| Document Title | Specification of Network Management Interface |
|---|---|
| **Document Owner** | AUTOSAR |
| **Document Responsibility** | AUTOSAR |
| **Document Identification No** | 228 |

| | |
|---|---|
| **Document Status** | published |
| **Part of AUTOSAR Standard** | Classic Platform |
| **Part of Standard Release** | R21-11 |

| Document Change History | | | |
|---|---|---|---|
| **Date** | **Release** | **Changed by** | **Description** |
| 2021-11-25 | R21-11 | AUTOSAR Release Management | • Reworked Partial Network functionality and Partial Network Cluster handling<br>• Caveats that were no real requirements were change to notes<br>• Editorial changes |
| 2020-11-30 | R20-11 | AUTOSAR Release Management | • Support for synchronized PNC shutdown functionality<br>• Introduction of Dynamic PNC-to-channel-mapping and PNC Learning algorithm<br>• Added limitation regarding Nm Coordinator functionality when using 10BASE-T1S in combination with PLCA media access<br>• Support for passing NM state change to SwC<br>• Editorial changes |
| 2019-11-28 | R19-11 | AUTOSAR Release Management | • Minor changes<br>• Multicore Distribution support (draft) added<br>• Changed Document Status from Final to published |

| | | | |
|---|---|---|---|
| 2018-10-31 | 4.4.0 | AUTOSAR Release Management | • Removed LinNM from the architecture<br>• Removed obsolete elements<br>• 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<br>• removed dependencies of ComMChannels to each other in respect to NMVariants<br>• 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<br>• Make the CarWakeup feature available<br>• Debugging support marked as obsolete<br>• Editorial changes |
| 2014-10-31 | 4.2.1 | AUTOSAR Release Management | • Corrections on the requirement tracing<br>• Clarification at use of callback versus callout<br>• Editorial changes |
| 2014-03-31 | 4.1.3 | AUTOSAR Release Management | • Rework of wakeup and abortion of coordinated shutdown<br>• Rework of coordination of nested sub-busses |

| | | | |
|---|---|---|---|
| 2013-10-31 | 4.1.2 | AUTOSAR Release Managment | • Remove DEM usage<br>• Correct multiplicity and dependency of configuration parameter<br>• Corrections on RemoteSleepIndication feature<br>• Corrections on MainFunction and coordinated shutdown<br>• Formal correction on REQ Tags<br>• Editorial changes<br>• Removed chapter(s) on change documentation |
| 2013-03-15 | 4.1.1 | AUTOSAR Administration | • Introduction J1939Nm<br>• Merged and corrected calculation of delay timer for Coordination Algorithm<br>• Correction of parametrization and Services for Coordinator Synchonization Algorithm<br>• Moved Nm_Passive_Mode_Enabled Parameter back to global container |
| 2011-12-22 | 4.0.3 | AUTOSAR Administration | • NmMultipleChannelsEnabled removed<br>• Added Mandatory Interfaces provided by ComM to Chapter 8.6.1<br>• move NmPassiveMode<br>• Enabled form global configuration to channel configuration<br>• Removed Nm_ReturnType<br>• Fixed some min and max values of FloatPAramDef configuration parameters<br>• Added support of NmCarWakup-Feature<br>• Added support of coordinated shutdown of nested sub-busses |
| 2009-12-18 | 4.0.1 | AUTOSAR Administration | • Release check added<br>• DET Error Code for false Pointer added<br>• ChannelID harmonized in COM-Stack<br>• Nm-State-changes in Userdata via NmIf |

| | | | |
|---|---|---|---|
| 2010-02-02 | 3.1.4 | AUTOSAR Administration | • Remove explicit support for OSEK NM from specification<br>• NM Coordinator functionality reworked (chapter 7.2 and 7.2.4)<br>• Debugging functionality added<br>• Link time configuration variant introduced<br>• 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**

# Table of Contents

# 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 PNC timer handling and synchronized PNC shutdown. If Partial Network is enabled, AUTOSAR NM aggregates all internal PNC requests (notified by ComM) and all external PNC requests (notified by the <Bus>NMs) and manages the PNC timers of each notified PNC. Additionally, if using the synchronized PNC shutdown functionality, AUTOSAR NM acts as an interface layer between ComM and <Bus>Nm's. It forwards requests from ComM to <Bus>Nm's and vice versa. PNC timer handling and synchronized PNC shutdown 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 |
| CC | Communication controller |
| ComM | Communication Manager module |
| EcuM | ECU State Manager module |
| DEM | Diagnostic Event Manager module |
| Nm | Generic Network Management Interface module, this ist the abreviation used for this module throughout this specification |
| NM | Network Management |
| OEM | Original Equipment Manufacturer |
| CBV | Control Bit Vector in NM-message |
| PNC | Partial network cluster |

| 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. |
| PN info | Represent the Partial Network information in a NM frame |
| PN info range | Represent the length of a Partial Network information in bytes |
| PNC bit | One bit which represent a particular Partial Network in the Partial Network Info Range |

| Terms: | Definition: |
|---|---|
| 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 |
| Top-level PNC coordinator | The top-level PNC coordinator is an ECU that acts as PNC gateway in the network and that handles at least one PNC as actively coordinated on all assigned channels. If synchronized PNC shutdown is enabled, the top-level PNC coordinator triggers for these PNCs the shutdown, if no other ECU in the network request them. Note that for different PNCs it is possible to have different top-level PNC coordinators. For the same PNC only one top-level coordinator is supported. |
| Intermediate PNC coordinator | An intermediate PNC coordinator is an ECU that acts as PNC gateway in the network and that handles at least one PNC as passively coordinated on at least one assigned channel. If synchronized PNC shutdown is enabled, it forwards received shutdown requests for these PNCs to the corresponding actively coordinated channels and starts their shutdown accordingly. |
| 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 request 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 and forward the PN information as PN shutdown message on the affected channels.<br><br>Note: An intermediate PNC coordinators has to forward the PN information of received PN shutdown message as fast as possible to ensure a nearly synchronized shutdown of the affected PNCs across the PN topology. |

# 3 Related documentation

## 3.1 Input documents

[1] Glossary
AUTOSAR_TR_Glossary

[2] General Specification of Basic Software Modules
AUTOSAR_SWS_BSWGeneral

[3] Specification of CAN Network Management
AUTOSAR_SWS_CANNetworkManagement

[4] Specification of FlexRay Network Management
AUTOSAR_SWS_FlexRayNetworkManagement

[5] Specification of UDP Network Management
AUTOSAR_SWS_UDPNetworkManagement

[6] Specification of Network Management for SAE J1939
AUTOSAR_SWS_SAEJ1939NetworkManagement

[7] General Requirements on Basic Software Modules
AUTOSAR_SRS_BSWGeneral

[8] Requirements on AUTOSAR Network Management
AUTOSAR_RS_NetworkManagement

[9] Specification of Communication Manager
AUTOSAR_SWS_COMManager

[10] Guide to BSW Distribution
AUTOSAR_EXP_BSWDistributionGuide

## 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.

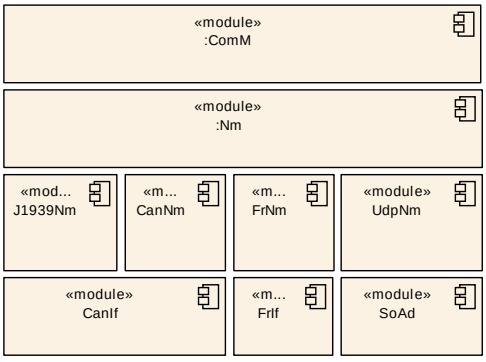


**Figure 4.1: Nm stack modules**

## 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 BusNm 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.

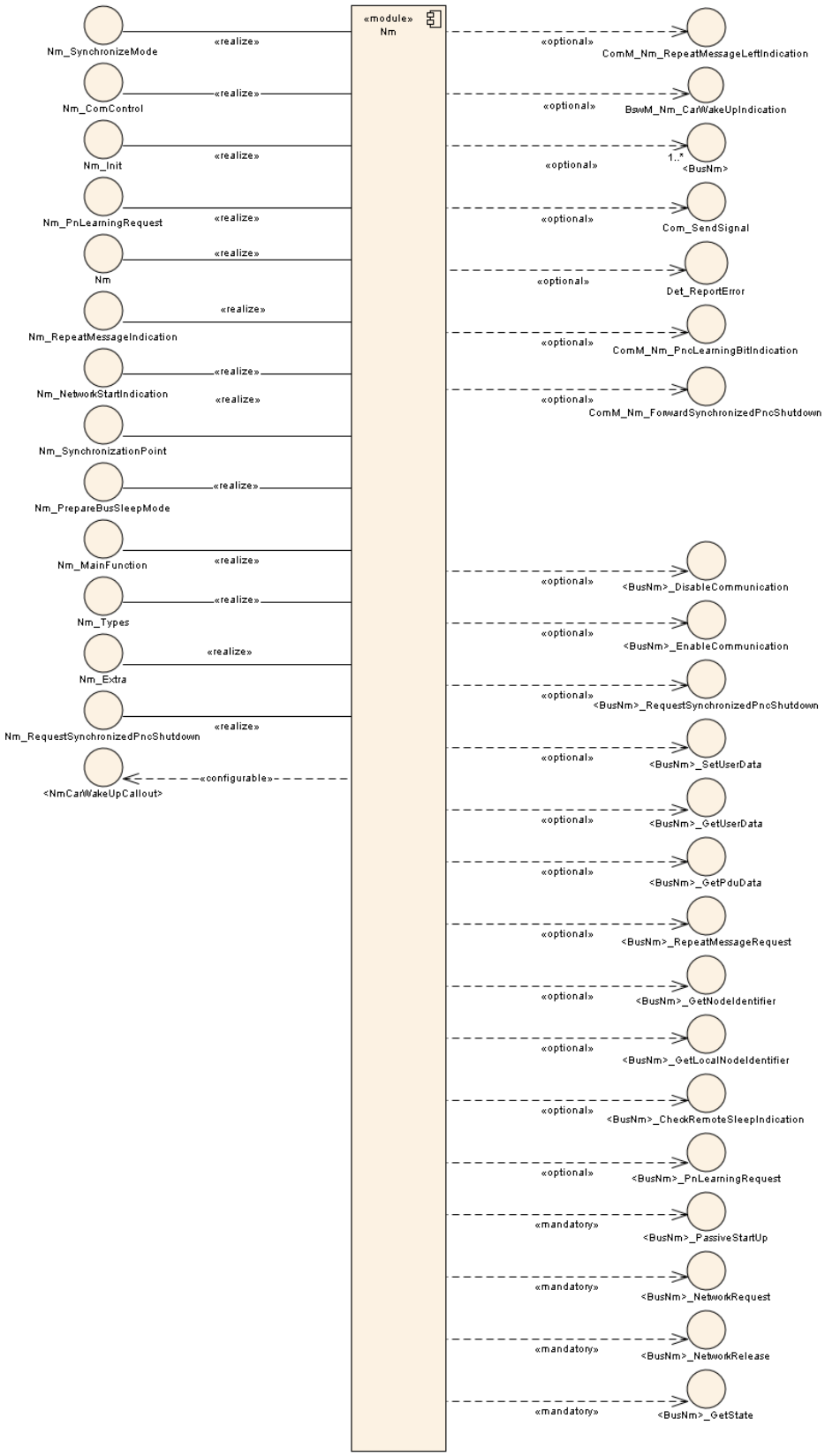


**Figure 5.1: Nm's interfaces to other modules**

### 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 (**<BusNm>**). 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 section 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 <BusNm>. The AUTOSAR bus-NMs are then explicitly supported since they implement the interfaces of <BusNm>.

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]** ⌈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:

⌋(*SRS_BSW_00159*, *SRS_BSW_00345*, *SRS_BSW_00419*)

### 5.2.2 Header file structure

**[SWS_Nm_00124]** ⌈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) and Compiler.h (for compiler specific language extensions) are 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)

⌋(*SRS_BSW_00348*, *SRS_BSW_00353*, *SRS_BSW_00357*, *SRS_BSW_00384*)

**[SWS_Nm_00243]** ⌈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.

⌋(*SRS_BSW_00171*, *SRS_BSW_00301*, *SRS_BSW_00384*)

# 6 Requirements traceability

The following tables references the requirements specified in [7] as well as [8] and links to the fulfillment of these.

| Requirement | Description | Satisfied by |
|---|---|---|
| **[RS_Nm_00043]** | NM shall not prohibit bus traffic with NM not being initialized | [SWS_Nm_00999] |
| **[RS_Nm_00044]** | The NM shall be applicable to different types of communication systems which are in the scope of Autosar and support a bus sleep mode. | [SWS_Nm_00051] [SWS_Nm_00172] [SWS_Nm_00274] [SWS_Nm_00276] |
| **[RS_Nm_00045]** | NM shall provide services to coordinate shutdown of NM-clusters independently of each other | [SWS_Nm_00167] [SWS_Nm_00168] |
| **[RS_Nm_00046]** | It shall be possible to trigger the startup of all Nodes at any Point in Time | [SWS_Nm_00031] [SWS_Nm_00032] |
| **[RS_Nm_00047]** | NM shall provide a service to request to keep the bus awake and a service to cancel this request. | [SWS_Nm_00002] [SWS_Nm_00003] [SWS_Nm_00032] [SWS_Nm_00046] [SWS_Nm_00171] |
| **[RS_Nm_00048]** | NM shall put the communication controller into sleep mode if there is no bus communication | [SWS_Nm_00046] |
| **[RS_Nm_00050]** | The NM shall provide the current state of NM | [SWS_Nm_00043] [SWS_Nm_00114] [SWS_Nm_00275] |
| **[RS_Nm_00051]** | NM shall inform application when NM state changes occur. | [SWS_Nm_00031] [SWS_Nm_00032] [SWS_Nm_00046] [SWS_Nm_00114] [SWS_Nm_00156] [SWS_Nm_00158] [SWS_Nm_00159] [SWS_Nm_00161] [SWS_Nm_00162] [SWS_Nm_00163] [SWS_Nm_00230] [SWS_Nm_00249] [SWS_Nm_00487] |
| **[RS_Nm_00052]** | The NM interface shall signal to the application that all other ECUs are ready to sleep. | [SWS_Nm_00192] |

| Requirement | Description | Satisfied by |
|---|---|---|
| **[RS_Nm_00054]** | There shall be a deterministic time from the point where all nodes agree to go to bus sleep to the point where bus is switched off. | [SWS_Nm_00171] |
| **[RS_Nm_00137]** | NM shall perform communication system error handling for errors that have impact on the NM behavior. | [SWS_Nm_00999] |
| **[RS_Nm_00142]** | NM shall provide a mechanism to limit its bus load. | [SWS_Nm_00999] |
| **[RS_Nm_00144]** | NM shall support communication clusters of up to 64 ECUs | [SWS_Nm_00999] |
| **[RS_Nm_00145]** | On a properly configured node, NM shall tolerate a loss of a predefined number of NM messages | [SWS_Nm_00999] |
| **[RS_Nm_00146]** | The NM shall tolerate a time jitter of NM messages in one or more ECUs | [SWS_Nm_00999] |
| **[RS_Nm_00149]** | The timing of NM shall be configurable. | [SWS_Nm_00175] [SWS_Nm_00281] [SWS_Nm_00284] |
| **[RS_Nm_00150]** | Specific features of the Network Management shall be configurable | [SWS_Nm_00055] [SWS_Nm_00134] [SWS_Nm_00136] [SWS_Nm_00138] [SWS_Nm_00140] [SWS_Nm_00150] [SWS_Nm_00164] [SWS_Nm_00165] [SWS_Nm_00166] [SWS_Nm_00241] [SWS_Nm_00251] [SWS_Nm_00255] [SWS_Nm_00273] [SWS_Nm_00277] [SWS_Nm_00278] [SWS_Nm_00279] [SWS_Nm_00290] |
| **[RS_Nm_00151]** | The Network Management algorithm shall allow any node to integrate into an already running NM cluster | [SWS_Nm_00031] [SWS_Nm_00032] |

| Requirement | Description | Satisfied by |
|---|---|---|
| **[RS_Nm_00152]** | The specification and implementation shall be split-up into a communication system independent and communication system dependent parts. | [SWS_Nm_00999] |
| **[RS_Nm_00153]** | The Network Management shall optionally provide a possibility to detect present nodes | [SWS_Nm_00038] [SWS_Nm_00230] |
| **[RS_Nm_00154]** | The Network Management API shall be independent from the communication bus | [SWS_Nm_00006] [SWS_Nm_00012] [SWS_Nm_00276] |
| **[RS_Nm_02503]** | The NM API shall optionally give the possibility to send user data | [SWS_Nm_00035] [SWS_Nm_00250] [SWS_Nm_00252] [SWS_Nm_00285] |
| **[RS_Nm_02504]** | The NM API shall optionally give the possibility to get user data | [SWS_Nm_00036] [SWS_Nm_00291] |
| **[RS_Nm_02506]** | The NM API shall give the possibility to read the source node identifier of the sender | [SWS_Nm_00039] |
| **[RS_Nm_02508]** | Every node shall have a node identifier associated with it that is unique in the NM-cluster. | [SWS_Nm_00040] |
| **[RS_Nm_02509]** | The NM interface shall signal to the application that at least one ECU is not ready to sleep anymore. | [SWS_Nm_00193] |
| **[RS_Nm_02511]** | It shall be possible to configure the Network Management of a node so that it does not contribute to the cluster shutdown decision. | [SWS_Nm_00168] [SWS_Nm_00228] |

| Requirement | Description | Satisfied by |
|---|---|---|
| **[RS_Nm_02512]** | The NM shall give the possibility to enable or disable the network management related communication configured for an active NM node | [SWS_Nm_00033] [SWS_Nm_00034] |
| **[RS_Nm_02513]** | NM shall provide functionality which enables upper layers to control the sleep mode. | [SWS_Nm_00006] [SWS_Nm_00012] [SWS_Nm_00031] [SWS_Nm_00032] [SWS_Nm_00033] [SWS_Nm_00042] [SWS_Nm_00154] [SWS_Nm_00155] |
| **[RS_Nm_02514]** | It shall be possible to group networks into *NM Coordination Clusters* | [SWS_Nm_00001] [SWS_Nm_00168] |
| **[RS_Nm_02515]** | NM shall offer a generic possibility to run other NMs than the AUTOSAR-NMs | [SWS_Nm_00051] [SWS_Nm_00119] [SWS_Nm_00166] [SWS_Nm_00276] |
| **[RS_Nm_02516]** | All AUTOSAR NM instances shall support the NM Coordinator functionality including Bus synchronization on demand | [SWS_Nm_00169] [SWS_Nm_00171] [SWS_Nm_00173] [SWS_Nm_00174] [SWS_Nm_00175] [SWS_Nm_00176] [SWS_Nm_00177] [SWS_Nm_00194] [SWS_Nm_00284] |
| **[RS_Nm_02517]** | CanNm shall support Partial Networking on CAN | [SWS_Nm_00302] [SWS_Nm_00308] [SWS_Nm_00310] [SWS_Nm_00311] [SWS_Nm_00312] [SWS_Nm_00313] [SWS_Nm_00314] [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_00999] [SWS_Nm_CONSTR_00001] |
| **[RS_Nm_02519]** | The NM Control Bit Vector shall contain a PNI (Partial Network Information) bit. | [SWS_Nm_00999] |
| **[RS_Nm_02527]** | Nm shall implement a filter algorithm dropping all NM messages that are not relevant for the ECU | [SWS_Nm_00308] [SWS_Nm_00311] [SWS_Nm_00312] [SWS_Nm_00999] [SWS_Nm_CONSTR_00001] |

| Requirement | Description | Satisfied by |
|---|---|---|
| **[RS_Nm_02528]** | Nm shall provide a service which allows for instantaneous sending of NM messages. | [SWS_Nm_00999] |
| **[RS_Nm_02535]** | The NM coordination shall support the coordination of nested sub-buses | [SWS_Nm_00254] [SWS_Nm_00256] [SWS_Nm_00257] [SWS_Nm_00259] [SWS_Nm_00261] [SWS_Nm_00262] [SWS_Nm_00267] [SWS_Nm_00271] [SWS_Nm_00272] [SWS_Nm_00280] |
| **[RS_Nm_02536]** | NM shall provide functionality to start-up without requesting the network. | [SWS_Nm_00031] [SWS_Nm_00119] [SWS_Nm_00245] [SWS_Nm_00250] |
| **[RS_Nm_02537]** | The NM Coordinator shall be able to abort the coordinated shutdown | [SWS_Nm_00181] [SWS_Nm_00182] [SWS_Nm_00183] [SWS_Nm_00185] [SWS_Nm_00235] [SWS_Nm_00236] [SWS_Nm_00267] |
| **[RS_Nm_02543]** | NM shall forward requests for synchronized PNC shutdown | [SWS_Nm_00506] [SWS_Nm_91005] |
| **[RS_Nm_02544]** | NM shall forward the indication of a PN shutdown message | [SWS_Nm_00507] [SWS_Nm_91006] [SWS_Nm_91007] [SWS_Nm_91008] [SWS_Nm_91009] |
| **[RS_Nm_02545]** | <Bus>NM shall handle requests for synchronized PNC shutdown | [SWS_Nm_91005] |
| **[RS_Nm_02548]** | <Bus>Nm shall be able to propagate and evaluate the need for synchronized PNC shutdown in the role of a top-level PNC coordinator or intermediate PNC coordinator (optional) | [SWS_Nm_00305] |
| **[RS_Nm_02550]** | FrNm shall support Partial Networking on FlexRay | [SWS_Nm_00999] |
| **[RS_Nm_02562]** | Nm shall support channel-specific storage of IRA | [SWS_Nm_00330] |
| **[RS_Nm_02563]** | Nm shall calculate the combined partial network request status EIRA | [SWS_Nm_00302] [SWS_Nm_00313] [SWS_Nm_00314] [SWS_Nm_00317] [SWS_Nm_00318] [SWS_Nm_00319] [SWS_Nm_00320] |
| **[RS_Nm_02564]** | Nm shall calculate the status of the external partial network requests ERA | [SWS_Nm_00322] [SWS_Nm_00323] [SWS_Nm_00324] [SWS_Nm_00325] [SWS_Nm_00326] [SWS_Nm_00328] [SWS_Nm_00329] |

| Requirement | Description | Satisfied by |
|---|---|---|
| **[RS_Nm_02565]** | <Bus>Nm shall communicate EIRA and ERA requests to the upper layers using dedicated APIs | [SWS_Nm_00321] [SWS_Nm_00327] |
| **[SRS_BSW_00003]** | All software modules shall provide version and identification information | [SWS_Nm_00044] |
| **[SRS_BSW_00004]** | All Basic SW Modules shall perform a pre-processor check of the versions of all imported include files | [SWS_Nm_00999] |
| **[SRS_BSW_00005]** | Modules of the $\mu$C Abstraction Layer (MCAL) may not have hard coded horizontal interfaces | [SWS_Nm_00999] |
| **[SRS_BSW_00006]** | The source code of software modules above the $\mu$C Abstraction Layer (MCAL) shall not be processor and compiler dependent. | [SWS_Nm_00999] |
| **[SRS_BSW_00007]** | All Basic SW Modules written in C language shall conform to the MISRA C 2012 Standard. | [SWS_Nm_00999] |
| **[SRS_BSW_00009]** | All Basic SW Modules shall be documented according to a common standard. | [SWS_Nm_00999] |
| **[SRS_BSW_00010]** | The memory consumption of all Basic SW Modules shall be documented for a defined configuration for all supported platforms. | [SWS_Nm_00999] |
| **[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] |

| Requirement | Description | Satisfied by |
|---|---|---|
| **[SRS_BSW_00160]** | Configuration files of AUTOSAR Basic SW module shall be readable for human beings | [SWS_Nm_00999] |
| **[SRS_BSW_00161]** | The AUTOSAR Basic Software shall provide a microcontroller abstraction layer which provides a standardized interface to higher software layers | [SWS_Nm_00999] |
| **[SRS_BSW_00162]** | The AUTOSAR Basic Software shall provide a hardware abstraction layer | [SWS_Nm_00999] |
| **[SRS_BSW_00164]** | The Implementation of interrupt service routines shall be done by the Operating System, complex drivers or modules | [SWS_Nm_00999] |
| **[SRS_BSW_00167]** | All AUTOSAR Basic Software Modules shall provide configuration rules and constraints to enable plausibility checks | [SWS_Nm_00999] |
| **[SRS_BSW_00168]** | SW components shall be tested by a function defined in a common API in the Basis-SW | [SWS_Nm_00999] |
| **[SRS_BSW_00170]** | The AUTOSAR SW Components shall provide information about their dependency from faults, signal qualities, driver demands | [SWS_Nm_00999] |
| **[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] |

| Requirement | Description | Satisfied by |
|---|---|---|
| **[SRS_BSW_00172]** | The scheduling strategy that is built inside the Basic Software Modules shall be compatible with the strategy used in the system | [SWS_Nm_00999] |
| **[SRS_BSW_00300]** | All AUTOSAR Basic Software Modules shall be identified by an unambiguous name | [SWS_Nm_00999] |
| **[SRS_BSW_00301]** | All AUTOSAR Basic Software Modules shall only import the necessary information | [SWS_Nm_00117] [SWS_Nm_00243] |
| **[SRS_BSW_00302]** | All AUTOSAR Basic Software Modules shall only export information needed by other modules | [SWS_Nm_00999] |
| **[SRS_BSW_00304]** | All AUTOSAR Basic Software Modules shall use only AUTOSAR data types instead of native C data types | [SWS_Nm_00999] |
| **[SRS_BSW_00305]** | Data types naming convention | [SWS_Nm_00999] |
| **[SRS_BSW_00306]** | AUTOSAR Basic Software Modules shall be compiler and platform independent | [SWS_Nm_00999] |
| **[SRS_BSW_00307]** | Global variables naming convention | [SWS_Nm_00999] |
| **[SRS_BSW_00308]** | AUTOSAR Basic Software Modules shall not define global data in their header files, but in the C file | [SWS_Nm_00999] |
| **[SRS_BSW_00309]** | All AUTOSAR Basic Software Modules shall indicate all global data with read-only purposes by explicitly assigning the const keyword | [SWS_Nm_00999] |
| **[SRS_BSW_00310]** | API naming convention | [SWS_Nm_00999] |
| **[SRS_BSW_00312]** | Shared code shall be reentrant | [SWS_Nm_00999] |

| Requirement | Description | Satisfied by |
|---|---|---|
| **[SRS_BSW_00314]** | All internal driver modules shall separate the interrupt frame definition from the service routine | [SWS_Nm_00999] |
| **[SRS_BSW_00318]** | Each AUTOSAR Basic Software Module file shall provide version numbers in the header file | [SWS_Nm_00999] |
| **[SRS_BSW_00321]** | The version numbers of AUTOSAR Basic Software Modules shall be enumerated according specific rules | [SWS_Nm_00999] |
| **[SRS_BSW_00323]** | All AUTOSAR Basic Software Modules shall check passed API parameters for validity | [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_00505] [SWS_Nm_00508] |
| **[SRS_BSW_00325]** | The runtime of interrupt service routines and functions that are running in interrupt context shall be kept short | [SWS_Nm_00999] |
| **[SRS_BSW_00327]** | Error values naming convention | [SWS_Nm_00232] |
| **[SRS_BSW_00328]** | All AUTOSAR Basic Software Modules shall avoid the duplication of code | [SWS_Nm_00999] |
| **[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_00331]** | All Basic Software Modules shall strictly separate error and status information | [SWS_Nm_00999] |
| **[SRS_BSW_00333]** | For each callback function it shall be specified if it is called from interrupt context or not | [SWS_Nm_00028] |

| Requirement | Description | Satisfied by |
|---|---|---|
| **[SRS_BSW_00334]** | All Basic Software Modules shall provide an XML file that contains the meta data | [SWS_Nm_00999] |
| **[SRS_BSW_00335]** | Status values naming convention | [SWS_Nm_00999] |
| **[SRS_BSW_00336]** | Basic SW module shall be able to shutdown | [SWS_Nm_00999] |
| **[SRS_BSW_00337]** | Classification of development errors | [SWS_Nm_00232] |
| **[SRS_BSW_00339]** | Reporting of production relevant error status | [SWS_Nm_00999] |
| **[SRS_BSW_00341]** | Module documentation shall contains all needed informations | [SWS_Nm_00999] |
| **[SRS_BSW_00342]** | It shall be possible to create an AUTOSAR ECU out of modules provided as source code and modules provided as object code, even mixed | [SWS_Nm_00999] |
| **[SRS_BSW_00343]** | The unit of time for specification and configuration of Basic SW modules shall be preferably in physical time unit | [SWS_Nm_00999] |
| **[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_00346]** | All AUTOSAR Basic Software Modules shall provide at least a basic set of module files | [SWS_Nm_00999] |
| **[SRS_BSW_00347]** | A Naming seperation of different instances of BSW drivers shall be in place | [SWS_Nm_00999] |
| **[SRS_BSW_00348]** | All AUTOSAR standard types and constants shall be placed and organized in a standard type header file | [SWS_Nm_00124] |

| Requirement | Description | Satisfied by |
|---|---|---|
| **[SRS_BSW_00350]** | All AUTOSAR Basic Software Modules shall allow the enabling/disabling of detection and reporting of development errors. | [SWS_Nm_00999] |
| **[SRS_BSW_00351]** | Encapsulation of compiler specific methods to map objects | [SWS_Nm_00999] |
| **[SRS_BSW_00353]** | All integer 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] |
| **[SRS_BSW_00359]** | All AUTOSAR Basic Software Modules callback functions shall avoid return types other than void if possible | [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_00360]** | AUTOSAR Basic Software Modules callback functions are allowed to have parameters | [SWS_Nm_00999] |
| **[SRS_BSW_00361]** | All mappings of not standardized keywords of compiler specific scope shall be placed and organized in a compiler specific type and keyword header | [SWS_Nm_00999] |

| Requirement | Description | Satisfied by |
|---|---|---|
| **[SRS_BSW_00369]** | All AUTOSAR Basic Software Modules shall not return specific development error codes via the API | [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_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_00374]** | All Basic Software Modules shall provide a readable module vendor identification | [SWS_Nm_00999] |
| **[SRS_BSW_00375]** | Basic Software Modules shall report wake-up reasons | [SWS_Nm_00999] |
| **[SRS_BSW_00377]** | A Basic Software Module can return a module specific types | [SWS_Nm_00999] |
| **[SRS_BSW_00378]** | AUTOSAR shall provide a boolean type | [SWS_Nm_00999] |
| **[SRS_BSW_00379]** | All software modules shall provide a module identifier in the header file and in the module XML description file. | [SWS_Nm_00999] |
| **[SRS_BSW_00380]** | Configuration parameters being stored in memory shall be placed into separate c-files | [SWS_Nm_00999] |
| **[SRS_BSW_00383]** | The Basic Software Module specifications shall specify which other configuration files from other modules they use at least in the description | [SWS_Nm_00999] |
| **[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] |

| Requirement | Description | Satisfied by |
|---|---|---|
| **[SRS_BSW_00386]** | The BSW shall specify the configuration for detecting an error | [SWS_Nm_00232] [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_00505] [SWS_Nm_00508] |
| **[SRS_BSW_00388]** | Containers shall be used to group configuration parameters that are defined for the same object | [SWS_Nm_00999] |
| **[SRS_BSW_00389]** | Containers shall have names | [SWS_Nm_00999] |
| **[SRS_BSW_00390]** | Parameter content shall be unique within the module | [SWS_Nm_00999] |
| **[SRS_BSW_00392]** | Parameters shall have a type | [SWS_Nm_00999] |
| **[SRS_BSW_00393]** | Parameters shall have a range | [SWS_Nm_00999] |
| **[SRS_BSW_00394]** | The Basic Software Module specifications shall specify the scope of the configuration parameters | [SWS_Nm_00999] |
| **[SRS_BSW_00395]** | The Basic Software Module specifications shall list all configuration parameter dependencies | [SWS_Nm_00999] |
| **[SRS_BSW_00396]** | The Basic Software Module specifications shall specify the supported configuration classes for changing values and multiplicities for each parameter/ container | [SWS_Nm_00999] |
| **[SRS_BSW_00397]** | The configuration parameters in pre-compile time are fixed before compilation starts | [SWS_Nm_00999] |

| Requirement | Description | Satisfied by |
|---|---|---|
| **[SRS_BSW_00398]** | The link-time configuration is achieved on object code basis in the stage after compiling and before linking | [SWS_Nm_00999] |
| **[SRS_BSW_00399]** | Parameter-sets shall be located in a separate segment and shall be loaded after the code | [SWS_Nm_00999] |
| **[SRS_BSW_00400]** | Parameter shall be selected from multiple sets of parameters after code has been loaded and started | [SWS_Nm_00999] |
| **[SRS_BSW_00401]** | Documentation of multiple instances of configuration parameters shall be available | [SWS_Nm_00999] |
| **[SRS_BSW_00402]** | Each module shall provide version information | [SWS_Nm_00999] |
| **[SRS_BSW_00403]** | The Basic Software Module specifications shall specify for each parameter/container whether it supports different values or multiplicity in different configuration sets | [SWS_Nm_00999] |
| **[SRS_BSW_00404]** | BSW Modules shall support post-build configuration | [SWS_Nm_00999] |
| **[SRS_BSW_00405]** | BSW Modules shall support multiple configuration sets | [SWS_Nm_00030] |
| **[SRS_BSW_00406]** | A static status variable denoting if a BSW module is initialized shall be initialized with value 0 before any APIs of the BSW module is called | [SWS_Nm_00999] |
| **[SRS_BSW_00407]** | Each BSW module shall provide a function to read out the version information of a dedicated module implementation | [SWS_Nm_00044] |

| Requirement | Description | Satisfied by |
|---|---|---|
| **[SRS_BSW_00408]** | All AUTOSAR Basic Software Modules configuration parameters shall be named according to a specific naming rule | [SWS_Nm_00999] |
| **[SRS_BSW_00409]** | All production code error ID symbols are defined by the Dem module and shall be retrieved by the other BSW modules from Dem configuration | [SWS_Nm_00999] |
| **[SRS_BSW_00410]** | Compiler switches shall have defined values | [SWS_Nm_00999] |
| **[SRS_BSW_00411]** | All AUTOSAR Basic Software Modules shall apply a naming rule for enabling/ disabling the existence of the API | [SWS_Nm_00999] |
| **[SRS_BSW_00413]** | An index-based accessing of the instances of BSW modules shall be done | [SWS_Nm_00999] |
| **[SRS_BSW_00414]** | Init functions shall have a pointer to a configuration structure as single parameter | [SWS_Nm_00030] [SWS_Nm_00282] [SWS_Nm_00283] |
| **[SRS_BSW_00415]** | Interfaces which are provided exclusively for one module shall be separated into a dedicated header file | [SWS_Nm_00999] |
| **[SRS_BSW_00416]** | The sequence of modules to be initialized shall be configurable | [SWS_Nm_00999] |
| **[SRS_BSW_00417]** | Software which is not part of the SW-C shall report error events only after the DEM is fully operational. | [SWS_Nm_00999] |
| **[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] |

| Requirement | Description | Satisfied by |
|---|---|---|
| **[SRS_BSW_00422]** | Pre-de-bouncing of error status information is done within the DEM | [SWS_Nm_00999] |
| **[SRS_BSW_00423]** | BSW modules with AUTOSAR interfaces shall be describable with the means of the SW-C Template | [SWS_Nm_00999] |
| **[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_00426]** | BSW Modules shall ensure data consistency of data which is shared between BSW modules | [SWS_Nm_00999] |
| **[SRS_BSW_00427]** | ISR functions shall be defined and documented in the BSW module description template | [SWS_Nm_00999] |
| **[SRS_BSW_00428]** | A BSW module shall state if its main processing function(s) has to be executed in a specific order or sequence | [SWS_Nm_00999] |
| **[SRS_BSW_00429]** | Access to OS is restricted | [SWS_Nm_00999] |
| **[SRS_BSW_00432]** | Modules should have separate main processing functions for read/receive and write/transmit data path | [SWS_Nm_00999] |
| **[SRS_BSW_00433]** | Main processing functions are only allowed to be called from task bodies provided by the BSW Scheduler | [SWS_Nm_00999] |

| Requirement | Description | Satisfied by |
|---|---|---|
| **[SRS_BSW_00437]** | Memory mapping shall provide the possibility to define RAM segments which are not to be initialized during startup | [SWS_Nm_00999] |
| **[SRS_BSW_00438]** | Configuration data shall be defined in a structure | [SWS_Nm_00999] |
| **[SRS_BSW_00439]** | Enable BSW modules to handle interrupts | [SWS_Nm_00999] |
| **[SRS_BSW_00440]** | The callback function invocation by the BSW module shall follow the signature provided by RTE to invoke servers via Rte_Call API | [SWS_Nm_00999] |
| **[SRS_BSW_00441]** | Naming convention for type, macro and function | [SWS_Nm_00999] |
| **[SRS_BSW_00447]** | Standardizing Include file structure of BSW Modules Implementing Autosar Service | [SWS_Nm_00999] |
| **[SRS_BSW_00448]** | Module SWS shall not contain requirements from Other Modules | [SWS_Nm_00999] |
| **[SRS_BSW_00449]** | BSW Service APIs used by Autosar Application Software shall return a Std_ ReturnType | [SWS_Nm_00999] |
| **[SRS_BSW_00450]** | A Main function of a un-initialized module shall return immediately | [SWS_Nm_00121] |
| **[SRS_BSW_00451]** | Hardware registers shall be protected if concurrent access to these registers occur | [SWS_Nm_00999] |
| **[SRS_BSW_00452]** | Classification of runtime errors | [SWS_Nm_00999] |
| **[SRS_BSW_00453]** | BSW Modules shall be harmonized | [SWS_Nm_00999] |
| **[SRS_BSW_00454]** | An alternative interface without a parameter of category DATA_REFERENCE shall be available. | [SWS_Nm_00999] |

| Requirement | Description | Satisfied by |
|---|---|---|
| **[SRS_BSW_00456]** | A Header file shall be defined in order to harmonize BSW Modules | [SWS_Nm_00999] |
| **[SRS_BSW_00457]** | Callback functions of Application software components shall be invoked by the Basis SW | [SWS_Nm_00999] |
| **[SRS_BSW_00458]** | Classification of production errors | [SWS_Nm_00999] |
| **[SRS_BSW_00459]** | It shall be possible to concurrently execute a service offered by a BSW module in different partitions | [SWS_Nm_00484] [SWS_Nm_00485] [SWS_Nm_00486] |
| **[SRS_BSW_00460]** | Reentrancy Levels | [SWS_Nm_00999] |
| **[SRS_BSW_00461]** | Modules called by generic modules shall satisfy all interfaces requested by the generic module | [SWS_Nm_00999] |
| **[SRS_BSW_00462]** | All Standardized Autosar Interfaces shall have unique requirement Id / number | [SWS_Nm_00999] |
| **[SRS_BSW_00463]** | Naming convention of callout prototypes | [SWS_Nm_00999] |
| **[SRS_BSW_00464]** | File names shall be considered case sensitive regardless of the filesystem in which they are used | [SWS_Nm_00999] |
| **[SRS_BSW_00465]** | It shall not be allowed to name any two files so that they only differ by the cases of their letters | [SWS_Nm_00999] |
| **[SRS_BSW_00466]** | Classification of extended production errors | [SWS_Nm_00999] |
| **[SRS_BSW_00467]** | The init / deinit services shall only be called by BswM or EcuM | [SWS_Nm_00999] |
| **[SRS_BSW_00469]** | Fault detection and healing of production errors and extended production errors | [SWS_Nm_00999] |
| **[SRS_BSW_00470]** | Execution frequency of production error detection | [SWS_Nm_00999] |

| Requirement | Description | Satisfied by |
|---|---|---|
| **[SRS_BSW_00471]** | Do not cause dead-locks on detection of production errors - the ability to heal from previously detected production errors | [SWS_Nm_00999] |
| **[SRS_BSW_00472]** | Avoid detection of two production errors with the same root cause. | [SWS_Nm_00999] |
| **[SRS_BSW_00473]** | Classification of transient faults | [SWS_Nm_00999] |
| **[SRS_BSW_00477]** | The functional interfaces of AUTOSAR BSW modules shall be specified in C99 | [SWS_Nm_00999] |
| **[SRS_BSW_00478]** | Timing limits of main functions | [SWS_Nm_00292] |
| **[SRS_BSW_00479]** | Interfaces for handling request from external devices | [SWS_Nm_00999] |
| **[SRS_BSW_00480]** | NullPointer Errors shall follow a naming rule | [SWS_Nm_00999] |
| **[SRS_BSW_00481]** | Invalid configuration set selection errors shall follow a naming rule | [SWS_Nm_00999] |
| **[SRS_BSW_00482]** | Get Version Informationfunction shall follow a naming rule | [SWS_Nm_00044] |
| **[SRS_ModeMgm_-09250]** | PNC activation requests shall be exchanged with the Network Management via a PNC bit vector | [SWS_Nm_00317] [SWS_Nm_00321] [SWS_Nm_00327] [SWS_Nm_91006] [SWS_Nm_91008] |

# 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 PNC network 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.⌋ *(RS_Nm_00154, RS_Nm_02513)*

**[SWS_Nm_00012]** ⌈The Nm shall convert callback functions called by the bus specific NM layers to generic callbacks to the ComM.⌋ *(RS_Nm_00154, RS_Nm_02513)*

**[SWS_Nm_00091]** ⌈The Base functionality of Nm may be implemented completely or partly using macros.⌋ *(SRS_BSW_00330)*

## 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]** ⌈If the *NM Coordinator functionality* is configured, the configuration parameter NmCycletimeMainFunction shall be configured (see [SWS_Nm_00118]).⌋*(SRS_BSW_00478)*

**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.⌋*(RS_Nm_02514)*

**[SWS_Nm_00256]** ⌈The NM-Coordinator shall support two or more NM-Coordinators connected to the same NM Cluster.⌋*(RS_Nm_02535)*

**[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.⌋*(RS_Nm_00044, RS_Nm_02515)*

**Note:** The required functionality, callbacks and interfaces are specified in subsection 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.⌋*(RS_Nm_00150)*

**[SWS_Nm_00167]** ⌈It shall be possible to configure multiple NM coordination clusters that shall be coordinated independently.⌋*(RS_Nm_00045)*

**[SWS_Nm_00168]** ⌈Each bus shall belong to zero or one NM coordination cluster.⌋*(RS_Nm_00045, RS_Nm_02511, RS_Nm_02514)*

**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.⌋ *(RS_Nm_02516)*

**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.⌋ *(RS_Nm_00047)*

**[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.⌋ *(RS_Nm_00047)*

**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 <BusNm>_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.⌋*(RS_Nm_02511)*

**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 subsection 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.

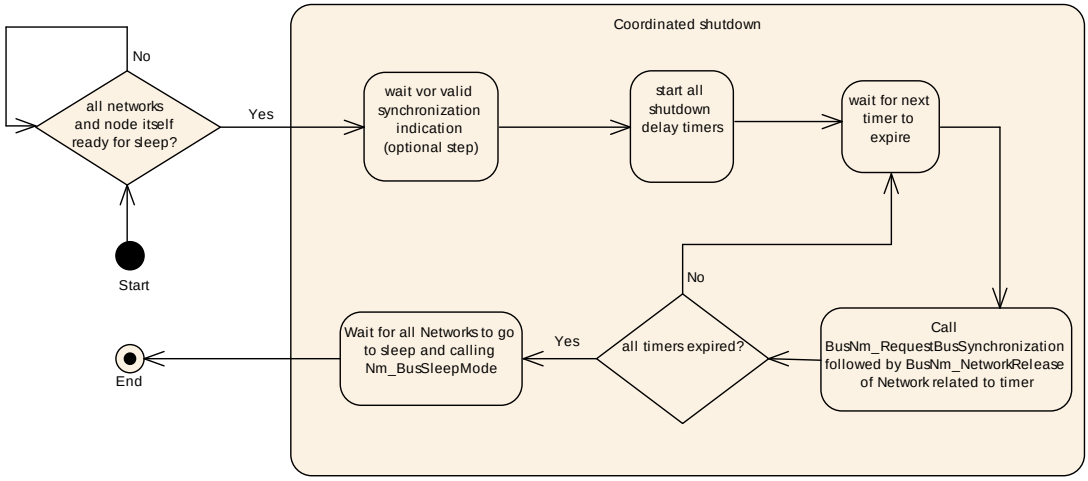


**Figure 7.1: Overview of the coordination algorithm with the coordinated shutdown as part of it**

**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 ( NmMainFunction ) 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.⌋*(RS_Nm_00047, RS_Nm_02516, RS_Nm_00054)*

**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.⌋*(RS_Nm_00044)*

**[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_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 <BusNm>_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.⌋*(RS_Nm_02516)*

**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]).⌋*(RS_Nm_02516)*

**[SWS_Nm_00175]** ⌈When the coordinated shutdown is started, a shutdown delay timer shall be activated for each currently awake 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.⌋*(RS_Nm_00149, RS_Nm_02516)*

**[SWS_Nm_00284]** ⌈If the NmGlobalCoordinatorTime is zero the shutdown delay timer of all channels shall also be zero.⌋*(RS_Nm_00149, RS_Nm_02516)*

**Note:** The **TSHUTDOWN_CHANNEL** can be calculated as described in subsection 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 BusNmType is not set to NM_BUSNM_LOCALNM release the network by calling the <BusNm>_RequestBusSynchronization followed by <BusNm>_NetworkRelease. In case BusNmType is set to NM_BUSNM_LOCALNM **Nm** shall inform ComM about shutdown by calling ComM_Nm_BusSleepMode().⌋*(RS_Nm_02516)*

**Note:** In the AUTOSAR Classic Platform, CanNm_PassiveStartUp, J1939Nm_PassiveStartUp, FrNm_PassiveStartUp and UdpNm_PassiveStartUp have been specified as the predefined interfaces corresponding to <BusNm>_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.⌋*(RS_Nm_02516)*

**Note:** See Section subsection 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.

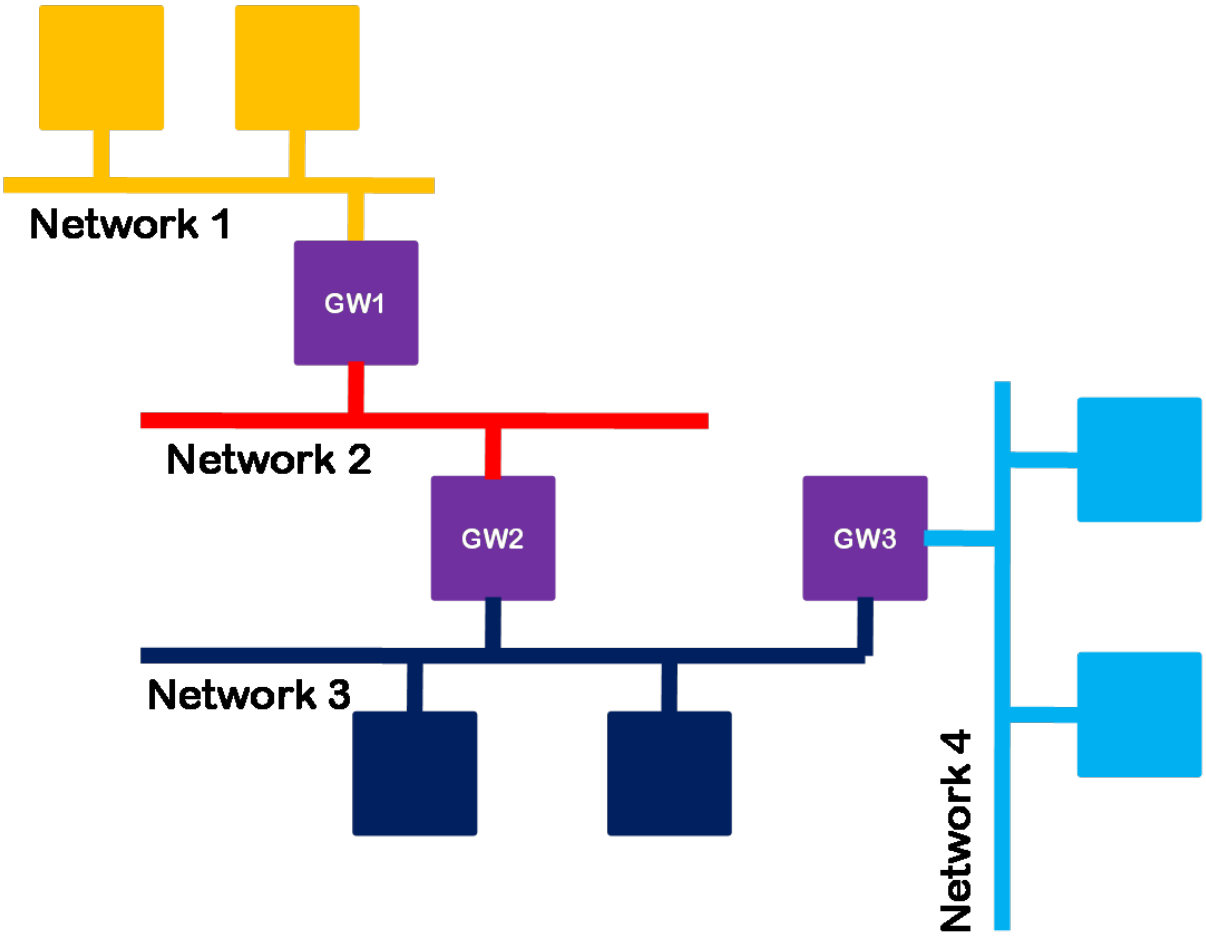


**Figure 7.2: Use Case Nested Gateways**

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`.⌋*(RS_Nm_02535)*

**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.⌋(*RS_Nm_02535*)

**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.

**[SWS_Nm_00259]** ⌈The NM Coordinator shall set the *NMcoordinatorSleepReady* bit in the NM message via <BusNm>_SetSleepReadyBit to the value 1 at his actively coordinated channels,
IF
all nodes of the NM Coordination cluster are ready to sleep (RemoteSleepIndication)
AND
IF NmSynchronizingNetwork is enabled a Nm_SynchronizationPoint() call has been received on the corresponding channel
AND
If all channels of this NM Coordination cluster are configured as NmActiveCoordinator == TRUE. AND
If the NmBusType is not configured to NM_BUSNM_LOCALNM.⌋(*RS_Nm_02535*)

**Note:** for Position of Coordinator Bits in CBV see according **<BusNm>** specifications.

**Note:** This applies to the top most coordinator (no passively coordinated channel).

**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.

**Note:** Nodes which contain a passively coordinated channel will set the bit according to the requirement in [SWS_Nm_00261].

**[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 <BusNm>_SetSleepReadyBit on all actively coordinated channels.⌋(*RS_Nm_02535*)

**[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 <BusNm>_SetSleepReadyBit on all actively coordinated channels.⌋(*RS_Nm_02535*)

**Note:** On its passively coordinated channel a NM Coordinator would not set the *Sleep Ready* bit ever (via **<busNm>** 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]** ⌈The NM Coordinator shall start coordinated shutdown after the *Sleep Ready* Bit with SET status has been requested.⌋*(RS_Nm_02535)*

**[SWS_Nm_00281]** ⌈NmGlobalCoordinatorTime shall be set at least to the maximum time needed to shut down all Networks coordinated.⌋*(RS_Nm_00149)*

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

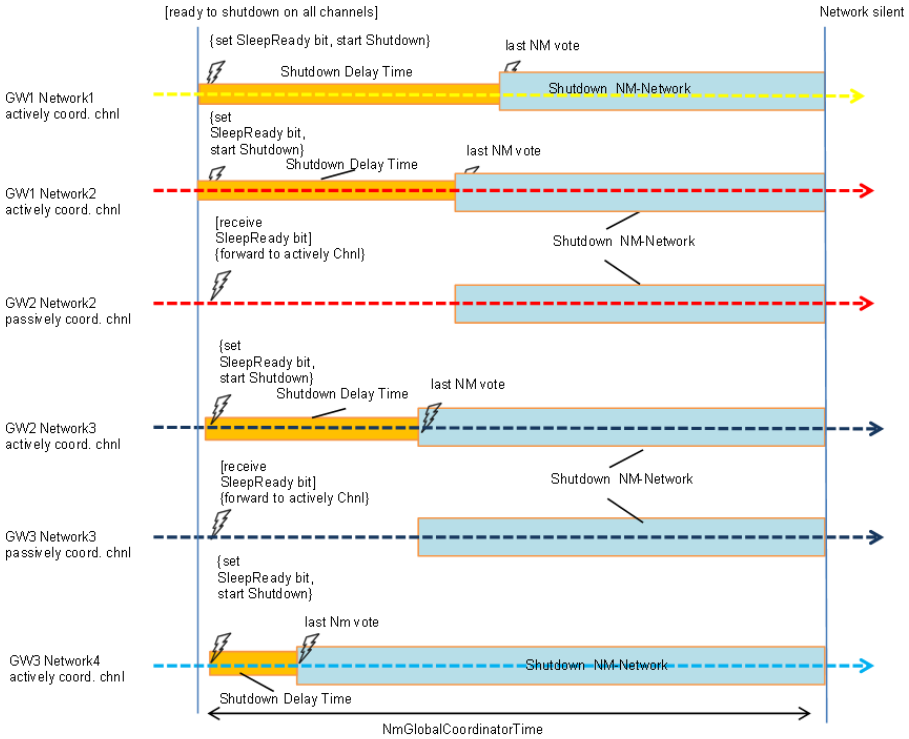


**Figure 7.3: Shutdown with Nm_GlobalCoordinatorTime**

**[SWS_Nm_00267]** ⌈NM Coordinator shall set the *NMCoordinatorSleepReady* bit to UNSET (0) via API call to <BusNm>_SetSleepReadyBit on all actively coordinated channels if the coordinated shutdown has been aborted for any reason.⌋*(RS_Nm_02535, RS_Nm_02537)*

**Note:** Details about aborted shutdown can be found in subsection 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_SynchronizationIndication (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**.

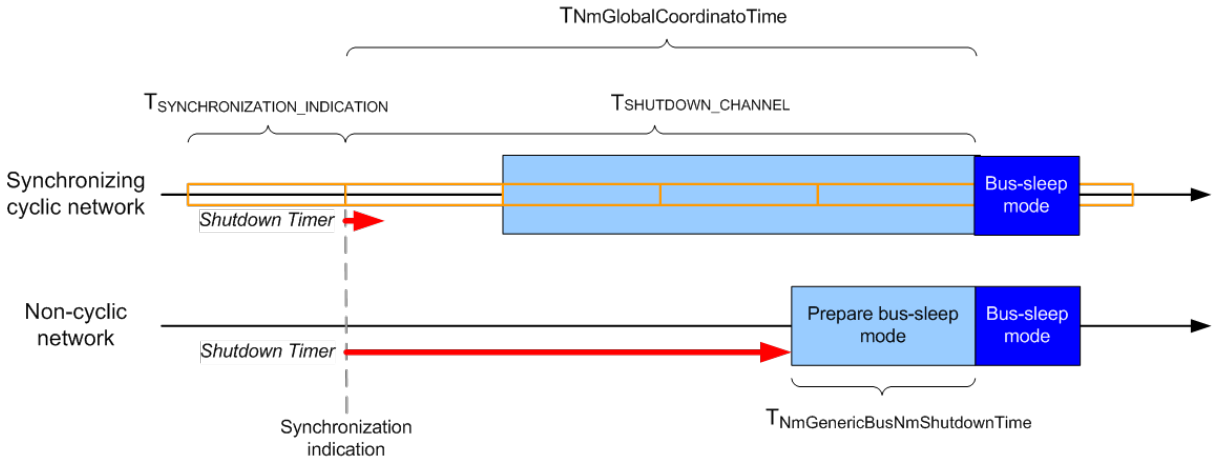


**Figure 7.4: Timing example one**

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**.

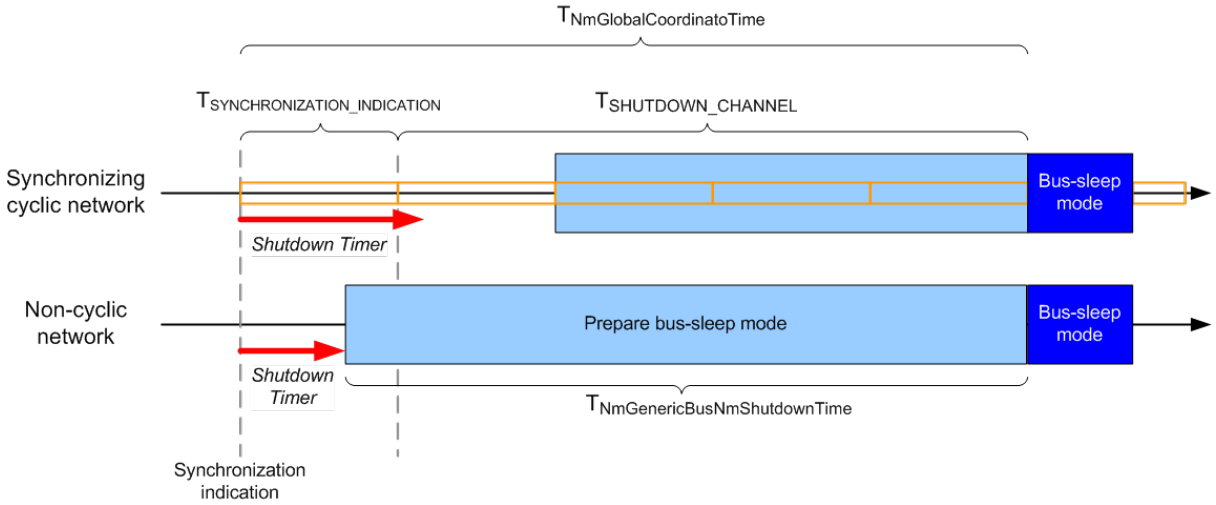


**Figure 7.5: Timing example two**

## 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 <BusNm>_NetworkRequest to the lower layer and accordingly, the network shall be counted as requested by the NM coordinator.⌋ *(RS_Nm_02536)*

**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.

⌋*(RS_Nm_02537)*

**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“.⌋*(RS_Nm_02537)*

**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 **<busNm'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().⌋*(RS_Nm_02537)*

**[SWS_Nm_00185]** ⌈If the coordination algorithm has been aborted, all conditions that guard the initiation of the coordinated shutdown shall be evaluated again.⌋*(RS_Nm_02537)*

**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 <BusNm>_NetworkRequest, that network shall not be considered awake when the conditions for initiating a coordinated shutdown are evaluated again.⌋*(RS_Nm_02537)*

**Rationale:** Any **<BusNm>** 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 **<BusNm>** 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 <BusNm>_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 <BusNm>_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.⌋*(RS_Nm_02537)*

**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 **<BusNm>**. The coordinated shutdown will eventually be initiated again as long as the problem with the **<BusNm>** persists. It is up to the **<BusNm>** 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

### 7.4.1 PNC bit vector filter algorithm

The intention of the PNC 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 bit vector received by the **<Bus>Nm** (passive PNC requests, initiated remotely by another ECU in the network) and it evaluates the PNC bit vector transmitted by **ComM** (active request, initiated by a local application). Every bit of the PNC 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]**{DRAFT} ⌈The PNC bit vector filter algorithm shall evaluate the bytes of the given PNC bit vector in the context of Nm_PncBitVectorRxIndication.⌋*(RS_Nm_02517, RS_Nm_02527)*

**[SWS_Nm_00310]**{DRAFT} ⌈Every bit (PNC bit) of the PNC bit vector represents one Partial Network Cluster (PNC). If the PNC bit is set to 1 the Partial Network Cluster is requested. If the bit is set to 0 there is no request for this PNC.⌋*(RS_Nm_02517)*

**[SWS_Nm_00311]**{DRAFT} ⌈The PNC bit vector filter algorithm shall compare (bit-wise AND) the received PNC bit vector with the PN filter mask to detect if a relevant PNC is requested or not. Each PNC bit of the PN filter mask shall have the following meaning:

- 0: The PNC request is irrelevant for the ECU. The communication stack of the ECU is not kept awake if this bit is set in a received PNC bit vector.
- 1: The PNC request is relevant for the ECU. The communication stack of the ECU is kept awake if this bit is set in a received PNC bit vector.

⌋*(RS_Nm_02517, RS_Nm_02527)*

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

**[SWS_Nm_00312]**{DRAFT} ⌈If `Nm_PncBitVectorRxIndication` is called, `NmPnEiraCalcEnabled` or `NmPnEraCalcEnabled` is set to TRUE and at least one PNC bit is detected as relevant PNC request according to [SWS_Nm_00311] in the given PNC bit vector, the OUT parameter `RelevantPncRequestDetectedPtr` shall be set to TRUE. Otherwise the value of `RelevantPncRequestDetectedPtr` shall be set to FALSE.⌋*(RS_Nm_02517, RS_Nm_02527)*

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 bit vector has a length of 2 bytes (`NmPncBitVectorLength`):

| **Byte 0** | **Byte 1** |
|---|---|
| PNC bit vector | |
| 0x12 | 0x8E |

**Table 7.1: Example of PNC bit vector**

For this example two `NmPnFilterMaskByte`s 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 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]**{DRAFT} ⌈The length of all configured NmPnFilterMaskBytes 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 NmPnFilterMaskBytes differ from the configured NmPncBitVectorLength of the corresponding NM-channel.⌋*(RS_Nm_02517, RS_Nm_02527)*

### 7.4.2 Aggregation of PNC requests

#### 7.4.2.1 Aggregation of internal and external Partial Network Cluster

This feature is used by every ECU that is member of a Partial Network Cluster (PNC). Typically such ECUs has to switch I-PDU-Groups according to requested PNCs. This is used for a proper communication behavior within the Partial Network (e.g. to prevent false timeouts). Therefore I-PDU-Groups shall be started if the corresponding PNC is internally or externally requested. As soon as a PNC in neither internally or externally requested, the corresponding I-PDU-Group shall be stopped. Internal PNC requests indicate ECU local communication needs and also called as "active PNC request". External PNC requests indicate communication needs of a remote ECU in the network and also called as "passive PNC request". The logic to control I-PDU-Groups is handled as interaction between ComM and BswM. ComM indicates the current state of a particular PNC statemachine to BswM. The BswM controls the corresponding I-PDU groups by means of mode arbitration and mode control. The **<Bus>Nm** provides the received information (PNC bit vector) regarding passive PNC requests. Active PNC requests are forwarded by the ComM via the Nm to the**<Bus>Nm**s. Passive PNC requests are received by the **<Bus>Nm**s and forwarded to the Nm. Nm handles received PNC request with respect to a so-called PN filter mask. The PN filter mask define which of the received PNC requests are relevant. Nm collects and maintains the states of internal PNC requests (a.k.a active PNC requests) and external PNC request (a.k.a. passive PNC requests). The aggregated state of the internal/external requested PNCs is called "**E**xternal **I**nternal **R**equest **A**rray (EIRA)". Changes of the states within EIRA are forwarded by Nm to ComM. ComM needs this information to handle changes in the corresponding PNC state machines. To switch the I-PDU-Groups synchronously on all direct connected ECUs, Nm shall provide the information of a request change (PNC request changed from requested to released and vice versa) to ComM at (almost) the same time on every ECU. Therefore the Nm maintain timers (so-called PNC reset timer) of each PNC request in the EIRA. The PNC reset timer is restarted every time the corresponding PNC is requested within received PNC bit vector and every time the corresponding PNC request is transmitted.

**[SWS_Nm_00302]**{DRAFT} ⌈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).⌋*(RS_Nm_02517, RS_Nm_02563)*

**[SWS_Nm_00313]**{DRAFT} ⌈If

- NmPnEiraCalcEnabled is TRUE
- and a PNC bit vector is received via Nm_PncBitVectorRxIndication(<Nm-channel>, <PncBitVector of external PNC requests>)
- 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".⌋*(RS_Nm_02517, RS_Nm_02563)*

**[SWS_Nm_00314]**{DRAFT} ⌈If

- NmPnEiraCalcEnabled is TRUE
- and a transmission of a PNC bit vector is indicated via Nm_PncBitVectorTxIndication(<NM-channel>, <buffer to provide the unfiltered PNC bit vector of aggregated internal PNC requests >)
- and PNCs are requested within the stored unfiltered PNC bit vector (bits set to 1)
- and the requested PNCs of the stored unfiltered PNC 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".⌋*(RS_Nm_02517, RS_Nm_02563)*

**[SWS_Nm_00332]**{DRAFT} ⌈If NmPnEiraCalcEnabled is set to TRUE and Nm_PncBitVectorTxIndication(NetworkHandle, PncBitVectorPtr) is called, then Nm shall copy the unfiltered PNC bit vector for internal PNC requests of the given NM-channel to the buffer indicated by PncBitVectorPtr.⌋*()*

**[SWS_Nm_00330]**{DRAFT} ⌈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).⌋*(RS_Nm_02517, RS_Nm_02562)*

**[SWS_Nm_00317]**{DRAFT} ⌈If Nm_UpdateIRA(<NM-channel>, <PncBitVector of aggregated internal PNC requests>) is called, the Nm shall store the received unfilterred PncBitVector 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.⌋*(RS_Nm_02517, RS_Nm_02563, SRS_ModeMgm_09250)*

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]**{DRAFT} ⌈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.⌋*(RS_Nm_02517, RS_Nm_02563)*

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]**{DRAFT} ⌈If NmPnEiraCalcEnabled is TRUE and every time a PNC is stored as "PNC requested" (see [SWS_Nm_00313] and [SWS_Nm_00314]), then the monitoring for this PNC shall be restarted with respect to NmPnResetTime in the context of Nm_Mainfunction.⌋*(RS_Nm_02517, RS_Nm_02563)*

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 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]**{DRAFT} ⌈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.⌋*(RS_Nm_02517, RS_Nm_02563)*

**[SWS_Nm_00321]**{DRAFT} ⌈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], [SWS_Nm_00314] and [SWS_Nm_00320]), then Nm shall forward the changed state of EIRA to ComM by calling ComM_Nm_UpdateEIRA (<PncBitVector of EIRA>)).⌋*(RS_Nm_02517, RS_Nm_02565, SRS_ModeMgm_09250)*

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

**Note:** This feature is used by the ECUs where the PNC gateway functionality is enabled to collect the external PNC requests per channel. The external PNC requests have to be coordinated across all affected channels. The logic of the PNC coordination is provided by ComM. Therefore each channel is configured if it is actively or passively coordinated. On active coordinated channels external PNC requests are mirrored back to the channel where the PNC request was received and also forwarded to other (required) channels. On passive coordinated channels external PNC requests are forwarded to other (required) channels without mirroring back on the channel from where the external PNC request was received. This avoids endless mirroring loop of "phantom PNC requests", if 2 ECUs have PNC gateway functionality enabled and connected to the same channel. The Nm module provides the information if PNCs are externally requested or released to ComM and manages the PNC timer handling for each relevant PNC and per channel. The aggregated state of the external requested PNs is called "**E**xternal **R**equest **A**rray" (ERA).

**[SWS_Nm_00322]**{DRAFT} ⌈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).⌋*(RS_Nm_02517, RS_Nm_02564)*

**[SWS_Nm_00323]**{DRAFT} ⌈If

- NmPnEraCalcEnabled is TRUE
- and a PNC bit vector is received via Nm_PncBitVectorRxIndication(<NM-channel>, <PncBitVector of external PNC requests>)
- 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.⌋*(RS_Nm_02517, RS_Nm_02564)*

**[SWS_Nm_00324]**{DRAFT} ⌈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.⌋*(RS_Nm_02517, RS_Nm_02564)*

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]**{DRAFT} ⌈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.⌋(*RS_Nm_02517, RS_Nm_02564*)

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 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]**{DRAFT} ⌈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).⌋(*RS_Nm_02517, RS_Nm_02564*)

**[SWS_Nm_00327]**{DRAFT} ⌈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>, <PncBitVector of ERA>).⌋(*RS_Nm_02517, RS_Nm_02565, SRS_ModeMgm_09250*)

**[SWS_Nm_00331]**{DRAFT} ⌈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]).⌋(*RS_Nm_02517*)

### 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]**{DRAFT} ⌈If NmPartialNetworkSupportEnabled is set to TRUE, then all PNC reset timers shall be stopped at initialization.⌋(*RS_Nm_02517, RS_Nm_02564*)

**[SWS_Nm_00329]**{DRAFT} ⌈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

⌋(*RS_Nm_02517*, *RS_Nm_02564*)

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 bit vector exists per channel
  - One EIRA as PNC bit vector exist

At initialization the PNC reset timers are in state "stopped" and the EIRA / ERA PNC 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 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 bit vector with relevant PNC requests is received or if a PNC bit vector with relevant PNC requests is transmitted. ERA is always updated if a PNC 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 toplevel 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 forwards the call immediately to the affected **<Bus>Nm**s. The **<Bus>Nm**s store all requests and handle them in the context of the <Bus>Nm_Mainfunction. Therefore, the **<Bus>Nm** has to aggregate the request for a synchronized PNC shutdown into a byte array (`PN info`), where each bit represents a particular PNC. The `PN info` is created and transmitted per <Bus>NMChannel.

If a PN shutdown message is received by an ECU in the role of an intermediate PNC coordinator, the **<Bus>Nm**s extract the PN info 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 all affected **<Bus>Nm**s via Nm 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 PN info and reset PN reset time).

**[SWS_Nm_00506]**{DRAFT} ⌈If function Nm_RequestSynchronizedPncShutdown is called, NmSynchronizedPncShutdownEnabled is set to TRUE and NmStandardBusType is set to a type other than NM_BUSNM_LOCALNM, the function shall immediately forward the call to the affected **<Bus>Nm** by calling <Bus>Nm_RequestSynchronizedPncShutdown. Otherwise the function shall return with E_NOT_OK.⌋*(RS_Nm_02543)*

**[SWS_Nm_00507]**{OBSOLETE} ⌈The indication of a received PN shutdown message via callback function Nm_ForwardSynchronizedPncShutdown shall be forwarded to **ComM** with the corresponding ComMChannel by calling ComM_Nm_ForwardSynchronizedPncShutdown(<ComMChannel>).⌋*(RS_Nm_02544)*

**[SWS_Nm_00305]**{DRAFT} ⌈If the recpetion 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 bit set to '1') in the given PNC bit vector and forward the indication to ComM with the corresponding ComMChannel by calling ComM_Nm_ForwardSynchronizedPncShutdown(<ComMChannel>, <PncBitVector>).⌋*(RS_Nm_02548)*

Note: The PNC bit vector of a received PN shutdown message shall be used to release the PNCs for a synchronized shutdown and pass this ERA information to the ComM module. The synchronized PNC shutdown has to be handled as fast as possible. Therefore, the ComM module is informed immediately.

## 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 `<BusNm>_NetworkRequest`, `<BusNm>_NetworkRelease` and `<BusNm>_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**:

- <BusNm>_NetworkRequest - [SWS_Nm_00119]
- <BusNm>_NetworkRelease - [SWS_Nm_00119]
- <BusNm>_RequestBusSynchronization - [SWS_Nm_00119]
- <BusNm>_CheckRemoteSleepIndication - [SWS_Nm_00119]

**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 **BusNm** used by the *coordinator functionality* of the Nm module must provide the API <BusNm>_NetworkRequest.

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

**Note:** The <BusNm>_RequestBusSynchronization is called by **Nm** immediately before <BusNm>_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 <BusNm>_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 <BusNm>_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 <BusNm>_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_RemoteSleepIndication has been notified and continue until either the network has been released ( <BusNm>_NetworkRelease) or the Nm_RemoteSleepCanncellation 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:

- <BusNm>_RequestSynchronizedPncShutdown - [SWS_Nm_00166][SWS_Nm_00506]

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_LOCAL_NM` 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.

**[SWS_Nm_00484]** ⌈The Nm module shall apply appropriate mechanisms to allow calls of its APIs from other partitions than its main function, e.g. by providing a Nm satellite.⌋ *(SRS_BSW_00459)*

**[SWS_Nm_00485]** ⌈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.⌋*(SRS_BSW_00459)*

**[SWS_Nm_00486]** ⌈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.⌋*(SRS_BSW_00459)*

**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.⌋*(RS_Nm_02503)*

**[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.⌋*(RS_Nm_02503)*

**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 Table 7.4 occur, `Nm_StateChangeNotification` shall call `Com_SendSignal(uint8, Com_SignalIdType, const void*)` for the signal referenced by `NmStateReportSignalRef` with the value according to Table 7.4⌋*(RS_Nm_00051)*

**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.⌋*(RS_Nm_00051)*

| 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) |

**Table 7.4: Network Management States**

**[SWS_Nm_00501]**{DRAFT} ⌈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

Section 7.2 "Error Handling" of the document [2, SWS_BSW General] 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]** ⌈

| *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 |

⌋*(SRS_BSW_00327, SRS_BSW_00337, SRS_BSW_00385, SRS_BSW_00386)*

### 7.9.2 Runtime Errors

There are no runtime errors.

### 7.9.3 Transient Faults

There are no transient faults.

### 7.9.4 Production Errors

There are no production errors.

### 7.9.5 Extended Production Errors

There are no extended production errors.

# 8 API specification

## 8.1 Imported types

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

**[SWS_Nm_00117]** ⌈

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

⌋(*SRS_BSW_00301*)

## 8.2 Type definitions

### 8.2.1 Nm_ModeType

**[SWS_Nm_00274]** ⌈

| *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 | | |

⌋(*RS_Nm_00044*)

### 8.2.2 Nm_StateType

**[SWS_Nm_00275]** ⌈

| *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 | | |

⌋(*RS_Nm_00050*)

### 8.2.3 Nm_BusNmType

**[SWS_Nm_00276]** ⌈

| | | | |
|---|---|---|---|
| ***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 | | |

⌋(*RS_Nm_00044*, *RS_Nm_00154*, *RS_Nm_02515*)

### 8.2.4 Nm_ConfigType

**[SWS_Nm_00282]** ⌈

| | |
|---|---|
| ***Name*** | Nm_ConfigType |
| ***Kind*** | Structure |

▽

△

| | | |
|---|---|---|
| ***Elements*** | implementation specific | |
| | ***Type*** | – |
| | ***Comment*** | – |
| ***Description*** | Configuration data structure of the Nm module. | |
| ***Available via*** | Nm.h | |

⌋(*SRS_BSW_00414*)

## 8.3 Function definitions

### 8.3.1 Standard services provided by NM Interface

#### 8.3.1.1 Nm_Init

**[SWS_Nm_00030]** ⌈

| | | |
|---|---|---|
| ***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 | |

⌋(*SRS_BSW_00101*, *SRS_BSW_00344*, *SRS_BSW_00358*, *SRS_BSW_00405*, *SRS_BSW_00414*)

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

**[SWS_Nm_00283]** ⌈The Configuration pointer `ConfigPtr` shall always have a NULL_PTR value.⌋(*SRS_BSW_00414*)

**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]** ⌈

| | | |
|---|---|---|
| ***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<br>E_NOT_OK: Passive start of network management has failed<br>NetworkHandle does not exist (development only)<br>Module not yet initialized (development only) |
| ***Description*** | This function calls the <BusNm>_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 | |

⌋*(RS_Nm_00046, RS_Nm_00051, RS_Nm_00151, RS_Nm_02513, RS_Nm_02536)*

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

**[SWS_Nm_00488]** ⌈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.⌋*(SRS_BSW_00323, SRS_BSW_00369, SRS_BSW_00386)*

#### 8.3.1.3 Nm_NetworkRequest

**[SWS_Nm_00032]** ⌈

| | | |
|---|---|---|
| ***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<br>E_NOT_OK: Requesting of bus communication has failed<br>NetworkHandle does not exist (development only)<br>Module not yet initialized (development only) |
| ***Description*** | This function calls the <BusNm>_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 | |

⌋(*RS_Nm_00046*, *RS_Nm_00047*, *RS_Nm_00051*, *RS_Nm_02513*, *RS_Nm_00151*)

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

**[SWS_Nm_00130]** ⌈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]** ⌈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.⌋*(SRS_BSW_00323*, *SRS_BSW_00369*, *SRS_BSW_00386)*

#### 8.3.1.4 Nm_NetworkRelease

**[SWS_Nm_00046]** ⌈

| | | |
|---|---|---|
| ***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<br>E_NOT_OK: Releasing of bus communication has failed<br>NetworkHandle does not exist (development only)<br>Module not yet initialized (development only) |
| ***Description*** | This function calls the <BusNm>_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 | |

⌋(*RS_Nm_00047*, *RS_Nm_00048*, *RS_Nm_00051*)

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

**[SWS_Nm_00132]** ⌈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]** ⌈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.⌋*(SRS_BSW_00323*, *SRS_BSW_00369*, *SRS_BSW_00386)*

### 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 `<BusNm>_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]** ⌈

| | | |
|---|---|---|
| ***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<br>E_NOT_OK: Disabling of NM PDU transmission ability has failed.<br>NetworkHandle does not exist (development only)<br>Module not yet initialized (development only) |

▽

△

| | |
|---|---|
| ***Description*** | Disables the NM PDU transmission ability. For that purpose <BusNm>_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 |

⌋*(RS_Nm_02513, RS_Nm_02512)*

**Note:** Caveats of `Nm_DisableCommunication`: The **<BusNm>** 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`.⌋*(RS_Nm_00150)*

**[SWS_Nm_00286]** ⌈If `Nm_DisableCommunication` 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_00491]** ⌈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.⌋ *(SRS_BSW_00323, SRS_BSW_00369, SRS_BSW_00386)*

#### 8.3.2.2 Nm_EnableCommunication

**[SWS_Nm_00034]** ⌈

| | | |
|---|---|---|
| ***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<br>E_NOT_OK: Enabling of NM PDU transmission ability has failed.<br>NetworkHandle does not exist (development only)<br>Module not yet initialized (development only) |
| ***Description*** | Enables the NM PDU transmission ability. For that purpose <BusNm>_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 | |

⌋*(RS_Nm_02512)*

**Note:** Caveats of Nm_EnableCommunication: The **<BusNm>** 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.⌋*(RS_Nm_00150)*

**[SWS_Nm_00287]** ⌈If Nm_EnableCommunication 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_00492]** ⌈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.⌋*(SRS_BSW_00323, SRS_BSW_00369, SRS_BSW_00386)*

### 8.3.3 Partial Network services provided by NM Interface

#### 8.3.3.1 Nm_RequestSynchronizedPncShutdown

**[SWS_Nm_91005]**{DRAFT} ⌈

| | | |
|---|---|---|
| ***Service Name*** | Nm_RequestSynchronizedPncShutdown (draft) | |
| ***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<br>E_NOT_OK: Request for a synchronized PNC shutdown has failed, e.g.<br>NetworkHandle does not exist (development only)<br>Module not yet initialized (development only) |
| ***Description*** | This function forward the request for a synchronized PNC shutdown of a particular PNC given by PncId to the affected <Bus>Nm by calling <Bus>Nm_RequestSynchronizedPncShutdown The function call is only valid if NmStandardBusType is not set to NM_BUSNM_LOCALNM (e.g. CanNm_RequestSynchronizedPncShutdown function is called for NM_BUSNM_CANNM).<br>**Tags:** atp.Status=draft | |
| ***Available via*** | Nm.h | |

⌋*(RS_Nm_02545, RS_Nm_02543)*

**[SWS_Nm_00508]**{DRAFT} ⌈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.⌋*(SRS_BSW_00323, SRS_BSW_00369, SRS_BSW_00386)*

#### 8.3.3.2 Nm_UpdateIRA

**[SWS_Nm_91007]**{DRAFT} ⌈

| | | |
|---|---|---|
| ***Service Name*** | Nm_UpdateIRA (draft) | |
| ***Syntax*** | `void Nm_UpdateIRA ( NetworkHandleType NetworkHandle, const uint8* PncBitVectorPtr )` | |
| ***Service ID [hex]*** | 0x26 | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Non Reentrant for the same NetworkHandle, reentrant otherwise | |
| ***Parameters (in)*** | NetworkHandle | Identification of the NM-channel |
| | PncBitVectorPtr | Pointer to the bit vector with all PNC 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.<br>**Tags:** atp.Status=draft | |
| ***Available via*** | Nm.h | |

⌋*(RS_Nm_02544)*

### 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]** ⌈

| | |
|---|---|
| ***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<br>E_NOT_OK: Setting of user data has failed<br>NetworkHandle does not exist (development only)<br>Module not yet initialized (development only) |
| ***Description*** | Set user data for NM messages transmitted next on the bus. For that purpose <BusNm>_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 | |

⌋*(RS_Nm_02503)*

**Note:** Caveats of `Nm_SetUserData`: The **<BusNm>** 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`.⌋*(RS_Nm_00150)*

**[SWS_Nm_00288]** ⌈If `Nm_SetUserData` 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_00241]** ⌈Configuration of `Nm_SetUserData`: If `NmComUserDataSupport` is `TRUE` the API `Nm_SetUserData` shall not be available.⌋*(RS_Nm_00150)*

**[SWS_Nm_00493]** ⌈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.⌋*(SRS_BSW_00323, SRS_BSW_00369, SRS_BSW_00386)*

#### 8.3.4.2 Nm_GetUserData

**[SWS_Nm_00036]** ⌈

| | | |
|---|---|---|
| ***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<br>E_NOT_OK: Getting of user data has failed<br>NetworkHandle does not exist (development only)<br>Module not yet initialized (development only) |
| ***Description*** | Get user data out of the last successfully received NM message. For that purpose <BusNm>_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 | |

⌋(*RS_Nm_02504*)

**Note:** Caveats of `Nm_GetUserData`: The **<BusNm>** 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`.⌋(*RS_Nm_00150*)

**[SWS_Nm_00494]** ⌈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.⌋(*SRS_BSW_00323*, *SRS_BSW_00369*, *SRS_BSW_00386*)

### 8.3.4.3 Nm_GetPduData

**[SWS_Nm_00037]** ⌈

| | | |
|---|---|---|
| ***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<br>E_NOT_OK: Getting of NM PDU data has failed<br>NetworkHandle does not exist (development only)<br>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 **<BusNm>** and the **Nm** itself are initialized correctly.

**[SWS_Nm_00495]** ⌈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.⌋*(SRS_BSW_00323, SRS_BSW_00369, SRS_BSW_00386)*

#### 8.3.4.4 Nm_RepeatMessageRequest

**[SWS_Nm_00038]** ⌈

| | | |
|---|---|---|
| ***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<br>E_NOT_OK: Setting of Repeat Message Request Bit has failed<br>NetworkHandle does not exist (development only)<br>Module not yet initialized (development only) |
| ***Description*** | Set Repeat Message Request Bit for NM messages transmitted next on the bus. For that purpose <BusNm>_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 | |

⌋*(RS_Nm_00153)*

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

**[SWS_Nm_00289]** ⌈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]** ⌈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.⌋ *(SRS_BSW_00323, SRS_BSW_00369, SRS_BSW_00386)*

#### 8.3.4.5 Nm_GetNodeIdentifier

**[SWS_Nm_00039]** ⌈

| | | |
|---|---|---|
| ***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<br>E_NOT_OK: Getting of the node identifier out of the last received NM-message has failed<br>NetworkHandle does not exist (development only)<br>Module not yet initialized (development only) |
| ***Description*** | Get node identifier out of the last successfully received NM-message. The function <BusNm>_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 | |

⌋*(RS_Nm_02506)*

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

**[SWS_Nm_00497]** ⌈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.⌋*(SRS_BSW_00323, SRS_BSW_00369, SRS_BSW_00386)*

### 8.3.4.6 Nm_GetLocalNodeIdentifier

**[SWS_Nm_00040]** ⌈

| | | |
|---|---|---|
| ***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<br>E_NOT_OK: Getting of the node identifier of the local node has failed<br>NetworkHandle does not exist (development only)<br>Module not yet initialized (development only) |
| ***Description*** | Get node identifier configured for the local node. For that purpose <BusNm>_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 | |

⌋*(RS_Nm_02508)*

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

**[SWS_Nm_00498]** ⌈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.⌋ *(SRS_BSW_00323, SRS_BSW_00369, SRS_BSW_00386)*

### 8.3.4.7 Nm_CheckRemoteSleepIndication

**[SWS_Nm_00042]** ⌈

| | |
|---|---|
| ***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<br>E_NOT_OK: Checking of remote sleep indication bits has failed<br>NetworkHandle does not exist (development only)<br>Module not yet initialized (development only) |
| ***Description*** | Check if remote sleep indication takes place or not. For that purpose <BusNm>_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 | |

⌋*(RS_Nm_02513)*

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

**[SWS_Nm_00290]** ⌈If `Nm_CheckRemoteSleepIndication` 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.⌋*(RS_Nm_00150)*

**[SWS_Nm_00150]** ⌈Configuration of `Nm_CheckRemoteSleepIndication`: This function is only available if `NmRemoteSleepIndEnabled` is set to `TRUE`.⌋*(RS_Nm_00150)*

**[SWS_Nm_00499]** ⌈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.⌋*(SRS_BSW_00323, SRS_BSW_00369, SRS_BSW_00386)*

### 8.3.4.8 Nm_GetState

**[SWS_Nm_00043]** ⌈

| | | |
|---|---|---|
| ***Service Name*** | Nm_GetState | |
| ***Syntax*** | `Std_ReturnType Nm_GetState (`<br>`  NetworkHandleType nmNetworkHandle,`<br>`  Nm_StateType* nmStatePtr,`<br>`  Nm_ModeType* nmModePtr`<br>`)` | |
| ***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<br>E_NOT_OK: Getting of NM state has failed<br>NetworkHandle does not exist (development only)<br>Module not yet initialized (development only) |
| ***Description*** | Returns the state of the network management. The function <BusNm>_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 | |

⌋(*RS_Nm_00050*)

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

**[SWS_Nm_00500]** ⌈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.⌋(*SRS_BSW_00323*, *SRS_BSW_00369*, *SRS_BSW_00386*)

### 8.3.4.9 Nm_GetVersionInfo

**[SWS_Nm_00044]** ⌈

| | | |
|---|---|---|
| ***Service Name*** | Nm_GetVersionInfo | |
| ***Syntax*** | `void Nm_GetVersionInfo ( Std_VersionInfoType* nmVerInfoPtr )` | |
| ***Service ID [hex]*** | 0x0f | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Reentrant | |
| ***Parameters (in)*** | None | |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | nmVerInfoPtr | Pointer to where to store the version information of this module. |
| ***Return value*** | None | |
| ***Description*** | This service returns the version information of this module. | |
| ***Available via*** | Nm.h | |

⌋(*SRS_BSW_00003*, *SRS_BSW_00407*, *SRS_BSW_00482*)

#### 8.3.4.10 Nm_PnLearningRequest

**[SWS_Nm_91003]**{DRAFT} ⌈

| | | |
|---|---|---|
| ***Service Name*** | Nm_PnLearningRequest (draft) | |
| ***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<br>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 <BusNm>_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.<br>**Tags:** atp.Status=draft | |
| ***Available via*** | Nm.h | |

⌋*()*

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

**[SWS_Nm_00502]**{DRAFT} ⌈`Nm_PnLearningRequest` shall only be available if `NmDynamicPncToChannelMappingSupport` is set to `TRUE`.⌋*()*

**[SWS_Nm_00503]**{DRAFT} ⌈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]**{DRAFT} ⌈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.⌋*(SRS_BSW_00323, SRS_BSW_00369, SRS_BSW_00386)*

## 8.4 Call-back notifications

Callback notifications are called by the lower layer's bus-specific Network Management modules. For the Base functionality of Nm ( section 7.1) the call-backs shall be forwarded to the upper layer's ComM. For the NM Coordinator functionality of Nm (

section 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]** ⌈All callbacks of the Nm shall assume that they can run either in task or in interrupt context.⌋ *(SRS_BSW_00333)*

### 8.4.1 Standard Call-back notifications

#### 8.4.1.1 Nm_NetworkStartIndication

**[SWS_Nm_00154]** ⌈

| | | |
|---|---|---|
| ***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 | |

⌋ *(SRS_BSW_00359, RS_Nm_02513)*

**[SWS_Nm_00155]** ⌈The indication through callback function `Nm_NetworkStartIndication`: shall be forwarded to **ComM** by calling the `ComM_Nm_NetworkStartIndication`.⌋ *(RS_Nm_02513)*

#### 8.4.1.2 Nm_NetworkMode

**[SWS_Nm_00156]** ⌈

| | |
|---|---|
| ***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 | |

⌋(*SRS_BSW_00359*, *RS_Nm_00051*)

**[SWS_Nm_00158]** ⌈The indication through callback function `Nm_NetworkMode`: shall be forwarded to **ComM** by calling the `ComM_Nm_NetworkMode`.⌋(*RS_Nm_00051*)

#### 8.4.1.3 Nm_BusSleepMode

**[SWS_Nm_00162]** ⌈

| | | |
|---|---|---|
| ***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 | |

⌋(*SRS_BSW_00359*, *RS_Nm_00051*)

**[SWS_Nm_00163]** ⌈The indication through callback function `Nm_BusSleepMode`: shall be forwarded to **ComM** by calling the `ComM_Nm_BusSleepMode`.⌋(*RS_Nm_00051*)

#### 8.4.1.4 Nm_PrepareBusSleepMode

**[SWS_Nm_00159]** ⌈

| | | |
|---|---|---|
| ***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 | |

⌋(*SRS_BSW_00359*, *RS_Nm_00051*)

**[SWS_Nm_00161]** ⌈The indication through callback function `Nm_PrepareBusSleepMode`: shall be forwarded to **ComM** by calling `ComM_Nm_PrepareBusSleepMode`.⌋(*RS_Nm_00051*)

#### 8.4.1.5 NM_SynchronizeMode

**[SWS_Nm_91002]** ⌈

| | | |
|---|---|---|
| ***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 | |

⌋()

#### 8.4.1.6 Nm_RemoteSleepIndication

**[SWS_Nm_00192]** ⌈

| | | |
|---|---|---|
| ***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 | |

⌋(*SRS_BSW_00359*, *RS_Nm_00052*)

**[SWS_Nm_00277]** ⌈Configuration of `Nm_RemoteSleepIndication`: This function is only available if `NmRemoteSleepIndEnabled` is set to `TRUE`.⌋(*RS_Nm_00150*)

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.

#### 8.4.1.7 Nm_RemoteSleepCancellation

**[SWS_Nm_00193]** ⌈

| | | |
|---|---|---|
| ***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 | |

⌋(*SRS_BSW_00359*, *RS_Nm_02509*)

**[SWS_Nm_00278]** ⌈Configuration of `Nm_RemoteSleepCancellation`: This function is only available if `NmRemoteSleepIndEnabled` is set to `TRUE`.⌋*(RS_Nm_00150)*

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.

#### 8.4.1.8 Nm_SynchronizationPoint

**[SWS_Nm_00194]** ⌈

| | | |
|---|---|---|
| ***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 | |

⌋*(SRS_BSW_00359, RS_Nm_02516)*

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.

#### 8.4.1.9 Nm_CoordReadyToSleepIndication

**[SWS_Nm_00254]** ⌈

| | | |
|---|---|---|
| ***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 |

⌋*(SRS_BSW_00359, RS_Nm_02535)*

**[SWS_Nm_00255]** ⌈Configuration of `Nm_CoordReadyToSleepIndication`: Optional
If `NmCoordinatorSyncSupport` is set to `TRUE` , the Nm shall provide the API `Nm_CoordReadyToSleepIndication`.⌋*(RS_Nm_00150)*

#### 8.4.1.10 Nm_CoordReadyToSleepCancellation

**[SWS_Nm_00272]** ⌈

| | | |
|---|---|---|
| ***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 | |

⌋*(SRS_BSW_00359, RS_Nm_02535)*

**[SWS_Nm_00273]** ⌈Configuration of `Nm_CoordReadyToSleepCancellation`: Optional
If `NmCoordinatorSyncSupport` is set to `TRUE` , the Nm shall provide the API `Nm_CoordReadyToSleepCancellation`.⌋*(RS_Nm_00150)*

#### 8.4.1.11 Nm_ForwardSynchronizedPncShutdown

**[SWS_Nm_91009]**{DRAFT} ⌈

| | | |
|---|---|---|
| ***Service Name*** | Nm_ForwardSynchronizedPncShutdown (draft) | |
| ***Syntax*** | `void Nm_ForwardSynchronizedPncShutdown ( NetworkHandleType NetworkHandle, const uint8* PncBitVectorPtr )` | |
| ***Service ID [hex]*** | 0x28 | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Non Reentrant for the same NetworkHandle, reentrant otherwise | |
| ***Parameters (in)*** | NetworkHandle | Identification of the NM-channel |
| | PncBitVectorPtr | Pointer to the bit vector with all PNC 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.<br>**Tags:** atp.Status=draft | |
| ***Available via*** | Nm.h | |

⌋(*RS_Nm_02544*)

### 8.4.1.12 Nm_PncBitVectorRxIndication

**[SWS_Nm_91006]**{DRAFT} ⌈

| | | |
|---|---|---|
| ***Service Name*** | Nm_PncBitVectorRxIndication (draft) | |
| ***Syntax*** | `void Nm_PncBitVectorRxIndication ( NetworkHandleType NetworkHandle, const uint8* PncBitVectorPtr, 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 |
| | PncBitVectorPtr | Pointer to the bit vector with all PNC 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 PncBitVector |
| ***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 bit vector. This is used to aggregate the external PNC requests. The function evaluate if a relevant PNC request (PNC bit set to '1') is available in the given PNC bit vector. If a relevant PNC request is available (PNC bit passes the PNC bit vector filter), then the RelevantPncRequestDetectedPtr refers to a boolean with value set to TRUE. Otherwise refer to booelan with value set to FALSE. RelevantPncRequestDetectedPtr is evaluated by the callee <Bus>Nm module to qualify the further processing of the received NM-PDU.<br>**Tags:** atp.Status=draft |
| ***Available via*** | Nm.h |

⌋(*RS_Nm_02544*, *SRS_ModeMgm_09250*)

#### 8.4.1.13 Nm_PncBitVectorTxIndication

**[SWS_Nm_91008]**{DRAFT} ⌈

| | | |
|---|---|---|
| ***Service Name*** | Nm_PncBitVectorTxIndication (draft) | |
| ***Syntax*** | `void Nm_PncBitVectorTxIndication ( NetworkHandleType NetworkHandle, uint8* PncBitVectorPtr )` | |
| ***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)*** | PncBitVectorPtr | Pointer to the bit vector with all PNC 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.<br>**Tags:** atp.Status=draft | |
| ***Available via*** | Nm.h | |

⌋(*RS_Nm_02544*, *SRS_ModeMgm_09250*)

### 8.4.2 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.

#### 8.4.2.1 Nm_PduRxIndication

**[SWS_Nm_00112]** ⌈

| | | |
|---|---|---|
| ***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 | |

⌋*(SRS_BSW_00359)*

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.⌋*(RS_Nm_00150)*

#### 8.4.2.2 Nm_StateChangeNotification

**[SWS_Nm_00114]** ⌈

| | | |
|---|---|---|
| ***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 <BusNm> has changed. |
| ***Available via*** | Nm.h |

⌋*(SRS_BSW_00359, RS_Nm_00050, RS_Nm_00051)*

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.⌋*(RS_Nm_00150)*

#### 8.4.2.3 Nm_RepeatMessageIndication

**[SWS_Nm_00230]** ⌈

| | | |
|---|---|---|
| ***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 | |

⌋*(SRS_BSW_00359, RS_Nm_00153, RS_Nm_00051)*

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]**{DRAFT} ⌈If `Nm_RepeatMessageIndication` is called with `pnLearningBitSet` set to `TRUE` and `NmDynamicPncToChannelMappingEnabled` is set to `TRUE` for the provided `nmNetworkHandle` Nm shall call `ComM_Nm_PncLearningBitIndication` with the corresponding network handle.⌋*()*

#### 8.4.2.4 Nm_TxTimeoutException

**[SWS_Nm_00234]** ⌈

| | | |
|---|---|---|
| ***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 | |

⌋*(SRS_BSW_00359)*

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

#### 8.4.2.5 Nm_CarWakeUpIndication

**[SWS_Nm_00250]** ⌈

| | | |
|---|---|---|
| ***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 | |

⌋*(SRS_BSW_00359, RS_Nm_02503, RS_Nm_02536)*

**[SWS_Nm_00251]** ⌈Configuration of `Nm_CarWakeUpIndication`: Optional
If `NmCarWakeUpRxEnabled` is `TRUE`, The Nm shall provide the API `Nm_CarWakeUpIndication`.⌋*(RS_Nm_00150)*

## 8.5 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]** ⌈A scheduled main function shall only contain logic related to the *NM Coordinator functionality*.⌋*(SRS_BSW_00373)*

**[SWS_Nm_00121]** ⌈In case the main function is called before the Nm has been initialized, the main function shall immediately return without yielding an error.⌋*(SRS_BSW_00450)*

**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.

### 8.5.1 Nm_MainFunction

**[SWS_Nm_00118]** ⌈

| | |
|---|---|
| ***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 |

⌋*(SRS_BSW_00424, SRS_BSW_00425)*

**[SWS_Nm_00279]** ⌈If `NmCoordinatorSupportEnabled` is set to TRUE, the `Nm_MainFunction` API shall be available.⌋*(RS_Nm_00150)*

## 8.6 Expected interfaces

This chapter lists all interfaces required from other modules.

### 8.6.1 Mandatory Interfaces

This chapter lists all interfaces required from other modules.

**[SWS_Nm_00119]** ⌈

| API Function | Header File | Description |
|---|---|---|
| <BusNm>_GetState | – | Returns the state and the mode of the network management. |
| <BusNm>_NetworkRelease | – | Release the network, since ECU doesn't have to communicate on the bus. |
| <BusNm>_NetworkRequest | – | Request the network, since ECU needs to communicate on the bus. |
| <BusNm>_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_ForwardSynchronizedPnc Shutdown (obsolete) | 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. Therefore, ComM will immediately release the affected PNC state machines and forward the PN information 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.<br>**Tags:** atp.Status=obsolete |
| 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. |

⌋(*RS_Nm_02515*, *RS_Nm_02536*)

### 8.6.2 Optional Interfaces

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

**[SWS_Nm_00166]** ⌈

| API Function | Header File | Description |
|---|---|---|
| <BusNm>_CheckRemoteSleep Indication | – | Check if remote sleep indication takes place or not. |
| <BusNm>_DisableCommunication | – | Disable the NM PDU transmission ability. |
| <BusNm>_EnableCommunication | – | Enable the NM PDU transmission ability. |
| <BusNm>_GetLocalNodeIdentifier | – | Get node identifier configured for the local node. |
| <BusNm>_GetNodeIdentifier | – | Get node identifier out of the last successfully received NM-message. |
| <BusNm>_GetPduData | – | Pointer where NM PDU shall be copied to. |
| <BusNm>_GetUserData | – | Get user data out of the last successfully received NM message. |
| <BusNm>_PnLearningRequest | – | – |
| <BusNm>_RepeatMessageRequest | – | Request a Repeat Message Request to be transmitted next on the bus. |
| <BusNm>_RequestBus Synchronization | – | Request bus synchronization. |
| <BusNm>_RequestSynchronizedPnc Shutdown | – | – |
| <BusNm>_SetSleepReadyBit | – | Set the NM Coordinator Sleep Ready bit in the Control Bit Vector |
| <BusNm>_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 (draft) | 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 PncBitVectorPtr. Therefore, ComM will immediately release the affected PNC state machines and forward the PN information 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.<br>**Tags:** atp.Status=draft |
| ComM_Nm_PncLearningBitIndication (draft) | ComM_Nm.h | Service to indicate that an NM message with set PNC Learning Bit has been received.<br>**Tags:** atp.Status=draft |
| ComM_Nm_RepeatMessageLeft Indication (draft) | ComM_Nm.h | Notification that the state of all <BusNm> has left RepeatMessage. This interface is used to indicate by the optional Dynamic PNC-to-channel-mapping feature to indicate that learning phase ends.<br>**Tags:** atp.Status=draft |
| Det_ReportError | Det.h | Service to report development errors. |

(*RS_Nm_00150*, *RS_Nm_02515*)

### 8.6.3 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.

#### 8.6.3.1 NmCarWakeUpCallout

**[SWS_Nm_00291]** ⌈

| | | |
|---|---|---|
| ***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 | |

⌋(*RS_Nm_02504*)

## 8.7 Version Check

For details refer to the chapter 5.1.8 "Version Check" in [2, SWS_BSWGeneral].

# 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.

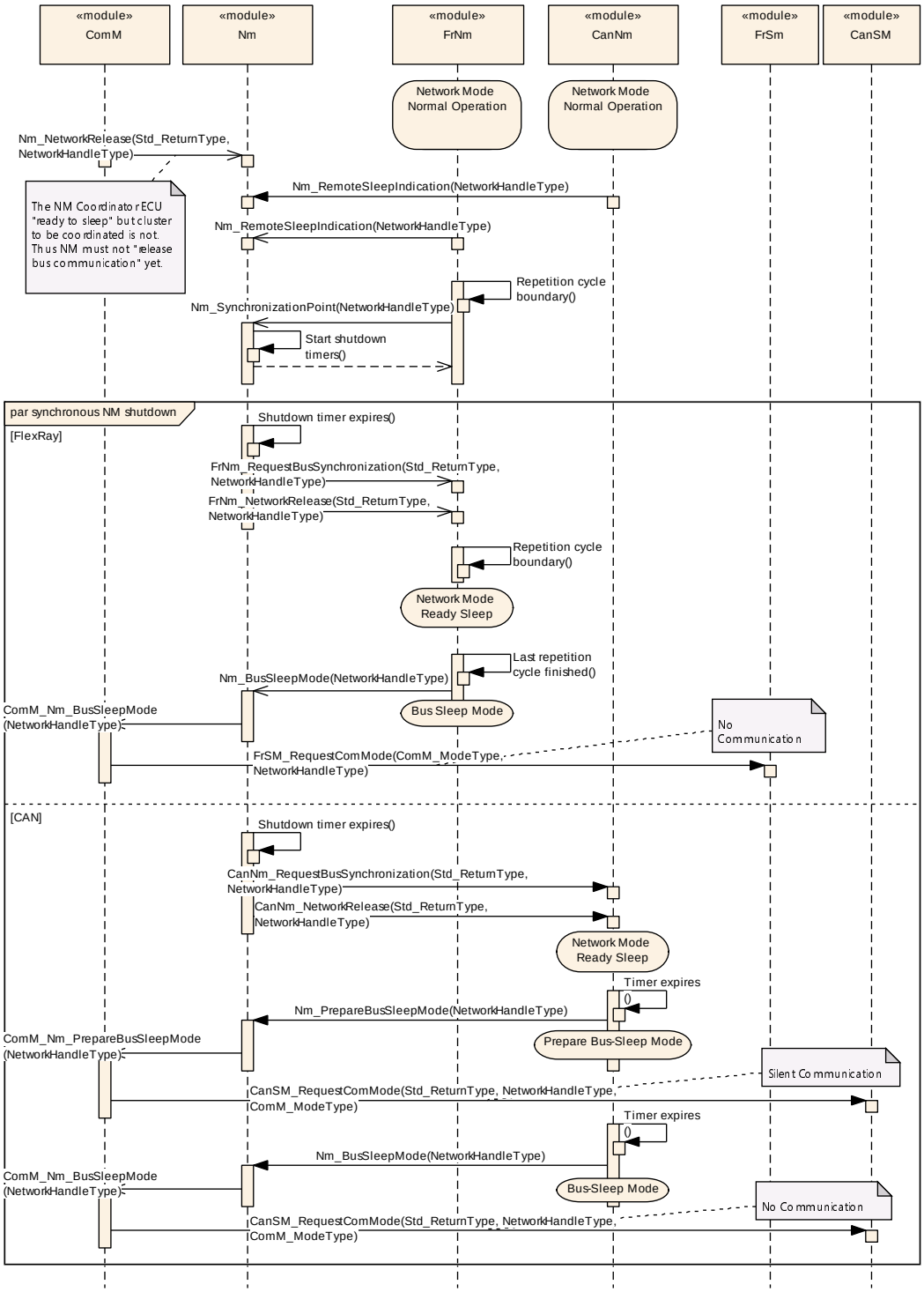


**Figure 9.1: Nm Coordination**

## 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`

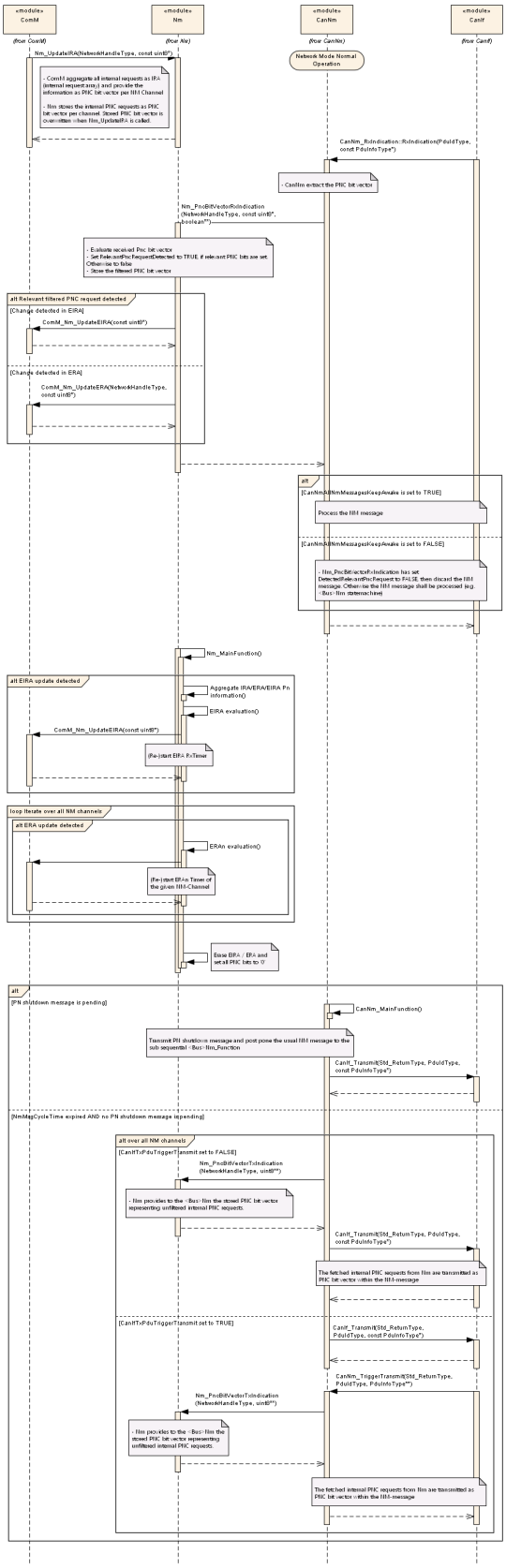


**Figure 9.2: Partial Network functionality**

# 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. section 10.1 describes fundamentals. section 10.2, section 10.3 and section 10.4 specifies the structure (containers) and the parameters of the Nm. The section 10.5 specifies published information of the Nm.

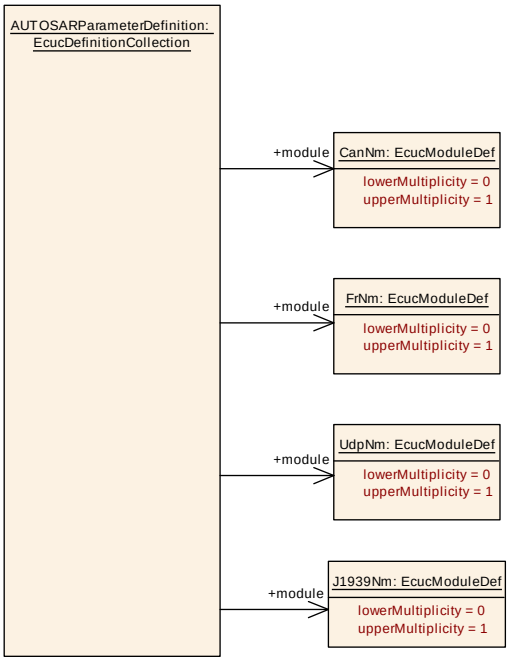


**Figure 10.1: Network Management Overview**

## 10.1 How to read this chapter

For details refer to the [2, chapter 10.1 “Introduction to configuration specification” in SWS_BSWGeneral]

## 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

| Module SWS Item | ECUC_Nm_00243 | |
|---|---|---|
| Module Name | Nm | |
| Module Description | The Generic Network Management Interface module | |
| Post-Build Variant Support | false | |
| Supported Config Variants | VARIANT-LINK-TIME, VARIANT-PRE-COMPILE | |
| Included Containers | | |
| Container Name | Multiplicity | Scope / Dependency |
| 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. |

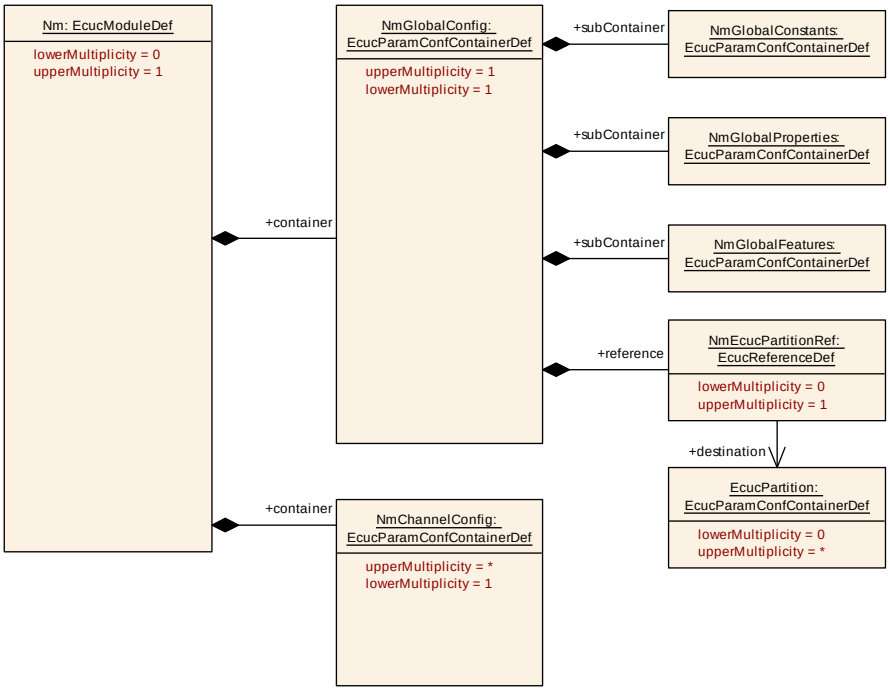


**Figure 10.2: Nm configuration container overview**

## 10.3 Global configurable parameters

### 10.3.1 NmGlobalConfig

| SWS Item | [ECUC_Nm_00196] |
|---|---|
| Container Name | NmGlobalConfig |
| Parent Container | Nm |

| Description | This container contains all global configuration parameters of the Nm Interface. |
|---|---|
| **Configuration Parameters** | |

| Name | NmEcucPartitionRef [ECUC_Nm_00245] | | |
|---|---|---|---|
| Parent Container | NmGlobalConfig | | |
| Description | Reference to EcucPartition, where Nm module is assigned to. | | |
| Multiplicity | 0..1 | | |
| Type | Reference to EcucPartition | | |
| Post-Build Variant Multiplicity | false | | |
| Post-Build Variant Value | false | | |
| Multiplicity Configuration Class | Pre-compile time | X | All Variants |
| | Link time | – | |
| | Post-build time | – | |
| Value Configuration Class | Pre-compile time | X | All Variants |
| | Link time | – | |
| | Post-build time | – | |
| Scope / Dependency | scope: local | | |

| Included Containers | | |
|---|---|---|
| Container Name | Multiplicity | Scope / Dependency |
| NmGlobalConstants | 1 | |
| NmGlobalFeatures | 1 | |
| NmGlobalProperties | 1 | |

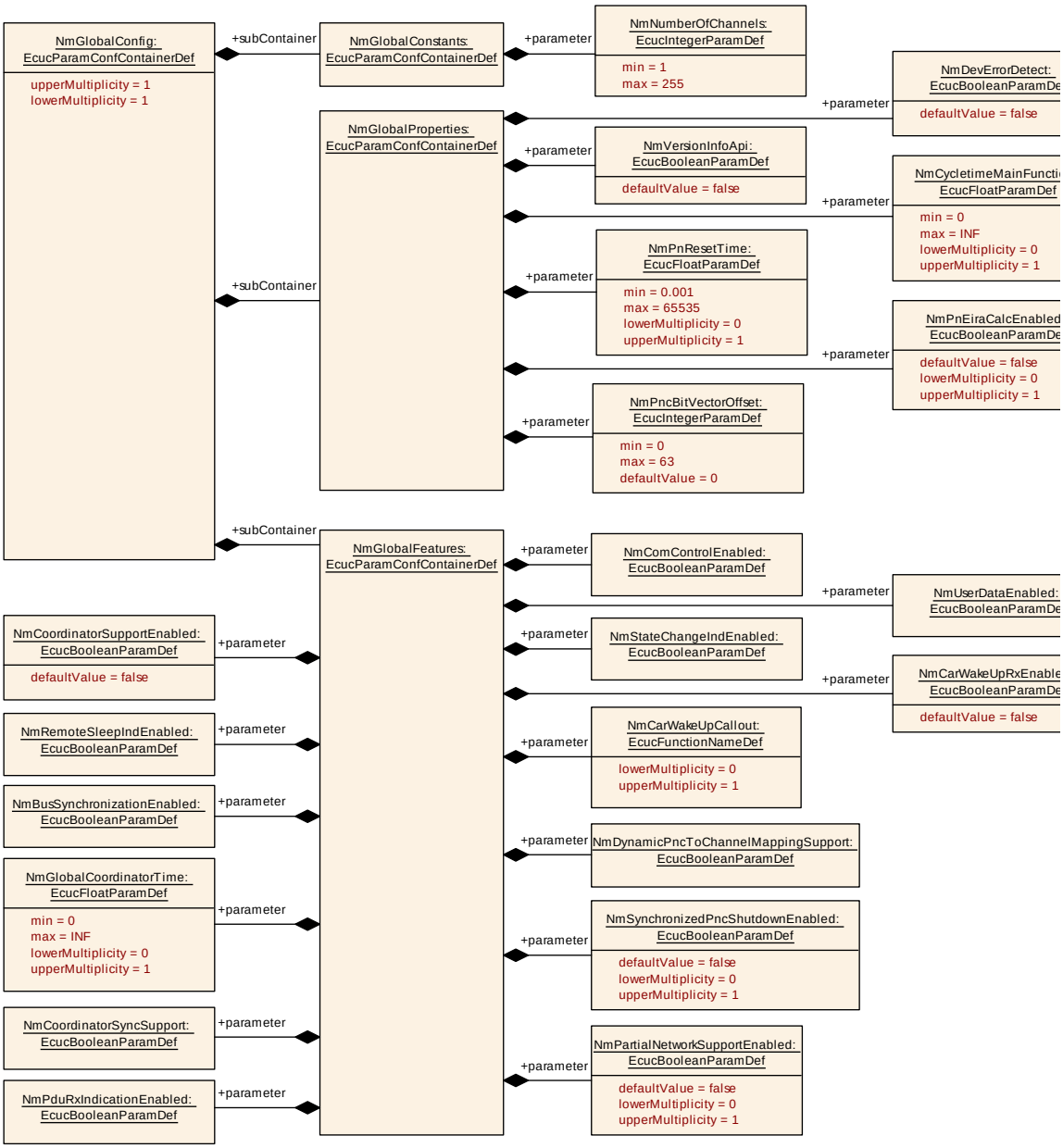


**Figure 10.3: NmGlobalConfig overview**

## 10.3.2 NmGlobalConstants

| SWS Item | [ECUC_Nm_00198] |
|---|---|
| Container Name | NmGlobalConstants |
| Parent Container | NmGlobalConfig |
| Description | |
| Configuration Parameters | |

| | | | |
|---|---|---|---|
| **Name** | NmNumberOfChannels [ECUC_Nm_00201] | | |
| **Parent Container** | NmGlobalConstants | | |
| **Description** | Number of NM channels allowed within one ECU. | | |
| **Multiplicity** | 1 | | |
| **Type** | EcucIntegerParamDef | | |
| **Range** | 1 .. 255 | | |
| **Default Value** | | | |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time** | X | All Variants |
| | **Link time** | – | |
| | **Post-build time** | – | |
| **Scope / Dependency** | scope: local | | |

| **No Included Containers** |
|---|

### 10.3.3 NmGlobalProperties

| | |
|---|---|
| **SWS Item** | [ECUC_Nm_00199] |
| **Container Name** | NmGlobalProperties |
| **Parent Container** | NmGlobalConfig |
| **Description** | |
| **Configuration Parameters** | |

| | | | |
|---|---|---|---|
| **Name** | NmCycletimeMainFunction [ECUC_Nm_00205] | | |
| **Parent Container** | NmGlobalProperties | | |
| **Description** | The period between successive calls to the Main Function of the NM Interface in seconds. | | |
| **Multiplicity** | 0..1 | | |
| **Type** | EcucFloatParamDef | | |
| **Range** | ]0 .. INF[ | | |
| **Default Value** | | | |
| **Post-Build Variant Multiplicity** | false | | |
| **Post-Build Variant Value** | false | | |
| **Multiplicity Configuration Class** | **Pre-compile time** | X | VARIANT-PRE-COMPILE |
| | **Link time** | X | VARIANT-LINK-TIME |
| | **Post-build time** | – | |
| **Value Configuration Class** | **Pre-compile time** | X | VARIANT-PRE-COMPILE |
| | **Link time** | X | VARIANT-LINK-TIME |
| | **Post-build time** | – | |
| **Scope / Dependency** | scope: local<br>dependency: If NmCoordinatorSupportEnabled is set to TRUE, then the NmCycletimeMainFunction shall be configured. | | |

| | | | |
|---|---|---|---|
| **Name** | NmDevErrorDetect [ECUC_Nm_00203] | | |
| **Parent Container** | NmGlobalProperties | | |
| **Description** | Switches the development error detection and notification on or off.<br>• true: detection and notification is enabled.<br>• false: detection and notification is disabled. | | |
| **Multiplicity** | 1 | | |
| **Type** | EcucBooleanParamDef | | |
| **Default Value** | false | | |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time** | X | All Variants |
| | **Link time** | – | |
| | **Post-build time** | – | |
| **Scope / Dependency** | scope: local | | |

| | | | |
|---|---|---|---|
| **Name** | NmPncBitVectorOffset [ECUC_Nm_00252] | | |
| **Parent Container** | NmGlobalProperties | | |
| **Description** | Parameter to configure the offset in bytes of the PNC bit vector that contains the PNC requests, which is transmitted within NM PDU by the corresponding <Bus>Nm.<br><br>**Tags:**<br>atp.Status=draft | | |
| **Multiplicity** | 1 | | |
| **Type** | EcucIntegerParamDef | | |
| **Range** | 0 .. 63 | | |
| **Default Value** | 0 | | |
| **Value Configuration Class** | **Pre-compile time** | X | VARIANT-PRE-COMPILE |
| | **Link time** | X | VARIANT-LINK-TIME |
| | **Post-build time** | – | |
| **Scope / Dependency** | scope: ECU<br>dependency: This parameter is only valid if NmPartialNetworkSupportEnabled is set to TRUE.<br><br>NmPncBitVectorOffset == Number of <Bus>Nm Sytem Bytes OR NmPncBitVectorOffset + NmPncBitVectorLength == NM PduLength. | | |

| | | | |
|---|---|---|---|
| **Name** | NmPnEiraCalcEnabled [ECUC_Nm_00251] | | |
| **Parent Container** | NmGlobalProperties | | |
| **Description** | Specifies if NmIf calculates the PNC request information for internal and external requests (EIRA)<br><br>true: PN request are calculated<br><br>false: PN request are not calculated<br><br>**Tags:**<br>atp.Status=draft | | |
| **Multiplicity** | 0..1 | | |
| **Type** | EcucBooleanParamDef | | |
| **Default Value** | false | | |
| **Multiplicity Configuration Class** | **Pre-compile time** | X | VARIANT-PRE-COMPILE |
| | **Link time** | X | VARIANT-LINK-TIME |
| | **Post-build time** | – | |
| **Value Configuration Class** | **Pre-compile time** | X | VARIANT-PRE-COMPILE |
| | **Link time** | X | VARIANT-LINK-TIME |
| | **Post-build time** | – | |
| **Scope / Dependency** | scope: local<br>dependency: This parameter can only be set to TRUE if NmPartialNetworkSupportEnabled is set to TRUE. | | |

| | | | |
|---|---|---|---|
| **Name** | NmPnResetTime [ECUC_Nm_00250] | | |
| **Parent Container** | NmGlobalProperties | | |
| **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.<br><br>**Tags:**<br>atp.Status=draft | | |
| **Multiplicity** | 0..1 | | |
| **Type** | EcucFloatParamDef | | |
| **Range** | [0.001 .. 65535] | | |
| **Default Value** | | | |
| **Multiplicity Configuration Class** | **Pre-compile time** | X | VARIANT-PRE-COMPILE |
| | **Link time** | X | VARIANT-LINK-TIME |
| | **Post-build time** | – | |
| **Value Configuration Class** | **Pre-compile time** | X | VARIANT-PRE-COMPILE |
| | **Link time** | X | VARIANT-LINK-TIME |
| | **Post-build time** | – | |

| | |
|---|---|
| **Scope / Dependency** | scope: local<br>dependency: This parameter is only valid if NmPartialNetworkSupportEnabled is set to TRUE.<br><br>NmPnResetTime > <Can\|Udp>NmMsgCycleTime<br><br>NmPnResetTime > FrNmDataCycle * FR Cycle Time |

| | | | |
|---|---|---|---|
| **Name** | NmVersionInfoApi [ECUC_Nm_00204] | | |
| **Parent Container** | NmGlobalProperties | | |
| **Description** | Pre-processor switch for enabling Version Info API support. | | |
| **Multiplicity** | 1 | | |
| **Type** | EcucBooleanParamDef | | |
| **Default Value** | false | | |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time** | X | All Variants |
| | **Link time** | – | |
| | **Post-build time** | – | |
| **Scope / Dependency** | scope: local | | |

| |
|---|
| **No Included Containers** |

### 10.3.4 NmGlobalFeatures

| | |
|---|---|
| **SWS Item** | [ECUC_Nm_00200] |
| **Container Name** | NmGlobalFeatures |
| **Parent Container** | NmGlobalConfig |
| **Description** | |
| **Configuration Parameters** | |

| | | | |
|---|---|---|---|
| **Name** | NmBusSynchronizationEnabled [ECUC_Nm_00208] | | |
| **Parent Container** | NmGlobalFeatures | | |
| **Description** | Pre-processor switch for enabling bus synchronization support of the <BusNm>s. This feature is required for NM Coordinator nodes only. | | |
| **Multiplicity** | 1 | | |
| **Type** | EcucBooleanParamDef | | |
| **Default Value** | | | |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time** | X | All Variants |
| | **Link time** | – | |
| | **Post-build time** | – | |

| | |
|---|---|
| **Scope / Dependency** | scope: local<br>dependency: This parameter must be enabled if NmCoordinatorSupportEnabled is enabled. |

| | | | |
|---|---|---|---|
| **Name** | NmCarWakeUpCallout [ECUC_Nm_00234] | | |
| **Parent Container** | NmGlobalFeatures | | |
| **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. | | |
| **Multiplicity** | 0..1 | | |
| **Type** | EcucFunctionNameDef | | |
| **Default Value** | | | |
| **Regular Expression** | | | |
| **Post-Build Variant Multiplicity** | false | | |
| **Post-Build Variant Value** | false | | |
| **Multiplicity Configuration Class** | **Pre-compile time** | X | VARIANT-PRE-COMPILE |
| | **Link time** | X | VARIANT-LINK-TIME |
| | **Post-build time** | – | |
| **Value Configuration Class** | **Pre-compile time** | X | VARIANT-PRE-COMPILE |
| | **Link time** | X | VARIANT-LINK-TIME |
| | **Post-build time** | – | |
| **Scope / Dependency** | scope: local<br>dependency: only available if NmCarWakeUpRxEnabled == TRUE | | |

| | | | |
|---|---|---|---|
| **Name** | NmCarWakeUpRxEnabled [ECUC_Nm_00235] | | |
| **Parent Container** | NmGlobalFeatures | | |
| **Description** | Enables or disables CWU detection. FALSE - CarWakeUp not supported TRUE - CarWakeUp supported | | |
| **Multiplicity** | 1 | | |
| **Type** | EcucBooleanParamDef | | |
| **Default Value** | false | | |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time** | X | VARIANT-PRE-COMPILE |
| | **Link time** | X | VARIANT-LINK-TIME |
| | **Post-build time** | – | |
| **Scope / Dependency** | scope: local | | |

| | | | |
|---|---|---|---|
| **Name** | NmComControlEnabled [ECUC_Nm_00210] | | |
| **Parent Container** | NmGlobalFeatures | | |
| **Description** | Pre-processor switch for enabling the Communication Control support. | | |
| **Multiplicity** | 1 | | |
| **Type** | EcucBooleanParamDef | | |
| **Default Value** | | | |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time** | X | All Variants |
| | **Link time** | – | |
| | **Post-build time** | – | |
| **Scope / Dependency** | scope: local | | |

| | | | |
|---|---|---|---|
| **Name** | NmCoordinatorSupportEnabled [ECUC_Nm_00206] | | |
| **Parent Container** | NmGlobalFeatures | | |
| **Description** | Pre-processor switch for enabling NM Coordinator support. | | |
| **Multiplicity** | 1 | | |
| **Type** | EcucBooleanParamDef | | |
| **Default Value** | false | | |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time** | X | All Variants |
| | **Link time** | – | |
| | **Post-build time** | – | |
| **Scope / Dependency** | scope: local<br>dependency: Only valid if at least one NM channel exists which has NmPassiveModeEnabled set to FALSE. | | |

| | | | |
|---|---|---|---|
| **Name** | NmCoordinatorSyncSupport [ECUC_Nm_00240] | | |
| **Parent Container** | NmGlobalFeatures | | |
| **Description** | Enables/disables the coordinator synchronisation support. | | |
| **Multiplicity** | 1 | | |
| **Type** | EcucBooleanParamDef | | |
| **Default Value** | | | |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time** | X | VARIANT-PRE-COMPILE |
| | **Link time** | X | VARIANT-LINK-TIME |
| | **Post-build time** | – | |
| **Scope / Dependency** | scope: local<br>dependency: NmCoordinatorSyncSupport shall only be valid if NmCoordinatorSupportEnabled is TRUE. | | |

| | | | |
|---|---|---|---|
| **Name** | NmDynamicPncToChannelMappingSupport [ECUC_Nm_00246] | | |
| **Parent Container** | NmGlobalFeatures | | |
| **Description** | Precompile time switch to enable the dynamic PNC-to-channel-mapping handling.<br><br>False: Dynamic PNC-to-channel-mapping is disabled True: Dynamic PNC-to-channel-mapping is enabled<br><br>**Tags:**<br>atp.Status=draft | | |
| **Multiplicity** | 1 | | |
| **Type** | EcucBooleanParamDef | | |
| **Default Value** | | | |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time** | X | All Variants |
| | **Link time** | – | |
| | **Post-build time** | – | |
| **Scope / Dependency** | scope: local | | |

| | | | |
|---|---|---|---|
| **Name** | NmGlobalCoordinatorTime [ECUC_Nm_00237] | | |
| **Parent Container** | NmGlobalFeatures | | |
| **Description** | This parameter defines the maximum shutdown time of a connected and coordinated NM-Cluster. Note:This includes nested connections. | | |
| **Multiplicity** | 0..1 | | |
| **Type** | EcucFloatParamDef | | |
| **Range** | [0 .. INF] | | |
| **Default Value** | | | |
| **Post-Build Variant Multiplicity** | false | | |
| **Post-Build Variant Value** | false | | |
| **Multiplicity Configuration Class** | **Pre-compile time** | X | All Variants |
| | **Link time** | – | |
| | **Post-build time** | – | |
| **Value Configuration Class** | **Pre-compile time** | X | All Variants |
| | **Link time** | – | |
| | **Post-build time** | – | |
| **Scope / Dependency** | scope: local<br>dependency: NmGlobalCoordinatorTime shall only be valid if NmCoordinatorSupportEnabled is TRUE. | | |

| | | | |
|---|---|---|---|
| **Name** | NmPartialNetworkSupportEnabled [ECUC_Nm_00253] | | |
| **Parent Container** | NmGlobalFeatures | | |
| **Description** | Pre-processor switch for enabling the Nm Partial Network support.<br><br>**Tags:**<br>atp.Status=draft | | |
| **Multiplicity** | 0..1 | | |
| **Type**<br>**Default Value** | EcucBooleanParamDef<br>false | | |
| **Multiplicity Configuration Class** | **Pre-compile time** | X | VARIANT-PRE-COMPILE |
| | **Link time** | X | VARIANT-LINK-TIME |
| | **Post-build time** | – | |
| **Value Configuration Class** | **Pre-compile time** | X | VARIANT-PRE-COMPILE |
| | **Link time** | X | VARIANT-LINK-TIME |
| | **Post-build time** | – | |
| **Scope / Dependency** | scope: local<br>dependency: This parameter can only be set to TRUE if NmCoordinatorSupportEnabled is set to FALSE. | | |

| | | | |
|---|---|---|---|
| **Name** | NmPduRxIndicationEnabled [ECUC_Nm_00214] | | |
| **Parent Container** | NmGlobalFeatures | | |
| **Description** | Pre-processor switch for enabling the PDU Rx Indication. | | |
| **Multiplicity** | 1 | | |
| **Type**<br>**Default Value** | EcucBooleanParamDef<br> | | |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time** | X | All Variants |
| | **Link time** | – | |
| | **Post-build time** | – | |
| **Scope / Dependency** | scope: local | | |

| | | | |
|---|---|---|---|
| **Name** | NmRemoteSleepIndEnabled [ECUC_Nm_00207] | | |
| **Parent Container** | NmGlobalFeatures | | |
| **Description** | Pre-processor switch for enabling Remote Sleep Indication support. This feature is required for a Gateway or Nm Coordinator functionality.<br><br>Note that this feature should not be used if all NM channels have Passive Mode enabled. | | |
| **Multiplicity** | 1 | | |
| **Type** | EcucBooleanParamDef | | |
| **Default Value** | | | |
| **Post-Build Variant Value** | false | | |

| | | | |
|---|---|---|---|
| **Value Configuration Class** | **Pre-compile time** | X | All Variants |
| | **Link time** | – | |
| | **Post-build time** | – | |
| **Scope / Dependency** | scope: local<br>dependency: If NmCoordinatorSupportEnabled == TRUE then NmRemoteSleepIndEnabled = TRUE | | |

| | | | |
|---|---|---|---|
| **Name** | NmStateChangeIndEnabled [ECUC_Nm_00215] | | |
| **Parent Container** | NmGlobalFeatures | | |
| **Description** | Pre-processor switch for enabling the Network Management state change notification. | | |
| **Multiplicity** | 1 | | |
| **Type** | EcucBooleanParamDef | | |
| **Default Value** | | | |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time** | X | All Variants |
| | **Link time** | – | |
| | **Post-build time** | – | |
| **Scope / Dependency** | scope: local<br>dependency: NmStateChangeIndEnabled = TRUE if NmDynamicPncToChannelMappingSupport == TRUE | | |

| | | | |
|---|---|---|---|
| **Name** | NmSynchronizedPncShutdownEnabled [ECUC_Nm_00249] | | |
| **Parent Container** | NmGlobalFeatures | | |
| **Description** | Enables or disables support of synchronized PNC shutdown.<br><br>FALSE: synchronized PNC shutdown is disabled<br><br>TRUE: synchronized PNC shutdown is enabled<br><br>**Tags:**<br>atp.Status=draft | | |
| **Multiplicity** | 0..1 | | |
| **Type** | EcucBooleanParamDef | | |
| **Default Value** | false | | |
| **Multiplicity Configuration Class** | **Pre-compile time** | X | VARIANT-PRE-COMPILE |
| | **Link time** | X | VARIANT-LINK-TIME |
| | **Post-build time** | – | |
| **Value Configuration Class** | **Pre-compile time** | X | VARIANT-PRE-COMPILE |
| | **Link time** | X | VARIANT-LINK-TIME |
| | **Post-build time** | – | |

| | |
|---|---|
| **Scope / Dependency** | scope: local<br>dependency: DRAFT:<br><br>This parameter can only be set to TRUE if NmPartialNetworkSupportEnabled is set to TRUE. |

| | | | |
|---|---|---|---|
| **Name** | NmUserDataEnabled [ECUC_Nm_00211] | | |
| **Parent Container** | NmGlobalFeatures | | |
| **Description** | Pre-processor switch for enabling User Data support. | | |
| **Multiplicity** | 1 | | |
| **Type** | EcucBooleanParamDef | | |
| **Default Value** | | | |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time** | X | All Variants |
| | **Link time** | – | |
| | **Post-build time** | – | |
| **Scope / Dependency** | scope: local | | |

| |
|---|
| **No Included Containers** |

# 10.4 Channel configurable parameters

## 10.4.1 NmChannelConfig

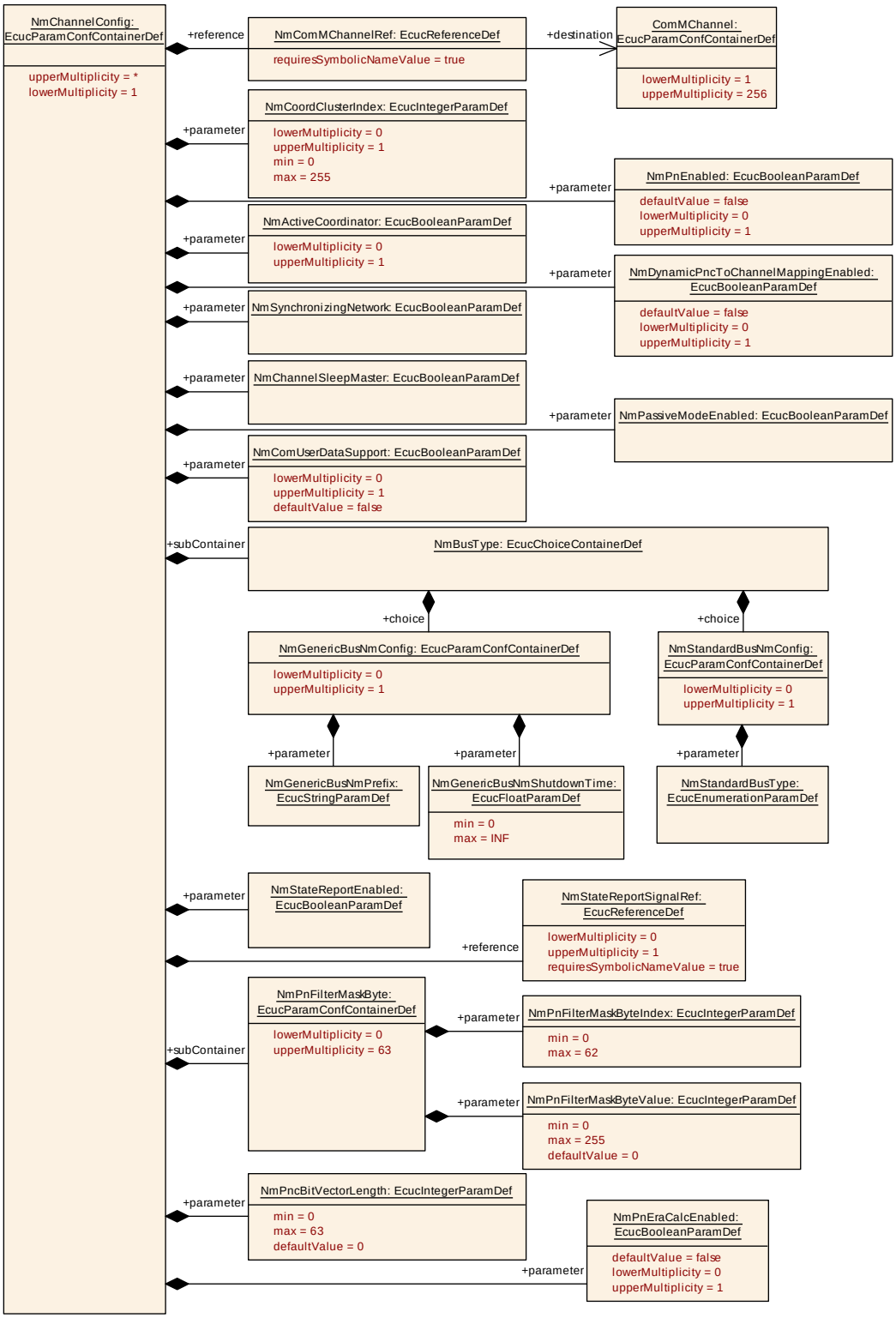


**Figure 10.4: NmChannelConfig overview**

| SWS Item | [ECUC_Nm_00197] |
|---|---|
| Container Name | NmChannelConfig |
| Parent Container | Nm |
| Description | This container contains the configuration (parameters) of the bus channel(s). The channel parameter shall be harmonized within the whole communication stack. |
| Configuration Parameters | |

| Name | NmActiveCoordinator [ECUC_Nm_00236] | | |
|---|---|---|---|
| Parent Container | NmChannelConfig | | |
| Description | This parameter indicates whether a NM channel - part of a Nm Coordination cluster - will be coordinated actively (NmActiveCoordinator = TRUE) or passively (NmActiveCoordinator = FALSE). | | |
| Multiplicity | 0..1 | | |
| Type | EcucBooleanParamDef | | |
| Default Value | | | |
| Post-Build Variant Multiplicity | false | | |
| Post-Build Variant Value | false | | |
| Multiplicity Configuration Class | Pre-compile time | X | All Variants |
| | Link time | – | |
| | Post-build time | – | |
| Value Configuration Class | Pre-compile time | X | All Variants |
| | Link time | – | |
| | Post-build time | – | |
| Scope / Dependency | scope: local<br>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). | | |

| | | | |
|---|---|---|---|
| **Name** | NmChannelSleepMaster [ECUC_Nm_00227] | | |
| **Parent Container** | NmChannelConfig | | |
| **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.<br><br>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_RemoteSleepCancellation will be made from the <BusNm> representing this channel.<br><br>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. | | |
| **Multiplicity** | 1 | | |
| **Type** | EcucBooleanParamDef | | |
| **Default Value** | | | |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time** | X | VARIANT-PRE-COMPILE |
| | **Link time** | X | VARIANT-LINK-TIME |
| | **Post-build time** | – | |
| **Scope / Dependency** | scope: local<br>dependency: If the parameter NmCoordClusterIndex is not defined, this parameter is not valid. | | |

| | | | |
|---|---|---|---|
| **Name** | NmComUserDataSupport [ECUC_Nm_00241] | | |
| **Parent Container** | NmChannelConfig | | |
| **Description** | This parameter indicates whether on a NM channel user data is accessed via Com signals or by SetUserData API. | | |
| **Multiplicity** | 0..1 | | |
| **Type** | EcucBooleanParamDef | | |
| **Default Value** | false | | |
| **Post-Build Variant Multiplicity** | false | | |
| **Post-Build Variant Value** | false | | |
| **Multiplicity Configuration Class** | **Pre-compile time** | X | All Variants |
| | **Link time** | – | |
| | **Post-build time** | – | |
| **Value Configuration Class** | **Pre-compile time** | X | All Variants |
| | **Link time** | – | |
| | **Post-build time** | – | |
| **Scope / Dependency** | scope: local<br>dependency: NmComUserDataSupport shall be equal to <Bus>NmComUserDataSupport | | |

| | | | |
|---|---|---|---|
| **Name** | NmCoordClusterIndex [ECUC_Nm_00221] | | |
| **Parent Container** | NmChannelConfig | | |
| **Description** | If this parameter is undefined for a channel, the corresponding bus does not belong to an NM coordination cluster. | | |
| **Multiplicity** | 0..1 | | |
| **Type** | EcucIntegerParamDef | | |
| **Range** | 0 .. 255 | | |
| **Default Value** | | | |
| **Post-Build Variant Multiplicity** | false | | |
| **Post-Build Variant Value** | false | | |
| **Multiplicity Configuration Class** | **Pre-compile time** | X | VARIANT-PRE-COMPILE |
| | **Link time** | X | VARIANT-LINK-TIME |
| | **Post-build time** | – | |
| **Value Configuration Class** | **Pre-compile time** | X | VARIANT-PRE-COMPILE |
| | **Link time** | X | VARIANT-LINK-TIME |
| | **Post-build time** | – | |
| **Scope / Dependency** | scope: local<br>dependency: If NmCoordClusterIndex is defined than NmPassiveModeEnabled has to be FALSE for this channel. | | |

| | | | |
|---|---|---|---|
| **Name** | NmDynamicPncToChannelMappingEnabled [ECUC_Nm_00248] | | |
| **Parent Container** | NmChannelConfig | | |
| **Description** | Channel-specific parameter to enable the dynamic PNC-to-channel-mapping feature.<br><br>False: Dynamic PNC-to-channel-mapping is disabled True: Dynamic PNC-to-channel-mapping is enabled<br><br>**Tags:**<br>atp.Status=draft | | |
| **Multiplicity** | 0..1 | | |
| **Type** | EcucBooleanParamDef | | |
| **Default Value** | false | | |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time** | X | VARIANT-PRE-COMPILE |
| | **Link time** | X | VARIANT-LINK-TIME |
| | **Post-build time** | – | |
| **Scope / Dependency** | scope: local<br>dependency: Shall only be TRUE if NmDynamicPncToChannelMappingSupport is TRUE | | |

| | | | |
|---|---|---|---|
| **Name** | NmPassiveModeEnabled [ECUC_Nm_00242] | | |
| **Parent Container** | NmChannelConfig | | |
| **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. | | |
| **Multiplicity** | 1 | | |
| **Type** | EcucBooleanParamDef | | |
| **Default Value** | | | |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time** | X | All Variants |
| | **Link time** | – | |
| | **Post-build time** | – | |
| **Scope / Dependency** | scope: local<br>dependency: if ComMNmVariant == FULL then NmPassiveModeEnabled = FALSE; NmPassiveModeEnabled shall be equal to <Bus>NmPassiveModeEnabled | | |

| | | | |
|---|---|---|---|
| **Name** | NmPncBitVectorLength [ECUC_Nm_00258] | | |
| **Parent Container** | NmChannelConfig | | |
| **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.<br><br>**Tags:**<br>atp.Status=draft | | |
| **Multiplicity** | 1 | | |
| **Type** | EcucIntegerParamDef | | |
| **Range** | 0 .. 63 | | |
| **Default Value** | 0 | | |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time** | X | VARIANT-PRE-COMPILE |
| | **Link time** | X | VARIANT-LINK-TIME |
| | **Post-build time** | – | |
| **Scope / Dependency** | scope: ECU<br>dependency: This parameter is only valid if NmPartialNetworkSupportEnabled is set to TRUE. | | |

| | |
|---|---|
| **Name** | NmPnEnabled [ECUC_Nm_00254] |
| **Parent Container** | NmChannelConfig |
| **Description** | If this parameter is true, then this NM channel supports Partial Networking.<br><br>**Tags:**<br>atp.Status=draft |
| **Multiplicity** | 0..1 |
| **Type** | EcucBooleanParamDef |
| **Default Value** | false |

| | | | |
|---|---|---|---|
| **Multiplicity Configuration Class** | **Pre-compile time** | X | VARIANT-PRE-COMPILE |
| | **Link time** | X | VARIANT-LINK-TIME |
| | **Post-build time** | – | |
| **Value Configuration Class** | **Pre-compile time** | X | VARIANT-PRE-COMPILE |
| | **Link time** | X | VARIANT-LINK-TIME |
| | **Post-build time** | – | |
| **Scope / Dependency** | scope: local<br>dependency: This parameter is only valid if NmPartialNetworkSupportEnabled is set to TRUE. | | |

| | | | |
|---|---|---|---|
| **Name** | NmPnEraCalcEnabled [ECUC_Nm_00259] | | |
| **Parent Container** | NmChannelConfig | | |
| **Description** | Specifies if NmIf calculates the PN request information for external requests. (ERA)<br><br>**Tags:**<br>atp.Status=draft | | |
| **Multiplicity** | 0..1 | | |
| **Type** | EcucBooleanParamDef | | |
| **Default Value** | false | | |
| **Multiplicity Configuration Class** | **Pre-compile time** | X | VARIANT-PRE-COMPILE |
| | **Link time** | X | VARIANT-LINK-TIME |
| | **Post-build time** | – | |
| **Value Configuration Class** | **Pre-compile time** | X | VARIANT-PRE-COMPILE |
| | **Link time** | X | VARIANT-LINK-TIME |
| | **Post-build time** | – | |
| **Scope / Dependency** | scope: local<br>dependency: This parameter is only valid if NmPartialNetworkSupportEnabled is set to TRUE. | | |

| | | | |
|---|---|---|---|
| **Name** | NmStateReportEnabled [ECUC_Nm_00231] | | |
| **Parent Container** | NmChannelConfig | | |
| **Description** | Specifies if the NMS shall be set for the corresponding network. false: No NMS shall be set true: The NMS shall be set | | |
| **Multiplicity** | 1 | | |
| **Type** | EcucBooleanParamDef | | |
| **Default Value** | | | |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time** | X | All Variants |
| | **Link time** | – | |
| | **Post-build time** | – | |
| **Scope / Dependency** | scope: local<br>dependency: only available if NmStatChangeIndEnabled and NmComUserDataSupport are configured to TRUE. | | |

| | | | |
|---|---|---|---|
| **Name** | NmSynchronizingNetwork [ECUC_Nm_00223] | | |
| **Parent Container** | NmChannelConfig | | |
| **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. | | |
| **Multiplicity** | 1 | | |
| **Type** | EcucBooleanParamDef | | |
| **Default Value** | | | |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time** | X | VARIANT-PRE-COMPILE |
| | **Link time** | X | VARIANT-LINK-TIME |
| | **Post-build time** | – | |
| **Scope / Dependency** | scope: local<br>dependency: If the parameter NmCoordClusterIndex is not defined, this parameter is not valid. Only one network can be configured as synchronizing network (NmSynchronizingNetwork = TRUE) per coordination cluster (same NmCoordClusterIndex value per channel). NmSynchronizingNetwork can only be set to true if NmActiveCoordinator is true for all networks which have the same NmCoordClusterIndex. | | |

| | | | |
|---|---|---|---|
| **Name** | NmComMChannelRef [ECUC_Nm_00217] | | |
| **Parent Container** | NmChannelConfig | | |
| **Description** | Reference to the corresponding ComM Channel. | | |
| **Multiplicity** | 1 | | |
| **Type** | Symbolic name reference to ComMChannel | | |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time** | X | VARIANT-PRE-COMPILE |
| | **Link time** | X | VARIANT-LINK-TIME |
| | **Post-build time** | – | |
| **Scope / Dependency** | scope: local | | |

| | | | |
|---|---|---|---|
| **Name** | NmStateReportSignalRef [ECUC_Nm_00232] | | |
| **Parent Container** | NmChannelConfig | | |
| **Description** | Reference to the signal for setting the NMS by calling Com_SendSignal for the respective channel. | | |
| **Multiplicity** | 0..1 | | |
| **Type** | Symbolic name reference to ComSignal | | |
| **Post-Build Variant Multiplicity** | false | | |
| **Post-Build Variant Value** | false | | |
| **Multiplicity Configuration Class** | **Pre-compile time** | X | All Variants |
| | **Link time** | – | |
| | **Post-build time** | – | |

| | | | |
|---|---|---|---|
| **Value Configuration Class** | **Pre-compile time** | X | All Variants |
| | **Link time** | – | |
| | **Post-build time** | – | |
| **Scope / Dependency** | scope: local<br>dependency: Signal must be configured in COM. Only available if NmStateReportEnabled == true | | |

| **Included Containers** | | |
|---|---|---|
| **Container Name** | **Multiplicity** | **Scope / Dependency** |
| NmBusType | 1 | |
| NmPnFilterMaskByte | 0..63 | Information for the filter of the PNC bit vector.<br><br>**Tags:**<br>atp.Status=draft |

### 10.4.2 NmPnFilterMaskByte

| | | | |
|---|---|---|---|
| **SWS Item** | [ECUC_Nm_00255] | | |
| **Container Name** | NmPnFilterMaskByte | | |
| **Parent Container** | NmChannelConfig | | |
| **Description** | Information for the filter of the PNC bit vector.<br><br>**Tags:**<br>atp.Status=draft | | |
| **Multiplicity Configuration Class** | **Pre-compile time** | X | VARIANT-PRE-COMPILE |
| | **Link time** | X | VARIANT-LINK-TIME |
| | **Post-build time** | – | |
| **Configuration Parameters** | | | |

| | | | |
|---|---|---|---|
| **Name** | NmPnFilterMaskByteIndex [ECUC_Nm_00256] | | |
| **Parent Container** | NmPnFilterMaskByte | | |
| **Description** | Index of the filter mask byte. Specifies the position within the filter mask byte array.<br><br>**Tags:**<br>atp.Status=draft | | |
| **Multiplicity** | 1 | | |
| **Type** | EcucIntegerParamDef | | |
| **Range** | 0 .. 62 | | |
| **Default Value** | | | |
| **Value Configuration Class** | **Pre-compile time** | X | VARIANT-PRE-COMPILE |
| | **Link time** | X | VARIANT-LINK-TIME |
| | **Post-build time** | – | |

| | |
|---|---|
| **Scope / Dependency** | scope: local<br>dependency: This parameter is only valid if NmPartialNetworkSupportEnabled is set to TRUE, NmPnFilterMaskByteIndex < NmPncBitVectorLength. |

| | | | |
|---|---|---|---|
| **Name** | NmPnFilterMaskByteValue [ECUC_Nm_00257] | | |
| **Parent Container** | NmPnFilterMaskByte | | |
| **Description** | Parameter to configure the filter mask byte.<br><br>**Tags:**<br>atp.Status=draft | | |
| **Multiplicity** | 1 | | |
| **Type** | EcucIntegerParamDef | | |
| **Range** | 0 .. 255 | | |
| **Default Value** | 0 | | |
| **Value Configuration Class** | **Pre-compile time** | X | VARIANT-PRE-COMPILE |
| | **Link time** | X | VARIANT-LINK-TIME |
| | **Post-build time** | – | |
| **Scope / Dependency** | scope: local<br>dependency: This parameter is only valid if NmPartialNetworkSupportEnabled is set to TRUE. | | |

| |
|---|
| **No Included Containers** |

### 10.4.3 NmBusType

| | |
|---|---|
| **SWS Item** | [ECUC_Nm_00218] |
| **Container Name** | NmBusType |
| **Parent Container** | NmChannelConfig |
| **Description** | |
| **Configuration Parameters** | |

| **Container Choices** | | |
|---|---|---|
| **Container Name** | **Multiplicity** | **Scope / Dependency** |
| NmGenericBusNmConfig | 0..1 | |
| NmStandardBusNm Config | 0..1 | |

### 10.4.4 NmGenericBusNmConfig

| | |
|---|---|
| **SWS Item** | [ECUC_Nm_00225] |
| **Container Name** | NmGenericBusNmConfig |
| **Parent Container** | NmBusType |
| **Description** | |
| **Configuration Parameters** | |

| | | | |
|---|---|---|---|
| **Name** | NmGenericBusNmPrefix [ECUC_Nm_00219] | | |
| **Parent Container** | NmGenericBusNmConfig | | |
| **Description** | The prefix which identifies the generic <BusNm>. This will be used to determine the API name to be called by Nm for the provided interfaces of the <BusNm>. This string will used for the module prefix before the "_" character in the API call name. | | |
| **Multiplicity** | 1 | | |
| **Type** | EcucStringParamDef | | |
| **Default Value** | | | |
| **Regular Expression** | | | |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time** | X | VARIANT-PRE-COMPILE |
| | **Link time** | X | VARIANT-LINK-TIME |
| | **Post-build time** | – | |
| **Scope / Dependency** | scope: local | | |

| | | | |
|---|---|---|---|
| **Name** | NmGenericBusNmShutdownTime [ECUC_Nm_00239] | | |
| **Parent Container** | NmGenericBusNmConfig | | |
| **Description** | This parameter shall be used to calculate shutdown delay time. | | |
| **Multiplicity** | 1 | | |
| **Type** | EcucFloatParamDef | | |
| **Range** | [0 .. INF] | | |
| **Default Value** | | | |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time** | X | VARIANT-PRE-COMPILE |
| | **Link time** | X | VARIANT-LINK-TIME |
| | **Post-build time** | – | |
| **Scope / Dependency** | scope: local | | |

| |
|---|
| **No Included Containers** |

### 10.4.5 NmStandardBusNmConfig

| | |
|---|---|
| **SWS Item** | [ECUC_Nm_00226] |
| **Container Name** | NmStandardBusNmConfig |
| **Parent Container** | NmBusType |
| **Description** | |
| **Configuration Parameters** | |

| | | | |
|---|---|---|---|
| **Name** | NmStandardBusType [ECUC_Nm_00220] | | |
| **Parent Container** | NmStandardBusNmConfig | | |
| **Description** | Identifies the bus type of the channel for standard AUTOSAR <BusNm>s and is used to determine which set of API calls to be called by Nm for the <BusNm>s. Note: The Ethernet bus' NM is UdpNm ! | | |
| **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) | |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time** | X | VARIANT-PRE-COMPILE |
| | **Link time** | X | VARIANT-LINK-TIME |
| | **Post-build time** | – | |
| **Scope / Dependency** | scope: local<br>dependency: Configuring value to NM_BUSNM_LOCALNM is only allowed if NmCoordClusterIndex for the corresponding channel is defined (i.e channel is coordinated). | | |

**No Included Containers**

## 10.5 Published Information

For details refer to the chapter 10.3 "Published Information" in [2, SWS_BSWGeneral].

# A Not applicable requirements

**[SWS_Nm_00999] Not applicable requirements** ⌈These requirements are not applicable to this specification.⌋ *(RS_Nm_00043, RS_Nm_00137, RS_Nm_00142, RS_Nm_00144, RS_Nm_00145, RS_Nm_00146, RS_Nm_00152, RS_Nm_02517, RS_Nm_02550, RS_Nm_02519, RS_Nm_02527, RS_Nm_02528, SRS_BSW_00004, SRS_BSW_00005, SRS_BSW_00006, SRS_BSW_00007, SRS_BSW_00009, SRS_BSW_00010, SRS_BSW_00160, SRS_BSW_00161, SRS_BSW_00162, SRS_BSW_00164, SRS_BSW_00167, SRS_BSW_00168, SRS_BSW_00170, SRS_BSW_00172, SRS_BSW_00300, SRS_BSW_00302, SRS_BSW_00304, SRS_BSW_00305, SRS_BSW_00306, SRS_BSW_00307, SRS_BSW_00308, SRS_BSW_00309, SRS_BSW_00310, SRS_BSW_00312, SRS_BSW_00314, SRS_BSW_00318, SRS_BSW_00321, SRS_BSW_00325, SRS_BSW_00328, SRS_BSW_00331, SRS_BSW_00334, SRS_BSW_00335, SRS_BSW_00336, SRS_BSW_00339, SRS_BSW_00341, SRS_BSW_00342, SRS_BSW_00343, SRS_BSW_00346, SRS_BSW_00347, SRS_BSW_00350, SRS_BSW_00351, SRS_BSW_00360, SRS_BSW_00361, SRS_BSW_00374, SRS_BSW_00375, SRS_BSW_00377, SRS_BSW_00378, SRS_BSW_00379, SRS_BSW_00380, SRS_BSW_00383, SRS_BSW_00388, SRS_BSW_00389, SRS_BSW_00390, SRS_BSW_00392, SRS_BSW_00393, SRS_BSW_00394, SRS_BSW_00395, SRS_BSW_00397, SRS_BSW_00398, SRS_BSW_00399, SRS_BSW_00400, SRS_BSW_00401, SRS_BSW_00402, SRS_BSW_00403, SRS_BSW_00404, SRS_BSW_00406, SRS_BSW_00408, SRS_BSW_00409, SRS_BSW_00410, SRS_BSW_00411, SRS_BSW_00413, SRS_BSW_00415, 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_00439, SRS_BSW_00440, SRS_BSW_00441, SRS_BSW_00447, SRS_BSW_00448, SRS_BSW_00449, SRS_BSW_00451, SRS_BSW_00452, SRS_BSW_00453, SRS_BSW_00454, SRS_BSW_00456, SRS_BSW_00457, SRS_BSW_00458, SRS_BSW_00460, SRS_BSW_00461, SRS_BSW_00462, SRS_BSW_00463, SRS_BSW_00464, SRS_BSW_00465, SRS_BSW_00466, SRS_BSW_00467, SRS_BSW_00469, SRS_BSW_00470, SRS_BSW_00471, SRS_BSW_00472, SRS_BSW_00473, SRS_BSW_00396, SRS_BSW_00477, SRS_BSW_00479, SRS_BSW_00480, SRS_BSW_00481)*