Document Title	Specification of Network Management Interface
Document Category	Software specification (SWS)
Document UID	228
Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R26-11

Document Change History			
Date	**Release**	**Changed by**	**Description**
2026-11-26	R26-11	AUTOSAR Release Management	• Added support for Dynamic PNC for CAN • Removed Nm_GetVersionInfo() API • Removed RS NM references • Editorial changes
2025-11-27	R25-11	AUTOSAR Release Management	• Shutdown clarification for actively coordinated channels • Editorial changes
2024-11-27	R24-11	AUTOSAR Release Management	• Editorial changes
2023-11-23	R23-11	AUTOSAR Release Management	• NM harmonization • Editorial changes
2022-11-24	R22-11	AUTOSAR Release Management	• Refined Partial Network Cluster handling • Editorial changes
2021-11-25	R21-11	AUTOSAR Release Management	• Reworked Partial Network functionality and Partial Network Cluster handling • Caveats that were no real requirements were change to notes • Editorial changes

△

2020-11-30	R20-11	AUTOSAR Release Management	• Support for synchronized PNC shutdown functionality • Introduction of Dynamic PNC-to-channel-mapping and PNC Learning algorithm • Added limitation regarding Nm Coordinator functionality when using 10BASE-T1S in combination with PLCA media access • Support for passing NM state change to SwC • Editorial changes
2019-11-28	R19-11	AUTOSAR Release Management	• Minor changes • Multicore Distribution support (draft) added • Changed Document Status from Final to published
2018-10-31	4.4.0	AUTOSAR Release Management	• Removed LinNM from the architecture • Removed obsolete elements • Header File Cleanup
2017-12-08	4.3.1	AUTOSAR Release Management	• minor corrections / clarifications / editorial changes; For details please refer to the ChangeDocumentation
2016-11-30	4.3.0	AUTOSAR Release Management	• Add functionality for synchronizing channel A and channel B • removed dependencies of ComMChannels to each other in respect to NMVariants • minor corrections / clarifications / editorial changes; For details please refer to the ChangeDocumentation
2015-07-31	4.2.2	AUTOSAR Release Management	• "'Coordination algorithm"' and "'Coordinated shutdown"' redefined • Make the CarWakeup feature available • Debugging support marked as obsolete • Editorial changes

▽

△

2014-10-31	4.2.1	AUTOSAR Release Management	• Corrections on the requirement tracing • Clarification at use of callback versus callout • Editorial changes
2014-03-31	4.1.3	AUTOSAR Release Management	• Rework of wakeup and abortion of coordinated shutdown • Rework of coordination of nested sub-busses
2013-10-31	4.1.2	AUTOSAR Release Managment	• Remove DEM usage • Correct multiplicity and dependency of configuration parameter • Corrections on RemoteSleepIndication feature • Corrections on MainFunction and coordinated shutdown • Formal correction on REQ Tags • Editorial changes • Removed chapter(s) on change documentation
2013-03-15	4.1.1	AUTOSAR Administration	• Introduction J1939Nm • Merged and corrected calculation of delay timer for Coordination Algorithm • Correction of parametrization and Services for Coordinator Synchonization Algorithm • Moved Nm_Passive_Mode_Enabled Parameter back to global container
2011-12-22	4.0.3	AUTOSAR Administration	• NmMultipleChannelsEnabled removed • Added Mandatory Interfaces provided by ComM to Chapter 8.6.1 • move NmPassiveMode • Enabled form global configuration to channel configuration • Removed Nm_ReturnType • Fixed some min and max values of FloatPAramDef configuration parameters

▽

▽

△

			△ • Added support of NmCarWakup-Feature • Added support of coordinated shutdown of nested sub-busses
2009-12-18	4.0.1	AUTOSAR Administration	• Release check added • DET Error Code for false Pointer added • ChannelID harmonized in COM-Stack • Nm-State-changes in Userdata via NmIf
2010-02-02	3.1.4	AUTOSAR Administration	• Remove explicit support for OSEK NM from specification • NM Coordinator functionality reworked (chapter 7.2 and 7.2.4) • Debugging functionality added • Link time configuration variant introduced • Legal disclaimer revised
2008-08-13	3.1.1	AUTOSAR Administration	• Legal disclaimer revised
2007-12-21	3.0.1	AUTOSAR Administration	• 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.

Table of Contents

1 Introduction and functional overview 11

2 Acronyms and Abbreviations 12

3 Related documentation 14
3.1 Input documents . . . 14
3.2 Related specification . . . 14

4 Constraints and assumptions 15
4.1 Limitations . . . 15
4.2 Specific limitations of the current release . . . 15
4.3 Applicability to automotive domains . . . 16

5 Dependencies to other modules 17
5.1 Interfaces to modules . . . 17
5.1.1 ComM, CanNm, J1939Nm, FrNm, UdpNm, generic bus specific NM layers and CDD . . . 19
5.1.2 Error handling modules . . . 19
5.1.3 BSW Scheduler . . . 19
5.2 File structure . . . 20
5.2.1 Code file structure . . . 20
5.2.2 Header file structure . . . 20

6 Requirements traceability 21

7 Functional specification 24
7.1 Base functionality . . . 24
7.2 NM Coordinator functionality . . . 25
7.2.1 Applicability of the NM Coordinator functionality . . . 25
7.2.2 Keeping coordinated busses alive . . . 26
7.2.3 Shutdown of coordinated busses . . . 27
7.2.4 Coordination of nested sub-busses . . . 29
7.2.5 Calculation of shutdown timers . . . 33
7.2.6 Synchronization Use Case 1 - Synchronous command . . . 33
7.2.7 Synchronization Use Case 2 - Synchronous initiation . . . 34
7.2.8 Synchronization Use Case 3 - Synchronous network sleep . . . 34
7.2.8.1 Examples . . . 35
7.3 Wakeup and abortion of the coordinated shutdown . . . 37
7.3.1 External network wakeup . . . 37
7.3.2 Coordinated wakeup . . . 37
7.3.3 Abortion of the coordinated shutdown . . . 37
7.4 Partial Network functionality . . . 39
7.4.1 PNC Request Bit Vector filter algorithm . . . 40
7.4.2 Aggregation of PNC requests . . . 41

7.4.2.1 Aggregation of internal and external Partial Network Cluster . . 41
7.4.2.2 Aggregation of external Partial Network Cluster 44
7.4.3 EIRA / ERA state and PNC reset timer handling 45
7.4.4 Synchronized PNC shutdown functionality 48
7.5 Prerequisites of bus specific Network Management modules 51
7.5.1 Prerequisites for basic functionality 51
7.5.2 Prerequisites for NM Coordinator functionality 52
7.5.3 Prerequisites of Partial Network functionality 55
7.5.3.1 Prerequisite for aggregation of PNC requests 55
7.5.3.2 Prerequisites for synchronized PNC shutdown functionality . . . 55
7.5.4 Configuration of global parameters for bus specific networks 55
7.6 NM_BUSNM_LOCALNM 56
7.7 Multicore Distribution 56
7.8 Additional Functionality 56
7.8.1 Nm_CarWakeUpIndication 56
7.8.2 Nm_StateChangeNotification 57
7.9 Error classification 58
7.9.1 Development Errors 58
7.9.2 Runtime Errors 58
7.9.3 Production Errors 59
8 API specification 60
8.1 Imported types 60
8.2 Type definitions 60
8.2.1 Nm_ModeType 60
8.2.2 Nm_StateType 60
8.2.3 Nm_BusNmType 61
8.2.4 Nm_ConfigType 61
8.3 Function definitions 62
8.3.1 Standard services provided by NM Interface 62
8.3.1.1 Nm_Init 62
8.3.1.2 Nm_PassiveStartUp 63
8.3.1.3 Nm_NetworkRequest 63
8.3.1.4 Nm_NetworkRelease 64
8.3.2 Communication control services provided by NM Interface 65
8.3.2.1 Nm_DisableCommunication 66
8.3.2.2 Nm_EnableCommunication 67
8.3.3 Partial Network services provided by NM Interface 68
8.3.3.1 Nm_RequestSynchronizedPncShutdown 68
8.3.3.2 Nm_UpdateIRA 68
8.3.4 Extra services provided by NM Interface 69
8.3.4.1 Nm_SetUserData 69
8.3.4.2 Nm_GetUserData 70
8.3.4.3 Nm_GetPduData 71

8.3.4.4 Nm_RepeatMessageRequest . . . 72
8.3.4.5 Nm_GetNodeIdentifier . . . 73
8.3.4.6 Nm_GetLocalNodeIdentifier . . . 73
8.3.4.7 Nm_CheckRemoteSleepIndication . . . 74
8.3.4.8 Nm_GetState . . . 75
8.3.4.9 Nm_PnLearningRequest . . . 76
8.3.5 Standard Call-back notifications . . . 77
8.3.5.1 Nm_NetworkStartIndication . . . 77
8.3.5.2 Nm_NetworkMode . . . 78
8.3.5.3 Nm_BusSleepMode . . . 78
8.3.5.4 Nm_PrepareBusSleepMode . . . 79
8.3.5.5 Nm_SynchronizeMode . . . 79
8.3.5.6 Nm_RemoteSleepIndication . . . 80
8.3.5.7 Nm_RemoteSleepCancellation . . . 80
8.3.5.8 Nm_SynchronizationPoint . . . 81
8.3.5.9 Nm_CoordReadyToSleepIndication . . . 82
8.3.5.10 Nm_CoordReadyToSleepCancellation . . . 82
8.3.5.11 Nm_ForwardSynchronizedPncShutdown . . . 83
8.3.5.12 Nm_PncBitVectorRxIndication . . . 84
8.3.5.13 Nm_PncBitVectorTxIndication . . . 84
8.3.5.14 Nm_PncBitVectorTxConfirmation . . . 85
8.3.6 Extra Call-back notifications . . . 85
8.3.6.1 Nm_PduRxIndication . . . 86
8.3.6.2 Nm_StateChangeNotification . . . 86
8.3.6.3 Nm_RepeatMessageIndication . . . 87
8.3.6.4 Nm_TxTimeoutException . . . 88
8.3.6.5 Nm_CarWakeUpIndication . . . 88
8.4 Scheduled functions . . . 89
8.4.1 Nm_MainFunction . . . 89
8.5 Expected interfaces . . . 90
8.5.1 Mandatory Interfaces . . . 90
8.5.2 Optional Interfaces . . . 90
8.5.3 Configurable Interfaces . . . 92
8.5.3.1 NmCarWakeUpCallout . . . 92
9 Sequence diagrams 93
9.1 Basic functionality . . . 93
9.2 Seq of NM Coordinator functionality . . . 93
9.3 Sequence of Partial network functionality . . . 95
10 Configuration specification 100
10.1 How to read this chapter . . . 100
10.2 Configuration parameters . . . 100
10.2.1 Nm . . . 101

10.3 Global configurable parameters . . . 101
10.3.1 NmGlobalConfig . . . 101
10.3.2 NmGlobalConstants . . . 103
10.3.3 NmGlobalProperties . . . 104
10.3.4 NmGlobalFeatures . . . 107
10.4 Channel configurable parameters . . . 114
10.4.1 NmChannelConfig . . . 114
10.4.2 NmPnFilterMaskByte . . . 120
10.4.2.1 Post-Build configurable PNC Filter Masks . . . 120
10.4.3 NmBusType . . . 122
10.4.4 NmGenericBusNmConfig . . . 123
10.4.5 NmStandardBusNmConfig . . . 124
10.5 Published Information . . . 125
A Not applicable requirements 126
B Change history of AUTOSAR traceable items 127
B.1 Traceable item history of this document according to AUTOSAR Release R22-11 . . . 127
B.1.1 Added Specification Items in R22-11 . . . 127
B.1.2 Changed Specification Items in R22-11 . . . 127
B.1.3 Deleted Specification Items in R22-11 . . . 130
B.1.4 Added Constraints in R22-11 . . . 130
B.1.5 Changed Constraints in R22-11 . . . 130
B.1.6 Deleted Constraints in R22-11 . . . 130
B.2 Traceable item history of this document according to AUTOSAR Release R23-11 . . . 131
B.2.1 Added Specification Items in R23-11 . . . 131
B.2.2 Changed Specification Items in R23-11 . . . 131
B.2.3 Deleted Specification Items in R23-11 . . . 131
B.2.4 Added Constraints in R23-11 . . . 131
B.2.5 Changed Constraints in R23-11 . . . 131
B.2.6 Deleted Constraints in R23-11 . . . 131
B.3 Traceable item history of this document according to AUTOSAR Release R24-11 . . . 132
B.3.1 Added Constraints in R24-11 . . . 132
B.3.2 Changed Constraints in R24-11 . . . 132
B.3.3 Deleted Constraints in R24-11 . . . 132
B.3.4 Added Specification Items in R24-11 . . . 132
B.3.5 Changed Specification Items in R24-11 . . . 132
B.3.6 Deleted Specification Items in R24-11 . . . 132
B.4 Traceable item history of this document according to AUTOSAR Release R25-11 . . . 132
B.4.1 Added Constraints in R25-11 . . . 132
B.4.2 Changed Constraints in R25-11 . . . 132

B.4.3 Deleted Constraints in R25-11 . . . 133
B.4.4 Added Specification Items in R25-11 . . . 133
B.4.5 Changed Specification Items in R25-11 . . . 133
B.4.6 Deleted Specification Items in R25-11 . . . 133
B.5 Traceable item history of this document according to AUTOSAR Release R26-11 . . . 133
B.5.1 Added Constraints in R26-11 . . . 133
B.5.2 Changed Constraints in R26-11 . . . 133
B.5.3 Deleted Constraints in R26-11 . . . 133
B.5.4 Added Specification Items in R26-11 . . . 133
B.5.5 Changed Specification Items in R26-11 . . . 134
B.5.6 Deleted Specification Items in R26-11 . . . 135

1 Introduction and functional overview

This document describes the concept, interfaces and configuration of the **Network Management Interface** module.

The **Network Management Interface** is an adaptation layer between the AUTOSAR Communication Manager and the AUTOSAR bus specific network management modules (e.g. CAN Network Management and FlexRay Network Management). This is also referred to as Basic functionality.

Additionally, this document describes the following optional features:

- interoperability between several networks connected to the same (coordinator) ECU that run AUTOSAR NM, where "interoperability" means that these networks can be put to sleep synchronously. This is also referred to as *NM Coordinator functionality*.
- Partial Network Cluster (PNC) handling, including handling of PNC timers and handling of synchronized PNC shutdown requests. If Partial Network is enabled, AUTOSAR NM aggregates all internal PNC requests (notified by ComM), all external PNC requests (notified by the <Bus>Nm's) and manage the PNC timers of each notified PNC. Additionally, if using the synchronized PNC shutdown functionality, AUTOSAR NM collect all synchronized PNC shutdown requests (notified by ComM) and control the PNC shutdown handling. For transmission of PN shutdown messages the AUTOSAR NM indicate the <Bus>Nm to fetch the aggregated PN shutdown requests. On reception of PN shutdown message the AUTOSAR NM acts as an interface layer between <Bus>Nm's and ComM. It forwards requests from <Bus>Nm's to ComM. Handling of PNC timer and handling of synchronized PNC shutdown requests are also referred to as *Partial Network functionality*.

Support of the *NM Coordinator functionality* and *Partial Network functionality* are optional. A **Network Management Interface** implementation can either support only Basic functionality or one of the following combinations:

- Basic functionality and *NM Coordinator functionality*.
- Basic functionality and *Partial Network functionality*.

The **Network Management Interface** is constructed to support generic lower layer modules that follow a fixed set of requirement for bus specific NM modules. This will allow third parties to offer support for OEM specific or legacy NM protocols such as direct OSEK NM.

2 Acronyms and Abbreviations

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

Abbreviation / Acronym:	Description:
CanIf	CAN Interface module
CanNm	CAN Network Management module
ComM	Communication Manager module
EcuM	ECU State Manager module
Nm	Generic Network Management Interface module, this ist the abreviation used for this module throughout this specification
CBV	Control Bit Vector in NM-message
PNC	Partial network cluster

Table 2.1: Acronyms and abbreviations used in the scope of this Document

Terms:	Definition:
Bus-Sleep Mode	Network mode where all interconnected communication controllers are in the sleep mode.
NM-Channel	Logical channel associated with the NM-cluster
NM-Cluster	Set of NM nodes coordinated with the use of the NM algorithm.
NM-Coordinator	A functionality of the Nm which allows coordination of network sleep for multiple NM Channels.
NM-Message	Packet of information exchanged for purposes of the NM algorithm.
NM-Timeout	Timeout in the NM algorithm that initiates transition into Bus-Sleep Mode.
NM User Data	Supplementary application specific piece of data that is attached to every NM message sent on the bus.
Node Identifier	Node address information exchanged for purposes of the NM algorithm.
Node Identifier List	List of Node Identifiers recognized by the NM algorithm.
Bus	Physical communication medium to which a NM node/ecu is connected to.
network	Entity of all NM nodes/ecus which are connected to the same bus.
channel	Logical bus to which the NM node/ecu is connected to.
Coordinated shutdown	Shutdown of two or more busses in a way that their shutdown is finished coinciding.
Coordination algorithm	Initiation of coordinated shutdown in case all conditions are met.
PNC Request Bit Vector	Represent the Partial Network information in a NM frame
PNC Request Bit Vector length	Represent the length of a Partial Network information in bytes
PNC Request Bit	One bit which represent a particular Partial Network in the Partial Network Info Range
PN filter mask	Vector of filter mask bytes defined by configuration container(s) NmPnFilterMaskByte per channel to filter relevant PNC requests for the PNC timer handling

Terms:	Definition:
Top-level PNC coordinator	An ECU acts as top-level PNC coordinator for those PNCs which are actively coordinated on all assigned channels. This ECU has the PNC gateway functionality enabled. The top-level PNC coordinator triggers for those PNCs a synchronized PNC shutdown, if no other ECU in the network requests them and if the synchronized PNC shutdown is enabled. Note: For different PNCs it is possible to have different top-level PNC coordinators.
Intermediate PNC coordinator	An ECU acts as intermediate PNC coordinator for those PNCs which are passively coordinated on at least one channel. This ECU has the PNC gateway functionality enabled. The intermediate PNC coordinator forwards a synchronized PNC shutdown to active coordinated channels for PNCs which are passively coordinated, if the synchronized PNC shutdown is enabled
PNC leaf node	A PNC leaf node is an ECU that act neither as top-level PNC coordinator nor as an intermediate PNC coordinator. It act as an ECU without a PNC gateway in the network and process PN shutdown message as usual NM messages.
PN shutdown message	A top-level PNC coordinator transmits PN shutdown messages to indicate a synchronized PNC shutdown across the PN topology. A PN shutdown message is a NM message which has PNSR bit in the control bit vector and all PNCs which are indicated for a synchronized shutdown set to '1'. An intermediate PNC coordinator which receive a PN shutdown message has to forward the PNC Request Bit Vector as PN shutdown message on the affected channels. Note: An intermediate PNC coordinators has to forward the PNC Request Bit Vector of received PN shutdown message as fast as possible to ensure a nearly synchronized shutdown of the affected PNCs across the PN topology.

Table 2.2: Terms used in the scope of this Document

3 Related documentation

3.1 Input documents

[1] Glossary
AUTOSAR_FO_TR_Glossary

[2] General Specification of Basic Software Modules
AUTOSAR_CP_SWS_BSWGeneral

[3] Specification of CAN Network Management
AUTOSAR_CP_SWS_CANNetworkManagement

[4] Specification of FlexRay Network Management
AUTOSAR_CP_SWS_FlexRayNetworkManagement

[5] Specification of UDP Network Management
AUTOSAR_CP_SWS_UDPNetworkManagement

[6] Specification of Network Management for SAE J1939
AUTOSAR_CP_SWS_SAEJ1939NetworkManagement

[7] General Requirements on Basic Software Modules
AUTOSAR_CP_RS_BSWGeneral

[8] Guide to Mode Management
AUTOSAR_CP_EXP_ModeManagementGuide

[9] Specification of Communication Manager
AUTOSAR_CP_SWS_COMManager

[10] Guide to BSW Distribution
AUTOSAR_CP_EXP_BSWDistributionGuide

[11] System Template
AUTOSAR_CP_TPS_SystemTemplate

3.2 Related specification

AUTOSAR provides a General Specification on Basic Software modules [2, SWS BSW General], which is also valid for the Generic Network Management Interface.

Thus, the specification SWS BSW General shall be considered as additional and required specification for the Generic Network Management Interface.

4 Constraints and assumptions

4.1 Limitations

1. The Generic Network Management Interface can only be applied to communication systems that support broadcast communication and 'bus-sleep mode'.
2. There is only one instance of the Generic Network Management Interface layer for all NM-Clusters. This instance manages all channels where a NM is used.
3. The Generic Network Management Interface shall only include the common modes, definitions and return values of different bus specific NM layers.
4. The Generic Network Management Interface shall only include the common modes, definitions and return values of different bus specific NM layers.
5. If 10BASE-T1S is used in combination with PLCA media access, then Nm Coordinator functionality are not supported.

   Note: Consequently, the configuration parameter `NmCoordinatorSupportEnabled` shall be set to false.
6. NM Coordination functionality in combination with Partial Network functionality and vice versa is not supported.

Figure 4.1 shows a typical example of the AUTOSAR NM stack.

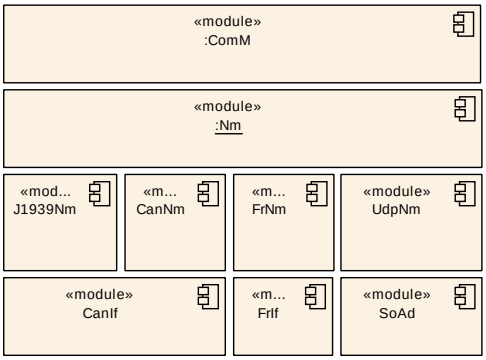

Figure 4.1: Nm stack modules

4.2 Specific limitations of the current release

The following limitations reflect desired functionality that has yet not been implemented or agreed upon, but might be added for future releases:

- No support of a back-up coordinator ECU (fault tolerance).

Also; explicit support for OSEK NM has been completely removed from this specification as of AUTOSAR Release 4.0. OSEK NM can still be supported by extending

the CanNm or by introducing a Complex Driver (CDD) on <Bus>Nm level as a generic BusNm. Supporting the OSEK NM through a CDD is not specified by AUTOSAR.

4.3 Applicability to automotive domains

The AUTOSAR NM Interface is generic and provides flexible configuration; it is independent of the underlying communication system and can be applied to any automotive domain under limitations provided above.

5 Dependencies to other modules

5.1 Interfaces to modules

Figure 5.1 shows the interfaces provided to and required from other modules in the AUTOSAR BSW.

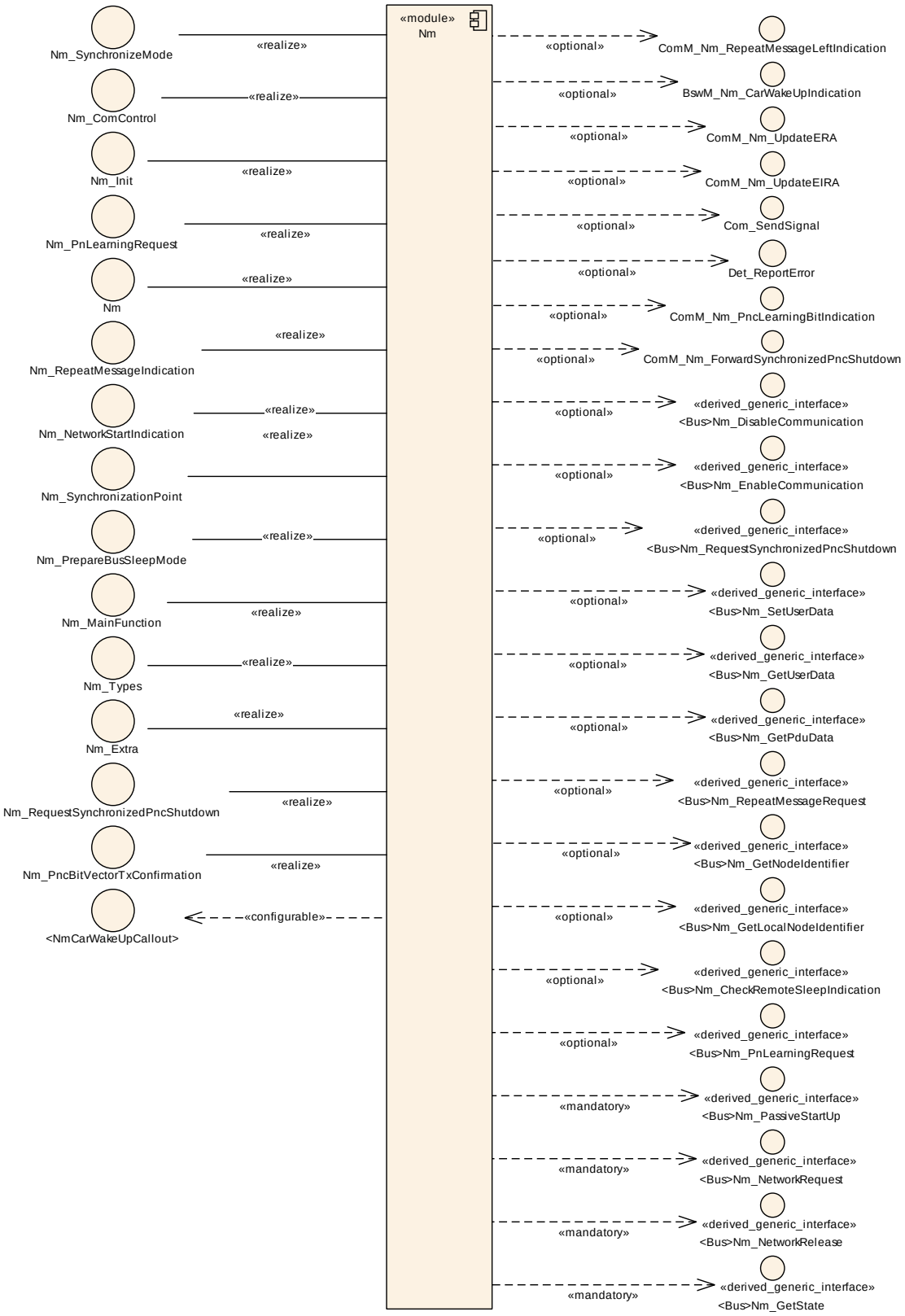

Figure 5.1: Nm's interfaces to other modules

5.1.1 ComM, CanNm, J1939Nm, FrNm, UdpNm, generic bus specific NM layers and CDD

The Generic Network Management Interface module (**Nm**) provides services to the Communication Manager (**ComM**) and uses services of the bus specific Network Management modules:

- CAN Network Management ([3, **CanNm**])
- FlexRay Network Management ([4, **FrNm**])
- Ethernet Network Management ([5, **UdpNm**]).
- J1939 Network Management ([6, **J1939Nm**]).

For Buses which do not need to provide Network Management Information on the bus like for example a LIN-bus the Bus-Type can be configured as "local Nm". With respect to callbacks, the **Nm** provides notification callbacks to the bus specific Network Management modules and calls the notification callbacks provided by the **ComM**.

In addition to the official AUTOSAR NM-modules above, Nm also support generic bus specific NM layers (**<Bus>Nm**). Any component which implements the required provided interfaces and uses the provided callback functions of Nm can be used as a bus specific NM. See Chapter 7.5 for the prerequisites for a generic bus specific NM.

Rationale: Nm is specified to support generic bus specific NM layers by adding generic lower layer modules as Complex Drivers. As such, Nm does not explicitly use the services by the official AUTOSAR bus-NM modules (CanNm, FrNm and UdpNm), but rather the services of the generic <Bus>Nm. The AUTOSAR bus-NMs are then explicitly supported since they implement the interfaces of <Bus>Nm.

The optional CarWakeUp-Functionality needs a Complex Driver which Coordinates Basic Software Mode Management.

5.1.2 Error handling modules

Nm reports development errors to the Default Error Tracer according to [SWS_Nm_00232].

5.1.3 BSW Scheduler

In case of the NM Coordinator functionality and depending on the configuration, the Nm will need cyclic invocation of it's main scheduling function in order to evaluate and detect when timers have expired.

5.2 File structure

5.2.1 Code file structure

[SWS_Nm_00247]

Upstream requirements: SRS_BSW_00159, SRS_BSW_00345, SRS_BSW_00419

⌈The code file structure shall not be defined within this specification completely. At this point it shall be pointed out that the code-file structure shall include the following files named:

⌋

5.2.2 Header file structure

[SWS_Nm_00124]

Upstream requirements: SRS_BSW_00348, SRS_BSW_00353, SRS_BSW_00357, SRS_BSW_00384

⌈The following header files shall be included by the Nm Interface module:

- Std_Types.h (for AUTOSAR standard types)
  Note: Platform_Types.h (for platform specific types) is indirectly included via AUTOSAR standard types.
- ComM_Nm.h (for Communication Manager callback functions)
- BswM_Nm.h (If the BswM is used for CarWakeup-functionalitiy)

⌋

[SWS_Nm_00243]

Upstream requirements: SRS_BSW_00171, SRS_BSW_00301, SRS_BSW_00384

⌈The Nm Interface shall optionally include the header file of Default Error Tracer (depending on the pre-processor switch NmDevErrorDetect, see ECUC_Nm_00203).

- Det.h for service of the Default Error Tracer.

⌋

6 Requirements traceability

The following tables references the requirements specified in [7] and links to the fulfillment of these.

Requirement	Description	Satisfied by
[SRS_BSW_00101]	The Basic Software Module shall be able to initialize variables and hardware in a separate initialization function	[SWS_Nm_00030]
[SRS_BSW_00159]	All modules of the AUTOSAR Basic Software shall support a tool based configuration	[SWS_Nm_00247]
[SRS_BSW_00171]	Optional functionality of a Basic-SW component that is not required in the ECU shall be configurable at pre-compile-time	[SWS_Nm_00243]
[SRS_BSW_00301]	All AUTOSAR Basic Software Modules shall only import the necessary information	[SWS_Nm_00117] [SWS_Nm_00243]
[SRS_BSW_00310]	API naming convention	[SWS_Nm_00037] [SWS_Nm_91003]
[SRS_BSW_00323]	All AUTOSAR Basic Software Modules shall check passed API parameters for validity	[SWS_Nm_00130] [SWS_Nm_00132] [SWS_Nm_00286] [SWS_Nm_00287] [SWS_Nm_00288] [SWS_Nm_00289] [SWS_Nm_00488] [SWS_Nm_00489] [SWS_Nm_00490] [SWS_Nm_00491] [SWS_Nm_00492] [SWS_Nm_00493] [SWS_Nm_00494] [SWS_Nm_00495] [SWS_Nm_00496] [SWS_Nm_00497] [SWS_Nm_00498] [SWS_Nm_00499] [SWS_Nm_00500] [SWS_Nm_00503] [SWS_Nm_00505] [SWS_Nm_00508]
[SRS_BSW_00327]	Error values naming convention	[SWS_Nm_00232]
[SRS_BSW_00330]	It shall be allowed to use macros instead of functions where source code is used and runtime is critical	[SWS_Nm_00091]
[SRS_BSW_00333]	For each callback function it shall be specified if it is called from interrupt context or not	[SWS_Nm_00028]
[SRS_BSW_00337]	Classification of development errors	[SWS_Nm_00232]
[SRS_BSW_00344]	BSW Modules shall support link-time configuration	[SWS_Nm_00030]
[SRS_BSW_00345]	BSW Modules shall support pre-compile configuration	[SWS_Nm_00247]
[SRS_BSW_00348]	All AUTOSAR standard types and constants shall be placed and organized in a standard type header file	[SWS_Nm_00124]
[SRS_BSW_00353]	All type definitions of target and compiler specific scope shall be placed and organized in a single type header	[SWS_Nm_00124]
[SRS_BSW_00357]	For success/failure of an API call a standard return type shall be defined	[SWS_Nm_00124]
[SRS_BSW_00358]	The return type of init() functions implemented by AUTOSAR Basic Software Modules shall be void	[SWS_Nm_00030]



△

Requirement	Description	Satisfied by
[SRS_BSW_00359]	Callback Function Return Types for AUTOSAR BSW	[SWS_Nm_00112] [SWS_Nm_00114] [SWS_Nm_00154] [SWS_Nm_00156] [SWS_Nm_00159] [SWS_Nm_00162] [SWS_Nm_00192] [SWS_Nm_00193] [SWS_Nm_00194] [SWS_Nm_00230] [SWS_Nm_00234] [SWS_Nm_00250] [SWS_Nm_00254] [SWS_Nm_00272]
[SRS_BSW_00369]	All AUTOSAR Basic Software Modules shall not return specific development error codes via the API	[SWS_Nm_00130] [SWS_Nm_00132] [SWS_Nm_00286] [SWS_Nm_00287] [SWS_Nm_00288] [SWS_Nm_00289] [SWS_Nm_00488] [SWS_Nm_00489] [SWS_Nm_00490] [SWS_Nm_00491] [SWS_Nm_00492] [SWS_Nm_00493] [SWS_Nm_00494] [SWS_Nm_00495] [SWS_Nm_00496] [SWS_Nm_00497] [SWS_Nm_00498] [SWS_Nm_00499] [SWS_Nm_00500] [SWS_Nm_00503] [SWS_Nm_00505] [SWS_Nm_00508]
[SRS_BSW_00373]	The main processing function of each AUTOSAR Basic Software Module shall be named according the defined convention	[SWS_Nm_00020]
[SRS_BSW_00384]	The Basic Software Module specifications shall specify at least in the description which other modules they require	[SWS_Nm_00124] [SWS_Nm_00243]
[SRS_BSW_00385]	List possible error notifications	[SWS_Nm_00232] [SWS_Nm_91011]
[SRS_BSW_00386]	The BSW shall specify the configuration and conditions for detecting an error	[SWS_Nm_00130] [SWS_Nm_00132] [SWS_Nm_00232] [SWS_Nm_00286] [SWS_Nm_00287] [SWS_Nm_00288] [SWS_Nm_00289] [SWS_Nm_00488] [SWS_Nm_00489] [SWS_Nm_00490] [SWS_Nm_00491] [SWS_Nm_00492] [SWS_Nm_00493] [SWS_Nm_00494] [SWS_Nm_00495] [SWS_Nm_00496] [SWS_Nm_00497] [SWS_Nm_00498] [SWS_Nm_00499] [SWS_Nm_00500] [SWS_Nm_00503] [SWS_Nm_00505] [SWS_Nm_00508]
[SRS_BSW_00405]	BSW Modules shall support multiple configuration sets	[SWS_Nm_00030]
[SRS_BSW_00414]	Init functions shall have a pointer to a configuration structure as single parameter	[SWS_Nm_00030] [SWS_Nm_00282]
[SRS_BSW_00419]	If a pre-compile time configuration parameter is implemented as `const` it should be placed into a separate c-file	[SWS_Nm_00247]
[SRS_BSW_00424]	BSW module main processing functions shall not be allowed to enter a wait state	[SWS_Nm_00118]
[SRS_BSW_00425]	The BSW module description template shall provide means to model the defined trigger conditions of schedulable objects	[SWS_Nm_00118]
[SRS_BSW_00450]	A Main function of a un-initialized module shall return immediately	[SWS_Nm_00121]
[SRS_BSW_00452]	Classification of runtime errors	[SWS_Nm_91011]

▽

△

Requirement	Description	Satisfied by
[SRS_BSW_00459]	It shall be possible to concurrently execute a service offered by a BSW module in different partitions	[SWS_Nm_00037] [SWS_Nm_00485] [SWS_Nm_00486] [SWS_Nm_91003]
[SRS_BSW_00460]	Reentrancy Levels	[SWS_Nm_00037] [SWS_Nm_91003]
[SRS_BSW_00461]	Modules called by generic modules shall satisfy all interfaces requested by the generic module	[SWS_Nm_00037] [SWS_Nm_91003]
[SRS_BSW_00478]	Timing limits of main functions	[SWS_Nm_00292]
[SRS_BSW_00484]	Input parameters of scalar and enum types shall be passed as a value.	[SWS_Nm_00037] [SWS_Nm_91003]
[SRS_ModeMgm_-09248]	it shall be possible to distinguish between internal and external PNC activation requests	[SWS_Nm_00302] [SWS_Nm_00330]
[SRS_ModeMgm_-09250]	PNC activation requests shall be exchanged with the Network Management via a `PNC Request Bit Vector`	[SWS_Nm_00317] [SWS_Nm_00321] [SWS_Nm_00327] [SWS_Nm_91006] [SWS_Nm_91008] [SWS_Nm_91010]

Table 6.1: Requirements Tracing

7 Functional specification

The NM Interface functionality consists of three parts:

- The *Base functionality* necessary to run, together with the bus specific NM modules, AUTOSAR NM on an ECU.
- The *NM Coordinator functionality* used by gateway ECUs to synchronously shut down one ore more busses.
- The Partial Network functionality is devided in 2 sub parts:
  - The handling of PNC requests (filtering, aggregation and monitoring) is used by any ECU which is member of an Partial Network.
  - The PNC timer handling is used by any ECU which is member of an Partial Network.
  - The *synchronized PNC shutdown functionality* used by PNC gateway ECUs in the role of a top-level PNC coordinator or intermediate PNC coordinator to synchronously shutdown particular PNCs across the PN topology.

7.1 Base functionality

The Generic Network Management Interface module (Nm) shall act as a bus-independent adaptation layer between the bus-specific Network Management modules (such as CanNm, J1939Nm, FrNm and UdpNm) and the Communication Manager module (ComM).

Note: The Nm does not provide interface functions beyond those specified in this document. The Nm will provide an interface to the ComM, that does not contain specific knowledge about the type of the underlying busses, and that nevertheless is sufficient to accomplish the necessary network management functions. The algorithm handled by the Nm is bus independent.

Note: It is also required that other service layer modules access network management functions exclusively via Nm and that no bypasses to bus specific NM functions exist

[SWS_Nm_00006] ⌈The Nm shall convert generic function calls from the ComM to bus specific functions of the bus specific NM layer.⌋

[SWS_Nm_00012] ⌈The Nm shall convert callback functions called by the bus specific NM layers to generic callbacks to the ComM.⌋

[SWS_Nm_00091]

Upstream requirements: SRS_BSW_00330

⌈The Base functionality of Nm may be implemented completely or partly using macros.⌋

7.2 NM Coordinator functionality

NM Coordinator functionality is a functionality of **Nm** that uses a coordination algorithm to coordinate the shutdown of NM on all, or one or more independent subsets of the busses that the ECU is connected to.
Dependent on configuration, the coordination algorithm can be configured to achieve different levels of synchronization of the shutdown.

An ECU using an NM that actively performs the *NM Coordinator functionality* is commonly referred to as an NM Coordinator. However, in this specification this term is synonymous with the *NM Coordinator functionality* when used in requirements.

Note: Consider that certain bus types have different nomenclature on the terms Network, Channel, Cluster.

[SWS_Nm_00292]

Upstream requirements: SRS_BSW_00478

⌈If the *NM Coordinator functionality* is configured, the configuration parameter NmCycletimeMainFunction shall be configured (see [SWS_Nm_00118]).⌋

Note: The NM Coordinator may use this to calculate the timeout status of internal timers.

7.2.1 Applicability of the NM Coordinator functionality

[SWS_Nm_00001] ⌈The coordination algorithm shall be able to handle a topology where several coordinated busses are connected to one NM Coordinator.⌋

[SWS_Nm_00256] ⌈The NM-Coordinator shall support two or more NM-Coordinators connected to the same NM Cluster.⌋

[SWS_Nm_00051] ⌈The NM Coordinator shall be able to coordinate busses running the official AUTOSAR bus specific NMs as well as all other generic bus NMs implementing the required functionality, callbacks and interfaces.⌋

Note: The required functionality, callbacks and interfaces are specified in Chapter 7.5.2. Coordinator Support for **J1939Nm** is not needed as the **J1939Nm** does not support shutdown handling.

[SWS_Nm_00055] ⌈The NM Interface configuration shall provide the parameter NmCoordinatorSupportEnabled to define if the support of the *NM Coordinator functionality* is present or not.⌋

[SWS_Nm_00167] ⌈It shall be possible to configure multiple NM coordination clusters that shall be coordinated independently.⌋

[SWS_Nm_00168] ⌈Each bus shall belong to zero or one NM coordination cluster.⌋

Rationale: The configuration parameter NmCoordClusterIndex is used for specifying to which coordination cluster a bus belongs. If this parameter is undefined for a channel, the corresponding bus does not belong to an NM coordination cluster.

[SWS_Nm_00169] ⌈Shutdown shall only be coordinated on the presently awake networks of a coordination cluster. Networks that are already in "bus-sleep mode" shall still be monitored but not coordinated.⌋

Rationale: The NM Coordinator does not require all busses in a coordination cluster to be awake, working with subsets of the coordination cluster resp. partial networks, to perform coordinated shutdown. It always monitors the shutdown initiation conditions and when these are met, it performs a coordinated shutdown of all the presently awake buses in the coordination cluster.

Note: It is outside the scope of the **Nm** to provide synchronized wakeup for coordinated busses. It is up to the application (-> vehicle mode management) to wake up the required resp. all channels if one channel wake up occurs.

7.2.2 Keeping coordinated busses alive

[SWS_Nm_00002] ⌈As long as the node implementing the NM Coordinator is not ready to go to sleep on at least one of the busses in a coordination cluster (i.e. that it has actively requested the network), the NM Coordinator shall ensure that the network is requested on all currently active busses in that coordination cluster.⌋

[SWS_Nm_00003] ⌈As long as at least one bus in the coordination cluster is not ready to sleep (i.e. because another node than the NM Coordinator is requesting that bus), the NM Coordinator shall still ensure that the network is requested on all currently active busses in that coordination cluster even if the local ECU itself is ready to go to sleep on all busses of that coordination cluster.⌋

Rationale: The **bus specific NM**s will indicate to **Nm** if the bus is ready to go to sleep or not by calling the callbacks Nm_RemoteSleepIndication and Nm_RemoteSleepCancellation. The local ECU will indicate if it is ready to go to sleep or not on a network using the API functions Nm_NetworkRelease and Nm_NetworkRequest.

Rationale: The **Nm** requests the network on a bus by calling the bus specific NM function <Bus>Nm_NetworkRequest.

Since all AUTOSAR bus specific NMs are built on the principle that one AUTOSAR node can keep the bus alive as long as it keeps the network requested, the NM Coordinator will keep all busses of the coordination cluster awake by requesting the network for the **bus specific NM**s.

The two requirements [SWS_Nm_00002] and [SWS_Nm_00003] above can be summarized as follows: as long as at least one node (including the node implementing the

NM Coordinator) keeps any of the busses in the coordination cluster awake, the NM Coordinator shall keep all busses of that coordination cluster awake.

[SWS_Nm_00228] ⌈If a bus of a coordination cluster has the parameter NmChannelSleepMaster set to **TRUE**, the NM Coordinator shall consider that bus ready to sleep at all times and shall not await an invocation of Nm_RemoteSleepIndication from that bus before starting shutdown of that network.⌋

Rationale: This property shall be set for all **bus specific NM**s where the sleep of the bus can be absolutely decided by the local node only and that no other nodes of that bus can oppose that decision. An example of such a network is LIN where the local AUTOSAR ECU will always be the LIN bus master and can always solely decide when the network shall go to sleep.

7.2.3 Shutdown of coordinated busses

The level of synchronization achievable is dependent on the configuration. See Chapter 7.2.5, Figure 7.1 shows an overview of the coordination algorithm. As described in Section 7.2.1, the coordination algorithm and coordinated shutdown shall be applied independently per NM coordination cluster.

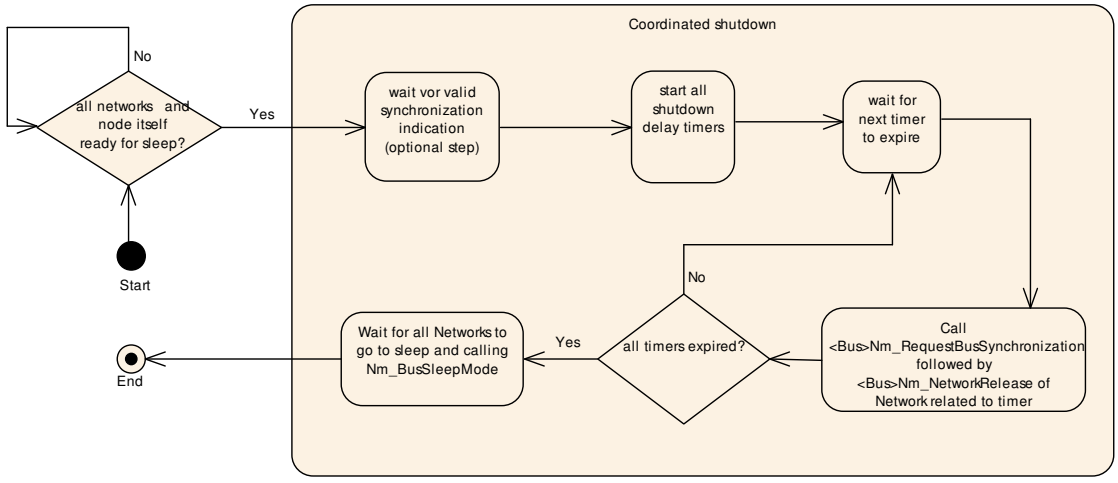

Figure 7.1: Overview of the coordination algorithm with the coordinated shutdown as part of it

Note: There is no limitation where the actions performed by the coordination algorithm shall take place.
This can be done either by the Nm main function (Nm_MainFunction) or module indication / callbacks.

[SWS_Nm_00171] ⌈When all networks of a coordination cluster are either ready to go to sleep or already in ”bus-sleep mode“ the NM Coordinator shall start the coordinated shutdown on all awake networks. The NM Coordinator shall evaluate continuously if the coordinated shutdown can be started.⌋

Rationale: Evaluation of shutdown conditions can be also done in other API calls then the main function. The evaluation can be segmented then to check only the specific conditions affected by the API calls there, hence it is not necessary to re-evaluate all conditions in every main processing period and every API call.

[SWS_Nm_00172] ⌈If the configuration parameter NmSynchronizingNetwork is **TRUE** for any of the busses in a coordination cluster, the coordination shutdown shall be delayed until a network that is configured as synchronizing network for this coordination cluster invoked Nm_SynchronizationPoint.⌋

[SWS_Nm_00293] ⌈If on a coordinated network the coordinator detects a mode change to NM_MODE_SYNCHRONIZE,NM_MODE_PREPARE_BUS_SLEEP or NM_MODE_BUS_SLEEP AND the coordinated cluster this network belongs to has not started the shutdown process AND if there is no internal network mode request for that channel by ComM, the coordinator shall treat this network as remote sleep and shall call <Bus>Nm_NetworkRelease for this network. If additionally for this network the configuration parameter NmSynchronizingNetwork is **TRUE** then the coordinator shall not wait for Nm_SynchronizationPoint on this network.⌋

Rationale: If one or more of the networks in the NM coordination clusters is cyclic (such as FlexRay), a higher level of synchronized shutdown will be achieved if the algorithm is synchronized with one of the included cyclic networks. If configured so, the shutdown timers for all coordinated networks will not be started until the synchronizing network has called the Nm_SynchronizationPoint.

Rationale: Although only one network per NM coordination cluster should be configured to indicate synchronization points, this will allow the *NM Coordinator functionality* to filter out all synchronization indications except those that is originate from the network that is configured to be the synchronizing network of each coordination cluster.

[SWS_Nm_00173] ⌈If not all conditions to start the coordinated shutdown have been met, or if the coordinated shutdown has already been started (but not aborted), calls to Nm_SynchronizationPoint shall be ignored.⌋

Rationale: In some cases, non-synchronizing networks can take longer time to go to sleep. If this happens, the coordinated shutdown will be started based on one synchronization indication, but as the synchronizing network will not be released directly it will continue to invoke (several) more synchronization indications which can safely be ignored.

[SWS_Nm_00174] ⌈If the configuration parameter NmSynchronizingNetwork is **FALSE** for all of the presently awake busses in a coordination cluster, the timers shall be started after all shutdown conditions have been met, without waiting for a call to Nm_SynchronizationPoint(). (see also [SWS_Nm_00172]).⌋

[SWS_Nm_00175] ⌈When the coordinated shutdown is started, a shutdown delay timer shall be activated for each currently awake, actively coordinated channel in the coordination cluster. Each timer shall be set to NmGlobalCoordinatorTime. In

case NmBusType is not set to NM_BUSNM_LOCALNM additionally the shutdown time of the specific channel **TSHUTDOWN_CHANNEL** shall be subtracted.⌋

[SWS_Nm_00284] ⌈If the NmGlobalCoordinatorTime is zero the shutdown delay timer of all channels shall also be zero.⌋

Note: The **TSHUTDOWN_CHANNEL** can be calculated as described in Chapter 7.2.5 or with following formulas:
CanNm: Ready Sleep Time + Prepare BusSleep Time
FrNm: Ready Sleep Time, e.g.: (FrNmReadySleepCnt+1) * FrNmRepetitionCycle * "Duration of one Flexray Cycle"
GenericNm: NmGenericBusNmShutdownTime

[SWS_Nm_00176] ⌈When a shutdown timer expires for a network, **Nm** shall in case Nm_BusNmType is not set to NM_BUSNM_LOCALNM release the network by calling the <Bus>Nm_RequestBusSynchronization followed by <Bus>Nm_NetworkRelease. In case BusNmType is set to NM_BUSNM_LOCALNM **Nm** shall inform ComM about shutdown by calling ComM_Nm_BusSleepMode.⌋

Note: In the AUTOSAR Classic Platform, CanNm_PassiveStartUp, J1939Nm_PassiveStartUp, FrNm_PassiveStartUp and UdpNm_PassiveStartUp have been specified as the predefined interfaces corresponding to <Bus>Nm_PassiveStartUp.

[SWS_Nm_00177] ⌈**Nm** shall keep track of all networks that have been released but have not yet reported "bus-sleep mode". If the shutdown is aborted, these networks shall still be considered active networks.⌋

Note: See Section Chapter 7.3.3.

Definition: When all networks have been released and all networks are in "bus-sleep mode", the coordinated shutdown is completed.

7.2.4 Coordination of nested sub-busses

To support the coordination of nested sub-busses the Nm-Coordinators need be configured to build up a coordination hierarchy. The top most NM Coordinator has only actively coordinated channels (NmActiveCoordinator == **TRUE**) per coordination cluster. This NM Coordinator has to initiate the coordinated shutdown for all other coordinators. An nested NM Coordinator receive his shutdown indication information from his passively configured channel (NmActiveCoordinator == **FALSE**) and provides this information to following NM Coordinators via his actively coordinated channels (NmActiveCoordinator == **TRUE**).

The Figure 7.2 will explain this as an example.

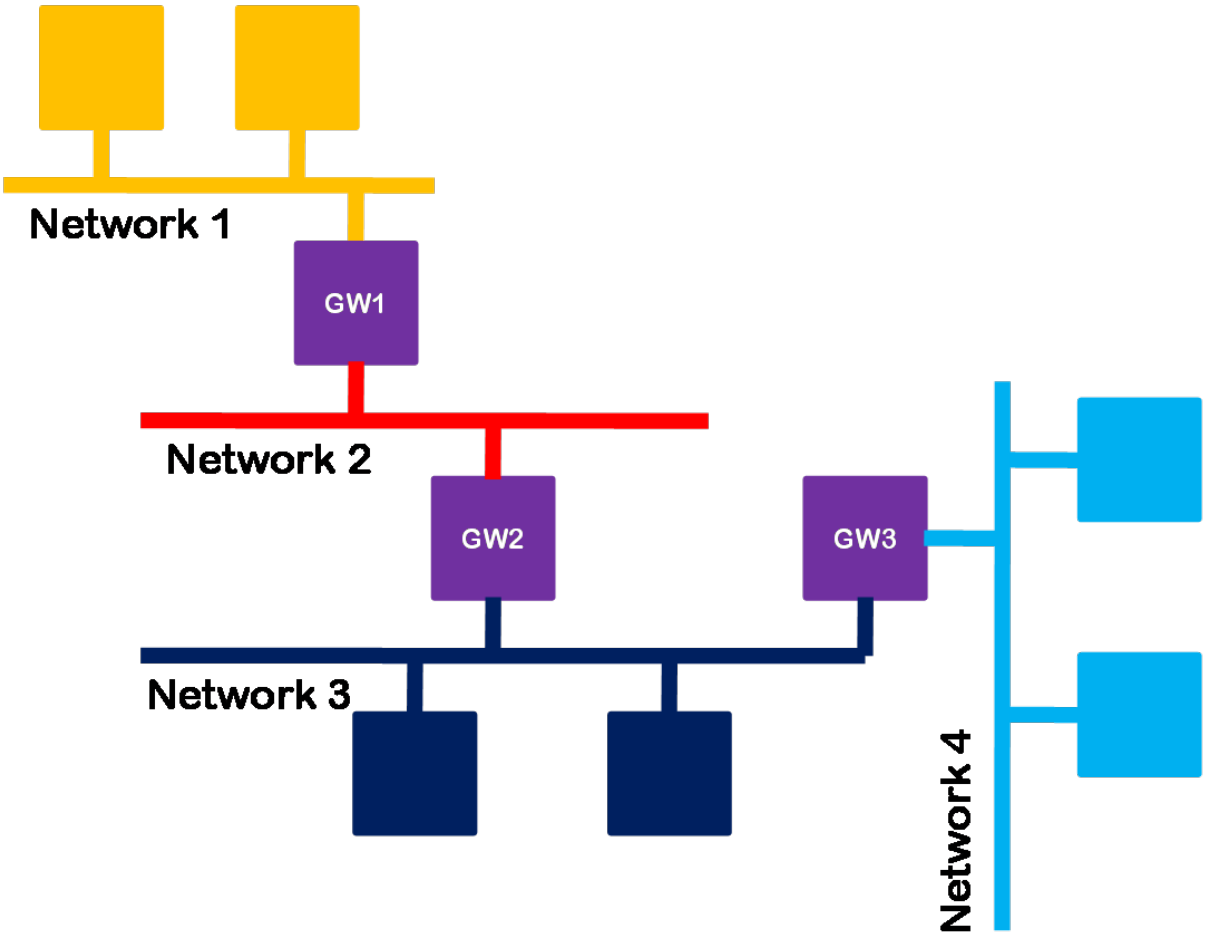

Figure 7.2: Use Case Nested Gateways

The exemplary topology shown in Figure 7.2 has the following coordination approach. GW 1 have configured the channel onto Network 1 and Network 2 as actively coordinating channels. Where GW 2 is configured with Network 2 connection as passively coordinated channel, but with actively coordinated channel on Network 3. GW 3 than needs to be configured on Network 3 as passively coordinated channel but as actively coordinated channel for his connection to the Network 4.

[SWS_Nm_00280] ⌈The functionality of coordinating nested sub busses shall be available if the `NmCoordinatorSyncSupport` parameter is set to **TRUE**.⌋

Note: All requirements within this chapter are valid "per `Nm Coordination Cluster`" (see [SWS_Nm_00167]).

The `NmActiveCoordinator` parameter indicates, if an `NM Coordinator` behaves on this channel in actively manner
(Actively coordinated channel) [`NmActiveCoordinator` = **TRUE**]
or behave in a passively manner
(Passively coordinated channel) [`NmActiveCoordinator` = **FALSE**].

[SWS_Nm_00257] ⌈On its passively coordinated channels a NM-Coordinator shall send Nm messages only if the node has a network management request pending or a connected network which is coordinated actively by that NM Coordinator is not ready to sleep.⌋

Rationale: This prevents that 2 NM Coordinators at the same channel, send NM messages when they are ready to sleep and therefore keep the bus awake. Without this mechanism it would not be possible to detect if there is at least one other node active.

Note: The described behavior in this requirement extends [SWS_Nm_00003] for passively coordinated channels.

[SWS_Nm_00259] ⌈The NM Coordinator shall set the *NMcoordinatorSleepReady* bit in the NM message via <Bus>Nm_SetSleepReadyBit to the value **1** at his actively coordinated channels,
that have their NmBusType not configured to NM_BUSNM_LOCALNM,
IF
coordinated shutdown has started (according to [SWS_Nm_00171], [SWS_Nm_00172], [SWS_Nm_00174])
AND
If all channels of this NM Coordination cluster are configured as NmActive-Coordinator == **TRUE**.⌋

Note: For Position of Coordinator Bits in CBV see according **<Bus>Nm** specifications.

Note: This applies to the top most coordinator (no passively coordinated channel). Nodes which contain a passively coordinated channel will set the bit according to the requirement in [SWS_Nm_00261].

Rationale: Nodes which contain passively coordinated channels do not need a synchronization point as they are synchronized by the sleep ready bit of their active coordinator already.

[SWS_Nm_00261] ⌈If Nm_CoordReadyToSleepIndication is received on a passively coordinated channel the NmCoordinator shall set the *NMCoordinatorSleepReady* bit to **SET (1)** via API call to <Bus>Nm_SetSleepReadyBit on all actively coordinated channels.⌋

[SWS_Nm_00271] ⌈If Nm_CoordReadyToSleepCancellation is received on a passively coordinated channel the NmCoordinator shall set the *NMCoordinatorSleepReady* bit to **UNSET (0)** via API call to <Bus>Nm_SetSleepReadyBit on all actively coordinated channels.⌋

Note: On its passively coordinated channel a NM Coordinator would not set the *Sleep Ready* bit ever (via **<Bus>Nm** function call) but forward a received status change of *Sleep ready* bit onto its actively coordinated channels.

Note: On its actively coordinated channel(s) a NM Coordinator a call of Nm_CoordReadyToSleepIndication and Nm_CoordReadyToSleepCancellationis not expected.

[SWS_Nm_00262] ⌈NM Coordinators with passively coordinated channels shall start coordinated shutdown after the *Sleep Ready* Bit with **SET** status has been requested.⌋

[SWS_Nm_00281] ⌈NmGlobalCoordinatorTime shall be set at least to the maximum time needed to shut down all Networks coordinated.⌋

Note: This includes all nested connections.(for example see Figure 7.3)

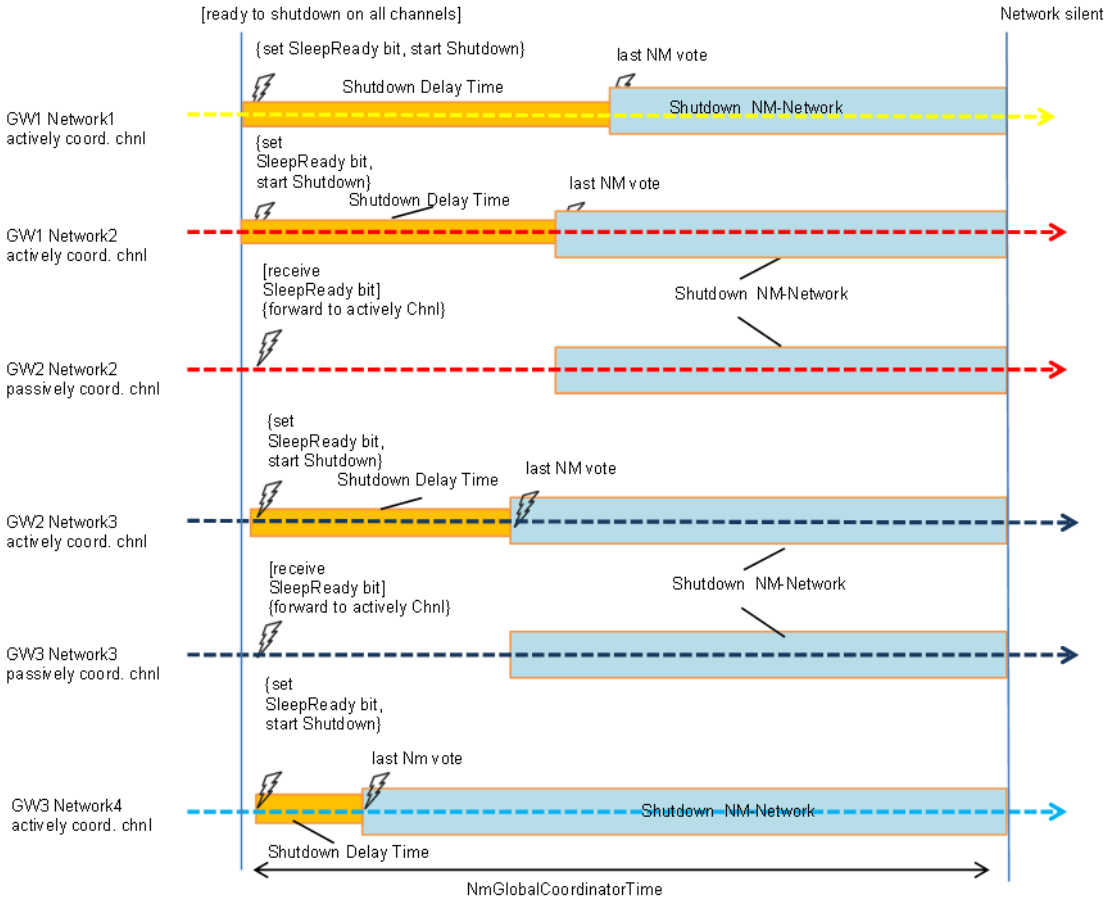

Figure 7.3: Shutdown with Nm_GlobalCoordinatorTime

[SWS_Nm_00267] ⌈NM Coordinator shall set the *NMCoordinatorSleepReady* bit to **UNSET (0)** via API call to <Bus>Nm_SetSleepReadyBit on all actively coordinated channels if the coordinated shutdown has been aborted for any reason.⌋

Note: Details about aborted shutdown can be found in Chapter 7.3.3.

7.2.5 Calculation of shutdown timers

The coordination algorithm is quite flexible since the level of synchronization achievable depends on the configuration of switches and timers. Depending on which event or point in time that is the goal to synchronize on, the configuration shall be done differently. This Chapter contains guide on how to achieve three different levels of synchronization. It is up to the configuration to follow these guidelines or to achieve a separate order of synchronization by choosing his/her own particular configuration. Therefore, this Section will not contain any requirement, only recommendations.

Note that absolute synchronization will never be possible to achieve. The jitter factors that determine the preciseness of the synchronization involve the processing period of the **Nm**, the exactness of the timers and the busload for non-deterministic busses. Correctly configured, the Use Cases described below will give the best possible synchronization that is achievable considering these circumstances.

Previous version of the NM Coordinator included the possibility for the coordinator algorithm to delay the start of the coordinated shutdown "a number of rounds". This specific delay has been removed but a similar behavior can still be obtained by increasing all shutdown timers (configuration parameter NmGlobalCoordinatorTime). Special care must be taken when cyclic networks (such as FlexRay) are used when this increased delay time should be quantified to the synchronization indication periodicity of those networks.

7.2.6 Synchronization Use Case 1 - Synchronous command

This Use Case focuses on how to synchronize the point in time where the different networks are released.
This results in the fastest possible total shutdown of all networks, but with the downside that the networks will not enter "bus-sleep mode" at the same time.

Rationale: One example of this Use Case is when several CAN networks shall be kept alive as long as any CAN-node is requesting one of the networks; but when all nodes are ready to go to sleep it does not matter if "bus-sleep mode" is entered at the same time for the different networks.

Since the Use Case does not consider any cyclic behavior of the networks, the synchronization parameter NmSynchronizingNetwork shall be set to **FALSE** for all networks and no **bus specific NM** shall be configured to invoke the Nm_SynchronizationPoint callback.
To achieve the fastest possible shutdown, the shutdown timer parameter NmGlobalCoordinatorTime needs to be set to 0.0.

7.2.7 Synchronization Use Case 2 - Synchronous initiation

This Use Case is an extension of Use Case 1, but here consideration is taken to the fact that for some networks the request to release the network will only be acted upon at specific points in time. This Use Case will command a simultaneous shutdown like in Use Case 1, but will wait until a point in time suitable for the synchronizing network.

Rationale: One example of this Use Case is when one FlexRay network and several CAN networks where the time when all networks are active shall be maximized, but the networks shall still be put to sleep as fast as possible.

Since this Use Case shall consider the cyclic behavior of a selected network, one of the networks shall have its synchronization parameter `NmSynchronizingNetwork` set to **TRUE** while the other networks shall have this parameter set to **FALSE**. The synchronizing network's **bus specific NM** shall also be configured to invoke the `Nm_SynchronizationPoint` callback at suitable points in time where the shutdown shall be initiated.
To achieve the fastest possible shutdown, the shutdown timer parameter `NmGlobalCoordinatorTime` needs to be set to `0.0`.

7.2.8 Synchronization Use Case 3 - Synchronous network sleep

This Use Case will focus on synchronizing the point in time where the different networks enters "bus-sleep mode". It will wait for indication from a synchronizing network, and then delay the network releases of all networks based on timing values so that the transition from "network mode" (or "prepare bus-sleep mode") into "bus-sleep mode" is as synchronized as possible.

Rationale: One example of this Use Case is when one FlexRay network and several CAN networks shall stop communicating at the same time.

Since this Use Case shall consider the cyclic behavior of a selected network, of the networks - preferably the cyclic one - shall have its synchronization parameter `NmSynchronizingNetwork` set to **TRUE** while the other networks shall have this parameter set to `FALSE`. The synchronizing network's **bus specific NM** shall also be configured to invoke the `Nm_SynchronizationPoint` callback at suitable points in time where the shutdown shall be initiated.

To calculate the shutdown timer **TSHUTDOWN_CHANNEL** of each network, specific knowledge of each networks timing behavior must be obtained.

For all networks, **TSHUTDOWN_CHANNEL** must be calculated, this is the minimum time it will take the network to enter "bus-sleep mode". For non-cyclic networks (such as CAN), the time shall be measured from the point in time when the network is released until it enters "bus-sleep mode". For cyclic networks (such as FlexRay) the time shall also include the full range from the synchronization indication made just before

the network is released. For Generic **BusNms** the time is given by the configuration parameter NmGenericBusNmShutdownTime.

For the synchronizing network, **TSYNCHRONIZATION_INDICATION** must be determined. This is the time between any two consecutive calls made by that **bus specific NM** to Nm_SynchronizationPoint.

The NmGlobalCoordinatorTime shall be the total time that is needed for the coordination algorithm. This includes the shutdown time of nested sub-busses. Start with setting NmGlobalCoordinatorTime to the same value as **TSHUTDOWN_CHANNEL** for the synchronizing network. If the **TSHUTDOWN_CHANNEL** for any other network is greater than NmGlobalCoordinatorTime, extend NmGlobalCoordinatorTime with **TSYNCHRONIZATION_INDICATION** repeatedly until NmGlobalCoordinatorTime is equal to, or larger than any **TSHUTDOWN_CHANNEL**.

The shutdown delay timer for each network shall be calculated as NmGlobalCoordinatorTime - **TSHUTDOWN_CHANNEL** for that network.

For the cyclic networks this parameter must then be increased slightly in order to make sure that the network release will occur between to synchronization indications, slightly after Nm_SynchronizationPoint (would) have been called. The amount of time to extend the timer depends on the implementation and configuration of the **bus specific NM** but should be far smaller than **TSYNCHRONIZATION_INDICATION**.

7.2.8.1 Examples

In the first case (Figure 7.4), the synchronizing network holds the largest **TSHUTDOWN_CHANNEL**, which will therefore equal the NmGlobalCoordinatorTime. For the synchronizing network, the shutdown delay timer will be NmGlobalCoordinatorTime - **TSHUTDOWN_CHANNEL**, which is zero, but then a small amount of time is added to make sure that the Nm will wait to release the network between the two synchronization points.
For the Non-cyclic network, the shutdown delay timer will simply be NmGlobalCoordinatorTime - **TSHUTDOWN_CHANNEL**.

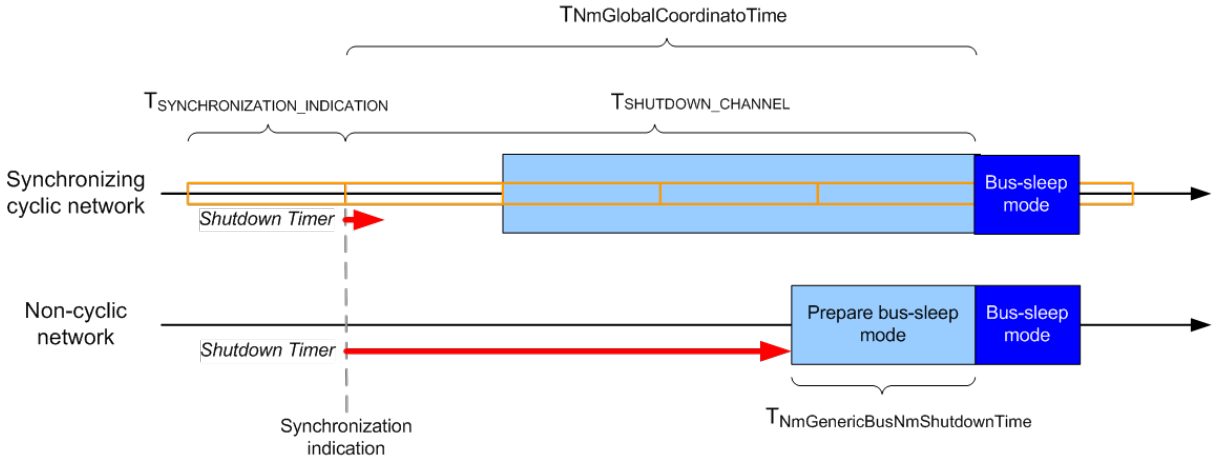

Figure 7.4: Timing example one

In the second case (Figure 7.5), the non-cyclic network takes very long time to shut down and therefore holds the largest **TSHUTDOWN_CHANNEL**. The `NmGlobalCoordinatorTime` has now been obtained by taking the synchronizing network's (slightly shorter) **TSHUTDOWN_CHANNEL** adding **TSYNCHRONIZATION_INDICATION** once to this value.

For the synchronizing network, the shutdown timer will be `NmGlobalCoordinatorTime` - **TSHUTDOWN_CHANNEL**, with a small amount of time added to make sure that the **Nm** will wait to release the network between the two synchronization points.
For the Non-cyclic network, the shutdown timer will simply be `NmGlobalCoordinatorTime` - **TSHUTDOWN_CHANNEL**.

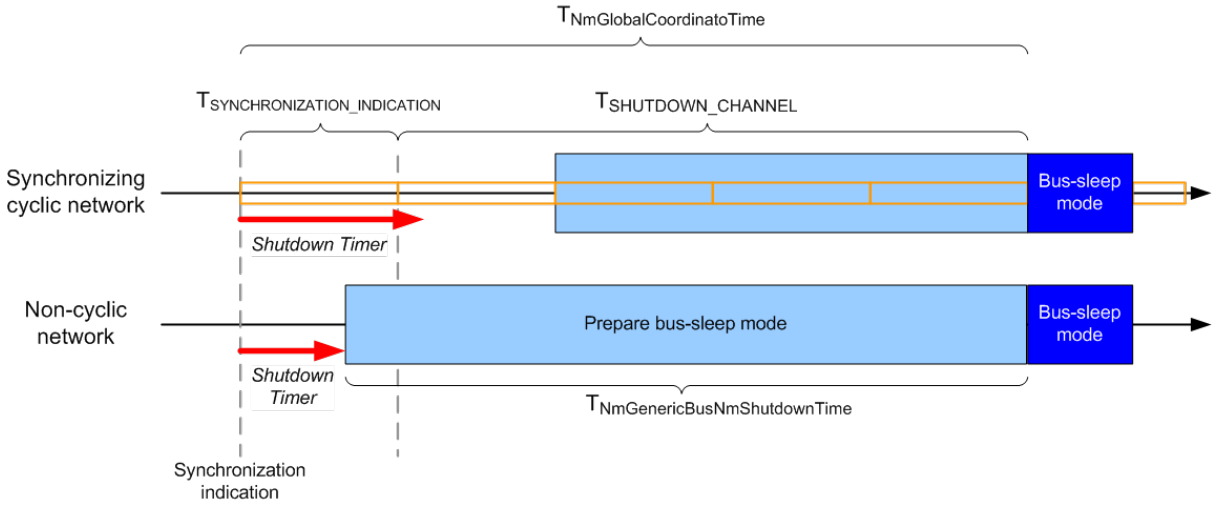

Figure 7.5: Timing example two

7.3 Wakeup and abortion of the coordinated shutdown

Nm is not responsible for normal wakeup of the node or the networks this will be done by the COM Manager (**ComM**).

7.3.1 External network wakeup

For both *Basic functionality* and *NM Coordination functionality*, **Nm** will forward wakeup indications from the networks (indicated by the bus specific NMs calling the callback `Nm_NetworkStartIndication`) to the **ComM** by calling `ComM_Nm_NetworkStartIndication`(). **ComM** will then call `Nm_PassiveStartUp`, which will be forwarded by Nm to the corresponding interface of the bus specific NM.
Processing of wake-up events for channels in bus-sleep (related to transceiver and controller state) will be handled by **EcuM** and **ComM**. No interaction of the **Nm** apply here. **Nm** will get the network request from **ComM** as statet above, depending on the wake-up validation and the respective communication needs.

[SWS_Nm_00245] ⌈If the **ComM** calls `Nm_PassiveStartUp`() for a network that is part of a coordinated cluster of networks, the **Nm** coordinator functionality shall treat this call as if the **ComM** had called `Nm_NetworkRequest`(). In case BusNmType is not set to `NM_BUSNM_LOCALNM` the **Nm** shall forward a call of `<Bus>Nm_NetworkRequest` to the lower layer and accordingly, the network shall be counted as requested by the NM coordinator.⌋

Note: In other words: Calls of `Nm_PassiveStartUp` for networks that are part of a cluster of coordinated networks shall be "translated" to / handled as calls of `Nm_NetworkRequest`.

7.3.2 Coordinated wakeup

Depending on the configuration, **ComM** can start multiple networks based on the indication from one network. It is recommended to configure the **ComM** to automatically start all network of a `NM Coordination Cluster` if one of the networks indicates network start, but this is not always necessary. Since the wakeup of network is outside the scope of **Nm**, this is independent of if the *NM Coordination functionality* is used or not.

7.3.3 Abortion of the coordinated shutdown

If the *NM Coordination functionality* is activated and `coordinated shutdown` has been initiated on an `NM Coordination Cluster`, dependent on the coordinator algorithm configuration it might take time before each included bus is actually released. If any node on one of the coordinated buses changes its state and starts requesting

the network before all networks are released, race conditions can occur in the coordination algorithm. This can happen in four ways:

1. A node on a network that has not yet been released and is still in 'network mode' starts requesting the network again. This will be detected by the **bus specific NM** which will inform **Nm** by calling Nm_RemoteSleepCancellation.
2. A node on a network that has already been released and has indicated "prepare bus-sleep mode" but not "bus-sleep mode" starts requesting the network again. This will be detected by the **bus specific NM** that will automatically change state to "network mode" and inform **Nm** by calling Nm_NetworkMode.
3. The **ComM** requests the network on any of the networks in the NM Coordination Cluster.
4. The coordinator which actively coordinates this network sends Nm message with cleared Ready-Sleep Bit. This will be detected by the Bus spec NM (only on passively coordinated channels) and forwarded to the NM by calling Nm_CoordReadyToSleepCancellation.

The generic approach is to abort the shutdown and start requesting the networks again. However, networks that have already gone into "bus-sleep mode" shall not be automatically woken up; this must be requested explicitly by **ComM**.

[SWS_Nm_00181] ⌈The coordinated shutdown shall be aborted if any network in that NM Coordination Cluster,

- indicates Nm_RemoteSleepCancellation or
- indicates Nm_NetworkMode or
- indicates Nm_CoordReadyToSleepCancellation
- or the **ComM** request one of the networks with Nm_NetworkRequestor Nm_PassiveStartUp.

⌋

Note: Nm_NetworkStartIndication is not a trigger to abort the coordinated shutdown, as this is handled by the upper layer.

[SWS_Nm_00182] ⌈If the coordinated shutdown is aborted, NM Coordinator shall call ComM_Nm_RestartIndication for all networks that already indicated "bus sleep".⌋

Rationale: Since **Nm** cannot take decision to wake networks on its own, this must be decided by **ComM** just as in the (external) wakeup case.

[SWS_Nm_00183] ⌈If the coordinated shutdown is aborted, NM Coordinator shall in case BusNmType is not set NM_BUSNM_LOCALNM request the network from the **<Bus>Nm**'s for the networks that have not indicated "bus sleep". In case BusNmType

is set to NM_BUSNM_LOCALNM **Nm** shall inform **ComM** about network startup by calling ComM_Nm_NetworkMode.⌋

[SWS_Nm_00185] ⌈If the coordination algorithm has been aborted, all conditions that guard the initiation of the coordinated shutdown shall be evaluated again.⌋

Rationale: When a coordinated shutdown has been aborted, in most cases there are now networks in that NM Coordination Cluster that do not longer indicate that network sleep is possible, and thus the NM Coordinator must keep all presently non-sleeping networks awake. There can be cases where none of the conditions have been changed, which will only lead to a re-initiation of the coordinated shutdown.

[SWS_Nm_00235] ⌈If a coordinated shutdown has been aborted and **Nm** receives **E_NOT_OK** on a <Bus>Nm_NetworkRequest, that network shall not be considered awake when the conditions for initiating a coordinated shutdown are evaluated again.⌋

Rationale: Any **<Bus>Nm** that needs to be re-requested during an aborted coordinated shutdown have previously been released, both by **ComM** and by **Nm**. It is the responsibility of the **<Bus>Nm** to inform the **ComM** (through **Nm**) that the network really has been released and therefore the **ComM** will have knowledge of the network state even though the error response on Nm_NetworkRequest never reached the **ComM** directly.

[SWS_Nm_00236] ⌈If a coordinated shutdown has been initiated and **Nm** receives **E_NOT_OK** on a <Bus>Nm_NetworkRelease, the shutdown shall be immediately aborted. For all networks that have not entered ”bus-sleep mode“, **Nm** shall request the networks. This includes the network that indicated an error for <Bus>Nm_NetworkRelease. As soon as this has been done, the conditions for initiating coordinated shutdown can be evaluated again. This applies also to networks that were not actively participating in the current coordinated shutdown.⌋

Rationale: If a network cannot be released, it shall immediately be requested again to synchronize the states between the NM Coordinator in the **Nm** and the **<Bus>Nm**. The coordinated shutdown will eventually be initiated again as long as the problem with the **<Bus>Nm** persists. It is up to the **<Bus>Nm** to report any problems directly to the **DEM** and/or Default Error Tracer so the NM Coordinator shall only try to release the networks until it is successful.

7.4 Partial Network functionality

An overview regarding the partial network cluster functionality can be found in document Guide to Mode Management [8]

7.4.1 PNC Request Bit Vector filter algorithm

The intention of the PNC Request Bit Vector filter algorithm is to include all PNC requests that are relevant for the ECU for the PNC handling and to exclude all received PNC requests that are not relevant for the ECU. Additionally the filter algorithm is used to qualify relevant PNC request for transmission. PNC requests which are qualified to be relevant (re-)start the corresponding PNC timer.

In order to distinguish between PNC requests that are relevant for the ECU and PNC requests that are not relevant, the NM evaluates the PNC Request Bit Vector received by the **<Bus>Nm** (passive PNC requests, initiated remotely by another ECU in the network) and it evaluates the PNC Request Bit Vector transmitted by **ComM** (active request, initiated by a local application). Every bit of the PNC Request Bit Vector represents one PNC.

PNCs are statically configured. One PNC denotes the participation of an ECU to a specific partial network cluster (PNC).

[SWS_Nm_00308] ⌈The PNC Request Bit Vector filter algorithm shall evaluate the bytes of the given PNC Request Bit Vector in the context of Nm_PncBitVectorRxIndication.⌋

Note: If <Bus>NmAllNmMessagesKeepAwake> is enabled, ECU might still be kept alive, even if no relevant PNC request was received.

[SWS_Nm_00312] ⌈If Nm_PncBitVectorRxIndication is called, NmPnEiraCalcEnabled or NmPnEraCalcEnabled is set to TRUE and at least one PNC Request Bit is detected as relevant PNC request in the given PNC Request Bit Vector, the OUT parameter RelevantPncRequestDetectedPtr shall be set to TRUE. Otherwise the value of RelevantPncRequestDetectedPtr shall be set to FALSE.⌋

Note: The value of RelevantPncRequestDetectedPtr is used by the caller **<Bus>Nm** to qualify if the received NMPDU shall be considered for further Rx Indication handling.

Example: The given PNC Request Bit Vector has a length of 2 bytes (NmPncBitVectorLength):

Byte 0	**Byte 1**
PNC Request Bit Vector	
0x12	0x8E

Table 7.1: Example of PNC Request Bit Vector

For this example two NmPnFilterMaskBytes would be defined, e.g

- NmPnFilterMaskByteIndex = 0 with NmPnFilterMaskByteValue = 0x01
- NmPnFilterMaskByteIndex = 1 with NmPnFilterMaskByteValue = 0x97

The filter algorithm actions and result would then be:

Filter Mask Value (Byte)	Compared to received **PNC Request Bit Vector**	Resulting in
0x01 (Byte 0)	0x12 (NM PDU Byte 4)	0x00 (no relevant PNC request)
0x97 (Byte 1)	0x8E (NM PDU Byte 5)	0x86 (relevant PNC request)

Table 7.2: Example PN Filter Algorithm

As one byte contains relevant information, the value of the boolean parameter `RelevantPncRequestDetectedPtr` is set to TRUE.

[SWS_Nm_CONSTR_00001] ⌈The length of all configured `NmPnFilterMaskByte`s shall have the same length (in bytes) as the `NmPncBitVectorLength` of the corresponding NM-channel. A configuration tool shall reject a configuration as invalid (error), if the length of the configured `NmPnFilterMaskByte`s differ from the configured `NmPncBitVectorLength` of the corresponding NM-channel.⌋

7.4.2 Aggregation of PNC requests

7.4.2.1 Aggregation of internal and external Partial Network Cluster

[SWS_Nm_00302]

Upstream requirements: SRS_ModeMgm_09248

⌈If `NmPnEiraCalcEnabled` is set to TRUE, then Nm shall provide the possibility to store external and internal requested PNCs combined over all NM-channels where `NmPnEnabled` is set to TRUE. At initialization the values of all PNCs within EIRA shall be set to 0 (PNC released).⌋

[SWS_Nm_00313] ⌈If

- `NmPnEiraCalcEnabled` is TRUE
- and a `PNC Request Bit Vector` is received via `Nm_PncBitVectorRxIndication` or `Nm_ForwardSynchronizedPncShutdown`
- and PNCs are requested within this message (bits set to 1)
- and the requested PNCs are set to 1 within the configured PN filter mask

then `NM` shall store the request information (value 1) for these PNCs as "PNC requested".⌋

[SWS_Nm_00534] ⌈If

- `NmPnEiraCalcEnabled` is set to TRUE

- and a transmission of a PNC Request Bit Vector is indicated via Nm_PncBitVectorTxIndication
- and no requests for a synchronized PNC shutdown are pending on the given NM-channel
- and PNCs are requested within the stored unfiltered PNC Request Bit Vector (bits set to 1)
- and the requested PNCs of the stored unfiltered PNC Request Bit Vector for internal PNC requests are set to 1 within the configured PN filter mask

then NM shall store the request information (value 1) for these PNCs as "PNC requested".⌋

[SWS_Nm_00535] ⌈If NmPnEiraCalcEnabled is TRUE, Nm_PncBitVectorTxIndication is called and no requests for synchronized PNC shutdown are pending on the given NM-channel, then Nm shall copy the unfiltered PNC Request Bit Vector for internal PNC requests of the given NM-channel to the buffer indicated by PncRequestBitVectorPtr.⌋

[SWS_Nm_00536] ⌈If

- NmPnEiraCalcEnabled is set to TRUE
- and requests for a synchronized PNC shutdown are pending on the given NM-channel
- and a transmission of a PNC Request Bit Vector is indicated via Nm_PncBitVectorTxIndication
- and PNCs are requested for a synchronized PNC shutdown within the PNC Request Bit Vector (bits set to 1)
- and the aggregated synchronized PNC requests of the PNC Request Bit Vector are set to 1 within the configured PN filter mask,

then Nm shall store the request information (value 1) for these PNCs as "PNC requested".⌋

[SWS_Nm_00537] ⌈If NmPnEiraCalcEnabled is set to TRUE, Nm_PncBitVectorTxIndication is called and requests for synchronized PNC shutdown are pending on the given NM-channel, then Nm shall set the buffer indicated by PncRequestBitVectorPtr as follows:

- set all according bits to 1 that relate to a PNC requested for synchronized PNC shutdown of the given NM-channel
- set all other bits to 0

⌋

Note: The Nm module has to aggregate all PNCs which were indicated for a synchronized PNC shutdown and transfer the pncId's to a byte array (PNC Request Bit Vector). Each bit (PNC Request Bit) of the PNC Request Bit Vector represent a particular PNC. The byteIndex and bitindex within the PNC Request Bit Vector of PNC Request Bit can be determined as follows:

- byteIndex = (PncId div 8)
- bitIndex = (PncId mod 8)

[SWS_Nm_00330]

Upstream requirements: SRS_ModeMgm_09248

⌈For all configured NM-channel where NmPartialNetworkSupportEnabled is set TRUE, Nm shall provide the possibility to store external and internal requested PNCs combined over all NM-channels where NmPnEnabled is set to TRUE. At initialization the values of all PNCs within EIRA shall be set to 0 (PNC released).⌋

[SWS_Nm_00317]

Upstream requirements: SRS_ModeMgm_09250

⌈If Nm_UpdateIRA is called, the NM shall store the received unfilterred PncRequestBitVector per given NM-channel. Therefore, the NM module shall copy the amount of bytes with respect of the configured length (see NmPncBitVectorLength) of the given NM-Channel.⌋

Note: NM stores unfilterred internal PNC requests in order to support the possibility of requesting a PNC on a specific channel even if the PNC is not assigned to the channel.

[SWS_Nm_00318] ⌈If NmPnEiraCalcEnabled is TRUE, then NM shall provide a possibility to monitor each relevant PNC in the context of Nm_MainFunction. The monitoring shall consider the PNC state, if the PNC is still externally or internally requested on at least one of the relevant NM-channels.⌋

Note: This means, only one timer is required to handle one PNC on multiple connected physical channels. For example: only 8 EIRA reset timers are required to handle the requests of a Gateway with 6 physical channels and 8 partial network clusters. This is possible because the switch of PNC related PDU-Groups is done global for the ECU and independent of the physical channel.

[SWS_Nm_00319] ⌈If NmPnEiraCalcEnabled is TRUE and every time a PNC is stored as "PNC requested" (see [SWS_Nm_00313]), then the monitoring for this PNC shall be restarted with respect to NmPnResetTime in the context of Nm_MainFunction.⌋

Note: NmPnResetTime has to be configured to a value greater than <Bus>NmMsgCycleTime. If NmPnResetTime is configured to a value smaller than <Bus>NmMsgCycleTime and only one ECU requests the PNC, the request state toggles in the EIRA,

because the request state reset before the requesting ECU is able to transmit the PNC Request Bit Vector within the next NM message.

Note: NmPnResetTime has to be configured to a value smaller than <Bus>NmTimeoutTime to avoid that the timer elapses after **<Bus>Nm** already changed to a state where it is expected that application communication is disabled (e.g. change to Prepare Bus Sleep (**UdpNm**, **CanNm**) or Bus Sleep (**FrNm**)).

[SWS_Nm_00320] ⌈If NmPnEiraCalcEnabled is TRUE and a PNC is not requested again within NmPnResetTime the corresponding stored value for this PNC shall be set to PNC released (value 0) in the context of Nm_MainFunction.⌋

[SWS_Nm_00321]

Upstream requirements: SRS_ModeMgm_09250

⌈If NmPnEiraCalcEnabled is TRUE and the stored value for a PNC is set to PNC requested or back to PNC released (see [SWS_Nm_00313] and [SWS_Nm_00320]), then NM shall forward the changed state of EIRA to ComM by calling ComM_Nm_UpdateEIRA(<PncRequestBitVector of EIRA>)).⌋

Note: If a PN shutdown message is received (PNSR is set to 1), no special handling is needed, because the according PNC state machines need to stay in COMM_PNC_READY_SLEEP. Only the ERA PDU is handled in a different way (see [SWS_Nm_00305])

7.4.2.2 Aggregation of external Partial Network Cluster

[SWS_Nm_00322] ⌈If NmPnEraCalcEnabled is TRUE, then NM shall provide the possibility to store relevant external PNCs request per channel. At initialization the values of all PNCs shall be set to 0 (PNC released).⌋

[SWS_Nm_00323] ⌈If

- NmPnEraCalcEnabled is TRUE
- and a PNC Request Bit Vector is received via Nm_PncBitVectorRxIndication
- and PNCs are requested within this message (bits set to 1)
- and the requested PNCs are set to 1 within the configured PN filter mask

then NM shall store the PNC request information (value 1) for these PNCs as "PNC requested" per given channel.⌋

[SWS_Nm_00324] ⌈If NmPnEraCalcEnabled is TRUE, then NM shall provide a possibility to monitor each relevant PNC per channel in the context of Nm_MainFunction. The monitoring shall consider the PNC state, if the PNC is still externally requested on the corresponding NM-channels.⌋

Note: This means, a separate timer is required to handle one PNC on multiple physical channels. For example: 48 ERA PNC reset timers are required to handle the external PNC requests of a PNC gateway with 6 physical channels and 8 partial network clusters. It is not possible to combine the PNC reset timer as it is possible for EIRA PNC timers, because the external PNC request mustn't be mirrored back to the bus / network from where the PNC request was received. Thus, it is required to detect the physical channel that is the source of the PNC request.

[SWS_Nm_00325] ⌈If NmPnEraCalcEnabled is TRUE, then every time a PNC is stored as "PNC requested" (see [SWS_Nm_00323]), the monitoring for this PNC shall be restarted with respect to NmPnResetTime in the context of Nm_MainFunction.⌋

Note: NmPnResetTime has to be configured to a value greater than <Bus>NmMsgCycleTime. If NmPnResetTime is configured to a value smaller than <Bus>NmMsgCycleTime and only one ECU requests the PNC, the request state toggles in the EIRA because the request state is rested before the requesting ECU is able to transmit the PNC Request Bit Vector within the next NM message.

Note: NmPnResetTime has to be configured to a value smaller than <Bus>NmTimeoutTime to avoid that the timer elapses after **<Bus>Nm** already changed to a state where it is expected that application communication is disabled (e.g. change to Prepare Bus Sleep (**UdpNm**, **CanNm**) or Bus Sleep (**FrNm**)).

[SWS_Nm_00326] ⌈If NmPnEraCalcEnabled is TRUE and a PNC is not requested again within NmPnResetTime the corresponding stored value for this PN shall be set to PNC released (value 0).⌋

[SWS_Nm_00327]

Upstream requirements: SRS_ModeMgm_09250

⌈If NmPnEraCalcEnabled is TRUE and the stored value for a PNC is set to PNC requested or back to PNC released (see [SWS_Nm_00323] and [SWS_Nm_00326]), then NM shall forward the changed state of ERA of the affected NM-channel to ComM by calling ComM_Nm_UpdateERA(<ComMChannel>, <PncRequestBitVector of ERA>).⌋

[SWS_Nm_00331] ⌈If NmPnEiraCalcEnabled is TRUE and NmPnEraCalcEnabled is TRUE , the PNC status information has to be stored separately for both, the EIRA and ERA information (compare [SWS_Nm_00302] and [SWS_Nm_00322]).⌋

7.4.3 EIRA / ERA state and PNC reset timer handling

PNC reset timers for ERA and EIRA are handeled in the context of Nm_Mainfunction. PNC reset timers are used for the monitoring of PNC requests. Based on the current available PNC requests and the current state of the corresponding PNC reset timers, particular actions have to be performed (e.g. re-start PNC timer, set requested PNC to PNC released)

[SWS_Nm_00328] ⌈If NmPartialNetworkSupportEnabled is set to TRUE, then all PNC reset timers shall be stopped at initialization.⌋

[SWS_Nm_00329] ⌈If NmPartialNetworkSupportEnabled is set to TRUE, then the evaluation of the PNC states in combination with PNC reset timers shall consider at least the following order:

- Update PNC reset timers
- Evaluate the PNC states
- Perform actions based on the current state of PNC reset timer and PNC requests

⌋

The monitoring of PNC requests and the PNC reset timer handling is kept as implementation specific. The following section shows an example, how the EIRA / ERA evaluation and PNC reset timer could be handeled in the main function of NM. This is descriped to support the understanding of the overall mechanism of the PNC handling in NM.

Example: The following example is based on the following assumptions:

- each PNC reset timer has 3 states(stopped, elasped, running)
- if a PNC reset timer is started, the PNC reset timer is loaded with NmPnResetTime
- a running PNC reset timer is decremented in each call of the main function, until it reaches value '0'
- PNC gateway functionality is enabled:
  - One ERA as PNC Request Bit Vector exists per channel
  - One EIRA as PNC Request Bit Vector exist

At initialization the PNC reset timers are in state "stopped" and the EIRA / ERA PNC Request Bit Vectors are set value '0' (PNC released). The PNC reset timers are started if a relevant PNC (PNC that pass the PN filter mask) is received where the value is set to '1' (e.g. [SWS_Nm_00325]). In each main function all PNC timers are decremented as first step. Afterwards the current PNC requests are evaluated in combination with the state of the according PNC reset timer. Particiular actions are performed based on the evaluation result. As final step the EIRA / ERA PNC Request Bit Vectors are erased (set to '0'). This is needed to refresh the current state of PNC requests until the next main function call and to detect changes from PNC requested to PNC released and vice versa. (Please note: EIRA is always updated if a PNC Request Bit Vector with relevant PNC requests is received or if a PNC Request Bit Vector with relevant PNC requests is transmitted. ERA is always updated if a PNC Request Bit Vector with relevant PNC request of the according channel is received.) Based on this example EIRA and ERA is used to store a snapshot of PNC requests between

2 main function calls. In each main function call PNC requests conveyed from the EIRA / ERA storage to either PNC reset timer and/or as change to ComM.

Table 7.3 shows the evaluation details of the example. Column "EIRA / ERA PNC state" and "PNC reset timer state" is the input and column "Evalution result" is the output of the evaluation. The output describes which action has to be performed.

EIRA / ERA PNC state)	PNC reset timer state	Evaluation result
PNC requested	Stopped	Start PNC reset timer, set PNC reset timer state to "running" and inform ComM regarding the PNC state change
PNC requested	Elapsed	Restart PNC reset timer, set PNC reset timer state to "running" and inform ComM regarding the PNC state change
PNC requested	Running	Restart PNC reset timer
PNC released	Stopped	Do nothing
PNC released	Elapsed	Set PNC reset timer to "stopped" and inform ComM regarding the PNC state change
PNC released	Running	Do nothing (Please note: time of the PNC reset timer was already decrement as very first action in the main function)

Table 7.3: Example for EIRA / ERA PNC state handling in combination with PNC reset timer state.

7.4.4 Synchronized PNC shutdown functionality

The synchronized PNC shutdown is a functionality which is a cooperation of **ComM**, **Nm** and **<Bus>Nm** to ensure a synchronized PNC shutdown at almost the same point in time across the whole PN topology. A synchronized PNC shutdown is handled by ECUs in role of a top-level PNC coordinator or intermediate PNC coordinator and where the PNC gateway is enabled. If **ComM** of an ECU in the role of a top-level PNC coordinator detects that a PNC is released, the **ComM** requests a synchronized PNC shutdown by calling Nm_RequestSynchronizedPncShutdown per ComMChannel and ComMPnc. The Nm module store all requests and handle them in the context of the Nm_Mainfunction. The Nm module indicate the affected **<Bus>Nms** regarding an activated PNC shutdown process. The **<Bus>Nms** call the Nm module to provide the aggregated requests for a synchronized PNC shutdown as PNC Request Bit Vector per given NM-channel. The **<Bus>Nms** use the provided PNC Request Bit Vector to assemble a NM-PDU as PN shutdown message and transmit this NM message on the according NM channel. If a PN shutdown message is received by an ECU in the role of an intermediate PNC coordinator, the **<Bus>Nm**s extract the PNC Request Bit Vector from the received PN shutdown message and forward the information by calling the callback function **Nm_ForwardSynchronizedPncShutdown**. The callback function will immediately forward the indication to **ComM** by calling ComM_Nm_ForwardSynchronizedPncShutdown. **ComM** will immediately request a synchronized PNC shutdown of all actively PNC coordinated (coordinated by a PNC

gateway) ComMChannels. The requests for a synchronized PNC shutdown are forwarded to Nm module per NM-channel and handled in the same way as described in the previous section.

If a PNC leaf node receives a top-level PNC coordinator Nm frame, then it will handle the frame as a usual NM message (update the local PNC Request Bit Vector and reset PN reset time).

[SWS_Nm_00533] ⌈If NmPartialNetworkSupportEnabled is set to TRUE, then PNC shutdown handling shall be considered as deactivated at initialization.⌋

[SWS_Nm_00521] ⌈If function Nm_RequestSynchronizedPncShutdown is called, NmSynchronizedPncShutdownEnabled is set to TRUE and NmStandardBusType of the given NM-channel is set to a type other than NM_BUSNM_LOCALNM, the Nm module shall store the given PNC (PncId) per given NM-channel (NetworkHandle) as a pending request for a synchronized PNC shutdown.⌋

[SWS_Nm_00305] ⌈If the reception of PN shutdown message via callback function Nm_ForwardSynchronizedPncShutdown is indicated, then NM shall stop the ERA related monitoring of external PNC requests of the PNCs (see [SWS_Nm_00324])) which are indicated for an PNC shutdown (PNC Request Bit set to '1') in the given PNC Request Bit Vector, set the corresponding ERA bits to '0' in the ERA of the indicated NM-channel and forward the indication to ComM with the corresponding ComM-Channel by calling ComM_Nm_ForwardSynchronizedPncShutdown(<ComMChannel>, <PncRequestBitVector>).⌋

Note:

- The PNC Request Bit Vector of a received PN shutdown message shall be used to release the PNCs for a synchronized shutdown. Explicitly clear the affected ERA bits of the indication NM-channel, stop the ERA monitoring for the indicated PNCs of the indicated NM-channel and pass the indication for a synchronized PNC shutdown to the ComM module. The synchronized PNC shutdown has to be handled as fast as possible. Therefore, the ComM module is informed immediately.
- Stopping the ERA related monitoring should not result in a call of ComM_UpdateERA. ComM ensure a proper handling of the ComM internal ERA bits within the context of ComM_Nm_ForwardSynchronizedPncShutdown. This should support an unambiguous handling of the PNC state machine for a synchronized PNC shutdown

[SWS_Nm_00525] ⌈If NmPnShutdownMessageRetransmissionDuration is configured and Nm_RequestSynchronizedPncShutdown is called (refer to [SWS_Nm_00523]), then the corresponding retransmission timer for PN shutdown messages shall be started with NmPnShutdownMessageRetransmissionDuration on all affected NM-channels.⌋

[SWS_Nm_00523] ⌈If NmSynchronizedPncShutdownEnabled is set to TRUE, requests for synchronized PNC shutdown are pending and PNC shutdown handling has been deactived, then Nm shall activate the PNC shutdown handling by calling <Bus>Nm_ActivateTxPnShutdownMsg.⌋

[SWS_Nm_00524] ⌈If NmSynchronizedPncShutdownEnabled is set to TRUE, no requests for synchronized PNC shutdown are pending, PNC shutdown handling has been activated, then Nm shall deactivate the PNC shutdown handling by calling <Bus>Nm_DeactivateTxPnShutdownMsg.⌋

[SWS_Nm_00527] ⌈If NmSynchronizedPncShutdownEnabled is set to TRUE, PNC shutdown handling has been activated (refer to [SWS_Nm_00523]) and Nm_PncBitVectorTxConfirmation is called with E_OK, then the Nm module shall consider those PNC IDs, stored as pending request for a synchronized PNC shutdown of the given NM-channel, as completed which were indicated by the given PNC Request Bit Vector (PNC Request Bit set '1') and remove them from storage. Additionally, if NmPnShutdownMessageRetransmissionDuration is configured, then the Nm module shall cancel the retransmission timer for PN shutdown messages of the affected NM-channel.⌋

Note:

- Nm has to ensure that new request for a synchronized PNC shutdown (indicated via Nm_RequestSynchronizedPncShutdown) are not lost, during an ongoing PNC shutdown process.
- The <Bus>Nm has to handle the re-transmission of the NM-PDU as PN shutdown message as long as Nm_PncBitVectorTxConfirmation is called with **E_NOT_OK** or the re-transmission timer for PN shutdown messages is running.

[SWS_Nm_00529] ⌈If NmSynchronizedPncShutdownEnabled is set to TRUE and the Nm module has stored PNC IDs as pending request for a synchronized PNC shutdown, then Nm shall remove those PNC IDs from storage which are either externally or internally requested again:

- Nm shall check on reception of an PNC Request Bit Vector via call of Nm_PncBitVectorRxIndication, if externally requested PNCs are received.
- Nm shall check on update of an PNC Request Bit Vector via call of Nm_UpdateIRA, if internal PNC requests are available .

⌋

[SWS_Nm_00530] ⌈If NmSynchronizedPncShutdownEnabled is set to TRUE, NmPnShutdownMessageRetransmissionDuration is not configured, the corresponding <Bus>Nm module of this NM-channel has indicated transmission request via call of Nm_PncBitVectorTxIndication and Nm_PncBitVectorTxConfirmation is called with result **E_NOT_OK** (transmission of the given PNC Request Bit Vector failed), then the Nm shall remove the PNC IDs stored as pending

request for a synchronized PNC shutdown of the corresponding NM-channel and report the runtime error NM_E_TRANSMISSION_OF_PN_SHUTDOWN_MESSAGE_FAILED to **DET**.⌋

[SWS_Nm_00532] ⌈If NmSynchronizedPncShutdownEnabled is set to TRUE and a retransmission timer for a PN shutdown message (see NmPnShutdownMessageRetransmissionDuration) expires, then Nm shall remove the pending request for a synchronized PNC shutdown of the corresponding NM-channel from the storage and report the runtime error NM_E_TRANSMISSION_OF_PN_SHUTDOWN_MESSAGE_FAILED to **DET**.⌋

Note: If a retransmission timer for PN shutdown messages is configured and the NM message was not transmitted successfully (Nm_PncBitVectorTxConfirmation is called with result **E_NOT_OK**), then the duration of PNC shutdown process continue. In the worst case the PN shutdown process coincide with a post poned NM message transmitted with **<Bus>NmMsgCycleTime**. But in any case, if the capability to transmitted NM messages is not re-covered within the PN reset time (EIRA), the PNCs will shutdown not synchronized, which might lead to timeout errors on application level.

7.5 Prerequisites of bus specific Network Management modules

This chapter gives an overview of the API calls that are used for the *Basic functionality*, the *NM Coordination functionality* and *synchronized PNC shutdown functionality* as well as information on the expected behavior of the **bus specific NM** for both functionalities.

For specific requirements of the interfaces and the configuration parameters for enabling/disabling the API's, refer to chapter 8.

7.5.1 Prerequisites for basic functionality

The **Nm** only acts as a forwarding layer between the **ComM** and the **bus specific NM** for the *basic functionality*.
All API calls made from the upper layer shall be forwarded to the corresponding API call of the lower layer. All callbacks of **Nm** invoked by the lower layer shall be forwarded to the corresponding callback of the upper layer.

The *Basic functionality* provides the following API calls to the ComM:

- Nm_NetworkRequest - [SWS_Nm_00032]
- Nm_NetworkRelease - [SWS_Nm_00046]
- Nm_PassiveStartUp - [SWS_Nm_00031]

Note: This implies that the **bus specific NM** provides the corresponding functions `<Bus>Nm_NetworkRequest`, `<Bus>Nm_NetworkRelease` and `<Bus>Nm_PassiveStartUp`.

The *Basic functionality* forwards the following API callbacks to the **ComM**:

- `Nm_NetworkStartIndication` - [SWS_Nm_00154]
- `Nm_NetworkMode` - [SWS_Nm_00156]
- `Nm_BusSleepMode` - [SWS_Nm_00162]
- `Nm_PrepareBusSleepMode` - [SWS_Nm_00159]

Note: This implies that the **ComM** provides the corresponding callback functions `ComM_Nm_NetworkStartIndication`, `ComM_Nm_NetworkMode`, `ComM_Nm_BusSleepMode` and `ComM_Nm_PrepareBusSleepMode`.

The **Nm** provides a number of API calls to the upper layers that are not used by **ComM**. These are provided for OEM specific extensions of the NM stack and are not required by any AUTOSAR module. They shall be forwarded to the corresponding API calls provided by the **bus specific NM**s.

The *Basic functionality* provides the following API calls to any OEM extension of an upper layer:

- `Nm_DisableCommunication` - [SWS_Nm_00033]
- `Nm_EnableCommunication` - [SWS_Nm_00034]
- `Nm_SetUserData` - [SWS_Nm_00035]
- `Nm_GetUserData` - [SWS_Nm_00036]
- `Nm_GetPduData` - [SWS_Nm_00037]
- `Nm_RepeatMessageRequest` - [SWS_Nm_00038]
- `Nm_GetNodeIdentifier` - [SWS_Nm_00039]
- `Nm_GetLocalNodeIdentifier` - [SWS_Nm_00040]
- `Nm_CheckRemoteSleepIndication` - [SWS_Nm_00042]
- `Nm_GetState` - [SWS_Nm_00043]

Note: This implies that the **bus specific NM** optionally provides the corresponding functions.

7.5.2 Prerequisites for NM Coordinator functionality

The `coordination algorithm` makes use of the following interfaces of the **bus specific NM**:

- <Bus>Nm_NetworkRequest - [SWS_Nm_00119]
- <Bus>Nm_NetworkRelease - [SWS_Nm_00119]
- <Bus>Nm_RequestBusSynchronization - [SWS_Nm_00166]
- <Bus>Nm_CheckRemoteSleepIndication - [SWS_Nm_00166]

Note: All NM networks configured to be part of a coordinated cluster of the *NM coordinator functionality* must have the corresponding Bus NM configured to be able to actively send out NM messages (e.g. CANNM_PASSIVE_MODE_ENABLED = false). As a result of this configuration restriction, all **<Bus>Nm** used by the *coordinator functionality* of the Nm module must provide the API <Bus>Nm_NetworkRequest.

Note: Any configuration where a network is part of a coordinated cluster of networks where the corresponding **<Bus>Nm** is configured as passive is invalid.

Note: The <Bus>Nm_RequestBusSynchronization is called by **Nm** immediately before <Bus>Nm_NetworkRelease in order to allow non-synchronous networks to synchronize before the network is released. For some networks, this call has no meaning. The **bus specific NM** shall still provide this interface in order to support the generality of the *NM Coordinator functionality*, but can choose to provide an empty implementation.

Rationale: The <Bus>Nm_CheckRemoteSleepIndication is never explicitly mentioned in the coordination algorithm. Its use is dependent on the implementation.

The coordination algorithm requires that the following callbacks of the **Nm** can be invoked by the **bus specific NM**:

- Nm_NetworkStartIndication - [SWS_Nm_00154]
- Nm_NetworkMode - [SWS_Nm_00156]
- Nm_BusSleepMode - [SWS_Nm_00162]
- Nm_PrepareBusSleepMode - [SWS_Nm_00159]
- Nm_SynchronizeMode - [SWS_Nm_91002]
- Nm_RemoteSleepIndication - [SWS_Nm_00192]
- Nm_RemoteSleepCancellation - [SWS_Nm_00193]
- Nm_SynchronizationPoint - [SWS_Nm_00194]

Note: The Nm_NetworkStartIndication, Nm_NetworkMode, Nm_BusSleepMode and Nm_PrepareBusSleepMode are used by the coordination algorithm to keep track of the status of the different networks and to handle aborted shutdown (see Chapter 7.3.3).

Note: The Nm_RemoteSleepIndication and Nm_RemoteSleepCancellation are used by the coordination algorithm to determine when all conditions for

initiating the coordinated shutdown are met. The indication will be called by the **bus specific NM** when it detects that all other nodes on the network (except for itself) is ready to go to "bus-sleep mode". Some implementations will also make use of the API call <Bus>Nm_CheckRemoteSleepIndication.

Note: A **bus specific NM** which is included in a coordination cluster must monitor its bus to identify when all other nodes on the network is ready to go to sleep. When this occurs, the **bus specific NM** shall call the callback Nm_RemoteSleepIndication of **Nm**. (See [SWS_Nm_00192]).

Note: After a **bus specific NM** which is included in a coordination cluster has signaled to **Nm** that all other nodes on the network is ready to go to sleep (See [SWS_Nm_00192]), it must continue monitoring its bus to identify if any node starts requesting the network again, implying that the bus is no longer ready to go to sleep. When this occurs, the **bus specific NM** shall call the callback Nm_RemoteSleepCancellation of **Nm**. (See [SWS_Nm_00193]).

Note: The Remote Sleep Indication and Cancellation functionality is further specified in the respective bus specific NM.

Rationale: The Nm_SynchronizationPoint shall be called by the **bus specific NM** in order to inform the coordination algorithm of a suitable point in time to initiate the coordinated shutdown. For cyclic networks this is typically at cycle boundaries. For non-cyclic networks this must be defined by other means. Each *NM Coordination Cluster* can be configured to make use of synchronization indications or not (See [SWS_Nm_00172]), and if they are used, the coordination algorithm filters indications and only acts on indications from networks that are configured as synchronizing networks.

Note: Please note for implementation of <bus>Nm: Cyclic networks invoke the Nm_SynchronizationPoint repeatedly when no other nodes request the network. The invocation is typically made at boundaries in the **bus specific NM** protocol when changes in the NM voting will occur.

It is assumed that any call to <Bus>Nm_ReleaseNetwork made between two of these Nm_SynchronizationPoint will be acted upon at the same point in time as the next Nm_SynchronizationPoint would have been invoked.

Rationale: The synchronization indication shall start when Nm_RemoteSleep-Indication has been notified and continue until either the network has been released (<Bus>Nm_NetworkRelease) or the Nm_RemoteSleepCancellation is called.

Note:: For the use case of coordinating Flexray-channel A + B if there is no other Network inside the NM Cluster, hence,if an NM Coordinator contains only one NM Channel, the NmActiveCoordinator for this NmChannelConfig needs to be set to TRUE and the NmChannelSleepMaster needs be set to FALSE to allow the channel to coordinate itself . Note: The Value of "NmSynchronizingNetwork" is only relevant if this network is in the same coordination cluster with other networks.

7.5.3 Prerequisites of Partial Network functionality

7.5.3.1 Prerequisite for aggregation of PNC requests

The aggregation of PNC requests, requires that the following callback function of the Nm can be invoked by the bus specific NM:

- `Nm_PncBitVectorRxIndication`
- `Nm_PncBitVectorTxIndication`

The aggregation of PNC requests functionality provides the following API, to be called by the ComM:

- `Nm_UpdateIRA`

7.5.3.2 Prerequisites for synchronized PNC shutdown functionality

The synchronized PNC shutdown functionality makes use of the following interface of the bus specific NM:

- `<Bus>Nm_RequestSynchronizedPncShutdown` - [SWS_Nm_00166][SWS_Nm_00521]

The synchronized PNC shutdown functionality requires that the following callback of the Nm can be invoked by the bus specific NM:

- `Nm_ForwardSynchronizedPncShutdown` - [SWS_Nm_91007]

The synchronized PNC shutdown functionality provides the following API, to be called by the ComM:

- `Nm_RequestSynchronizedPncShutdown` - [SWS_Nm_91005]

7.5.4 Configuration of global parameters for bus specific networks

The **Nm**'s configuration contains parameters that regulate support of optional features found in the **bus specific NM**s. Since **Nm** is only a pass-through interface layer regarding features that are not used by the *NM Coordinator functionality*, enabling these in **Nm**'s configuration will in many cases only enable the pass-through of the controlling API functions and the callback indications from the bus specific layers.

Many of the parameters defined for NM are used only as a source for global configuration of all bus specific NM modules. Corresponding parameters of the bus specific NMs are derived from these parameters.

7.6 NM_BUSNM_LOCALNM

[SWS_Nm_00483] ⌈If BusNmType is NM_BUSNM_LOCALNM and **ComM** requests Nm_PassiveStartUp or Nm_NetworkRequest then **Nm** shall inform **ComM** about start of network by calling ComM_Nm_NetworkMode.
Rationale : Buses of type NM_BUSNM_LOCALNM which are coordinated do not have a network management message but are synchronized e.g. by a master - slave concept like LIN). These Bus-Types are always directly started on request by **ComM** but the shutdown will be done by coordinator algorithm.⌋

7.7 Multicore Distribution

In its role as central module dealing with different network types the Nm interaction spans across partitions in case the Com-Stack is distributed and so shall provide required multi-core features to ensure a clean architecture and keep the network dependent clusters free of multi-partition (multi-core) addons.

Note: The Nm module is expected to apply appropriate mechanisms to allow calls of its APIs from other partitions than its main function, e.g. by providing a Nm satellite.

[SWS_Nm_00485]

Upstream requirements: SRS_BSW_00459

⌈Nm shall interact with <Bus>Nm (i.e. call <Bus>Nm APIs) only in the partition, where the respective <Bus>Nm module is assigned to.⌋

[SWS_Nm_00486]

Upstream requirements: SRS_BSW_00459

⌈The Nm kernel shall be assigned to the same partition as ComM kernel in order to keep the interaction between these two modules on an intra-partition basis.⌋

Note: Even though the basic software (and the Com-Stack in particular) is distributed across several partitions, ComM [9] and Nm Masters should reside in the same partition in order to keep mode interfaces between the two modules simple (for further information see chapter Master/Satellite-approach in [10, Guide to BSW Distribution]).

7.8 Additional Functionality

7.8.1 Nm_CarWakeUpIndication

[SWS_Nm_00252] ⌈If the <bus>Nm calls Nm_CarWakeUpIndication and NmCarWakeUpCallout is defined, the NM Interface shall call the callout function defined by NmCarWakeUpCallout with nmNetworkHandle as parameter.⌋

[SWS_Nm_00285] ⌈If the <bus>Nm calls `Nm_CarWakeUpIndication` and `NmCarWakeUpCallout` is not defined, the NM Interface shall call the function BswM_Nm_CarWakeUpIndication with nmNetworkHandle as parameter.⌋

Note: The application, called by `NmCarWakeUpCallout`, is responsible to manage the Car Wake Up (CWU) request and distribute the Request to other Nm channels by setting the CWU bit in its own Nm message. This application drops the CWU request if the request is not repeated within a specific time.

Note: The callout is declared as specified within SWS_BSW_00039 and SWS_BSW_00135.

7.8.2 Nm_StateChangeNotification

[SWS_Nm_00249] ⌈If `NmStateReportEnabled` is set to **TRUE** and `NmStateReportSignalRef` is configured, when one of the state transitions mentioned in [SWS_Nm_00509] occur, `Nm_StateChangeNotification` shall call `Com_SendSignal(uint8, Com_SignalIdType, const void*)` for the signal referenced by `NmStateReportSignalRef` with the value according to [SWS_Nm_00509].⌋

Note: The transmitted signal has to be at least a 6 bit signal in Com that should be part of the NM message.

[SWS_Nm_00487] ⌈When `Nm_StateChangeNotification` is called to report a change of the Nm state, Nm shall call `BswM_Nm_StateChangeNotification()` with the reported current state.⌋

[SWS_Nm_00509] Definiton of network management states in Nm module ⌈

Bit	Value	Name	Description
0	1	NM_RM_BSM	NM in state RepeatMessage (transition from BusSleepMode)
1	2	NM_RM_PBSM	NM in state RepeatMessage (transition from PrepareBusSleepMode)
2	4	NM_NO_RM	NM in state NormalOperation (transition from RepeatMessage)
3	8	NM_NO_RS	NM in state NormalOperation (transition from ReadySleep)
4	16	NM_RM_RS	NM in state RepeatMessage (transition from ReadySleep)
5	32	NM_RM_NO	NM in state RepeatMessage (transition from NormalOperation)

⌋

[SWS_Nm_00501] ⌈If `NmDynamicPncToChannelMappingEnabled` is set to `TRUE` and `Nm_StateChangeNotification` is called, then

ComM_Nm_RepeatMessageLeftIndication() shall be called when nmPreviousState is set to NM_STATE_REPEAT_MESSAGE and nmCurrentState is different from NM_STATE_REPEAT_MESSAGE.⌋

7.9 Error classification

Chapter [2, General Specification of Basic Software Modules] 7.2 *"Error Handling"* describes the error handling of the Basic Software in detail. Above all, it constitutes a classification scheme consisting of five error types which may occur in BSW modules.

Based on this foundation, the following section specifies particular errors arranged in the respective subsections below.

7.9.1 Development Errors

[SWS_Nm_00232] Definition of development errors in module Nm

Upstream requirements: SRS_BSW_00327, SRS_BSW_00337, SRS_BSW_00385, SRS_BSW_00386

⌈

Type of error	*Related error code*	*Error value*
API service used without Nm interface initialization	NM_E_UNINIT	0x00
API Service called with wrong parameter but not with NULL-pointer	NM_E_INVALID_CHANNEL	0x01
API service called with a NULL pointer	NM_E_PARAM_POINTER	0x02

⌋

7.9.2 Runtime Errors

[SWS_Nm_91011] Definition of runtime errors in module Nm

Upstream requirements: SRS_BSW_00385, SRS_BSW_00452

⌈

Type of error	*Related error code*	*Error value*
Retransmission timer for a PN shutdown message has expired, because a PN shutdown message could not be transmitted on the network within the configured duration of re-transmission.	NM_E_TRANSMISSION_OF_PN_SHUTDOWN_MESSAGE_FAILED	0x10

⌋

7.9.3 Production Errors

There are no production errors.

8 API specification

8.1 Imported types

In this chapter all types included from the following modules are listed.

[SWS_Nm_00117] Definition of imported datatypes of module Nm

Upstream requirements: SRS_BSW_00301

⌈

Module	*Header File*	*Imported Type*
Com	Com.h	Com_SignalIdType
Comtype	ComStack_Types.h	NetworkHandleType
	ComStack_Types.h	PNCHandleType
Std	Std_Types.h	Std_ReturnType

⌋

8.2 Type definitions

8.2.1 Nm_ModeType

[SWS_Nm_00274] Definition of datatype Nm_ModeType ⌈

Name	Nm_ModeType		
Kind	Enumeration		
Range	NM_MODE_BUS_SLEEP	–	Bus-Sleep Mode
	NM_MODE_PREPARE_ BUS_SLEEP	–	Prepare-Bus Sleep Mode
	NM_MODE_ SYNCHRONIZE	–	Synchronize Mode
	NM_MODE_NETWORK	–	Network Mode
Description	Operational modes of the network management.		
Available via	NmStack_types.h		

⌋

8.2.2 Nm_StateType

[SWS_Nm_00275] Definition of datatype Nm_StateType ⌈

Name	Nm_StateType		
Kind	Enumeration		
Range	NM_STATE_UNINIT	0x00	Uninitialized State

▽

△

	NM_STATE_BUS_SLEEP	0x01	Bus-Sleep State
	NM_STATE_PREPARE_ BUS_SLEEP	0x02	Prepare-Bus State
	NM_STATE_READY_ SLEEP	0x03	Ready Sleep State
	NM_STATE_NORMAL_ OPERATION	0x04	Normal Operation State
	NM_STATE_REPEAT_ MESSAGE	0x05	Repeat Message State
	NM_STATE_ SYNCHRONIZE	0x06	Synchronize State
	NM_STATE_OFFLINE	0x07	Offline State
Description	States of the network management state machine.		
Available via	NmStack_types.h		

⌋

8.2.3 Nm_BusNmType

[SWS_Nm_00276] Definition of datatype Nm_BusNmType ⌈

Name	Nm_BusNmType		
Kind	Enumeration		
Range	NM_BUSNM_CANNM	–	CAN NM type
	NM_BUSNM_FRNM	–	FR NM type
	NM_BUSNM_UDPNM	–	UDP NM type
	NM_BUSNM_GENERICNM	–	Generic NM type
	NM_BUSNM_UNDEF	–	NM type undefined; it shall be defined as FFh
	NM_BUSNM_J1939NM	–	SAE J1939 NM type (address claiming)
	NM_BUSNM_LOCALNM	–	Local NM Type
Description	BusNm Type		
Available via	NmStack_types.h		

⌋

8.2.4 Nm_ConfigType

[SWS_Nm_00282] Definition of datatype Nm_ConfigType

Upstream requirements: SRS_BSW_00414

⌈

Name	Nm_ConfigType
Kind	Structure
Elements	implementation specific

▽

△

	Type	–
	Comment	–
Description	Configuration data structure of the Nm module.	
Available via	Nm.h	

⌋

8.3 Function definitions

This chapter contains all APIs provided by the module.

8.3.1 Standard services provided by NM Interface

8.3.1.1 Nm_Init

[SWS_Nm_00030] Definition of API function Nm_Init

Upstream requirements: SRS_BSW_00101, SRS_BSW_00344, SRS_BSW_00358, SRS_BSW_00405, SRS_BSW_00414

⌈

Service Name	Nm_Init	
Syntax	`void Nm_Init ( const Nm_ConfigType* ConfigPtr )`	
Service ID [hex]	0x00	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	ConfigPtr	Pointer to the selected configuration set.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Initializes the NM Interface.	
Available via	Nm.h	

⌋

Note: Caveats of `Nm_Init`: This service function has to be called after the initialization of the respective bus interface.

Note: The Configuration pointer **ConfigPtr** is currently not used and shall therefore be set NULL_PTR value.

8.3.1.2 Nm_PassiveStartUp

[SWS_Nm_00031] Definition of API function Nm_PassiveStartUp ⌈

Service Name	Nm_PassiveStartUp	
Syntax	`Std_ReturnType Nm_PassiveStartUp ( NetworkHandleType NetworkHandle )`	
Service ID [hex]	0x01	
Sync/Async	Asynchronous	
Reentrancy	Non-reentrant for the same NetworkHandle, reentrant otherwise	
Parameters (in)	NetworkHandle	Identification of the NM-channel
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: No error `E_NOT_OK`: Passive start of network management has failed NetworkHandle does not exist (development only) Module not yet initialized (development only)
Description	This function calls the <Bus>Nm_PassiveStartUp function in case NmBusType is not set to NM_BUSNM_LOCALNM (e.g. CanNm_PassiveStartUp function is called for NM_BUSNM_CANNM).	
Available via	Nm.h	

⌋

Note: Caveats of `Nm_PassiveStartUp`: The **<Bus>Nm** and the **Nm** itself are initialized correctly.

[SWS_Nm_00488]

Upstream requirements: SRS_BSW_00323, SRS_BSW_00369, SRS_BSW_00386

⌈If the pre-processor switch `NmDevErrorDetect` is set to `TRUE`, the function `Nm_PassiveStartUp` shall raise the error `NM_E_INVALID_CHANNEL` if the parameter `NetworkHandle` is not a configured network handle.⌋

8.3.1.3 Nm_NetworkRequest

[SWS_Nm_00032] Definition of API function Nm_NetworkRequest ⌈

Service Name	Nm_NetworkRequest	
Syntax	`Std_ReturnType Nm_NetworkRequest ( NetworkHandleType NetworkHandle )`	
Service ID [hex]	0x02	
Sync/Async	Asynchronous	
Reentrancy	Non-reentrant for the same NetworkHandle, reentrant otherwise	
Parameters (in)	NetworkHandle	Identification of the NM-channel
Parameters (inout)	None	
Parameters (out)	None	

▽

△

Return value	Std_ReturnType	E_OK: No error E_NOT_OK: Requesting of bus communication has failed NetworkHandle does not exist (development only) Module not yet initialized (development only)
Description	This function calls the <Bus>Nm_NetworkRequest (e.g. CanNm_NetworkRequest function is called if channel is configured as CAN) function in case NmBusType is not set to NM_BUSNM_LOCALNM.	
Available via	Nm.h	

⌋

Note: Caveats of Nm_NetworkRequest: The **<Bus>Nm** and the **Nm** itself are initialized correctly.

[SWS_Nm_00130]

Upstream requirements: SRS_BSW_00323, SRS_BSW_00369, SRS_BSW_00386

⌈If Nm_NetworkRequest is called with a network handle where NmPassiveModeEnabled is set to TRUE it shall not execute any functionality and return with **E_NOT_OK**. If NmDevErrorDetect is set to TRUE then it shall raise the error NM_E_INVALID_CHANNEL in this case.⌋

[SWS_Nm_00489]

Upstream requirements: SRS_BSW_00323, SRS_BSW_00369, SRS_BSW_00386

⌈If the pre-processor switch NmDevErrorDetect is set to TRUE, the function Nm_NetworkRequest shall raise the error NM_E_INVALID_CHANNEL if the parameter NetworkHandle is not a configured network handle.⌋

8.3.1.4 Nm_NetworkRelease

[SWS_Nm_00046] Definition of API function Nm_NetworkRelease ⌈

Service Name	Nm_NetworkRelease	
Syntax	`Std_ReturnType Nm_NetworkRelease ( NetworkHandleType NetworkHandle )`	
Service ID [hex]	0x03	
Sync/Async	Asynchronous	
Reentrancy	Non-reentrant for the same NetworkHandle, reentrant otherwise	
Parameters (in)	NetworkHandle	Identification of the NM-channel
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: No error E_NOT_OK: Releasing of bus communication has failed NetworkHandle does not exist (development only) Module not yet initialized (development only)

▽

△

Description	This function calls the <Bus>Nm_NetworkRelease bus specific function in case NmBusType is not set to NM_BUSNM_LOCALNM (e.g. CanNm_NetworkRelease function is called if channel is configured as CAN).
Available via	Nm.h

⌋

Note: Caveats of `Nm_NetworkRelease`: The **<Bus>Nm** and the **Nm** itself are initialized correctly.

[SWS_Nm_00132]

Upstream requirements: SRS_BSW_00323, SRS_BSW_00369, SRS_BSW_00386

⌈If `Nm_NetworkRelease` is called with a network handle where`NmPassiveModeEnabled`is set to `TRUE` it shall not execute any functionality and return with **E_NOT_OK**. If `NmDevErrorDetect` is set to `TRUE` then it shall raise the error `NM_E_INVALID_CHANNEL` in this case.⌋

[SWS_Nm_00490]

Upstream requirements: SRS_BSW_00323, SRS_BSW_00369, SRS_BSW_00386

⌈If the pre-processor switch `NmDevErrorDetect` is set to `TRUE`, the function `Nm_NetworkRelease` shall raise the error `NM_E_INVALID_CHANNEL` if the parameter `NetworkHandle` is not a configured network handle.⌋

8.3.2 Communication control services provided by NM Interface

The following services are provided by NM Interface to allow the Diagnostic Communication Manager (**DCM**) to control the transmission of NM Messages.

Note: To run the `coordination algorithm` correctly, it has to be ensured that NM PDU transmission ability is enabled before the ECU is shut down. If `<Bus>Nm_NetworkRelease` is called while NM PDU transmission ability is disabled, the ECU will shut down after NM PDU transmission ability has been re-enabled again. Therefore the ECU can also shut down in case of race conditions (e.g. diagnostic session left shortly before enabling communication) or a wrong usage of communication control.

8.3.2.1 Nm_DisableCommunication

[SWS_Nm_00033] Definition of API function Nm_DisableCommunication ⌈

Service Name	Nm_DisableCommunication	
Syntax	`Std_ReturnType Nm_DisableCommunication ( NetworkHandleType NetworkHandle )`	
Service ID [hex]	0x04	
Sync/Async	Asynchronous	
Reentrancy	Non-reentrant for the same NetworkHandle, reentrant otherwise	
Parameters (in)	NetworkHandle	Identification of the NM-channel
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: No error E_NOT_OK: Disabling of NM PDU transmission ability has failed. NetworkHandle does not exist (development only) Module not yet initialized (development only)
Description	Disables the NM PDU transmission ability. For that purpose <Bus>Nm_DisableCommunication shall be called in case NmBusType is not set to NM_BUSNM_LOCALNM (e.g. CanNm_Disable Communication function is called if channel is configured as CAN).	
Available via	Nm.h	

⌋

Note: Caveats of Nm_DisableCommunication: The **<Bus>Nm** and the **Nm** itself are initialized correctly.

[SWS_Nm_00134] ⌈Configuration of Nm_DisableCommunication: This function is only available if NmComControlEnabled is set to TRUE.⌋

[SWS_Nm_00286]

Upstream requirements: SRS_BSW_00323, SRS_BSW_00369, SRS_BSW_00386

⌈If Nm_DisableCommunication is called with a network handle whereNmPassive-ModeEnabledis set to TRUE it shall not execute any functionality and return with **E_NOT_OK**. If NmDevErrorDetect is set to TRUE then it shall raise the error NM_E_INVALID_CHANNEL in this case.⌋

[SWS_Nm_00491]

Upstream requirements: SRS_BSW_00323, SRS_BSW_00369, SRS_BSW_00386

⌈If the pre-processor switch NmDevErrorDetect is set to TRUE, the function Nm_DisableCommunication shall raise the error NM_E_INVALID_CHANNEL if the parameter NetworkHandle is not a configured network handle.⌋

8.3.2.2 Nm_EnableCommunication

[SWS_Nm_00034] Definition of API function Nm_EnableCommunication ⌈

Service Name	Nm_EnableCommunication	
Syntax	`Std_ReturnType Nm_EnableCommunication ( NetworkHandleType NetworkHandle )`	
Service ID [hex]	0x05	
Sync/Async	Asynchronous	
Reentrancy	Non-reentrant for the same NetworkHandle, reentrant otherwise	
Parameters (in)	NetworkHandle	Identification of the NM-channel
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: No error E_NOT_OK: Enabling of NM PDU transmission ability has failed. NetworkHandle does not exist (development only) Module not yet initialized (development only)
Description	Enables the NM PDU transmission ability. For that purpose <Bus>Nm_EnableCommunication shall be called in case NmBusType is not set to NM_BUSNM_LOCALNM. (e.g. CanNm_Enable Communication function is called if channel is configured as CAN).	
Available via	Nm.h	

⌋

Note: Caveats of Nm_EnableCommunication: The **<Bus>Nm** and the **Nm** itself are initialized correctly.

[SWS_Nm_00136] ⌈Configuration of Nm_EnableCommunication: This function is only available if NmComControlEnabled is set to TRUE.⌋

[SWS_Nm_00287]

Upstream requirements: SRS_BSW_00323, SRS_BSW_00369, SRS_BSW_00386

⌈If Nm_EnableCommunication is called with a network handle whereNmPassive-ModeEnabledis set to TRUE it shall not execute any functionality and return with **E_NOT_OK**. If NmDevErrorDetect is set to TRUE then it shall raise the error NM_E_INVALID_CHANNEL in this case.⌋

[SWS_Nm_00492]

Upstream requirements: SRS_BSW_00323, SRS_BSW_00369, SRS_BSW_00386

⌈If the pre-processor switch NmDevErrorDetect is set to TRUE, the function Nm_EnableCommunication shall raise the error NM_E_INVALID_CHANNEL if the parameter NetworkHandle is not a configured network handle.⌋

8.3.3 Partial Network services provided by NM Interface

8.3.3.1 Nm_RequestSynchronizedPncShutdown

[SWS_Nm_91005] Definition of API function Nm_RequestSynchronizedPncShutdown ⌈

Service Name	Nm_RequestSynchronizedPncShutdown	
Syntax	`Std_ReturnType Nm_RequestSynchronizedPncShutdown ( NetworkHandleType NetworkHandle, PNCHandleType PncId )`	
Service ID [hex]	0x24	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant for the same NetworkHandle, reentrant otherwise	
Parameters (in)	NetworkHandle	Identification of the NM-channel
	PncId	Identification of the Pnc which is requested for a synchronized shutdown across the PNC network topology
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: No error `E_NOT_OK`: Request for a synchronized PNC shutdown has failed, e.g. NetworkHandle does not exist (development only) Module not yet initialized (development only)
Description	This function store the request for a synchronized PNC shutdown of a particular PNC given by PncId per given NM-Channel. The handling of the synchronized PNC shutdown process is mainly done in the context of the Nm_Mainfunction. The function call is only valid if NmStandardBusType is not set to NM_BUSNM_LOCALNM as a <Bus>Nm like CanNm is needed to transmit the PNC shutdown requests.	
Available via	Nm.h	

⌋

[SWS_Nm_00508]

Upstream requirements: SRS_BSW_00323, SRS_BSW_00369, SRS_BSW_00386

⌈If the pre-processor switch `NmDevErrorDetect` is set to `TRUE`, the function `Nm_RequestSynchronizedPncShutdown` shall raise the error `NM_E_INVALID_CHANNEL` if the parameter `NetworkHandle` is not a configured network handle.⌋

8.3.3.2 Nm_UpdateIRA

[SWS_Nm_91007] Definition of callback function Nm_UpdateIRA ⌈

Service Name	Nm_UpdateIRA
Syntax	`void Nm_UpdateIRA ( NetworkHandleType NetworkHandle, const uint8* PncRequestBitVectorPtr )`
Service ID [hex]	0x26

▽

△

Sync/Async	Synchronous	
Reentrancy	Non Reentrant for the same NetworkHandle, reentrant otherwise	
Parameters (in)	NetworkHandle	Identification of the NM-channel
	PncRequestBitVectorPtr	Pointer to the bit vector with all PNC Request Bits set to “1” of internal requested PNCs (IRA)
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Indication by ComM of internal PNC requests. This is used to aggregate the internal PNC requests.	
Available via	Nm.h	

⌋

8.3.4 Extra services provided by NM Interface

The following services are provided by NM Interface for OEM specific extensions of the NM stack and are not required by any AUTOSAR module.

8.3.4.1 Nm_SetUserData

[SWS_Nm_00035] Definition of API function Nm_SetUserData ⌈

Service Name	Nm_SetUserData	
Syntax	`Std_ReturnType Nm_SetUserData ( NetworkHandleType NetworkHandle, const uint8* nmUserDataPtr )`	
Service ID [hex]	0x06	
Sync/Async	Synchronous	
Reentrancy	Non-reentrant for the same NetworkHandle, reentrant otherwise	
Parameters (in)	NetworkHandle	Identification of the NM-channel
	nmUserDataPtr	User data for the next transmitted NM message
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: No error `E_NOT_OK`: Setting of user data has failed NetworkHandle does not exist (development only) Module not yet initialized (development only)
Description	Set user data for NM messages transmitted next on the bus. For that purpose <Bus>Nm_Set UserData shall be called in case NmBusType is not set to NM_BUSNM_LOCALNM. (e.g. Can Nm_SetUserData function is called if channel is configured as CAN).	
Available via	Nm.h	

⌋

Note: Caveats of `Nm_SetUserData`: The **<Bus>Nm** and the **Nm** itself are initialized correctly.

[SWS_Nm_00138] ⌈Configuration of Nm_SetUserData: This function is only available if NmUserDataEnabled is set to TRUE.⌋

[SWS_Nm_00288]

Upstream requirements: SRS_BSW_00323, SRS_BSW_00369, SRS_BSW_00386

⌈If Nm_SetUserData is called with a network handle whereNmPassiveModeEnabledis set to TRUE it shall not execute any functionality and return with **E_NOT_OK**. If NmDevErrorDetect is set to TRUE then it shall raise the error NM_E_INVALID_CHANNEL in this case.⌋

[SWS_Nm_00241] ⌈Configuration of Nm_SetUserData: If NmComUserDataSupport is TRUE the API Nm_SetUserData shall not be available.⌋

[SWS_Nm_00493]

Upstream requirements: SRS_BSW_00323, SRS_BSW_00369, SRS_BSW_00386

⌈If the pre-processor switch NmDevErrorDetect is set to TRUE, the function Nm_SetUserData shall raise the error NM_E_INVALID_CHANNEL if the parameter NetworkHandle is not a configured network handle.⌋

8.3.4.2 Nm_GetUserData

[SWS_Nm_00036] Definition of API function Nm_GetUserData ⌈

Service Name	Nm_GetUserData	
Syntax	`Std_ReturnType Nm_GetUserData ( NetworkHandleType NetworkHandle, uint8* nmUserDataPtr )`	
Service ID [hex]	0x07	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	NetworkHandle	Identification of the NM-channel
Parameters (inout)	None	
Parameters (out)	nmUserDataPtr	Pointer where user data out of the last successfully received NM message shall be copied to
Return value	Std_ReturnType	E_OK: No error E_NOT_OK: Getting of user data has failed NetworkHandle does not exist (development only) Module not yet initialized (development only)
Description	Get user data out of the last successfully received NM message. For that purpose <Bus>Nm_GetUserData shall be called in case NmBusType is not set to NM_BUSNM_LOCALNM. (e.g. CanNm_GetUserData function is called if channel is configured as CAN).	
Available via	Nm.h	

⌋

Note: Caveats of Nm_GetUserData: The **<Bus>Nm** and the **Nm** itself are initialized correctly.

[SWS_Nm_00140] ⌈Configuration of Nm_GetUserData: This function is only available if NmUserDataEnabled is set to TRUE.⌋

[SWS_Nm_00494]

Upstream requirements: SRS_BSW_00323, SRS_BSW_00369, SRS_BSW_00386

⌈If the pre-processor switch NmDevErrorDetect is set to TRUE, the function Nm_GetUserData shall raise the error NM_E_INVALID_CHANNEL if the parameter NetworkHandle is not a configured network handle.⌋

8.3.4.3 Nm_GetPduData

[SWS_Nm_00037] Definition of API function Nm_GetPduData

Upstream requirements: SRS_BSW_00310, SRS_BSW_00460, SRS_BSW_00461, SRS_BSW_00484, SRS_BSW_00459

⌈

Service Name	Nm_GetPduData	
Syntax	`Std_ReturnType Nm_GetPduData ( NetworkHandleType NetworkHandle, uint8* nmPduData )`	
Service ID [hex]	0x08	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	NetworkHandle	Identification of the NM-channel
Parameters (inout)	None	
Parameters (out)	nmPduData	Pointer where NM PDU shall be copied to.
Return value	Std_ReturnType	E_OK: No error E_NOT_OK: Getting of NM PDU data has failed NetworkHandle does not exist (development only) Module not yet initialized (development only)
Description	Get the whole PDU data out of the most recently received NM message. For that purpose <Bus Nm>_GetPduData shall be called in case NmBusType is not set to NM_BUSNM_LOCALNM. (e.g. CanNm_GetPduData function is called if channel is configured as CAN).	
Available via	Nm.h	

⌋

Note: Caveats of Nm_GetPduData: The **<Bus>Nm** and the **Nm** itself are initialized correctly.

[SWS_Nm_00495]

Upstream requirements: SRS_BSW_00323, SRS_BSW_00369, SRS_BSW_00386

⌈If the pre-processor switch NmDevErrorDetect is set to TRUE, the function Nm_GetPduData shall raise the error NM_E_INVALID_CHANNEL if the parameter NetworkHandle is not a configured network handle.⌋

8.3.4.4 Nm_RepeatMessageRequest

[SWS_Nm_00038] Definition of API function Nm_RepeatMessageRequest ⌈

Service Name	Nm_RepeatMessageRequest	
Syntax	`Std_ReturnType Nm_RepeatMessageRequest ( NetworkHandleType NetworkHandle )`	
Service ID [hex]	0x09	
Sync/Async	Asynchronous	
Reentrancy	Non-reentrant for the same NetworkHandle, reentrant otherwise	
Parameters (in)	NetworkHandle	Identification of the NM-channel
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: No error `E_NOT_OK`: Setting of Repeat Message Request Bit has failed NetworkHandle does not exist (development only) Module not yet initialized (development only)
Description	Set Repeat Message Request Bit for NM messages transmitted next on the bus. For that purpose <Bus>Nm_RepeatMessageRequest shall be called in case NmBusType is not set to NM_BUSNM_LOCALNM. (e.g. CanNm_RepeatMessageRequest function is called if channel is configured as CAN). This will force all nodes on the bus to transmit NM messages so that they can be identified.	
Available via	Nm.h	

⌋

Note: Caveats of `Nm_RepeatMessageRequest`: The **<Bus>Nm** and the **Nm** itself are initialized correctly.

[SWS_Nm_00289]

Upstream requirements: SRS_BSW_00323, SRS_BSW_00369, SRS_BSW_00386

⌈If `Nm_RepeatMessageRequest` is called with a network handle where`NmPassiveModeEnabled`is set to `TRUE` it shall not execute any functionality and return with **E_NOT_OK**. If `NmDevErrorDetect` is set to `TRUE` then it shall raise the error `NM_E_INVALID_CHANNEL` in this case.⌋

[SWS_Nm_00496]

Upstream requirements: SRS_BSW_00323, SRS_BSW_00369, SRS_BSW_00386

⌈If the pre-processor switch `NmDevErrorDetect` is set to `TRUE`, the function `Nm_RepeatMessageRequest` shall raise the error `NM_E_INVALID_CHANNEL` if the parameter `NetworkHandle` is not a configured network handle.⌋

8.3.4.5 Nm_GetNodeIdentifier

[SWS_Nm_00039] Definition of API function Nm_GetNodeIdentifier ⌈

Service Name	Nm_GetNodeIdentifier	
Syntax	`Std_ReturnType Nm_GetNodeIdentifier ( NetworkHandleType NetworkHandle, uint8* nmNodeIdPtr )`	
Service ID [hex]	0x0a	
Sync/Async	Synchronous	
Reentrancy	Non-reentrant for the same NetworkHandle, reentrant otherwise	
Parameters (in)	NetworkHandle	Identification of the NM-channel
Parameters (inout)	None	
Parameters (out)	nmNodeIdPtr	Pointer where node identifier out of the last successfully received NM-message shall be copied to
Return value	Std_ReturnType	`E_OK`: No error `E_NOT_OK`: Getting of the node identifier out of the last received NM-message has failed NetworkHandle does not exist (development only) Module not yet initialized (development only)
Description	Get node identifier out of the last successfully received NM-message. The function <Bus>Nm_GetNodeIdentifier shall be called in case NmBusType is not set to NM_BUSNM_LOCALNM. (e.g. CanNm_GetNodeIdentifier function is called if channel is configured as CAN).	
Available via	Nm.h	

⌋

Note: Caveats of `Nm_GetNodeIdentifier`: The **<Bus>Nm** and the **Nm** itself are initialized correctly.

[SWS_Nm_00497]

Upstream requirements: SRS_BSW_00323, SRS_BSW_00369, SRS_BSW_00386

⌈If the pre-processor switch `NmDevErrorDetect` is set to `TRUE`, the function `Nm_GetNodeIdentifier` shall raise the error `NM_E_INVALID_CHANNEL` if the parameter `NetworkHandle` is not a configured network handle.⌋

8.3.4.6 Nm_GetLocalNodeIdentifier

[SWS_Nm_00040] Definition of API function Nm_GetLocalNodeIdentifier ⌈

Service Name	Nm_GetLocalNodeIdentifier
Syntax	`Std_ReturnType Nm_GetLocalNodeIdentifier ( NetworkHandleType NetworkHandle, uint8* nmNodeIdPtr )`
Service ID [hex]	0x0b
Sync/Async	Synchronous
Reentrancy	Non-reentrant for the same NetworkHandle, reentrant otherwise

▽

△

Parameters (in)	NetworkHandle	Identification of the NM-channel
Parameters (inout)	None	
Parameters (out)	nmNodeIdPtr	Pointer where node identifier of the local node shall be copied to
Return value	Std_ReturnType	E_OK: No error E_NOT_OK: Getting of the node identifier of the local node has failed NetworkHandle does not exist (development only) Module not yet initialized (development only)
Description	Get node identifier configured for the local node. For that purpose <Bus>Nm_GetLocalNode Identifier shall be called in case NmBusType is not set to NM_BUSNM_LOCALNM. (e.g. Can Nm_GetLocalNodeIdentifier function is called if channel is configured as CAN).	
Available via	Nm.h	

⌋

Note: Caveats of Nm_GetLocalNodeIdentifier: The **<Bus>Nm** and the **Nm** itself are initialized correctly.

[SWS_Nm_00498]

Upstream requirements: SRS_BSW_00323, SRS_BSW_00369, SRS_BSW_00386

⌈If the pre-processor switch NmDevErrorDetect is set to TRUE, the function Nm_GetLocalNodeIdentifier shall raise the error NM_E_INVALID_CHANNEL if the parameter NetworkHandle is not a configured network handle.⌋

8.3.4.7 Nm_CheckRemoteSleepIndication

[SWS_Nm_00042] Definition of API function Nm_CheckRemoteSleepIndication ⌈

Service Name	Nm_CheckRemoteSleepIndication	
Syntax	`Std_ReturnType Nm_CheckRemoteSleepIndication ( NetworkHandleType nmNetworkHandle, boolean* nmRemoteSleepIndPtr )`	
Service ID [hex]	0x0d	
Sync/Async	Synchronous	
Reentrancy	Non-reentrant for the same NetworkHandle, reentrant otherwise	
Parameters (in)	nmNetworkHandle	Identification of the NM-channel
Parameters (inout)	None	
Parameters (out)	nmRemoteSleepIndPtr	Pointer where check result of remote sleep indication shall be copied to
Return value	Std_ReturnType	E_OK: No error E_NOT_OK: Checking of remote sleep indication bits has failed NetworkHandle does not exist (development only) Module not yet initialized (development only)
Description	Check if remote sleep indication takes place or not. For that purpose <Bus>Nm_CheckRemote SleepIndication shall be called in case NmBusType is not set to NM_BUSNM_LOCALNM. (e.g. CanNm_CheckRemoteSleepIndication function is called if channel is configured as CAN).	
Available via	Nm.h	

⌋

Note: Caveats of Nm_CheckRemoteSleepIndication: The **<Bus>Nm** and the **Nm** itself are initialized correctly.

[SWS_Nm_00290] ⌈If Nm_CheckRemoteSleepIndication is called with a network handle whereNmPassiveModeEnabledis set to TRUE it shall not execute any functionality and return with **E_NOT_OK**. If NmDevErrorDetect is set to TRUE then it shall raise the error NM_E_INVALID_CHANNEL in this case.⌋

[SWS_Nm_00150] ⌈Configuration of Nm_CheckRemoteSleepIndication: This function is only available if NmRemoteSleepIndEnabled is set to TRUE.⌋

[SWS_Nm_00499]

Upstream requirements: SRS_BSW_00323, SRS_BSW_00369, SRS_BSW_00386

⌈If the pre-processor switch NmDevErrorDetect is set to TRUE, the function Nm_CheckRemoteSleepIndication shall raise the error NM_E_INVALID_CHANNEL if the parameter nmNetworkHandle is not a configured network handle.⌋

8.3.4.8 Nm_GetState

[SWS_Nm_00043] Definition of API function Nm_GetState ⌈

Service Name	Nm_GetState	

Syntax	`Std_ReturnType Nm_GetState (`
	`  NetworkHandleType nmNetworkHandle,`
	`  Nm_StateType* nmStatePtr,`
	`  Nm_ModeType* nmModePtr`
	`)`	

Service ID [hex]	0x0e	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	nmNetworkHandle	Identification of the NM-channel
Parameters (inout)	None	
Parameters (out)	nmStatePtr	Pointer where state of the network management shall be copied to
	nmModePtr	Pointer to the location where the mode of the network management shall be copied to
Return value	Std_ReturnType	E_OK: No error E_NOT_OK: Getting of NM state has failed NetworkHandle does not exist (development only) Module not yet initialized (development only)
Description	Returns the state of the network management. The function <Bus>Nm_GetState shall be called in case NmBusType is not set to NM_BUSNM_LOCALNM. (e.g. CanNm_GetState function is called if channel is configured as CAN).	
Available via	Nm.h	

⌋

Note: Caveats of Nm_GetState: The **<Bus>Nm** and the **Nm** itself are initialized correctly.

[SWS_Nm_00500]

Upstream requirements: SRS_BSW_00323, SRS_BSW_00369, SRS_BSW_00386

⌈If the pre-processor switch NmDevErrorDetect is set to TRUE, the function Nm_GetState shall raise the error NM_E_INVALID_CHANNEL if the parameter nmNetworkHandle is not a configured network handle.⌋

8.3.4.9 Nm_PnLearningRequest

[SWS_Nm_91003] Definition of API function Nm_PnLearningRequest

Upstream requirements: SRS_BSW_00310, SRS_BSW_00460, SRS_BSW_00461, SRS_BSW_00484, SRS_BSW_00459

⌈

Service Name	Nm_PnLearningRequest	
Syntax	`Std_ReturnType Nm_PnLearningRequest ( NetworkHandleType NetworkHandle )`	
Service ID [hex]	0x22	
Sync/Async	Asynchronous	
Reentrancy	Non Reentrant for the same NetworkHandle, reentrant otherwise	
Parameters (in)	NetworkHandle	Identification of the NM-channel
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: No error E_NOT_OK: PN Learning Request has failed or is not configured for the networkHandle
Description	Set Repeat Message Request Bit and Partial Network Learning Bit for NM messages transmitted next on the bus. For that purpose <Bus>Nm_PnLearningRequest shall be called (e.g. CanNm_PnLearningRequest function if channel is configured as CAN). This will force all nodes to enter the PNC Learning Phase and re-enter Repeat Message Stat. This is needed for the optional Dynamic PNC-to-channel-mapping feature.	
Available via	Nm.h	

⌋

Note: Caveats of Nm_PnLearningRequest: The <Bus>Nm and the Nm itself are initialized correctly.

[SWS_Nm_00502] ⌈Nm_PnLearningRequest shall only be available if NmDynamicPncToChannelMappingSupport is set to TRUE.⌋

[SWS_Nm_00503]

Upstream requirements: SRS_BSW_00323, SRS_BSW_00369, SRS_BSW_00386

⌈If Nm_PnLearningRequest is called with a network handle for a bus system where NmDynamicPncToChannelMappingEnabled is set to FALSE Nm shall not execute any functionality and return with E_NOT_OK. If NmDevErrorDetect is set to TRUE then it shall raise the error NM_E_INVALID_CHANNEL in this case.⌋

[SWS_Nm_00505]

Upstream requirements: SRS_BSW_00323, SRS_BSW_00369, SRS_BSW_00386

⌈If the pre-processor switch `NmDevErrorDetect` is set to `TRUE`, the function `Nm_PnLearningRequest` shall raise the error `NM_E_INVALID_CHANNEL` if the parameter `NetworkHandle` is not a configured network handle.⌋

8.3.5 Standard Call-back notifications

Callback notifications are called by the lower layer's bus-specific Network Management modules. For the Base functionality of Nm (Chapter 7.1) the call-backs shall be forwarded to the upper layer's ComM. For the NM Coordinator functionality of Nm (Chapter 7.2) the call-backs will provide indications used to control the NM Coordinator and the optional Dynamic PNC-to-channel-mapping feature.

[SWS_Nm_00028]

Upstream requirements: SRS_BSW_00333

⌈All callbacks of the Nm shall assume that they can run either in task or in interrupt context.⌋

8.3.5.1 Nm_NetworkStartIndication

[SWS_Nm_00154] Definition of callback function Nm_NetworkStartIndication

Upstream requirements: SRS_BSW_00359

⌈

Service Name	Nm_NetworkStartIndication	
Syntax	`void Nm_NetworkStartIndication ( NetworkHandleType nmNetworkHandle )`	
Service ID [hex]	0x11	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Parameters (in)	nmNetworkHandle	Identification of the NM-channel
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Notification that a NM-message has been received in the Bus-Sleep Mode, what indicates that some nodes in the network have already entered the Network Mode.	
Available via	Nm.h	

⌋

[SWS_Nm_00155] ⌈The indication through callback function `Nm_NetworkStartIndication`: shall be forwarded to **ComM** by calling the `ComM_Nm_NetworkStartIndication`.⌋

8.3.5.2 Nm_NetworkMode

[SWS_Nm_00156] Definition of callback function Nm_NetworkMode

Upstream requirements: SRS_BSW_00359

⌈

Service Name	Nm_NetworkMode	
Syntax	`void Nm_NetworkMode ( NetworkHandleType nmNetworkHandle )`	
Service ID [hex]	0x12	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Parameters (in)	nmNetworkHandle	Identification of the NM-channel
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Notification that the network management has entered Network Mode.	
Available via	Nm.h	

⌋

[SWS_Nm_00158] ⌈The indication through callback function `Nm_NetworkMode`: shall be forwarded to **ComM** by calling the `ComM_Nm_NetworkMode`.⌋

8.3.5.3 Nm_BusSleepMode

[SWS_Nm_00162] Definition of callback function Nm_BusSleepMode

Upstream requirements: SRS_BSW_00359

⌈

Service Name	Nm_BusSleepMode	
Syntax	`void Nm_BusSleepMode ( NetworkHandleType nmNetworkHandle )`	
Service ID [hex]	0x14	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Parameters (in)	nmNetworkHandle	Identification of the NM-channel
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Notification that the network management has entered Bus-Sleep Mode.	
Available via	Nm.h	

⌋

[SWS_Nm_00163] ⌈The indication through callback function `Nm_BusSleepMode`: shall be forwarded to **ComM** by calling the `ComM_Nm_BusSleepMode`.⌋

8.3.5.4 Nm_PrepareBusSleepMode

[SWS_Nm_00159] Definition of callback function Nm_PrepareBusSleepMode

Upstream requirements: SRS_BSW_00359

⌈

Service Name	Nm_PrepareBusSleepMode	
Syntax	`void Nm_PrepareBusSleepMode ( NetworkHandleType nmNetworkHandle )`	
Service ID [hex]	0x13	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Parameters (in)	nmNetworkHandle	Identification of the NM-channel
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Notification that the network management has entered Prepare Bus-Sleep Mode.	
Available via	Nm.h	

⌋

[SWS_Nm_00161] ⌈The indication through callback function `Nm_PrepareBusSleepMode`: shall be forwarded to **ComM** by calling `ComM_Nm_PrepareBusSleepMode`.⌋

8.3.5.5 Nm_SynchronizeMode

[SWS_Nm_91002] Definition of callback function Nm_SynchronizeMode ⌈

Service Name	Nm_SynchronizeMode	
Syntax	`void Nm_SynchronizeMode ( NetworkHandleType nmNetworkHandle )`	
Service ID [hex]	0x21	
Sync/Async	Asynchronous	
Reentrancy	Reentrant but not for the same channel	
Parameters (in)	nmNetworkHandle	Identification of the NM-channel
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Notification that the network management has entered Synchronize Mode.	
Available via	Nm.h	

⌋

8.3.5.6 Nm_RemoteSleepIndication

[SWS_Nm_00192] Definition of callback function Nm_RemoteSleepIndication

Upstream requirements: SRS_BSW_00359

⌈

Service Name	Nm_RemoteSleepIndication	
Syntax	`void Nm_RemoteSleepIndication ( NetworkHandleType nmNetworkHandle )`	
Service ID [hex]	0x17	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Parameters (in)	nmNetworkHandle	Identification of the NM-channel
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Notification that the network management has detected that all other nodes on the network are ready to enter Bus-Sleep Mode.	
Available via	Nm.h	

⌋

[SWS_Nm_00277] ⌈Configuration of `Nm_RemoteSleepIndication`: This function is only available if `NmRemoteSleepIndEnabled` is set to `TRUE`.⌋

The notification that all other nodes on the network are ready to enter Bus-Sleep Mode is only needed for internal purposes of the NM Coordinator.

Note: When *NM Coordinator functionality* is disabled `Nm_RemoteSleepIndication` can be an empty function.

8.3.5.7 Nm_RemoteSleepCancellation

[SWS_Nm_00193] Definition of callback function Nm_RemoteSleepCancellation

Upstream requirements: SRS_BSW_00359

⌈

Service Name	Nm_RemoteSleepCancellation	
Syntax	`void Nm_RemoteSleepCancellation ( NetworkHandleType nmNetworkHandle )`	
Service ID [hex]	0x18	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Parameters (in)	nmNetworkHandle	Identification of the NM-channel
Parameters (inout)	None	

▽

△

Parameters (out)	None
Return value	None
Description	Notification that the network management has detected that not all other nodes on the network are longer ready to enter Bus-Sleep Mode.
Available via	Nm.h

⌋

[SWS_Nm_00278] ⌈Configuration of `Nm_RemoteSleepCancellation`: This function is only available if `NmRemoteSleepIndEnabled` is set to `TRUE`.⌋

The notification that not all other nodes on the network are longer ready to enter Bus-Sleep Mode is only needed for internal purposes of the NM Coordinator.

Note: When *NM Coordinator functionality* is disabled `Nm_RemoteSleepCancellation` can be an empty function.

8.3.5.8 Nm_SynchronizationPoint

[SWS_Nm_00194] Definition of callback function Nm_SynchronizationPoint

Upstream requirements: SRS_BSW_00359

⌈

Service Name	Nm_SynchronizationPoint	
Syntax	`void Nm_SynchronizationPoint ( NetworkHandleType nmNetworkHandle )`	
Service ID [hex]	0x19	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Parameters (in)	nmNetworkHandle	Identification of the NM-channel
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Notification to the NM Coordinator functionality that this is a suitable point in time to initiate the coordinated shutdown on.	
Available via	Nm.h	

⌋

The notification that this is a suitable point in time to initiate the `coordinated shutdown` is only needed for internal purposes of the NM Coordinator.

8.3.5.9 Nm_CoordReadyToSleepIndication

[SWS_Nm_00254] Definition of callback function Nm_CoordReadyToSleepIndication

Upstream requirements: SRS_BSW_00359

⌈

Service Name	Nm_CoordReadyToSleepIndication	
Syntax	`void Nm_CoordReadyToSleepIndication ( NetworkHandleType nmChannelHandle )`	
Service ID [hex]	0x1e	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	nmChannelHandle	Identification of the NM-channel
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Sets an indication, when the NM Coordinator Sleep Ready bit in the Control Bit Vector is set	
Available via	Nm.h	

⌋

[SWS_Nm_00255] ⌈Configuration of `Nm_CoordReadyToSleepIndication`: Optional
If `NmCoordinatorSyncSupport` is set to `TRUE` , the Nm shall provide the API `Nm_CoordReadyToSleepIndication`.⌋

8.3.5.10 Nm_CoordReadyToSleepCancellation

[SWS_Nm_00272] Definition of callback function Nm_CoordReadyToSleepCancellation

Upstream requirements: SRS_BSW_00359

⌈

Service Name	Nm_CoordReadyToSleepCancellation	
Syntax	`void Nm_CoordReadyToSleepCancellation ( NetworkHandleType nmChannelHandle )`	
Service ID [hex]	0x1f	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	nmChannelHandle	Identification of the NM-channel
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	

▽

△

Description	Cancels an indication, when the NM Coordinator Sleep Ready bit in the Control Bit Vector is set back to 0.
Available via	Nm.h

⌋

[SWS_Nm_00273] ⌈Configuration of `Nm_CoordReadyToSleepCancellation`: Optional
If `NmCoordinatorSyncSupport` is set to `TRUE` , the Nm shall provide the API `Nm_CoordReadyToSleepCancellation`.⌋

8.3.5.11 Nm_ForwardSynchronizedPncShutdown

[SWS_Nm_91009] Definition of API function Nm_ForwardSynchronizedPncShutdown ⌈

Service Name	Nm_ForwardSynchronizedPncShutdown	
Syntax	`void Nm_ForwardSynchronizedPncShutdown ( NetworkHandleType NetworkHandle, const uint8* PncRequestBitVectorPtr )`	
Service ID [hex]	0x28	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant for the same NetworkHandle, reentrant otherwise	
Parameters (in)	NetworkHandle	Identification of the NM-channel
	PncRequestBitVectorPtr	Pointer to the bit vector with all PNC Request Bits set to “1” which are indicated for a synchronized PNC shutdown
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Notification that the network management has received a PN shutdown message on a particular NM-channel. This is used to grant a nearly synchronized PNC shutdown across the entire PN topology.	
Available via	Nm.h	

⌋

8.3.5.12 Nm_PncBitVectorRxIndication

[SWS_Nm_91006] Definition of callback function Nm_PncBitVectorRxIndication

Upstream requirements: SRS_ModeMgm_09250

⌈

Service Name	Nm_PncBitVectorRxIndication	
Syntax	`void Nm_PncBitVectorRxIndication ( NetworkHandleType NetworkHandle, const uint8* PncRequestBitVectorPtr, boolean* RelevantPncRequestDetectedPtr )`	
Service ID [hex]	0x25	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant for the same NetworkHandle, reentrant otherwise	
Parameters (in)	NetworkHandle	Identification of the NM-channel
	PncRequestBitVectorPtr	Pointer to the bit vector with all PNC Request Bits set to “1” of external requested PNCs
Parameters (inout)	None	
Parameters (out)	RelevantPncRequest DetectedPtr	Pointer to a boolean variable which indicates, if a relevant PNC request is available in the given PncRequestBitVector
Return value	None	
Description	Indication that a bus specific network management has received a NM message on a particular NM-channel that contain a PNC Request Bit Vector. This is used to aggregate the external PNC requests. The function evaluate if a relevant PNC request (PNC Request Bit set to ’1’) is available in the given PNC Request Bit Vector. If a relevant PNC request is available (PNC Request Bit passes the PNC Request Bit Vector filter), then the RelevantPncRequestDetected Ptr refers to a boolean with value set to TRUE. Otherwise refer to boolean with value set to FALSE. RelevantPncRequestDetectedPtr is evaluated by the callee <Bus>Nm module to qualify the further processing of the received NM-PDU.	
Available via	Nm.h	

⌋

8.3.5.13 Nm_PncBitVectorTxIndication

[SWS_Nm_91008] Definition of callback function Nm_PncBitVectorTxIndication

Upstream requirements: SRS_ModeMgm_09250

⌈

Service Name	Nm_PncBitVectorTxIndication	
Syntax	`void Nm_PncBitVectorTxIndication ( NetworkHandleType NetworkHandle, uint8* PncRequestBitVectorPtr )`	
Service ID [hex]	0x27	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant for the same NetworkHandle, reentrant otherwise	
Parameters (in)	NetworkHandle	Identification of the NM-channel
Parameters (inout)	None	

▽

△

Parameters (out)	PncRequestBitVectorPtr	Pointer to the bit vector with all PNC Request Bits set to “1” of internal requested PNCs
Return value	None	
Description	Function called by <Bus>Nms to request the aggregated internal PNC requests for transmission within the Nm message.	
Available via	Nm.h	

⌋

8.3.5.14 Nm_PncBitVectorTxConfirmation

[SWS_Nm_91010] Definition of callback function Nm_PncBitVectorTxConfirmation

Upstream requirements: SRS_ModeMgm_09250

⌈

Service Name	Nm_PncBitVectorTxConfirmation	
Syntax	`void Nm_PncBitVectorTxConfirmation ( NetworkHandleType NetworkHandle, const uint8* PncRequestBitVectorPtr, Std_ReturnType result )`	
Service ID [hex]	0x29	
Sync/Async	Synchronous	
Reentrancy	Non-reentrant for the same NetworkHandle, reentrant otherwise	
Parameters (in)	NetworkHandle	Identification of the NM-channel
	PncRequestBitVectorPtr	Pointer to the PNC Request Bit Vector which was considered for transmission
	result	E_OK: The PNC Request Bit Vector has been transmitted E_NOT_OK: The transmission of the PNC Request Bit Vector failed
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Function called by <Bus>Nms to confirm the state of the transmission for the given PNC Request Bit Vector on the given NM-Channel.	
Available via	Nm.h	

⌋

8.3.6 Extra Call-back notifications

The following call-back notifications are provided by NM Interface for OEM specific extensions of bus specific NM components and are not required by any AUTOSAR module. In the context of the Basic functionality and NM Coordinator functionality they have no specific usage.

8.3.6.1 Nm_PduRxIndication

[SWS_Nm_00112] Definition of callback function Nm_PduRxIndication

Upstream requirements: SRS_BSW_00359

⌈

Service Name	Nm_PduRxIndication	
Syntax	`void Nm_PduRxIndication ( NetworkHandleType nmNetworkHandle )`	
Service ID [hex]	0x15	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Parameters (in)	nmNetworkHandle	Identification of the NM-channel
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Notification that a NM message has been received.	
Available via	Nm.h	

⌋

The notification that an NM message has been received is only needed for OEM specific extensions of the *NM Coordinator*.

[SWS_Nm_00164] ⌈Configuration of `Nm_PduRxIndication`: This function is only available if `NmPduRxIndicationEnabled` is set to TRUE.⌋

8.3.6.2 Nm_StateChangeNotification

[SWS_Nm_00114] Definition of callback function Nm_StateChangeNotification

Upstream requirements: SRS_BSW_00359

⌈

Service Name	Nm_StateChangeNotification	
Syntax	`void Nm_StateChangeNotification ( NetworkHandleType nmNetworkHandle, Nm_StateType nmPreviousState, Nm_StateType nmCurrentState )`	
Service ID [hex]	0x16	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Parameters (in)	nmNetworkHandle	Identification of the NM-channel
	nmPreviousState	Previous state of the NM-channel
	nmCurrentState	Current (new) state of the NM-channel

▽

△

Parameters (inout)	None
Parameters (out)	None
Return value	None
Description	Notification that the state of the lower layer <Bus>Nm has changed.
Available via	Nm.h

⌋

The notification that the state of the bus-specific NM has changed is only needed for OEM specific extensions and for the optional Dynamic PNC-to-channel-mapping feature.

[SWS_Nm_00165] ⌈Configuration of `Nm_StateChangeNotification`: This function is only available if `NmStateChangeIndEnabled` is set to TRUE.⌋

8.3.6.3 Nm_RepeatMessageIndication

[SWS_Nm_00230] Definition of callback function Nm_RepeatMessageIndication

Upstream requirements: SRS_BSW_00359

⌈

Service Name	Nm_RepeatMessageIndication	
Syntax	`void Nm_RepeatMessageIndication ( NetworkHandleType nmNetworkHandle, boolean pnLearningBitSet )`	
Service ID [hex]	0x1a	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Parameters (in)	nmNetworkHandle	Identification of the NM-channel
	pnLearningBitSet	TRUE if also the Partial Network Learning Bit was received, FALSE otherwise
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Service to indicate that an NM message with set Repeat Message Re- quest Bit has been received. This is needed for node detection and the Dynamic PNC-to-channel-mapping feature.	
Available via	Nm.h	

⌋

The notification that an NM message with the set Repeat Message Bit has been received is only needed for OEM specific extensions and for the optional Dynamic PNC-to-channel-mapping feature.

[SWS_Nm_00504] ⌈If `Nm_RepeatMessageIndication` is called with `pnLearningBitSet` set to `TRUE` and `NmDynamicPncToChannelMapping-Enabled` is set to `TRUE` for the provided `nmNetworkHandle` Nm shall call

`ComM_Nm_PncLearningBitIndication` with the corresponding network handle.⌋

8.3.6.4 Nm_TxTimeoutException

[SWS_Nm_00234] Definition of callback function Nm_TxTimeoutException

Upstream requirements: SRS_BSW_00359

⌈

Service Name	Nm_TxTimeoutException	
Syntax	`void Nm_TxTimeoutException ( NetworkHandleType nmNetworkHandle )`	
Service ID [hex]	0x1b	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Parameters (in)	nmNetworkHandle	–
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Service to indicate that an attempt to send an NM message failed.	
Available via	Nm.h	

⌋

The notification that an attempt to send an NM message failed is only needed for OEM specific extensions of the Nm.

8.3.6.5 Nm_CarWakeUpIndication

[SWS_Nm_00250] Definition of callback function Nm_CarWakeUpIndication

Upstream requirements: SRS_BSW_00359

⌈

Service Name	Nm_CarWakeUpIndication	
Syntax	`void Nm_CarWakeUpIndication ( NetworkHandleType nmChannelHandle )`	
Service ID [hex]	0x1d	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	nmChannelHandle	Identification of the NM-channel
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	

▽

△

Description	This function is called by a <Bus>Nm to indicate reception of a CWU request.
Available via	Nm.h

⌋

[SWS_Nm_00251] ⌈Configuration of `Nm_CarWakeUpIndication`: Optional
If `NmCarWakeUpRxEnabled` is **TRUE**, The Nm shall provide the API `Nm_CarWakeUpIndication`.⌋

8.4 Scheduled functions

Since the Base functionality (Chapter 7.1) does not contain any logic that needs to be invoked outside the scope of call from the upper or lower layer, the main function is only needed to implement the NM Coordinator functionality (Chapter 7.2).

[SWS_Nm_00020]

Upstream requirements: SRS_BSW_00373

⌈A scheduled main function shall only contain logic related to the *NM Coordinator functionality*.⌋

[SWS_Nm_00121]

Upstream requirements: SRS_BSW_00450

⌈In case the main function is called before the Nm has been initialized, the main function shall immediately return without yielding an error.⌋

Rationale: In case the NM Coordinator functionality is not used and/or disabled, calling the main function shall not yield in an error, but nothing should be performed.

8.4.1 Nm_MainFunction

[SWS_Nm_00118] Definition of scheduled function Nm_MainFunction

Upstream requirements: SRS_BSW_00424, SRS_BSW_00425

⌈

Service Name	Nm_MainFunction
Syntax	`void Nm_MainFunction ( void )`
Service ID [hex]	0x10
Description	This function implements the processes of the NM Interface, which need a fix cyclic scheduling.
Available via	SchM_Nm.h

⌋

[SWS_Nm_00279] ⌈If NmCoordinatorSupportEnabled is set to TRUE, the Nm_MainFunction API shall be available.⌋

8.5 Expected interfaces

This chapter lists all interfaces required from other modules.

8.5.1 Mandatory Interfaces

This chapter lists all interfaces required from other modules.

[SWS_Nm_00119] Definition of mandatory interfaces required by module Nm ⌈

API Function	*Header File*	*Description*
<Bus>Nm_GetState	–	Returns the state and the mode of the network management.
<Bus>Nm_NetworkRelease	–	Release the network, since ECU doesn't have to communicate on the bus.
<Bus>Nm_NetworkRequest	–	Request the network, since ECU needs to communicate on the bus.
<Bus>Nm_PassiveStartUp	–	Passive startup of the NM. It triggers the transition from Bus-Sleep Mode to the Network Mode without requesting the network.
ComM_Nm_BusSleepMode	ComM_Nm.h	Notification that the network management has entered Bus-Sleep Mode. This callback function should perform a transition of the hardware and transceiver to bus-sleep mode.
ComM_Nm_NetworkMode	ComM_Nm.h	Notification that the network management has entered Network Mode.
ComM_Nm_NetworkStartIndication	ComM_Nm.h	Indication that a NM-message has been received in the Bus Sleep Mode, what indicates that some nodes in the network have already entered the Network Mode.
ComM_Nm_PrepareBusSleepMode	ComM_Nm.h	Notification that the network management has entered Prepare Bus-Sleep Mode. Reentrancy: Reentrant (but not for the same NM-Channel)
ComM_Nm_RestartIndication	ComM_Nm.h	If NmIf has started to shut down the coordinated busses, AND not all coordinated busses have indicated bus sleep state, AND on at least on one of the coordinated busses NM is restarted, THEN the NM Interface shall call the callback function ComM_Nm_RestartIndication with the nmNetworkHandle of the channels which have already indicated bus sleep state.

⌋

8.5.2 Optional Interfaces

This chapter defines all interfaces that are required to fulfill an optional functionality of the module.

[SWS_Nm_00166] Definition of optional interfaces requested by module Nm ⌈

API Function	*Header File*	*Description*
<Bus>Nm_ActivateTxPnShutdownMsg	–	NM indicate to activate the transmission of PN shutdown messages on the given NM-Channel. This results in transmission of a NM-PDU with PNSR bit set to 1 (PN shutdown message).
<Bus>Nm_CheckRemoteSleep Indication	–	Check if remote sleep indication takes place or not.
<Bus>Nm_DeactivateTxPnShutdown Msg	–	NM indicate to deactive the transmission of PN shutdown messages on the given NM-Channel. This result in transmission of a usual NM-PDUs with PNSR bit set to 0.
<Bus>Nm_DisableCommunication	–	Disable the NM PDU transmission ability.
<Bus>Nm_EnableCommunication	–	Enable the NM PDU transmission ability.
<Bus>Nm_GetLocalNodeIdentifier	–	Get node identifier configured for the local node.
<Bus>Nm_GetNodeIdentifier	–	Get node identifier out of the last successfully received NM-message.
<Bus>Nm_GetPduData	–	Pointer where NM PDU shall be copied to.
<Bus>Nm_GetUserData	–	Get user data out of the last successfully received NM message.
<Bus>Nm_PnLearningRequest	–	–
<Bus>Nm_RepeatMessageRequest	–	Request a Repeat Message Request to be transmitted next on the bus.
<Bus>Nm_RequestBus Synchronization	–	Request bus synchronization.
<Bus>Nm_RequestSynchronizedPnc Shutdown	–	–
<Bus>Nm_SetSleepReadyBit	–	Set the NM Coordinator Sleep Ready bit in the Control Bit Vector
<Bus>Nm_SetUserData	–	Set user data for NM messages transmitted next on the bus.
BswM_Nm_CarWakeUpIndication	BswM_Nm.h	Function called by Nm to indicate a CarWakeup.
Com_SendSignal	Com.h	The service Com_SendSignal updates the signal object identified by SignalId with the signal referenced by the SignalDataPtr parameter.
ComM_Nm_ForwardSynchronizedPnc Shutdown	ComM_Nm.h	If an ECU in role of an intermediate PNC coordinator receives a PN shutdown message via a <Bus>Nm, then ComM is immediately indicated via ComM_Nm_ForwardSynchronizedPncShutdown to forward the request for a synchronized PNC shutdown of the affected PNCs given by Pnc RequestBitVectorPtr. Therefore, ComM will immediately release the affected PNC state machines and forward the PNC Request Bit Vector to the affected ComM Channels and the corresponding NM channels, respectively. Note: This supports a nearly synchronized PNC shutdown across the PN topology from the top-level PNC coordinator down to the subordinated PNC node.
ComM_Nm_PncLearningBitIndication	ComM_Nm.h	Service to indicate that an NM message with set PNC Learning Bit has been received.
ComM_Nm_RepeatMessageLeft Indication	ComM_Nm.h	Notification that the state of all <Bus>Nm has left RepeatMessage. This interface is used to indicate by the optional Dynamic PNC-to-channel-mapping feature to indicate that learning phase ends.
ComM_Nm_UpdateEIRA	ComM_Nm.h	Function to indicate the current aggregated external / internal PNC request called by Nm.

▽

△

API Function	Header File	Description
ComM_Nm_UpdateERA	ComM_Nm.h	Function to indicate the current external PNC request per channel called by Nm.
Det_ReportError	Det.h	Service to report development errors.

⌋

8.5.3 Configurable Interfaces

In this chapter all interfaces are listed where the target function could be configured. The target function is usually a call-back function. The names of these kind of interfaces are not fixed because they are configurable.

8.5.3.1 NmCarWakeUpCallout

[SWS_Nm_00291] Definition of configurable interface <NmCarWakeUpCallout> ⌈

Service Name	<NmCarWakeUpCallout>	
Syntax	`void <NmCarWakeUpCallout> ( NetworkHandleType nmNetworkHandle )`	
Service ID [hex]	0x20	
Sync/Async	Asynchronous	
Reentrancy	Non Reentrant	
Parameters (in)	nmNetworkHandle	Identification of the NM-channel
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Callout function to be called by Nm_CarWakeUpIndication()	
Available via	Nm_Externals.h	

⌋

9 Sequence diagrams

9.1 Basic functionality

The role of the *Basic functionality* of the **Nm** is to act as a dispatcher of functions between the ComM and the Bus Specific NM modules. Therefore, no sequence diagram is provided.

9.2 Seq of NM Coordinator functionality

Figure shows the sequence diagram for the shutdown of network of the *NM Coordinator* functionality.

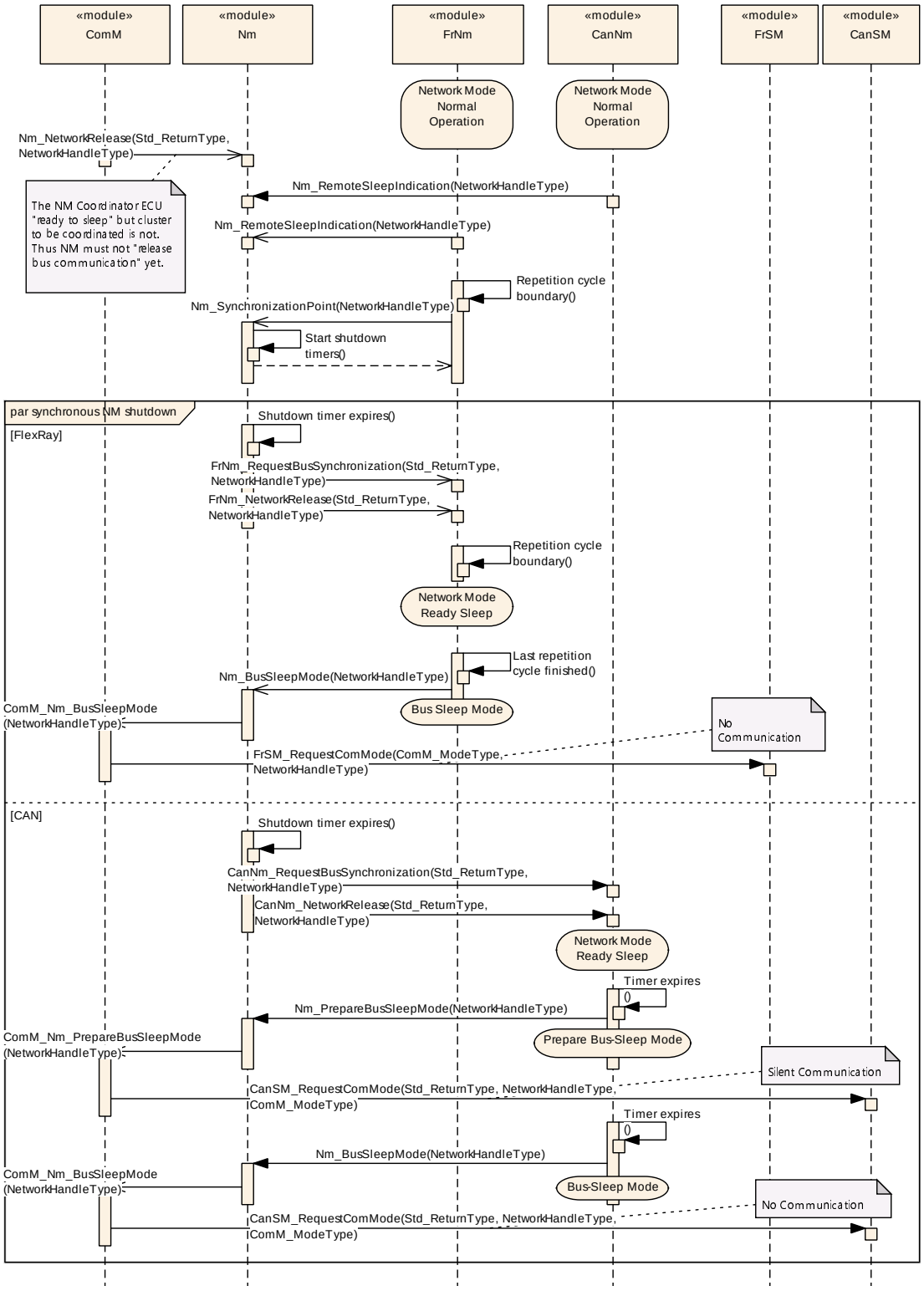

Figure 9.1: Nm Coordination

9.3 Sequence of Partial network functionality

The following sequence diagram shows the interaction between **Nm** and the **CanIf** module as example. The following deviations has to be considered if using FlexRay communincation stack

- FrNm has no ECUC parameter similar to `CanNmAllNmMessagesKeepAwake`
- FrNm needs to check ECUC parameter `FrIfImmediate` of the NM PDU configured in FrIf
- If using FrNm, the NM Pdu is always fechted via `FrNm_TriggerTransmit`. There is no ECUC parameter similar to `CanIfTxPduTriggerTransmit`

The following deviations have to be considered if using an Ethernet communication stack:

- The UdpNm module interacts with the SoAd (and NOT with the EthIf). Therefore UdpNm has to call `SoAd_IfTransmit` to trigger a transmission of a NM PDU. The SoAd has to call `UdpNm_SoAdIfRxIndication` to indicate the reception of NM PDU
- UpdNm needs to the check the ECUC parameter `SoAdBswModules/-SoAdIfTriggerTransmit` of the SoAd, to determine if the NM PDU is fetched via call of `UdpNm_SoAdIfTriggerTransmit`

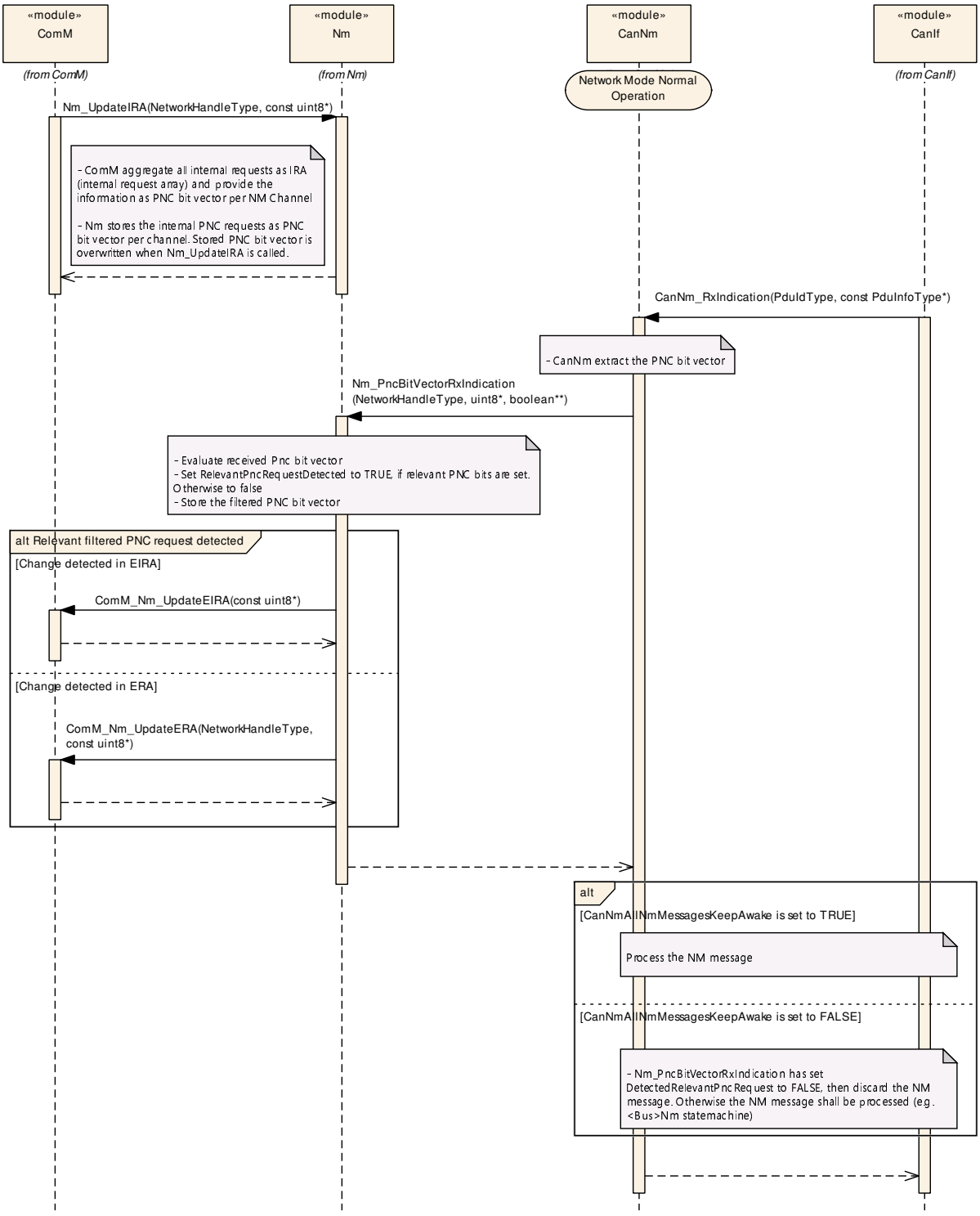

Figure 9.2: Partial Network functionality (part 1)

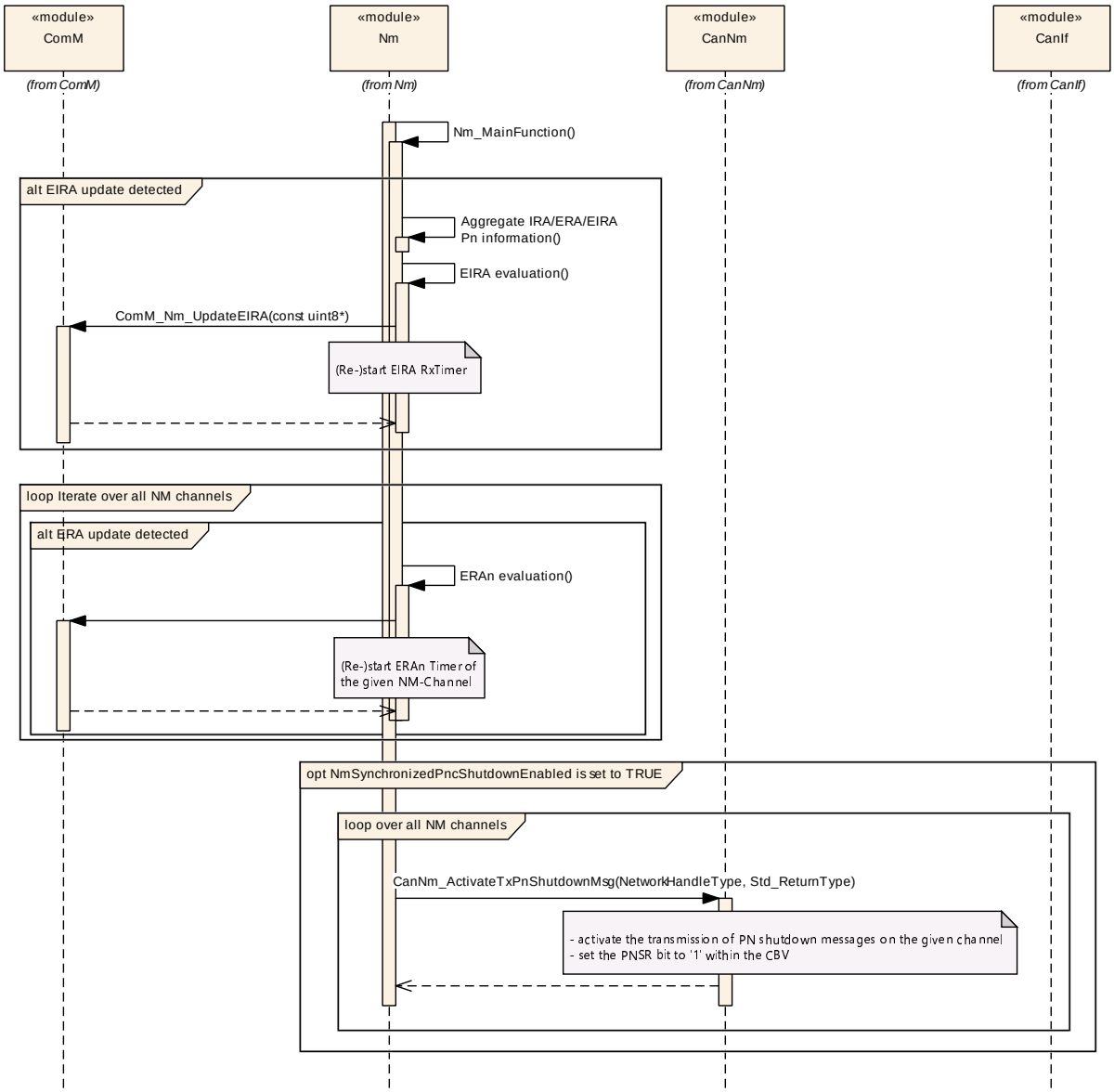

Figure 9.3: Partial Network functionality (part 2)

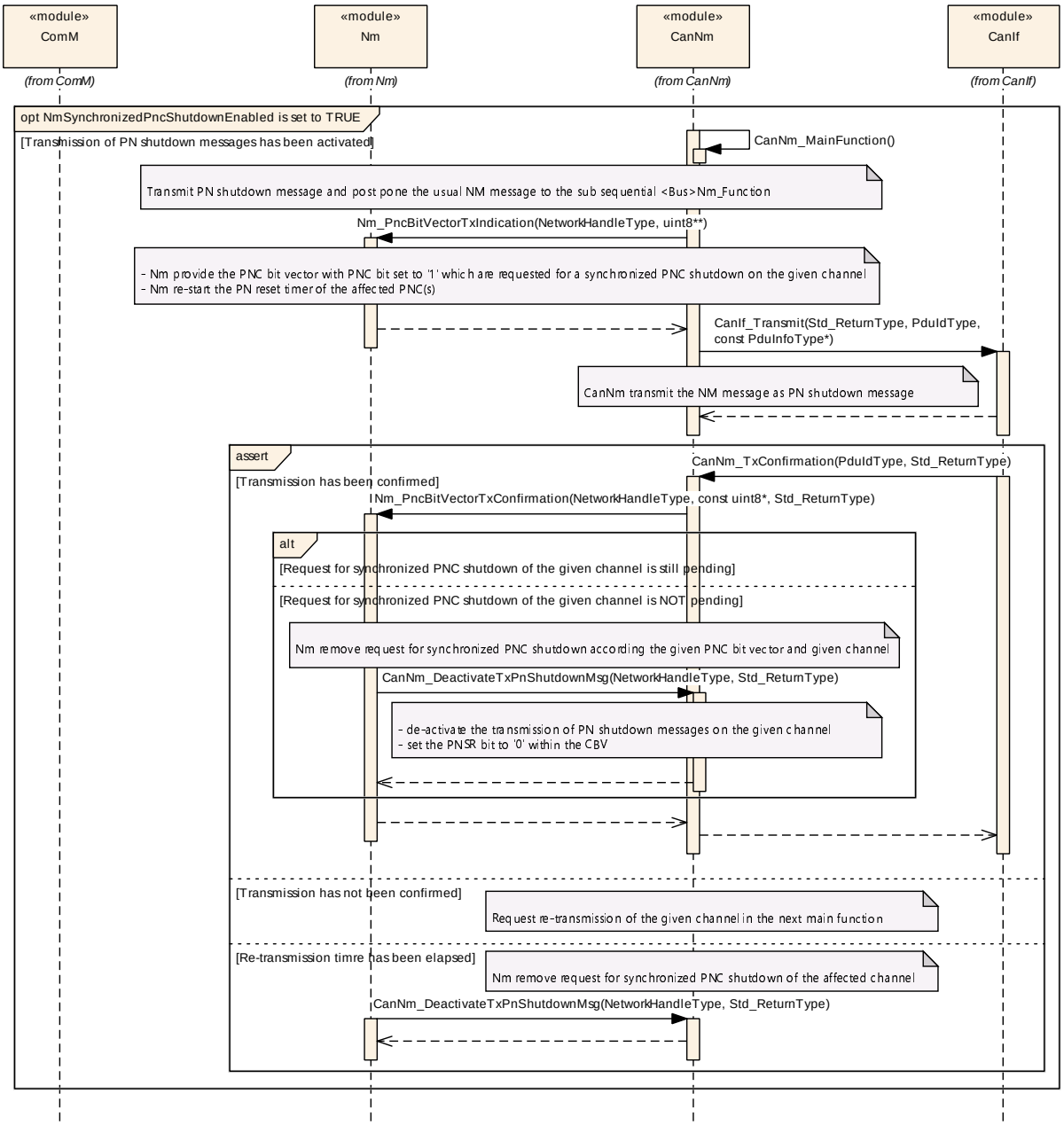

Figure 9.4: Partial Network functionality (part 3)

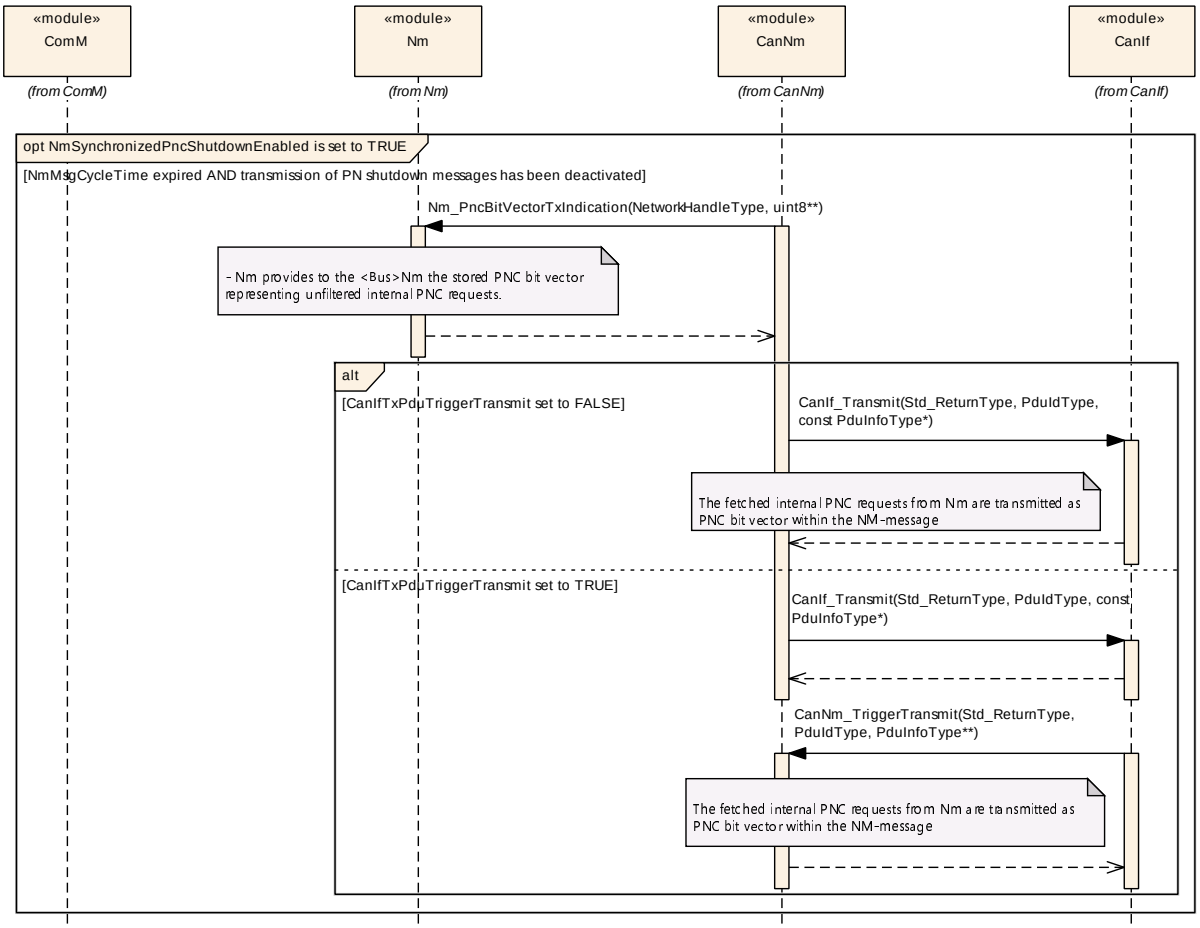

Figure 9.5: Partial Network functionality (part 4)

10 Configuration specification

The following chapter contains tables of all configuration parameters and switches used to determine the functional units of the Generic Network Management Interface. The default values of configuration parameters are denoted as bold.

In general, this chapter defines configuration parameters and their clustering into containers. Chapter 10.1 describes fundamentals. Chapter 10.2, Chapter 10.3 and Chapter 10.4 specifies the structure (containers) and the parameters of the Nm. The Chapter 10.5 specifies published information of the Nm.

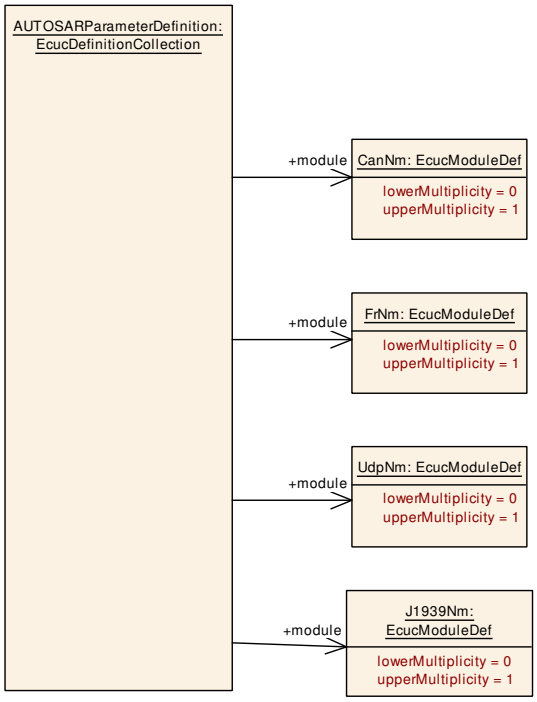

Figure 10.1: Network Management Overview

10.1 How to read this chapter

For details refer to [2] Chapter 10.1 *"Introduction to configuration specification"*.

10.2 Configuration parameters

The following Chapters summarize all configuration parameters for the Nm. The detailed meanings of most parameters are described in Chapter 7 and Chapter 8.
Note that the behavior and configuration of Nm is closely dependent on the behavior and configuration of the different bus specific NM modules used.

10.2.1 Nm

[ECUC_Nm_00243] Definition of EcucModuleDef Nm ⌈

Module Name	Nm
Description	The Generic Network Management Interface module
Post-Build Selectable Support	true
Post-Build Loadable Support	true

Included Containers		
Container Name	**Multiplicity**	**Description**
NmChannelConfig	1..*	This container contains the configuration (parameters) of the bus channel(s). The channel parameter shall be harmonized within the whole communication stack.
NmGlobalConfig	1	This container contains all global configuration parameters of the Nm Interface.

⌋

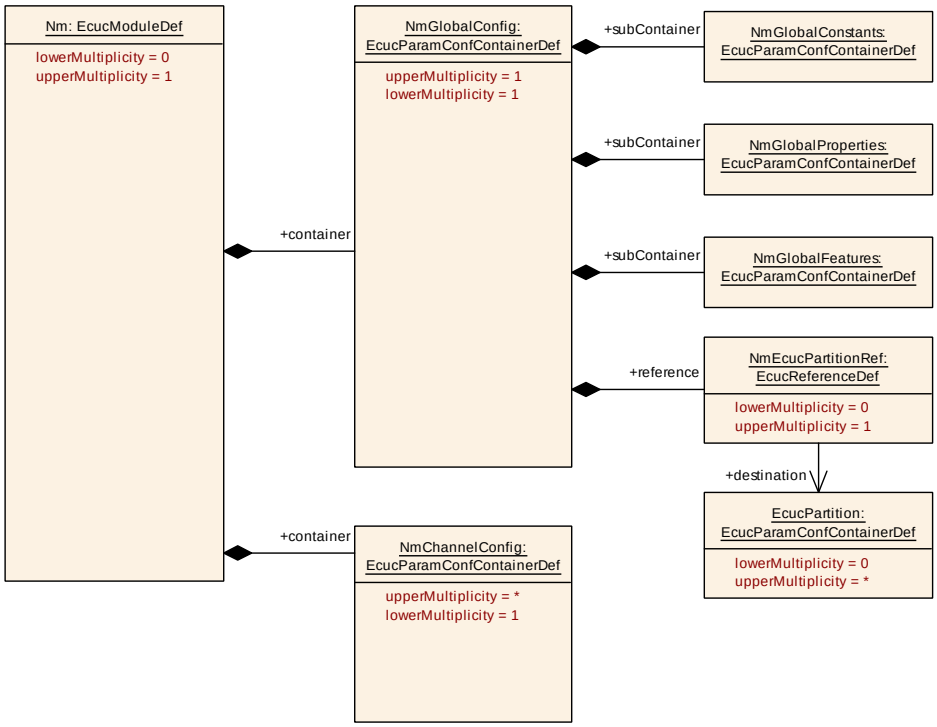

Figure 10.2: Nm configuration container overview

10.3 Global configurable parameters

10.3.1 NmGlobalConfig

[ECUC_Nm_00196] Definition of EcucParamConfContainerDef NmGlobalConfig ⌈

Container Name	NmGlobalConfig
Parent Container	Nm
Upstream mapping defined	–
Description	This container contains all global configuration parameters of the Nm Interface.
Multiplicity	1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	Multiplicity	ECUC ID
NmEcucPartitionRef	0..1	[ECUC_Nm_00245]

Included Containers		
Container Name	Multiplicity	Description
NmGlobalConstants	1	Global Nm constants
NmGlobalFeatures	1	Global Nm features
NmGlobalProperties	1	Global Nm properties

⌋

[ECUC_Nm_00245] Definition of EcucReferenceDef NmEcucPartitionRef ⌈

Parameter Name	NmEcucPartitionRef
Parent Container	NmGlobalConfig
Upstream mapping defined	–
Description	Reference to EcucPartition, where Nm module is assigned to.
Multiplicity	0..1
Type	Reference to EcucPartition
Pre-Compile only	true

⌋

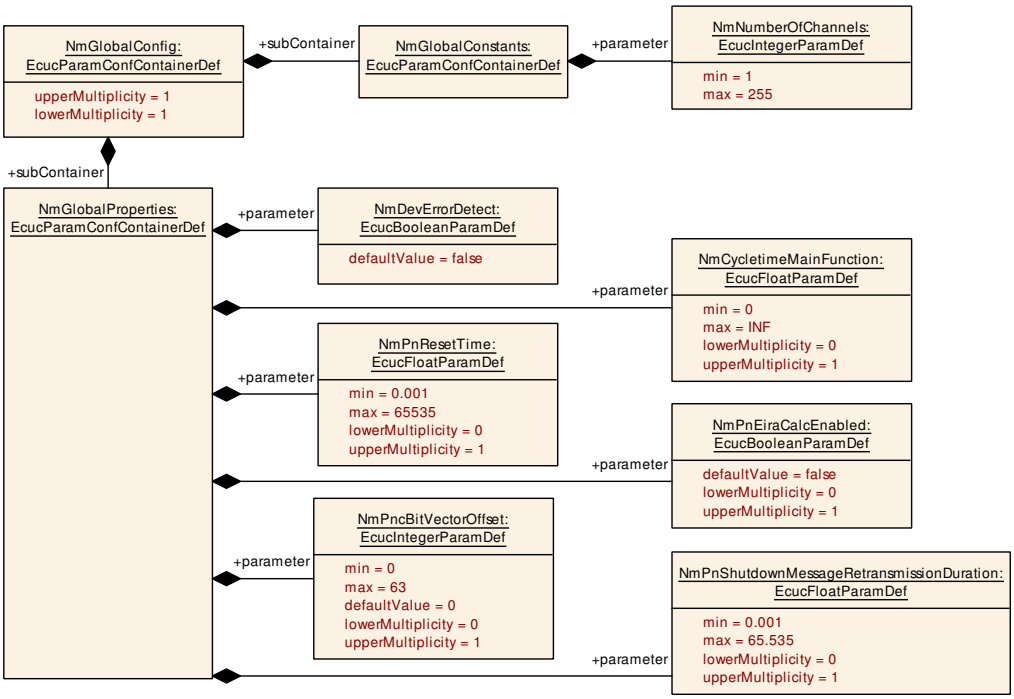

Figure 10.3: NmGlobalConfig overview

10.3.2 NmGlobalConstants

[ECUC_Nm_00198] Definition of EcucParamConfContainerDef NmGlobalConstants ⌈

Container Name	NmGlobalConstants
Parent Container	NmGlobalConfig
Upstream mapping defined	–
Description	Global Nm constants
Multiplicity	1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
NmNumberOfChannels	1	[ECUC_Nm_00201]

No Included Containers

⌋

[ECUC_Nm_00201] Definition of EcucIntegerParamDef NmNumberOfChannels ⌈

Parameter Name	NmNumberOfChannels	
Parent Container	NmGlobalConstants	
Upstream mapping defined	–	
Description	Number of NM channels allowed within one ECU.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	1 .. 255	
Default value	–	
Pre-Compile only	true	

⌋

10.3.3 NmGlobalProperties

[ECUC_Nm_00199] Definition of EcucParamConfContainerDef NmGlobalProperties ⌈

Container Name	NmGlobalProperties
Parent Container	NmGlobalConfig
Upstream mapping defined	–
Description	Global Nm properties
Multiplicity	1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
NmCycletimeMainFunction	0..1	[ECUC_Nm_00205]
NmDevErrorDetect	1	[ECUC_Nm_00203]
NmPncBitVectorOffset	0..1	[ECUC_Nm_00252]
NmPnEiraCalcEnabled	0..1	[ECUC_Nm_00251]
NmPnResetTime	0..1	[ECUC_Nm_00250]
NmPnShutdownMessageRetransmissionDuration	0..1	[ECUC_Nm_00260]

No Included Containers

⌋

[ECUC_Nm_00205] Definition of EcucFloatParamDef NmCycletimeMainFunction ⌈

Parameter Name	NmCycletimeMainFunction
Parent Container	NmGlobalProperties
Upstream mapping defined	[11, CP_TPS_SystemTemplate]

▽

△

Description	The period between successive calls to the Main Function of the NM Interface in seconds. Dependency: If NmCoordinatorSupportEnabled is set to TRUE, then the NmCycletime MainFunction shall be configured.	
Multiplicity	0..1	
Type	EcucFloatParamDef	
Range	]0 .. INF[	
Default value	–	
Pre-Compile only	true	

⌋

[ECUC_Nm_00203] Definition of EcucBooleanParamDef NmDevErrorDetect ⌈

Parameter Name	NmDevErrorDetect
Parent Container	NmGlobalProperties
Upstream mapping defined	–
Description	Switches the development error detection and notification on or off. • true: detection and notification is enabled. • false: detection and notification is disabled.
Multiplicity	1
Type	EcucBooleanParamDef
Default value	false
Pre-Compile only	true

⌋

[ECUC_Nm_00252] Definition of EcucIntegerParamDef NmPncBitVectorOffset ⌈

Parameter Name	NmPncBitVectorOffset	
Parent Container	NmGlobalProperties	
Upstream mapping defined	[11, CP_TPS_SystemTemplate]	
Description	Parameter to configure the offset in bytes of the PNC Request Bit Vector that contains the PNC requests, which is transmitted within NM PDU by the corresponding <Bus>Nm. Dependency: This parameter is only valid if NmPartialNetworkSupportEnabled is set to TRUE. NmPncBitVectorOffset == Number of <Bus>Nm Sytem Bytes OR NmPncBit VectorOffset + NmPncBitVectorLength == NM PduLength.	
Multiplicity	0..1	
Type	EcucIntegerParamDef	
Range	0 .. 63	
Default value	0	
Pre-Compile only	true	

⌋

[ECUC_Nm_00251] Definition of EcucBooleanParamDef NmPnEiraCalcEnabled ⌈

Parameter Name	NmPnEiraCalcEnabled
Parent Container	NmGlobalProperties
Upstream mapping defined	[11, CP_TPS_SystemTemplate]
Description	Specifies if NmIf calculates the PNC request information for internal and external requests (EIRA) true: PN request are calculated false: PN request are not calculated Note: A PNC coordinator (NmPnEraCalcEnabled set to TRUE) has always set NmPn EiraCalcEnabled to TRUE. Dependency: This parameter can only be set to TRUE if NmPartialNetworkSupport Enabled is set to TRUE. If NmPnEraCalcEnabled is set to TRUE than this parameter shall be set to TRUE.
Multiplicity	0..1
Type	EcucBooleanParamDef
Default value	false
Pre-Compile only	true

⌋

[ECUC_Nm_00250] Definition of EcucFloatParamDef NmPnResetTime ⌈

Parameter Name	NmPnResetTime
Parent Container	NmGlobalProperties
Upstream mapping defined	[11, CP_TPS_SystemTemplate]
Description	Specifies the runtime of the reset time in seconds. This reset time is valid for the reset of PNC requests in the EIRA and in the ERA. The value shall be the same for every channel. Thus it is a global config parameter. Dependency: This parameter is only valid if NmPartialNetworkSupportEnabled is set to TRUE. NmPnResetTime > <Can|Udp>NmMsgCycleTime NmPnResetTime > FrNm DataCycle * FR Cycle Time
Multiplicity	0..1
Type	EcucFloatParamDef
Range	[0.001 .. 65535]
Default value	–
Pre-Compile only	true

⌋

[ECUC_Nm_00260] Definition of EcucFloatParamDef NmPnShutdownMessage RetransmissionDuration ⌈

Parameter Name	NmPnShutdownMessageRetransmissionDuration
Parent Container	NmGlobalProperties
Upstream mapping defined	[11, CP_TPS_SystemTemplate]

▽

△

Description	Specifies the duration in seconds of the retransmission phase of a PN shutdown message. A retransmission shall be performed per affected NM channel, as long as the PN shutdown message could not be successfully sent and the retransmission timer is running. The value shall be a multiple integral NmMainFunctionPeriod. Dependency: Only valid NmSynchronizedPncShutdownEnabled == TRUE. NmPn ShutdownMessageRetransmissionDuration <= NmPnResetTime.
Multiplicity	0..1
Type	EcucFloatParamDef
Range	[0.001 .. 65.535]
Default value	–
Pre-Compile only	true

⌋

10.3.4 NmGlobalFeatures

[ECUC_Nm_00200] Definition of EcucParamConfContainerDef NmGlobalFeatures ⌈

Container Name	NmGlobalFeatures
Parent Container	NmGlobalConfig
Upstream mapping defined	–
Description	Global Nm features
Multiplicity	1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
NmBusSynchronizationEnabled	1	[ECUC_Nm_00208]
NmCarWakeUpCallout	0..1	[ECUC_Nm_00234]
NmCarWakeUpRxEnabled	1	[ECUC_Nm_00235]
NmComControlEnabled	1	[ECUC_Nm_00210]
NmCoordinatorSupportEnabled	1	[ECUC_Nm_00206]
NmCoordinatorSyncSupport	1	[ECUC_Nm_00240]
NmDynamicPncToChannelMappingSupport	1	[ECUC_Nm_00246]
NmGlobalCoordinatorTime	0..1	[ECUC_Nm_00237]
NmPartialNetworkSupportEnabled	0..1	[ECUC_Nm_00253]
NmPduRxIndicationEnabled	1	[ECUC_Nm_00214]
NmRemoteSleepIndEnabled	1	[ECUC_Nm_00207]
NmStateChangeIndEnabled	1	[ECUC_Nm_00215]
NmSynchronizedPncShutdownEnabled	0..1	[ECUC_Nm_00249]
NmUserDataEnabled	1	[ECUC_Nm_00211]

No Included Containers

⌋

[ECUC_Nm_00208] Definition of EcucBooleanParamDef NmBusSynchronization Enabled ⌈

Parameter Name	NmBusSynchronizationEnabled
Parent Container	NmGlobalFeatures
Upstream mapping defined	[11, CP_TPS_SystemTemplate]
Description	Pre-processor switch for enabling bus synchronization support of the <Bus>Nms. This feature is required for NM Coordinator nodes only. Dependency: This parameter must be enabled if NmCoordinatorSupportEnabled is enabled.
Multiplicity	1
Type	EcucBooleanParamDef
Default value	–
Pre-Compile only	true

⌋

[ECUC_Nm_00234] Definition of EcucFunctionNameDef NmCarWakeUpCallout ⌈

Parameter Name	NmCarWakeUpCallout
Parent Container	NmGlobalFeatures
Upstream mapping defined	–
Description	Name of the callout function to be called if Nm_CarWakeUpIndication() is called. If this parameter is not configured, the Nm will call BswM_Nm_CarWakeUpIndication. Dependency: only available if NmCarWakeUpRxEnabled == TRUE
Multiplicity	0..1
Type	EcucFunctionNameDef
Default value	–
Regular Expression	–
Pre-Compile only	true

⌋

[ECUC_Nm_00235] Definition of EcucBooleanParamDef NmCarWakeUpRxEnabled ⌈

Parameter Name	NmCarWakeUpRxEnabled
Parent Container	NmGlobalFeatures
Upstream mapping defined	–
Description	Enables or disables CWU detection. FALSE - CarWakeUp not supported TRUE - Car WakeUp supported
Multiplicity	1
Type	EcucBooleanParamDef
Default value	false
Pre-Compile only	true

⌋

[ECUC_Nm_00210] Definition of EcucBooleanParamDef NmComControlEnabled ⌈

Parameter Name	NmComControlEnabled
Parent Container	NmGlobalFeatures
Upstream mapping defined	[11, CP_TPS_SystemTemplate]
Description	Pre-processor switch for enabling the Communication Control support.
Multiplicity	1
Type	EcucBooleanParamDef
Default value	–
Pre-Compile only	true

⌋

[ECUC_Nm_00206] Definition of EcucBooleanParamDef NmCoordinatorSupportEnabled ⌈

Parameter Name	NmCoordinatorSupportEnabled
Parent Container	NmGlobalFeatures
Upstream mapping defined	[11, CP_TPS_SystemTemplate]
Description	Pre-processor switch for enabling NM Coordinator support. Dependency: Only valid if at least one NM channel exists which has NmPassiveModeEnabled set to FALSE.
Multiplicity	1
Type	EcucBooleanParamDef
Default value	false
Pre-Compile only	true

⌋

[ECUC_Nm_00240] Definition of EcucBooleanParamDef NmCoordinatorSyncSupport ⌈

Parameter Name	NmCoordinatorSyncSupport
Parent Container	NmGlobalFeatures
Upstream mapping defined	[11, CP_TPS_SystemTemplate]
Description	Enables/disables the coordinator synchronisation support. Dependency: NmCoordinatorSyncSupport shall only be valid if NmCoordinatorSupportEnabled is TRUE.
Multiplicity	1
Type	EcucBooleanParamDef
Default value	–
Pre-Compile only	true

⌋

[ECUC_Nm_00246] Definition of EcucBooleanParamDef NmDynamicPncToChannelMappingSupport

Status: DRAFT

⌈

Parameter Name	NmDynamicPncToChannelMappingSupport
Parent Container	NmGlobalFeatures
Upstream mapping defined	[11, CP_TPS_SystemTemplate]
Description	Precompile time switch to enable the dynamic PNC-to-channel-mapping handling. False: Dynamic PNC-to-channel-mapping is disabled True: Dynamic PNC-to-channel-mapping is enabled **Tags:** atp.Status=draft
Multiplicity	1
Type	EcucBooleanParamDef
Default value	–
Pre-Compile only	true

⌋

[ECUC_Nm_00237] Definition of EcucFloatParamDef NmGlobalCoordinatorTime

⌈

Parameter Name	NmGlobalCoordinatorTime	
Parent Container	NmGlobalFeatures	
Upstream mapping defined	[11, CP_TPS_SystemTemplate]	
Description	This parameter defines the maximum shutdown time of a connected and coordinated NM-Cluster. Note:This includes nested connections. Dependency: NmGlobalCoordinatorTime shall only be valid if NmCoordinatorSupport Enabled is TRUE.	
Multiplicity	0..1	
Type	EcucFloatParamDef	
Range	[0 .. INF]	
Default value	–	
Pre-Compile only	true	

⌋

[ECUC_Nm_00253] Definition of EcucBooleanParamDef NmPartialNetworkSupportEnabled ⌈

Parameter Name	NmPartialNetworkSupportEnabled
Parent Container	NmGlobalFeatures
Upstream mapping defined	[11, CP_TPS_SystemTemplate]
Description	Pre-processor switch for enabling the Nm Partial Network support. Dependency: This parameter can only be set to TRUE if NmCoordinatorSupport Enabled is set to FALSE.
Multiplicity	0..1
Type	EcucBooleanParamDef
Default value	false
Pre-Compile only	true

⌋

[ECUC_Nm_00214] Definition of EcucBooleanParamDef NmPduRxIndicationEnabled ⌈

Parameter Name	NmPduRxIndicationEnabled
Parent Container	NmGlobalFeatures
Upstream mapping defined	[11, CP_TPS_SystemTemplate]
Description	Pre-processor switch for enabling the PDU Rx Indication.
Multiplicity	1
Type	EcucBooleanParamDef
Default value	–
Pre-Compile only	true

⌋

[ECUC_Nm_00207] Definition of EcucBooleanParamDef NmRemoteSleepIndEnabled ⌈

Parameter Name	NmRemoteSleepIndEnabled
Parent Container	NmGlobalFeatures
Upstream mapping defined	[11, CP_TPS_SystemTemplate]
Description	Pre-processor switch for enabling Remote Sleep Indication support. This feature is required for a Gateway or Nm Coordinator functionality. Note that this feature should not be used if all NM channels have Passive Mode enabled. Dependency: If NmCoordinatorSupportEnabled == TRUE then NmRemoteSleepInd Enabled = TRUE
Multiplicity	1
Type	EcucBooleanParamDef
Default value	–
Pre-Compile only	true

⌋

[ECUC_Nm_00215] Definition of EcucBooleanParamDef NmStateChangeIndEnabled ⌈

Parameter Name	NmStateChangeIndEnabled
Parent Container	NmGlobalFeatures
Upstream mapping defined	[11, CP_TPS_SystemTemplate]
Description	Pre-processor switch for enabling the Network Management state change notification. Dependency: NmStateChangeIndEnabled = TRUE if NmDynamicPncToChannel MappingSupport == TRUE
Multiplicity	1
Type	EcucBooleanParamDef
Default value	–
Pre-Compile only	true

⌋

[ECUC_Nm_00249] Definition of EcucBooleanParamDef NmSynchronizedPncShutdownEnabled ⌈

Parameter Name	NmSynchronizedPncShutdownEnabled
Parent Container	NmGlobalFeatures
Upstream mapping defined	–
Description	Enables or disables support of synchronized PNC shutdown. FALSE: synchronized PNC shutdown is disabled TRUE: synchronized PNC shutdown is enabled Dependency: This parameter can only be set to TRUE if NmPartialNetworkSupport Enabled is set to TRUE.
Multiplicity	0..1
Type	EcucBooleanParamDef
Default value	false
Pre-Compile only	true

⌋

[ECUC_Nm_00211] Definition of EcucBooleanParamDef NmUserDataEnabled ⌈

Parameter Name	NmUserDataEnabled
Parent Container	NmGlobalFeatures
Upstream mapping defined	[11, CP_TPS_SystemTemplate]
Description	Pre-processor switch for enabling User Data support.
Multiplicity	1
Type	EcucBooleanParamDef
Default value	–
Pre-Compile only	true

⌋

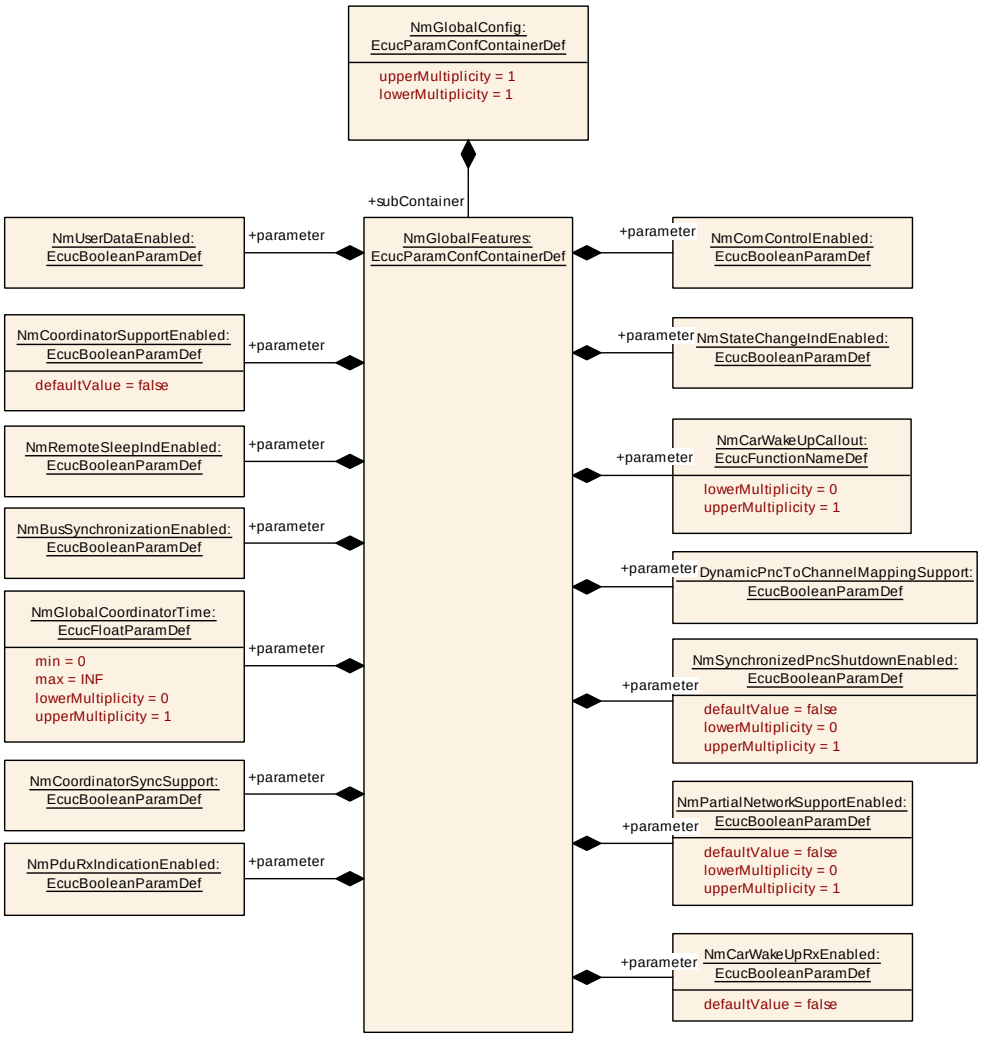

Figure 10.4: NmGlobalFeatures overview

10.4 Channel configurable parameters

10.4.1 NmChannelConfig

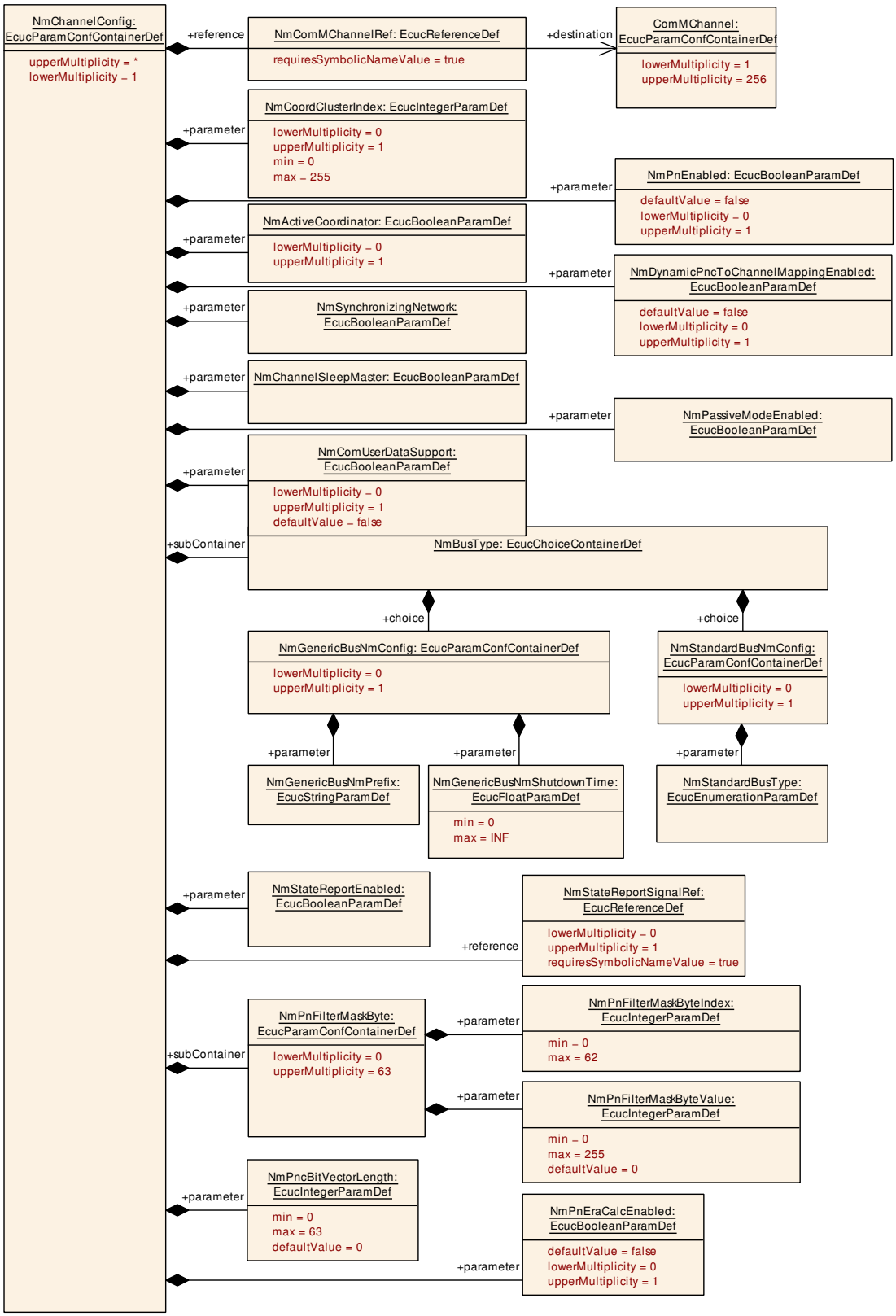

Figure 10.5: NmChannelConfig overview

[ECUC_Nm_00197] Definition of EcucParamConfContainerDef NmChannelConfig ⌈

Container Name	NmChannelConfig
Parent Container	Nm
Upstream mapping defined	[11, CP_TPS_SystemTemplate]
Description	This container contains the configuration (parameters) of the bus channel(s). The channel parameter shall be harmonized within the whole communication stack.
Multiplicity	1..*
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
NmActiveCoordinator	0..1	[ECUC_Nm_00236]
NmChannelSleepMaster	1	[ECUC_Nm_00227]
NmComUserDataSupport	0..1	[ECUC_Nm_00241]
NmCoordClusterIndex	0..1	[ECUC_Nm_00221]
NmDynamicPncToChannelMappingEnabled	0..1	[ECUC_Nm_00248]
NmPassiveModeEnabled	1	[ECUC_Nm_00242]
NmPncBitVectorLength	1	[ECUC_Nm_00258]
NmPnEnabled	0..1	[ECUC_Nm_00254]
NmPnEraCalcEnabled	0..1	[ECUC_Nm_00259]
NmStateReportEnabled	1	[ECUC_Nm_00231]
NmSynchronizingNetwork	1	[ECUC_Nm_00223]
NmComMChannelRef	1	[ECUC_Nm_00217]
NmStateReportSignalRef	0..1	[ECUC_Nm_00232]

Included Containers		
Container Name	**Multiplicity**	**Description**
NmBusType	1	Choose the NmBusType
NmPnFilterMaskByte	0..63	Information for the filter of the PNC Request Bit Vector.

⌋

[ECUC_Nm_00236] Definition of EcucBooleanParamDef NmActiveCoordinator ⌈

Parameter Name	NmActiveCoordinator
Parent Container	NmChannelConfig
Upstream mapping defined	[11, CP_TPS_SystemTemplate]
Description	This parameter indicates whether a NM channel - part of a Nm Coordination cluster - will be coordinated actively (NmActiveCoordinator = TRUE) or passively (NmActive Coordinator = FALSE). Dependency: If the NmCoordinatorSyncSupport is set to true this feature is available. Only one channel per Coordination cluster can have NmActiveCoordinator = FALSE. This parameter is mandatory if this channel belongs to a Coordination cluster (see ECUC_Nm_00221). Value cannot be set to FALSE in case BusNmType is set to NM_BUSNM_LOCALNM (i.e. no passive coordination for this type).
Multiplicity	0..1

▽

△

Type	EcucBooleanParamDef
Default value	–
Pre-Compile only	true

⌋

[ECUC_Nm_00227] Definition of EcucBooleanParamDef NmChannelSleepMaster ⌈

Parameter Name	NmChannelSleepMaster
Parent Container	NmChannelConfig
Upstream mapping defined	[11, CP_TPS_SystemTemplate]
Description	This parameter shall be set to indicate if the sleep of this network can be absolutely decided by the local node only and that no other nodes can oppose that decision. If this parameter is set to TRUE, the Nm shall assume that the channel is always ready to go to sleep and that no calls to Nm_RemoteSleepIndication or Nm_RemoteSleep Cancellation will be made from the <Bus>Nm representing this channel. If this parameter is set to FALSE, the Nm shall not assume that the network is ready to sleep until a call has been made to Nm_RemoteSleepCancellation. Dependency: If the parameter NmCoordClusterIndex is not defined, this parameter is not valid.
Multiplicity	1
Type	EcucBooleanParamDef
Default value	–
Pre-Compile only	true

⌋

[ECUC_Nm_00241] Definition of EcucBooleanParamDef NmComUserDataSupport ⌈

Parameter Name	NmComUserDataSupport
Parent Container	NmChannelConfig
Upstream mapping defined	[11, CP_TPS_SystemTemplate]
Description	This parameter indicates whether on a NM channel user data is accessed via Com signals or by SetUserData API. Dependency: NmComUserDataSupport shall be equal to <Bus>NmComUserData Support
Multiplicity	0..1
Type	EcucBooleanParamDef
Default value	false
Pre-Compile only	true

⌋

[ECUC_Nm_00221] Definition of EcucIntegerParamDef NmCoordClusterIndex ⌈

Parameter Name	NmCoordClusterIndex
Parent Container	NmChannelConfig
Upstream mapping defined	[11, CP_TPS_SystemTemplate]

▽

△

Description	If this parameter is undefined for a channel, the corresponding bus does not belong to an NM coordination cluster. Dependency: If NmCoordClusterIndex is defined than NmPassiveModeEnabled has to be FALSE for this channel.
Multiplicity	0..1
Type	EcucIntegerParamDef
Range	0 .. 255
Default value	–
Pre-Compile only	true

⌋

[ECUC_Nm_00248] Definition of EcucBooleanParamDef NmDynamicPncToChannelMappingEnabled

Status: DRAFT

⌈

Parameter Name	NmDynamicPncToChannelMappingEnabled
Parent Container	NmChannelConfig
Upstream mapping defined	[11, CP_TPS_SystemTemplate]
Description	Channel-specific parameter to enable the dynamic PNC-to-channel-mapping feature. False: Dynamic PNC-to-channel-mapping is disabled True: Dynamic PNC-to-channel-mapping is enabled **Tags:** atp.Status=draft Dependency: Shall only be TRUE if NmDynamicPncToChannelMappingSupport is TRUE
Multiplicity	0..1
Type	EcucBooleanParamDef
Default value	false
Pre-Compile only	true

⌋

[ECUC_Nm_00242] Definition of EcucBooleanParamDef NmPassiveModeEnabled ⌈

Parameter Name	NmPassiveModeEnabled
Parent Container	NmChannelConfig
Upstream mapping defined	[11, CP_TPS_SystemTemplate]
Description	This parameter indicates whether a NM channel is active,e.g. can request communication and keep the bus awake, or passive, e.g. can just be woken up and kept awake by other ECUs. Dependency: if ComMNmVariant == FULL then NmPassiveModeEnabled = FALSE; NmPassiveModeEnabled shall be equal to <Bus>NmPassiveModeEnabled
Multiplicity	1
Type	EcucBooleanParamDef
Default value	–
Pre-Compile only	true

⌋

[ECUC_Nm_00258] Definition of EcucIntegerParamDef NmPncBitVectorLength ⌈

Parameter Name	NmPncBitVectorLength
Parent Container	NmChannelConfig
Upstream mapping defined	[11, CP_TPS_SystemTemplate]
Description	Parameter to configure the length of the PNC bit request information in bytes, which is transmitted within NM PDU by the corresponding <Bus>Nm. Dependency: This parameter is only valid if NmPartialNetworkSupportEnabled is set to TRUE.
Multiplicity	1
Type	EcucIntegerParamDef
Range	0 .. 63
Default value	0
Pre-Compile only	true

⌋

[ECUC_Nm_00254] Definition of EcucBooleanParamDef NmPnEnabled ⌈

Parameter Name	NmPnEnabled
Parent Container	NmChannelConfig
Upstream mapping defined	[11, CP_TPS_SystemTemplate]
Description	If this parameter is true, then this NM channel supports Partial Networking. Dependency: This parameter is only valid if NmPartialNetworkSupportEnabled is set to TRUE.
Multiplicity	0..1
Type	EcucBooleanParamDef
Default value	false
Pre-Compile only	true

⌋

[ECUC_Nm_00259] Definition of EcucBooleanParamDef NmPnEraCalcEnabled ⌈

Parameter Name	NmPnEraCalcEnabled
Parent Container	NmChannelConfig
Upstream mapping defined	[11, CP_TPS_SystemTemplate]
Description	Specifies if NmIf calculates the PN request information for external requests. (ERA) Dependency: This parameter is only valid if NmPartialNetworkSupportEnabled is set to TRUE.
Multiplicity	0..1
Type	EcucBooleanParamDef
Default value	false
Pre-Compile only	true

⌋

[ECUC_Nm_00231] Definition of EcucBooleanParamDef NmStateReportEnabled ⌈

Parameter Name	NmStateReportEnabled
Parent Container	NmChannelConfig
Upstream mapping defined	–
Description	Specifies if the NMS shall be set for the corresponding network. false: No NMS shall be set true: The NMS shall be set Dependency: only available if NmStatChangeIndEnabled and NmComUserData Support are configured to TRUE.
Multiplicity	1
Type	EcucBooleanParamDef
Default value	–
Pre-Compile only	true

⌋

[ECUC_Nm_00223] Definition of EcucBooleanParamDef NmSynchronizingNetwork ⌈

Parameter Name	NmSynchronizingNetwork
Parent Container	NmChannelConfig
Upstream mapping defined	[11, CP_TPS_SystemTemplate]
Description	If this parameter is true, then this network is a synchronizing network for the NM coordination cluster which it belongs to. The network is expected to call Nm_ SynchronizationPoint() at regular intervals. Dependency: If the parameter NmCoordClusterIndex is not defined, this parameter is not valid. Only one network can be configured as synchronizing network (Nm SynchronizingNetwork = TRUE) per coordination cluster (same NmCoordClusterIndex value per channel). NmSynchronizingNetwork can only be set to true if NmActive Coordinator is true for all networks which have the same NmCoordClusterIndex.
Multiplicity	1
Type	EcucBooleanParamDef
Default value	–
Pre-Compile only	true

⌋

[ECUC_Nm_00217] Definition of EcucReferenceDef NmComMChannelRef ⌈

Parameter Name	NmComMChannelRef
Parent Container	NmChannelConfig
Upstream mapping defined	–
Description	Reference to the corresponding ComM Channel.
Multiplicity	1
Type	Symbolic name reference to ComMChannel
Pre-Compile only	true

⌋

[ECUC_Nm_00232] Definition of EcucReferenceDef NmStateReportSignalRef ⌈

Parameter Name	NmStateReportSignalRef
Parent Container	NmChannelConfig
Upstream mapping defined	–
Description	Reference to the signal for setting the NMS by calling Com_SendSignal for the respective channel. Dependency: Signal must be configured in COM. Only available if NmStateReport Enabled == true
Multiplicity	0..1
Type	Symbolic name reference to ComSignal
Pre-Compile only	true

⌋

10.4.2 NmPnFilterMaskByte

10.4.2.1 Post-Build configurable PNC Filter Masks

To support the Dynamic PNC for CAN concept, the configuration parameter `NmPnFilterMaskByteValue` shall be configurable using the Post-Build mechanism.

The specification shall be extended so that the container NmPnFilterMaskByte and the parameter `NmPnFilterMaskByteValue` support Post-Build configuration.

The objective of this change is to allow activation or deactivation of preconfigured Partial Network Clusters after software deployment without requiring reprogramming of the complete ECU software stack. Instead, only the corresponding Post-Build configuration data is updated.

The following restrictions shall apply:

- The number of configured `NmPnFilterMaskByte` containers shall not change after deployment.
- `NmPncBitVectorLength` shall not be modified after deployment.
- `NmPncBitVectorOffset` shall not be modified after deployment.
- Only the value of `NmPnFilterMaskByteValue` may be adapted through a Post-Build update.
- Post-Build updates shall only activate or deactivate `PNC`s that were already provisioned in the original ECU configuration.
- Define the use of the `ConfigPtr` parameter of `Nm_Init` for Post-Build configurations and remove restriction that requires `ConfigPtr` to be `NULL_PTR`.

These restrictions ensure that the NM PDU structure and the `PNC Request Bit Vector` layout remain unchanged and therefore maintain interoperability with ECUs that do not support Dynamic PNC for CAN.

[ECUC_Nm_00255] Definition of EcucParamConfContainerDef NmPnFilterMaskByte ⌈

Container Name	NmPnFilterMaskByte
Parent Container	NmChannelConfig
Upstream mapping defined	[11, CP_TPS_SystemTemplate]
Description	Information for the filter of the PNC Request Bit Vector.
Multiplicity	0..63
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
NmPnFilterMaskByteIndex	1	[ECUC_Nm_00256]
NmPnFilterMaskByteValue	1	[ECUC_Nm_00257]

No Included Containers

⌋

[ECUC_Nm_00256] Definition of EcucIntegerParamDef NmPnFilterMaskByteIndex ⌈

Parameter Name	NmPnFilterMaskByteIndex	
Parent Container	NmPnFilterMaskByte	
Upstream mapping defined	[11, CP_TPS_SystemTemplate]	
Description	Index of the filter mask byte. Specifies the position within the filter mask byte array. Dependency: This parameter is only valid if NmPartialNetworkSupportEnabled is set to TRUE, NmPnFilterMaskByteIndex < NmPncBitVectorLength.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 62	
Default value	–	
Pre-Compile only	true	

⌋

[ECUC_Nm_00257] Definition of EcucIntegerParamDef NmPnFilterMaskByteValue ⌈

Parameter Name	NmPnFilterMaskByteValue	
Parent Container	NmPnFilterMaskByte	
Upstream mapping defined	[11, CP_TPS_SystemTemplate]	
Description	Parameter to configure the filter mask byte. Dependency: This parameter is only valid if NmPartialNetworkSupportEnabled is set to TRUE.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 255	

▽

△

Default value	0	
Post-Build Value	Loadable	Selectable

⌋

10.4.3 NmBusType

[ECUC_Nm_00218] Definition of EcucChoiceContainerDef NmBusType ⌈

Choice Container Name	NmBusType
Parent Container	NmChannelConfig
Upstream mapping defined	[11, CP_TPS_SystemTemplate]
Description	Choose the NmBusType
Multiplicity	1
Pre-Compile only	true

No Included Parameters

Container Choices		
Container Name	**Multiplicity**	**Description**
NmGenericBusNmConfig	0..1	Generic Bus Nm (busses not standardized by AUTOSAR)
NmStandardBusNmConfig	0..1	Standard Bus Nm busses standardized by AUTOSAR)

⌋

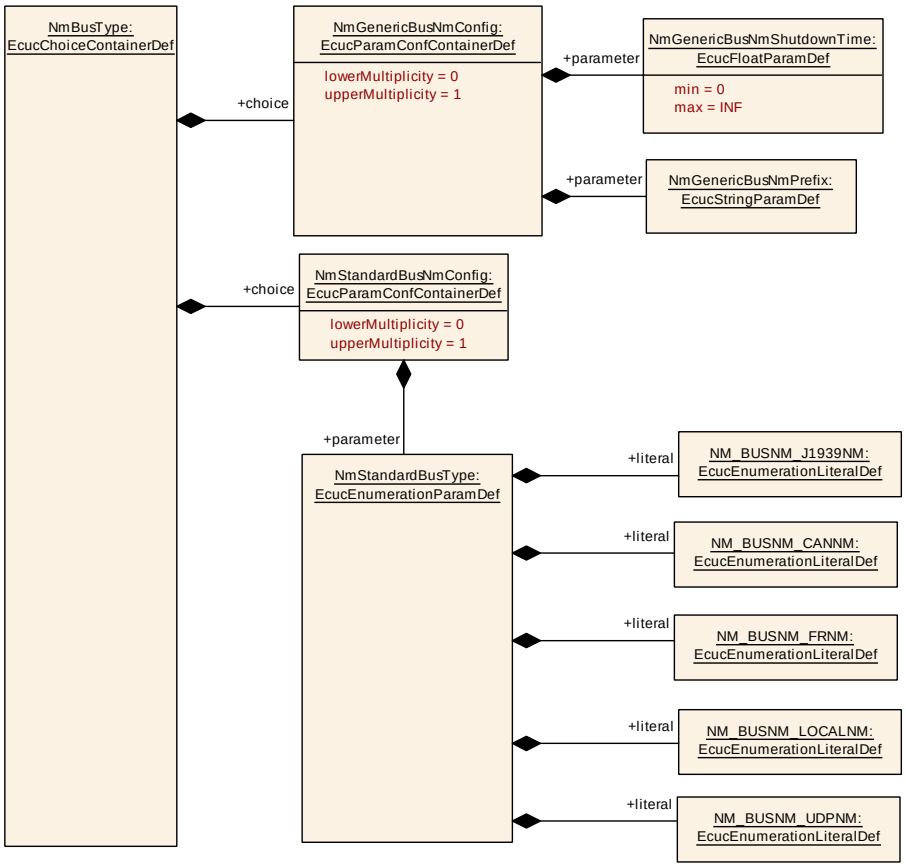

Figure 10.6: NmBusType overview

10.4.4 NmGenericBusNmConfig

[ECUC_Nm_00225] Definition of EcucParamConfContainerDef NmGenericBusNmConfig ⌈

Container Name	NmGenericBusNmConfig
Parent Container	NmBusType
Upstream mapping defined	–
Description	Generic Bus Nm (busses not standardized by AUTOSAR)
Multiplicity	0..1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
NmGenericBusNmPrefix	1	[ECUC_Nm_00219]
NmGenericBusNmShutdownTime	1	[ECUC_Nm_00239]

No Included Containers

⌋

[ECUC_Nm_00219] Definition of EcucStringParamDef NmGenericBusNmPrefix ⌈

Parameter Name	NmGenericBusNmPrefix
Parent Container	NmGenericBusNmConfig
Upstream mapping defined	–
Description	The prefix which identifies the generic <Bus>Nm. This will be used to determine the API name to be called by Nm for the provided interfaces of the <Bus>Nm. This string will used for the module prefix before the "_" character in the API call name.
Multiplicity	1
Type	EcucStringParamDef
Default value	–
Regular Expression	–
Pre-Compile only	true

⌋

[ECUC_Nm_00239] Definition of EcucFloatParamDef NmGenericBusNmShutdownTime ⌈

Parameter Name	NmGenericBusNmShutdownTime	
Parent Container	NmGenericBusNmConfig	
Upstream mapping defined	–	
Description	This parameter shall be used to calculate shutdown delay time.	
Multiplicity	1	
Type	EcucFloatParamDef	
Range	[0 .. INF]	
Default value	–	
Pre-Compile only	true	

⌋

10.4.5 NmStandardBusNmConfig

[ECUC_Nm_00226] Definition of EcucParamConfContainerDef NmStandardBusNmConfig ⌈

Container Name	NmStandardBusNmConfig
Parent Container	NmBusType
Upstream mapping defined	–
Description	Standard Bus Nm busses standardized by AUTOSAR)
Multiplicity	0..1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
NmStandardBusType	1	[ECUC_Nm_00220]

No Included Containers

⌋

[ECUC_Nm_00220] Definition of EcucEnumerationParamDef NmStandardBusType ⌈

Parameter Name	NmStandardBusType	
Parent Container	NmStandardBusNmConfig	
Upstream mapping defined	–	
Description	Identifies the bus type of the channel for standard AUTOSAR <Bus>Nms and is used to determine which set of API calls to be called by Nm for the <Bus>Nms. Note: The Ethernet bus' NM is UdpNm ! Dependency: Configuring value to NM_BUSNM_LOCALNM is only allowed if NmCoord ClusterIndex for the corresponding channel is defined (i.e channel is coordinated).	
Multiplicity	1	
Type	EcucEnumerationParamDef	
Range	NM_BUSNM_CANNM	CAN bus
	NM_BUSNM_FRNM	FlexRay bus
	NM_BUSNM_J1939NM	J1939 bus (address claiming)
	NM_BUSNM_LOCALNM	Local Bus (e.g. LIN bus)
	NM_BUSNM_UDPNM	Ethernet bus (using UDP)
Pre-Compile only	true	

⌋

10.5 Published Information

For details refer to [2] Chapter 10.3 *"Published Information"*.

A Not applicable requirements

[SWS_Nm_NA_00999] Not applicable requirements

Upstream requirements: SRS_BSW_00004, SRS_BSW_00167, SRS_BSW_00168, SRS_BSW_00170, SRS_BSW_00336, SRS_BSW_00339, SRS_BSW_00375, SRS_BSW_00380, SRS_BSW_00383, SRS_BSW_00388, SRS_BSW_00389, SRS_BSW_00390, SRS_BSW_00392, SRS_BSW_00393, SRS_BSW_00395, SRS_BSW_00396, SRS_BSW_00397, SRS_BSW_00398, SRS_BSW_00399, SRS_BSW_00400, SRS_BSW_00402, SRS_BSW_00403, SRS_BSW_00404, SRS_BSW_00406, SRS_BSW_00409, SRS_BSW_00416, SRS_BSW_00417, SRS_BSW_00422, SRS_BSW_00423, SRS_BSW_00426, SRS_BSW_00427, SRS_BSW_00428, SRS_BSW_00429, SRS_BSW_00432, SRS_BSW_00433, SRS_BSW_00437, SRS_BSW_00438, SRS_BSW_00451, SRS_BSW_00458, SRS_BSW_00467, SRS_BSW_00469, SRS_BSW_00470, SRS_BSW_00471, SRS_BSW_00472, SRS_BSW_00490, SRS_BSW_00491, SRS_ModeMgm_00049, SRS_ModeMgm_09071, SRS_ModeMgm_09078, SRS_ModeMgm_09080, SRS_ModeMgm_09081, SRS_ModeMgm_09083, SRS_ModeMgm_09084, SRS_ModeMgm_09085, SRS_ModeMgm_09087, SRS_ModeMgm_09089, SRS_ModeMgm_09090, SRS_ModeMgm_09132, SRS_ModeMgm_09133, SRS_ModeMgm_09141, SRS_ModeMgm_09149, SRS_ModeMgm_09155, SRS_ModeMgm_09156, SRS_ModeMgm_09157, SRS_ModeMgm_09168, SRS_ModeMgm_09172, SRS_ModeMgm_09207, SRS_ModeMgm_09243, SRS_ModeMgm_09244, SRS_ModeMgm_09245, SRS_ModeMgm_09246, SRS_ModeMgm_09247, SRS_ModeMgm_09249, SRS_ModeMgm_09251, SRS_ModeMgm_09256, SRS_ModeMgm_09257, SRS_ModeMgm_09259, SRS_ModeMgm_09260, SRS_ModeMgm_09261, SRS_ModeMgm_09262, SRS_ModeMgm_09263, SRS_ModeMgm_09265, SRS_ModeMgm_09266, SRS_ModeMgm_09267, SRS_ModeMgm_09268, SRS_ModeMgm_09269, SRS_ModeMgm_09278, SRS_ModeMgm_09279

⌈These requirements are not applicable to this specification.⌋

B Change history of AUTOSAR traceable items

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

B.1 Traceable item history of this document according to AUTOSAR Release R22-11

B.1.1 Added Specification Items in R22-11

Number	Heading
[SWS_Nm_00521]	
[SWS_Nm_00523]	
[SWS_Nm_00524]	
[SWS_Nm_00525]	
[SWS_Nm_00527]	
[SWS_Nm_00529]	
[SWS_Nm_00530]	
[SWS_Nm_00532]	
[SWS_Nm_00533]	
[SWS_Nm_00534]	
[SWS_Nm_00535]	
[SWS_Nm_00536]	
[SWS_Nm_00537]	
[SWS_Nm_91010]	
[SWS_Nm_91011]	
[SWS_Nm_NA_00999]	Not applicable requirements

Table B.1: Added Specification Items in R22-11

B.1.2 Changed Specification Items in R22-11

Number	Heading
[SWS_Nm_00030]	
[SWS_Nm_00031]	
[SWS_Nm_00032]	
[SWS_Nm_00033]	

▽

△

Number	Heading
[SWS_Nm_00034]	
[SWS_Nm_00035]	
[SWS_Nm_00036]	
[SWS_Nm_00037]	
[SWS_Nm_00038]	
[SWS_Nm_00039]	
[SWS_Nm_00040]	
[SWS_Nm_00042]	
[SWS_Nm_00043]	
[SWS_Nm_00044]	
[SWS_Nm_00046]	
[SWS_Nm_00112]	
[SWS_Nm_00114]	
[SWS_Nm_00117]	
[SWS_Nm_00118]	
[SWS_Nm_00119]	
[SWS_Nm_00124]	
[SWS_Nm_00154]	
[SWS_Nm_00156]	
[SWS_Nm_00159]	
[SWS_Nm_00162]	
[SWS_Nm_00166]	
[SWS_Nm_00176]	
[SWS_Nm_00183]	
[SWS_Nm_00192]	
[SWS_Nm_00193]	
[SWS_Nm_00194]	
[SWS_Nm_00230]	
[SWS_Nm_00232]	
[SWS_Nm_00234]	
[SWS_Nm_00235]	
[SWS_Nm_00236]	
[SWS_Nm_00245]	
[SWS_Nm_00250]	
[SWS_Nm_00254]	
[SWS_Nm_00259]	
[SWS_Nm_00261]	
[SWS_Nm_00267]	

▽

△

Number	Heading
[SWS_Nm_00271]	
[SWS_Nm_00272]	
[SWS_Nm_00274]	
[SWS_Nm_00275]	
[SWS_Nm_00276]	
[SWS_Nm_00282]	
[SWS_Nm_00291]	
[SWS_Nm_00293]	
[SWS_Nm_00302]	
[SWS_Nm_00305]	
[SWS_Nm_00308]	
[SWS_Nm_00310]	
[SWS_Nm_00311]	
[SWS_Nm_00312]	
[SWS_Nm_00313]	
[SWS_Nm_00317]	
[SWS_Nm_00318]	
[SWS_Nm_00319]	
[SWS_Nm_00320]	
[SWS_Nm_00321]	
[SWS_Nm_00322]	
[SWS_Nm_00323]	
[SWS_Nm_00324]	
[SWS_Nm_00325]	
[SWS_Nm_00326]	
[SWS_Nm_00327]	
[SWS_Nm_00328]	
[SWS_Nm_00329]	
[SWS_Nm_00330]	
[SWS_Nm_00331]	
[SWS_Nm_00501]	
[SWS_Nm_00502]	
[SWS_Nm_00503]	
[SWS_Nm_00504]	
[SWS_Nm_00505]	
[SWS_Nm_00508]	
[SWS_Nm_91002]	
[SWS_Nm_91003]	

▽

△

Number	Heading
[SWS_Nm_91005]	
[SWS_Nm_91006]	
[SWS_Nm_91007]	
[SWS_Nm_91008]	
[SWS_Nm_91009]	
[SWS_Nm_CONSTR_00001]	

Table B.2: Changed Specification Items in R22-11

B.1.3 Deleted Specification Items in R22-11

Number	Heading
[SWS_Nm_00314]	
[SWS_Nm_00332]	
[SWS_Nm_00506]	
[SWS_Nm_00507]	
[SWS_Nm_00999]	Not applicable requirements

Table B.3: Deleted Specification Items in R22-11

B.1.4 Added Constraints in R22-11

none

B.1.5 Changed Constraints in R22-11

none

B.1.6 Deleted Constraints in R22-11

none

B.2 Traceable item history of this document according to AUTOSAR Release R23-11

B.2.1 Added Specification Items in R23-11

Number	Heading
[SWS_Nm_00509]	Definiton of network management states in Nm module

Table B.4: Added Specification Items in R23-11

B.2.2 Changed Specification Items in R23-11

Number	Heading
[SWS_Nm_00312]	
[SWS_Nm_91003]	Definition of API function Nm_PnLearningRequest
[SWS_Nm_91005]	Definition of API function Nm_RequestSynchronizedPncShutdown

Table B.5: Changed Specification Items in R23-11

B.2.3 Deleted Specification Items in R23-11

Number	Heading
[SWS_Nm_00310]	
[SWS_Nm_00311]	

Table B.6: Deleted Specification Items in R23-11

B.2.4 Added Constraints in R23-11

none

B.2.5 Changed Constraints in R23-11

none

B.2.6 Deleted Constraints in R23-11

none

B.3 Traceable item history of this document according to AUTOSAR Release R24-11

B.3.1 Added Constraints in R24-11

none

B.3.2 Changed Constraints in R24-11

none

B.3.3 Deleted Constraints in R24-11

none

B.3.4 Added Specification Items in R24-11

none

B.3.5 Changed Specification Items in R24-11

none

B.3.6 Deleted Specification Items in R24-11

none

B.4 Traceable item history of this document according to AUTOSAR Release R25-11

B.4.1 Added Constraints in R25-11

none

B.4.2 Changed Constraints in R25-11

none

B.4.3 Deleted Constraints in R25-11

none

B.4.4 Added Specification Items in R25-11

none

B.4.5 Changed Specification Items in R25-11

Number	Heading
[SWS_Nm_00175]	
[SWS_Nm_00259]	
[SWS_Nm_00262]	

Table B.7: Changed Specification Items in R25-11

B.4.6 Deleted Specification Items in R25-11

none

B.5 Traceable item history of this document according to AUTOSAR Release R26-11

B.5.1 Added Constraints in R26-11

none

B.5.2 Changed Constraints in R26-11

none

B.5.3 Deleted Constraints in R26-11

none

B.5.4 Added Specification Items in R26-11

none

B.5.5 Changed Specification Items in R26-11

Number	Heading
[ECUC_Nm_00197]	Definition of EcucParamConfContainerDef NmChannelConfig
[ECUC_Nm_00199]	Definition of EcucParamConfContainerDef NmGlobalProperties
[ECUC_Nm_00205]	Definition of EcucFloatParamDef NmCycletimeMainFunction
[ECUC_Nm_00217]	Definition of EcucReferenceDef NmComMChannelRef
[ECUC_Nm_00219]	Definition of EcucStringParamDef NmGenericBusNmPrefix
[ECUC_Nm_00220]	Definition of EcucEnumerationParamDef NmStandardBusType
[ECUC_Nm_00221]	Definition of EcucIntegerParamDef NmCoordClusterIndex
[ECUC_Nm_00223]	Definition of EcucBooleanParamDef NmSynchronizingNetwork
[ECUC_Nm_00225]	Definition of EcucParamConfContainerDef NmGenericBusNmConfig
[ECUC_Nm_00226]	Definition of EcucParamConfContainerDef NmStandardBusNmConfig
[ECUC_Nm_00227]	Definition of EcucBooleanParamDef NmChannelSleepMaster
[ECUC_Nm_00234]	Definition of EcucFunctionNameDef NmCarWakeUpCallout
[ECUC_Nm_00235]	Definition of EcucBooleanParamDef NmCarWakeUpRxEnabled
[ECUC_Nm_00236]	Definition of EcucBooleanParamDef NmActiveCoordinator
[ECUC_Nm_00239]	Definition of EcucFloatParamDef NmGenericBusNmShutdownTime
[ECUC_Nm_00240]	Definition of EcucBooleanParamDef NmCoordinatorSyncSupport
[ECUC_Nm_00243]	Definition of EcucModuleDef Nm
[ECUC_Nm_00248]	Definition of EcucBooleanParamDef NmDynamicPncToChannelMapping Enabled
[ECUC_Nm_00249]	Definition of EcucBooleanParamDef NmSynchronizedPncShutdownEnabled
[ECUC_Nm_00250]	Definition of EcucFloatParamDef NmPnResetTime
[ECUC_Nm_00251]	Definition of EcucBooleanParamDef NmPnEiraCalcEnabled
[ECUC_Nm_00252]	Definition of EcucIntegerParamDef NmPncBitVectorOffset
[ECUC_Nm_00253]	Definition of EcucBooleanParamDef NmPartialNetworkSupportEnabled
[ECUC_Nm_00254]	Definition of EcucBooleanParamDef NmPnEnabled
[ECUC_Nm_00255]	Definition of EcucParamConfContainerDef NmPnFilterMaskByte
[ECUC_Nm_00256]	Definition of EcucIntegerParamDef NmPnFilterMaskByteIndex
[ECUC_Nm_00257]	Definition of EcucIntegerParamDef NmPnFilterMaskByteValue
[ECUC_Nm_00258]	Definition of EcucIntegerParamDef NmPncBitVectorLength
[ECUC_Nm_00259]	Definition of EcucBooleanParamDef NmPnEraCalcEnabled
[ECUC_Nm_00260]	Definition of EcucFloatParamDef NmPnShutdownMessageRetransmission Duration
[SWS_Nm_00117]	Definition of imported datatypes of module Nm

Table B.8: Changed Specification Items in R26-11

B.5.6 Deleted Specification Items in R26-11

Number	Heading
[ECUC_Nm_00204]	Definition of EcucBooleanParamDef NmVersionInfoApi
[SWS_Nm_00044]	Definition of API function Nm_GetVersionInfo
[SWS_Nm_00283]	
[SWS_Nm_00484]	

Table B.9: Deleted Specification Items in R26-11