

Document Title	Specification of Network Management
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	898

Document Status	published
Part of AUTOSAR Standard	Adaptive Platform
Part of Standard Release	R23-11

Document Change History			
Date	Release	Changed by	Description
2023-11-23	R23-11	AUTOSAR Release Management	<ul style="list-style-type: none"> • Replaced Network Management Service Interface with C++ API • Several quality improvements
2022-11-24	R22-11	AUTOSAR Release Management	<ul style="list-style-type: none"> • Added clarifications regarding Operational Modes • Several quality improvements
2021-11-25	R21-11	AUTOSAR Release Management	<ul style="list-style-type: none"> • Several quality improvements • Removed chapter 10
2020-11-30	R20-11	AUTOSAR Release Management	<ul style="list-style-type: none"> • Several quality improvements • Changed NetworkState DataType from bool to NetworkStateType
2019-11-28	R19-11	AUTOSAR Release Management	<ul style="list-style-type: none"> • Added Functional Cluster Lifecycle Chapter • Several quality improvements • Improved linking to Manifest • Changed Document Status from Final to published
2019-03-29	19-03	AUTOSAR Release Management	<ul style="list-style-type: none"> • Introduced Service Interface for interaction via SM • Introduced possibility to group PNCs/Channels/VLANs



△

2018-10-30	18-10	AUTOSAR Release Management	<ul style="list-style-type: none">• Updated interaction with State Management• Removed APIs and Services (interaction is done via SM)• Temporary removed user data access to applications
2018-03-29	18-03	AUTOSAR Release Management	<ul style="list-style-type: none">• Initial release

Disclaimer

This work (specification and/or software implementation) and the material contained in it, as released by AUTOSAR, is for the purpose of information only. AUTOSAR and the companies that have contributed to it shall not be liable for any use of the work.

The material contained in this work is protected by copyright and other types of intellectual property rights. The commercial exploitation of the material contained in this work requires a license to such intellectual property rights.

This work may be utilized or reproduced without any modification, in any form or by any means, for informational purposes only. For any other purpose, no part of the work may be utilized or reproduced, in any form or by any means, without permission in writing from the publisher.

The work has been developed for automotive applications only. It has neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Contents

1	Introduction and functional overview	6
2	Acronyms and Abbreviations	7
3	Related documentation	8
3.1	Input documents & related standards and norms	8
3.2	Further applicable specification	8
4	Constraints and assumptions	9
4.1	Known Limitations	9
4.2	Applicability to car domains	9
5	Dependencies to other Functional Clusters	10
5.1	Provided Interfaces	10
5.2	Required Interfaces	11
5.3	Platform dependencies	11
5.4	Protocol layer dependencies	11
6	Requirements Tracing	12
7	Functional specification	14
7.1	Architectural Overview	14
7.2	Network Management Algorithm	16
7.3	NetworkControl	17
7.4	Operational Modes	19
7.4.1	Network Mode	20
7.4.1.1	Repeat Message State	21
7.4.1.2	Normal Operation State	22
7.4.1.3	Ready Sleep State	22
7.4.2	Prepare Bus-Sleep Mode	23
7.4.3	Bus-Sleep Mode	23
7.5	Message Format	24
7.5.1	Source Node Identifier	24
7.5.2	Control Bit Vector	24
7.5.3	User Data	24
7.6	Nm Transmission	26
7.6.1	Transmission Scheduling	26
7.7	Nm User Data Handling	27
7.8	Partial Networking	27
7.8.1	Partial Network State Machine	27
7.8.2	Rx Handling of NM messages	27
7.8.3	Tx Handling of NM messages	27
7.8.4	NM message Filter Algorithm	28
7.9	Functional Cluster Lifecycle	29
7.9.1	Startup	29

7.9.2	Shutdown	29
8	API specification	30
8.1	Errors	30
8.1.1	NmErrc	30
8.1.2	NmException class	31
8.1.3	NmErrorDomain class	31
8.2	NetworkHandle class	34
8.2.1	Special member functions	34
8.2.2	Class Specific Data Types	36
8.2.2.1	NetworkStateChangeNotifier	36
8.2.2.2	NetworkStateType	36
8.2.3	GetNetworkState	37
8.2.4	GetNetworkRequestedState	37
8.2.5	SetNetworkRequestedState	38
8.2.6	RegisterNetworkStateChangeNotifier	38
8.2.7	UnregisterNetworkStateChangeNotifier	39
8.2.8	RegisterNetworkRequestedStateChangeNotifier	40
8.2.9	UnregisterNetworkRequestedStateChangeNotifier	41
9	Service Interfaces	42
A	Mentioned Manifest Elements	43
B	History of Specification Items	50
B.1	Specification Item History of this document compared to AUTOSAR R22-11.	50
B.1.1	Added Specification Items in R23-11	50
B.1.2	Changed Specification Items in R23-11	51
B.1.3	Deleted Specification Items in R23-11	52
B.2	Specification Item History of this document compared to AUTOSAR R21-11.	52
B.2.1	Added Specification Items in R22-11	52
B.2.2	Changed Specification Items in R22-11	52
B.2.3	Deleted Specification Items in R22-11	52

1 Introduction and functional overview

This specification describes the functionality, API and the configuration of the Network Management for the AUTOSAR Adaptive Platform.

Adaptive Network Management (NM) is intended to work independent of the communication stack used. Its main purpose is to coordinate the transition between normal operation and bus-sleep mode of the underlying networks (physical and partial networks).

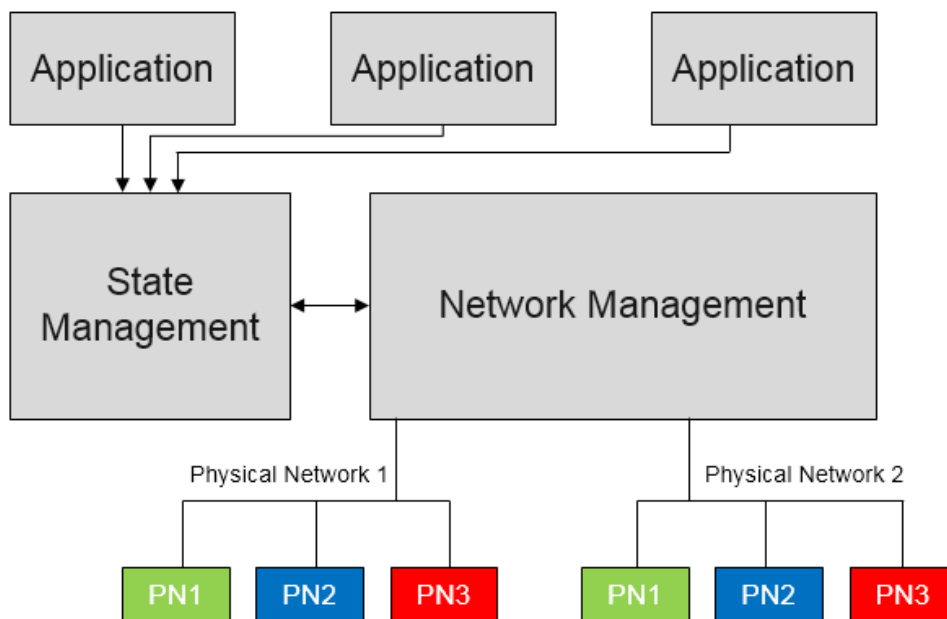


Figure 1.1: Architecture overview with example applications

2 Acronyms and Abbreviations

The glossary below includes acronyms and abbreviations relevant to the Adaptive Network Management that are not included in the AUTOSAR glossary [1].

Abbreviation / Acronym:	Description:
API	Application Programming Interface
CBV	Control Bit Vector
CM	Communication Management
CWU	Car Wakeup
EM	Execution Management
IP	Internet Protocol
MTU	Maximum Transmission Unit
NM	Network Management
NM Node	A node that supports network management. Please note that network node, node and NM node are used with the same meaning throughout the document.
PN	Partial Network
PNI	Partial Network Information
PNL	Partial Network Learning
UDP	User Datagram Protocol

Terms:	Description:
Bus communication	Communication on the physical medium
Logical Network	A network in which devices can be addressed independent from the actual network technology.
NM cluster	Set of NM nodes coordinated with the use of the NM algorithm.
NM message	Refers to the payload transmitted in a packet. It contains the NM User Data, Partial Network Information as well as the Control Bit Vector and the Source Node Identifier.
NM packet	Refers to an Ethernet Frame containing an IP as well as an UDP header in addition to a NM message. Please note that adaptive network management is currently only supported for Ethernet.
PN communication	Communication during partial network operation
Physical channel	A channel enabling communication using physical devices, such as I/O ports and cables.
Repeat Message Request Bit Indication	Repeat Message Bit set in the Control Bit Vector of a received NM message.
Internally Requested	At least one field NetworkRequestedState associated to that channel/network/PNC/VLAN is set to kFullCom.
Externally Requested	A Network Management Message associated to that channel/network/PNC/VLAN has been received. In case of PNC associated means the bit corresponding to this PNC had the value 1.
FULL_COM	Communication over the network is possible/allowed, the network is up.
NO_COM	Communication over the network is impossible/disabled, the network is down.

3 Related documentation

3.1 Input documents & related standards and norms

- [1] Glossary
AUTOSAR_FO_TR_Glossary
- [2] General Requirements specific to Adaptive Platform
AUTOSAR_AP_RS_General
- [3] Specification of the AUTOSAR Network Management Protocol
AUTOSAR_FO_PRS_NetworkManagementProtocol
- [4] Requirements on AUTOSAR Network Management
AUTOSAR_FO_RS_NetworkManagement
- [5] Specification of State Management
AUTOSAR_AP_SWS_StateManagement

3.2 Further applicable specification

AUTOSAR provides a General Specification on Basic Software modules [2, RS General], which is also valid for the [NM](#).

4 Constraints and assumptions

4.1 Known Limitations

The Adaptive Network Management is actually only supporting UdpNM.

The Adaptive Network Management does not allow node detection (Repeat Message State) but only handles incoming requests.

The Adaptive Network Management cannot be configured as the master network coordinator.

The Adaptive Network Management does not support coordinated shutdown using the information in CBV.

The Adaptive Network Management does not support passive mode and passive start-up. Passive start-up would mean that a node has started (i.e. goes to Normal mode), but the network has been woken up by another node.

Modeling part for mapping the logical networks to the BitVector positions as defined in chapter 7.3 is not available in the manifest.

Update and access of User Data was removed as the service interface to Applications has been removed. State Management will control the network request/release and it must be clarified if user data changes/indications shall be done via State Management or directly by applications.

4.2 Applicability to car domains

AUTOSAR Adaptive Network Management can be used for all car domains.

5 Dependencies to other Functional Clusters

This chapter provides an overview of the dependencies to other Functional Clusters in the AUTOSAR Adaptive Platform.

Section 5.1 “[Provided Interfaces](#)” lists the interfaces provided by Network Management to other Functional Clusters. Section 5.2 “[Required Interfaces](#)” lists the interfaces required by Network Management.

5.1 Provided Interfaces

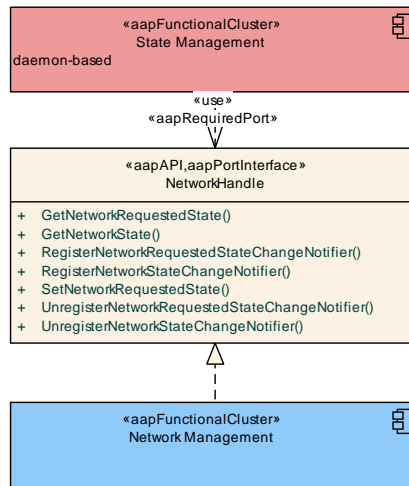


Figure 5.1: Interfaces provided by Network Management to other Functional Clusters

Interface	Functional Cluster	Purpose
NetworkHandle	State Management	This interface shall be used to retrieve information about the network status of a <code>NetworkHandle</code> .

Table 5.1: Interfaces provided to other Functional Clusters

5.2 Required Interfaces

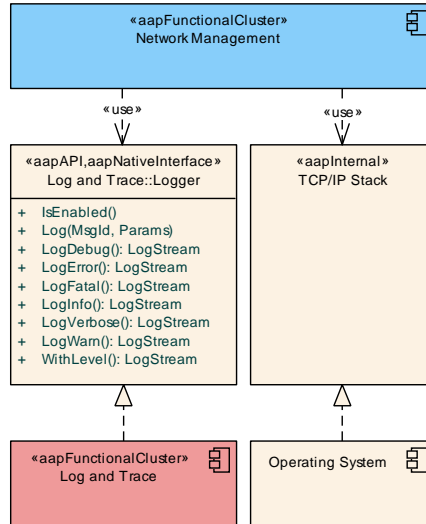


Figure 5.2: Interfaces required by Network Management from other Functional Clusters

Functional Cluster	Interface	Purpose
Log and Trace	Logger	Network Management shall use this interface to log standardized messages.

Table 5.2: Interfaces required from other Functional Clusters

5.3 Platform dependencies

This specification is part of the AUTOSAR Adaptive Platform and therefore depends on it.

5.4 Protocol layer dependencies

The Adaptive Network Management is based on the protocol mentioned in PRS NetworkManagementProtocol [3].

Adaptive Network Management uses functionality of the underlying communication stack in order to send or receive NM messages on the physical networks.

6 Requirements Tracing

The following table references the requirements specified in RS Adaptive Network Management [4] and links to the fulfillment of these. Please note that if column “Satisfied by” is empty for a specific requirement, or a requirement is not reported in the table, it means that this requirement is not fulfilled by this document.

Requirement	Description	Satisfied by
[RS_AP_00119]	Return values / application errors.	[SWS_ANM_01105] [SWS_ANM_01106] [SWS_ANM_01107] [SWS_ANM_01109] [SWS_ANM_01110]
[RS_AP_00120]	Method and Function names.	[SWS_ANM_01102] [SWS_ANM_01105] [SWS_ANM_01106] [SWS_ANM_01107] [SWS_ANM_01108] [SWS_ANM_01109] [SWS_ANM_01110]
[RS_AP_00121]	Parameter names.	[SWS_ANM_01102] [SWS_ANM_01107] [SWS_ANM_01108] [SWS_ANM_01110]
[RS_AP_00122]	Type names.	[SWS_ANM_01100] [SWS_ANM_01101] [SWS_ANM_01103] [SWS_ANM_01104] [SWS_ANM_01111]
[RS_AP_00125]	Enumerator and constant names.	[SWS_ANM_01111]
[RS_AP_00127]	Usage of ara::core types.	[SWS_ANM_01100] [SWS_ANM_01101] [SWS_ANM_01111]
[RS_AP_00132]	noexcept behavior of API functions	[SWS_ANM_01102] [SWS_ANM_01105] [SWS_ANM_01106] [SWS_ANM_01107] [SWS_ANM_01109] [SWS_ANM_01110]
[RS_AP_00135]	Avoidance of shared ownership.	[SWS_ANM_01102] [SWS_ANM_01110]
[RS_AP_00140]	Usage of "final specifier" in ara types.	[SWS_ANM_01101]
[RS_AP_00149]	Guidance on error handling.	[SWS_ANM_01111]
[RS_Nm_00043]	Nm shall not prohibit bus traffic with Nm not being initialized	[SWS_ANM_00090]
[RS_Nm_00044]	The Nm shall be applicable to different types of communication systems which are in the scope of AUTOSAR and support a bus sleep mode.	[SWS_ANM_00005] [SWS_ANM_00006] [SWS_ANM_00007] [SWS_ANM_00008] [SWS_ANM_00009] [SWS_ANM_00012] [SWS_ANM_00013] [SWS_ANM_00016] [SWS_ANM_00017] [SWS_ANM_00021] [SWS_ANM_00044] [SWS_ANM_00046] [SWS_ANM_00047] [SWS_ANM_00062] [SWS_ANM_00070] [SWS_ANM_00092]
[RS_Nm_00045]	Nm shall provide services to coordinate shutdown of Nm-clusters independently of each other	[SWS_ANM_01009]
[RS_Nm_00047]	Nm shall provide a service to request to keep the bus awake and a service to cancel this request.	[SWS_ANM_00011] [SWS_ANM_00014] [SWS_ANM_00015] [SWS_ANM_00016] [SWS_ANM_00018] [SWS_ANM_00019] [SWS_ANM_00020] [SWS_ANM_00022] [SWS_ANM_00023] [SWS_ANM_00025] [SWS_ANM_00066] [SWS_ANM_00086] [SWS_ANM_00087] [SWS_ANM_00088]
[RS_Nm_00048]	Nm shall put the communication controller into sleep mode if there is no bus communication	[SWS_ANM_00024]





Requirement	Description	Satisfied by
[RS_Nm_00050]	The Nm shall provide the current state of Nm	[SWS_ANM_00063] [SWS_ANM_00083] [SWS_ANM_01000] [SWS_ANM_01001] [SWS_ANM_01002] [SWS_ANM_01003] [SWS_ANM_01004] [SWS_ANM_01005] [SWS_ANM_01006] [SWS_ANM_01007] [SWS_ANM_01008]
[RS_Nm_00051]	Nm shall inform application when Nm state changes occur.	[SWS_ANM_01007] [SWS_ANM_01008] [SWS_ANM_01010] [SWS_ANM_01011] [SWS_ANM_01012] [SWS_ANM_01013] [SWS_ANM_01014] [SWS_ANM_01015] [SWS_ANM_01020] [SWS_ANM_01358]
[RS_Nm_00054]	There shall be a deterministic time from the point where all nodes agree to go to bus sleep to the point where bus is switched off.	[SWS_ANM_00024]
[RS_Nm_00150]	Specific features of the Network Management shall be configurable	[SWS_ANM_00007] [SWS_ANM_00013] [SWS_ANM_00029] [SWS_ANM_00033] [SWS_ANM_00035] [SWS_ANM_00040] [SWS_ANM_00044] [SWS_ANM_00047] [SWS_ANM_00051] [SWS_ANM_00081] [SWS_ANM_00084] [SWS_ANM_00085] [SWS_ANM_00089] [SWS_ANM_00095]
[RS_Nm_00151]	The Network Management algorithm shall allow any node to integrate into an already running Nm cluster	[SWS_ANM_00037] [SWS_ANM_00038] [SWS_ANM_00071] [SWS_ANM_00091]
[RS_Nm_02505]	The Nm shall optionally set the local node identifier to the Nm-message	[SWS_ANM_00033] [SWS_ANM_00034]
[RS_Nm_02508]	Every node shall have a node identifier associated with it that is unique in the Nm-cluster.	[SWS_ANM_00034]
[RS_Nm_02513]	Nm shall provide functionality which enables upper layers to control the sleep mode.	[SWS_ANM_01009]
[RS_Nm_02517]	CanNm shall support Partial Networking on CAN	[SWS_ANM_00067] [SWS_ANM_00092]
[RS_Nm_02519]	The Nm Control Bit Vector shall contain a PNI (Partial Network Information) bit.	[SWS_ANM_00051] [SWS_ANM_00055]
[RS_Nm_02527]	Nm shall implement a filter algorithm dropping all Nm messages that are not relevant for the ECU	[SWS_ANM_00081]
[RS_Nm_02528]	Nm shall provide a service which allows for instantaneous sending of Nm messages.	[SWS_ANM_00094]
[RS_Nm_02546]	UdpNm shall support Partial Networking on Ethernet	[SWS_ANM_00051] [SWS_ANM_00055]

Table 6.1: RequirementsTracing

7 Functional specification

The Adaptive Network Management offers services that allows to request and query the network states for logical network handles that can be mapped to physical or partial networks.

To do so, the following functionalities are provided:

- 1) Interfaces for requesting and releasing logical network handles
- 2) Support for partial networking

7.1 Architectural Overview

Figure 7.1 gives an overview of the Adaptive NM service.

The following figure shows an overview on the interaction between [5] and *Network Management* as well as an example mapping between logical networks, partial networks and physical networks.

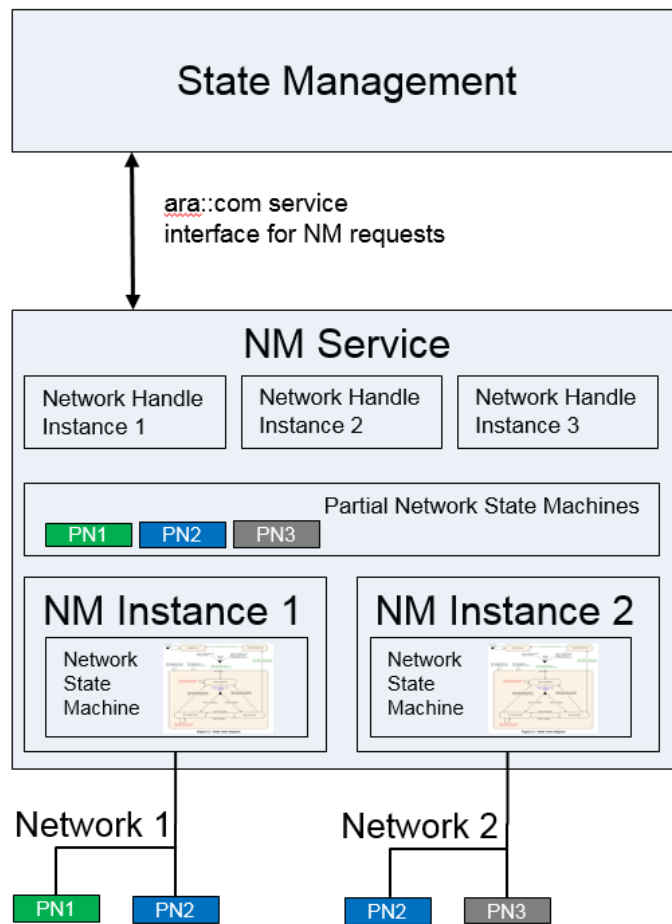


Figure 7.1: Overview Of Network Management

7.2 Network Management Algorithm

The AUTOSAR Adaptive NM is based on decentralized direct network management strategy, which means that every network node performs activities self-sufficient depending only on the NM packets received and/or transmitted within the communication system.

The AUTOSAR Adaptive NM algorithm is based on periodic NM packets, which are received by all nodes in the cluster via multicast. Reception of NM packets indicates that sending nodes want to keep the NM-cluster awake. If any node is ready to go to sleep mode, it stops sending NM packets, but as long as NM packets from other nodes are received, it postpones transition to sleep mode. Finally, if a dedicated timer elapses because no NM packets are received anymore, every node initiates transition to the sleep mode.

If any node in the NM-cluster requires bus-communication, it can keep the NM-cluster awake by transmitting NM packets.

The main concept of the AUTOSAR Adaptive NM coordination algorithm as described in [3] can be summarized by the following key-behavior:

Every network node transmits periodic NM messages as long as it requires bus-communication; otherwise it does not transmit NM messages.

7.3 NetworkControl

Logical network handles are mapped to one or more partial or physical networks, while a partial network itself can be mapped to one or multiple physical networks. By using the logical network handle all assigned partial networks, VLANs and underlying physical channel(s) are controlled together.

With the introduction of the State Management functional cluster, Network Management no longer receives logical network requests from applications, instead they are controlled by the State Management. State Management can split the one or more applications in multiple functions that might require network communication. Applications (or part of) would then request different functions to be activated/deactivated from State Management and then State Management would in turn, depending on configuration, request/release different logical networks. NM checks then the requested state over all logical networks handles and will activate or deactivate the according physical networks.

[SWS_ANM_00063] [Each port offered by NM shall enable control of one logical `NmNetworkHandle` which in turn can be mapped to partial networks or VLANs.] ([RS_Nm_00050](#))

Note: In the Manifest the untagged VLAN represents the physical ethernet channel.

[SWS_ANM_00066] [Each logical `NmNetworkHandle` shall be mapped to partial networks (via `PncMappingIdent`) and/or VLANs (via `EthernetCommunicationConnector`). Configurations where a logical `NmNetworkHandle` maps the same VLAN directly and via partial network(s) shall not be possible.] ([RS_Nm_00047](#))

[SWS_ANM_00067] [If partial networking is used a mapping between partial network(s) and `EthernetCommunicationConnector` shall be configured in `PncMapping`.] ([RS_Nm_02517](#)) Note: One Partial Network can be mapped to several VLAN(s)

[SWS_ANM_00083] [The return value of the `ara::nm::NetworkHandle::GetNetworkState` shall be `kFullCom`, if all PNCs, VLANs and/or physical channels associated to this instance of `NmNetworkHandle` are in `FULL_COM`. Otherwise the value shall be `kNoCom`.] ([RS_Nm_00050](#))

Note: The consequence of [\[SWS_ANM_00083\]](#) is, that a lowest wins strategy is applied.

[SWS_ANM_00084] [If `ara::nm::NetworkHandle::SetNetworkRequestedState` is called with `kFullCom`, the Network Management shall consider each PNC, VLAN and/or physical channel associated with the instance of `NmNetworkHandle` as internally requested.] ([RS_Nm_00150](#))

Note: The consequence of [\[SWS_ANM_00084\]](#) is, that a highest wins strategy is applied, that means that if in any instance of `NmNetworkHandle` the network requested state is set to `FULL_COM` (via `ara::nm::NetworkHandle::SetNetworkRequestedState` is called with `kFullCom`), the target state of the associated PNC/VLAN/channel(s) is Network Mode, substate Normal Operation State.

[SWS_ANM_00085] [Network Management shall bring (or keep) all networks/physical channels to `FULL_COM` that are either internally requested or externally requested.] (*RS_Nm_00150*)

[SWS_ANM_00086] [A PNC shall be considered in `FULL_COM`, if all physical channels, to which it is mapped, are in `FULL_COM` and the PNC is internally or externally requested. This includes a call of the registered change notification via `ara::nm::NetworkHandle::RegisterNetworkStateChangeNotifier` or via `ara::nm::NetworkHandle::RegisterNetworkStateChangeNotifier` using an execution context executor. If the notification is registered via `ara::nm::NetworkHandle::RegisterNetworkStateChangeNotifier` using an execution context executor, the call shall be done in the context of the passed `executor`.] (*RS_Nm_00047*)

[SWS_ANM_00087] Handling of external wake-up [Upon detection of of an external wake-up, `FULL_COM` shall be the target state for the corresponding channel(s). If Network Management is configured for that channel, the target state shall be `Network Mode`, with the default initial sub state `Repeat Message`.] (*RS_Nm_00047*)

Note: Its up to the Platform Implementation how an external wakeup event is detected.

[SWS_ANM_00088] Default target state after start-up [The default target state after start up for channels for which no external wake-up has been detected shall be `NO_COM`.] (*RS_Nm_00047*)

7.4 Operational Modes

This chapter describes the operational modes of the AUTOSAR Adaptive NM.

[SWS_ANM_00062]{DRAFT} [NM shall realize the behavior of the state machine mentioned in the figure described below for every physical channel separately.] ([RS_Nm_00044](#))

Note: The state machine in Figure 7.2 is applied to physical channels. In case of partial networking, the Nm module should additionally take care of relevant PNs.

The Network Management contains three operational modes:

- Network Mode, see [7.4.1](#)
- Prepare Bus-Sleep Mode, see [7.4.2](#)
- Bus-Sleep Mode, see [7.4.3](#)

These modes will not be visible to the Adaptive Application as it is.

When the NM is in Network mode it implies that the network is requested or active. And the logical network information bit will be set to 1.

When the NM is in Prepare Bus-Sleep or Bus-Sleep Mode, It implies that the network is released or inactive. And the logical network information bit will be set to 0.

The following figure shows the state diagram. Mode change related transitions are denoted in green and error handling related transmissions in red.

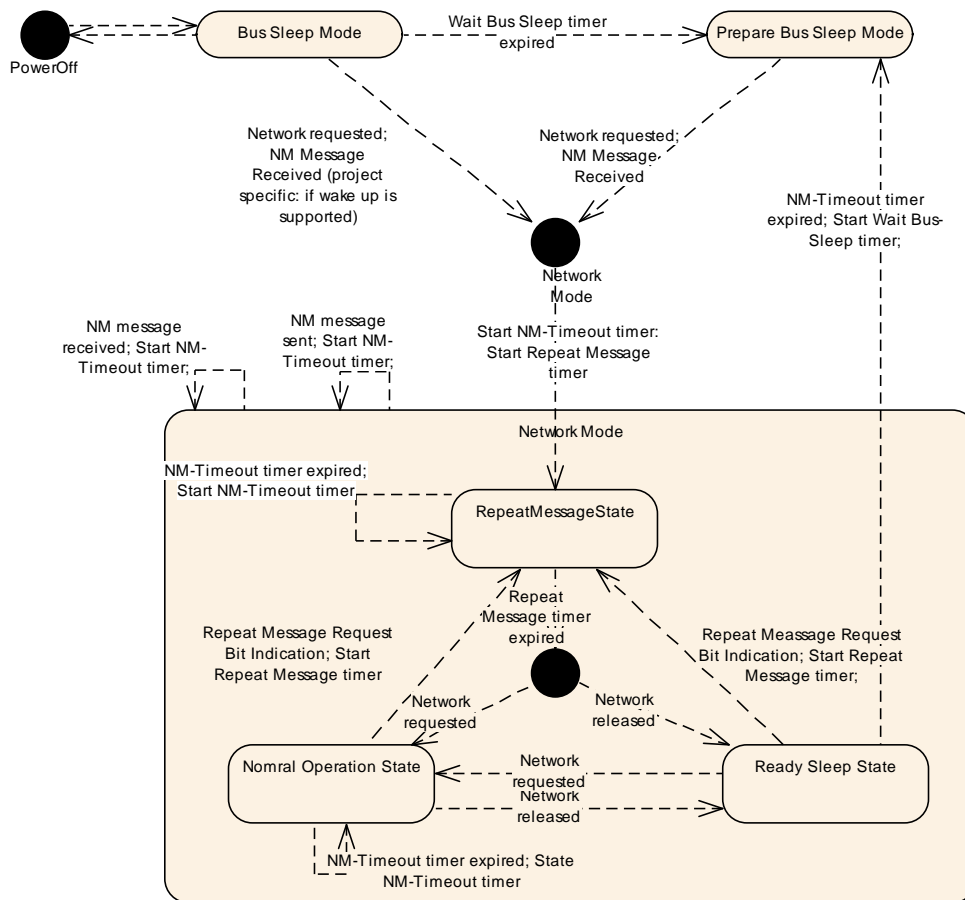


Figure 7.2: State Chart Diagram

7.4.1 Network Mode

[SWS_ANM_00005] [The Network Mode shall consist of three internal states:

- Repeat Message State
- Normal Operation State
- Ready Sleep State

](RS_Nm_00044)

For more information check the following chapters:

- Repeat Message State, see 7.4.1.1
- Normal Operation State, see 7.4.1.2
- Ready Sleep State, see 7.4.1.3

[SWS_ANM_00006] [When the Network Mode is entered from Bus-Sleep Mode or Prepare Bus-Sleep Mode, by default, the Repeat Message State shall be entered.] (RS_Nm_00044)

[SWS_ANM_00007] [When the Network Mode is entered, the NM-Timeout Timer shall be started with the value `nmNetworkTimeout`.] ([RS_Nm_00044](#), [RS_Nm_00150](#))

[SWS_ANM_00008] [Upon successful reception of a NM message in Network Mode, the NM-Timeout Timer shall be restarted with the value `nmNetworkTimeout`.] ([RS_Nm_00044](#))

[SWS_ANM_00009] [Upon successful transmission of a NM message in Network Mode, the NM-Timeout Timer shall be restarted with the value `nmNetworkTimeout`.] ([RS_Nm_00044](#))

7.4.1.1 Repeat Message State

The Repeat Message State ensures, that any transition from Bus-Sleep or Prepare Bus-Sleep to the Network Mode becomes visible for the other nodes on the network. Additionally, it ensures that any node stays active for a minimum amount of time.

[SWS_ANM_00011] [When the Repeat Message State of Network Mode is entered from Bus-Sleep Mode, Prepare-Bus-Sleep Mode or from within Network Mode (Normal Operation State or Ready Sleep State) transmission of NM messages shall be (re-) started.] ([RS_Nm_00047](#))

[SWS_ANM_00012] [When the NM-Timeout Timer expires in the Repeat Message State, the NM-Timeout Timer shall be restarted.] ([RS_Nm_00044](#))

[SWS_ANM_00013] [The NM shall stay in the Repeat Message State for a configurable amount of time determined by the `nmRepeatMessageTime`; after that time the Repeat Message State shall be left.] ([RS_Nm_00044](#), [RS_Nm_00150](#))

[SWS_ANM_00014] [When Repeat Message State is left, the Normal Operation State shall be entered, if the network has been requested.] ([RS_Nm_00047](#))

[SWS_ANM_00015] [When Repeat Message State is left, the Ready Sleep State shall be entered, if the network has been released.] ([RS_Nm_00047](#))

[SWS_ANM_00070] [The Repeat Message State of Network Mode is entered from Bus-Sleep Mode or Prepare-Bus-Sleep Mode by default, when a network (channel/P-NC/VLAN) is requested by calling `ara::nm::NetworkHandle::SetNetworkRequestedState` with the value `kFullCom` and the NM module shall transmit a NM message immediately.] ([RS_Nm_00044](#))

[SWS_ANM_00092]{DRAFT} nmPnHandleMultipleNetworkRequests [If in Ready Sleep State, Normal Operation State or Repeat Message State and `nmPnHandleMultipleNetworkRequests` is set to true and the requested state of the channel/VLAN or an associated PNC changes, Repeat Message State shall be (re-)entered.] ([RS_Nm_00044](#), [RS_Nm_02517](#))

7.4.1.2 Normal Operation State

The Normal Operation State ensures that any node can keep the NM-cluster awake as long as the network functionality is required.

[SWS_ANM_00016] [When the Normal Operation State is entered from Ready Sleep State, transmission of NM messages shall be started immediately.] ([RS_Nm_00047](#), [RS_Nm_00044](#))

[SWS_ANM_00017] [When the NM-Timeout Timer expires in the Normal Operation State, the NM-Timeout Timer shall be restarted.] ([RS_Nm_00044](#))

[SWS_ANM_00018] [When the network is released and the current state is Normal Operation State, the Normal Operation State shall be left and the Ready Sleep state shall be entered.] ([RS_Nm_00047](#))

[SWS_ANM_00019] [If Repeat Message Request Bit (set in the CBV of the received NM message) is received in the Normal Operation State, the Normal Operation State shall be left and the Repeat Message State shall be entered.] ([RS_Nm_00047](#))

7.4.1.3 Ready Sleep State

The Ready Sleep State ensures that any node in the NM-cluster waits with the transition to the Prepare Bus-Sleep Mode as long as any other node keeps the NM-cluster awake.

[SWS_ANM_00020] [When the Ready Sleep State is entered from Repeat Message State or Normal Operation State, transmission of NM messages shall be stopped.] ([RS_Nm_00047](#))

[SWS_ANM_00021] [When the NM-Timeout Timer expires in the Ready Sleep State, the Ready Sleep State shall be left and the Prepare Bus-Sleep Mode shall be entered.] ([RS_Nm_00044](#))

[SWS_ANM_00022] [When the network is requested (by calling `ara::nm::NetworkHandle::SetNetworkRequestedState` with `kFullCom`) and the current state is the Ready Sleep State, the Ready Sleep State shall be left and the Normal Operation State shall be entered.] ([RS_Nm_00047](#))

[SWS_ANM_00023] [If Repeat Message Request Bit (set in the CBV of the received NM message) is received in the Ready Sleep State, the Ready Sleep State shall be left and the Repeat Message State shall be entered.] ([RS_Nm_00047](#))

Note: Handling of multiple transition conditions which might arise at the same time (e.g. NM-Timeout timer expires vs. network is requested) is considered to be implementation-specific.

7.4.2 Prepare Bus-Sleep Mode

The purpose of the Prepare Bus Sleep state is to ensure that all nodes have time to stop their network activity before the Bus Sleep state is entered. Bus activity is calmed down (i.e. queued messages are transmitted in order to empty all TX-buffers) and finally there is no activity on the bus in the Prepare Bus-Sleep Mode.

[SWS_ANM_00024] [The NM shall stay in the Prepare Bus-Sleep Mode for an amount of time determined by the `nmWaitBusSleepTime`; after that time, the Prepare Bus-Sleep Mode shall be left and the Bus-Sleep Mode shall be entered.] (*RS_Nm_00048*, *RS_Nm_00054*)

[SWS_ANM_00025] [Upon successful reception of a NM message in the Prepare Bus-Sleep Mode, the Prepare Bus-Sleep Mode shall be left and the Network Mode shall be entered; by default, the Repeat Message State is entered.] (*RS_Nm_00047*)

Rationale: Other nodes in the cluster are still in Prepare Bus-Sleep Mode; in the exceptional situation described above, transition into the Bus-Sleep Mode shall be avoided and bus-communication shall be restored as fast as possible.

7.4.3 Bus-Sleep Mode

The purpose of the Bus-Sleep state is to reduce power consumption in the node, when no messages are to be exchanged. Transmission and reception capabilities can be switched off if supported by hardware.

If a configurable amount of time determined by `nmNetworkTimeout` + `nmWaitBusSleepTime` is identically configured for all nodes in the network management cluster, all nodes in the network management cluster that are coordinated with use of the AUTOSAR NM algorithm perform the transition into the Bus-Sleep Mode at approximately the same time.

[SWS_ANM_00029]{DRAFT} [In Bus-Sleep Mode the return value of the corresponding `ara::nm::NetworkHandle::GetNetworkState` are `kNoCom` (see also **[SWS_ANM_00083]**).] (*RS_Nm_00150*)

Note: Reception of a message during bus sleep (if receiving capability is not switched off) is not explicitly handled in this specification as for example wake-up is considered project specific.

Note: In Bus-Sleep Mode, it is assumed that all nodes in a cluster are in this state. Typically, all nodes request the communication approximately at the same time by a common trigger, for instance a wake-up line.

7.5 Message Format

Message Layout is shown in [3].

Note: As mentioned in [3], the length of an NM packet shall not exceed the MTU (Maximum Transmission Unit) of the underlying physical transport layer.

7.5.1 Source Node Identifier

[SWS_ANM_00033] [The location of the source node identifier shall be taken from `nmNidPosition`. If `nmNidPosition` is not set, NID shall not be contained in the Nm messages (see [PRS_Nm_00074]).] (*RS_Nm_00150*, *RS_Nm_02505*)

[SWS_ANM_00034] [The source node identifier shall be set with configurable Node-Id value `nmNodeId` unless the location of the source node identifier is set to Off (see [PRS_Nm_00013]).] (*RS_Nm_02508*, *RS_Nm_02505*)

7.5.2 Control Bit Vector

The format (bit-layout) and definition of the CBV is described in [3].

[SWS_ANM_00035] [The location of the Control Bit Vector shall be configurable using `nmCbvPosition`. If `nmCbvPosition` is not set, CBV shall not be contained in the Nm messages (see [PRS_Nm_00075]).] (*RS_Nm_00150*)

[SWS_ANM_00037] [Repeat Message Request Bit shall always be set to 0 in the transmitted NM message.] (*RS_Nm_00151*)

[SWS_ANM_00038] [Active Wakeup Bit shall always be set to 0 in the transmitted NM message.] (*RS_Nm_00151*)

[SWS_ANM_00071] [NM Coordinator Sleep Ready Bit shall always be set to 0 in the transmitted NM message.] (*RS_Nm_00151*)

[SWS_ANM_00091] [Partial Network Learning Bit (PNL) shall always be set to 0 in the transmitted NM message.] (*RS_Nm_00151*)

7.5.3 User Data

[SWS_ANM_00040] [If NM user data is configured (i.e. `nmUserDataLength` is existing with a value greater than 0) it shall be always included in the NM message (see [PRS_Nm_00158]).] (*RS_Nm_00150*)

Note: the range (in bytes) that contains the user data in the received NM message is defined by `nmUserDataLength`.

Note: UserData does not include the PNI in case the Partial Networking is active. Received and Transmitted UserData does not overlap with the PNI.

[SWS_ANM_00095]{DRAFT} [If `nmUserDataLength` is existing with a value greater than 0 and no data is provided by the application then the Nm shall set the user data to 0 before sending the NM message.] ([RS_Nm_00150](#))

Note: Currently there is no standardized API to get/set user data.

7.6 Nm Transmission

7.6.1 Transmission Scheduling

Note: The periodic transmission mode is used in the "Repeat Message State" and "Normal Operation State".

[SWS_ANM_00044] [If the Repeat Message State is entered ([SWS_ANM_00070]), the transmission of NM message shall be delayed by `nmMsgCycleOffset` after entering the Repeat Message State.]([RS_Nm_00044](#), [RS_Nm_00150](#))

[SWS_ANM_00046] [If transmission of NM messages has been started and the NM Message Cycle Timer expires, a NM message transmission shall be initiated.]([RS_Nm_00044](#))

[SWS_ANM_00047] [If the NM Message Cycle Timer expires it shall be restarted with `nmMsgCycleTime`.]([RS_Nm_00044](#), [RS_Nm_00150](#))

[SWS_ANM_00094] Immediate Nm Transmissions [Upon an active network request (by calling `ara::nm::NetworkHandle::SetNetworkRequestedState` with `kFullCom`) immediate Nm messages shall be sent according to PRS_Nm_00334 using:

- `nmImmediateNmCycleTime` as `NmImmediateNmCycleTime` and
- `nmImmediateNmTransmissions` as `NmImmediateNmTransmissions`

]([RS_Nm_02528](#))

7.7 Nm User Data Handling

Note: Although contained in the underlying Protocol Specification [3] currently no use case is seen for user data.

7.8 Partial Networking

7.8.1 Partial Network State Machine

The partial network state machine mentioned in Figure 7.1 is supposed to be implementation specific. Note: Although being implementation specific, the implemented behaviour shall conform to the Partial Networking requirements described in [3].

7.8.2 Rx Handling of NM messages

Note : Reception Handling of PNC bit vector as described in [3] is switched on/off by `NmCluster.nmPncParticipation`

[SWS_ANM_00051] [If `nmPncParticipation` is TRUE and the PNI bit in the received NM message is 1, the NM shall consider this messages as relevant to update the internal states. Otherwise the message is ignored.] (*RS_Nm_00150, RS_Nm_02519, RS_Nm_02546*)

[SWS_ANM_00052]{DRAFT} [If `nmPncParticipation` is set to TRUE and the PNI bit in the received NM message is 0, NM shall still process the user data information.]
()

[SWS_ANM_00053]{DRAFT} [The reset time value `PnResetTime` shall be configured by `pnResetTimer`.] ()

[SWS_ANM_00056]{DRAFT} [`pnResetTimer` shall be configured to a value greater than `nmMsgCycleTime`.] ()

7.8.3 Tx Handling of NM messages

[SWS_ANM_00054]{DRAFT} [`nmPncParticipation` shall enable/disable the reception and transmission of PN information.] ()

Note: The usage of the CBV is mandatory in case Partial Networking is used. This has to be ensured by configuration in the respective platform.

7.8.4 NM message Filter Algorithm

[SWS_ANM_00055] [The range (in bytes) that contains the Partial Network request information (PNC bit vector) in the received NM message shall be defined by PNC bit vector offset (`pncVectorOffset`) starting from byte 0 and PNC bit vector length (`pncVectorLength`).] ([RS_Nm_02519](#), [RS_Nm_02546](#))

Example:

- PNC bit vector Offset = 3
- PNC bit vector Length = 2

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 0	Control Bit Vector (default)							
Byte 1	User data 0							
Byte 2	User data 1							
Byte 3	PNC bit vector 0							
Byte 4	PNC bit vector 1							

Table 7.1: NM message layout example

In the above example only Byte 3 and Byte 4 of the NM message contain Partial Network request information.

Note: Every bit of the PNC bit vector represents one Partial Network. If the bit is set to 1 the Partial Network is requested. If the bit is set to 0 there is no request for this Partial Network.

[SWS_ANM_00081] [The Nm shall filter out messages containing Partial Network request information if they do not contain at least one bit set to 1 that corresponds to a Partial Network that is configured in a `NmNetworkHandle` and `allNmMessagesKeepAwake` is FALSE.] ([RS_Nm_00150](#), [RS_Nm_02527](#))

Note: When activated the Nm Message Filter Algorithm will filter out any Nm Message not containing at least one relevant Partial Network being requested (its Bit in the PN bit vector set to 1).

[SWS_ANM_00089] `allNmMessagesKeepAwake` [If no relevant Partial Network is requested in the received NM Message and `allNmMessagesKeepAwake` is TRUE the Message shall not be filtered out from further Rx Indication handling.] ([RS_Nm_00150](#))

Note: This is required to enable the ECU to stay awake on any kind of NM Message.

7.9 Functional Cluster Lifecycle

7.9.1 Startup

No special startup handling needed for Network Management. The environment is expected to take care that Network Management is running and able to serve communication requests as soon as network communication is needed. If and how the Nm is actually start up in advance depends on platform constraints like e.g. fast (re-)start etc.

7.9.2 Shutdown

[SWS_ANM_00090] Communication Shutdown [When a `SIGTERM` is received by NM, any active Network Requests via `ara::nm::NetworkHandle::SetNetworkRequestedState` shall be withdrawn and `ara::nm::NetworkHandle::GetNetworkState` shall return `kNoCom` until process termination.] (*RS_Nm_00043*)

Note: The NetworkHardware might be shutdown afterwards. It is assumed that State-Management takes care that no shutdown is initiated while Network Communication is still needed and that active Network Requests during shutdown are an exceptional situation.

8 API specification

8.1 Errors

8.1.1 NmErrc

[SWS_ANM_01111] Definition of API enum `ara::nm::NmErrc` [

Kind:	enumeration
Header file:	<code>#include "ara/nm/nm_error_domain.h"</code>
Forwarding header file:	<code>#include "ara/nm/nm_fwd.h"</code>
Scope:	namespace <code>ara::nm</code>
Symbol:	<code>NmErrc</code>
Underlying type:	<code>ara::core::ErrorDomain::CodeType</code>
Syntax:	<code>enum class NmErrc : ara::core::ErrorDomain::CodeType {...};</code>
Values:	<code>kServiceNotAvailable= 1</code> The connection to the daemon is currently lost.
Description:	Defines the errors for NetworkManagement. The enumeration values 0 - 255 are reserved for AUTOSAR assigned errors, the stack provider is free to define additional errors starting from 256.

]([RS_AP_00122](#), [RS_AP_00125](#), [RS_AP_00127](#), [RS_AP_00149](#))

[SWS_ANM_01109] Definition of API function `ara::nm::GetNmDomain` [

Kind:	function
Header file:	<code>#include "ara/nm/nm_error_domain.h"</code>
Scope:	namespace <code>ara::nm</code>
Symbol:	<code>GetNmDomain()</code>
Syntax:	<code>constexpr const ara::core::ErrorDomain & GetNmDomain () noexcept;</code>
Return value:	<code>const ara::core::ErrorDomain &</code> The global <code>NmErrorDomain</code> object.
Exception Safety:	<code>noexcept</code>
Thread Safety:	<code>re-entrant</code>
Description:	Returns the global <code>NmErrorDomain</code> object.

]([RS_AP_00119](#), [RS_AP_00120](#), [RS_AP_00132](#))

[SWS_ANM_01110] Definition of API function `ara::nm::MakeErrorCode` [

Kind:	function
Header file:	<code>#include "ara/nm/nm_error_domain.h"</code>
Scope:	namespace <code>ara::nm</code>
Symbol:	<code>MakeErrorCode(NmErrc code, ara::core::ErrorDomain::SupportDataType data)</code>
Syntax:	<code>constexpr ara::core::ErrorCode MakeErrorCode (NmErrc code, ara::core::ErrorDomain::SupportDataType data) noexcept;</code>
Parameters (in):	<code>code</code> Error code number. <code>data</code> Vendor defined data associated with the error.
Return value:	<code>ara::core::ErrorCode</code> An <code>ErrorCode</code> object.





Exception Safety:	noexcept
Thread Safety:	re-entrant
Description:	Creates an error code.

]([RS_AP_00119](#), [RS_AP_00120](#), [RS_AP_00121](#), [RS_AP_00132](#), [RS_AP_00135](#))

8.1.2 NmException class

[SWS_ANM_01100] Definition of API class `ara::nm::NmException` [

Kind:	class
Header file:	#include "ara/nm/nm_error_domain.h"
Forwarding header file:	#include "ara/nm/nm_fwd.h"
Scope:	namespace <code>ara::nm</code>
Symbol:	<code>NmException</code>
Base class:	<code>ara::core::Exception</code>
Syntax:	<code>class NmException : public ara::core::Exception {...};</code>
Description:	Exception type thrown by NetworkManagement.

]([RS_AP_00122](#), [RS_AP_00127](#))

[SWS_ANM_01102] Definition of API function `ara::nm::NmException::NmException` [

Kind:	function
Header file:	#include "ara/nm/nm_error_domain.h"
Scope:	class <code>ara::nm::NmException</code>
Symbol:	<code>NmException(ara::core::ErrorCode errorCode)</code>
Syntax:	<code>explicit NmException (ara::core::ErrorCode errorCode) noexcept;</code>
Parameters (in):	<code>errorCode</code> The error code.
Exception Safety:	noexcept
Description:	Construct a new NetworkManagement exception object containing an error code.

]([RS_AP_00120](#), [RS_AP_00121](#), [RS_AP_00132](#), [RS_AP_00135](#))

8.1.3 NmErrorDomain class

[SWS_ANM_01101] Definition of API class `ara::nm::NmErrorDomain` [

Kind:	class
Header file:	#include "ara/nm/nm_error_domain.h"
Forwarding header file:	#include "ara/nm/nm_fwd.h"
Scope:	namespace <code>ara::nm</code>





Symbol:	NmErrorDomain
Base class:	ara::core::ErrorDomain
Syntax:	<code>class NmErrorDomain final : public ara::core::ErrorDomain {...};</code>
Unique ID:	0x8000'0000'0000'0042
Description:	Defines the error domain for Nmsistency.

|(RS_AP_00122, RS_AP_00127, RS_AP_00140)

[SWS_ANM_01105] Definition of API function ara::nm::NmErrorDomain::NmErrorDomain [

Kind:	function
Header file:	#include "ara/nm/nm_error_domain.h"
Scope:	class ara::nm::NmErrorDomain
Symbol:	NmErrorDomain()
Syntax:	<code>NmErrorDomain () noexcept;</code>
Exception Safety:	noexcept
Thread Safety:	no
Description:	Creates a NmErrorDomain instance.

|(RS_AP_00119, RS_AP_00120, RS_AP_00132)

[SWS_ANM_01106] Definition of API function ara::nm::NmErrorDomain::Name [

Kind:	function	
Header file:	#include "ara/nm/nm_error_domain.h"	
Scope:	class ara::nm::NmErrorDomain	
Symbol:	Name()	
Syntax:	<code>const char * Name () const noexcept override;</code>	
Return value:	const char *	The name of the error domain.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Description:	Returns the name of the error domain.	

|(RS_AP_00119, RS_AP_00120, RS_AP_00132)

[SWS_ANM_01107] Definition of API function ara::nm::NmErrorDomain::Message [

Kind:	function	
Header file:	#include "ara/nm/nm_error_domain.h"	
Scope:	class ara::nm::NmErrorDomain	
Symbol:	Message(CodeType errorCode)	
Syntax:	<code>const char * Message (CodeType errorCode) const noexcept override;</code>	
Parameters (in):	errorCode	The error code number.
Return value:	const char *	The message associated with the error code.
Exception Safety:	noexcept	





Thread Safety:	no
Description:	Returns the message associated with the error code.

]([RS_AP_00119](#), [RS_AP_00120](#), [RS_AP_00121](#), [RS_AP_00132](#))

[SWS_ANM_01108] Definition of API function `ara::nm::NmErrorDomain::ThrowAsException` [

Kind:	function
Header file:	#include "ara/nm/nm_error_domain.h"
Scope:	class ara::nm::NmErrorDomain
Symbol:	ThrowAsException(const ara::core::ErrorCode &errorCode)
Syntax:	void ThrowAsException (const ara::core::ErrorCode &errorCode) const override;
Parameters (in):	errorCode The error to throw.
Return value:	None
Exception Safety:	not exception safe
Thread Safety:	no
Description:	Throws the exception associated with the error code.

]([RS_AP_00120](#), [RS_AP_00121](#))

[SWS_ANM_01103] Definition of API type `ara::nm::NmErrorDomain::Errc` [

Kind:	type alias
Header file:	#include "ara/nm/nm_error_domain.h"
Scope:	class ara::nm::NmErrorDomain
Symbol:	Errc
Syntax:	using Errc = NmErrc;
Description:	Alias for the error code value enumeration.

]([RS_AP_00122](#))

[SWS_ANM_01104] Definition of API type `ara::nm::NmErrorDomain::Exception` [

Kind:	type alias
Header file:	#include "ara/nm/nm_error_domain.h"
Scope:	class ara::nm::NmErrorDomain
Symbol:	Exception
Syntax:	using Exception = NmException;
Description:	Alias for the exception base class.

]([RS_AP_00122](#))

8.2 NetworkHandle class

[SWS_ANM_01000] Definition of API class `ara::nm::NetworkHandle` [

Kind:	class
Header file:	#include "ara/nm/network_handle.h"
Forwarding header file:	#include "ara/nm/nm_fwd.h"
Scope:	namespace <code>ara::nm</code>
Symbol:	<code>NetworkHandle</code>
Syntax:	<code>class NetworkHandle final {...};</code>
Description:	Class <code>NetworkHandle</code> is the access to the network handle referenced by the <code>InstanceSpecifier</code> . Provides information about network status per <code>NetworkHandle</code> . Intended to be only used by <code>StateManagement</code> !

]([RS_Nm_00050](#))

8.2.1 Special member functions

[SWS_ANM_01001]{DRAFT} Definition of API function `ara::nm::NetworkHandle::NetworkHandle` [

Kind:	function
Header file:	#include "ara/nm/network_handle.h"
Scope:	<code>class ara::nm::NetworkHandle</code>
Symbol:	<code>NetworkHandle(const ara::core::InstanceSpecifier &specifier)</code>
Syntax:	<code>explicit NetworkHandle (const ara::core::InstanceSpecifier &specifier) noexcept;</code>
Parameters (in):	specifier InstanceSpecifier to an PortPrototype of an NetworkHandle Interface
Exception Safety:	noexcept
Description:	<code>NetworkHandle</code> constructor.

]([RS_Nm_00050](#))

[SWS_ANM_01002]{DRAFT} Definition of API function `ara::nm::NetworkHandle::~~NetworkHandle` [

Kind:	function
Header file:	#include "ara/nm/network_handle.h"
Scope:	<code>class ara::nm::NetworkHandle</code>
Symbol:	<code>~NetworkHandle()</code>
Syntax:	<code>~NetworkHandle () noexcept;</code>
Exception Safety:	noexcept
Description:	<code>NetworkHandle</code> destructor.

]([RS_Nm_00050](#))

[SWS_ANM_01003]{DRAFT} Definition of API function ara::nm::NetworkHandle::NetworkHandle [

Kind:	function	
Header file:	#include "ara/nm/network_handle.h"	
Scope:	class ara::nm::NetworkHandle	
Symbol:	NetworkHandle(NetworkHandle &&stbc)	
Syntax:	NetworkHandle (NetworkHandle &&stbc) noexcept;	
Parameters (in):	stbc	The NetworkHandle object to be moved.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Description:	Move constructor for NetworkHandle.	

](RS_Nm_00050)

[SWS_ANM_01004]{DRAFT} Definition of API function ara::nm::NetworkHandle::operator= [

Kind:	function	
Header file:	#include "ara/nm/network_handle.h"	
Scope:	class ara::nm::NetworkHandle	
Symbol:	operator=(NetworkHandle &&stbc)	
Syntax:	NetworkHandle & operator= (NetworkHandle &&stbc) & noexcept;	
Parameters (in):	stbc	The NetworkHandle object to be moved.
Return value:	NetworkHandle &	The moved NetworkHandle object.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Description:	Move assignment operator for NetworkHandle. *	

](RS_Nm_00050)

[SWS_ANM_01005]{DRAFT} Definition of API function ara::nm::NetworkHandle::NetworkHandle [

Kind:	function	
Header file:	#include "ara/nm/network_handle.h"	
Scope:	class ara::nm::NetworkHandle	
Symbol:	NetworkHandle(const NetworkHandle &)	
Syntax:	NetworkHandle (const NetworkHandle &)=delete;	
Description:	The copy constructor for NetworkHandle shall not be used.	

](RS_Nm_00050)

[SWS_ANM_01006]{DRAFT} Definition of API function ara::nm::NetworkHandle::operator= [

Kind:	function
Header file:	#include "ara/nm/network_handle.h"
Scope:	class ara::nm::NetworkHandle
Symbol:	operator=(const NetworkHandle &)
Syntax:	NetworkHandle & operator= (const NetworkHandle &)=delete;
Description:	The copy assignment operator for NetworkHandle shall not be used.

]([RS_Nm_00050](#))

8.2.2 Class Specific Data Types

8.2.2.1 NetworkStateChangeNotifier

[SWS_ANM_01020]{DRAFT} Definition of API type ara::nm::NetworkHandle::NetworkStateChangeNotifier [

Kind:	type alias
Header file:	#include "ara/nm/network_handle.h"
Scope:	class ara::nm::NetworkHandle
Symbol:	NetworkStateChangeNotifier
Syntax:	using NetworkStateChangeNotifier = std::function<void(const NetworkStateType&)>;
Description:	A function wrapper for the handler function that gets called by the Communication Management software in case the network state has changed.

]([RS_Nm_00051](#))

8.2.2.2 NetworkStateType

[SWS_ANM_01358]{DRAFT} Definition of API enum ara::nm::NetworkHandle::NetworkStateType [

Kind:	enumeration	
Header file:	#include "ara/nm/network_handle.h"	
Forwarding header file:	#include "ara/nm/nm_fwd.h"	
Scope:	class ara::nm::NetworkHandle	
Symbol:	NetworkStateType	
Underlying type:	std::uint32_t	
Syntax:	enum class NetworkStateType : std::uint32_t {...};	
Values:	kNoCom= 0	not all PNCs, VLANs and/or physical channels associated to this instance of the NetworkState are in kFullCom.
	kFullCom= 1	all PNCs, VLANs and/or physical channels associated to this instance of the NetworkState are in kFullCom.





Description:	Enumeration of elementary supervision status. Scoped Enumeration of uint32_t.
---------------------	---

](RS_Nm_00051)

8.2.3 GetNetworkState

[SWS_ANM_01007]{DRAFT} Definition of API function `ara::nm::NetworkHandle::GetNetworkState` [

Kind:	function	
Header file:	#include "ara/nm/network_handle.h"	
Scope:	class <code>ara::nm::NetworkHandle</code>	
Symbol:	GetNetworkState()	
Syntax:	<code>ara::core::Result< NetworkStateType > GetNetworkState () const noexcept;</code>	
Return value:	<code>ara::core::Result< NetworkStateType ></code>	The network state of the corresponding PNC / VLAN / Physical Network
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Errors:	<code>NmErrc::kServiceNot Available</code>	The connection to the daemon is lost, so the application must implement an appropriate error strategy, such as taking over the previous NetworkState.
Description:	Method to obtain the current network state i.e. PNC / VLAN / Physical Network is currently active or not.	

](RS_Nm_00050, RS_Nm_00051)

8.2.4 GetNetworkRequestedState

[SWS_ANM_01008]{DRAFT} Definition of API function `ara::nm::NetworkHandle::GetNetworkRequestedState` [

Kind:	function	
Header file:	#include "ara/nm/network_handle.h"	
Scope:	class <code>ara::nm::NetworkHandle</code>	
Symbol:	GetNetworkRequestedState()	
Syntax:	<code>ara::core::Result< NetworkStateType > GetNetworkRequestedState () const noexcept;</code>	
Return value:	<code>ara::core::Result< NetworkStateType ></code>	The network requested state of the corresponding PNC / VLAN / Physical Network
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Errors:	<code>NmErrc::kServiceNot Available</code>	The connection to the daemon is lost, so the application must implement an appropriate error strategy, such as taking over the previous NetworkState.
Description:	Method to obtain the current network requested state i.e. if the PNC / VLAN / Physical Network is currently requested or released.	

]([RS_Nm_00050](#), [RS_Nm_00051](#))

8.2.5 SetNetworkRequestedState

[SWS_ANM_01009]{DRAFT} Definition of API function `ara::nm::NetworkHandle::SetNetworkRequestedState` [

Kind:	function	
Header file:	#include "ara/nm/network_handle.h"	
Scope:	class <code>ara::nm::NetworkHandle</code>	
Symbol:	SetNetworkRequestedState(NetworkStateType networkState)	
Syntax:	<code>ara::core::Result< void > SetNetworkRequestedState (NetworkStateType networkState) noexcept;</code>	
Parameters (in):	networkState	The request state to be set.
Return value:	<code>ara::core::Result< void ></code>	–
Exception Safety:	noexcept	
Thread Safety:	unsafe	
Errors:	NmErrc::kServiceNot Available	The connection to the daemon is lost, so the application needs to retry later.
Description:	A method that can be used to set a new network requested state. Setting a new new network requested state will request or release the PNC / VLAN / Physical Network.	

]([RS_Nm_00045](#), [RS_Nm_02513](#))

8.2.6 RegisterNetworkStateChangeNotifier

[SWS_ANM_01010] Definition of API function `ara::nm::NetworkHandle::RegisterNetworkStateChangeNotifier` [

Kind:	function	
Header file:	#include "ara/nm/network_handle.h"	
Scope:	class <code>ara::nm::NetworkHandle</code>	
Symbol:	RegisterNetworkStateChangeNotifier(NetworkStateChangeNotifier notifier)	
Syntax:	<code>ara::core::Result< void > RegisterNetworkStateChangeNotifier (NetworkStateChangeNotifier notifier) noexcept;</code>	
Parameters (in):	notifier	The function to unregister.
Return value:	<code>ara::core::Result< void ></code>	–
Exception Safety:	noexcept	
Thread Safety:	unsafe	
Description:	Register a notifier function which is called if the current network state is changed (i.e. changed into FullCom or into NoCom). A maximum of one notifier can be registered. Every further registration overwrites the current registration.	

]([RS_Nm_00051](#))

[SWS_ANM_01015] Definition of API function `ara::nm::NetworkHandle::RegisterNetworkStateChangeNotifier` [

Kind:	function	
Header file:	#include "ara/nm/network_handle.h"	
Scope:	class <code>ara::nm::NetworkHandle</code>	
Symbol:	<code>RegisterNetworkStateChangeNotifier(NetworkStateChangeNotifier notifier, ExecutorT &&executor)</code>	
Syntax:	<pre>template <typename ExecutorT> ara::core::Result< void > RegisterNetworkStateChangeNotifier (Network StateChangeNotifier notifier, ExecutorT &&executor) noexcept;</pre>	
Parameters (in):	notifier	The function to unregister.
	executor	Executioner object in which any asynchronous computation spawn by <code>RegisterNetworkStateChangeNotifier</code> shall be invoked.
Return value:	<code>ara::core::Result< void ></code>	-
Exception Safety:	noexcept	
Thread Safety:	unsafe	
Description:	<p>Register a notifier function which is called if the current network state is changed (i.e. changed into FullCom or into NoCom) using an execution context executor.</p> <p>A maximum of one notifier can be registered. Every further registration overwrites the current registration.</p>	

]([RS_Nm_00051](#))

8.2.7 UnregisterNetworkStateChangeNotifier

[SWS_ANM_01011]{DRAFT} Definition of API function `ara::nm::NetworkHandle::UnregisterNetworkStateChangeNotifier` [

Kind:	function	
Header file:	#include "ara/nm/network_handle.h"	
Scope:	class <code>ara::nm::NetworkHandle</code>	
Symbol:	<code>UnregisterNetworkStateChangeNotifier()</code>	
Syntax:	<pre>void UnregisterNetworkStateChangeNotifier () noexcept;</pre>	
Return value:	None	
Exception Safety:	noexcept	
Thread Safety:	unsafe	
Description:	Unregister a notifier function which is called if a current network state is changed.	

]([RS_Nm_00051](#))

8.2.8 RegisterNetworkRequestedStateChangeNotifier

[SWS_ANM_01012] Definition of API function `ara::nm::NetworkHandle::RegisterNetworkRequestedStateChangeNotifier` [

Kind:	function	
Header file:	#include "ara/nm/network_handle.h"	
Scope:	class <code>ara::nm::NetworkHandle</code>	
Symbol:	RegisterNetworkRequestedStateChangeNotifier(NetworkStateChangeNotifier notifier)	
Syntax:	<pre>ara::core::Result< void > RegisterNetworkRequestedStateChangeNotifier (NetworkStateChangeNotifier notifier) noexcept;</pre>	
Parameters (in):	notifier	The function to register.
Return value:	<code>ara::core::Result< void ></code>	–
Exception Safety:	noexcept	
Thread Safety:	unsafe	
Description:	Register a notifier function which is called if the current network requested state is changed (i.e. changed into FullCom or into NoCom). A maximum of one notifier can be registered. Every further registration overwrites the current registration.	

]([RS_Nm_00051](#))

[SWS_ANM_01014] Definition of API function `ara::nm::NetworkHandle::RegisterNetworkRequestedStateChangeNotifier` [

Kind:	function	
Header file:	#include "ara/nm/network_handle.h"	
Scope:	class <code>ara::nm::NetworkHandle</code>	
Symbol:	RegisterNetworkRequestedStateChangeNotifier(NetworkStateChangeNotifier notifier, ExecutorT &&executor)	
Syntax:	<pre>template <typename ExecutorT> ara::core::Result< void > RegisterNetworkRequestedStateChangeNotifier (NetworkStateChangeNotifier notifier, ExecutorT &&executor) noexcept;</pre>	
Parameters (in):	notifier	The function to register.
	executor	Executioner object in which any asynchronous computation spawn by RegisterNetworkRequestedStateChangeNotifier shall be invoked.
Return value:	<code>ara::core::Result< void ></code>	–
Exception Safety:	noexcept	
Thread Safety:	unsafe	
Description:	Register a notifier function which is called if the current network requested state is changed (i.e. changed into FullCom or into NoCom) using an execution context executor. A maximum of one notifier can be registered. Every further registration overwrites the current registration.	

]([RS_Nm_00051](#))

8.2.9 UnregisterNetworkRequestedStateChangeNotifier

[SWS_ANM_01013]{DRAFT} Definition of API function `ara::nm::NetworkHandle::UnregisterNetworkRequestedStateChangeNotifier` [

Kind:	function
Header file:	#include "ara/nm/network_handle.h"
Scope:	class ara::nm::NetworkHandle
Symbol:	UnregisterNetworkRequestedStateChangeNotifier()
Syntax:	<code>void UnregisterNetworkRequestedStateChangeNotifier () noexcept;</code>
Return value:	None
Exception Safety:	noexcept
Thread Safety:	unsafe
Description:	Unregister a notifier function which is called if a current network requested state is changed. .

]([RS_Nm_00051](#))

9 Service Interfaces

Network Management does not contain any Service Interfaces.

A Mentioned Manifest Elements

For the sake of completeness, this chapter contains a set of class tables representing meta-classes mentioned in the context of this document but which are not contained directly in the scope of describing specific meta-model semantics.

Chapter is generated.

Class	EthernetCommunicationConnector			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::Fibex4Ethernet::EthernetTopology			
Note	Ethernet specific attributes to the CommunicationConnector.			
Base	<i>ARObject, CommunicationConnector, Identifiable, MultilanguageReferrable, Referrable</i>			
Aggregated by	EculInstance.connector, MachineDesign.communicationConnector			
Attribute	Type	Mult.	Kind	Note
apApplicationEndpoint	ApApplicationEndpoint	*	aggr	Collection of Application Addresses that are used on the CommunicationConnector.
canXIProps	CanXIProps	*	ref	If the Ethernet frames handled by this Ethernet CommunicationConnector are tunneled through CAN XL, then this reference shall refer the CanXIProps which contains the specific configuration parameters of the CAN XL controller of the physical CAN XL connection to be used for tunneling.
maximumTransmissionUnit	PositiveInteger	0..1	attr	This attribute specifies the maximum transmission unit in bytes.
neighborCacheSize	PositiveInteger	0..1	attr	This attribute specifies the size of neighbor cache or ARP table in units of entries.
pathMtuEnabled	Boolean	0..1	attr	If enabled the IPv4/IPv6 processes incoming ICMP "Packet Too Big" messages and stores a MTU value for each destination address.
pathMtuTimeout	TimeValue	0..1	attr	If this value is >0 the IPv4/IPv6 will reset the MTU value stored for each destination after n seconds.
unicastNetworkEndpoint	NetworkEndpoint	*	ref	Network Endpoint that defines the IPAddress of the machine.

Table A.1: EthernetCommunicationConnector

Class	MachineDesign			
Package	M2::AUTOSARTemplates::AdaptivePlatform::SystemDesign			
Note	This meta-class represents the ability to define requirements on a Machine in the context of designing a system. Tags: atp.recommendedPackage=MachineDesigns			
Base	<i>ARElement, ARObject, AtpClassifier, AtpFeature, AtpStructureElement, CollectableElement, FibexElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable, UploadableDesignElement, UploadablePackageElement</i>			
Aggregated by	ARPackage.element, <i>AtpClassifier.atpFeature</i>			
Attribute	Type	Mult.	Kind	Note
accessControl	AccessControlEnum	0..1	attr	This attribute defines how the access restriction to the Service Instance is defined.





Class	MachineDesign			
communicationConnector	CommunicationConnector	*	aggr	This aggregation defines the network connection of the machine. Stereotypes: atpSplitable Tags: atp.Splitkey=communicationConnector.shortName
communicationController	CommunicationController	*	aggr	CommunicationControllers of the Machine that are used for description of 10-Base-T1S topologies Stereotypes: atpSplitable Tags: atp.Splitkey=communicationController.shortName
ethIpProps	EthIpProps	*	ref	Machine specific IP attributes.
pncPrepareSleepTimer	TimeValue	0..1	attr	Time in seconds the PNC state machine shall wait in PNC_PREPARE_SLEEP.
pnResetTimer	TimeValue	0..1	attr	Specifies the runtime of the reset timer in seconds. This reset time is valid for the reset of PN requests.
serviceDiscoveryConfig	ServiceDiscoveryConfiguration	*	aggr	Set of service discovery configuration settings that are defined on the machine for individual CommunicationConnectors. Stereotypes: atpSplitable Tags: atp.Splitkey=serviceDiscoveryConfig
tcplplcmpProps	EthTcplplcmpProps	*	ref	Machine specific ICMP (Internet Control Message Protocol) attributes
tcplpProps	EthTcplpProps	*	ref	Machine specific Tcplp Stack attributes.

Table A.2: MachineDesign

Class	NmCluster (abstract)			
Package	M2::AUTOSARTemplates::SystemTemplate::NetworkManagement			
Note	Set of NM nodes coordinated with use of the NM algorithm.			
Base	ARObject, Identifiable, MultilanguageReferrable, Referrable			
Subclasses	CanNmCluster, FlexrayNmCluster, UdpNmCluster			
Aggregated by	NmConfig.nmCluster			
Attribute	Type	Mult.	Kind	Note
communicationCluster	CommunicationCluster	0..1	ref	Association to a CommunicationCluster in the topology description.
nmNode	NmNode	*	aggr	Collection of NmNodes of the NmCluster. atpVariation: Derived, because NmNode can be variable. Stereotypes: atpSplitable; atpVariation Tags: atp.Splitkey=nmNode.shortName, nmNode.variationPoint.shortLabel vh.latestBindingTime=postBuild
nmPncParticipation	Boolean	0..1	attr	Defines whether this NmCluster contributes to the partial network mechanism.
pncClusterVectorLength	PositiveInteger	0..1	attr	Optionally defines the length of the PNC Vector per CommunicationCluster (and VLAN in case of UdpNm). If not defined then System.pncVectorLength applies. Should only make the PNC Vector shorter (or same length as defined in System.pncVectorLength).

Table A.3: NmCluster

Class	NmNetworkHandle			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::AdaptiveModuleImplementation			
Note	Group of partialNetworks and/or VLANs that can be controlled collectively.			
Base	ARObject, Referrable			
Aggregated by	NmInstantiation.networkHandle			
Attribute	Type	Mult.	Kind	Note
partialNetwork	PncMappingIdent	*	ref	Reference to a Partial Network that is included in the NmNetworkHandle. Stereotypes: atpSplitable Tags: atp.Splitkey=partialNetwork
vlan	EthernetCommunicationConnector	*	ref	Reference to a VLAN that is included in the NmNetworkHandle.

Table A.4: NmNetworkHandle

Class	NmNode (abstract)			
Package	M2::AUTOSARTemplates::SystemTemplate::NetworkManagement			
Note	The linking of NmEcus to NmClusters is realized via the NmNodes.			
Base	ARObject, Identifiable, MultilanguageReferrable, Referrable			
Subclasses	CanNmNode, FlexrayNmNode, UdpNmNode			
Aggregated by	NmCluster.nmNode			
Attribute	Type	Mult.	Kind	Note
machine	MachineDesign	0..1	ref	Reference to the machine that contains the NmNode.
nmNodeIid	Integer	0..1	attr	Node identifier of local NmNode. Shall be unique in the NmCluster.

Table A.5: NmNode

Class	PncMapping			
Package	M2::AUTOSARTemplates::SystemTemplate::PncMapping			
Note	Describes a mapping between one or several Virtual Function Clusters onto Partial Network Clusters. A Virtual Function Cluster is realized by a PortGroup. A Partial Network Cluster is realized by one or more ServiceInstances.			
Base	ARObject, Describable			
Aggregated by	SystemMapping.pncMapping			
Attribute	Type	Mult.	Kind	Note
ident	PncMappingIdent	0..1	aggr	This adds the ability to become referrable to PncMapping.
physicalChannel	PhysicalChannel	*	ref	This reference maps the partial network to a communication channel. Stereotypes: atpSplitable Tags: atp.Splitkey=physicalChannel





Class	PncMapping			
pncConsumedProvidedServiceInstanceGroup	ConsumedProvidedServiceInstanceGroup	*	ref	ConsumedProvidedServiceInstanceGroup used in a Partial Network Cluster. This reference is optional, since this could be used for starting and stopping ConsumedProvidedServiceInstanceGroup according the requested partial network, but is not necessarily needed. Stereotypes: atpSplittable; atpVariation Tags: atp.Splitkey=pncConsumedProvidedServiceInstanceGroup.consumedProvidedServiceInstanceGroup, pncConsumedProvidedServiceInstanceGroup.variationPoint.shortLabel, vh.latestBindingTime=postBuild
pncIdentifier	PositiveInteger	0..1	attr	Identifier of the Partial Network Cluster. This number represents the absolute bit position of this Partial Network Cluster in the NM Pdu.
pncWakeupEnable	Boolean	0..1	attr	If this parameter is available and set to true then this PNC will be woken up as soon as a channel wakeup occurs on a channel where this PNC is assigned to. This is ensured by adding this PNC to the corresponding channel wakeup sources during upstream mapping.
serviceInstance	AdaptivePlatformServiceInstance	*	ref	Reference to ServiceInstances that are participating in a Partial Network Cluster.
shortLabel	Identifier	0..1	attr	This attribute specifies an identifying shortName for the PncMapping. It shall be unique in the System scope.
vfc	PortGroup	*	iref	Virtual Function Cluster to be mapped onto a Partial Network Cluster. This reference is optional in case that the System Description doesn't use a complete Software Component Description (VFB View). This supports the inclusion of legacy systems. InstanceRef implemented by: PortGroupInSystemInstanceRef

Table A.6: PncMapping

Class	PncMappingIdent			
Package	M2::AUTOSARTemplates::SystemTemplate::PncMapping			
Note	This meta-class is created to add the ability to become the target of a reference to the non-Referrable PncMapping.			
Base	ARObject, Referrable			
Aggregated by	PncMapping.ident			
Attribute	Type	Mult.	Kind	Note
-	-	-	-	-

Table A.7: PncMappingIdent

Class	System
Package	M2::AUTOSARTemplates::SystemTemplate
Note	The top level element of the System Description. Tags: atp.recommendedPackage=Systems
Base	ARElement, ARObject, AtpClassifier, AtpFeature, AtpStructureElement, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable, UploadableDesignElement, UploadablePackageElement





Class	System			
Aggregated by	ARPackage.element, <i>AtpClassifier.atpFeature</i>			
Attribute	Type	Mult.	Kind	Note
fibexElement	FibexElement	*	ref	Reference to ASAM FIBEX elements specifying Communication and Topology. All Fibex Elements used within a System Description shall be referenced from the System Element. atpVariation: In order to describe a product-line, all Fibex Elements can be optional. Stereotypes: atpSplitable; atpVariation Tags: atp.Splitkey=fibexElement.fibexElement, fibexElement.variationPoint.shortLabel vh.latestBindingTime=postBuild
interpolationRoutineMappingSet	InterpolationRoutineMappingSet	*	ref	This reference identifies the InterpolationRoutineMapping Sets that are relevant in the context of the enclosing System.
mapping	SystemMapping	*	aggr	Aggregation of all mapping aspects relevant in the System Description. Stereotypes: atpSplitable; atpVariation Tags: atp.Splitkey=mapping.shortName, mapping.variationPoint.shortLabel vh.latestBindingTime=postBuild
pncVectorLength	PositiveInteger	0..1	attr	Length of the partial networking request release information vector (in bytes).
pncVectorOffset	PositiveInteger	0..1	attr	Absolute offset (with respect to the NM-PDU) of the partial networking request release information vector that is defined in bytes as an index starting with 0.
rootSoftwareComposition	RootSwCompositionPrototype	0..1	aggr	Aggregation of the root software composition, containing all software components in the System in a hierarchical structure. This element is not required when the System description is used for a network-only use-case. atpVariation: The RootSwCompositionPrototype can vary. Stereotypes: atpSplitable; atpVariation Tags: atp.Splitkey=rootSoftwareComposition.shortName, rootSoftwareComposition.variationPoint.shortLabel vh.latestBindingTime=systemDesignTime
systemVersion	RevisionLabelString	0..1	attr	Version number of the System Description.

Table A.8: System

Class	UdpNmCluster			
Package	M2::AUTOSARTemplates::SystemTemplate::NetworkManagement			
Note	Udp specific NmCluster attributes			
Base	<i>ARObject</i> , <i>Identifiable</i> , <i>MultilanguageReferrable</i> , <i>NmCluster</i> , <i>Referrable</i>			
Aggregated by	NmConfig.nmCluster			
Attribute	Type	Mult.	Kind	Note
networkConfiguration	UdpNmNetworkConfiguration	0..1	aggr	Configuration of a UDP port and UDP multicast IP address of the Nm communication on a VLAN.





Class	UdpNmCluster			
nmCbvPosition	Integer	0..1	attr	Defines the position of the control bit vector within the Nm Pdu (Byte position). If this attribute is not configured, the Control Bit Vector is not used.
nmImmediateNmCycleTime	TimeValue	0..1	attr	Defines the immediate NmPdu cycle time in seconds which is used for nmImmediateNmTransmissions NmPdu transmissions. This attribute is only valid if nmImmediateNmTransmissions is greater one.
nmImmediateNmTransmissions	PositiveInteger	0..1	attr	Defines the number of immediate NmPdus which shall be transmitted. If the value is zero no immediate NmPdus are transmitted. The cycle time of immediate NmPdus is defined by nmImmediateNmCycleTime.
nmMsgCycleTime	TimeValue	0..1	attr	Period of a NmPdu in seconds. It determines the periodic rate in the periodic transmission mode with bus load reduction and is the basis for transmit scheduling in the periodic transmission mode without bus load reduction.
nmNetworkTimeout	TimeValue	0..1	attr	Network Timeout for NmPdus in seconds. It denotes the time how long the UdpNm shall stay in the Network Mode before transition into Prepare Bus-Sleep Mode shall take place.
nmNidPosition	Integer	0..1	attr	Defines the byte position of the source node identifier within the NmPdu. If this attribute is not configured, the Node Identification is not used.
nmRepeatMessageTime	TimeValue	0..1	attr	Timeout for Repeat Message State in seconds. Defines the time how long the NM shall stay in the Repeat Message State.
nmUserDataLength	Integer	0..1	attr	Defines the length in bytes of the user data contained in the Nm message. User data excludes the PNC bit vector.
nmUserDataOffset	PositiveInteger	0..1	attr	Specifies the offset (in bytes) of the user data information in the NM message. User data excludes the PNC bit vector.
nmWaitBusSleepTime	TimeValue	0..1	attr	Timeout for bus calm down phase in seconds. It denotes the time how long the CanNm shall stay in the Prepare Bus-Sleep Mode before transition into Bus-Sleep Mode shall take place.
vlan	EthernetPhysicalChannel	0..1	ref	Reference to the vlan (represented by the Ethernet PhysicalChannel) this UdpNmCluster shall apply to.

Table A.9: UdpNmCluster

Class	UdpNmNode			
Package	M2::AUTOSARTemplates::SystemTemplate::NetworkManagement			
Note	Udp specific NM Node attributes.			
Base	ARObject, Identifiable, MultilanguageReferrable, NmNode, Referrable			
Aggregated by	NmCluster.nmNode			
Attribute	Type	Mult.	Kind	Note
allNmMessagesKeepAwake	Boolean	0..1	attr	Specifies if Nm drops irrelevant NM PDUs. false: Only NM PDUs with a Partial Network Information Bit (PNI) = true and containing a Partial Network request for this ECU trigger the standard RX indication handling and thus keep the ECU awake true: Every NM PDU triggers the standard RX indication handling and keeps the ECU awake
communicationConnector	EthernetCommunicationConnector	0..1	ref	Reference to the CommunicationConnector that represents the UdpNmNode in the topology description.





Class	UdpNmNode			
nmMsgCycle Offset	TimeValue	0..1	attr	Node specific time offset in the periodic transmission node. It determines the start delay of the transmission. Specified in seconds.
nmPnHandle MultipleNetwork Requests	Boolean	0..1	attr	Specifies if NM performs an additional transition from Network Mode to Repeat Message State (true) or not (false).

Table A.10: UdpNmNode

B History of Specification Items

Please note that the lists in this chapter also include specification items that have been removed from the specification in a later version. These specification items do not appear as hyperlinks in the document.

B.1 Specification Item History of this document compared to AUTOSAR R22-11.

B.1.1 Added Specification Items in R23-11

Number	Heading
[SWS_ANM_00052]	
[SWS_ANM_00053]	
[SWS_ANM_00054]	
[SWS_ANM_00056]	
[SWS_ANM_00095]	
[SWS_ANM_01000]	Definition of API class <code>ara::nm::NetworkHandle</code>
[SWS_ANM_01001]	Definition of API function <code>ara::nm::NetworkHandle::NetworkHandle</code>
[SWS_ANM_01002]	Definition of API function <code>ara::nm::NetworkHandle::~~NetworkHandle</code>
[SWS_ANM_01003]	Definition of API function <code>ara::nm::NetworkHandle::NetworkHandle</code>
[SWS_ANM_01004]	Definition of API function <code>ara::nm::NetworkHandle::operator=</code>
[SWS_ANM_01005]	Definition of API function <code>ara::nm::NetworkHandle::NetworkHandle</code>
[SWS_ANM_01006]	Definition of API function <code>ara::nm::NetworkHandle::operator=</code>
[SWS_ANM_01007]	Definition of API function <code>ara::nm::NetworkHandle::GetNetworkState</code>
[SWS_ANM_01008]	Definition of API function <code>ara::nm::NetworkHandle::GetNetworkRequestedState</code>
[SWS_ANM_01009]	Definition of API function <code>ara::nm::NetworkHandle::SetNetworkRequestedState</code>
[SWS_ANM_01010]	Definition of API function <code>ara::nm::NetworkHandle::RegisterNetworkStateChangeNotifier</code>
[SWS_ANM_01011]	Definition of API function <code>ara::nm::NetworkHandle::UnregisterNetworkStateChangeNotifier</code>
[SWS_ANM_01012]	Definition of API function <code>ara::nm::NetworkHandle::RegisterNetworkRequestedStateChangeNotifier</code>
[SWS_ANM_01013]	Definition of API function <code>ara::nm::NetworkHandle::UnregisterNetworkRequestedStateChangeNotifier</code>
[SWS_ANM_01014]	Definition of API function <code>ara::nm::NetworkHandle::RegisterNetworkRequestedStateChangeNotifier</code>
[SWS_ANM_01015]	Definition of API function <code>ara::nm::NetworkHandle::RegisterNetworkStateChangeNotifier</code>



△

Number	Heading
[SWS_ANM_01020]	Definition of API type ara::nm::NetworkHandle::NetworkStateChangeNotifier
[SWS_ANM_01100]	Definition of API class ara::nm::NmException
[SWS_ANM_01101]	Definition of API class ara::nm::NmErrorDomain
[SWS_ANM_01102]	Definition of API function ara::nm::NmException::NmException
[SWS_ANM_01103]	Definition of API type ara::nm::NmErrorDomain::Errc
[SWS_ANM_01104]	Definition of API type ara::nm::NmErrorDomain::Exception
[SWS_ANM_01105]	Definition of API function ara::nm::NmErrorDomain::NmErrorDomain
[SWS_ANM_01106]	Definition of API function ara::nm::NmErrorDomain::Name
[SWS_ANM_01107]	Definition of API function ara::nm::NmErrorDomain::Message
[SWS_ANM_01108]	Definition of API function ara::nm::NmErrorDomain::ThrowAsException
[SWS_ANM_01109]	Definition of API function ara::nm::GetNmDomain
[SWS_ANM_01110]	Definition of API function ara::nm::MakeErrorCode
[SWS_ANM_01111]	Definition of API enum ara::nm::NmErrc
[SWS_ANM_01358]	Definition of API enum ara::nm::NetworkHandle::NetworkStateType

Table B.1: Added Specification Items in R23-11

B.1.2 Changed Specification Items in R23-11

Number	Heading
[SWS_ANM_00022]	
[SWS_ANM_00029]	
[SWS_ANM_00051]	
[SWS_ANM_00062]	
[SWS_ANM_00070]	
[SWS_ANM_00083]	
[SWS_ANM_00084]	
[SWS_ANM_00086]	
[SWS_ANM_00090]	Communication Shutdown
[SWS_ANM_00094]	Immediate Nm Transmissions

Table B.2: Changed Specification Items in R23-11

B.1.3 Deleted Specification Items in R23-11

Number	Heading
[SWS_ANM_00093]	
[SWS_ANM_91000]	
[SWS_ANM_91001]	

Table B.3: Deleted Specification Items in R23-11

B.2 Specification Item History of this document compared to AUTOSAR R21-11.

B.2.1 Added Specification Items in R22-11

none

B.2.2 Changed Specification Items in R22-11

Number	Heading
[SWS_ANM_00055]	
[SWS_ANM_00083]	
[SWS_ANM_00093]	
[SWS_ANM_91000]	
[SWS_ANM_91001]	

Table B.4: Changed Specification Items in R22-11

B.2.3 Deleted Specification Items in R22-11

Number	Heading
[SWS_ANM_00004]	
[SWS_ANM_00028]	

Table B.5: Deleted Specification Items in R22-11