MICROSAR OSEK Direct Network Management

Technical Reference

Complex Device Driver
Version 6.00.00

Authors	Markus Drescher, Leticia Garcia Herrera
Status	Released

Document Information

History

Author	Date	Version	Remarks
Markus Drescher	2011-02-25	1.00.00	ESCAN00048938 Creation
Markus Drescher	2011-04-01	1.00.01	ESCAN00049772 Adapted API descriptions of NmOsek_CanSM_BusOffBegin (section 4.4.2) / NmOsek_CanSM_BusOffEnd (section 4.4.2.2) ESCAN00049879 Updated sections 1.1, 2.1, 2.3, 2.5, 2.7, 2.8.2, 2.9, 2.10, 3.2, (removed), (removed), 4.2, 4.4, (removed), (removed), 5.2, 5.3, 6.1
Markus Drescher	2011-05-25	1.00.02	ESCAN00050824 Adapted example in section 2.10.2.3 ESCAN00050924 Updated sections 2.1, (removed), (removed), 2.10.3, 4.1, 4.4, (removed), (removed)
Markus Drescher	2011-10-26	1.01.00	ESCAN00052317 Updated sections (removed), 4.1, (removed), (removed) ESCAN00052322 Updated sections (removed), (removed), (removed) ESCAN00052620 Updated section (removed) ESCAN00052621 Updated section 2.10.3 ESCAN00052665 Updated sections 4.1, (removed), (removed) ESCAN00052925 Updated sections 4.1, (removed), (removed) ESCAN00053703 Updated sections (removed), 4.1, (removed), (removed), (removed), (removed), (removed), (removed)
Markus Drescher	2012-05-09	1.02.00	ESCAN00058551 Updated sections (removed), (removed) ESCAN00058552 Updated sections (removed), (removed) ESCAN00058553 Updated sections (removed), (removed) ESCAN00059017 Updated sections (removed), (removed)
Markus Drescher	2012-10-26	1.02.01	ESCAN00060624 Harmonized CddNmOsekLeaveLHOnlyOnRxIndication Enabled descriptions in sections (removed), (removed) ESCAN00062286 Updated sections (removed), 4.1, 5.3.1.7, 5.4.3

			ESCAN00062287 Updated section (removed) ESCAN00062508 Created section (removed), Updated sections (removed), (removed) ESCAN00062509 Created sections (removed), (removed)
Markus Drescher	2013-05-13	1.02.02	ESCAN00066773 Updated chapter 4.3 ESCAN00067327 Updated Figure 1-1
Markus Drescher	2013-09-03	1.02.03	ESCAN00069704 Updated chapter (removed) ESCAN00070181 Updated Figure 1-1
Markus Drescher	2014-12-04	1.03.00	ESCAN00049912 Updated chapter 4.1 ESCAN00078515 Updated chapters 2.1, (removed), 2.5.1, 4.1, 4.2.1.1; created chapter (removed) ESCAN00080355 Updated chapter 2.10.8 ESCAN00080357 Updated chapter 2.3
Markus Drescher	2015-03-17	2.00.00	ESCAN00080496 Updated Reference Documents, updated Figure 1-1, adapted chapters 2.1, 2.3, 2.4, 2.5, 2.8, 2.9, 2.10, 3.1, 3.2, 3.3, 4.1, 4.2, 4.3, 4.4, 6; removed chapters ‘Plausibility Checks for Parameters’, ‘Parameter Checking’, ‘Compiler Abstraction and Memory Mapping’, ‘Configuration’; moved OEM-specific chapters to [6]
Markus Drescher	2015-04-14	3.00.00	ESCAN00081714 Created chapter 2.10.10; updated chapter 4.3 ESCAN00082043 Updated Table 2-8 ESCAN00082863 Updated chapters 2.10.5.2, 2.10.7, 4.3
Markus Drescher	2015-11-19	4.00.00	ESCAN00084669 Created chapters 2.8.3 and 2.10.11, adapted chapters 1.1 and 4.3 ESCAN00084672 Adapted chapters 1, 2.1 ESCAN00086609 Adapted chapter 2.10.10.1 ESCAN00086775 Adapted Table 4-29 and Table 4-30
Markus Drescher	2016-07-11	5.00.00	ESCAN00087038 Adapted chapters 2.5.1, 2.8.3, 2.10.11; added chapter 4.2.1.5 ESCAN00090252 Adapted chapter 2.10.10
Leticia Garcia Herrera	2017-02-10	6.00.00	FEATC-870 Adapted chapters 2.8.3 and 4.3; added chapter 2.10.12

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_SWS_CANNetworkManagement.pdf	V3.3.0
[2]	AUTOSAR	AUTOSAR_SWS_DevelopmentErrorTracer.pdf	V3.2.0
[3]	AUTOSAR	AUTOSAR_SWS_NetworkManagementInterface.pdf	V3.0.0
[4]	AUTOSAR	AUTOSAR_TR_BSWModuleList.pdf	V1.6.0
[5]	OSEK	nm253.pdf	2.5.3
[6]	Vector	TechnicalReference_NmOsek_<OEM>.pdf	see delivery
[7]	Vector	TechnicalReference_Nm.pdf	see delivery
[8]	Vector	TechnicalReference_CanSM.pdf	see delivery
[9]	Vector	TechnicalReference_Rte.pdf	see delivery
[10]	Vector	Startup_<OEM>_SLP<SLP>.pdf	see delivery
[11]	Vector	TechnicalReference_MICROSAR_IdentityManager.pdf	see delivery
[12]	Vector	TechnicalReference_Com.pdf	see delivery
[13]	Vector	TechnicalReference_CanIf.pdf	see delivery

Scope of the Document

This technical reference describes the general use of the OSEK Network Management basis software.

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

Contents

© 2017 Vector Informatik GmbH

Illustrations

Tables

© 2017 Vector Informatik GmbH

1 Introduction

This document describes the functionality, API and configuration of the MICROSAR BSW Complex Device Driver module NmOsek.

Supported AUTOSAR Release*:	4	
Supported Configuration Variants:	pre-compile, post-build-loadable, post-build-selectable	
Vendor ID:	NMOSEK_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	NMOSEK_MODULE_ID	255 decimal (according to ref. [4])

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

This document describes the handling of the software component “MICROSAR OSEK Direct Network Management (Nm_AsrNmDirOsek)”.

That means:

- How to integrate and initialize
- How to configure
- How to operate
- Basic understanding of the NmOsek component

Additionally, this document describes the API functions of NmOsek.

For details on the NM algorithms, please refer to the specification of the OSEK [5].

1.1 Architecture Overview

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

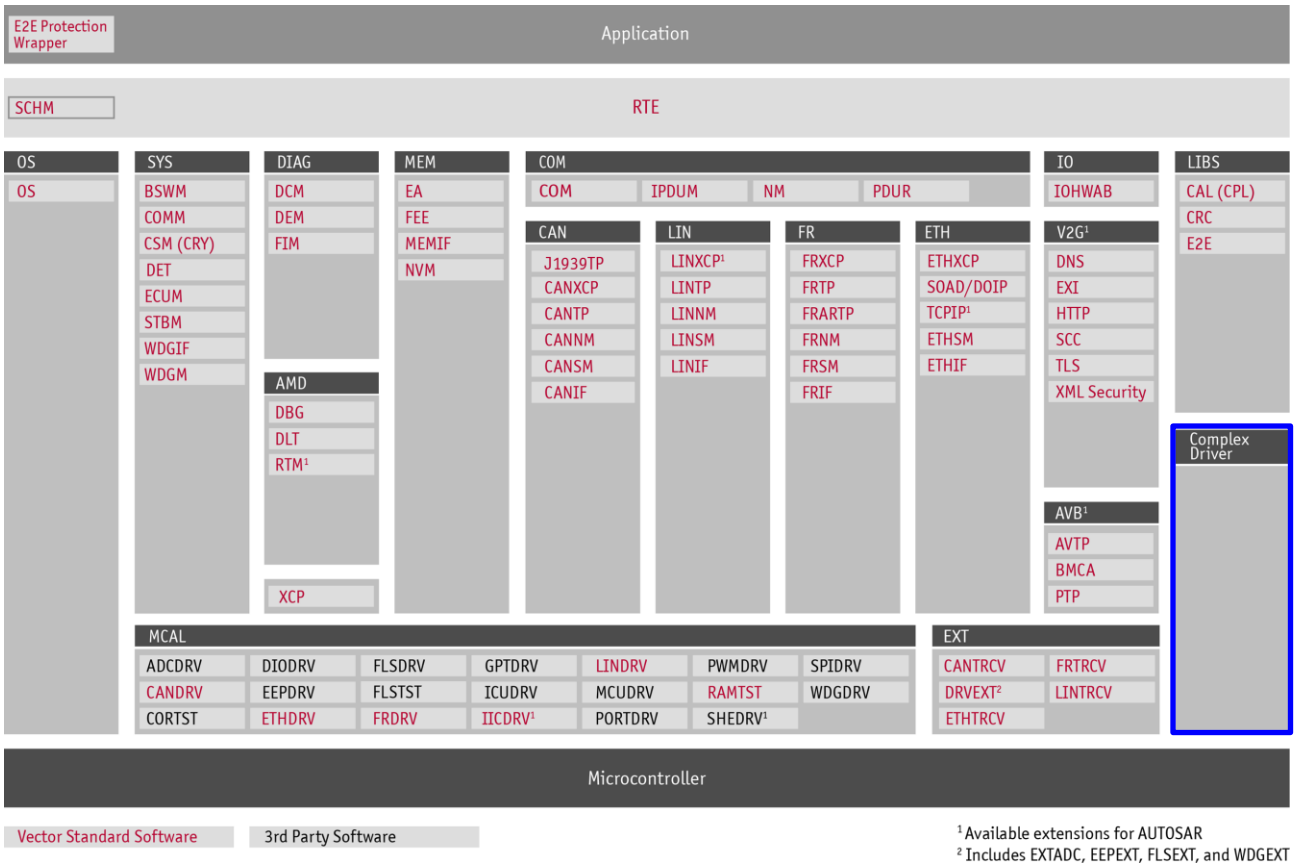

Figure 1-1 AUTOSAR 4.x Architecture Overview

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

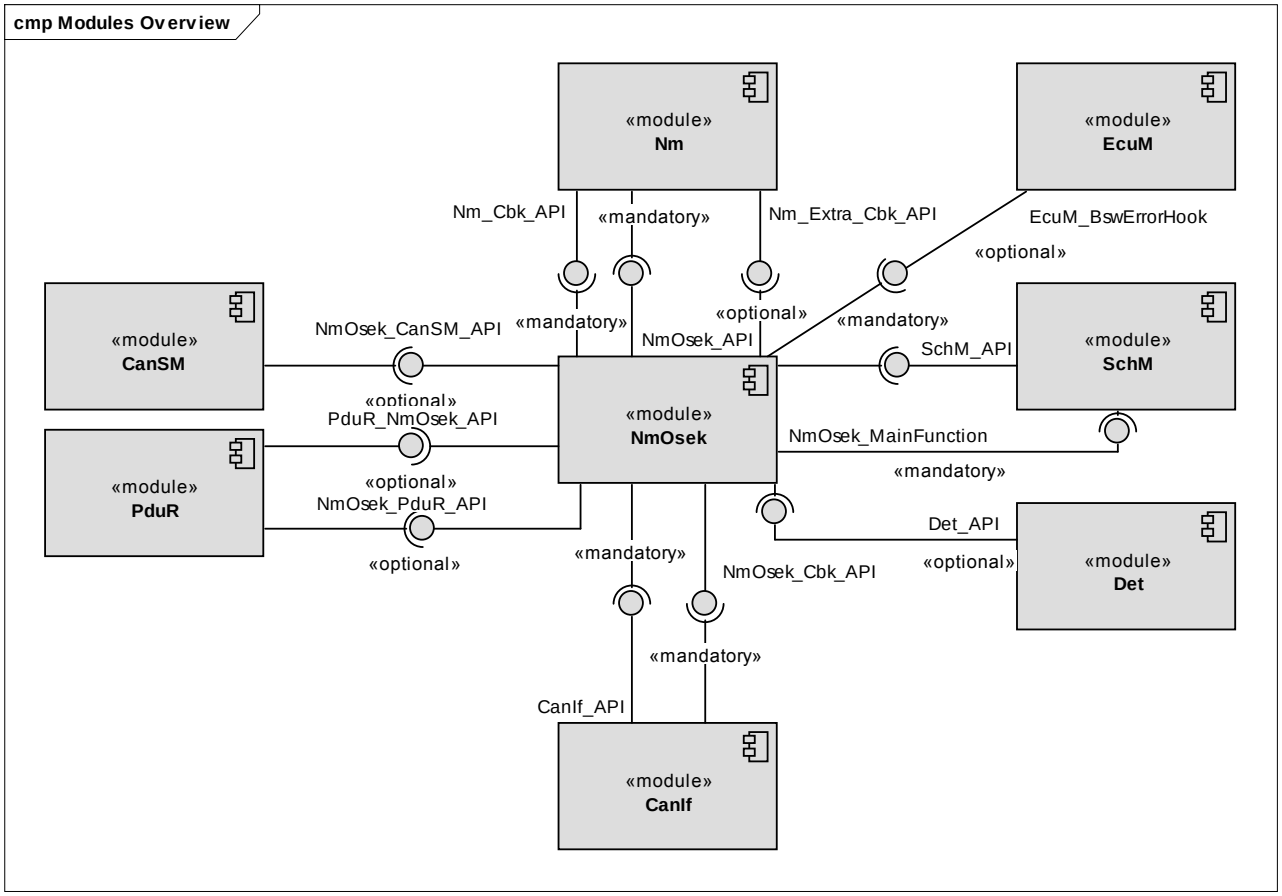

Figure 1-2 Interfaces to adjacent modules of the NmOsek

Note that NmOsek does not provide any service ports for the RTE so that applications can access services via the RTE. However there are some features that allow applications to access services offered by NmOsek. These features are described in the following chapter.

2 Functional Description

2.1 Features

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

Since NmOsek is a Complex Device Driver it is not part of the AUTOSAR standard. However some of its features are similar to those of the AUTOSAR CAN Network Management Basic Software Module whose functionality is specified in [1]. NmOsek is implemented as CAN-specific 'Generic BusNm' as suggested in [3]. Usually the standard AUTOSAR Network Management consists of the following components:

1. Network Management Interface (Nm)[1]
2. CAN Network Management (CanNm)[1]
3. FlexRay Network Management (FrNm)[1]
4. LIN Network Management (LinNm)[1]
5. UDP Network Management (UdpNm)[1]
6. J1939 Network Management (J1939Nm)[1]

The Network Management Interface (1) as specified in [3] may also be used in combination with a Generic BusNm instead of the standard BusNms (2-5) on each controller. For details about this concept please refer to [3].

A Generic BusNm needs to fulfill some AUTOSAR features. These are given in the following tables:

> Table 2-1 Supported AUTOSAR features

> Table 2-2 Not supported AUTOSAR features

For further information of not supported features see also chapter 5.

Vector Informatik provides further NmOsek functionality of the Direct OSEK Network Management that is specified in the OSEK Network Management specification [5].

> Table 2-3 Supported OSEK NM features

Since not all these features are applicable in an AUTOSAR environment there are some replacements which are listed in

> Table 2-4 OSEK NM replacement features.

The following features specified in [1] are supported:

Supported AUTOSAR Features
Operational Modes and Network States in analogy to the CAN Network Management (see section 2.3)
Initialization (see section 2.2)
Remote Sleep Indication (see section 2.10.4)

[1] Not covered by this document

Supported AUTOSAR Features
User Data Handling (see section 2.10.5)
Passive Mode (see section2.10.1)
Node Identifier Services (see section 2.10.6)
Network Management PDU Rx Indication (see section 2.10.7)
State Change Notification (see section 2.10.8)
Communication Control (see section 2.10.9)
Transmission Error Handling (always active) (see section 4.4.1)
Post-Build Loadable
MICROSAR Identity Manager using Post-Build Selectable

Table 2-1 Supported AUTOSAR features

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

Not Supported AUTOSAR Features
Bus Load Reduction Mechanism
AUTOSAR Debugging Concept

Table 2-2 Not supported AUTOSAR features

The following features specified in [5] are also provided:

OSEK NM Features
Providing Advanced Status Information (see section 2.10.3)
Controlling the transition to the bus sleep state and vice versa (see section 2.7)
Determination and monitoring of the network configuration (see section 2.10.2)

Table 2-3 Supported OSEK NM features

The following features are provided as a replacement for the OSEK features:

Replacement OSEK NM Features
BusOff Occurrence and Recovery Notification (instead of BusOff handling by the module) (see section 2.9)
Exchanging user data the AUTOSAR way (see section 2.10.5)

Table 2-4 OSEK NM replacement features

2.2 Initialization

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

Also refer to the function definition of NmOsek_Init() in section 4.2.1.1.

Caution
NmOsek assumes some variables to be initialized to zero at start-up. If your embedded target does not initialize RAM within the start-up code the function 'NmOsek_InitMemory()' has to be called during start-up and before the actual component initialization has been performed. Please also refer to the function definition of NmOsek_InitMemory() in section 4.2.1.

2.3 States

There are several states and sub-states within the Direct OSEK NM:

- NmOn/NmOff
- NmAwake/NmWaitSleep/NmBusSleep
- NmReset
- NmNormal/NmLimpHome
- NmRun/NmPrepSleep
- NmActive/NmPassive

2.3.1 Overview

The mentioned states have a hierarchical linkage (see Figure 2-1).

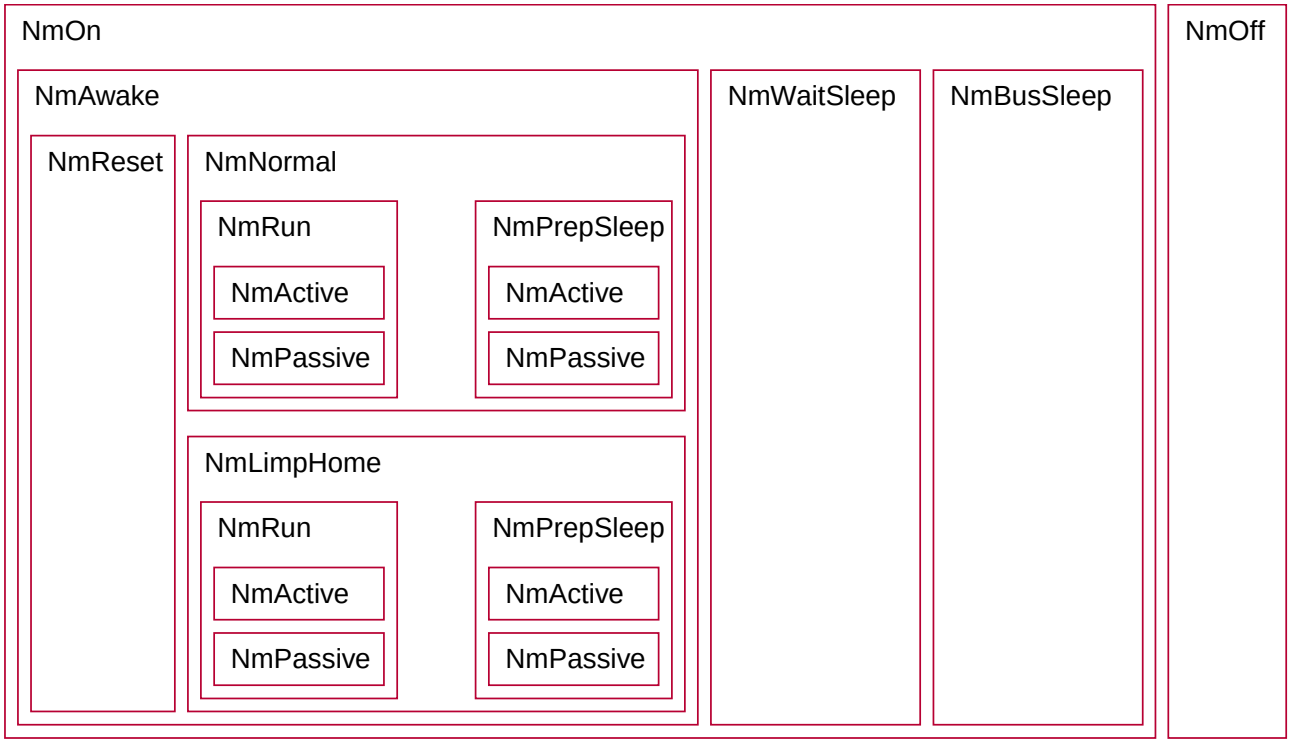

Figure 2-1 States of NmOsek in hierarchical context

Reference name	AUTOSAR NM State	Application status	NM states	Bus	CAN
Reset	Repeat Message Wait Network Startup[2] Wait Network Gw Msg Active[2]	The application may or may not require communication.	NmOn NmAwake NmReset	AWAKE	Online
Normal	Repeat Message Normal Operation Wait Network Gw And Event Msg Active[2]	The application requires communication.	NmOn NmAwake NmRun	AWAKE	Online
SleepInd	Repeat Message Ready Sleep	The application does not require communication anymore.	NmOn NmAwake NmPrepSleep	AWAKE	Online
WaitBusSleep	Prepare Bus Sleep	The application does not require communication.	NmOn NmWaitSleep	AWAKE	Online
Sleep	Bus Sleep	The application does not require communication.	NmOn NmBusSleep	ASLEEP	Offline
PowerOff	(uninitialized)	N/A	N/A	ASLEEP	N/A

Table 2-5 States of OSEK NM

2.3.2 NmOn / NmOff

The state NmOn is the normal state of NmOsek: NmOsek is ready to take part in the logical ring. It sends its NM messages and monitors the network. Internal state transitions can occur. NmOn is entered after initialization (i.e. NmOsek_Init() has been called).

NmOsek cannot be switched to the NmOff state at runtime. NmOsek remains disabled until NmOsek_Init() was called. If it is disabled it does not take part in the logical ring. There are no NM messages from this node and no internal state transitions of NmOsek. Especially, there will be no transition to BusSleep.

2.3.3 NmAwake / NmWaitBusSleep / NmBusSleep

State NmWaitBusSleep is entered if all participating NM nodes have signaled their readiness for BusSleep, i.e. a NM message with a set SleepAcknowledge flag was received or transmitted.

This state is kept for some time before the transition to NmBusSleep occurs automatically.

State NmAwake is entered if there is any need for communication.

[2] Only available if ‘Wake Up Frame Enabled’ is turned ON in the configuration (setting might not be available in your delivery)

2.3.4 NmReset

NmReset exists in order to start building the logical ring. Therefore, when it is left, the ALIVE message is being sent.

NmReset is a transient state (a state that is entered and being left to the next state without an extra trigger) if it is entered when the logical ring needs to be re-established (e.g. due to a missing RING message reception, i.e. TMax timer has timed out or due to a reason to leave NmLimpHome / NmWaitBusSleep / NmBusSleep). The next state is NmNormal if the reception/transmission attempts do not exceed rx_limit/tx_limit (see [5] for details). Otherwise the next state is NmLimpHome.

NmReset may also be a non-transient state if 'Wake Up Frame Enabled' is turned ON in the configuration[2]. In this case NmReset is entered after a wake-up event has occurred (communication request by higher layer or wake-up event on CAN bus) in case TPWONWaitBusSleep[2] has elapsed. When it has been entered in this case, a timer is started with a value of TBWDisableSend[2]. OSEK NM then remains in NmReset until TBWDisableSend[2] has elapsed.

2.3.5 NmNormal / NmLimpHome

NmNormal is the default state if NmOsek is awake. In this state, NmOsek runs the algorithm to establish and run the logical ring. NmOsek can also determine the current network configuration.

State NmLimpHome is entered if a fatal bus error occurs or if NmOsek can't send or receive NM messages. In this state NmOsek cyclically tries to send so-called LimpHome messages. If message transmission and reception works again (e.g. bus error has been removed), this state is left.

2.3.6 NmRun / NmPrepSleep

If NmOsek is in state NmActive in the context of NmRun and NmNormal the node sends its NM message according to the specification.

NmPrepSleep is entered when NmOsek_NetworkRelease() is called. This API indicates that this node does not need communication anymore and the NM node waits for the network to enter BusSleep mode.

NmRun is entered when NmOsek_NetworkRequest() is called. Note that neither NmOsek_NetworkRelease() nor NmOsek_NetworkRequest() are available if the 'Passive Mode Enabled' switch is activated in the NM Interface configuration. In this case the state NmRun is also not available.

2.3.7 NmActive / NmPassive

The NmPassive state becomes available if the 'Passive Mode Enabled' feature is activated in the NM Interface configuration or the 'Com Control Enabled' feature is activated in the NM Interface configuration (see also section 2.10.9).

The NmActive state is unavailable if the 'Passive Mode Enabled' feature is enabled. So NmOsek is always in NmPassive in this case (refer to [7] for further details). That implies that the application can never request communication.

NmOsek is normally in NmActive if the 'Passive Mode Enabled' feature is disabled. In that case a transition to NmPassive is only possible if the 'Com Control Enabled' feature is activated in the Network Management Interface configuration. Table 2-6 provides a

summary of these configuration possibilities. Note that ‘Passive Mode Enabled’ and ‘Com Control Enabled’ cannot be active at the same time.

The state NmActive is the normal state of NmOsek: NmOsek actively takes part in the logical ring. In case the ‘Com Control Enabled’ feature is activated, NmActive can be activated (if not already active) by calling API NmOsek_EnableCommunication().

For the NmPassive state some limitations have to be considered. This state prevents NmOsek from active participation in the ring. That implies that the own network node cannot keep that network awake. In case the ‘Com Control Enabled’ feature is activated and the current NmOsek state is NmAwake (i.e. neither NmBusSleep nor NmWaitBusSleep), NmPassive can be entered by calling the API service NmOsek_DisableCommunication(). Note that the communication must not be released by the application until NmActive has been entered again. If an ECU is in NmPassive state, it remains in its previous main state. It listens to the bus without sending any own NM message.

NM Interface Configuration / State Availability	Passive Mode Enabled: Off Com Control Enabled: Off	Passive Mode Enabled: Off Com Control Enabled: On	Passive Mode Enabled: On Com Control Enabled: Off	Passive Mode Enabled: On Com Control Enabled: On
NmActive	■	■		N/A
NmPassive		■	■	

Table 2-6 Availability of NmActive / NmPassive in dependency of Nm Configuration Settings

Please take a look at [1] and [3] for further details about the ‘Communication Control’ feature.

There is no token inspection and there are no state transitions. However, application messages can still be sent.

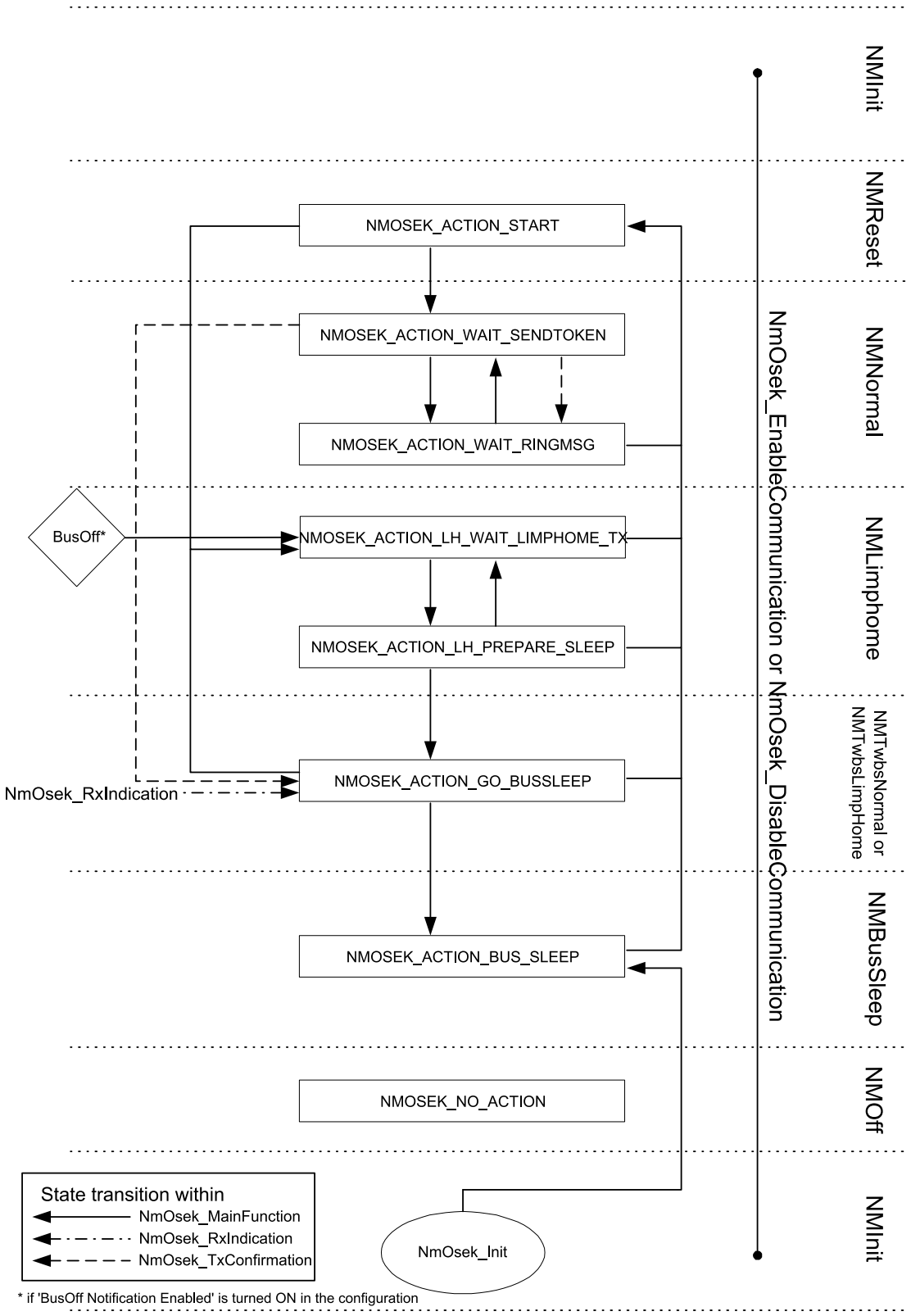

Figure 2-2 Internal state-machine of NmOsek

2.4 Main Function

There is only one main function provided by NmOsek called ‘NmOsek_MainFunction’. This function needs to be called cyclically in the interval configured in the main configuration

settings (refer to section 4.2.1.18 for details). This is usually the task of the AUTOSAR Schedule Manager (SCHM) being part of the RTE. Please refer to [9] for further details about configuring main functions.

2.5 Error Handling

2.5.1 Development Error Reporting

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

If another module is used for development error reporting, the function prototype for reporting the error can be configured by the integrator, but must have the same signature as the service `Det_ReportError()`.

The reported NmOsek ID is 255. The instance ID of NmOsek can be configured at pre-compile time.

The reported service IDs identify the services which are described in 4.2. The following table presents the service IDs and the related services. If a service is not listed in this table, refer also to [6]:

Service ID	Service
0x00	NmOsek_Init()
0x01	NmOsek_PassiveStartUp()
0x02	NmOsek_NetworkRequest()
0x03	NmOsek_NetworkRelease()
0x04	NmOsek_SetUserData()
0x05	NmOsek_GetUserData()
0x06	NmOsek_GetNodeIdentifier()
0x07	NmOsek_GetLocalNodeIdentifier()
0x08	NmOsek_RepeatMessageRequest()
0x0A	NmOsek_GetPduData()
0x0B	NmOsek_GetState()
0x0C	NmOsek_DisableCommunication()
0x0D	NmOsek_EnableCommunication()
0x0F	NmOsek_TxConfirmation()
0x10	NmOsek_RxIndication()
0x13	NmOsek_MainFunction()
0x15	NmOsek_Transmit()
0x20	NmOsek_GetStatus()
0x21	NmOsek_GetConfig()
0x22	NmOsek_CmpConfig()
0x30	NmOsek_CanSM_BusOffBegin()
0x31	NmOsek_CanSM_BusOffEnd()

Service ID	Service
0xC0	NmOsek_RequestBusSynchronization()
0xD0	NmOsek_CheckRemoteSleepIndication()
0xE0	NmOsek_CancelWaitForTxConfOrRxInd()
0xF1	NmOsek_GetVersionInfo()

Table 2-7 Service IDs

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

Error Code		Description
0x01	NMOSEK_E_UNINIT	API service used without module initialization or service called for inactive channel in this pre-defined variant if Post-Build-Selectable support is used
0x02	NMOSEK_E_INVALID_CHANNEL	API service used with wrong channel handle
0x03	NMOSEK_E_INIT_FAILED	NmOsek initialization has failed
0x04	NMOSEK_E_WRONG_WAIT_BUS_SLEEP_FLAG	WaitBusSleep Flag is not correctly set
0x05	NMOSEK_E_WRONG_BUS_SLEEP_FLAG	BusSleep Flag is not correctly set
0x0A	NMOSEK_E_PARAM_CONFIG	API service NmOsek_Init() called with wrong parameter
0x0C	NMOSEK_E_PARAM_POINTER	API service used with invalid pointer parameter (NULL)
0x11	NMOSEK_E_ALREADY_INITIALIZED	The service NmOsek_Init() is called while the module is already initialized
0x21	NMOSEK_E_PDUR_TRIGGERTX_ERROR	Call of PduR_NmOsekTriggerTransmit() failed due to an unexpected PDU length

Table 2-8 Errors reported to DET

2.5.2 Production Code Error Reporting

Currently the NmOsek module does not define any error that can be reported to the DEM module.

2.6 Standard Use Cases

The following table provides some standard use cases how services of the NmOsek are invoked.

Reference	Description	associated NmOsek services
Start (active) network operation	The system requires CAN communication and starts NmOsek. The system is not ready to sleep. NmOsek is informed that it requires the bus. If the current NM state is not NmAwake the function Nm_NetworkMode() is called (refer to [7] for details) within the next call of NmOsek_MainFunction().	NmOsek_NetworkRequest()
Start (passive) network operation	It is required to provide CAN communication due to bus activity. NmOsek is started passively. The system itself is ready to sleep. NmOsek is informed that it does not require the bus actively. If the current NM state is not NmAwake the function Nm_NetworkMode() is called (refer to [7] for details) within the next call of NmOsek_MainFunction().	NmOsek_PassiveStartUp()
Initialization	NmOsek is initialized in SLEEP mode. The communication path at that time is switched off.	NmOsek_Init()
Set sleep indication	The system is ready to sleep and does not need the bus anymore.	NmOsek_NetworkRelease()
Cancel sleep indication	The system is no more ready to sleep and needs the bus again.	NmOsek_NetworkRequest()
Stop the network operation	All ECUs in the network are ready to sleep. NmOsek has set itself into the BusSleep mode. NmOsek announces the successful transition to BusSleep mode with the help of Nm_BusSleepMode() (preceded by a call of Nm_PrepareBusSleepMode() when NmWaitBusSleep has been entered, details see [7]).	—
External wake-up caused by starting network operation	The system operates in local mode. An external wake-up event occurs. The system detects that communication is required. As the system requires the bus itself, NmOsek is started without a set SleepIndication flag.	NmOsek_NetworkRequest()
Demand network operation by the CAN bus	The system operates in local mode. The bus is asleep. The CAN controller detects a dominant level on the CAN bus. The NM node has to participate in the bus traffic because it was woken up by the bus. As the system does not require communication itself, NmOsek is started with a set SleepIndication flag.	NmOsek_PassiveStartUp()

Local wake-up caused by an event without communication demand	The system operates in local mode. An external wake-up event occurs. This event can be processed locally. There is no need to activate the communication. Therefore NmOsek is started in BusSleep mode.	NmOsek_Init()
switch off the CPU (Power-Off)	The system switches off the CPU with the help of a „power on logic“.	—

Table 2-9 Standard use cases

2.7 Control the Transition to the BusSleep Mode

NmOsek ensures that all active ECUs within a network will enter the BusSleep mode simultaneously.

This is required because an ECU will be woken up by any message on the CAN bus. That is the reason why a transition of a single ECU into the BusSleep mode only makes sense, if all the other ECUs are also ready for BusSleep mode and will not send further messages.

The NM algorithm (see [5]) detects the need to enter BusSleep mode or to wake up the bus again. That means that NmOsek decides when the CAN channel must be activated or deactivated.

NmOsek is ready for BusSleep if the system decides that no more CAN communication is needed. In this case NmOsek_NetworkRelease() was called by the system. NmOsek is also ready for BusSleep if only NmOsek_PassiveStartUp() was called but not NmOsek_NetworkRequest(). The readiness for BusSleep remains until the NmOsek service NmOsek_NetworkRequest() is called.

The transition to BusSleep may start when all network nodes are ready to sleep, i.e. Ring messages by all participants in the logical ring have been sent where the Sleep Indication bit is set. The last Ring message before the transition to BusSleep starts has both the Sleep Indication and the Sleep Acknowledge bit set. The next state is NmWaitBusSleep and Nm_PrepareBusSleepMode() is called when this state has been entered. The transition from NmWaitBusSleep to NmBusSleep takes place if there is no wake-up event after NmWaitBusSleep has been entered.

NmOsek calls the function Nm_BusSleepMode() after the transition into the BusSleep mode. The successful transition to BusSleep mode can also be checked by the status flag ‘BusSleep’ of NmOsek (see section 2.10.3) or the NmOsek_GetState() function (see section 4.2.1.14).

2.8 Network Management Message Transmission and Reception

2.8.1 AUTOSAR CAN Interface

NmOsek requests the transmission of NM messages by calling the service `CanIf_Transmit`. The application has to take care of the user data. For details refer to section 2.10.5.

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

```
void NmOsek_TxConfirmation(PduIdType nmOsekNmTxPduId)
```

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

```
void NmOsek_RxIndication(PduIdType nmOsekRxPduId,
   const uint8* CanSduPtr, const Can_IdType canId)
```

Hint: these functions are called by NmOsek_TxConfirmation_X() / NmOsek_RxIndication_X(), X = 0,…,(Number of configured OSEK NM channels - 1) if ‘Api Optimization’ is turned OFF in the configuration. If so, the NmOsek_TxConfirmation_X() / NmOsek_RxIndication_X() functions themselves are called by CanIf instead.

2.8.2 PDU Message Layout

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

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 7	User data 5 / Unused							
Byte 6	User data 4 / Unused							
Byte 5	User data 3 / Unused							
Byte 4	User data 2 / Unused							
Byte 3	User data 1 / Unused							
Byte 2	User data 0 / Unused							
Byte 1	Operation Code							
Byte 0	Destination Identifier							

Table 2-10 PDU NM Message Layout

The NM PDU elements are given in Table 2-11. For details about these bytes please refer to the OSEK specification [5] and to the OEM-specific Technical Reference [6].

NM PDU Element	Description
Destination Identifier	Recipient node identifier / successor in the logical ring that shall send the next NM Message
Operation Code	Operation Code that contains the NM message type (ALIVE, RING, LIMPHOME message) and Sleep Flags (Sleep.ind, Sleep.ack).
User Data	Optional user data bytes that might be filled in the NM TX PDU by using the Nm API Nm_SetUserData() and may be retrieved by calling Nm_GetUserData() (see also [7]). As an alternative, User Data may also be set by using Com Signals (if ‘Com User Data Support’ is turned ON in the Nm module).

Table 2-11 NM PDU Elements

ALIVE and RING messages are sent in NmNormal and NmNormalPrepSleep, LIMPHOME messages are sent in NmLimpHome. For more details, please refer to [5].

The NM message contains a Destination Identifier byte. This byte refers to the logical successor in the ring. The node addresses itself, if there is currently no logical successor.

For the algorithm how the Destination Identifier byte/logical successor is determined refer to chapter 2.2.6 of [5].

Furthermore, the NM Message contains a Sleep Indication bit and a Sleep Acknowledge bit.

The Sleep Indication bit is set to 1 if the own node is ready to sleep, otherwise to 0.

The Sleep Acknowledge bit is only set in RING messages. If all nodes have indicated their readiness to sleep during one ring cycle, the next node in the ring will send a RING message with Sleep Acknowledge bit set to 1. In other words, a node sends the Sleep Acknowledge Bit set to 1 if no NM message with the Sleep Indication bit set to 0 has been received during the last ring cycle. The Sleep Acknowledge bit is never set in ALIVE or LIMPHOME messages.

The last LIMPHOME message from a node before it goes to sleep has the Sleep Indication bit set to 1, the Sleep Acknowledge bit set to 0.

2.8.3 TX Message Observation

NmOsek provides the possibility to observe OSEK NM message transmission requests. This can be achieved by two possible approaches:

- defining the ‘Tx Confirmation Timeout Time’ parameter in the NmOsek channel configuration
- using the ‘Tx Deadline Monitoring’ mechanism of Com.

The basic algorithm of the first approach works as follows, the second approach is described in chapter 2.10.12:

- OSEK NM issues a transmission request for a NM message (CanIf_Transmit()).
- If the transmission request has been accepted (CanIf_Transmit() returns E_OK), the Message Timeout Timer is started with the configured ‘Tx Confirmation Timeout Time’.
- If a message transmission is successful on the bus, the Message Timeout Timer is stopped within NmOsek_TxConfirmation().
- If the Message Timeout Timer expires, Nm_TxTimeoutException() is called.
- If another NM message transmit request is accepted, the Message Timeout Timer is restarted with the configured ‘Tx Confirmation Timeout Time’.

This feature is mandatory for the feature ‘First Message shall be NM Message’ (see also chapter 2.10.11) and may also be mandatory for other OEM-specific features.

2.9 BusOff Occurrence and Recovery Notification

In AUTOSAR environments, CANSM is usually responsible for BusOff recoveries. Therefore, in contradiction to [5], OSEK NM is not responsible for BusOff recoveries but can be notified about BusOff occurrences by CanSM.

When a BusOff occurs the transition to NmLimpHome takes place if the CANSM notifies NmOsek about a BusOff occurrence. The timer TError is started. NmLimpHome may be left when the CANSM notifies NmOsek about a BusOff recovery and the timer TError has elapsed.

In order to use this feature, please make sure that the ‘BusOff Notification Enabled’ feature is activated in the NmOsek settings in the configuration tool. Also verify that the ‘BusOff Begin / End’ notifications are as well activated in the CANSM settings.

Caution
It is highly recommended to turn the ‘BusOff Notification Enabled’ setting ON in the configuration tool to have an OSEK NM behavior that conforms to [5].

If the ‘BusOff Notification Enabled’ feature is activated in the NmOsek configuration the implementation of NmOsek_CanSM_BusOffBegin() and NmOsek_CanSM_BusOffEnd() is available. The call of these functions is considered in the next NmOsek_MainFunction() function call (also refer to section 4.4.2 and 4.4.2.2 as well as the CANSM Technical Reference [8]).

As follows, the behavior of OSEK NM is described for two scenarios: either OSEK NM does not know about BusOff (‘BusOff Notification Enabled’ is turned OFF) or OSEK NM is notified by CANSM about the BusOff state (‘BusOff Notification Enabled’ is turned ON). Usually, the decision whether OSEK NM is notified about BusOff or not is decided by the OEM.

2.9.1.1 Behavior without Knowledge about BusOff State

In this setup, OSEK NM is not notified about BusOff occurrences and thus the transition to NmLimpHome may be delayed or even may not take place at all. This setup requires the ‘BusOff Notification Enabled’ setting to be disabled.

In case a BusOff occurs and the bus disturbance remains for a longer time (multiple BusOff events may be consequence), the transition to NmLimpHome occurs in case the number of missing TX Confirmations tx_{conf} and/or the number of RX Indications rx_{ind} for NM messages exceed their limit(s), that is if

$$(tx_{conf} > TxLimit) \vee (rx_{ind} > RxLimit)$$

evaluates to true.

Depending on the ‘Main Function Period’ $t_{NmCycle}$ (unit: ms) and the limits *TxLimit* (default value: 8 times) and *RxLimit* (default value: 4 times) it takes at least

$$LH_{delayed}(TxLimit, RxLimit, T_{RingMax}, T_{RingTyp}, t_{NmCycle}) :=$$

$$\min(\lfloor TxLimit/2 \rfloor * (T_{RingMax} + T_{RingTyp}), RxLimit * (T_{RingMax} + T_{RingTyp})) + t_{NmCycle} + t_{Additional}$$

milliseconds until the transition to NmLimpHome transition takes place where $t_{Additional}$ denotes the time it takes until the timer $T_{RingTyp}$ or $T_{RingMax}$ times out.

In the following figures, situations are depicted where a BusOff event occurs and the bus disturbance remains as long as it takes to enter NmLimpHome. Usually, NmLimpHome would be entered directly if OSEK NM had knowledge about the BusOff event and if BusOff has just been detected. Due to the lack of knowledge about BusOff, NmLimpHome is only entered due to the exceeded threshold of the *TxLimit* and/or *RxLimit*.

If BusOff occurs while another node is addressed by the last RING message, the sequence contains the attempts by OSEK NM to re-establish the RING by trying to send ALIVE and RING messages itself:

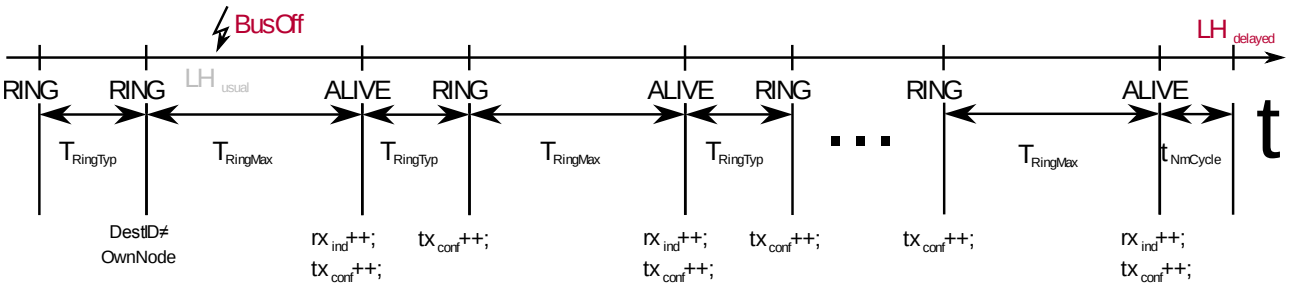

Figure 2-3 Temporal sequence when OSEK NM is not explicitly notified about the BusOff event if it is not addressed by the last RING message

The actual transition in Figure 2-3 is taken when the last increment of the tx_{conf} variable reaches *TxLimit*. This value is checked within the next main function call of NmOsek after $t_{NmCycle}$ milliseconds. This point is indicated by $LH_{delayed}$. Using the default values and $t_{NmCycle}$:= 20 ms, the NmLimpHome transition offset calculates to $LH_{delayed}$(8, 4, 260, 100, 20) = 1460 ms.

If BusOff occurs while the own node is addressed by the last RING message, the sequence of ALIVE and RING message looks like this:

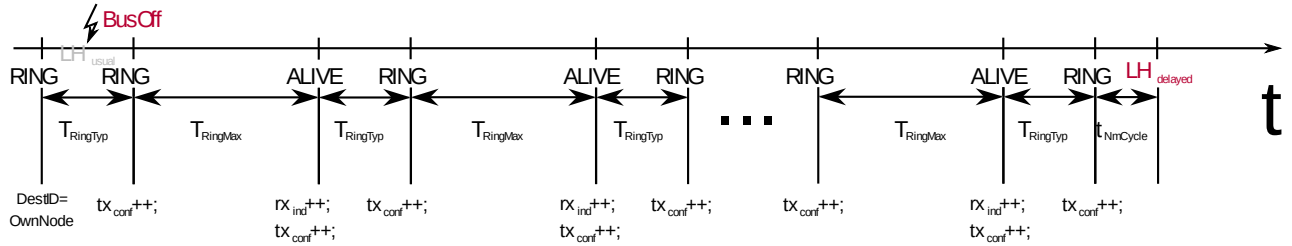

Figure 2-4 Temporal sequence when OSEK NM is not explicitly notified about the BusOff event if it is addressed by the last RING message

Note that there are other situations where BusOff may occur which are not illustrated here.

The LimpHome messages are delayed by $LH_{delayed}$, too.

Note
As it can be seen, the missing knowledge about BusOff is a deviation to the original OSEK NM Specification [5].

2.9.1.2 Behavior with Knowledge about BusOff State

If OSEK NM is notified about the BusOff state by CANSM or another module (e.g. CDD), the behavior concerning the transition to NmLimpHome is in accordance with the original OSEK NM Specification [5]. This setup requires the ‘BusOff Notification Enabled’ setting to be enabled.

However, OSEK NM does not perform the BusOff recovery itself but waits for the BusOff End notification until the LimpHome message is tried to be sent.

So the timings concerning BusOff recoveries in CANSM need to be configured in accordance with the OSEK NM timings.

An example for the BusOff recovery by CANSM and the BusOff notification to OSEK NM is depicted in Figure 2-5.

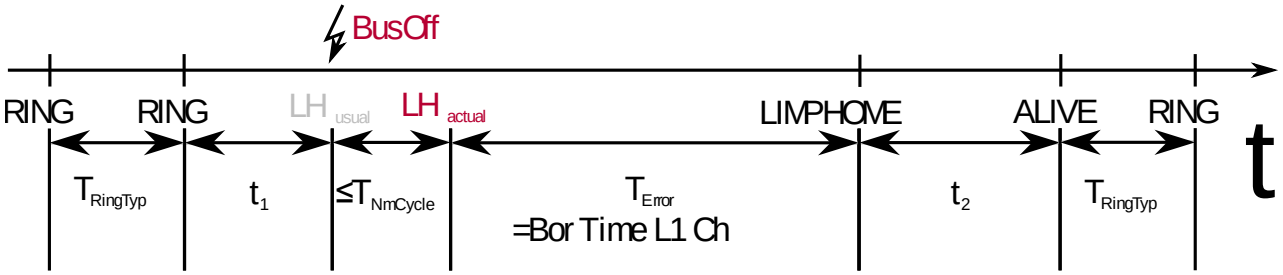

Figure 2-5 Temporal sequence when OSEK NM is notified about the BusOff event

In this example, after the last RING message has been received or transmitted, BusOff is detected after t_1 and the NM is notified about BusOff occurrence (through NmOsek_CanSM_BusOffBegin). Then, NmLimpHome is entered at LH_{actual} in the next NmTask function call (thus the maximum delay is $T_{NmCycle}$) and the timer T_{Error} is started. After T_{Error} has elapsed and the NM has been notified about a BusOff Recovery attempt (through NmOsek_CanSM_BusOffEnd), the NM LIMPHOME message is tried to be sent. If another NM message is received (in this example after t_2) and the NM message has been sent successfully, the NM restarts with an NM ALIVE and an NM RING message.

In Table 2-12, the configuration settings of OSEK NM and CANSM are listed that need to be in accordance with each other.

OSEK NM Setting	CANSM Setting
Module-wide settings	
BusOff Notification Enabled	BusOff Begin and BusOff End need to be defined
Channel-specific settings	
Limp Home Time	Bor Time L2 Ch

Table 2-12 Settings of OSEK NM and CANSM to be configured in accordance if ‘BusOff Notification’ is used

CANSM settings of ‘Bor Time L1 Ch’ and ‘Bor Time L2 Ch’ being different are explicitly allowed.

Note that CANSM has different strategies for returning to the ‘Bor Time L1 Ch’ time if ‘Bor Time L2 Ch’ has been previously used as BusOff recovery time when the bus disturbance is gone.

2.10 Other Features

This section describes further features of the NmOsek module.

2.10.1 Passive Mode

Nodes in passive mode cannot transmit NM messages and therefore they do not actively participate in the network. Due to this fact passive nodes cannot request the network and the state changes are performed according to the last received NM message or timeouts.

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

© 2017 Vector Informatik GmbH

NmOsek always stays in the NmPassive state. Please refer to section 2.3.7 for more details about the NmPassive state.

2.10.2 Determine and Monitor the Network Configuration

The network configuration is determined in the so-called “Start Phase”. Each NM node regards itself as ready for operation.

After the “Start Phase”, NmOsek monitors the network. This “Monitoring Phase” is used to detect new nodes as well as the loss of existing nodes.

This feature called ‘Use Nm Node List’ can be enabled/disabled in the configuration tool. In order to save runtime and memory resources, do not enable this feature unless you need it.

Please make sure that the ‘Number of Nm Config Nodes’ setting contains the maximum number of NM nodes in all networks (channels), i.e. check all channel-specific settings for the ‘NM Message Count’ attribute and verify that the ‘Number of Nm Config Nodes’ contains the maximum of all ‘NM Message Count’ values.

2.10.2.1 Configuration Update by NmOsek

The network nodes that are currently active on the channel are stored inside a byte array. Each bit in this array represents a network node. A bit inside this array is set if a network node has been detected as present in the logical ring. The term ‘network configuration’ refers to this byte array.

The network configuration of NmOsek is updated each time a NM message is received, i.e. usually in ISR-context. Please be aware of this behavior if multiple decisions in the application are necessary to gain a result (e.g. multiple nested ‘if”). Maybe it is necessary to copy the current network configuration to a temporary variable.

The configuration is discarded when the ring is not stable anymore, e.g. if a node gets lost or a bus error occurs.

2.10.2.2 Retrieve Configuration

The information about the network nodes that are currently active may be retrieved by using the API NmOsek_GetConfig().

2.10.2.3 Compare Configuration

NmOsek provides the possibility to compare the current network configuration with a reference configuration (see API NmOsek_CmpConfig()).

This reference configuration is stored in two arrays that have to be provided by the application:

Variable Name	Description
NmOsek_TargetConfigTable	Indicates the required status of the NM nodes. If the corresponding bit is set (‘1’), the associated NM node has to be present. If the bit is not set (‘0’), the node may not be present.
NmOsek_ConfigMaskTable	Indicates which NM nodes have to be checked. If the corresponding bit is set (‘1’), the presence of the associated NM node has to be evaluated. If the bit is not set (‘0’), the node is not of interest for the comparison.

Variable Name	Description
	The assignment between the node address and the corresponding bit/byte position is depicted in Table 2-14.

Table 2-13 Reference Configuration Tables

Byte	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0	7	6	5	4	3	2	1	0
1	15	14	13	12	11	10	9	8
2	23	22	21	20	19	18	17	16
…	…	…	…	…	…	…	…	…
n/8	n-1	n-2	n-3	n-4	n-5	n-6	n-7	n-8

Table 2-14 Assignment between node address and bit/byte position for n stations

Bit 0 of byte 0 represents the NM node with station address 0. Bit 7 of the most significant byte represents the NM node with the highest station address.

An ECU which should be present in the reference configuration is marked with ‘1’ at the corresponding position in both tables.

An ECU which should not be present in the reference configuration is marked with ‘1’ at the corresponding position in NmOsek_ConfigMaskTable and ‘0’ in the NmOsek_TargetMaskTable.

An ECU which should not be part of the comparison is marked with ‘0’ at the corresponding position in NmOsek_ConfigMaskTable.

NmOsek_CmpConfig() compares the current configuration with the reference only for those nodes that are marked with ‘1’ in ConfigMaskTable.

Example

```
#include "NmOsek.h"

/* ROM tables provided by the application */
CONST(NmOsek_NodeConfigType, NMOSEK_APPL_CONST)
NmOsek_TargetConfigTable[2] =
{
  { 0x69,  /* NM node 6,5,3,0 */   /* Configuration: Net1
*/
    0x00,
    0x80,  /* NM node 23 */
    0x80   /* NM node 31 */
  },
  { 0x02,  /* NM node 1 */     /* Configuration: LimpHome
*/
    0x12,  /* NM node 12,9 */
    0x11,  /* NM node 20,16 */
    0x40   /* NM node 30 */
  },
}
CONST(NmOsek_NodeConfigType, NMOSEK_APPL_CONST)
NmOsek_ConfigMaskTable[2] =
{
  { 0xF9,  /* NM node 7..3,0 */    /* Configuration: Net1
*/
    0x00,
    0x80,  /* NM node 23 */
    0x80   /* NM node 31 */
  },
  { 0x0F,  /* NM node 3..0 */     /* Configuration: LimpHome
*/
    0x12,  /* NM node 12,9 */
    0x11,  /* NM node 20,16 */
    0xC1   /* NM node 31,30,24 */
  },
}
/* application */
…
#define LIMPHOME_DESIRED_CONFIG    1
…
if( NmOsek_CmpConfig(LIMPHOME_DESIRED_CONFIG)==1 )
{…;}
```

2.10.3 Provide Status Information

NmOsek provides information on the status of the NM node and the network itself.

This status information can be accessed with the help of APIs NmOsek_GetStatus(). These two services differ in the way the result is passed back. The result itself is of type uint8 (see Table 2-15).

Bit	Name	Interpretation	Mask for bitwise AND
0	Ring Stable	Indicates if the current network is stable. Stable means that the configuration has not changed during the last ring cycle. 0 ring is not stable 1 ring is stable	0x01
1	Bus Off Error	Indicates if a Bus Off has occurred 0 no Bus Off error 1 Bus Off error occurred. CAN is locked. CAN driver is offline.	0x02
2	Active	Indicates the operation mode of NmOsek 0 NMPassive (see chapter 2.3.7) 1 NMActive	0x04
3	LimpHome	Indicates if NmOsek is in the mode LimpHome. LimpHome is entered, if NmOsek can't send or receive NM messages. 0 not in state LimpHome 1 in state LimpHome	0x08
4	BusSleep	Indicates if the bus is asleep or not 0 not in state BusSleep 1 in state BusSleep	0x10
5	WaitBusSleep	Indicates if NmOsek currently waits for BusSleep 0 NmOsek is not in state NmWaitBusSleep 1 NmOsek is in state NmWaitBusSleep	0x20
6	TxRingData Allowed	Indicates if a write access on the user ring data is allowed or not. 0 write access on ring data is allowed 1 write access on ring data is not allowed Note: This flag is not supported by all OEMs[3]	0x40
7	SleepIndication	Indicates if the application has signaled readiness for Sleep or not. 0 GotoMode(Awake) was called 1 GotoMode(SleepInd) was called	0x80

Table 2-15 Access on NmOsek status information

The OSEK specification [5] mentions a status flag that indicates if NmOsek is started or not (NmOn, NmOff). This status flag is not supported by the NmOsek component. Also refer to the OSEK specification for further details about these status flags.

The status information can be evaluated as follows:

[3] This flag is only set to 0 if the 'Token Monitoring' feature is enabled. The user data in the NM PDU may be set independently of the TxRingDataAllowed flag using the NmOsek_SetUserData() function.

Let

`uint8 osekStatus;`

and the call of

`NmOsek_GetStatus``(channel, &osekStatus)`

returns E_OK for a valid channel index `channel`. Then a status bit in `osekStatus` is set if and only if

`((osekStatus & bitMask) == bitMask)`

evaluates as true where `bitMask` is being a mask for a certain flag from Table 2-15.

In the following example the Active flag shall be evaluated.

Example

```
…
uint8 netState;
NetworkHandleType channel = 0u;

NmOsek_GetStatus(channel, &netState);

if ((netState & 0x04 /* Active */) == 0x04)
{…;}
else
{…;}
…
```

Note

The state of NmOsek is updated each time a NM message is received or transmitted, i.e. usually in ISR-context. Please be aware of this behavior if the application uses multiple decisions to get a result (e.g. multiple nested “if”).

2.10.4 Gateway Functionality: Remote Sleep Indication and Cancellation

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

`void` **`Nm_RemoteSleepIndication`**`(const NetworkHandleType)`

In case a NM message is received with Sleep.ind = Sleep.ack = 0 or a Skipped Alive message is transmitted the function

`void` **`Nm_RemoteSleepCancellation`**`(const NetworkHandleType)`

is called. This is also the case when the NmAwake state is entered or a NM message with Sleep.ind = 1 and Sleep.ack = 1 has been received but the application still needs bus communication.

The current Remote Sleep Status may be determined by the function

```
Std_ReturnType NmOsek_CheckRemoteSleepIndication(const
NetworkHandleType, boolean * const nmRemoteSleepIndPtr)
```

(refer to section 4.2.1.17 for more details about this function).

2.10.5 User Data Handling

User data may be set in the NM TX PDU and user data may be retrieved from NM RX PDUs.

2.10.5.1 Setting User Data in the NM TX PDU

There are two possibilities to set user data in the NM TX PDU. Setting User Data via the Com module is simpler if the signals inside the user data bytes are modelled in the database.

Both possibilities do not conform to [5] since user data can also be set when the logical ring is still unstable.

2.10.5.1.1 Setting User Data via the Com Module

This is an optional feature and can be used if ‘Com User Data Support’ is turned ON in the Nm module configuration.

OSEK NM supports the possibility to write the NM user data via Com signals (refer to [12] for further details). Therefore the signals have to be provided within one or two extra I-PDUs in the configuration. If two I-PDUs are used, the first I-PDU contains user data for wake-up (ALIVE) messages, e.g. reasons why an ‘ALIVE’ message is sent. The second I-PDU contains user data for awake (RING, LIMPHOME) messages, so for instance reasons for sending these messages are given.

The signals can for instance be set in the Application by using Rte_Write_<Signal name>() or by a CDD by using Com_SendSignal().

If only one user data I-PDU is used, its contents is used in ALIVE; RING and LIMPHOME messages.

Note that there needs to be a Routing Path in PduR for each user data I-PDU.

2.10.5.1.2 Setting User Data via Nm_SetUserData()

Inside the NM Interface configuration (cf. [7]) the ‘User Data Enabled’ switch might be activated (but it could also not be activated if your configuration prohibits this feature).

If activated, the NM Interface function Nm_SetUserData() can be used to set user data bytes (unless ‘Com User Data Support’ is ON).

All bytes that are not used as system bytes (see section 2.8.2 for details) can be customized by using Nm_SetUserData().

Caution
The user data set by Nm_SetUserData() may be lost after certain types of NM messages are being sent by the own node in the network.

2.10.5.2 Getting User Data from the NM RX PDU

Nm_GetUserData() and Nm_GetPduData() may be used to retrieve user data bytes from the most recently received NM PDU. It makes sense to call these functions in the context of Nm_PduRxIndication() or Nm_NmOsek_PduRxIndication() (there are possibilities to configure callout functions in Nm when this function is being called).
Within the context of Nm_NmOsek_PduRxIndication(), it is even easier to access the PDU data bytes: just use the pointer to the PduInfoType data structure.

2.10.6 Node Identifier Services

The local source node identifier can be retrieved by the service

```
Std_ReturnType NmOsek_GetLocalNodeIdentifier (
   const NetworkHandleType nmChannelHandle,
   uint8* const nmNodeIdPtr )
```

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

```
Std_ReturnType NmOsek_GetNodeIdentifier (
   const NetworkHandleType nmChannelHandle,
   uint8* nmNodeIdPtr )
```

Refer to section 4.2.1.10 and 4.2.1.11 for details about these functions.

2.10.7 NM PDU Receive Indication

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

```
Nm_NmOsek_PduRxIndication ( const NetworkHandleType
                            nmChannelHandle, const
                            PduInfoType * const pduInfo )
```

or if this function is not configured, it also possible to be notified by the optional function

```
Nm_PduRxIndication ( const NetworkHandleType nmChannelHandle )
```

OSEK NM notifies the callback directly to the NM Interface in context of the function `NmOsek_RxIndication()`.

Caution

`NmOsek_RxIndication()` might be called on interrupt level so Nm_NmOsek_PduRxIndication() / Nm_PduRxIndication() might be called on interrupt level as well.
The implementation of Nm_NmOsek_PduRxIndication() / Nm_PduRxIndication() should consume as few runtime as possible.
A burst of Nm_NmOsek_PduRxIndication() / Nm_PduRxIndication() function calls may occur on ECU startup when all nodes in the network are sending their Alive messages almost at the same time.

2.10.8 State Change Notifications

If the ‘State Change Ind Enabled’ feature is activated in the NM Interface configuration, the NmOsek uses the Nm_StateChangeNotification() service to notify the application about state changes.

In configurations where ‘State Change Ind Enabled’ is activated NmOsek calls:

Nm_StateChangeNotification(nmChannelHandle, oldState, newState)

This function is being called whenever a state change from one state to a different state from Nm_StateType (details about this type can be found in [7]) occurs.

If the ‘State Change Ind Enabled’ feature is activated a function is required whose name may be configured in the NM Interface configuration [7]. If the Vector MICROSAR BswM is available, State Change Notifications can be used for handling the activation and deactivation of RX/TX messages (‘Communication Control’; not to be confused with the following chapter) that functionality is implemented in this module. If not the application needs to implement this function (see ‘UL_Nm_StateChangeNotification’ in [7] for the function prototype).

2.10.9 Communication Control

In order to support ISO 14229 Communication Control Service $28 the network management has a message transmission control status, which allows disabling the transmission of NM messages. For this purpose the function

```
Std_ReturnType NmOsek_DisableCommunication (
   const NetworkHandleType nmChannelHandle )
```

can be called. The transmission of NM messages will be stopped within the next NmOsek_MainFunction() call.

The NM PDU transmission ability is enabled again by the service

```
Std_ReturnType NmOsek_EnableCommunication (
   const NetworkHandleType nmChannelHandle )
```

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

Note
The functions NmOsek_DisableCommunication() and NmOsek_EnableCommunication() are only available if the ‘Com Control Enabled’ feature is activated in the NM Interface configuration.

Note
The function NmOsek_DisableCommunication() transits to the NmPassive, the NmOsek_EnableCommunication() function transits to the NmActive state.
Refer to section 2.3.7 for further details.

2.10.10 NM Coordinator Support

NmOsek channels can be coordinated by the NM Coordinator. Therefore it supports the Remote Sleep Indication interfaces as described in section 2.10.4.

Caution
NmOsek channels can currently only be actively coordinated by NM Coordinator.
The following descriptions always assume that NmOsek is being coordinated actively.

In NmNormal, the coordinated shutdown takes $T_{RingTyp}$ + $T_{WaitBusSleep}$ after the transmission of the NM RING message with Sleep Indication bit set to 1, since the next node will send its NM RING message with Sleep Acknowledge bit set to 1 after $T_{RingTyp}$.

In NmLimpHome, the coordinated shutdown takes $T_{RingMax}$ + $T_{WaitBusSleep}$ after the transmission of the NM LIMPHOME message with Sleep Indication bit set to 1.

However, the setting ‘Generic Bus Nm Shutdown Time’ in Nm requires a static value for the coordinated shutdown duration, regardless of any state in NmOsek (NmNormal or NmLimpHome).

Therefore, NmOsek requires the setting in ‘Generic Bus Nm Shutdown Time’ to be $T_{RingMax}$ + $T_{WaitBusSleep}$ + 2 * $t_{NmCycle}$, where $t_{NmCycle}$ denotes the ‘Main Function Period’ of the Nm module. 2 * $t_{NmCycle}$ is being used as temporal safety interval so that NmOsek_NetworkRelease() will be called shortly before the message transmission. Since the duration of the shutdown procedure in NmOsek depends on whether it is in NmNormal or NmLimpHome, this time has been chosen as the “least common multiple” of both times.

Additionally, NmOsek supports the notification of a Synchronization Point via Nm_SynchronizationPoint(). A Synchronization Point notifies the NM Coordinator that global shutdown for a coordination cluster can be started after the notification if all coordinated channels in the cluster are ready to sleep. Since the latter information is only available in the Nm module, not in the NmOsek module, Nm_SynchronizationPoint() is called cyclically by NmOsek.

More precisely, NmOsek chooses the moment to call Nm_SynchronizationPoint() in a way that NM Interface will call NmOsek_NetworkRelease() shortly before the transmission of its NM RING/LIMPHOME message with the Sleep Indication bit set to 1. This happens

- when the logical ring is stable once during each ring cycle
- when NmOsek is in NmLimpHome once between each transmission of the LIMPHOME message.

The following subsections describe background knowledge about the calculation of the moments when the notification shall occur.

2.10.10.1 Synchronization Point Notification in NmNormal

The notification NmNormal occurs when the logical ring is stable once per ring cycle. One ring cycle shall be defined as the interval between the transmission of two RING messages by the ECU. This interval is also called ‘Ring Cycle Time’.

NmOsek can determine from the configuration how long it takes after a notification of Nm_SynchronizationPoint() that the NM Coordinator calls NmOsek_NetworkRelease(). This interval is called ‘Sync Point Network Release Interval Time’.

The Nm_SynchronizationPoint() function needs to be called once during one whole ring cycle. After it has been called, the function call of NmOsek_NetworkRelease() by Nm shall follow after ‘Sync Point Network Release Interval Time’. After the function call of NmOsek_NetworkRelease(), it always shall take ‘Generic Bus Nm Shutdown Time’[4] until Bus Sleep Mode is entered.
Therefore, Nm_SynchronizationPoint() is called ($T_{RingMax}$ + $t_{NmCycle}$ - $T_{RingTyp}$ + 'Sync Point Network Release Interval Time') % (‘Previous Ring Cycle Time’) before NmOsek sends its own NM message. For the calculation of ‘Previous Ring Cycle Time’, see below.
Note that the next message after the own NM message with Sleep Indication Bit 1 will be the one that triggers the transition to Wait Bus Sleep (RING message with Sleep Acknowledge bit set to 1).
So it needs to be calculated for the moment when Nm_SynchronizationPoint() is called.
The ‘Ring Cycle Time’ is measured for each Ring Cycle by NmOsek.
On the next TX RING, the currently measured ‘Ring Cycle Time’ is used as the ‘Previous Ring Cycle Time’ mentioned above and the Ring Cycle Time measurement is restarted.
That means that the time between each TX Ring is measured and saved as ‘Previous Ring Cycle Time’.
This needs to be calculated at run-time, because the number of nodes in the logical ring is dynamic and the interval between each RING message sent by each ECU may vary.

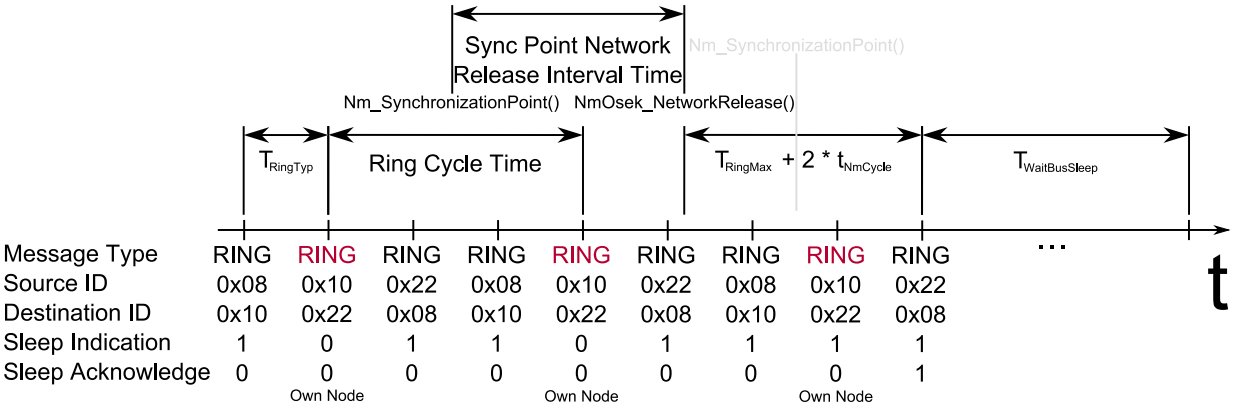

Figure 2-6 Synchronization Point Notification in NmNormal

In Figure 2-6, an example for a logical ring with three participants, nodes 0x08, 0x10 (own node) and 0x22 is provided. In this example, Ring Cycle Time is approximately 3 * $T_{RingTyp}$. Let ‘Sync Point Network Release Interval Time’ be 28 * $t_{NmCycle}$, $T_{RingTyp}$:= 10 * $t_{NmCycle}$, $T_{RingMax}$:= 26 * $t_{NmCycle}$. Since the logical ring shall be stable for some time, the ‘Previous Ring Cycle Time’ shall have the same value as the ‘Ring Cycle Time’, i.e. 3 * $T_{RingTyp}$.

[4] If ‘Wait Bus Sleep Extensions’ (OEM-specific feature, only available if mentioned in [6]) are turned ON and NmOsek is in NmNormal, it shall take ‘Generic Bus Nm Shutdown Time’ – ($T_{ErrorWaitBusSleep}$ – $T_{WaitBusSleep}$)

The moment where Nm_SynchronizationPoint() shall be calculated calculates as ((26 + 1 – 10 + 28) * $t_{NmCycle}$) % (30 * $t_{NmCycle}$) = 15 * $t_{NmCycle}$, i.e. it will be called 15 Nm ‘Main Function Periods’ before the transmission of the RING message in each ring cycle.

2.10.10.2 Synchronization Point Notification in NmLimpHome

The notification of Nm_SynchronizationPoint() in NmLimpHome occurs at a moment, so that the ‘Sync Point Network Release Interval Time’ elapses shortly before the LIMPHOME message transmission. Since there is no Ring Cycle Time in NmLimpHome but the cyclical transmission of the LIMPHOME message instead, the interval of the LIMPHOME message (T_{Error}) is used for modulo calculation. So Nm_SynchronizationPoint() is called once between two LIMPHOME messages, ‘Sync Point Network Release Interval Time’ % T_{Error} before the transmission of the LIMPHOME message.

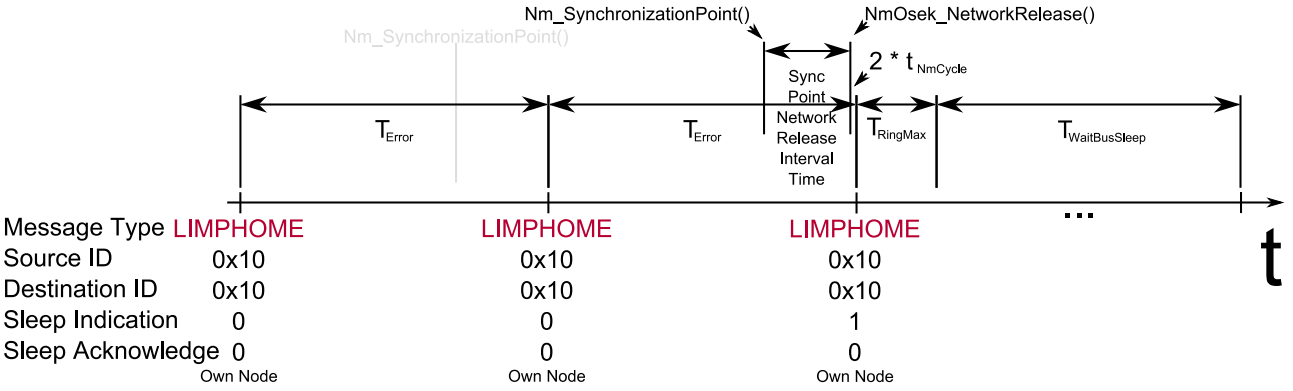

Figure 2-7 Synchronization Point Notification in NmLimpHome

Figure 2-7 depicts an example for NmLimpHome. Let ‘Sync Point Network Release Interval Time’ be 28 * $t_{NmCycle}$, T_{Error} be 100 * $t_{NmCycle}$, TRingMax be 26 * $t_{NmCycle}$. So Nm_SynchronizationPoint() is called (28 * $t_{NmCycle}$) % T_{Error} = 28 * $t_{NmCycle}$ before the LIMPHOME message transmission.

2.10.11 First Message shall be NM Message

NmOsek provides the possibility to ensure that the NM message shall be the first message on the bus when waking up from Bus Sleep.

This can be achieved by also using the Nm State Change Notifications to enable/disable TX I-DPU Groups (i.e. TX Application Messages in Com). If the feature is enabled, State Change Notifications to Repeat Message, Normal Operation or Ready Sleep via Nm_StateChangeNotification() are delayed until a NM message has been successfully transmitted or received. If one of these states shall be entered from Bus Sleep or Prepare Bus Sleep, the current state is changed to Wait Startup instead if neither a NM message has been received nor transmitted.

The TX I-PDU-Group handling (typically in BswM) can then be adapted to turn TX I-PDU-Groups on only if the Nm State is not equal to Bus Sleep, Prepare Bus Sleep, Wait Startup or BusOff (if ‘Bus Off Notification Enabled’ is turned ON).

© 2017 Vector Informatik GmbH

Note
In previous NmOsek versions, if the feature ‘First Message shall be NM Message’ is turned ON in the configuration, the usage of the CanIf setting ‘Pn Wakeup Tx Pdu Filter Support’ was mandatory. This is no longer required for channels with NmOsek as only NM algorithm.

This feature requires the ‘Tx Confirmation Timeout Time’ to be defined (see also chapter 2.8.3).

If message transmission of the NM message does not work for some reason, only NM Interface is being notified by Nm_TxTimeoutException() (also refer to chapter 2.8.3).

If NmOsek shall no longer wait until a successful reception or transmission of the NM message, the service function NmOsek_CancelWaitForTxConfOrRxInd() can be called. Then, state changes to Repeat Message, Normal Operation or Ready Sleep are also permitted.

2.10.12 Handle of the TX Deadline Monitoring by Com

NmOsek provides the possibility to handle the TX Deadline Monitoring for Nm-Pdus via Com. This feature requires 'Com User Data Support' to be turned ON in Nm and the ‘Tx Pdu User Data’ to be defined.

If the feature is active, the ‘Tx Confirmation Timeout Time’ of NmOsek is not considered for message observation (see chapter 2.8.3). Therefore, the parameter NmOsekTxConfirmationTimeoutTime is not considered for Tx-Deadline Monitoring and there will be no function call of Nm_TxTimeoutException().

Instead, a signal or a signal group inside the I-PDU which is referenced by NmOsekTxUserDataPduRef can be used for the timeout notification.

Similar to the sequence provided in chapter 2.8.3, the procedure is as follows:

- OSEK NM issues a function call of Com_TriggerIPDUSend().
- OSEK NM issues a transmission request for a NM message (CanIf_Transmit()).
- The Com module will handle the TX Deadline Monitoring.
- If a message transmission is successful on the bus, Com is informed about the successful transmission via PduR_NmOsekTxConfirmation().
- Shortly before another NM message transmit request is issued, Com_TriggerIPDUSend() is called again.

This feature can only be used if 'Passive Mode Enabled' is turned OFF in Nm and if 'First Message Shall Be Nm Message' (see chapter 2.10.11) is turned OFF in NmOsek.

3 Integration

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

3.1 Scope of Delivery

The delivery of the NmOsek 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	Description
NmOsek.c	■	This is the source file of NmOsek. It contains all implementations of the services provided by the module as well as the whole Network Management algorithm. The user **must not** change this file.
NmOsek.h	■	This is the header file for NmOsek APIs. The user **must not** change this file.
NmOsek_Cbk.h	■	This is the header file for NmOsek APIs. The user **must not** change this file.

Table 3-1 Static files

3.1.2 Dynamic Files

The dynamic files are generated by the configuration tool GENy.

File Name	Description
NmOsek_Cfg.c	This is the pre-compile variant configuration source file. The user **must not change** this file.
NmOsek_PBcfg.c	This is the post-build variant configuration source file. The user **must not change** this file.
NmOsek_Cfg.h	This is the configuration header file. The user **must not change** this file.

Table 3-2 Generated files

3.2 Include Structure

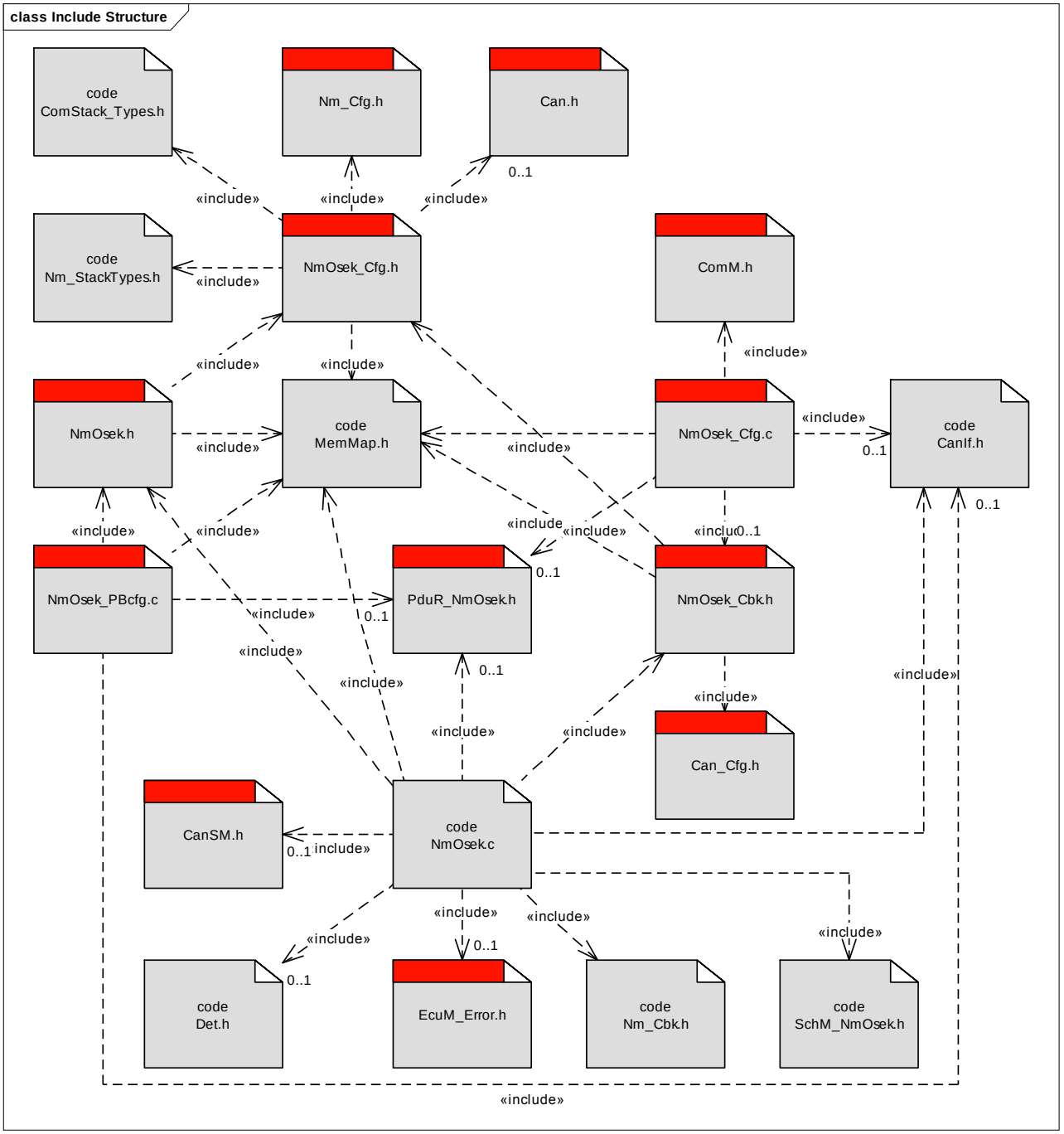

Figure 3-1 Include structure

3.3 Critical Sections

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

Critical sections are supported by using the BSW Scheduler inside the RTE.

NmOsek calls the following functions when entering a critical section i = 0,…,5:

```
void SchM_Enter_NmOsek_NMOSEK_EXCLUSIVE_AREA_i()
```

When the critical section is left the following functions are called by NmOsek:

```
void SchM_Exit_NmOsek_NMOSEK_EXCLUSIVE_AREA_i()
```

The critical section (exclusive area) codes passed to these functions 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 [9].

3.3.1 Critical Section Codes

In case of the BSW Scheduler is used for critical sections NmOsek uses several critical section codes which must lead to corresponding interrupt locks. The mapping is described in the following table:

Critical Section Define	Interrupt Lock
NMOSEK_EXCLUSIVE_AREA_0	No interruption by any interrupt is allowed. Therefore this section must always lock global interrupts.
NMOSEK_EXCLUSIVE_AREA_1	No interruption of NmOsek_NetworkRelease(), NmOsek_DisableCommunication() and NmOsek_EnableCommunication() by NmOsek_MainFunction() allowed. This means that global interrupts have to be used for this section only if NmOsek_NetworkRelease(), NmOsek_DisableCommunication() and NmOsek_EnableCommunication() can be interrupted by the following function: > NmOsek_MainFunction() Otherwise no interrupt locks are necessary.
NMOSEK_EXCLUSIVE_AREA_2	No interruption of NmOsek_PassiveStartUp() by NmOsek_MainFunction() allowed. This means that global interrupts must be locked if NmOsek_PassiveStartUp() can be interrupted by the following function: > NmOsek_MainFunction() Otherwise no interrupt locks are necessary.
NMOSEK_EXCLUSIVE_AREA_3	No interruption of NmOsek_SetUserData() by NmOsek_MainFunction() allowed This means that global interrupts must be locked if NmOsek_SetUserData() can be interrupted by the following functions: > NmOsek_MainFunction() Otherwise no interrupt locks are necessary.
NMOSEK_EXCLUSIVE_AREA_4	No interruption of NmOsek_GetUserData() by NmOsek_RxIndication() or NmOsek_GetPduData() allowed This means that global interrupts must be locked if NmOsek_GetUserData() can be interrupted by the following functions: > NmOsek_RxIndication() > NmOsek_GetPduData() Otherwise no interrupt locks are necessary.

NMOSEK_EXCLUSIVE_AREA_5	No interruption of NmOsek_GetPduData() allowed This means that global interrupts must be locked if NmOsek_GetPduData()can be interrupted by the following functions: > NmOsek_RxIndication() > NmOsek_GetUserData() Otherwise no interrupt locks are necessary.

Table 3-3 Critical Section Codes

4 API Description

For an interfaces overview please see Figure 1-2.

4.1 Type Definitions

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

Type Name	C-Type	Description	Value Range
NmOsek_ActionType	uint8	Type of the internal action state	NMOSEK_NO_ACTION (0x00) NmOsek_MainFunction() call has no effect for the channel
			NMOSEK_ACTION_START (0x01) Transient state that decides whether NmNormal or NmLimpHome should be entered
			NMOSEK_ACTION_WAIT_SENDTOKEN (0x02) Token at own node, wait before sending a Ring message
			NMOSEK_ACTION_WAIT_RINGMSG (0x03) Token at another node, waiting for Ring message
			NMOSEK_ACTION_LH_WAIT_LIMPHOME_TX (0x04) NmRun in NmLimpHome state
			NMOSEK_ACTION_LH_PREPARE_SLEEP (0x05) NmPrepSleep in NmLimpHome state
			NMOSEK_ACTION_GO_BUSSLEEP (0x06) NmWaitBusSleep state
			NMOSEK_ACTION_BUS_SLEEP (0x07) NmBusSleep state

Type Name	C-Type	Description	Value Range
			NMOSEK_ACTION_START_WITH_REPEAT (0x81) Same value as NMOSEK_ACTION_START plus repetition flag (0x80). When this value has been set to a state variable the actions for NMOSEK_ACTION_START are processed within the same call of NmOsek_MainFunction().
			NMOSEK_ACTION_LH_PREPARE_SLEEP_WITH_REPEAT (0x85) Same value as NMOSEK_ACTION_LH_PREPARE_SLEEP plus repetition flag (0x80). When this value has been set to a state variable the actions for NMOSEK_ACTION_LH_PREPARE_SLEEP are processed within the same call of NmOsek_MainFunction().

Table 4-1 Type definitions

NmOsek_NodeConfigType

This is a structure that represents the current network configuration. i.e. a byte array that indicates which nodes in the network are present or absent (for all channels). For details refer to 2.10.2. A pointer to a structure variable of this type is also used in the function NmOsek_GetConfig().

Note that this structure type definition is only available when the ‘Use Nm Node List’ feature is enabled in the NmOsek configuration.

Struct Element Name	C-Type	Description	Value Range
nmConfigChar	uint8[N] with N = (Number of maximum nodes in the network / 8) * (Number of NmOsek channels)	Global state byte	0-255 (for each byte in the array)

Table 4-2 NmOsek_NodeConfigType

4.2 Services Provided by NmOsek

4.2.1 Services Provided due to AUTOSAR Requirements

The following functions are implemented by NmOsek to fulfill the AUTOSAR requirement to implement a ‘Generic BusNm’ as CDD.

4.2.1.1 NmOsek_Init

Prototype	
`void NmOsek_Init(const NmOsek_ConfigType *nmOsekConfigPtr )`	
Parameter	
`nmOsekConfigPtr`	Configuration structure for initializing the module
Return code	
-	-
Functional Description	
Initialize the complete NmOsek module, i.e. all channels which are activated at configuration time are initialized.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function has to be called before any other function (except NmOsek_InitMemory() or NmOsek_GetVersionInfo()) > NmOsek_Cfg.h contains symbolic defines (NmOsek_Config_Ptr / NmOsek_Config_<Predefined Variant Name>_Ptr) that can be used as configuration parameter > This function is non-reentrant. > This function is synchronous.	
Expected Caller Context	
> At system start-up on task level	

Table 4-3 NmOsek_Init

4.2.1.2 NmOsek_PassiveStartUp

Prototype	
`Std_ReturnType NmOsek_PassiveStartUp( const NetworkHandleType nmChannelHandle )`	
Parameter	
`nmChannelHandle`	Identification of the NM-channel
Return code	
`Std_ReturnType`	E_OK - No error E_NOT_OK - Passive startup of network management has failed
Functional Description	
Passive startup of the OSEK NM. It triggers the transition from Bus-Sleep Mode (NmBusSleep) to the Network Mode (NmAwake) in Repeat Message State. This service has no effect if the current state is neither equal to Prepare Bus Sleep Mode nor equal to Bus Sleep Mode. In that case E_NOT_OK is returned.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > Called by NM Interface. > This function is non-reentrant. > This function is asynchronous. > Pre-condition: NmOsek is initialized	

Expected Caller Context
> Can be called from interrupt or task level

Table 4-4 NmOsek_PassiveStartUp

4.2.1.3 NmOsek_NetworkRequest

Prototype	
`Std_ReturnType NmOsek_NetworkRequest( const NetworkHandleType nmChannelHandle )`	
Parameter	
`nmChannelHandle`	Identification of the NM-channel
Return code	
`Std_ReturnType`	E_OK - No error E_NOT_OK - Requesting network has failed
Functional Description	
Request the network since ECU needs to communicate on the bus.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > Called by NM Interface. > This function is non-reentrant. > This function is asynchronous. > Pre-condition: NmOsek is initialized > This function is not available if the ‘Passive Mode Enabled’ feature is activated in the NM Interface configuration.	
Expected Caller Context	
> Can be called from interrupt or task level	

Table 4-5 NmOsek_NetworkRequest

4.2.1.4 NmOsek_NetworkRelease

Prototype	
`Std_ReturnType NmOsek_NetworkRelease( const NetworkHandleType nmChannelHandle )`	
Parameter	
`nmChannelHandle`	Identification of the NM-channel
Return code	
`Std_ReturnType`	E_OK - No error E_NOT_OK - Releasing network has failed
Functional Description	
Release the network since ECU doesn't have to communicate on the bus.	

Particularities and Limitations
> Service ID: see table 'Service IDs' > Called by NM Interface. > This function is non-reentrant. > This function is asynchronous. > Pre-condition: NmOsek is initialized > This function is not available if the ‘Passive Mode Enabled’ feature is activated in the NM Interface configuration.
Expected Caller Context
> Can be called from interrupt or task level

Table 4-6 NmOsek_NetworkRelease

4.2.1.5 NmOsek_DisableCommunication

Prototype	
`Std_ReturnType NmOsek_DisableCommunication ( const NetworkHandleType nmChannelHandle )`	
Parameter	
`nmChannelHandle`	Identification of the NM-channel
Return code	
`Std_ReturnType`	E_OK - No error E_NOT_OK - Disabling the NM PDU transmission ability has failed.
Functional Description	
Disable the NM PDU transmission ability due to an ISO14229 Communication Control (28hex) service.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > Called by NM Interface. > This function is non-reentrant. > This function is asynchronous. > Pre-condition: NmOsek is initialized > This function is only available if the ‘Com Control Enabled’ feature is activated in the NM Interface configuration.	
Expected Caller Context	
> Can be called from interrupt or task level	

Table 4-7 NmOsek_DisableCommunication

4.2.1.6 NmOsek_EnableCommunication

Prototype	
`Std_ReturnType NmOsek_EnableCommunication( const NetworkHandleType nmChannelHandle )`	
Parameter	
`nmChannelHandle`	Identification of the NM-channel

Return code	
`Std_ReturnType`	E_OK - No error E_NOT_OK - Enabling the NM PDU transmission ability has failed.
Functional Description	
Enable the NM PDU transmission ability due to an ISO14229 Communication Control (28hex) service.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > Called by NM Interface. > This function is non-reentrant. > This function is asynchronous. > Pre-condition: NmOsek is initialized > This function is only available if the ‘Com Control Enabled’ feature is activated in the NM Interface configuration.	
Expected Caller Context	
> Can be called from interrupt or task level	

Table 4-8 NmOsek_EnableCommunication

4.2.1.7 NmOsek_SetUserData

Prototype	
`Std_ReturnType NmOsek_SetUserData( const NetworkHandleType nmChannelHandle, const uint8 * const nmUserDataPtr )`	
Parameter	
`nmChannelHandle`	Identification of the NM-channel
`nmUserDataPtr`	Pointer where the user data for the next transmitted NM message shall be copied from
Return code	
`Std_ReturnType`	E_OK - No error E_NOT_OK - Setting user data has failed
Functional Description	
Set user data for NM messages transmitted next on the bus.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > Called by NM Interface. > This function is non-reentrant. > This function is synchronous. > Pre-condition: NmOsek is initialized > This function is only available if the ‘User Data Enabled’ feature is activated and neither the ‘Passive Mode Enabled’ feature nor the ‘Com User Data Support’ feature is activated in the NM Interface configuration.	
Expected Caller Context	
> Can be called from interrupt or task level	

Table 4-9 NmOsek_SetUserData

4.2.1.8 NmOsek_GetUserData

Prototype	
`Std_ReturnType NmOsek_GetUserData( const NetworkHandleType nmChannelHandle, uint8 * const nmUserDataPtr )`	
Parameter	
`nmChannelHandle`	Identification of the NM-channel
`nmUserDataPtr`	Pointer where user data out of the most recently received NM message shall be copied to
Return code	
`Std_ReturnType`	E_OK - No error E_NOT_OK - Getting user data has failed
Functional Description	
Get user data out of the most recently received NM message.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > Called by NM Interface. > This function is non-reentrant. > This function is synchronous. > Pre-condition: NmOsek is initialized > This function is only available if the ‘User Data Enabled’ feature is activated in the NM Interface configuration.	
Expected Caller Context	
> Can be called from interrupt or task level	

Table 4-10 NmOsek_GetUserData

4.2.1.9 NmOsek_Transmit

Prototype	
`Std_ReturnType NmOsek_Transmit( PduIdType NmOsekTxPduId, const PduInfoType *PduInfoPtr )`	
Parameter	
`NmOsekTxPduId`	L-PDU handle of CAN L-PDU to be transmitted. This handle specifies the corresponding CAN L-PDU ID and implicitly the CAN Driver instance as well as the corresponding CAN controller device.
`PduInfoPtr`	Pointer to a structure with CAN L-PDU related data: DLC and pointer to CAN L-SDU buffer
Return code	
`Std_ReturnType`	E_OK - Transmit request has been accepted Function could be called from interrupt level or from task level
Functional Description	
NmOsek_Transmit is implemented as an empty function and shall always return E_OK.	

Particularities and Limitations
> Service ID: see table 'Service IDs' > Called by NM Interface. > This function is reentrant. > This function is synchronous. > This function is only available if the ‘Com User Data Enabled’ feature is activated in the NM Interface configuration.
Expected Caller Context
> Can be called from interrupt or task level

Table 4-11 NmOsek_Transmit

4.2.1.10 NmOsek_GetNodeIdentifier

Prototype
`Std_ReturnType NmOsek_GetNodeIdentifier( const NetworkHandleType nmChannelHandle, uint8 * const nmNodeIdPtr )`

Parameter	
`nmChannelHandle`	Identification of the NM-channel
`nmNodeIdPtr`	Pointer where node identifier out of the most recently received NM PDU shall be copied to
Return code	
`Std_ReturnType`	E_OK - No error E_NOT_OK - Getting the node identifier out of the most recently received NM PDU has failed

Functional Description
Get node identifier of the most recently received NM PDU.

Particularities and Limitations
> Service ID: see table 'Service IDs' > Called by NM Interface. > This function is reentrant. > This function is synchronous. > Pre-condition: NmOsek is initialized > This function is only available if the ‘Node Id Enabled’ feature is activated in the NM Interface configuration.
Expected Caller Context
> Can be called from interrupt or task level

Table 4-12 NmOsek_GetNodeIdentifier

4.2.1.11 NmOsek_GetLocalNodeIdentifier

Prototype
`Std_ReturnType NmOsek_GetLocalNodeIdentifier( const NetworkHandleType nmChannelHandle, uint8 * const nmNodeIdPtr )`

Parameter	
nmChannelHandle	Identification of the NM-channel
nmNodeIdPtr	Pointer where node identifier of the local node shall be copied to
Return code	
Std_ReturnType	E_OK - No error E_NOT_OK - Getting the node identifier of the local node has failed
Functional Description	
Get node identifier configured for the local node.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > Called by NM Interface. > This function is reentrant. > This function is synchronous. > Pre-condition: NmOsek is initialized > This function is only available if the ‘Node Id Enabled’ feature is activated in the NM Interface configuration.	
Expected Caller Context	
> Can be called from interrupt or task level	

Table 4-13 NmOsek_GetLocalNodeIdentifier

4.2.1.12 NmOsek_RepeatMessageRequest

Prototype	
`Std_ReturnType NmOsek_RepeatMessageRequest( const NetworkHandleType nmChannelHandle )`	
Parameter	
nmChannelHandle	Identification of the NM-channel
Return code	
Std_ReturnType	E_NOT_OK - Transition to NMNormal Request (Repeat Message) has failed
Functional Description	
Request state change to Repeat Message (NMNormal) State.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > Called by NM Interface. > This function is reentrant. > This function is asynchronous. > Pre-condition: NmOsek is initialized > This function is only available if the ‘Node Detection Enabled’ feature is activated in the NM Interface configuration. > This function never returns E_OK since the equivalent state for the AUTOSAR ‘Repeat Message State’ that has been chosen for this implementation is the period of time where the logical ring is unstable. A transition to this state cannot be forced by a function call. For details about states please refer to section 2.3. > If there is a need to find out which nodes are available on the network at a certain time one can use the NmOsek_GetConfig() and NmOsek_CmpConfig() functions instead.	

Expected Caller Context
> Can be called from interrupt or task level

Table 4-14 NmOsek_RepeatMessageRequest

4.2.1.13 NmOsek_GetPduData

Prototype	
`Std_ReturnType NmOsek_GetPduData( const NetworkHandleType nmChannelHandle, uint8 * const nmPduDataPtr )`	
Parameter	
`nmChannelHandle`	Identification of the NM-channel
`nmPduDataPtr`	Pointer where NM PDU Data shall be copied to
Return code	
`Std_ReturnType`	E_OK - No error E_NOT_OK - Getting the NM PDU data has failed
Functional Description	
Get the whole PDU data out of the most recently received NM message.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > Called by NM Interface. > This function is non-reentrant. > This function is synchronous. > Pre-condition: NmOsek is initialized > This function is only available if the ‘Node Id Enabled’ feature or the ‘User Data Enabled’ feature is activated in the NM Interface configuration (also both of the features).	
Expected Caller Context	
> Can be called from interrupt or task level	

Table 4-15 NmOsek_GetPduData

4.2.1.14 NmOsek_GetState

Prototype	
`Std_ReturnType NmOsek_GetState ( const NetworkHandleType nmChannelHandle, Nm_StateType * const nmStatePtr, Nm_ModeType * const nmModePtr )`	
Parameter	
`nmChannelHandle`	Identification of the NM-channel
`nmStatePtr`	Pointer where the state of the network management shall be copied to
`nmModePtr`	Pointer where the mode of the network management shall be copied to
Return code	
`Std_ReturnType`	E_OK - No error E_NOT_OK - Getting the NM state has failed

Functional Description
Return the state and the mode of the network management.
Particularities and Limitations
> Service ID: see table 'Service IDs' > Called by NM Interface. > This function is reentrant. > This function is synchronous. > Pre-condition: NmOsek is initialized
Expected Caller Context
> Can be called from interrupt or task level

Table 4-16 NmOsek_GetState

4.2.1.15 NmOsek_GetVersionInfo

Prototype	
`void NmOsek_GetVersionInfo( Std_VersionInfoType *versioninfo )`	
Parameter	
`versioninfo`	Pointer to a variable where the version information of this module shall be stored Function could be called from interrupt level or from task level
Return code	
-	-
Functional Description	
This service returns the version information of this module.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > Called by application. > This function is reentrant. > This function is synchronous. > This function is only available if the ‘Version Info Api’ feature is activated in the NmOsek configuration.	
Expected Caller Context	
> Can be called from interrupt or task level	

Table 4-17 NmOsek_GetVersionInfo

4.2.1.16 NmOsek_RequestBusSynchronization

Prototype	
`Std_ReturnType NmOsek_RequestBusSynchronization( const NetworkHandleType nmChannelHandle )`	
Parameter	
`nmChannelHandle`	Identification of the NM-channel

Return code	
`Std_ReturnType`	E_NOT_OK – Requesting Bus Synchronization has failed
Functional Description	
Request bus synchronization.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > Called by NM Interface. > This function is reentrant. > This function is synchronous. > This function is only available if the ‘Bus Synchronization Enabled’ feature is activated in the NM Interface configuration. > The implementation of this function is empty since the Bus Synchronization feature is not applicable for this implementation and thus E_NOT_OK is always returned	
Expected Caller Context	
> Can be called from interrupt or task level	

Table 4-18 NmOsek_RequestBusSynchronization

4.2.1.17 NmOsek_CheckRemoteSleepIndication

Prototype	
`Std_ReturnType NmOsek_CheckRemoteSleepIndication( const NetworkHandleType nmChannelHandle, boolean * const nmRemoteSleepIndPtr )`	
Parameter	
`nmChannelHandle`	Identification of the NM-channel
`nmRemoteSleepIndPtr`	Pointer where check result of remote sleep indication shall be copied to
Return code	
`Std_ReturnType`	E_OK - No error E_NOT_OK - Checking remote sleep indication has failed
Functional Description	
Check if remote sleep state has been entered or not.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > Called by NM Interface. > This function is reentrant. > This function is synchronous. > Pre-condition: NmOsek is initialized > This function is only available if the ‘Remote Sleep Ind Enabled’ feature is activated in the NM Interface configuration. > The variable behind `nmRemoteSleep` is either set to TRUE if the remote sleep indication state has been entered or to FALSE otherwise.	
Expected Caller Context	
> Can be called from interrupt or task level	

Table 4-19 NmOsek_CheckRemoteSleepIndication

4.2.1.18 NmOsek_MainFunction

Prototype	
`void NmOsek_MainFunction( void )`	
Parameter	
-	-
Return code	
-	-
Functional Description	
Main function of the NmOsek which processes the OSEK NM algorithm. This function is responsible to handle all NmOsek instances.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > Called by SCHM. > This function is non-reentrant. > Pre-condition: NmOsek is initialized	
Expected Caller Context	
> Can be called from task level	

Table 4-20 NmOsek_MainFunction

4.2.1 Additional Services

The following functions may be used by the application.

4.2.1.1 NmOsek_InitMemory

Prototype	
`void NmOsek_InitMemory( void )`	
Parameter	
-	-
Return code	
-	-
Functional Description	
Initialize memory so that expected start values are set. Refer to section 2.2 for further details.	
Particularities and Limitations	
> Service ID: (none) > Called in start-up code. > Has to be called with disabled global interrupts. > This function is non-reentrant.	
Expected Caller Context	
> Can be called on task level	

Table 4-21 NmOsek_InitMemory

4.2.1.2 NmOsek_GetStatus

Prototype	
`Std_ReturnType NmOsek_GetStatus( const NetworkHandleType nmChannelHandle, uint8 * const nmStatusPtr )`	
Parameter	
`nmChannelHandle`	Identification of the NM-channel
`nmStatusPtr`	Pointer where the status byte of the network management shall be copied to
Return code	
`Std_ReturnType`	E_OK - No error E_NOT_OK - Getting the NM state has failed
Functional Description	
Return the global status byte of the network management.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > Called by application. > This function is synchronous. > This function is reentrant. > Pre-condition: NmOsek is initialized	
Expected Caller Context	
> Can be called on task or interrupt level	

Table 4-22 NmOsek_GetStatus

4.2.1.3 NmOsek_GetConfig

Prototype	
`void NmOsek_GetConfig( const NetworkHandleType nmChannelHandle, NmOsek_NodeConfigType *nmDataPtr )`	
Parameter	
`nmChannelHandle`	Identification of the NM-channel
`nmDataPtr`	Pointer where the node configuration array is copied to
Return code	
-	-
Functional Description	
This API copies the current node configuration into an application specific buffer.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > Called by application. > This function is synchronous. > This function is non-reentrant. > Pre-condition: NmOsek is initialized > This function is only available if the ‘Use Nm Node List’ feature is activated in the NmOsek configuration.	

Expected Caller Context
> Can be called on task level

Table 4-23 NmOsek_GetConfig

4.2.1.4 NmOsek_CmpConfig

Prototype	
`boolean NmOsek_CmpConfig( const NetworkHandleType nmChannelHandle, uint8 idx)`	
Parameter	
`nmChannelHandle`	Identification of the NM-channel
`idx`	Index of ConfigMaskTable and TargetConfigTable
Return code	
`boolean`	TRUE Configuration matches the reference FALSE Configuration does not match the reference
Functional Description	
This API compares the current NM node configuration with a reference configuration stored in memory.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > Called by application. > This function is synchronous. > This function is non-reentrant. > Pre-condition: NmOsek is initialized > This function is only available if the ‘Use Nm Node List’ feature is activated in the NmOsek configuration.	
Expected Caller Context	
> Can be called on task level	

Table 4-24 NmOsek_CmpConfig

4.2.1.5 NmOsek_CancelWaitForTxConfOrRxInd

Prototype	
`Std_ReturnType NmOsek_CancelWaitForTxConfOrRxInd ( const NetworkHandleType nmChannelHandle)`	
Parameter	
`nmChannelHandle`	Identification of the NM-channel
Return code	
`boolean`	E_OK – No error E_NOT_OK - The flag to indicate that NmOsek shall no longer wait for a TxConfirmation or RxIndication cannot be set in the current state or incorrect nmChannelHandle

Functional Description
Cancel the process of NmOsek to wait for a message confirmation or message reception Call this function to let NmOsek no longer wait for one of these events to happen: - NM PDU reception - NM PDU confirmation If this function has been called, neither a NM PDU reception nor a NM PDU confirmation needs to happen to prevent state changes to NM_STATE_REPEAT_MESSAGE, NM_STATE_NORMAL_OPERATION or NM_STATE_READY_SLEEP.
Particularities and Limitations
> Service ID: see table 'Service IDs' > Called by application. > This function is synchronous. > This function is reentrant. > Pre-condition: NmOsek is initialized > This function is only available if the ‘First Message Shall Be Nm Message’ feature is activated in the NmOsek configuration.
Expected Caller Context
> Can be called on task or interrupt level

Table 4-25 NmOsek_CancelWaitForTxConfOrRxInd

4.3 Services Used by NmOsek

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

Component	API
CANIF	CanIf_Transmit[5]
COM	Com_TriggerIPDUSend[6]
DET	Det_ReportError[7]
ECUM	EcuM_BswErrorHook[8]
NM	Nm_NetworkStartIndication
	Nm_NetworkMode
	Nm_PrepareBusSleepMode
	Nm_BusSleepMode
	Nm_StateChangeNotification[9]
	Nm_PduRxIndication[10]
	Nm_NmOsek_PduRxIndication[11]

[5] Only used if ‘Passive Mode Enabled’ is turned OFF in the NM Interface configuration.
[6] Only used if ‘Tx Deadline Monitoring in Com Enabled’ is turned ON
[7] Only used if ‘Dev Error Detect’ is turned ON.
[8] Only used if NmOsek is configured in the Configuration Variant VARIANT-POST-BUILD-LOADABLE (also possible in combination with Variant Support).
[9] Only used if the feature ‘State Change Ind Enabled’ is activated in the NM Interface configuration.
[10] Only used if the feature ‘Bus Nm Specific Pdu Rx Indication Enabled’ is deactivated and the feature ’Pdu Rx Indication Enabled’ is activated in the NM Interface configuration.

Component	API
	Nm_RemoteSleepIndication[12]
	Nm_RemoteSleepCancellation[12]
	Nm_SynchronizationPoint[13]
	Nm_TxTimeoutException[14]
PDUR	PduR_NmOsekTriggerTransmit[15]
	PduR_NmOsekTxConfirmation[15]
SCHM (RTE)	SchM_Enter_NmOsek_NMOSEK_EXCLUSIVE_AREA_i with i = 0,…,5
	SchM_Exit_NmOsek_NMOSEK_EXCLUSIVE_AREA_i with i = 0,…,5

Table 4-26 Services used by the NmOsek

4.4 Callback Functions

This chapter describes the callback functions that are implemented by the NmOsek and can be invoked by other modules.

4.4.1 Callback Functions That Exist Due To AUTOSAR Requirements

The following callback functions are called by CANIF.

4.4.1.1 NmOsek_TxConfirmation

Prototype	
`void NmOsek_TxConfirmation( PduIdType nmOsekTxPduId )`	
Parameter	
`nmOsekTxPduId`	CANIF Tx PDU ID of the transmitted NmOsek PDU
Return code	
-	-
Functional Description	
This service confirms a previous successfully processed CAN transmit request.	

[11] Only used if the feature ‘Bus Nm Specific Pdu Rx Indication Enabled’ is activated in the NM Interface configuration.
[12] Only used if the feature ‘Remote Sleep Ind Enabled’ is activated in the NM Interface configuration.
[13] Only used if the feature ‘Coordinator Support Enabled’ is activated in the NM Interface configuration and the ‘Synchronizing Network’ setting is turned ON for at least one NmChannelConfig container that belongs to a NmOsek channel
[14] Only used if the parameter ‘Tx Confirmation Timeout Time’ is defined and ‘Passive Mode Enabled’ is turned OFF in the NM Interface configuration.
[15] Only used if ‘User Data Tx Pdu’s are configured and ‘Com User Data Support’ is turned ON in the NM Interface configuration

Particularities and Limitations
> Service ID: see table 'Service IDs' > Called by CAN Interface. > This function is synchronous. > This function is reentrant (but not for the same channel determined by nmOsekTxPduId). > Pre-condition: NmOsek is initialized > This function is not available if the ‘Passive Mode Enabled’ feature is activated in the NM Interface configuration.
Expected Caller Context
> Can be called on task or interrupt level

Table 4-27 NmOsek_TxConfirmation

4.4.1.2 NmOsek_RxIndication

Prototype	
`void NmOsek_RxIndication( PduIdType nmOsekRxPduId, const uint8 *canSduPtr, const Can_IdType canId )`	
Parameter	
`nmOsekRxPduId`	Identification of the network through PDU-ID
`canSduPtr`	Pointer to received SDU
`canId`	CAN Identifier of the received SDU
Return code	
-	-
Functional Description	
This service indicates a successful reception of a received NM message to the NmOsek after passing all filters and validation checks.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > Called by CAN Interface. > This function is synchronous. > This function is non-reentrant. > Pre-condition: NmOsek is initialized	
Expected Caller Context	
> Can be called on task or interrupt level	

Table 4-28 NmOsek_RxIndication

4.4.2 Additional Callback Functions

The following callback functions have been added to support transitions to/from LimpHome due to notifications about BusOff occurrences/recoveries. These callback functions are implemented by NmOsek.

4.4.2.1 NmOsek_CanSM_BusOffBegin

Prototype	
`void NmOsek_CanSM_BusOffBegin ( NetworkHandleType NetworkHandle, uint8 *OnlineDelayCyclesPtr )`	
Parameter	
NetworkHandle	system channel index
OnlineDelayCyclesPtr	Pointer to the location where the number of delay cycles shall be stored. (ignored)
Return code	
-	-
Functional Description	
BusOff occurrence notification callback function.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > Called by CANSM if configured as ‘Bus Off Begin’ function in the CANSM configuration. Can also be called by a CDD. > This function is asynchronous. > This function is reentrant. > Pre-condition: NmOsek is initialized > This function is only available if the ‘BusOff Notification Enabled’ feature is activated in the NmOsek configuration. > If it is called by CANSM, the function prototype is provided by CanSM. > Refer to the CANSM Technical Reference [8] for more information about the BusOff notification feature.	
Expected Caller Context	
> Can be called on task or interrupt level	

Table 4-29 NmOsek_CanSM_BusOffBegin

4.4.2.2 NmOsek_CanSM_BusOffEnd

Prototype	
`void NmOsek_CanSM_BusOffEnd( NetworkHandleType NetworkHandle )`	
Parameter	
NetworkHandle	system channel index
Return code	
-	-
Functional Description	
BusOff recovery notification callback function.	

Particularities and Limitations
> Service ID: see table 'Service IDs' > Called by CANSM if configured as ‘Bus Off End’ function in the CANSM configuration. Can also be called by a CDD. > This function is asynchronous. > This function is reentrant. > Pre-condition: NmOsek is initialized > This function is only available if the ‘BusOff Notification Enabled’ feature is activated in the NmOsek configuration. > If it is called by CANSM, the function prototype is provided by CanSM. > Refer to the CANSM Technical Reference [8] for more information about the BusOff notification feature.
Expected Caller Context
> Can be called on task or interrupt level

Table 4-30 NmOsek_CanSM_BusOffEnd

5 AUTOSAR Standard Compliance

5.1 Deviations

There are no known deviations to the AUTOSAR Standard (when NmOsek is regarded as the CanNm module and APIs similar to those of the CanNm) (so called Generic BusNm concept).

5.2 Additions/ Extensions

5.2.1 Providing Advanced Status Information

In addition to the AUTOSAR-like function NmOsek_GetState() the function NmOsek_GetStatus() can be used for retrieval of internal status information.

Refer to section 2.10.3 for more information about this feature.

5.2.2 Controlling the Transition to the Bus Sleep State and Vice Versa

The OSEK NM algorithm is used for controlling these transitions instead of an AUTOSAR standard algorithm (like the algorithm of CanNm).

Refer to section 2.7 for details.

5.2.3 Determination and Monitoring of the Network Configuration

The network configuration, i.e. a list of the nodes that are participating inside the network, can be determined by using the functions NmOsek_GetConfig() and NmOsek_CmpConfig().

Refer to section for 2.10.2 further details.

5.3 Limitations

Since NmOsek is a Complex Device Driver there are no limitations.

6 Glossary and Abbreviations

6.1 Glossary

Term	Description
BusNm	A generic term for lower layer components of the AUTOSAR NM module that implement bus-specific NM behavior
DaVinci Configurator	Generation tool for MICROSAR components

Table 6-1 Glossary

6.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basic Software
CANIF	AUTOSAR CAN Interface
CanNm	AUTOSAR CAN Network Management
CANSM	AUTOSAR CAN State Manager
CDD	Complex Device Driver
DEM	Diagnostic Event Manager
DET	Development Error Tracer
ECU	Electronic Control Unit
FrNm	AUTOSAR FlexRay Network Management
Gw	Gateway
HIS	Hersteller Initiative Software
ID	Identifier
ISR	Interrupt Service Routine
J1939Nm	AUTOSAR SAE J1939 Network Management
LinNm	AUTOSAR LIN Network Management
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
Msg	Message
NM	Network Management
Nm	AUTOSAR Network Management Interface
NmOsek	Network Management Osek (this component)
OEM	Original Equipment Manufacturer
OSEK	Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug (Open Systems and the Corresponding Interfaces for

	Automotive Electronics.)
PDU	Protocol Data Unit
RTE	AUTOSAR Runtime Environment
SCHM	AUTOSAR Schedule Manager
SWS	Software Specification
UdpNm	AUTOSAR UDP Network Management

Table 6-2 Abbreviations

7 Contact

Visit our website for more information on

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

www.vector.com