VECTOR

MICROSAR UDP Network Management

Technical Reference

AUTOSAR 4.1.1

Version 3.01.00

Authors	Mark Harsch, Markus Schuster
Status	Released

Document Information

History

Author	Date	Version	Remarks
Marc Weber	2012-04-04	1.00.00	Creation of the document
Mark Harsch	2013-10-31	2.00.00	Rework due to implementation according to AUTOSAR 4.1.1
Mark Harsch	2014-02-17	2.01.00	Partial Networking support introduced
Mark Harsch	2015-11-02	2.01.01	ESCAN00077350: AR3-2679: Description BCD-coded return-value of XXX_GetVersionInfo() in TechRef
Mark Harsch	2015-12-22	3.00.00	▶ ESCAN00080876: Rename Technical Reference according to SPEC-63403 ▶ Minor editorial changes ▶ ESCAN00086902: Chapter 3.4 'Passive Mode' now states that Repeat Message Time > zero is also allowed for passive configurations
Frederik Dornemann	2016-02-05	3.00.01	▶ ESCAN00088116: Minor editorial changes (review findings) ▶ ESCAN00088119: Updated Technical Reference to new Logo and Layout
Markus Schuster	2016-11-22	3.01.00	▶ FEATC-324 Added chapter 3.17.3.1 ▶ FEATC-59 Extended chapter 3.8 ▶ FEATC-63 Added chapter 3.16

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_SWS_UDPNetworkManagement.pdf	V3.0.0
[2]	AUTOSAR	AUTOSAR_SWS_DevelopmentErrorTracer.pdf	V3.2.0
[3]	AUTOSAR	AUTOSAR_SWS_DiagnosticEventManager.pdf	V4.2.0
[4]	AUTOSAR	AUTOSAR_SWS_RTE.pdf	V3.2.0
[5]	AUTOSAR	AUTOSAR_TR_BSWModuleList.pdf	V1.0.0
[6]	AUTOSAR	AUTOSAR_SWS_PDURouter.pdf	V3.2.0
[7]	AUTOSAR	AUTOSAR_SWS_SocketAdaptor.pdf	V2.0.0
[8]	Vector	TechnicalReference_Asr_Nm.pdf	V3.0.0

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

Contents

Illustrations

Tables

1 Component History

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

Component Version	New Features
01.00.xx	Implementation according to the AUTOSAR SWS for the UDP Network Management module
02.00.xx	Implementation according to the AUTOSAR 4.1.1 SWS for the UDP Network Management module
02.01.xx	Support of Partial Networking
03.00.xx	AUTOSAR RfC 70333 – PDU length is allowed to be greater than 8 Byte
03.01.xx	Support Selective Shutdown of Partial Networking Call of UdpNm_SetSleepReadyBit() triggers NM message transmission

Table 1-1 Component history

2 Introduction

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

Supported AUTOSAR Release*:	4.1.1	
Supported Configuration Variants:	pre-compile	
Vendor ID:	UDPNM_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	UDPNM_MODULE_ID	33 decimal (according to ref. [5])

* For the precise AUTOSAR Release 4.1.1 please see the release specific documentation.

2.1 Architecture Overview

The AUTOSAR Network Management consists of the general NM Interface and the bus-specific NM modules. The UDP Network Management (UdpNm) module implements the network management functionality for the Ethernet. Network management frames are sent as UDP packets.

The purpose of the UdpNm is to detect whether an ECU, connected to the Ethernet channel, requires bus communication or whether all ECUs are ready to go into sleep mode.

The following figure shows where the UdpNm is located in the MICROSAR architecture.

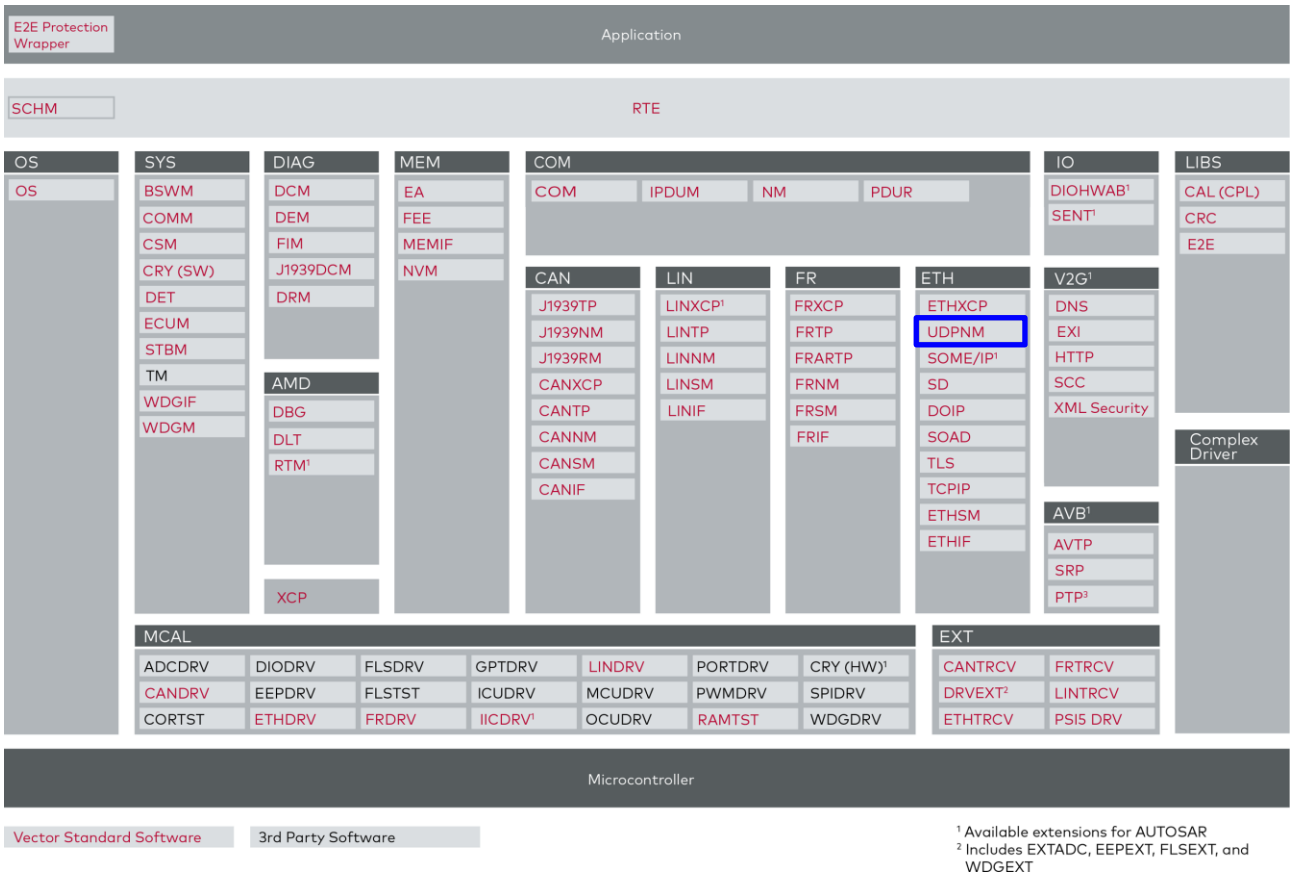

Figure 2-1 AUTOSAR 4.1 Architecture Overview

2.2 Adjacent Modules

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

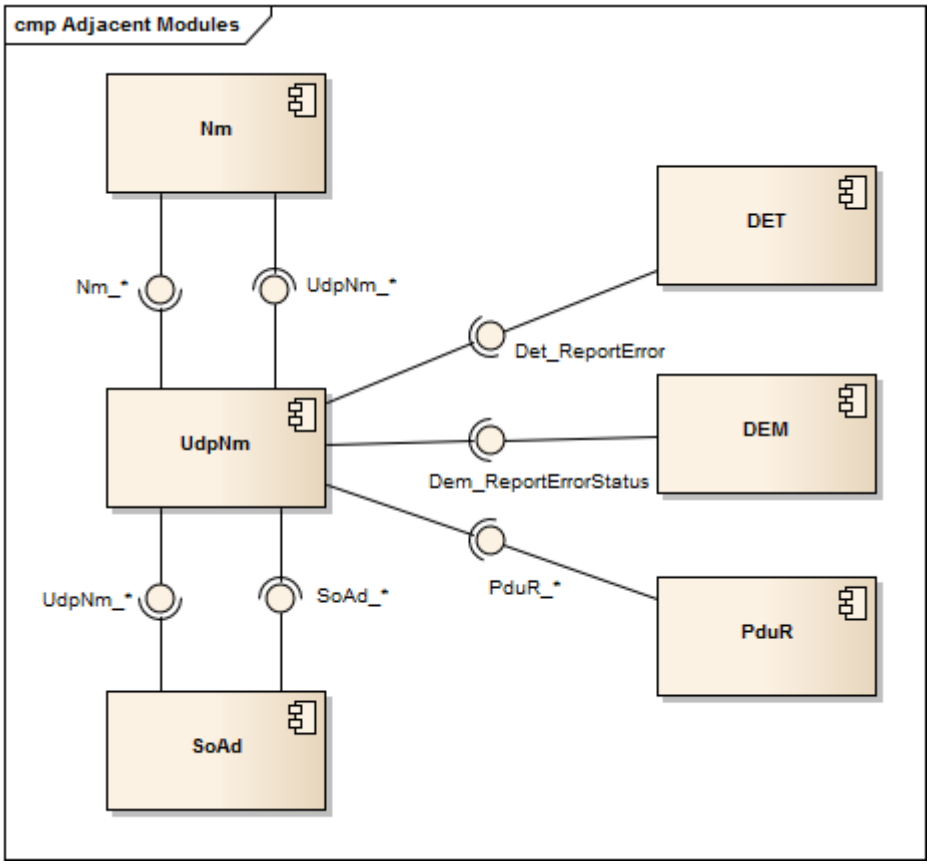

Figure 2-2 Interfaces to adjacent modules of the UdpNm [1]

Applications do not access the services of the BSW modules directly. They use the service ports provided by the BSW modules via RTE. Since the UdpNm has no service ports, the UdpNm can't be accessed via RTE by the application. The UdpNm interacts with the generic Network Management (NM) module.

3 Functional Description

The functional description is limited to the scope of the UdpNm module. If further information for features concerning NM algorithm or involving the NM Interface is needed please refer to the related Technical Reference or the AUTOSAR SWSs.

3.1 Features

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

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

> Table 3-1 Supported AUTOSAR standard conform features

> Table 3-2 Not supported AUTOSAR standard conform features

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

> Table 3-3 Features provided beyond the AUTOSAR standard

The following features specified in [1] are supported:

Supported AUTOSAR Standard Conform Features
User Data Handling
Node Detection
Remote Sleep Indication
Coordinator Synchronization Support
Bus Synchronization
Passive Mode Support
PDU Rx Indication
State Change Indication
Repeat Message Indication
Com User Data Support
Communication Control
Partial Networking
Retrieve version info
Report development errors to DET
Report production errors to DEM
Immediate NM Transmissions

Table 3-1 Supported AUTOSAR standard conform features

3.1.1 Deviations Against AUTOSAR 4.1.1

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

Not Supported AUTOSAR Standard Conform Features

Table 3-2 Not supported AUTOSAR standard conform features

3.1.2 Additions/ Extensions

The following features are provided beyond the AUTOSAR standard:

Features Provided Beyond The AUTOSAR Standard
AUTOSAR RfC 70333 – PDU length is allowed to be greater than 8 Byte

Table 3-3 Features provided beyond the AUTOSAR standard

3.1.3 Limitations

For description of Limitations please refer [1].

3.2 Initialization

The UdpNm is initialized by calling the `UdpNm_InitMemory()` and `UdpNm_Init()` functions. `UdpNm_InitMemory()` must only be called, if the compiler or startup code does not initialize variables correctly. The configuration of the module is pre-defined by DaVinci Configurator Pro during the configuration process.

3.3 Main Functions

For each Ethernet channel, on which the UdpNm is active, there is one separate main function - `UdpNm_MainFunction_x()`. The ‘x’ is replaced by the UdpNm channel configuration container name, i.e. `UdpNm_MainFunction_UdpNmChannelConfig()` is the main function for the UdpNm channel with the configuration container name ‘UdpNmChannelConfig’. All main functions must be called periodically by the BSW Scheduler Module (SchM). The period itself is configured channel specific during the configuration process.

3.4 States

The UdpNm is operational after correct initialization. The internal states of the module are shown in Figure 3-1.

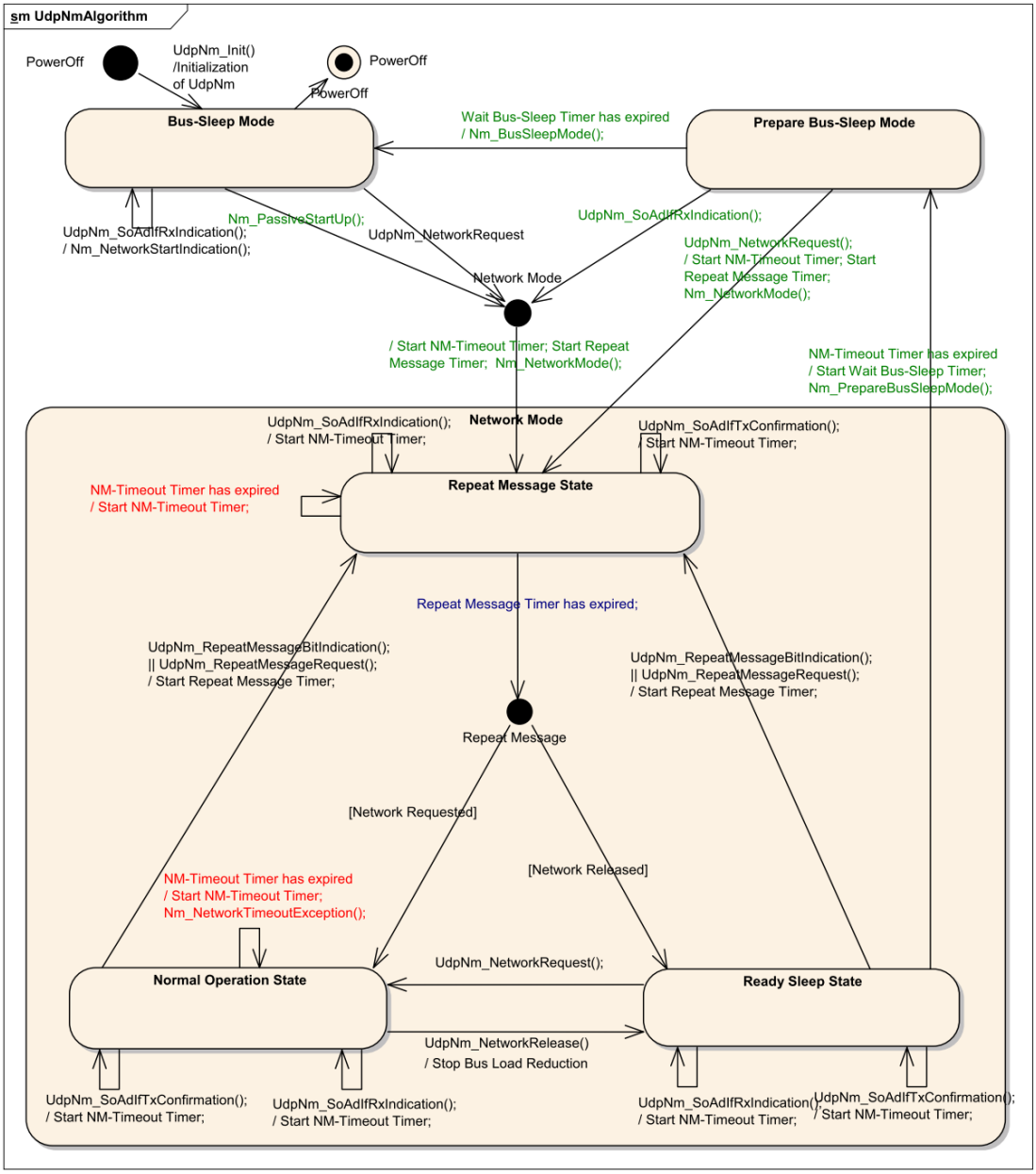

Figure 3-1 State Chart Diagram [1]

3.5 NM PDU structure

3.5.1 Payload

The payload of the NM PDU is structured as followed.

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 0	Source Node Identifier (optional)							
Byte 1	Control Bit Vector (optional)							
Byte 2	User Data 0							
Byte 3	User Data 1							
Byte 4	User Data 2							
Byte 5	User Data 3							
…	…							
Byte n	User Data n-2							

Table 3-4 NM PDU Payload Structure

It is possible to influence the payload structure channel specific.

The location of the Source Node Identifier and the Control Bit Vector can be changed. Either they can be switched or turned off but it is only allowed to locate theme within the first two byte of the payload.

3.5.2 Control Bit Vector (CBV)

The Control Bit Vector (CBV) is a bit field of size one byte. It contains multiple indication Bits to influence the behavior of the Nodes receiving the NM PDU.

Its structure is shown in the following table.

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 0	Res	PNI	Res	Res	NM Coordinator Sleep Ready	Res	Res	Repeat Message Request

Table 3-5 Control Bit Vector (CBV) Structure

Repeat Message Request

> 0: Repeat Message State not requested

> 1: Repeat Message State requested

NM Coordinator Sleep Ready

> 0: Start of synchronized shutdown is not requested by main coordinator

> 1: Start of synchronized shutdown is requested by main coordinator

Partial Network Information (PNI)

> 0: NM message contains no Partial Network request information

> 1: NM message contains Partial Network request information

Reserved (Res)

> 0: Reserved for future usage

3.6 User Data Handling

The User Data Handling allows manipulating and/or retrieving the data contained in the User Data field of an NM PDU. Access to the data is limited channel-wise.

Note
Feature is enabled in DaVinci Configurator Pro by setting **[User Data Enabled]**.

Manipulating the User Data to be transmitted

The API `UdpNm_SetUserData()` allows to set the data to be transmitted within the User Data field of the NM PDU.

Manipulation of the data in the User Data field is only possible if at the same time **[Com User Data Support]** is disabled. For further information on Com User Data Support please refer to section 3.14.

Retrieving the User Data received

The API `UdpNm_GetUserData()` allows to retrieve the data in the User Data field of last received NM PDU.

3.7 Remote Sleep Indication

The Remote Sleep Indication feature allows the node to detect whether it's the only one still requesting the bus and indicate the NM Interface about the circumstance.

Note
Feature is enabled by setting **[Remote Sleep Indication Enabled]** in DaVinci Configurator Pro.
The feature is required on **Gateway nodes only**.

Detection of Remote Sleep Indication

Remote Sleep Indication is detected during `Normal Operation State` by the lack of incoming NM PDUs sent from foreign nodes.

Detection is realized through a timer which is decremented if no NM PDU is received. If the timer exceeds, the Remote Sleep Indication is recognized.

The NM Interface is informed about the Remote Sleep Indication by a call to `Nm_RemoteSleepIndication()`.

Detection of Remote Sleep Cancellation

The Remote Sleep Indication passed to the NM Interface can be cancelled if a foreign node requests the bus again.

This cancellation is only accepted when receiving an NM PDU in `Normal Operation State` or `Ready Sleep State`. The NM Interface will be informed about another node requesting the bus again by calling `Nm_RemoteSleepCancelation()`.

3.8 Coordination Synchronization Support

The Coordination Synchronization Support feature allows the synchronization of the shutdown between multiple coordinators. Therefore the main coordinator sets the `NM Coordinator Sleep Ready` Bit in the CBV to indicate the demand for a synchronized shutdown.

Note
Feature is enabled by setting **[Coordinator Sync Support]** in DaVinci Configurator Pro.

Initiation of Coordinator Synchronous Shutdown

The Coordinator Synchronization is initiated by the main coordinator by calling `UdpNm_SetSleepReadyBit()`. This API will set the related Bit in the CBV of the NM PDU. In addition an asynchronous NM message is transmitted immediately in order to propagate the change as soon as possible.

Detection and Cancellation of Coordinator Synchronous Shutdown

The Coordinator Synchronous Shutdown request is detected by receiving a NM PDU with the `NM Coordinator Sleep Ready` Bit set in the CBV. This will lead to a call to `Nm_CoordReadyToSleepIndication()`.

This indication is recalled if a NM PDU is received with the `NM Coordinator Sleep Ready` Bit cleared again. The NM Interface is notified by `Nm_CoordReadyToSleepCancellation()`.

3.9 Bus Synchronization

The Bus Synchronization feature allows the node to send a NM PDU asynchronously by a call to `UdpNm_RequestBusSynchronization()`.

Nodes receiving this PDU will reset their NM Timeout Timer which leads to a synchronization of the time base for the transition to `Ready Sleep Mode`.

Note
Feature is enabled by setting **[Bus Synchronization Enabled]** in DaVinci Configurator Pro.

3.10 Passive Mode

The Passive Mode feature allows to restrict the node to only receiving NM PDUs. Transmission of PDUs is disabled in all states of the state machine any time.

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

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

Note
Feature is enabled by setting **[Passive Mode Enabled]** in DaVinci Configurator Pro.

3.11 PDU Rx Indication

The PDU Rx Indication feature allows notifying the NM Interface of an NM PDU reception.

If a NM PDU is received the UdpNm will call `Nm_PduRxIndication()` to indicate the reception to the NM Interface.

Note
Feature is enabled by setting **[PDU Rx Indication Enabled]** in DaVinci Configurator Pro.

3.12 State Change Indication

The State Change Indication allows the module to indicate every state change in its state machine during operation. Nm Interface will be notified by calling `Nm_StateChangeNotification()`.

Note
Feature is enabled by setting **[State Change Indication Enabled]** in DaVinci Configurator Pro.

3.13 Repeat Message Indication

The Repeat Message Indication feature enables the UdpNm to notify the Nm Interface about the reception of a NM PDU with the `Repeat Message Request` Bit set in the CBV. The notification is passed by calling the `Nm_RepeateMessageIndication()` API of Nm Interface.

Note
Feature is enabled by setting **[Repeat Message Indication Enabled]** in DaVinci Configurator Pro.

3.14 Com User Data Support

The Com User Data Support feature enables the module to retrieve the data to be placed in the User Data field of the NM PDU automatically.

On transmission of a NM PDU the UdpNm collects the User Data to be transmitted within the PDU from the PduR. Therefore it calls the `PduR_UdpNmTriggerTransmit()` API of PduR, which provides the corresponding data.

Note
Feature is enabled by setting **[Com User Data Support]** in DaVinci Configurator Pro. If this feature is enabled the common User Data Support feature can't be used to set the data in the User Data field of the NM PDU. Only the API to retrieve the last received data is available.

3.15 Communication Control

The Communication Control feature allows influencing the transmission of NM PDUs during runtime.

Transmission of NM PDUs is stopped by calling the API `UdpNm_DisableCommunication()`. As result there will be no PDUs transmitted in any state.

To enable the transmission again the API `UdpNm_EnableCommunication()` is provided.

Note
Feature is enabled by setting **[Com Control Enabled]** in DaVinci Configurator Pro.

3.16 Immediate Nm Transmissions

If an Active Wake-up occurs the UDP NM transmits the first NM message immediately (the NM message offset time is ignored) when entering Repeat Message State. For the next NM messages UDP NM uses a faster NM message cycle time. Afterwards it uses the normal NM message cycle time. This behavior is illustrated in Figure 3-2.

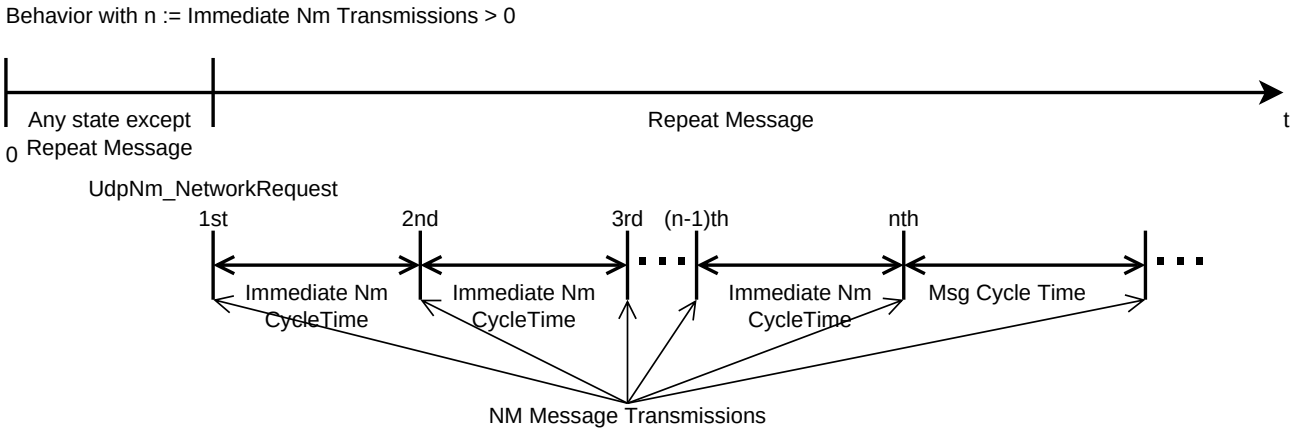

Figure 3-2 Immediate Nm Transmissions

The number of 'Immediate Nm Transmissions' is the number that is configured for this parameter. As it can be seen in Figure 3-2, after the first Immediate Nm Transmission the interval between the NM messages is 'Immediate Nm CycleTime' for *(n-1)* times. Then, the usual interval 'Msg Cycle Time' is used again.

Note that “Any state except Repeat Message” in Figure 3-2 refers to ‘Bus Sleep’ and ‘Prepare Bus Sleep’. If the setting ‘Pn Handle Multiple Network Requests’ is ON, it also refers to ‘Ready Sleep’ and ‘Normal Operation’.

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

Note
This feature should not be confused with the possibility for an immediate transmission if the ‘Com User Data Support’ feature is on.

Note
If the send request of an ‘immediate transmission’ is rejected by the lower layer (e.g. SoAd), the rejected send request is not considered as ‘immediate transmission’. It means that the counter that counts the number of ‘immediate transmissions’ *ImmediateNmMsgCount* is not decremented.

Example: Let ‘Immediate Nm Transmissions’ := 2. The initial counter value of *ImmediateNmMsgCount* is 1.

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

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

3.17 Partial Networking

The Partial Networking feature enables the ability to handle NM PDUs containing information about Partial Network (PN) requests. Furthermore the states are aggregated into the so called External-Internal-Request-Aggregation- and External-Request-Aggregation-PDUs, or cut short EIRA- and ERA-PDUs. These PDUs are shared with the upper layer to provide the PN states.

3.17.1 Partial Networking Information in NM PDU

The PN Information is places into the regular NM PDU. More precisely the information is contained in the User Data field.

Table 3-6 shows an example where 2 Byte of Partial Networking Information are contained in the User Data field. Furthermore, as also shown in the figure, the CBV isn't optional anymore. The PNI Bit contained in the CBV is mandatory for Partial Networking and set if PN Information is contained in the NM PDU.

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 0	Source Node Identifier (optional)							
Byte 1	Control Bit Vector							
Byte 2	User Data 0 (PN Information 0)							
Byte 3	User Data 1 (PN Information 1)							
Byte 4	User Data 2							
Byte 5	User Data 3							
…	…							
Byte n	User Data n-2							

Table 3-6 NM PDU Layout Structure with Partial Networking Information

The PN Information itself describes if a PN is requested. Every Bit in the PN Information stands for a request of the related PN. Table 3-7 shows an example of one Byte of PN Information. As shown the PNs 0, 3 and 6 are requested.

	PN7	PN6	PN5	PN4	PN3	PN2	PN1	PN0
Byte 0	released	requested	released	released	requested	released	released	requested

Table 3-7 Example of Partial Networking Information within User Data Field

3.17.2 Partial Network States

Every state change (transition from PN Released to PN Requested and vice versa) is indicated to the PduR by calling `PduR_UdpNmRxIndication()`. The need for a call to the Rx Indication is aggregated within one processing cycle of the PN Information in order to reduce the number of calls to at most one per cycle. Received Partial Networking requests are only processed if they pass the preconfigured PN Filter Mask. Refer to chapter 3.17.3 for further information.

Figure 3-3 shows the state machine for the PN states.

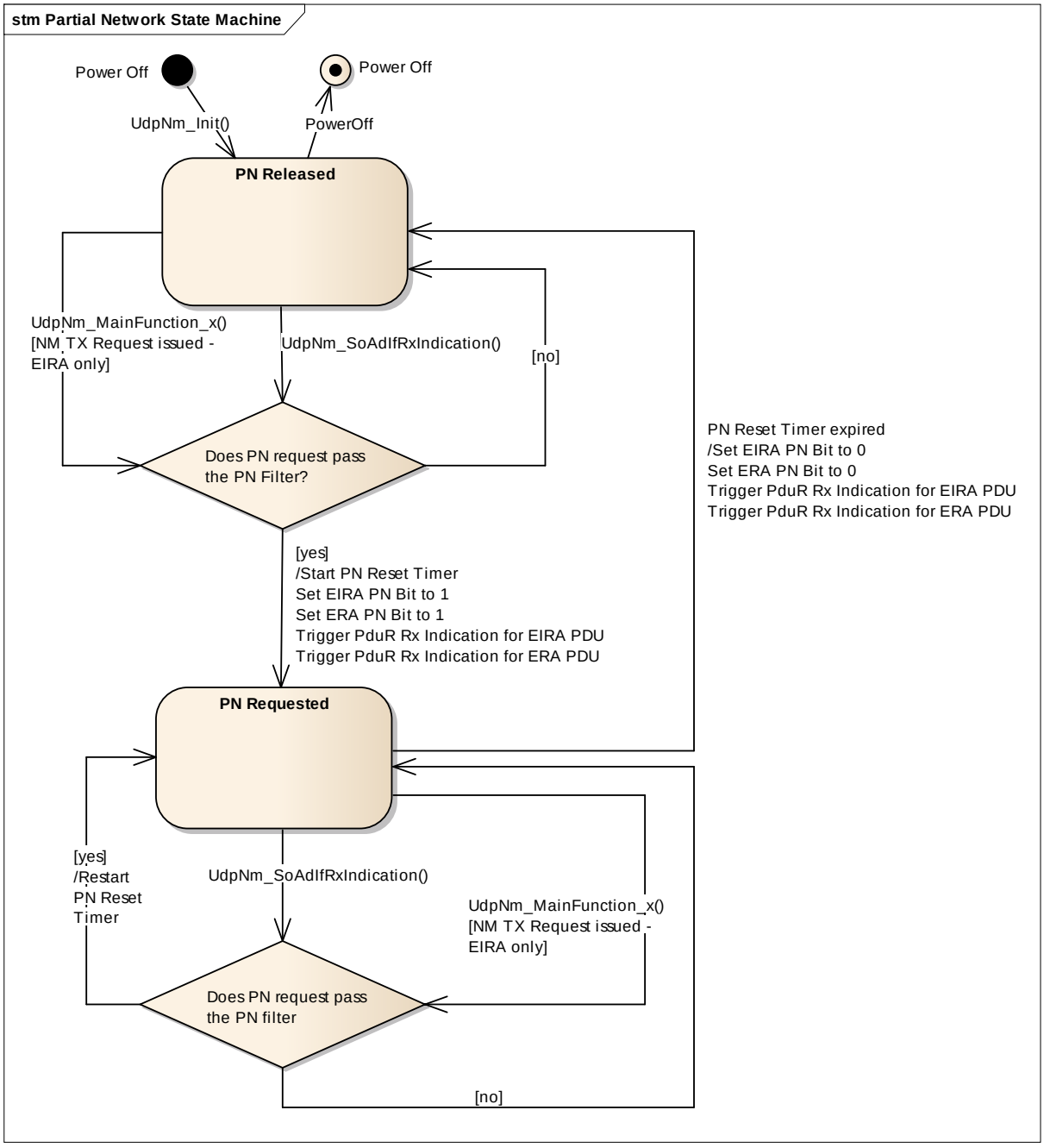

Figure 3-3 Partial Network state machine

EIRA

The EIRA-PDU is used to aggregate internal and external PN requests respectively states. These states are influenced by the transmission and reception of NM PDUs containing PN Information and an additional timeout releasing the PN if it isn't requested for a certain time anymore.

The scope of the EIRA-PDU is global. It collects the PN Information from all channels running Partial Networking.

Internal requests for PNs are contained in the User Data Tx PDU. The external requests are collected from the common Rx NM PDUs.

Note
EIRA calculation feature is enabled by setting **[PN EIRA Calc Enabled]** in DaVinci Configurator Pro.

ERA

The ERA-PDU is used to aggregate only external PN requests respectively states. States are influenced by the reception of NM PDUs containing PN Information and an additional timeout releasing the PN if it was not requested for a certain time.

The scope of the ERA-PDU is a channel scope. Therefor every channel enabling Partial Networking has its own ERA-PDU. This scope enables the upper layer to retrieve which channel requests a PN.
The requests are retrieved from the common Rx NM PDUs of the channel.

Note
ERA calculation feature is enabled channel-wise by setting **[PN ERA Calc Enabled]** in DaVinci Configurator Pro.

3.17.3 Partial Networking Information Filter

It is possible to filter not relevant PN Information. Therefore a filter mask is introduced and laid over the PN Information contained in the NM PDUs.
Table 3-8 shows an example of the filter mask processing. PN0 and PN3 are requested but not relevant for the node. The filter mask filters out the information and as a result only PN6 is recognized as requested.

	PN7	PN6	PN5	PN4	PN3	PN2	PN1	PN0
Byte 0	released	requested	released	released	requested	released	released	requested

	PN7	PN6	PN5	PN4	PN3	PN2	PN1	PN0
Filter Mask 0	not relevant	relevant	relevant	not relevant	not relevant	relevant	not relevant	not relevant
Result	not relevant	requested	released	not relevant	not relevant	released	not relevant	not relevant

Table 3-8 PN Information filtering

Note
PN Information filtering is enabled by configuring the **[PN Filter Mask Byte]** containers in DaVinci Configurator Pro.

3.17.3.1 Selective Shutdown

Selective shutdown describes the possibility to shut down channels and/or ECUs in case they have no need to participate in the current communication. The filtering is done according to description in chapter 3.17.3.1.

If a NM message is not relevant and the configuration parameter 'All Nm Messages Keep Awake' is true the standard NM message reception handling is done, otherwise the NM message is ignored. As a consequence the ethernet channel is subsequently shutdown in case no relevant PN is requested by the own ECU and by other ECUs.

3.17.4 Handle Multiple Network Requests

The Handle Multiple Network Requests feature allows to initiate a state transition from `Ready Sleep State`, `Normal Operation State` or `Repeat Message State` to the `Repeat Message State` by a call to `UdpNm_NetworkRequest()`.

Note
Feature is enabled by setting **[PN Handle Multiple Network Requests]** in DaVinci Configurator Pro.

3.18 Error Handling

3.18.1 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 `UDPNM_DEV_ERROR_DETECT == STD_ON`).

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

The reported UdpNm ID is 33.

The reported service IDs identify the services which are described in chapter 5. The following table presents the service IDs and the related services:

Service ID	Service
0x01	`UdpNm_Init()`
0x02	`UdpNm_NetworkRequest()`
0x03	`UdpNm_NetworkRelease()`
0x04	`UdpNm_SetUserData()`
0x05	`UdpNm_GetUserData()`
0x06	`UdpNm_GetNodeIdentifier()`
0x07	`UdpNm_GetLocalNodeIdentifier()`
0x08	`UdpNm_RepeatMessageRequest()`
0x09	`UdpNm_GetVersionInfo()`
0x0A	`UdpNm_GetPduData()`
0x0B	`UdpNm_GetState()`
0x0C	`UdpNm_DisableCommunication()`
0x0D	`UdpNm_EnableCommunication()`
0x0E	`UdpNm_PassiveStartUp()`
0x0F	`UdpNm_SoAdIfTxConfirmation()`
0x10	`UdpNm_SoAdIfRxIndication()`
0x11	`UdpNm_CheckRemoteSleepIndication()`
0x12	`UdpNm_SetCoordBits()`
0x13	`UdpNm_MainFunction_x()`
0x14	`UdpNm_RequestBusSynchronization()`
0x15	`UdpNm_Transmit()`
0x16	`UdpNm_SetSleepReadyBit()`
0x20	`UdpNm_VTransmitPdu()`

Table 3-9 Service IDs

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

Error Code		Description
0x01	`UDPNM_E_NO_INIT`	API service called before module initialization
0x02	`UDPNM_E_INVALID_CHANNEL`	API service called with an invalid channel handle
0x03	`UDPNM_E_INVALID_PDUID`	API service called with an invalid PDU ID
0x12	`UDPNM_E_NULL_POINTER`	API service called with NULL pointer
0x21	`UDPNM_E_PDUR_TRIGGERTX_ERROR`	API service PduR_UdpNmTriggerTransmit

Error Code		Description
		returned unsuccessful or out parameters have invalid values
0x22	UDPNM_E_INVALID_USER_DATA_LEN	User Data passed for transmission does not fit into allocated space for User Data in NM PDU.

Table 3-10 Errors reported to DET

3.18.2 Production Code Error Reporting

By default, production code related errors are reported to the DEM using the service `Dem_ReportErrorStatus()` as specified in [3], if production error reporting is enabled (i.e. the related DEM-Event must be configured in the configuration tool).

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

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

Error Code	Description
UDPNM_E_NETWORK_TIMEOUT	Network timeout occurred
UDPNM_E_TCPIP_TRANSMIT_ERROR	No TX confirmation received within the defined timeout

Table 3-11 Errors reported to DEM

4 Integration

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

4.1 Scope of Delivery

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

4.1.1 Static Files

File Name	Source Code Delivery	Description
UdpNm.c	■	Implementation
UdpNm.h	■	API declaration
UdpNm_Cbk.h	■	API call-back declaration
UdpNm_Priv.h	■	Component local macro declaration
UdpNm_Types.h	■	Data type declaration

Table 4-1 Static files

Do not edit manually
The static files must not be edited manually!

4.1.2 Dynamic Files

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

File Name	Description
UdpNm_Cfg.h	Pre-compile time parameter declaration
UdpNm_Lcfg.h	Link-time parameter declaration
UdpNm_Lcfg.c	Link-time parameter

Table 4-2 Generated files

Do not edit manually
The dynamic files must not be edited manually but be generated with the configuration tool to guarantee a valid configuration of the module.

4.2 Compiler Abstraction and Memory Mapping

The objects (e.g. variables, functions, constants) are declared by compiler independent definitions – the compiler abstraction definitions. Each compiler abstraction definition is assigned to a memory section.

The following table contains the memory section names and the compiler abstraction definitions of the UdpNm and illustrates their assignment among each other.

Memory Mapping Sections \ Compiler Abstraction Definitions	UDPNM_CONST	UDPNM_VAR_ZERO_INIT	UDPNM_VAR_NOINIT	UDPNM_CODE
UDPNM_START_SEC_CONST_UNSPECIFIED UDPNM_STOP_SEC_CONST_UNSPECIFIED	■			
UDPNM_START_SEC_VAR_ZERO_INIT_UNSPECIFIED UDPNM_STOP_SEC_VAR_ZERO_INIT_UNSPECIFIED		■		
UDPNM_START_SEC_VAR_NOINIT_UNSPECIFIED UDPNM_STOP_SEC_VAR_NOINIT_UNSPECIFIED			■	
UDPNM_START_SEC_VAR_NOINIT_32BIT UDPNM_STOP_SEC_VAR_NOINIT_32BIT			■	
UDPNM_START_SEC_VAR_NOINIT_8BIT UDPNM_STOP_SEC_VAR_NOINIT_8BIT			■	
UDPNM_START_SEC_CODE UDPNM_STOP_SEC_CODE				■

Table 4-3 Compiler abstraction and memory mapping

4.3 Critical Sections

For the UdpNm module one critical section has to be configured, called `UDPNM_EXCLUSIVE_AREA_0`. This critical section guarantees consistent read and write operations on the UDPNM PDU data and on the user data. It is used within the following services:

- `UdpNm_SetUserData()`
- `UdpNm_GetUserData()`
- `UdpNm_GetPduData()`
- `UdpNm_SoAdIfRxIndication()`

The run-time of the critical section is limited to the data copy process (copying of max. eight bytes). It is recommended to use a global interrupt lock for this critical section.

5 API Description

5.1 Type Definitions

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

Type Name	C-Type	Desc.	Value Range
UdpNm_NetworkStateType	uint8	Defines the possible UDPNM network states	UDPNM_NETWORK_STATE_RELEASED Network has been released
			UDPNM_NETWORK_STATE_REQUESTED Network has been requested
UdpNm_PduPositionType	uint8	Defines the possible positions within the UDPNM PDU	UDPNM_PDU_BYTE_0 Byte number zero within the UDPNM PDU
			UDPNM_PDU_BYTE_1 Byte number one within the UDPNM PDU
			UDPNM_PDU_OFF Not contained within the UDPNM PDU
UdpNm_ConfigType	uint8	Defines the UDPNM config-uration type	*Pointer to this type:*
			NULL_PTR Pre-compile time or link-time configuration
			Valid Address Pointer to the post-build configuration (currently not supported!)

Table 5-1 Type definitions

5.1.1 UdpNm_InitMemory

Prototype	
void **UdpNm_InitMemory** (void)	
Parameter	
void	none
Return Code	
void	none
Functional Description	
Initialize global variables.	
Particularities and Limitations	
AUTOSAR extension	
Pre-Conditions	

This function has to be called before any other calls to the module.
Call Context
Task level

Table 5-2 UdpNm_InitMemory

5.1.2 UdpNm_Init

Prototype	
`void UdpNm_Init (const UdpNm_ConfigType *UdpNmConfigPtr)`	
Parameter	
UdpNmConfigPtr	Pointer to a selected configuration structure
Return Code	
void	none
Functional Description	
Initialize the complete UdpNm module, i.e. all channels which are activated at configuration time are initialized. A UDP socket shall be set up with the TCP/IP stack.	
Particularities and Limitations	
none	
Pre-Conditions	
This function has to be called before usage of the module.	
Call Context	
Initialization	

Table 5-3 UdpNm_Init

5.1.3 UdpNm_PassiveStartUp

Prototype	
`Std_ReturnType UdpNm_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 AUTOSAR UdpNm. It triggers the transition from Bus-Sleep Mode to the Network Mode in Repeat Message State.	

Particularities and Limitations
none
Pre-Conditions
Module is initialized.
Call Context
Task level

Table 5-4 UdpNm_PassiveStartUp

5.1.4 UdpNm_NetworkRequest

Prototype	
`Std_ReturnType` **`UdpNm_NetworkRequest`** `(const NetworkHandleType NmChannelHandle)`	
Parameter	
NmChannelHandle	Identification of the NM-channel
Return Code	
Std_ReturnType	E_OK No error E_NOT_OK Requesting of network has failed
Functional Description	
Request the network, since ECU needs to communicate on the bus. Network state shall be changed to 'requested'.	
Particularities and Limitations	
none	
Pre-Conditions	
Module is initialized.	
Call Context	
Task level	

Table 5-5 UdpNm_NetworkRequest

5.1.5 UdpNm_NetworkRelease

Prototype	
`Std_ReturnType` **`UdpNm_NetworkRelease`** `(const NetworkHandleType NmChannelHandle)`	
Parameter	
NmChannelHandle	Identification of the NM-channel

Return Code	
Std_ReturnType	E_OK No error E_NOT_OK Releasing of network has failed
Functional Description	
Release the network, since ECU doesn't have to communicate on the bus. Network state shall be changed to 'released'.	
Particularities and Limitations	
none	
Pre-Conditions	
Module is initialized.	
Call Context	
Task level	

Table 5-6 UdpNm_NetworkRelease

5.1.6 UdpNm_DisableCommunication

Prototype	
`Std_ReturnType` **`UdpNm_DisableCommunication`** `(const NetworkHandleType NmChannelHandle)`	
Parameter	
NmChannelHandle	Identification of the NM-channel
Return Code	
Std_ReturnType	E_OK No error E_NOT_OK Disabling of NM PDU transmission ability has failed
Functional Description	
Disable the NM PDU transmission ability due to a ISO14229 Communication Control (0x28) service.	
Particularities and Limitations	
none	
Pre-Conditions	
Module is initialized.	
Call Context	
Task level	

Table 5-7 UdpNm_DisableCommunication

5.1.7 UdpNm_EnableCommunication

Prototype	
`Std_ReturnType` **`UdpNm_EnableCommunication`** `(const NetworkHandleType NmChannelHandle)`	
Parameter	
NmChannelHandle	Identification of the NM-channel
Return Code	
Std_ReturnType	E_OK No error E_NOT_OK Enabling of NM PDU transmission ability has failed
Functional Description	
Enable the NM PDU transmission ability due to a ISO14229 Communication Control (0x28) service.	
Particularities and Limitations	
none	
Pre-Conditions	
Module is initialized.	
Call Context	
Task level	

Table 5-8 UdpNm_EnableCommunication

5.1.8 UdpNm_SetUserData

Prototype	
`Std_ReturnType` **`UdpNm_SetUserData`** `(const NetworkHandleType NmChannelHandle, const uint8 *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 of user data has failed
Functional Description	
Set user data for all NM messages transmitted on the bus after this function has returned without error.	
Particularities and Limitations	
none	
Pre-Conditions	
Module is initialized.	
Call Context	

Task level

Table 5-9 UdpNm_SetUserData

5.1.9 UdpNm_GetUserData

Prototype	
`Std_ReturnType` **`UdpNm_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 of user data has failed
Functional Description	
Get user data from the most recently received NM message.	
Particularities and Limitations	
none	
Pre-Conditions	
Module is initialized.	
Call Context	
Task level	

Table 5-10 UdpNm_GetUserData

5.1.10 UdpNm_GetNodeIdentifier

Prototype	
`Std_ReturnType` **`UdpNm_GetNodeIdentifier`** `(const NetworkHandleType NmChannelHandle, uint8 *const nmNodeIdPtr)`	
Parameter	
NmChannelHandle	Identification of the NM-channel
nmNodeIdPtr	Pointer where the source node identifier from the most recently received NM PDU shall be copied to.
Return Code	
Std_ReturnType	E_OK No error E_NOT_OK Getting of the source node identifier from the most recently received NM PDU has failed

Functional Description
Get node identifier from the most recently received NM PDU.
Particularities and Limitations
none
Pre-Conditions
Module is initialized.
Call Context
Task level

Table 5-11 UdpNm_GetNodeIdentifier

5.1.11 UdpNm_GetLocalNodeIdentifier

Prototype	
`Std_ReturnType` **`UdpNm_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 of the node identifier of the local node has failed
Functional Description	
Get node identifier configured for the local node.	
Particularities and Limitations	
none	
Pre-Conditions	
Module is initialized.	
Call Context	
Task level	

Table 5-12 UdpNm_GetLocalNodeIdentifier

5.1.12 UdpNm_RepeatMessageRequest

Prototype	
`Std_ReturnType` **`UdpNm_RepeatMessageRequest`** `(const NetworkHandleType NmChannelHandle)`	
Parameter	
NmChannelHandle	Identification of the NM-channel

isRepeatMessageBitIndication	Flag indicating whether a Repeat Message Request Bit was received
Return Code	
Std_ReturnType	E_OK No error E_NOT_OK Setting of Repeat Message Request Bit has failed
Functional Description	
Trigger transition to Repeat Message State and set Repeat Message Request Bit for all NM messages transmitted on the bus after this function has returned without error (dependent on the parameter 'isRepeatMessageBitIndication').	
Particularities and Limitations	
none	
Pre-Conditions	
Module is initialized.	
Call Context	
Task level	

Table 5-13 UdpNm_RepeatMessageRequest

5.1.13 UdpNm_GetPduData

Prototype	
`Std_ReturnType` **`UdpNm_GetPduData`** `(const NetworkHandleType NmChannelHandle, uint8 *const nmPduDataPtr)`	
Parameter	
NmChannelHandle	Identification of the NM-channel
nmPduDataPtr	Pointer where NM PDU shall be copied to
Return Code	
Std_ReturnType	E_OK No error E_NOT_OK Getting of NM PDU data has failed
Functional Description	
Get the whole PDU data out of the most recently received NM message.	
Particularities and Limitations	
none	
Pre-Conditions	
Module is initialized.	
Call Context	
Task level	

Table 5-14 UdpNm_GetPduData

5.1.14 UdpNm_GetState

Prototype	
`Std_ReturnType` **`UdpNm_GetState`** `(const NetworkHandleType NmChannelHandle, Nm_StateType *const nmStatePtr, Nm_ModeType *const nmModePtr)`	
Parameter	
NmChannelHandle	Identification of the NM-channel
nmStatePtr	Pointer where 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 of NM state has failed
Functional Description	
Returns the state and the mode of the network management.	
Particularities and Limitations	
none	
Pre-Conditions	
Module is initialized.	
Call Context	
Task level	

Table 5-15 UdpNm_GetState

5.1.15 UdpNm_GetVersionInfo

Prototype	
`void` **`UdpNm_GetVersionInfo`** `(Std_VersionInfoType *versioninfo)`	
Parameter	
versioninfo	Pointer for version info
Return Code	
void	none
Functional Description	
Returns the version information, vendor ID and AUTOSAR module ID of the component. The versions are BCD-coded.	
Particularities and Limitations	
none	
Pre-Conditions	
Module is initialized.	
Call Context	
Initialization or task level	

Table 5-16 UdpNm_GetVersionInfo

5.1.16 UdpNm_RequestBusSynchronization

Prototype	
`Std_ReturnType` **`UdpNm_RequestBusSynchronization`** `(const NetworkHandleType NmChannelHandle)`	
Parameter	
NmChannelHandle	Identification of the NM-channel
Return Code	
Std_ReturnType	E_OK No error E_NOT_OK Requesting of bus synchronization has failed
Functional Description	
Request bus synchronization.	
Particularities and Limitations	
none	
Pre-Conditions	
Module is initialized.	
Call Context	
Task level	

Table 5-17 UdpNm_RequestBusSynchronization

5.1.17 UdpNm_CheckRemoteSleepIndication

Prototype	
`Std_ReturnType` **`UdpNm_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 of remote sleep indication bits has failed
Functional Description	
Check if remote sleep indication takes place or not.	
Particularities and Limitations	
none	
Pre-Conditions	

Module is initialized.
Call Context
Task level

Table 5-18 UdpNm_CheckRemoteSleepIndication

5.1.18 UdpNm_SetCoordBits

Prototype	
`Std_ReturnType` **`UdpNm_SetCoordBits`** `(const NetworkHandleType NmChannelHandle, const uint8 nmCoordBits)`	
Parameter	
NmChannelHandle	Identification of the NM-channel
nmCoordBits	2 bit value to set the NM coordinator ID in the control bit vector of each NM message (coding as depicted in Figure "Control Bit Vector")
Return Code	
Std_ReturnType	E_OK No error E_NOT_OK Setting the coordinator ID bits has failed
Functional Description	
Sets the NM coordinator ID in the control bit vector of each NM message.	
Particularities and Limitations	
none	
Pre-Conditions	
Module is initialized.	
Call Context	
Task level	

Table 5-19 UdpNm_SetCoordBits

5.1.19 UdpNm_SetSleepReadyBit

Prototype	
`Std_ReturnType` **`UdpNm_SetSleepReadyBit`** `(const NetworkHandleType NmChannelHandle, const boolean nmSleepReadyBit)`	
Parameter	
NmChannelHandle	Identification of the NM-channel
nmSleepReadyBit	Value written to ReadySleep Bit in CBV
Return Code	
Std_ReturnType	E_OK No error E_NOT_OK Writing of remote sleep indication bit has failed

Functional Description
Set the NM Coordinator Sleep Ready bit in the Control Bit Vector and trigger an asynchronous NM message transmission.
Particularities and Limitations
none
Pre-Conditions
Module is initialized.
Call Context
Task level

Table 5-20 UdpNm_SetSleepReadyBit

5.1.20 UdpNm_Transmit

Prototype	
`Std_ReturnType` **`UdpNm_Transmit`** `(PduIdType UdpNmTxPduId, const PduInfoType *UdpNmSrcPduInfoPtr)`	
Parameter	
UdpNmSrcPduId	This parameter contains a unique identifier referencing to the PDU Routing Table and thereby specifying the socket to be used for transmission of the data
UdpNmSrcPduInfoPtr	A pointer to a structure with socket related data: data length and pointer to a data buffer
Return Code	
Std_ReturnType	E_OK The request has been accepted E_NOT_OK The request has not been accepted, e.g. due to a still ongoing transmission in the corresponding socket or the to be transmitted message is too long
Functional Description	
UdpNm_Transmit is implemented as an empty function and shall always return E_OK. The function UdpNm_Transmit is only available if the configuration switch UdpNmComUserDataSupport is enabled.	
Particularities and Limitations	
none	
Pre-Conditions	
Module is initialized.	
Call Context	
Task level	

Table 5-21 UdpNm_Transmit

5.1.21 Services used by UdpNm

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

Component	API
Det	`Det_ReportError()`
Dem	`Dem_ReportErrorStatus()`
SchM	`SchM_Enter_UdpNm()`
SchM	`SchM_Exit_UdpNm()`
Nm	`Nm_StateChangeNotification()`
Nm	`Nm_NetworkMode()`
Nm	`Nm_RemoteSleepCancellation()`
Nm	`Nm_PrepareBusSleepMode()`
Nm	`Nm_BusSleepMode()`
Nm	`Nm_NetworkStartIndication()`
Nm	`Nm_RepeatMessageIndication()`
Nm	`Nm_CoordReadyToSleepIndication()`
Nm	`Nm_PduRxIndication()`
SoAd	`SoAd_IfTransmit()`
PduR	`PduR_UdpNmTriggerTransmit()`
PduR	`PduR_UdpNmRxIndication()`

Table 5-22 Services used by the UdpNm

5.2 Callback Functions

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

5.2.1 UdpNm_SoAdIfTxConfirmation

Prototype	
`void` **`UdpNm_SoAdIfTxConfirmation`** `(PduIdType UdpNmTxPduId)`	
Parameter	
`UdpNmTxPduId[in]`	Identification of the network through PDU-ID
Return code	
`void`	none
Functional Description	
This service confirms a previous successfully processed transmit request.	
Particularities and Limitations	
Dependent on which PduR API version is configured, either `UdpNm_SoAdIfTxConfirmation()` (AUTOSAR 4) or `UdpNm_SoAdTxConfirmation()` (AUTOSAR 3) is available.	

Call Context
Interrupt or task level

Table 5-23 UdpNm_SoAdIfTxConfirmation

5.2.2 UdpNm_SoAdIfRxIndication

Prototype	
`void UdpNm_SoAdIfRxIndication (PduIdType udpNmRxPduId, const uint8 *udpSduPtr)`	
Parameter	
`udpNmRxPduId[in]`	Identification of the network through PDU-ID
`udpSduPtr[in]`	Pointer to the received SDU
Return code	
`void`	none
Functional Description	
This service indicates a successful reception of a received NM message to the UDPNM after passing all filters and validation checks.	
Particularities and Limitations	
Dependent on which PduR API version is configured, either `UdpNm_SoAdIfRxIndication()` (AUTOSAR 4) or `UdpNm_SoAdRxIndication()` (AUTOSAR 3) is available.	
Call Context	
Interrupt or task level	

Table 5-24 UdpNm_SoAdIfRxIndication

6 Configuration

The UdpNm can be configured with the Vector configuration tool DaVinci Configurator Pro.

6.1 Configuration Variants

The UdpNm supports the configuration variant

> VARIANT-PRE-COMPILE

The configuration classes of the UdpNm parameters depend on the supported configuration variants. For their definitions please see the UdpNm_bswmd.arxml file.

6.2 Configuration with the DaVinci Configurator Pro

Configuration of the UdpNm is done within the “Basic Editor” of the DaVinci Configurator Pro. For purpose of the configuration elements please refere to the help text provided within the tool.

7 Glossary and Abbreviations

7.1 Glossary

Term	Description
DaVinci Configurator Pro	Vector BSW Configuration Tool

Table 7-1 Glossary

7.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
BSWMD	BSW Module Description
COMM	Communication Manager
DEM	Diagnostic Event Manager
DET	Development Error Tracer
ECU	Electronic Control Unit
EIRA	External Internal Request Aggregation
ERA	External Request Aggregation
GCE	Generic Editor
ID	Identifier
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
NM	Network Management
PDU	Packet Data Unit
PDUR	PDU Router
PN	Partial Network
RTE	Runtime Environment
RX	Receive
SCHM	BSW Scheduler Module
SDU	Service Data Unit
SWS	Software Specification
TCP	Transmission Control Protocol
TP	Transport Protocol
TX	Transmit

UDP	User Datagram Protocol
UdpNm	UDP Network Management

Table 7-2 Abbreviations

8 Contact

Visit our website for more information on

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

www.vector.com