further information and a detailed description of the API.

The reported Nm ID is 29.

The reported service IDs identify the services which are described in 4.2. The following table presents the service IDs and the related services:

Service ID	Service
0x00	Nm_Init
0x01	Nm_PassiveStartUp
0x02	Nm_NetworkRequest
0x03	Nm_NetworkRelease
0x04	Nm_DisableCommunication
0x05	Nm_EnableCommunication
0x06	Nm_SetUserData
0x07	Nm_GetUserData
0x08	Nm_GetPduData
0x09	Nm_RepeatMessageRequest
0x0A	Nm_GetNodeIdentifier
0x0B	Nm_GetLocalNodeIdentifier
0x0D	Nm_CheckRemoteSleepIndication
0x0E	Nm_GetState
0x0F	Nm_GetVersionInfo
0x10	Nm_MainFunction
0x11	Nm_NetworkStartIndication

Service ID	Service
0x12	Nm_NetworkMode
0x13	Nm_PrepareBusSleepMode
0x14	Nm_BusSleepMode
0x15	Nm_PduRxIndication
0x16	Nm_StateChangeNotification
0x17	Nm_RemoteSleepIndication
0x18	Nm_RemoteSleepCancellation
0x19	Nm_SynchronizationPoint
0x1A	Nm_RepeatMessageIndication
0x1B	Nm_TxTimeoutException
0x1C	Nm_<BusNm>_PduRxIndication
0x1D	Nm_CarWakeUpIndication
0x1E	Nm_CoordReadyToSleepIndication
0x1F	Nm_CoordReadyToSleepCancellation
0x20	Nm_InitMemory
0x28	Nm_ForwardSynchronizedPncShutdown
0x24	Nm_RequestSynchronizedPncShutdown
0x30	Nm_PreInit
0x31	Nm_PostInit
0x80	Nm_MainFunction_Satellite

Table 2-8 Service IDs

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

Error Code		Description
0x00	NM_E_UNINIT	API service used without module initialization.
0x01	NM_E_HANDLE_UNDEF	API service used with wrong network handle.
0x02	NM_E_PARAM_POINTER	API service called with a NULL pointer
0x20	NM_E_SYNCHRONIZATION_TIMEOUT[17]	`Nm_SynchronizationPoint` was not called within the configured synchronization timeout time.
0x21	NM_E_FUNCTION_PTR_IS_NULL	Pointer to a function to be called is equals NULL
0x22	NM_E_INVALID_STATE[17]	An invalid state has been passed to `Nm_StateChangeNotification` (only available if the optimization for only one BusNm on a channel is OFF)
0x23	NM_E_SAME_STATES[17]	The same states have been passed to `Nm_StateChangeNotification`
0x24	NM_E_NOT_AVAILABLE_IN_PASSIVE_MODE	Nm Passive Mode is not enabled for this channel
0x25	NM_E_INVALID_PARTITION[17]	API service called from invalid partition context
0x26	NM_E_NO_PREINIT[17]	API Nm_PreInit was not called prior to Nm_Init
0x27	NM_E_BUSNM_E_NOT_OK[17]	<Bus>Nm has returned E_NOT_OK when Nm forwarded the API service
0x28	NM_E_NO_POSTINIT[17]	API service used without module initialization of all instances
0x29	NM_E_ALREADY_INITIALIZED[17]	API service used while the module is already initialized on the current instance
0x30	NM_E_INVALID_GENDATA	Invalid generated data was detected.

Table 2-9 Errors reported to DET

2.12.2 Production Code Error Reporting

The Nm module currently does not have any error which has to be reported to the DEM.

[17] This error code is an extension to AUTOSAR. Refer also to chapter 2.1.2.1 ‘Additional DET Error Codes’.

3 Integration

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

3.1 Scope of Delivery

The delivery of the Nm contains the files which are described in the chapters 3.1.1 and 3.1.2:

3.1.1 Static Files

File Name	Source Code Delivery	Object Code Delivery	Description
Nm.c	■		This is the source file of the Nm.
Nm.lib		■	This is the library file built from the source file
Nm.h	■	■	This is the header file of the Nm.
Nm_Cbk.h	■	■	This is the callback header file of the Nm.
NmStack_Types.h	■	■	This is the Nm type definition header file of the Nm.

Table 3-1 Static files

3.1.2 Dynamic Files

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

File Name	Description
Nm_Cfg.h	This is the configuration header file.
Nm_Cfg.c	This is the configuration source file containing all pre-compile relevant content.
Nm_Lcfg.c	This is the configuration source file containing all link-time relevant content.
Nm_MemMap.h	This is the header file containing the Nm specific macros for memory mapping.

Table 3-2 Generated files

In case of Multi-Partition additional partition specific files are generated.

File Name	Description
Nm_Cfg_<OsApplication>.h	This is the configuration header file for the respective OsApplication.
Nm_Cfg_<OsApplication>.c	This is the configuration source file containing all pre-compile relevant content for the respective OsApplication.

Table 3-3 Generated Multi-Partition files

3.2 Include Structure

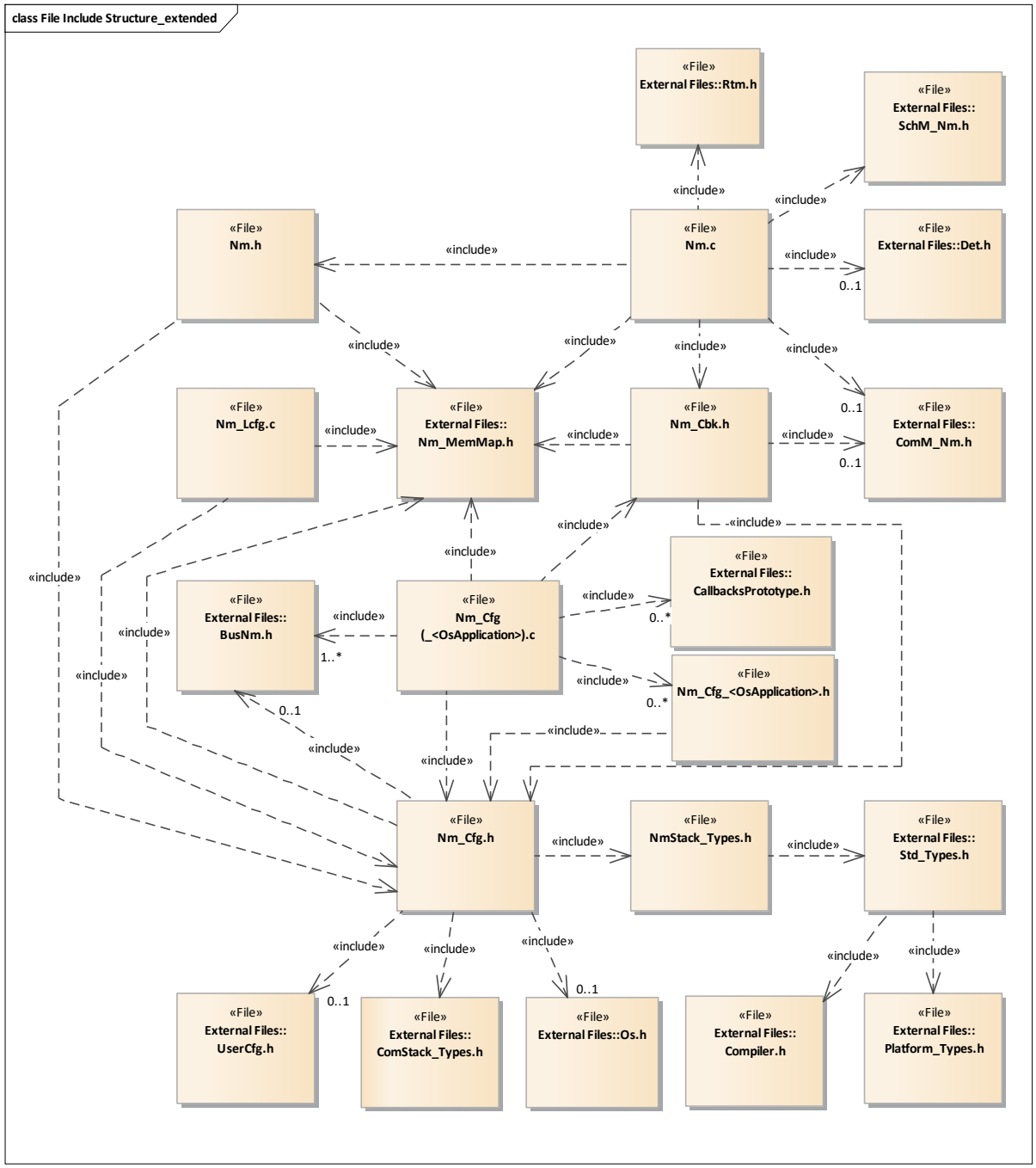

Figure 3-1 Include structure

Some includes are optional and depend on the configuration. `BusNm.h` stands for every used BusNm module and if multiple BusNm modules are used for each one the corresponding header is included. For example if CanNm and FrNm are used, the header files `CanNm.h` and `FrNm.h` are included. The files are included either in `Nm_Cfg.h` or `Nm_Cfg.c` depending on the configuration settings for ‘Macro Layer Optimization’. `CallbacksPrototype.h` and `UserCfg.h` stand for the respectively configured header files.

3.3 Critical Sections

The AUTOSAR standard provides with the BSW Scheduler (SchM) a BSW module, which handles entering and leaving critical sections.

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

```
void SchM_Enter_Nm_NM_EXCLUSIVE_AREA_i()                    ()
```

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

```
void SchM_Exit_Nm_NM_EXCLUSIVE_AREA_i()                     ()
```

The critical sections have to be defined and mapped to corresponding interrupt locks by the BSW Scheduler. Details which section needs what kind of interrupt lock are provided in the following section. For more information about the BSW Scheduler please refer to [5].

3.3.1 Exclusive Area 0

Interrupt Lock
No interruption by any interrupt is allowed. Therefore this section must always lock global interrupts.
Interfaces
> SchM_Enter_Nm_NM_EXCLUSIVE_AREA_0 > SchM_Exit_Nm_NM_EXCLUSIVE_AREA_0
Purpose
Ensures data consistency between BusNm and coordination algorithm.
Particularities and Limitations
This critical section is only relevant if the Nm coordinator is used.

Table 3-4 Exclusive Area 0

3.3.2 Exclusive Area 1

Interrupt Lock
No interruption by an interrupt is allowed if one of the following functions is executed in the context of an interrupt service routine: > Nm_NetworkMode > Nm_BusSleepMode > Nm_PrepareBusSleepMode > Nm_RemoteSleepIndication > Nm_RemoteSleepCancellation > Nm_CoordReadyToSleepIndication > Nm_CoordReadyToSleepCancellation If at least one of the above mentioned functions is executed in interrupt context, this section must always lock global interrupts.
Interfaces
> SchM_Enter_Nm_NM_EXCLUSIVE_AREA_1 > SchM_Exit_Nm_NM_EXCLUSIVE_AREA_1
Purpose
Ensures data consistency between BusNm and coordination algorithm.
Particularities and Limitations
This critical section is only relevant if the Nm coordinator is used and at least one channel contains more than one BusNm (e.g. CanNm and J1939Nm on one channel).

Table 3-5 Exclusive Area 1

3.3.3 Exclusive Area 2

Interrupt Lock
This exclusive area is only used in case of Multi-Partition. No interruption allowed by one of the following functions: > Nm_NetworkRequest > Nm_PassiveStartUp > Nm_NetworkMode > Nm_RemoteSleepCancellation > Nm_CoordReadyToSleepCancellation
Interfaces
> `SchM_Enter_Nm_NM_EXCLUSIVE_AREA_2` > `SchM_Exit_Nm_NM_EXCLUSIVE_AREA_2`
Purpose
Ensures data consistency of the coordinator state during data synchronization from satellite partitions.
Particularities and Limitations
This critical section is only relevant if Multi-Partition is configured, the Nm coordinator is used and at least one channel is a synchronizing network.

3.3.4 Exclusive Area 3

Interrupt Lock
This exclusive area is only used in case of Multi-Partition. It must be configured as a spin lock to protect the consistency of synchronized data
Interfaces
> `SchM_Enter_Nm_NM_EXCLUSIVE_AREA_3` > `SchM_Exit_Nm_NM_EXCLUSIVE_AREA_3`
Purpose
Ensures data consistency of the channel requested state during data synchronization from satellite partitions.
Particularities and Limitations
This critical section is only relevant if Multi-Partition is configured and the Nm coordinator is used.

3.4 Multi-Partition

After initialization, it has to be ensured that the Nm_ConfigPtr is protected agains memory write accesses from partitions with a lower ASIL than the Nm master partition. The Nm_ConfigPtr is mapped to the memory section NM_START_SEC_VAR_INIT_UNSPECIFIED.

4 API Description

For an interfaces overview please see .

4.1 Type Definitions

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

Type Name	C-Type	Description	Value Range
Nm_StateType	uint8	States of the bus-specific network management state machine. Not all states will be reached by each BusNm. For details refer to the Technical Reference or the AUTOSAR SWS of the corresponding BusNm.	NM_STATE_UNINIT No initialization has been performed.
			NM_STATE_BUS_SLEEP Nm entered sleep state due to initialization or shutdown.
			NM_STATE_PREPARE_BUS_SLEEP Nm prepares for entering sleep. This state is only relevant for BusNm modules on CAN, e.g. CanNm.
			NM_STATE_READY_SLEEP Communication is not needed any more by the application and no NM messages are transmitted.
			NM_STATE_NORMAL_OPERATION Communication is needed by the application and the NM message is transmitted
			NM_STATE_REPEAT_MESSAGE Nm has (re-)started and communication is enabled. Nm stays a configurable amount of time in this state and transmits its Nm message.
			NM_STATE_SYNCHRONIZE Start-up has been requested and Nm waits to be synchronized to the Repetition Cycle. This state is only relevant for FrNm.
			NM_STATE_OFFLINE Address Claiming is running or Address Loss has occurred. This state is only relevant for J1939Nm.
			NM_STATE_CHECK_WAKEUP State that is entered on external bus wake-up event. This state is only relevant for NmStMgr.
			NM_STATE_WAIT_STARTUP State that is entered on internal network request on NmStMgr

Type Name	C-Type	Description	Value Range
			channels. Nm is starting up and sends a wake-up message. No other messages can be transmitted in this state on NmOsek channels. This state is only relevant for NmStMgr and NmOsek.
			NM_STATE_WAIT_NETWORK_GW_MSG_ACTIVE Nm is starting up and was in state NM_STATE_WAIT_STARTUP before. The transmission of gateway messages is enabled in this state. This state is only relevant for NmOsek.
			NM_STATE_WAIT_NETWORK_GW_AND_EVENT_MSG_ACTIVE Nm is starting up and was in state NM_STATE_WAIT_NETWORK_GW_MSG_ACTIVE before. Gateway as well as event triggered messages can be transmitted in this state. This state is only relevant for NmOsek.
			NM_STATE_BUS_OFF This state is entered upon a BusOff notification. This state is only relevant for NmOsek.
Nm_ModeType	uint8	Modes of the bus-specific network management state machine. Not all modes will be reached by each BusNm. For details refer to the Technical Reference or the AUTOSAR SWS of the corresponding BusNm.	NM_MODE_BUS_SLEEP Nm entered sleep mode due to initialization or shutdown.
			NM_MODE_PREPARE_BUS_SLEEP Nm prepares for entering sleep. This mode is only relevant for BusNm modules on CAN, e.g. CanNm.
			NM_MODE_SYNCHRONIZE Start-up has been requested and Nm waits to be synchronized to the Repetition Cycle. This mode is only relevant for FrNm.
			NM_MODE_NETWORK Nm has (re-)started and communication is (partly) enabled.

Table 4-1 Type definitions

4.2 Services Provided by Nm

4.2.1 Nm_Init

Prototype	
`void Nm_Init ( const Nm_ConfigType * const nmConfigPtr )`	
Parameter	
`nmConfigPtr`	Configuration structure for initializing the module
Return code	
–	
Functional Description	
This function initializes the individual Nm instances, i.e. the Nm on the current partition context.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant. > This function has to be called for all individual Nm instances, i.e. for all partitions the Nm is assigned to. > This function has to be called after the pre-initialization of the Nm. > This function has to be called after the initialization of the respective bus interface. > This API is realized as a macro if ‘Macro Layer Optimization’ is enabled.	
Expected Caller Context	
> This function can be called from task level only.	

Table 4-2 Nm_Init

4.2.2 Nm_PassiveStartUp

Prototype	
`Std_ReturnType` **`Nm_PassiveStartUp`** `( const NetworkHandleType nmNetworkHandle )`	
Parameter	
`nmNetworkHandle`	Identification of the network
Return code	
`E_OK`	No error
`E_NOT_OK`	Passive start of network management has failed
Functional Description	
This function requests a passive start-up of the network management. The Nm calls therefore the passive start-up function of the respective BusNm (see also chapter 4.3 ‘Services Used by Nm’). **Note** When Nm_PassiveStartUp is called for a coordinated network the network request function of the respective BusNm(s) on the network is called instead of the passive start-up function.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant for the same network handle, reentrant otherwise.	
Expected Caller Context	
> This function can be called from task and interrupt level.	

Table 4-3 Nm_PassiveStartUp

4.2.3 Nm_NetworkRequest

Prototype	
`Std_ReturnType` **`Nm_NetworkRequest`** `( const NetworkHandleType nmNetworkHandle )`	
Parameter	
`nmNetworkHandle`	Identification of the network
Return code	
`E_OK`	No error
`E_NOT_OK`	Requesting the network has failed
Functional Description	
This function requests the network and the bus communication. The Nm calls therefore the network request function of the respective BusNm(s) on the network (see also chapter 4.3 ‘Services Used by Nm’).	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant for the same network handle, reentrant otherwise. > This function is only available if at least one network is not passive or CONFIG-VARIANT is LINK-TIME.	
Expected Caller Context	
> This function can be called from task and interrupt level.	

Table 4-4 Nm_NetworkRequest

4.2.4 Nm_NetworkRelease

Prototype	
`Std_ReturnType` **`Nm_NetworkRelease`** `( const NetworkHandleType nmNetworkHandle )`	
Parameter	
`nmNetworkHandle`	Identification of the network
Return code	
`E_OK`	No error
`E_NOT_OK`	Releasing the network has failed
Functional Description	
This function releases the network and the bus communication. The Nm calls therefore the network release function of the respective BusNm (see also chapter 4.3 ‘Services Used by Nm’). **Note** When Nm_NetworkRelease is called for a coordinated network, the network release function of the respective BusNm(s) is/are not called immediately. Instead, the network release function(s) is/are called when every network of the corresponding coordinator is ready to sleep.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous for not coordinated networks. > This function is asynchronous for coordinated networks. > This function is non-reentrant for the same network handle, reentrant otherwise. > This function is only available, if at least one network is not passive or CONFIG-VARIANT is LINK-TIME.	
Expected Caller Context	
> This function can be called from task and interrupt level.	

Table 4-5 Nm_NetworkRelease

4.2.5 Nm_DisableCommunication

Prototype	
`Std_ReturnType` **`Nm_DisableCommunication`** `(const NetworkHandleType nmNetworkHandle)`	
Parameter	
`nmNetworkHandle`	Identification of the network
Return code	
`E_OK`	No error
`E_NOT_OK`	Disabling the communication has failed
Functional Description	
This function disables the NM PDU transmission ability. The Nm calls therefore the disable communication function of the respective BusNm(s) on the network (see also chapter 4.3 ‘Services Used by Nm’).	
Particularities and Limitations	
> Service ID: see table 'Service IDs'	
> This function is synchronous.	
> This function is non-reentrant for the same network handle, reentrant otherwise.	
> This function is only available if ‘Com Control Enabled’ is enabled.	
Expected Caller Context	
> This function can be called from task and interrupt level.	

Table 4-6 Nm_DisableCommunication

4.2.6 Nm_EnableCommunication

Prototype	
`Std_ReturnType` **`Nm_EnableCommunication`** `(const NetworkHandleType nmNetworkHandle)`	
Parameter	
`nmNetworkHandle`	Identification of the network
Return code	
`E_OK`	No error
`E_NOT_OK`	Enabling the communication has failed
Functional Description	
This function enables the NM PDU transmission ability. The Nm calls therefore the enable communication function of the respective BusNm(s) on the network (see also chapter 4.3 ‘Services Used by Nm’).	
Particularities and Limitations	
> Service ID: see table 'Service IDs'	
> This function is synchronous.	
> This function is non-reentrant for the same network handle, reentrant otherwise.	
> This function is only available if ‘Com Control Enabled’ is enabled.	
Expected Caller Context	
> This function can be called from task and interrupt level.	

Table 4-7 Nm_EnableCommunication

4.2.7 Nm_SetUserData

Prototype	
`Std_ReturnType Nm_SetUserData ( const NetworkHandleType nmNetworkHandle, const uint8 * const nmUserDataPtr )`	
Parameter	
`nmNetworkHandle`	Identification of the network
`nmUserDataPtr`	Pointer to the user data that shall be transmitted in the next NM messages
Return code	
`E_OK`	No error
`E_NOT_OK`	Setting of user data has failed
Functional Description	
This function sets the user data that shall be transmitted within the next NM messages. The Nm calls therefore the set user data function of the respective BusNm(s) (see also chapter 4.3 ‘Services Used by Nm’).	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant for the same network handle, reentrant otherwise. > This function is only available if ‘User Data Enabled’ is enabled, ‘Com User Data Support’ is disabled and at least one network is not passive or CONFIG-VARIANT is LINK-TIME. > If multiple BusNms are configured on the channel, the Set User Data API will be called for all BusNms on the channel. This implies that the buffer behind the nmUserDataPtr has to provide enough data bytes so that all BusNms copy valid user data bytes. If the user data bytes shall be different for each BusNm, call the BusNm_SetUserData function directly instead.	
Caution In case of Multi-Partition it has to be ensured that the Set User Data API is called in this partition context where the `nmNetworkHandle`, which is passed to the API, is assigned to.	
Expected Caller Context	
> This function can be called from task and interrupt level.	

Table 4-8 Nm_SetUserData

4.2.8 Nm_GetUserData

Prototype	
`Std_ReturnType Nm_GetUserData ( const NetworkHandleType nmNetworkHandle, uint8 * const nmUserDataPtr )`	
Parameter	
`nmNetworkHandle`	Identification of the network
`nmUserDataPtr`	Pointer where the user data of the last received NM message shall be copied to
Return code	
`E_OK`	No error
`E_NOT_OK`	Getting of user data has failed
Functional Description	
This function copies the user data of the last received NM message to the location provided by the pointer. The Nm calls therefore the get user data function of the respective BusNm(s) on the network (see also chapter 4.3 ‘Services Used by Nm’).	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > This function is only available if ‘User Data Enabled’ is enabled. > If multiple BusNms are configured on the channel, the Get User Data API will be called for all BusNms on the channel. This implies that the buffer will contain the most recent user data bytes of one of the BusNms that is configured on the channel and that has implemented the service. It is recommended to call each BusNm_GetUserData function directly for channels with multiple BusNms, otherwise (one BusNm is configured on the channel) use Nm_GetUserData.	
Expected Caller Context	
> This function can be called from task and interrupt level.	

Table 4-9 Nm_GetUserData

4.2.9 Nm_GetPduData

Prototype	
`Std_ReturnType Nm_GetPduData ( const NetworkHandleType nmNetworkHandle, uint8 * const nmPduDataPtr )`	
Parameter	
`nmNetworkHandle`	Identification of the network
`nmUserDataPtr`	Pointer where the PDU data of the last received NM message shall be copied to
Return code	
`E_OK`	No error
`E_NOT_OK`	Getting of PDU data has failed
Functional Description	
This function copies the complete PDU data (system bytes and user data) of the last received NM message to the location provided by the pointer. The Nm calls therefore the get PDU data function of the respective BusNm(s) on the network (see also chapter 4.3 ‘Services Used by Nm’).	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > This function is only available if ‘User Data Enabled’ is enabled or ‘Node Id Enabled’ is enabled. > If multiple BusNms are configured on the channel, the Get Pdu Data API will be called for all BusNms on the channel. This implies that the buffer will contain the most recent PDU data bytes of one of the BusNms that is configured on the channel and that has implemented the service. It is recommended to call each BusNm_GetPduData function directly for channels with multiple BusNms, otherwise (one BusNm is configured on the channel) use Nm_GetPduData.	
Expected Caller Context	
> This function can be called from task and interrupt level.	

Table 4-10 Nm_GetPduData

4.2.10 Nm_RepeatMessageRequest

Prototype	
`Std_ReturnType` **`Nm_RepeatMessageRequest`** `(const NetworkHandleType nmNetworkHandle)`	
Parameter	
`nmNetworkHandle`	Identification of the network
Return code	
`E_OK`	No error
`E_NOT_OK`	Repeat message request has failed
Functional Description	
This function sets the repeat message request bit for the next NM message transmitted on the bus. This will force all NM nodes on the bus (including itself) to enter state ‘Repeat Message’ again and transmit NM messages. The Nm calls therefore the repeat message request function of the respective BusNm(s) on the network (see also chapter 4.3 ‘Services Used by Nm’).	
Particularities and Limitations	
> Service ID: see table 'Service IDs'	
> This function is synchronous.	
> This function is non-reentrant for the same network handle, reentrant otherwise.	
> This function is only available if ‘Node Detection Enabled’ is enabled.	
Expected Caller Context	
> This function can be called from task and interrupt level.	

Table 4-11 Nm_RepeatMessageRequest

4.2.11 Nm_GetNodeIdentifier

Prototype	
`Std_ReturnType Nm_GetNodeIdentifier ( const NetworkHandleType nmNetworkHandle, uint8 * const nmNodeIdPtr )`	
Parameter	
`nmNetworkHandle`	Identification of the network
`nmNodeIdPtr`	Pointer where the node identifier of the last received NM message shall be copied to
Return code	
`E_OK`	No error
`E_NOT_OK`	Getting of node identifier has failed
Functional Description	
This function copies the node identifier of the last received NM message to the location provided by the pointer. The Nm calls therefore the get node identifier function of the respective BusNm(s) on the network (see also chapter 4.3 ‘Services Used by Nm’).	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > This function is only available if ‘Node Id Enabled’ is enabled. > If multiple BusNms are configured on the channel, the Get Node Identifier API will be called for all BusNms on the channel. This implies that the buffer will contain the most recent node identifier of one of the BusNms that is configured on the channel and that has implemented the service. If one of the BusNm_GetNodeIdentifier calls returns E_NOT_OK, the buffer behind the nmNodeIdPtr may still have been manipulated due to the call of Nm_GetNodeIdentifier. It is recommended to call each BusNm_GetNodeIdentifier function directly for channels with multiple BusNms, otherwise (one BusNm is configured on the channel) use Nm_GetNodeIdentifier.	
Expected Caller Context	
> This function can be called from task and interrupt level.	

Table 4-12 Nm_GetNodeIdentifier

4.2.12 Nm_GetLocalNodeIdentifier

Prototype
`Std_ReturnType Nm_GetLocalNodeIdentifier ( const NetworkHandleType nmNetworkHandle, uint8 * const nmNodeIdPtr )`
Parameter
`nmNetworkHandle` — Identification of the network
`nmNodeIdPtr` — Pointer where the node identifier of the local node shall be copied to
Return code
`E_OK` — No error
`E_NOT_OK` — Getting of local node identifier has failed
Functional Description
This function copies the node identifier of the local node to the location provided by the pointer. The Nm calls therefore the get local node identifier function of the respective BusNm(s) on the network (see also chapter 4.3 ‘Services Used by Nm’).
Particularities and Limitations
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > This function is only available if ‘Node Id Enabled’ is enabled. > If multiple BusNms are configured on the channel, the Get Local Node Identifier API will be called for all BusNms on the channel. This implies that the buffer will contain the local node identifier of one of the BusNms that is configured on the channel and that has implemented the service. If one of the BusNm_GetLocalNodeIdentifier calls returns E_NOT_OK, the buffer behind the nmNodeIdPtr may still have been manipulated due to the call of Nm_GetLocalNodeIdentifier. It is recommended to call each BusNm_GetLocalNodeIdentifier function directly for channels with multiple BusNms, otherwise (one BusNm is configured on the channel) use Nm_GetLocalNodeIdentifier.
Expected Caller Context
> This function can be called from task and interrupt level.

Table 4-13 Nm_GetLocalNodeIdentifier

4.2.13 Nm_CheckRemoteSleepIndication

Prototype
`Std_ReturnType` **`Nm_CheckRemoteSleepIndication`** `( const NetworkHandleType nmNetworkHandle, boolean * const nmRemoteSleepIndPtr )`
Parameter
`nmNetworkHandle` — Identification of the network
`nmRemoteSleepIndPtr` — Pointer where the remote sleep status shall be copied to
Return code
`E_OK` — No error
`E_NOT_OK` — Checking of remote sleep status has failed
Functional Description
This function copies the remote sleep status to the location provided by the pointer. The Nm calls therefore the check remote sleep indication function of the respective BusNm(s) on the network (see also chapter 4.3 ‘Services Used by Nm’).
Particularities and Limitations
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > This function is only available if ‘Remote Sleep Ind Enabled’ is enabled. > If multiple BusNms are configured on the channel, the Check Remote Sleep Indication API will be called for all BusNms on the channel. This implies that the buffer will contain the overall remote sleep statuses of all BusNms on the channel. That means that if one of the Remote Sleep Indication statuses is false, the result will be false. Also, if one of the BusNm_CheckRemoteSleepIndication returns E_NOT_OK, the result will not be returned. If one is interested the Remote Sleep Indication status of a particular BusNm on the channel, it is recommended to call BusNm_CheckRemoteSleepIndication directly for channels with multiple BusNms.
Expected Caller Context
> This function can be called from task and interrupt level.

Table 4-14 Nm_CheckRemoteSleepIndication

4.2.14 Nm_GetState

Prototype	
`Std_ReturnType Nm_GetState ( const NetworkHandleType nmNetworkHandle, Nm_StateType * const nmStatePtr, Nm_ModeType * const nmModePtr )`	
Parameter	
`nmNetworkHandle`	Identification of the network
`nmStatePtr`	Pointer where the current network management state shall be copied to
`nmModePtr`	Pointer where the current network management mode shall be copied to
Return code	
`E_OK`	No error
`E_NOT_OK`	Checking of remote sleep status has failed
Functional Description	
This function copies the NM state and the NM mode to the location provided by the pointers. The Nm calls therefore the get state function of the respective BusNm(s) on the network (see also chapter 4.3 ‘Services Used by Nm’).	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > If multiple BusNms are configured on the channel, the Get State API will be called for all BusNms on the channel. This implies that the state buffer and the mode buffer will contain the numerically greatest state/mode of all BusNms. Also, if one of the BusNm_GetState returns E_NOT_OK, the result will not be returned. If one is interested in the state of a particular BusNm on the channel, it is recommended to call BusNm_GetState directly for channels with multiple BusNms.	
Expected Caller Context	
> This function can be called from task and interrupt level.	

Table 4-15 Nm_GetState

4.2.15 Nm_GetVersionInfo

Prototype	
`void` **`Nm_GetVersionInfo`** `( Std_VersionInfoType * nmVerInfoPtr )`	
Parameter	
`nmVerInfoPtr`	Pointer where the version information shall be copied to
Return code	
–	
Functional Description	
Nm_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 synchronous. > This function is reentrant. > This function is only available if ‘Version Info Api’ is enabled.	
Expected Caller Context	
> This function can be called from task and interrupt level.	

Table 4-16 Nm_GetNodeIdentifier

4.2.16 Nm_MainFunction

Prototype	
`void Nm_MainFunction ( void )`	
Parameter	
–	
Return code	
–	
Functional Description	
This function implements the handling of coordinated networks of the NM Interface. **Note** This function is not available if ‘Coordinator Support’ is turned OFF.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is not reentrant. > This function has to be called cyclically on task level by BSW Scheduler > This function must not be called by the application.	
Expected Caller Context	
> This function can be called from task level only.	

Table 4-17 Nm_MainFunction

4.2.17 Nm_InitMemory

Prototype	
`void Nm_InitMemory ( void )`	
Parameter	
–	
Return code	
–	
Functional Description	
If RAM is not automatically initialized at start-up, this function must be called from start-up code to ensure that variables which must be initialized with a certain value (e.g. initialization status with UNINIT value) are set to those values.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant. > This function is a Vector Extension. Refer also to chapter 2.1.2.5 ‘Memory Initialization’. > This function has to be called during start-up and before the initialization is executed. > This function is realized as a macro if ‘Coordinator Support’ is disabled.	
Expected Caller Context	
> This function can be called from task level only	

Table 4-18 Nm_InitMemory

4.2.18 Nm_PreInit

Prototype	
`void` **`Nm_PreInit`** `( const Nm_ConfigType * const nmConfigPtr )`	
Parameter	
`nmConfigPtr`	Configuration structure for initializing the module
Return code	
–	
Functional Description	
This function initializes the Nm configuration pointer and pre-initializes the Nm.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant. > This function has to be called prior to the initialization of the individual Nm instances (i.e. partitions). > This API is realized as a macro if ‘Macro Layer Optimization’ is enabled.	
Expected Caller Context	
> This function can be called from task level only.	

Table 4-19 Nm_PreInit

4.2.19 Nm_PostInit

Prototype	
`void Nm_PostInit ( void )`	
Parameter	
–	
Return code	
–	
Functional Description	
This function finalizes initialization of the Nm.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant. > This function has to be called on the Nm Master Partition. > This function has to be called after the initialization of all individual Nm instances (i.e. partitions). > This API is realized as a macro if ‘Macro Layer Optimization’ is enabled.	
Expected Caller Context	
> This function can be called from task level only.	

Table 4-20 Nm_PostInit

4.2.20 Nm_MainFunction_Satellite

Prototype	
`void` **`Nm_MainFunction_Satellite`** `( ApplicationType applicationID )`	
Parameter	
`applicationID`	Identifier of the currently running OS application.
Return code	
–	
Functional Description	
This function implements the synchronization and forwarding mechanism for multi-partition of the NM Interface. **Note** This function is only available if Multi-Partition is configured and one of the following features is turned ON: - ‘Coordinator Support’ - ‘Node Detection’ - ‘Communication Control’ - ‘User Data’ is used in combination with `Nm_SetUserData()`.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant for different partitions. > This function has to be called cyclically on task level by BSW Scheduler > This function has to be called on each Nm satellite partition, i.e. on each partition where an Nm channel is assigned to. > This function must not be called by the application.	
Expected Caller Context	
> This function can be called from task level only.	

Table 4-21 Nm_MainFunction_Satellite

4.2.21 Nm_RequestSynchronizedPncShutdown

Prototype	
`Std_ReturnType` **`Nm_RequestSynchronizedPncShutdown`** `(NetworkHandleType nmNetworkHandle, PNCHandleType pncId )`	
Parameter	
`nmNetworkHandle`	Identifier of the network
`pncId`	Identifier of the PNC
Return code	
`E_OK`	No error
`E_NOT_OK`	Request for a synchronized PNC shutdown has failed
Functional Description	
This function requests the synchronized PNC shutdown at the lower layer by calling the <BusNm>_RequestSynchronizedPncShutdown bus specific function.	
Particularities and Limitations	
> Service ID: see table 'Service IDs'	
> This function is synchronous.	
> This function is non reentrant for the same nmNetworkHandle, reentrant otherwise.	
> This function is only available if ‘Synchronized PNC Shutdown’ is enabled.	
Expected Caller Context	
> This function can be called from task and interrupt level.	

4.3 Services Used by Nm

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

Component	API
DET	Det_ReportError
BusNm (e.g. CanNm, FrNm, NmOsek)	BusNm_PassiveStartUp BusNm_NetworkRequest BusNm_NetworkRelease BusNm_DisableCommunication BusNm_EnableCommunication BusNm_SetUserData BusNm_GetUserData BusNm_GetPduData BusNm_RepeatMessageRequest BusNm_GetNodeIdentifier BusNm_GetLocalNodeIdentifier BusNm_CheckRemoteSleepIndication BusNm_RequestBusSynchronization

Component	API
	BusNm_SetSleepReadyBit BusNm_GetState BusNm_RequestSynchronizedPncShutdown
OS	GetCurrentApplicationID

Table 4-22 Services used by the Nm

4.4 Callback Functions

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

4.4.1 Nm_NetworkStartIndication

Prototype	
`void` **`Nm_NetworkStartIndication`** `( const NetworkHandleType nmNetworkHandle )`	
Parameter	
`nmNetworkHandle`	Identification of the network
Return code	
–	
Functional Description	
Notification that a NM message has been received in state ‘Bus Sleep’. This indicates that some nodes in the network have restarted and already entered ‘Network Mode’. This notification is forwarded to the ComM (see also chapter 4.5 ‘Callback Functions used by Nm’).	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant.	
Expected Caller Context	
> This function can be called from task and interrupt level.	

Table 4-23 Nm_NetworkStartIndication

4.4.2 Nm_NetworkMode

Prototype	
`void` **`Nm_NetworkMode`** `( const NetworkHandleType nmNetworkHandle )`	
Parameter	
`nmNetworkHandle`	Identification of the network
Return code	
–	
Functional Description	
Notification that network management has entered ‘Network Mode’. This notification is forwarded to the ComM (see also chapter 4.5 ‘Callback Functions used by Nm’).	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant if NM_COORDINATOR_SUPPORT_ENABLED is STD_OFF, otherwise it is reentrant only for different channel handles. > If multiple BusNms are used on a channel, the notification is only forwarded to ComM if it is the first BusNm that has entered ‘Network Mode’ on this channel. > If multiple BusNms are used on a channel, the new mode must also be returned by a BusNm_GetState call within the context of the Nm_NetworkMode call.	
Expected Caller Context	
> This function can be called from task and interrupt level.	

Table 4-24 Nm_NetworkMode

4.4.3 Nm_BusSleepMode

Prototype	
`void` **`Nm_BusSleepMode`** `( const NetworkHandleType nmNetworkHandle )`	
Parameter	
`nmNetworkHandle`	Identification of the network
Return code	
–	
Functional Description	
Notification that network management has entered ‘Bus Sleep Mode’. This notification is forwarded to the ComM (see also chapter 4.5 ‘Callback Functions used by Nm’).	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant if NM_COORDINATOR_SUPPORT_ENABLED is STD_OFF, otherwise it is reentrant only for different channel handles. > If multiple BusNms are used on a channel, the notification is only forwarded to ComM if it is the last BusNm that enters ‘Bus Sleep Mode’. > If multiple BusNms are used on a channel, the new mode must also be returned by a BusNm_GetState call within the context of the Nm_BusSleepMode call.	
Expected Caller Context	
> This function can be called from task and interrupt level.	

Table 4-25 Nm_BusSleepMode

4.4.4 Nm_PrepareBusSleepMode

Prototype	
`void` **`Nm_PrepareBusSleepMode`** `( const NetworkHandleType nmNetworkHandle )`	
Parameter	
`nmNetworkHandle`	Identification of the network
Return code	
–	
Functional Description	
Notification that network management has entered ‘Prepare Bus Sleep Mode’. This notification is forwarded to the ComM (see also chapter 4.5 ‘Callback Functions used by Nm’).	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant if NM_COORDINATOR_SUPPORT_ENABLED is STD_OFF, otherwise it is reentrant only for different channel handles. > This function is only available if CanNm, UdpNm or a Generic BusNm is used or if the Configuration Variant is VARIANT-LINK-TIME > If multiple BusNms are used on the channel, the notification is only forwarded if all other BusNms have left ‘Network Mode’. > If multiple BusNms are used on a channel, the new mode must also be returned by a BusNm_GetState call within the context of the Nm_PrepareBusSleepMode call.	
Expected Caller Context	
> This function can be called from task and interrupt level.	

Table 4-26 Nm_PrepareBusSleepMode

4.4.5 Nm_RemoteSleepIndication

Prototype	
`void` **`Nm_RemoteSleepIndication`** `( const NetworkHandleType nmNetworkHandle )`	
Parameter	
`nmNetworkHandle`	Identification of the network
Return code	
–	
Functional Description	
Notification that network management has detected that all other nodes on the network are ready to sleep. This notification is optionally forwarded to an upper layer by a configurable notification function (see also chapter 4.6.1.1 ’UL_Nm_RemoteSleepIndication’).	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant if NM_COORDINATOR_SUPPORT_ENABLED is STD_OFF, otherwise it is reentrant only for different channel handles. > This function is only available if ‘Remote Sleep Ind Enabled’ is enabled.	
Expected Caller Context	
> This function can be called from task and interrupt level.	

Table 4-27 Nm_RemoteSleepIndication

4.4.6 Nm_RemoteSleepCancellation

Prototype	
`void` **`Nm_RemoteSleepCancellation`** `( const NetworkHandleType nmNetworkHandle )`	
Parameter	
`nmNetworkHandle`	Identification of the network
Return code	
–	
Functional Description	
Notification that network management has detected that some other nodes on the network are not ready to sleep any more. This notification is optionally forwarded to an upper layer by a configurable notification function (see also chapter 4.6.1.2 ‘UL_Nm_RemoteSleepCancellation’).	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant if NM_COORDINATOR_SUPPORT_ENABLED is STD_OFF, otherwise it is reentrant only for different channel handles. > This function is only available if ‘Remote Sleep Ind Enabled’ is enabled.	
Expected Caller Context	
> This function can be called from task and interrupt level.	

Table 4-28 Nm_RemoteSleepCancellation

4.4.7 Nm_SynchronizationPoint

Prototype	
`void` **`Nm_SynchronizationPoint`** `( const NetworkHandleType nmNetworkHandle )`	
Parameter	
`nmNetworkHandle`	Identification of the network
Return code	
–	
Functional Description	
Notification to the NM Coordinator functionality that this is a suitable point in time to initiate the coordination algorithm.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > This function is only available if ‘Coordinator Support’ is enabled.	
Expected Caller Context	
> This function can be called from task and interrupt level.	

Table 4-29 Nm_SynchronizationPoint

4.4.8 Nm_<BusNm>_PduRxIndication

Prototype
`void Nm_<BusNm>_PduRxIndication( const NetworkHandleType nmNetworkHandle, const PduInfoType* const pduInfo );`
Parameter
`nmNetworkHandle` — Identification of the network
`pduInfo` — Pointer to the received PDU data
Return code
`-`
Functional Description
Notification that a NM message has been received by a specific BusNm on a channel. This notification is optionally forwarded to an upper layer by a configurable notification function (see also chapter 4.6.1.4 ‘UL_Nm_BusNmSpecificPduRxIndication’).
Particularities and Limitations
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > This function is only available if ‘Bus Nm Specific Pdu Rx Indication Enabled’ is enabled.
Expected Caller Context
> This function can be called from task and interrupt level.

Table 4-30 Nm_BusNmSpecificPduRxIndication

4.4.9 Nm_PduRxIndication

Prototype	
`void` **`Nm_PduRxIndication`** `( const NetworkHandleType nmNetworkHandle )`	
Parameter	
`nmNetworkHandle`	Identification of the network
Return code	
–	
Functional Description	
Notification that a NM message has been received. This notification is optionally forwarded to an upper layer by a configurable notification function (see also chapter 4.6.1.3 ‘UL_Nm_PduRxIndication’). This notification may also be forwarded to another configurable notication (see chapter 4.6.1.4 ‘UL_Nm_BusNmSpecificPduRxIndication’ for details). Note that the latter upper layer notification function will contain a NULL pointer for the pduInfo argument.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > This function is only available if ‘Pdu Rx Indication Enabled’ is enabled.	
Expected Caller Context	
> This function can be called from task and interrupt level.	

Table 4-31 Nm_PduRxIndication

4.4.10 Nm_StateChangeNotification

Prototype	
`void Nm_StateChangeNotification ( const NetworkHandleType nmNetworkHandle, const Nm_StateType nmPreviousState, const Nm_StateType nmCurrentState )`	
Parameter	
`nmNetworkHandle`	Identification of the network
`nmPreviousState`	Previous state of the BusNm on the respective network
`nmCurrentState`	Current state of the BusNm on the respective network
Return code	
–	
Functional Description	
Notification that network management state of the BusNm has changed. This notification is optionally forwarded to an upper layer by a configurable notification function (see also chapter 4.6.1.5 ‘UL_Nm_StateChangeNotification’).	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > This function is only available if ‘State Change Ind Enabled’ is enabled.	
Expected Caller Context	
> This function can be called from task and interrupt level.	

Table 4-32 Nm_StateChangeNotification

4.4.11 Nm_RepeatMessageIndication

Prototype	
`void` **`Nm_RepeatMessageIndication`** `( const NetworkHandleType nmNetworkHandle )`	
Parameter	
`nmNetworkHandle`	Identification of the network
Return code	
–	
Functional Description	
Notification that a NM message with set repeat message request bit has been received. This notification is optionally forwarded to an upper layer by a configurable notification function (see also chapter 4.6.1.6 ‘UL_Nm_RepeatMessageIndication’).	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > This function is only available if ‘Repeat Msg Ind Enabled’ is enabled.	
Expected Caller Context	
> This function can be called from task and interrupt level.	

Table 4-33 Nm_RepeatMessageIndication

4.4.12 Nm_TxTimeoutException

Prototype	
`void` **`Nm_TxTimeoutException`** `( const NetworkHandleType nmNetworkHandle )`	
Parameter	
`nmNetworkHandle`	Identification of the network
Return code	
–	
Functional Description	
Notification that NM message could not be sent for a certain time period. This notification is optionally forwarded to an upper layer by a configurable notification function (see also chapter 4.6.1.7 ‘UL_Nm_TxTimeoutException’).	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > This function is only available if at least one network is not passive or CONFIG-VARIANT is LINK-TIME.	
Expected Caller Context	
> This function can be called from task and interrupt level.	

Table 4-34 Nm_TxTimeoutException

4.4.13 Nm_CarWakeUpIndication

Prototype	
`void` **`Nm_CarWakeUpIndication`** `( const NetworkHandleType nmNetworkHandle )`	
Parameter	
`nmNetworkHandle`	Identification of the network
Return code	
–	
Functional Description	
Notification that a NM message with set Car Wake Up request bit has been received. This notification is optionally forwarded to an upper layer by a configurable notification function (see also chapter 4.6.1.8) ‘UL_Nm_CarWakeUpIndication’).	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > This function is only available if ‘Car Wake Up Rx Enabled’ is enabled.	
Expected Caller Context	
> This function can be called from task and interrupt level.	

Table 4-35 Nm_CarWakeUpIndication

4.4.14 Nm_CoordReadyToSleepIndication

Prototype	
`void` **`Nm_CoordReadyToSleepIndication`** `( const NetworkHandleType nmNetworkHandle )`	
Parameter	
`nmNetworkHandle`	Identification of the network
Return code	
–	
Functional Description	
Notification that a NM message with set Sleep Ready bit has been received.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > This function is only available if ‘Coordinator Support’ and ‘Coordinator Sync Support’ are enabled	
Expected Caller Context	
> This function can be called from task and interrupt level.	

Table 4-36 Nm_CoordReadyToSleepIndication

4.4.15 Nm_CoordReadyToSleepCancellation

Prototype	
`void Nm_CoordReadyToSleepCancellation (const NetworkHandleType nmNetworkHandle)`	
Parameter	
`nmNetworkHandle`	Identification of the network
Return code	
–	
Functional Description	
Notification that a NM message, which has not set Sleep Ready bit anymore, has been received.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > This function is only available if ‘Coordinator Support’ and ‘Coordinator Sync Support’ are enabled	
Expected Caller Context	
> This function can be called from task and interrupt level.	

Table 4-37 Nm_CoordReadyToSleepCancellation

4.4.16 Nm_ForwardSynchronizedPncShutdown

Prototype	
`void Nm_ForwardSynchronizedPncShutdown (const NetworkHandleType nmNetworkHandle, const uint8* pncBitVectorPtr)`	
Parameter	
`nmNetworkHandle` `pncBitVectorPtr`	Identification of the network Pointer to the PN information
Return code	
–	
Functional Description	
Forwards the synchronized PNC shutdown to the upper layer (ComM). Invoked by the bus specific NM.	
Particularities and Limitations	
> Service ID: see table ‘Service IDs’ > This function is synchronous. > This function is non reentrant for the same nmNetworkHandle, reentrant otherwise. > This function is only available if ‘Synchronized PNC Shutdown’ is enabled.	
Expected Caller Context	
> This function can be called from task and interrupt level.	

4.5 Callback Functions used by Nm

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

Component	API
ComM	ComM_Nm_NetworkStartIndication ComM_Nm_RestartIndication ComM_Nm_NetworkMode ComM_Nm_BusSleepMode ComM_Nm_PrepareBusSleepMode ComM_Nm_ForwardSynchronizedPncShutdown

Table 4-38 Callback Functions used by the Nm

4.6 Configurable Interfaces

4.6.1 Notifications

At its configurable interfaces the Nm defines notifications that can be mapped to callback functions provided by other modules. The mapping is not statically defined by the Nm but can be performed at configuration time. The function prototypes that can be used for the configuration have to match the appropriate function prototype signatures, which are described in the following sub-chapters. The name of those functions is configurable and provided names here are just examples. The header file names where the prototypes for those functions are provided has to be provided also in the configuration.

4.6.1.1 UL_Nm_RemoteSleepIndication

Prototype	
`void` **`UL_Nm_RemoteSleepIndication`** `( const NetworkHandleType nmNetworkHandle )`	
Parameter	
`nmNetworkHandle`	Identification of the network
Return code	
–	
Functional Description	
Notification that network management has detected that all other nodes on the network are ready to sleep.	
Particularities and Limitations	
> Service ID: Has to be provided by the module that implements this notification. > This function is synchronous. > This function is reentrant. > The name of this function is configurable. > This function is only available if ‘Remote Sleep Ind Enabled’ is enabled and a function name is configured. > If multiple BusNms are used on the channel, this function is only called if the last BusNm has indicated ‘Remote Sleep’.	
Expected Caller Context	
> This function can be called from task and interrupt level.	

Table 4-39 UL_Nm_RemoteSleepIndication

4.6.1.2 UL_Nm_RemoteSleepCancellation

Prototype	
`void` **`UL_Nm_RemoteSleepCancellation`** `( const NetworkHandleType nmNetworkHandle )`	
Parameter	
`nmNetworkHandle`	Identification of the network
Return code	
–	
Functional Description	
Notification that network management has detected that some other nodes on the network are not ready to sleep any more.	
Particularities and Limitations	
> Service ID: Has to be provided by the module that implements this notification. > This function is synchronous. > This function is reentrant. > The name of this function is configurable. > This function is only available if ‘Remote Sleep Ind Enabled’ is enabled and a function name is configured. > If multiple BusNms are used on the channel, this function is only called if the first BusNm has cancelled ‘Remote Sleep’.	
Expected Caller Context	
> This function can be called from task and interrupt level.	

Table 4-40 UL_Nm_RemoteSleepCancellation

4.6.1.3 UL_Nm_PduRxIndication

Prototype	
`void` **`UL_Nm_PduRxIndication`** `( const NetworkHandleType nmNetworkHandle )`	
Parameter	
`nmNetworkHandle`	Identification of the network
Return code	
–	
Functional Description	
Notification that a NM message has been received.	
Particularities and Limitations	
> Service ID: Has to be provided by the module that implements this notification. > This function is synchronous. > This function is reentrant. > The name of this function is configurable. > This function is only available if ‘Pdu Rx Indication Enabled’ is enabled and a function name is configured. > If multiple BusNms are used on the channel and this function is called, it cannot be distinguished which BusNm has triggered the call of this function. If the PDU contents are different between the PDUs of BusNms, one can use BusNm_GetPduData in the context of the callback function to get the most recently received PDU data of each BusNm.	
Expected Caller Context	
> This function can be called from task and interrupt level.	

Table 4-41 UL_Nm_PduRxIndication

4.6.1.4 UL_Nm_BusNmSpecificPduRxIndication

Prototype	
`void <FunctionName>( NetworkHandleType nmNetworkHandle, const PduInfoType* const pduInfo)`	
Parameter	
`nmNetworkHandle`	Identification of the network
`pduInfo`	Pointer to the received PDU data
Return code	
-	
Functional Description	
Notification that a NM message has been received. It can be differentiated from which BusNm it comes, in contrast to Nm_PduRxIndication 4.6.1.3.	
Particularities and Limitations	
> Service ID: Has to be provided by the module that implements this notification. > This function is synchronous. > This function is reentrant. > The name of this function is configurable. > This function is only available if 'Specific Pdu Rx Indication Enabled' is enabled and a function name is configured. > This function can be used to distinguish between each BusNm on the same channel by using different identifiers for each BusNm. It is not necessary to configure the function if there is only one BusNm on the channel. The 'Pdu Receive Ind Callback' can be used as an alternative for this purpose. > The argument pduInfo will always be NULL if the function is called from the context of Nm_PduRxIndication (see also chapter 4.4.9).	
Expected Caller Context	
> This function can be called from task and interrupt level.	

Table 4-42 Standard Bus Nm Pdu Rx Indication

4.6.1.5 UL_Nm_StateChangeNotification

Prototype	
`void UL_Nm_StateChangeNotification ( const NetworkHandleType nmNetworkHandle, const Nm_StateType nmPreviousState, const Nm_StateType nmCurrentState )`	
Parameter	
`nmNetworkHandle`	Identification of the network
`nmPreviousState`	Previous state of the BusNm on the respective network
`nmCurrentState`	Current state of the BusNm on the respective network
Return code	
–	
Functional Description	
Notification that network management state of the BusNm has changed.	
Particularities and Limitations	
> Service ID: Has to be provided by the module that implements this notification. > This function is synchronous. > This function is reentrant. > The name of this function is configurable. > This function is only available if ‘State Change Ind Enabled’ is enabled and a function name is configured. > If multiple BusNms are used on the channel and if this function used, it cannot be distinguished which BusNm has triggered the state change notification. The current state is always the numerically highest overall state of the BusNms on the channel. If an exact state needs to be determined for each BusNm, call BusNm_GetState directly.	
Expected Caller Context	
> This function can be called from task and interrupt level.	

Table 4-43 UL_Nm_StateChangeNotification

4.6.1.6 UL_Nm_RepeatMessageIndication

Prototype	
`void` **`UL_Nm_RepeatMessageIndication`** `( const NetworkHandleType nmNetworkHandle )`	
Parameter	
`nmNetworkHandle`	Identification of the network
Return code	
–	
Functional Description	
Notification that a NM message with set repeat message request bit has been received.	
Particularities and Limitations	
> Service ID: Has to be provided by the module that implements this notification. > This function is synchronous. > This function is reentrant. > The name of this function is configurable. > This function is only available if ‘Repeat Msg Ind Enabled’ is enabled and a function name is configured. > If multiple BusNms are used on the channel, it cannot be distinguished which BusNm has called this function.	
Expected Caller Context	
> This function can be called from task and interrupt level.	

Table 4-44 UL_Nm_RepeatMessageIndication

4.6.1.7 UL_Nm_TxTimeoutException

Prototype	
`void` **`UL_Nm_TxTimeoutException`** `( const NetworkHandleType nmNetworkHandle )`	
Parameter	
`nmNetworkHandle`	Identification of the network
Return code	
–	
Functional Description	
Notification that NM message could not be sent for a certain time period.	
Particularities and Limitations	
> Service ID: Has to be provided by the module that implements this notification. > This function is synchronous. > This function is reentrant. > The name of this function is configurable. > This function is only available if ‘Passive Mode Enabled’ is disabled and a function name is configured. > If multiple BusNms are used on the channel, it cannot be distinguished which BusNm has called this function.	
Expected Caller Context	
> This function can be called from task and interrupt level.	

Table 4-45 UL_Nm_TxTimeoutException

4.6.1.8 UL_Nm_CarWakeUpIndication

Prototype	
`void` **`UL_Nm_CarWakeUpIndication`** `( const NetworkHandleType nmNetworkHandle )`	
Parameter	
`nmNetworkHandle`	Identification of the network
Return code	
–	
Functional Description	
Notification that a NM message with set Car Wake Up request bit has been received.	
Particularities and Limitations	
> Service ID: Has to be provided by the module that implements this notification. > This function is synchronous. > This function is reentrant. > The name of this function is configurable. > This function is only available if ‘Car Wake Up Rx Enabled ‘is enabled and a function name is configured. > If multiple BusNms are used on the channel, it cannot be distinguished which BusNm has called this function.	
Expected Caller Context	
> This function can be called from task and interrupt level.	

Table 4-46 UL_Nm_CarWakeUpIndication

5 Glossary and Abbreviations

5.1 Glossary

Term	Description
BusNm	Bus-specific network management, e.g. CanNm, FrNm, NmOsek
DaVinci Configurator	Generation tool for MICROSAR Classic components

Table 5-1 Glossary

5.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
BswM	Basis Software Manager
ComM	Communication Manager
DEM	Diagnostic Event Manager
DET	Development Error Tracer
DLL	Data Link Layer
ECU	Electronic Control Unit
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
Nm	AUTOSAR Network Management Interface (this module)
NM	Network Management
NmOsek	OSEK Network Management (Vector module)
OSEK	Open Systems and the Corresponding Interfaces for Automotive Electronics (German term: “Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug”)
SchM	Schedule Manager
SWS	Software Specification

Table 5-2 Abbreviations

6 Contact

Visit our website for more information on

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

www.vector.com