| Document Title | Specification of CAN Generic Network Management |
|---|---|
| **Document Owner** | AUTOSAR GbR |
| **Document Responsibility** | AUTOSAR GbR |
| **Document Version** | 2.0.0 |
| **Document Status** | Draft |
| **Part of Release** | 2.1 |
| **Revision** | 0014 |

| Document Change History | | | |
|---|---|---|---|
| **Date** | **Version** | **Changed by** | **Change Description** |
| 31.01.2007 | 2.0.0 | AUTOSAR Administration | • Change of Module name from Generic NM to CAN Generic NM<br>• Change of Module prefix from 'Nm' to 'CNm'<br>• Change of architecture in NM Stack<br>• Rework of the whole API<br>• Minor changes of state machine<br>• Passive Mode added<br>• Communication Control added<br><br>• Legal disclaimer revised<br>• Release Notes added<br>• “Advice for users” revised<br>• “Revision Information” added |
| 23.06.2005 | 1.0.0 | AUTOSAR Administration | Initial Release |

# Release Notes

## Errata and known deficiencies

All modifications planned in the scope of Release 2.1 for the incorporation into this document are completed. The document, however, has not yet undergone the necessary finalization.

## Disclaimer



## Advice to users of AUTOSAR Specification Documents:

AUTOSAR Specification Documents may contain exemplary items (exemplary reference models, "use cases", and/or references to exemplary technical solutions, devices, processes or software).
Any such exemplary items are contained in the Specification Documents for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such Specification Documents, nor any later documentation of AUTOSAR conformance of products actually implementing such exemplary items, imply that intellectual property rights covering such exemplary items are licensed under the same rules as applicable to the AUTOSAR Standard.

# Table of Contents

# 1 Introduction and functional overview

This document describes the concept, core functionality, optional features, interfaces and configuration issues of the AUTOSAR CAN Generic Network Management (CNm).

The AUTOSAR CAN Generic Network Management is a hardware independent protocol that can only be used on CAN (for limitations see 4.1). Its main purpose is to coordinate the transition between normal operation and bus-sleep mode of the network.

In addition to the core functionality optional features are provided e.g. to implement a service to detect all present nodes or to detect if all other nodes are ready to sleep.

Adaptations of the AUTOSAR CAN Generic Network Management to the CAN bus are described in a separate specification (see [5]).

# 2 Acronyms, abbreviations and glossary

All acronyms and abbreviations used in this document are defined in SWS Generic NM Interface [6].

# 3 Related documentation

## 3.1 Input documents

[1] Layered Software Architecture
https://svn.autosar.org/repos/10Releases/
AUTOSAR_LayeredSoftwareArchitecture.pdf

[2] General Requirements on Basic Software Modules
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SRS_Genernal.pdf

[3] Requirement on Network Management
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SRS_NM.pdf

## 3.2 Related standards and norms

[4] OSEK/VDX NM Specification (ISO 17356-5), Version 2.5.3
http://www.osek-vdx.org/

## 3.3 Related AUTOSAR documents

[5] Specification of CAN Network Management,
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_CAN_NM.pdf

[6] Specification of Generic Network Management Interface
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_NMInterface.pdf

[7] Specification of Communication Manager
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_ComManager.pdf

[8] Specification of ECU State Manager
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_ECU_StateManager.pdf

[9] Specification of Operating System
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_OS.pdf

[10] Specification of Diagnostics Event Manager
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_DEM.pdf

[11] Specification of Development Error Tracer
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_DET.pdf

[12] Specification of Standard Types
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_StandardTypes.pdf

[13] Specification of Platform Types
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_PlatformTypes.pdf

[14] Specification of Compiler Abstraction
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_CompilerAbstraction.pdf

# 4 Constraints and assumptions

## 4.1 Limitations

1. One instance of CAN Generic NM is associated only one NM-Cluster in one network. One NM-Cluster can have only one instance of CAN Generic NM in one node.
2. One instance of CAN Generic NM is associated only one instance of Can-Interface within the same ECU.
3. The AUTOSAR CAN Generic NM can be applied to CAN only.

The
Figure 4-1 presents an AUTOSAR NM stack within an example ECU belonging to two CAN NM-clusters.

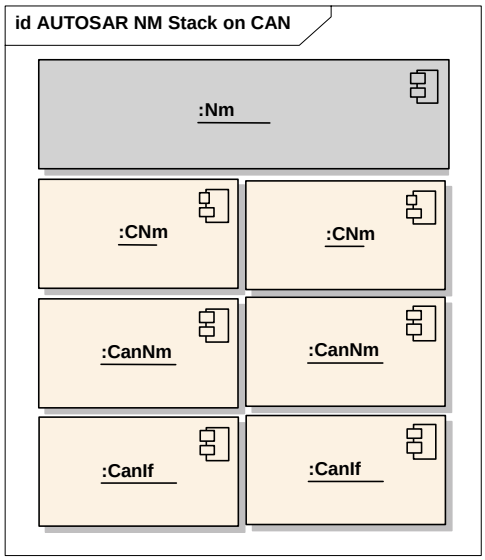


**Figure 4-1**

## 4.2 Applicability to car domains

AUTOSAR NM can be applied to any car domain under limitations provided above.

# 5 Dependencies to other modules

CAN Generic Network Management (CNm) uses services of the corresponding CAN specific NM adaptation layer (CanNm) and provides services to the Network Management Interface (Nm).

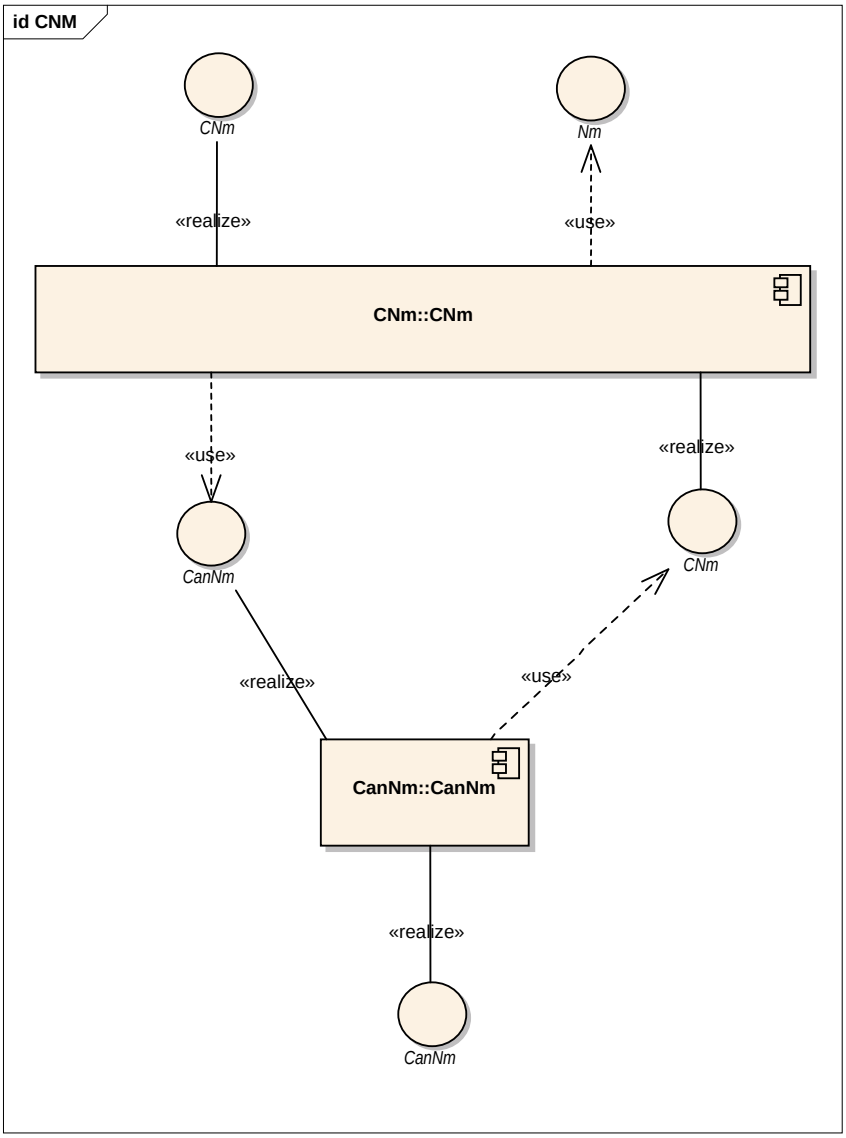


**Figure 5-1**

## 5.1 File structure

### 5.1.1 Code file structure

**CNM006:** The following C-files shall be provided by the CNm module.

- `CNm.c` (for implementation of provided functionality)
- `CNm_Lcfg.c` (for link time configurable parameters)
- `CNm_PBcfg.c` (for post build time configurable parameters)

### 5.1.2 Header file structure

**CNM007:** The following H-files shall be provided by the CNm module.

- `CNm.h` (for declaration of provided interface functions)
- `CNm_Cbk.h` (for declaration of provided call-back functions)
- `CNm_Cfg.h` (for pre-compile time configurable parameters)

**CNM008:** The following H-files shall be included by the CNm 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)
- `Nm_StackTypes.h` (for NM stack type definitions)
- `CanNm.h` (for interface of CAN specific NM)
- `Nm_Cbk.h` (for CAN Generic NM specific callbacks of NM Interface)
- `Det.h` (for interface of DET – optional, included only if Det is configured)
- `Dem.h` (for interface of DEM)
- `SchM_CNm.h` (for interface to Schedule Manager)

**CNM009:** The following configuration files shall be included within the CNm module.

- `CanNm_Cfg.h` (for configuration of CAN specific NM)
- `Det_Cfg.h` (for configuration of DET – optional, included only if Det is configured)
- `Dem_Cfg.h` (for configuration of DEM)

# 6 Requirements traceability

Document: General Requirements on Basic Software Modules [1] (Baseline: V1.10)

| ***Requirement*** | ***Satisfied by*** |
|---|---|
| 4.2 Functional Requirements | |
| 4.2.1 Configuration | |
| [BSW00344] Reference to link-time configuration | OK, see 8.3.1 |
| [BSW00404] Reference to post build time configuration | OK, see 8.3.1 |
| [BSW00405] Reference to multiple configuration sets | OK, see 8.3.1 |
| [BSW00345] Storage of Pre-compile-time configuration | OK, see CNM143 |
| [BSW159] Tool-based configuration | OK, see CNM136 |
| [BSW167] Static configuration checking | OK, see 10 |
| [BSW171] Configurability of optional functionality | OK, see 10 |
| [BSW170] Data for reconfiguration of AUTOSAR SW-Components | OK, SWS template used |
| [BSW00380] Separate C-Files for configuration parameters | OK, see CNM006 |
| [BSW00381] Separate H-File for configuration parameters | OK, see CNM007 |
| [BSW00382] Not-used configuration elements need to be listed | N/A, implementation specific |
| [BSW00383] List dependencies of configuration files | OK, see 5 |
| [BSW00384] List dependencies to other modules | OK, see 5 |
| [BSW00385] Configuration of error notifications | OK, see 7.9 |
| [BSW00386] How errors shall be detected | OK, see 7.9 |
| [BSW00387] Specify the configuration class of callback function | OK, see 8 |
| [BSW00388] Introduce containers | OK, see 10 |
| [BSW00389] Containers shall have names | OK, see 10 |
| [BSW00390] Parameter content shall be unique within the module | OK, see 10 |
| [BSW00391] Parameter shall have unique names | OK, see 10 |
| [BSW00392] Parameters shall have a type | OK, see 10 |
| [BSW00393] Parameters shall have a range | OK, see 10 |
| [BSW00394] Specify the scope of the parameters | OK, see 10 |
| [BSW00395] List the required parameters (per parameter) | OK, see 10 |
| [BSW00396] Configuration classes | OK, see 10 |
| [BSW00397] Pre-compile-time parameters | OK, see 10 |
| [BSW00398] Link-time parameters | OK, see 10 |
| [BSW00399] Loadable Post-build time parameters | OK, see 10 |
| [BSW00400] Selectable Post-build time parameters | OK, see 10 |
| [BSW00401] Creating multiplicity | OK, see 10 |
| [BSW00402] Published information | OK, see 10 |
| 4.2.2 Wake-Up | |
| [BSW00375] Notification of wake-up reason | N/A, Bus-Interface is responsible for notification of the wakeup reason. |
| 4.2.3 Initialization | |
| [BSW101] Initialization interface | OK, see CNM057, CNM064 |
| [BSW00406] Check module initialization | OK, see CNM055, CNM056 |
| 4.2.4 Normal Operation | |
| [BSW168] Diagnostic Interface of SW components | N/A, CAN Generic NM does not need to be tested by external devices. |
| [BSW00407] Function to read out published parameters | OK, see 8.3.14 |
| 4.2.5 Shutdown Operation | |
| [BSW00336] Shutdown interface | N/A, CAN Generic NM coordinates shutdown of |

| | |
|---|---|
| | bus-communication. |
| 4.2.6 Fault Operation and Error Detection | |
| [BSW00337] Classification of errors | OK, see CNM100 |
| [BSW00338] Detection and Reporting of development errors | OK, see CNM097, CNM195 |
| [BSW00369] Do not return development error codes via API | OK, seeCNM196, CNM98 |
| [BSW00339] Reporting of production relevant errors and exceptions | OK, see CNM099 |
| [BSW00323] API parameter checking | OK, see CNM131 |
| [BSW004] Version check | OK, seeCNM139, CNM135 |
| [BSW00409] Header files for production code error IDs | OK, see 5 |
| 4.3 Non-functional Requirements | |
| 4.3.1 Software Architecture Requirements | |
| [BSW161] Microcontroller abstraction | N/A, already abstracted |
| [BSW162] ECU layout abstraction | N/A, already abstracted |
| [BSW005] No hard coded horizontal interfaces within MCAL | N/A, CAN Generic NM is not located within MCAL |
| [BSW00166] BSW Module interfaces | OK, see 5 |
| 4.3.2 Software Integration Requirements | |
| [BSW164] Implementation of interrupt service routines | N/A, no interrupt routine implemented |
| [BSW00325] Runtime of interrupt service routines | N/A, no interrupt routine implemented |
| [BSW00326] Transition from ISRs to OS tasks | N/A, no interrupt routine implemented |
| [BSW00342] Usage of source code and object code | OK, see 10 |
| [BSW00343] Specification and configuration of time | OK, see 10 |
| [BSW160] Human-readable configuration data | OK, see 10 |
| 4.3.3 Software Module Design Requirements | |
| 4.3.3.1 Software quality | |
| [BSW007] HIS MISRA C | OK, HIS MISRA C is used |
| 4.3.3.2 Naming conventions | |
| [BSW00300] Module naming convention | OK, naming convention used respectively |
| [BSW00347] Naming separation of drivers | N/A, CAN Generic NM is no driver module |
| [BSW00305] Self-defined data types naming convention | OK, naming convention used respectively |
| [BSW00307] Global variables naming convention | OK, naming convention used respectively |
| [BSW00310] API naming convention | OK, naming convention used respectively |
| [BSW00373] Main processing function naming convention | OK, naming convention used respectively |
| [BSW00327] Error values naming convention | OK, naming convention used respectively |
| [BSW00335] Status values naming convention | OK, naming convention used respectively |
| [BSW00350] Development error detection keyword | OK, keyword used |
| [BSW00408] Configuration parameter naming convention | OK, see 10 |
| [BSW00410] Compiler switches shall have defined values | OK, see 10 |
| [BSW00411] Get version info keyword | OK, see 10 |
| 4.3.3.3 Module file structure | |
| [BSW00346] Basic set of module files | OK, see 5 |
| [BSW158] Separation of configuration from implementation | OK, see 5 and 10 |
| [BSW00314] Separation of interrupt frames and service routines | N/A, no interrupt frames implemented |
| [BSW00370] Separation of callback interface from API | OK, see CNM007 |

| | |
|---|---|
| 4.3.3.4 Standard header files | |
| [BSW00348] Standard type header | OK, see 5 |
| [BSW00353] Platform specific type header | OK, see 5 |
| [BSW00361] Compiler specific language extension header | OK, see 5 |
| 4.3.3.5 Module Design | |
| [BSW00301] Limit imported information | OK, see 5 |
| [BSW00302] Limit exported information | OK, see 5 |
| [BSW00328] Avoid duplication of code | OK, see 8 |
| [BSW00312] Shared code shall be reentrant | OK, see 8 |
| [BSW006] Platform independency | OK, see CNM069 |
| 4.3.3.6 Types and keywords | |
| [BSW00357] Standard API return type | OK, see 8 |
| [BSW00377] Module specific API return types | OK, see 8 |
| [BSW00304] AUTOSAR integer data types | OK, see 8 |
| [BSW00355] Do not redefine AUTOSAR integer data types | OK, see 8 |
| [BSW00378] AUTOSAR boolean type | OK, see 8 |
| [BSW00306] Avoid direct use of compiler and platform specific keywords | OK, see 8 |
| 4.3.3.7 Global data | |
| [BSW00308] Definition of global data | OK, see 10 |
| [BSW00309] Global data with read-only constraint | OK, see 10 |
| 4.3.3.8 Interface and API | |
| [BSW00371] Do not pass function pointers via API | OK, see 8 |
| [BSW00358] Return type of init() functions | OK, see 8.3.1 |
| [BSW00376] Return type and parameters of main processing functions | OK, see 8.5.1 |
| [BSW00359] Return type of callback functions | OK, see 8 |
| [BSW00360] Parameters of callback functions | OK, see 8 |
| [BSW00329] Avoidance of generic interfaces | OK, see 8 |
| [BSW00330] Usage of macros / inline functions instead of functions | OK, see 8 |
| [BSW00331] Separation of error and status values | OK, see CNM100 |
| 4.3.4 Software Documentation Requirements | |
| [BSW009] Module User Documentation | N/A, implementation specific |
| [BSW172] Compatibility and documentation of scheduling strategy | N/A, implementation specific |
| [BSW010] Memory resource documentation | N/A, implementation specific |
| [BSW00333] Documentation of callback function context | N/A, implementation specific |
| [BSW00374] Module vendor identification | OK, see 10 |
| [BSW00379] Module identification | OK, see 10 |
| [BSW003] Version identification | OK, see 10 |
| [BSW00318] Format of module version numbers | OK, see 10 |
| [BSW00321] Enumeration of module version numbers | OK, see 10 |
| [BSW00341] Microcontroller compatibility documentation | N/A, implementation specific |
| [BSW00334] Provision of XML file | N/A, implementation specific |

Document: Requirements on Basic Software Module NM [2]

| ***Requirement*** | ***Satisfied by*** |
|---|---|
| [BSW150] Configuration of functionality | CNM143, CNM079, CNM073, CNM088, CNM082 |
| [BSW151] Integration into running system | CNM064 |
| [BSW043] Bus Traffic without NM Initialization | CNM062 |
| [BSW044] Independency of Underlying Communication System | N/A, CAN Generic NM is CAN specific |
| [BSW045] Channel Selective Wake-Up/Shutdown | CNM130 |
| [BSW046] Trigger of startup of all Nodes at any Point in Time | CNM035 |
| [BSW047] Bus Sleep and Bus Keep Awake Service | CNM042, CNM043 |
| [BSW048] Bus Sleep Mode | CNM002 |
| [BSW050] Bus State Information | CNM010 |
| [BSW051] Bus State Change Information | CNM005 |
| [BSW052] Notification that all other ECUs are ready to sleep | CNM074 |
| [BSW02509] Notification that at least one other node is not ready to sleep anymore | CNM075 |
| [BSW02503] Sending user data | CNM080 |
| [BSW02504] Reading user data | CNM081 |
| [BSW153] Detection of present nodes | -- |
| [BSW02508] Unambiguous node identification per bus | N/A, see bus specific SWS |
| [BSW02505] Sending node identifier | -- |
| [BSW02506] Reading node identifier | CNM048 |
| BSW02507] Reading the configured NM Identifier of the local node | CNM049 |
| [BSW053] Deterministic Behavior in Case of Bus Unavailability | CNM019, CNM025 |
| [BSW137] Communication system error handling | N/A, see bus specific SWS |
| [BSW136] Coordination of coupled networks | CNM087 |
| [BSW139] Compliance with OSEK NM on one cluster | N/A, see NM gateway SWS |
| [BSW140] Compliance with OSEK NM on a gateway | N/A, see NM gateway SWS |
| [BSW054] Deterministic Time for Bus Sleep | CNM038 |
| [BSW142] Limitation of NM bus load | N/A, see bus specific SWS |
| [BSW143] Deterministic average bus load | N/A, see bus specific SWS |
| [BSW144] ECU cluster size | CNM072 |
| [BSW145] Robustness against NM message losses | N/A |
| [BSW146] Robustness against NM message jitter | N/A |
| [BSW147] Processor independent algorithm | CNM069 |
| [BSW148] Separation of Communication system dependent parts | N/A, CAN Generic NM is CAN specific |
| [BSW149] Configurable Timing | CNM138 |
| [BSW154] Bus independency of basic API | CNM130 |
| [BSW156] Bus independency of communication mechanisms | N/A, CAN Generic NM is CAN specific |
| [BSW02501] Number of hardware send buffer | N/A, see SWS FlexRay NM |
| [BSW02502] Number of hardware receive buffer | N/A, see SWS FlexRay NM |
| [BSW02512] CommunicationControl (28 hex) service support | CNM161, CNM162, CNM170, CNM165, CNM159, CNM169, CNM171, CNM173, CNM164, CNM166, CNM160, CNM163, CNM167, CNM168, CNM172, CNM174 |

# 7 Functional specification

## 7.1 Coordination algorithm

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

The AUTOSAR CAN Generic NM coordination algorithm is based on periodic NM PDUs, which are received by all nodes in the cluster via broadcast transmission. Reception of NM PDUs indicates that sending nodes want to keep the NM-cluster awake. If any node is ready to go to the Bus-Sleep Mode, it stops sending NM PDUs, but as long as NM PDUs from other nodes are received, it postpones transition to the Bus-Sleep Mode. Finally, if dedicated timer elapses because no NM PDUs are received anymore, every node initiates transition to the Bus-Sleep Mode.

If any node in the NM-cluster requires bus-communication, it can wake-up the NM-cluster from the Bus-Sleep Mode by transmitting NM PDUs. For more details concerning wakeup procedure itself please refer to the Mode Management (see [7], [8]).

The main concept of the AUTOSAR CAN Generic NM coordination algorithm can be defined by following two key-requirements:

**CNM001**: Every network node shall transmit periodic NM PDUs as long as it requires bus-communication; otherwise it shall transmit no NM PDUs.

**CNM002:** If bus communication is released and there are no NM PDUs on the bus for a configurable amount of time determined by **`CNM_TIMEOUT_TIME`** + **`CNM_WAIT_BUS_SLEEP_TIME`** (both configuration parameters) transition into the Bus-Sleep Mode shall be performed .

The overall state machine of the AUTOSAR CAN Generic NM coordination algorithm can be defined as follows:

**CNM004:** The AUTOSAR CAN Generic NM state machine shall contain states, transitions and triggers required for the AUTOSAR CAN Generic NM coordination algorithm seen from point of view of one single node in the NM-cluster .

**CNM003:** Transitions in the AUTOSAR CAN Generic NM state machine shall be triggered by calls of selected interface functions or by expiration of internal timers or counters.

The Figure 7-1 shows an overview of the state diagram for the AUTOSAR CAN Generic NM state machine from point of view of one single node in the NM-cluster. All services called by AUTOSAR CAN Generic NM are in italic script, the bus-communication state is underlined and the events triggering the state transitions are in normal script.

For more detailed description of the AUTOSAR CAN Generic NM state machine in sense of an UML state chart see API specification (see [11]).

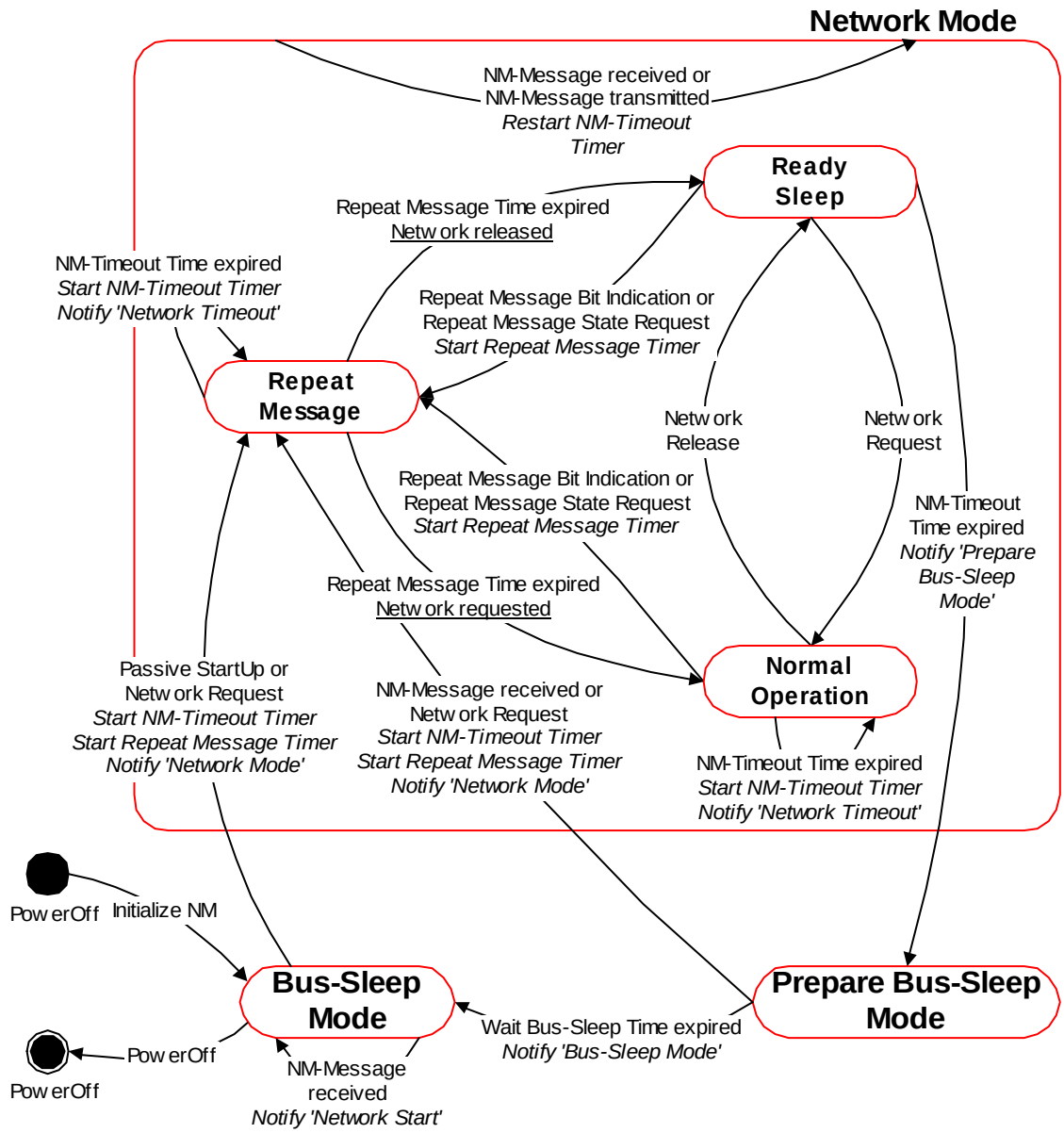


**Figure 7-1**

## 7.2 Operational modes

In the following chapter operational modes of the AUTOSAR CAN Generic NM coordination algorithm are described in detail.

**CNM010:** Service call **`CNm_GetState`** shall provide the information about the current state and the current mode of the NM state machine; operation of providing the NM state and NM mode shall guarantee data consistency between the provided values and the current values of the state and mode.

**CNM011**: The AUTOSAR CAN Generic NM shall consist of three operational modes:

- Network Mode
- Prepare Bus-Sleep Mode
- Bus-Sleep Mode

**CNM005:** Changes of the AUTOSAR CAN Generic NM operational modes shall be notified to the upper layer by means of callback functions.

### 7.2.1 Network Mode

**CNM012:** The Network Mode shall consist of three internal states:

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

**CNM013:** When the Network Mode is entered from Bus-Sleep Mode or Prepare Bus-Sleep Mode, by default, the Repeat Message State shall be entered .

**CNM014:** When the Network Mode is entered, the NM-Timeout Timer shall be started.

**CNM015:** When the Network Mode is entered, the NM shall notify the upper layer by calling **`Nm_NetworkMode`**.

**CNM016:** At successful reception of a NM PDU in the Network Mode, the NM-Timeout Timer shall be restarted.

**CNM017:** At successful transmission of a NM PDU in the Network Mode, the NM-Timeout Timer shall be restarted.

#### 7.2.1.1 Repeat Message State

For nodes that are not in passive mode the Repeat Message State ensures, that any transition from Bus-Sleep or Prepare Bus-Sleep to the Network Mode becomes visible to the other nodes on the network. Additionally it ensures that any node stays

active for a minimum amount of time. Optionally it can be used for detection of present nodes.

**CNM018**: When the Repeat Message State is entered from Bus-Sleep Mode, Prepare-Bus-Sleep Mode, Normal Operation State or Ready Sleep State transmission of NM PDUs shall be started unless passive mode is enabled.

**CNM019**: When the NM-Timeout Timer expires in the Repeat Message State, the NM-Timeout Timer shall be restarted.

**CNM021:** The NM shall stay in the Repeat Message State for a configurable amount of time determined by the **`CNM_REPEAT_MESSAGE_TIME`** (configuration parameter); after that time the Repeat Message State shall be left .

**CNM022:** When Repeat Message State is left, the Normal Operation State shall be entered, if the network has been requested (see CNM042).

**CNM023:** When Repeat Message State is left, the Ready Sleep State shall be entered, if the network has been released (see CNM043).

**CNM156:** When Repeat Message State is left and the option **`CNM_NODE_DETECTION_ENABLED`** is enabled, the Repeat Message Bit shall be cleared by calling **`CanNm_SetRepeatMessageBit`**.

#### 7.2.1.2 Normal Operation State

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

**CNM024:** When the Normal Operation State is entered from Ready Sleep State, transmission of NM PDUs shall be started unless passive mode is enabled.

**CNM025:** When the NM-Timeout Timer expires in the Normal Operation State, the NM-Timeout Timer shall be restarted.

**CNM027:** When the network is released and the current state is Normal Operation State, the Normal Operation State shall be left and the Ready Sleep state shall be entered.

**CNM028:** At Repeat Message Request Bit Indication in the Normal Operation State, the Normal Operation State shall be left and the Repeat Message State shall be entered unless the function CNm_DisableCommunication has been called.

**CNM153:** At Repeat Message Request (**`CNm_RepeatMessageRequest`**) in the Normal Operation State, the Normal Operation State shall be left and the Repeat Message State shall be entered unless the function CNm_DisableCommunication has been called.

**CNM155:** At Repeat Message Request in Normal Operation State the Repeat Message Bit shall be set by calling **`CanNm_SetRepeatMessageBit`** unless the function CNm_DisableCommunication has been called.

#### 7.2.1.3 Ready Sleep State

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

**CNM029:** When the Ready Sleep State is entered from Repeat Message State or Normal Operation State, transmission of NM PDUs shall be stopped unless passive mode is enabled.

**CNM030:** When the NM-Timeout Timer expires in the Ready Sleep State, the Ready Sleep State shall be left and the Prepare Bus-Sleep Mode shall be entered.

**CNM031:** When the network is requested and the current state is the Ready Sleep State, the Ready Sleep State shall be left and the Normal Operation State shall be entered.

**CNM032**: At Repeat Message Request Bit Indication in the Ready Sleep State, the Ready Sleep State shall be left and the Repeat Message State shall be entered.

**CNM154:** At Repeat Message Request (**`CNm_RepeatMessageRequest`**) in the Ready Sleep State, the Ready Sleep State shall be left and the Repeat Message State shall be entered.

**CNM157**: At Repeat Message Request in Ready Sleep State the Repeat Message Bit shall be set by calling **`CanNm_SetRepeatMessageBit`**.

### 7.2.2 Prepare Bus-Sleep Mode

Bus activity is calmed down (i.e. queued messages are transmitted in order to make all Tx-buffers empty) and finally there is no activity on the bus in the Prepare Bus-Sleep Mode.

**CNM033:** When Prepare Bus-Sleep Mode is entered, the NM shall notify the upper layer by calling **`Nm_PrepareBusSleepMode`**.

**CNM034:** The NM shall stay in the Prepare Bus-Sleep Mode for a configurable amount of time determined by the **`CNM_WAIT_BUS_SLEEP_TIME`** (configuration parameter); after that time the Prepare Bus-Sleep Mode shall be left and the Bus-Sleep Mode shall be entered.

**CNM035:** When the network has been requested in the Prepare Bus-Sleep Mode and Network Mode has been entered, the service **`CanNm_TriggerTransmission`** shall be called if **`CNM_IMMEDIATE_RESTART_ENABLED`** (configuration parameter) is defined.

**Rationale**: Other nodes in the cluster are still in Prepare Bus-Sleep Mode; in the exceptional situation described above transition into the Bus-Sleep Mode shall be avoided and bus-communication shall be restored as fast as possible.
Caused by the transmission offset for NM PDUs in CAN NM, the transmission of the first NM PDU in Repeat Message State can be delayed significantly. In order to avoid a delayed re-start of the network the transmission of a NM PDU can be requested immediately.

Note: If **CNM_IMMEDIATE_RESTART_ENABLED** is defined and a wake-up line is used, a burst of NM PDUs occurs if all network nodes get a network request in Prepare Bus-Sleep Mode.

**CNM036**: When the network is requested in the Prepare Bus-Sleep Mode, the Prepare Bus-Sleep Mode shall be left and the Network Mode shall be entered; by default the Repeat Message State is entered (see CNM013).

**CNM037**: At successful reception of a NM PDU in the Prepare Bus-Sleep Mode, the Prepare Bus-Sleep Mode shall be left and the Network Mode shall be entered; by default the Repeat Message State is entered (see CNM013).

### 7.2.3 Bus-Sleep Mode

The communication controller is switched into the sleep mode, respective wakeup mechanisms are activated and finally power consumption is reduced to the adequate level in the Bus-Sleep Mode.

**CNM038:** If a configurable amount of time determined by the **CNM_TIMEOUT_TIME** + **CNM_WAIT_BUS_SLEEP_TIME** (both configuration parameters) is identically configured for all nodes in the NM-cluster, then all nodes in the NM-cluster that are coordinated with use of the AUTOSAR NM algorithm shall perform the transition into the Bus-Sleep Mode at approximately the same time.

Note: The parameters **CNM_TIMEOUT_TIME** and **CNM_WAIT_BUS_SLEEP_TIME** should have the same values within all network nodes of the NM-cluster.
Depending on the specific implementation, transition into the Bus-Sleep Mode takes place exactly or approximately at the same time; time jitter for this transition depends on the following factors:

- internal clock precision (oscillator's drift),
- NM-task cycle time (if tasks are not synchronized with a global time),
- NM PDU waiting time in the Tx-queue (if transmission confirmation is made immediately after transmit request).

In the best case only oscillator's drift shall be taken into account for a configurable amount of time determined by the value **CNM_TIMEOUT_TIME** + **CNM_WAIT_BUS_SLEEP_TIME** (both configuration parameters).

**CNM039:** When Bus-Sleep Mode is entered, the NM shall notify the upper layer by calling **`Nm_BusSleepMode`**; it shall not be however the case if Bus-Sleep Mode is entered by default at initialization.

**CNM040:** At successful reception of a NM PDU in the Bus-Sleep Mode, the NM shall notify the upper layer by calling **`Nm_NetworkStartIndication`**.

Rationale: It is required to avoid race conditions and state inconsistency between Network and Mode Management. NM PDU reception in Bus-Sleep Mode must be handled depending on the current state of the ECU shutdown/startup process.

**CNM041:** If **`CNm_PassiveStartUp`** is called in the Bus-Sleep Mode, the Bus-Sleep Mode shall be left and the Network Mode shall be entered.

**Note:** In the Prepare Bus-Sleep Mode and Bus-Sleep Mode is assumed that the network is released, unless bus communication is explicitly requested.

**CNM144:** When the network is requested in Bus-Sleep Mode, the Bus-Sleep Mode shall be left and the Network Mode shall be entered; by default the Repeat Message State is entered (see CNM013).

## 7.3 Network states

Network states (i.e. ‘requested’ and ‘released’) are two additional states of the AUTOSAR CAN Generic NM state machine that exist in parallel to the state machine described in 0. Network states denote, whether the software components need to communicate on the bus (the network state is then ‘requested’); or whether the software components don’t have to communicate on the bus (the bus network state is then ‘released’); note that if the network is released an ECU may still communicate because some other ECU still request the network.

**CNM042:** The optional service call **`CNm_NetworkRequest`** shall request the network. I.e. network state shall be changed to ‘requested’.

**CNM043:** The optional service call **`CNm_NetworkRelease`** shall release the network. I.e. network state shall be changed to ‘released’.

## 7.4 Network management message transmission and reception

**CNM044:** The optional service call **`CanNm_StartTransmission`** shall start transmission of NM PDUs. Also refer to CNM141.

**CNM045**: The optional service call **`CanNm_StopTransmission`** shall stop transmission of NM PDUs. Also refer to CNM141.

**CNM046:** The optional service call **`CanNm_TriggerTransmission`** shall trigger transmission of a single NM PDU.

Rationale: This service is typically used for supporting the NM gateway extensions but also in exceptional situations in the NM state machine.

**CNM141:** The optional services **CanNm_StartTransmission**, **CanNm_StopTransmission**, and **CanNm_TriggerTransmission** shall be statically configurable with use of the **CNM_PASSIVE_MODE_ENABLED** switch (configuration parameter).

**CNM047**: NM PDU can contain the following information:

- Source Node Identifier (optional),
- Control Bit Vector (optional),
- User Data (optional).

Note: The Control Bit Vector is implicitly configured with **CNM_NODE_DETECTION_ENABLED**.

**CNM048:** The optional service call **CNm_GetNodeIdentifier** shall provide the node identifier out of the most recently received NM PDU.

**CNM049:** The optional service call **CNm_GetLocalNodeIdentifier** shall provide the node identifier configured for the local host node.

**CNM050**: The optional Control Bit Vector shall contain the following bits:

- Repeat Message Request Bit

**CNM147:** Support of Repeat Message Request Bit and Repeat Message State Request shall be statically configurable with use of the **CNM_NODE_DETECTION_ENABLED** switch (configuration parameter).

**CNM146:** The optional service call **CNm_RepeatMessageRequest** shall be statically configured with use of the **CNM_NODE_DETECTION_ENABLED** switch (configuration parameter).

**CNM158:** If the optional service **CNm_RepeatMessageRequest** is called in Repeat Message State, Prepare Bus-Sleep Mode or Bus-Sleep Mode, the service shall not be executed and **NM_E_NOT_EXECUTED** shall be returned.

**CNM111:** The optional service call **CNm_GetPduData** shall provide whole PDU data (Node ID, Control Bit Vector and User Data) of the most recently received NM PDU.

**CNM148**: The optional service **CNm_GetPduData** shall be statically configurable. It shall be available if **CNM_NODE_ID_ENABLED** or **CNM_NODE_DETECTION_ENABLED** or **CNM_USER_DATA_ENABLED** is defined.

**CNM051:** The NM PDU length shall be configured in the CanNm.

Network management message transmission and reception is bus specific and thus is defined in detail in CanNm (see [5]).

## 7.5 Initialization and startup

**CNM055:** After reset the Network Management state shall be set to NM_STATE_UNINIT.

**CNM056**: After successful initialization the Network Management state shall be set to NM_STATE_BUS_SLEEP, otherwise it shall be set to NM_STATE_UNINIT.

**CNM057**: Service call CNm_Init shall initialize CAN Generic NM.

**CNM058:** The NM shall be initialized after CanIf is initialized and before any other NM service is called.

**CNM059**: When initialized, the NM shall initialize CanNm by calling CanNm_Init.

**CNM060:** If initialized, by default, the network state shall be set to ‘released’.

**CNM061:** If initialized, by default, the Bus-Sleep Mode shall be entered.

**CNM062:** Bus traffic should not be prohibited if AUTOSAR CAN Generic NM is not initialized.

**CNM064:** Service call CNm_Init shall re-initialize the NM, if NM is already initialized.

**CNM065:** The function CNm_Init shall use the active configuration set provided through a reference to it (see 8.3.1).

**CNM068:** If CNm_PassiveStartUp is called in the Prepare Bus-Sleep Mode or Network Mode, the service shall not be executed and NM_E_NOT_EXECUTED shall be returned.

## 7.6 Execution

### 7.6.1 Processor architecture

**CNM069:** The AUTOSAR CAN Generic NM coordination algorithm shall be processor independent, which means, it shall not rely on any processor specific hardware support and thus shall be realizable on any processor architecture that is in the scope of AUTOSAR.

### 7.6.2 Timing parameters

**CNM070:** Timing parameters necessary for the AUTOSAR CAN Generic NM shall be determined by the following configuration parameters:

- NM-Timeout Time: **CNM_TIMEOUT_TIME**
- Repeat Message Time: **CNM_REPEAT_MESSAGE_TIME**
- Wait Bus-Sleep Time: **CNM_WAIT_BUS_SLEEP_TIME**
- Remote Sleep Indication Time: **CNM_REMOTE_SLEEP_IND_TIME** (optional)

Note: The NM-Timeout timer is started in situations specified by the requirements CNM014, CNM016, CNM017, CNM019, CNM025; expiration of the NM-Timeout timer triggers actions specified by the requirements CNM030, CNM026CNM019, CNM020CNM025.

## 7.7 Additional features

### 7.7.1 Cluster size

**CNM072:** The AUTOSAR CAN Generic NM algorithm shall support up to 64 nodes per NM-Cluster.

Note: The AUTOSAR CAN Generic NM algorithm can support an arbitrary number of nodes per NM-cluster (even more than default 64 nodes per cluster, if necessary) – it is only a matter of configuration, since the upper limit is not fixed and depends on the trade off between response time, fault-tolerance and resulted bus load configured for the AUTOSAR CAN Generic NM coordination algorithm.

### 7.7.2 Detection of Remote Sleep Indication (optional)

The so-called “Remote Sleep Indication” denotes a situation, where all nodes in the cluster are ready to sleep apart from one node, which still keeps the bus awake.

**CNM073:** Detection of remote sleep indication shall be statically configurable with use of the **CNM_REMOTE_SLEEP_IND_ENABLED** switch (configuration parameter).

**CNM074:** If no NM PDUs are received in the Normal Operation State for a configurable amount of time determined by the **CNM_REMOTE_SLEEP_IND_TIME** (configuration parameter), the NM shall notify the NM Interface that all other nodes in the cluster are ready to sleep (the so-called ‘Remote Sleep Indication’) by calling **Nm_RemoteSleepIndication**.

**CNM075:** If Remote Sleep Indication has been previously detected and if a NM PDU is received in the Normal Operation State again, the NM shall notify the NM Interface that some nodes in the cluster are not ready to sleep anymore (the so-called ‘Remote Sleep Cancellation’) by calling **Nm_RemoteSleepCancelation**.

**CNM076:** If Remote Sleep Indication has been previously detected and if Repeat Message State is entered from Normal Operation State, the NM shall notify the NM Interface that some nodes in the cluster are not ready to sleep anymore (the so-called 'Remote Sleep Cancellation') by calling **`Nm_RemoteSleepCancelation`**.

**CNM077:** Service call **`CNm_CheckRemoteSleepIndication`** shall provide the information about current status of Remote Sleep Indication (i.e. already detected or not).

**CNM078:** The NM shall reject a check of Remote Sleep Indication in Bus-Sleep Mode, Prepare Bus-Sleep Mode and Repeat Message State; the service shall not be executed and **`NM_E_NOT_EXECUTED`** shall be returned.

### 7.7.3 User data (optional)

**CNM079:** Support of NM user data shall be statically configurable with use of the **`CNM_USER_DATA_ENABLED`** switch (configuration parameter).

**CNM080:** When **`CNm_SetUserData`** is called the NM user data for NM PDUs transmitted next on the bus shall be set; operation of setting the NM user data shall guarantee data consistency.

**CNM081**: When **`CNm_GetUserData`** is called the NM user data out of the most recently received NM PDU shall be provided; operation of providing the NM user data shall guarantee data consistency.

Note: If user data is configured it will be sent for sure in the Repeat Message State. In the Normal Operation State it is up to the configuration of busload reduction. In the Ready Sleep State the user data will not be sent.

### 7.7.4 Busload reduction (optional)

Busload reduction allows limitation of busload to maximum two NM PDUs per NM message cycle time.

**CNM082:** Busload reduction shall be statically configurable with use of the **`CNM_BUS_LOAD_REDUCTION_ENABLED`** switch (configuration parameter).

**CNM083:** When the Repeat Message State is entered from Bus-Sleep Mode, Prepare Bus-Sleep Mode, Normal Operation or Ready Sleep State the busload reduction shall be deactivated.

**CNM084:** When the Normal Operation State is entered from Repeat Message State or Ready Sleep State the busload reduction shall be activated .

The busload reduction is defined in detail in the CAN specific NM specification ([5]).

### 7.7.5 Passive Mode (optional)

**CNM085:** Passive Mode shall be statically configurable with use of the **CNM_PASSIVE_MODE_ENABLED** switch (configuration parameter).

**CNM175:** Passive Mode shall be statically configured consistent for all instances within one ECU.

**CNM142:** If Passive Mode is used (configuration parameter **CNM_PASSIVE_MODE_ENABLED**) the following options shall not be used:

- Bus Synchronization
  (configuration parameter **CNM_BUS_SYNCHRONIZATION_ENABLED**)
- Bus Load Reduction
  (configuration parameter **CNM_BUS_LOAD_REDUCTION_ENABLED**)
- Remote Sleep Indication
  (configuration parameter **CNM_REMOTE_SLEEP_IND_ENABLED**)
- Node Detection
  (configuration parameter **CNM_NODE_DETECTION_ENABLED**)
- Communication Control
  (configuration parameter **CNM_COM_CONTROL_ENABLED**)

### 7.7.6 NM PDU Rx Indication (optional)

**CNM149:** At successful reception of a NM PDU the NM shall notify the upper layer by calling **Nm_PduRxIndication**.

Rationale: If any higher software layer needs to retrieve the NM PDU data of every NM PDU it is required to have a Rx Indication. Polling of the NM PDU data could result in loss of received NM PDU data in case of a NM PDU burst.

**CNM150:** The optional service **Nm_PduRxIndication** shall be statically configurable. It shall be available if **CNM_PDU_RX_INDICATION_ENABLED** is defined.

### 7.7.7 State change notification (optional)

**CNM151**: All changes of the AUTOSAR CAN Generic NM states shall be notified to the upper layer by calling **Nm_StateChangeNotification**.

**CNM152**: The optional service **Nm_StateChangeNotification** shall be statically configurable. It shall be available if **CNM_STATE_CHANGE_IND_ENABLED** is defined.

### 7.7.8 Communication Control (optional)

**CNM161:** Communication Control shall be statically configurable with use of the **CNM_COM_CONTROL_ENABLED** switch (configuration parameter).

**CNM170:** If initialized, by default, the network NM PDU transmission ability shall be enabled.

**CNM162**: The optional service **`CNm_DisableCommunication`** shall disable the NM PDU transmission ability if the network state is 'requested' and the current state is Normal Operation State.

Note: It is dangerous to use this service, since the NM PDUs are turned off and they need to be turned on after diagnostic session. The ECU cannot shutdown as long as the NM PDU transmission ability is disabled.

**CNM165:** The optional service **`CNm_DisableCommunication`** shall return **`NM_E_NOT_EXECUTED`**, if the network state is 'released'.

**CNM169:** The optional service **`CNm_DisableCommunication`** shall return **`NM_E_NOT_EXECUTED`**, if the current state is not Normal Operation State.

**CNM159**: When the NM PDU transmission ability is disabled, the transmission of NM PDUs shall be stopped by calling **`CanNm_StopTransmission`**.

**CNM171**: When the NM PDU transmission ability is disabled, the NM-Timeout Timer shall be stopped.

**CNM173**: When the NM PDU transmission ability is disabled, the detection of Remote Sleep Indication Timer shall be suspended.

**CNM164**: The optional service **`CNm_EnableCommunication`** shall enable the NM PDU transmission ability if the NM PDU transmission ability is disabled.

**CNM166:** The optional service **`CNm_EnableCommunication`** shall return **`NM_E_NOT_EXECUTED`** if the NM PDU transmission ability is enabled.

**CNM160:** When the NM PDU transmission ability is enabled, the transmission of NM PDUs shall be started by calling **`CanNm_StartTransmission`**.

**CNM172:** When the NM PDU transmission ability is enabled, the NM-Timeout Timer shall be restarted.

**CNM174:** When the NM PDU transmission ability is enabled, the detection of Remote Sleep Indication Timer shall be resumed.

**CNM163:** The optional service **`CNm_RequestBusSynchronization`** shall return **`NM_E_NOT_EXECUTED`** if the NM PDU transmission ability is disabled.

**CNM167:** Service call **`CNm_NetworkRelease`** shall return **`NM_E_NOT_EXECUTED`**, if the NM PDU transmission ability is disabled.

**CNM168**: The optional service **`CNm_RepeatMessageRequest`** shall return **`NM_E_NOT_EXECUTED`**, if the NM PDU transmission ability is disabled.

## 7.8 Application notes

### 7.8.1 Wakeup notification

Wakeup notification is defined in detail in the ECU State Manager specification ([8]).

### 7.8.2 Coordination of coupled networks

**CNM087**: The AUTOSAR CAN Generic NM shall support coordination of coupled networks with the following optional services:

- Bus synchronization on demand

Bus synchronization on demand allows synchronization of a NM-cluster for an arbitrary point of time; in result NM-Timeout Timers in all nodes of the NM-cluster are restarted.

**CNM088:** Support of bus synchronization on demand shall be statically configurable with use of the **`CNM_BUS_SYNCHRONIZATION_ENABLED`** switch (configuration parameter) .

**CNM089:** Service call **`CNm_RequestBusSynchronization`** shall trigger transmission of a single NM PDU independently of the normal periodic transmission using the service **`CanNm_TriggerTransmission`**.

**CNM090:** If **`CNm_RequestBusSynchronization`** is called in Bus-Sleep Mode and Prepare Bus-Sleep Mode the service shall not be executed and **`NM_E_NOT_EXECUTED`** shall be returned.

### 7.8.3 Interoperability with direct OSEK NM

Interoperability with direct OSEK NM is defined in detail in the network management gateway interoperability specification.

## 7.9 Error classification

**CNM095:** Reporting of development errors shall be statically configurable with use of the **`CNM_DEV_ERROR_DETECT`** switch (configuration parameter).

**CNM096:** Development errors shall not be returned by API functions; in case of an development error, the respective API function will return NM_E_NOT_OK, if applicable.

**CNM097**: Development errors shall be reported to the Development Error Tracer.

**CNM098:** Production errors shall not be returned by API functions; in case of an production error, the respective API function will return NM_E_NOT_OK, if applicable.

**CNM099:** Production errors shall be reported to the Diagnostic Event Manager.

**CNM100:** The following errors and exceptions shall be detectable by the NM depending on its build version (development/production mode).

| *Type or error* | *Relevance* | *Related error code* | *Value* |
|---|---|---|---|
| API service used without module initialization | Development | NM_E_NO_INIT | 01h |
| API service used with invalid channel handle | Development | NM_E_INVALID_CHANNEL | 02h |
| NM-Timeout Timer has abnormally expired outside of the Ready Sleep State; it may happen: (1) because of Bus-Off state, (2) if some ECU requests bus communication or node detection shortly before the NM-Timeout Timer expires so that a NM message can not be transmitted in time; this race condition applies to event-triggered systems | Production | NM_E_NETWORK_TIMEOUT | Assigned by DEM |
| NM-Timeout Timer has abnormally expired outside of the Ready Sleep State; it may happen: (1) because of Bus-Off state, (2) if some ECU requests bus communication or node detection shortly before the NM-Timeout Timer expires so that a NM message can not be transmitted in time; this race condition applies to event-triggered systems | Development | NM_E_DEV_NETWORK_TIMEOUT | 11h |
| NM transmission path has failed | Production | NM_E_TX_PATH_FAILED | Assigned by DEM |
| NM transmission path has failed | Development | NM_E_DEV_TX_PATH_FAILED | 12h |
| Null pointer has been passed as an argument | Development | NM_E_NULL_POINTER | 13h |

| | | | |
|---|---|---|---|
| (Does not apply to function **CNm_Init**) | | | |
| NM initialization has failed | Production | NM_E_INIT_FAILED | Assigned by DEM |

**CNM101:** If not initialized, the NM shall reject every API service apart from **CNm_Init**; the called function shall not be executed, but instead of that it shall report **NM_E_NO_INIT** to the Development Error Tracer and it shall return **NM_E_NOT_OK** to the calling function.

**CNM102:** When NM API service with an invalid NM-channel handle is called, the called function shall not be executed, but instead of that it shall report **NM_E_INVALID_CHANNEL** to the Development Error Tracer (the value of invalid channel handle shall be passed to DET as instance ID) and it shall return **NM_E_NOT_OK** to the calling function.

Note: NM-channel handle is invalid if it is different from allowed configured values.

**CNM103:** When the NM-Timeout Timer expires in the Repeat Message State, the NM shall report **NM_E_NETWORK_TIMEOUT** to Diagnostic Event Manager .

**CNM104:** When the NM-Timeout Timer expires in the Normal Operation State, the NM shall report **NM_E_NETWORK_TIMEOUT** to Diagnostic Event Manager.

# 8 API specification

API specification

**CNM130:** AUTOSAR CAN Generic NM API consists of services, which are CAN specific and can be called whenever they are required; each service apart from **CNm_Init** refers to one NM channel only.

The Figure 8-1 gives an overview of the CNm services.

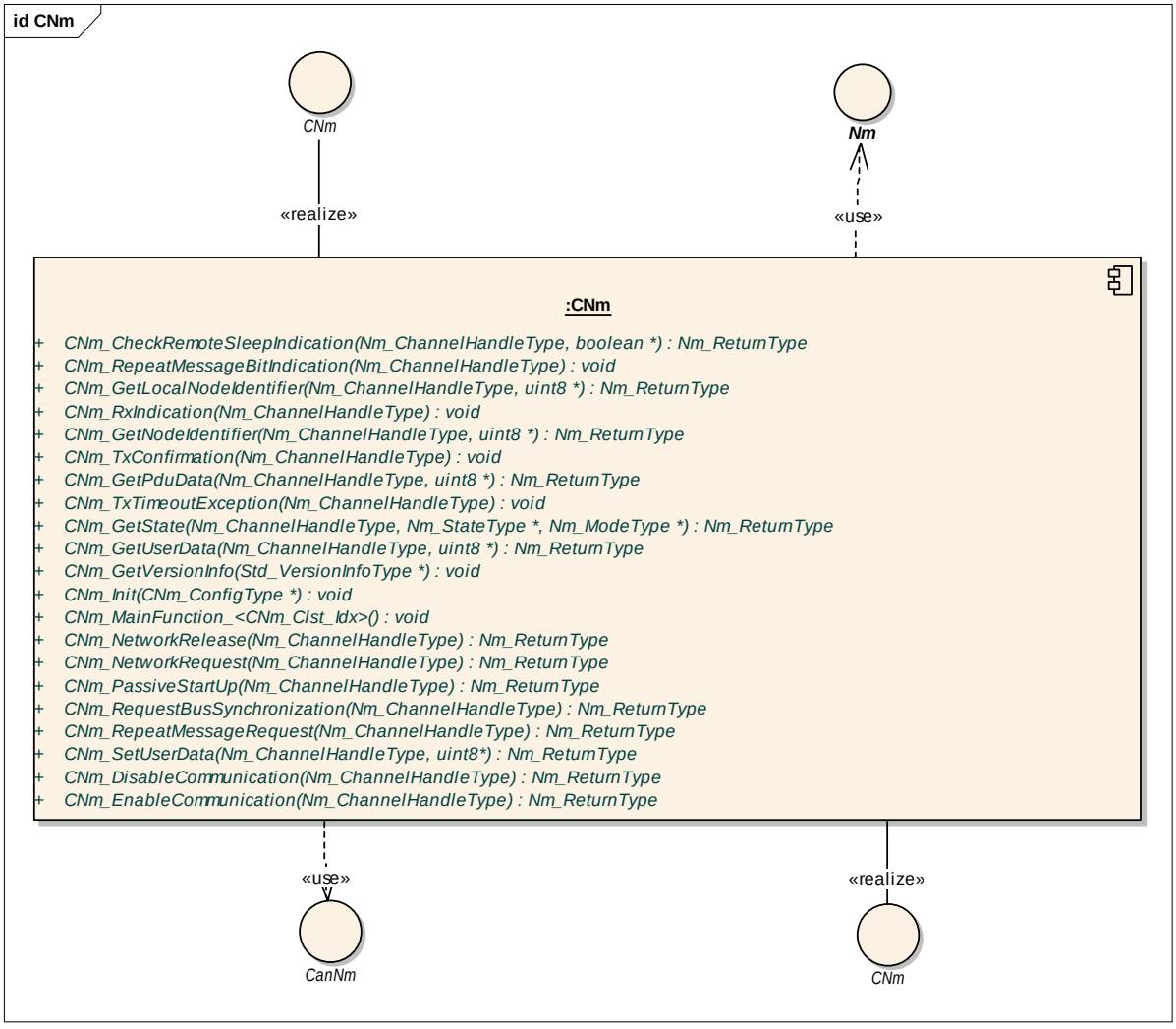


**Figure 8-1**

## 8.1 Imported types

In this chapter all types included from the following files are listed:

- Std_Types.h:

  - Std_ReturnType
  - Std_VersionInfoType

- Platform_Types.h:
  - uint8, uint16, uint32
  - uint8_least, uint16_least, uint32_least
- Nm_StackTypes.h

## 8.2 Type definitions

There is no CAN Generic NM specific type defined.

## 8.3 Function definitions: Nm Services provided to upper layers

### 8.3.1 CNm_Init

| | | |
|---|---|---|
| ***Service name:*** | CNm_Init | |
| ***Syntax:*** | `void CNm_Init ( const CNm_ConfigType * const cnmConfigPtr );` | |
| ***Service ID:*** | `0x00` | |
| ***Sync/Async:*** | Synchronous | |
| ***Reentrancy:*** | Non-reentrant | |
| ***Parameters (in):*** | `cnmConfigPtr` | Pointer to the selected configuration set |
| | -- | -- |
| | -- | -- |
| ***Parameters (out):*** | `None` | N/A |
| ***Return value:*** | `None` | N/A |
| | -- | -- |
| ***Description:*** | Initializes the complete AUTOSAR CAN Generic NM module, i.e. all channels that are activated (see also configuration parameter `CNM_CHANNEL_ACTIVE`) at configuration time are initialized.<br>If a NULL pointer is passed as an argument to this function the default configuration shall be used.<br>By default the NM starts in the Bus-Sleep Mode.<br>Initially – by default – the network is released.<br>This function shall call CanNm_Init. | |
| ***Caveats:*** | This service function has to be called after the initialization of CanIf. | |
| ***Configuration:*** | Mandatory | |

### 8.3.2 CNm_PassiveStartUp

| | | |
|---|---|---|
| ***Service name:*** | CNm_PassiveStartUp | |
| ***Syntax:*** | `Nm_ReturnType CNm_PassiveStartUp ( const Nm_ChannelHandleType nmChannelHandle );` | |
| ***Service ID:*** | `0x01` | |
| ***Sync/Async:*** | Asynchronous | |
| ***Reentrancy:*** | Reentrant (but not for the same NM-Channel) | |
| ***Parameters (in):*** | `nmChannelHandle` | Identification of the NM-channel |
| | -- | -- |
| | -- | -- |
| ***Parameters (out):*** | `None` | N/A |
| ***Return value:*** | `NM_E_OK` | No error |
| | `NM_E_NOT_OK` | Passive startup of network management has failed |
| | `NM_E_NOT_EXECUTED` | Passive startup of network management is currently not executed |
| ***Description:*** | Passive startup of the AUTOSAR CAN Generic NM. It triggers the transition from Bus-Sleep Mode to the Network Mode in Repeat Message State.<br>This service has no effect if the current state is not equal to Bus-Sleep Mode. In | |

| | |
|---|---|
| | that case NM_E_NOT_EXECUTED is returned. |
| ***Caveats:*** | CanNm and CNm itself are initialized correctly. |
| ***Configuration:*** | Mandatory |

### 8.3.3 CNm_NetworkRequest

| | | |
|---|---|---|
| ***Service name:*** | CNm_NetworkRequest | |
| ***Syntax:*** | Nm_ReturnType **CNm_NetworkRequest**<br>(<br>  const Nm_ChannelHandleType nmChannelHandle<br>); | |
| ***Service ID:*** | 0x02 | |
| ***Sync/Async:*** | Asynchronous | |
| ***Reentrancy:*** | Reentrant (but not for the same NM-Channel) | |
| ***Parameters (in):*** | nmChannelHandle | Identification of the NM-channel |
| | -- | -- |
| | -- | -- |
| ***Parameters (out):*** | None | N/A |
| ***Return value:*** | NM_E_OK | No error |
| | NM_E_NOT_OK | Requesting of network has failed |
| ***Description:*** | Request the network, since ECU needs to communicate on the bus. Network state shall be changed to 'requested' | |
| ***Caveats:*** | CanNm and CNm itself are initialized correctly. | |
| ***Configuration:*** | Optional (Only available if CNM_PASSIVE_MODE_ENABLED is not defined) | |

### 8.3.4 CNm_NetworkRelease

| | | |
|---|---|---|
| ***Service name:*** | CNm_NetworkRelease | |
| ***Syntax:*** | Nm_ReturnType **CNm_NetworkRelease**<br>(<br>  const Nm_ChannelHandleType nmChannelHandle<br>); | |
| ***Service ID:*** | 0x03 | |
| ***Sync/Async:*** | Asynchronous | |
| ***Reentrancy:*** | Reentrant (but not for the same NM-Channel) | |
| ***Parameters (in):*** | nmChannelHandle | Identification of the NM-channel |
| | -- | -- |
| | -- | -- |
| ***Parameters (out):*** | None | N/A |
| ***Return value:*** | NM_E_OK | No error |
| | NM_E_NOT_OK | Releasing of network has failed |
| | NM_E_NOT_EXECUTED | Releasing of network is currently not executed. |
| ***Description:*** | Release the network, since ECU doesn't have to communicate on the bus. Network state shall be changed to 'released'. | |
| ***Caveats:*** | CanNm and CNm itself are initialized correctly. | |
| ***Configuration:*** | Optional (Only available if CNM_PASSIVE_MODE_ENABLED is not defined) | |

### 8.3.5 CNm_DisableCommunication

| | | |
|---|---|---|
| ***Service name:*** | CNm_DisableCommunication | |
| ***Syntax:*** | `Nm_ReturnType CNm_DisableCommunication ( const Nm_ChannelHandleType nmChannelHandle );` | |
| ***Service ID:*** | `0x0C` | |
| ***Sync/Async:*** | Asynchronous | |
| ***Reentrancy:*** | Reentrant (but not for the same NM-Channel) | |
| ***Parameters (in):*** | `nmChannelHandle` | Identification of the NM-channel |
| ***Parameters (out):*** | `None` | N/A |
| ***Return value:*** | `NM_E_OK` | No error |
| | `NM_E_NOT_OK` | Disabling of NM PDU transmission ability has failed. |
| | `NM_E_NOT_EXECUTED` | Disabling of NM PDU transmission ability is not executed. |
| ***Description:*** | Disable the NM PDU transmission ability due to a ISO14229 Communication Control (28hex) service | |
| ***Caveats:*** | CanNm and CNm itself are initialized correctly. | |
| ***Configuration:*** | Optional (Only available if `CNM_COM_CONTROL_ENABLED` is defined) | |

### 8.3.6 CNm_EnableCommunication

| | | |
|---|---|---|
| ***Service name:*** | CNm_EnableCommunication | |
| ***Syntax:*** | `Nm_ReturnType CNm_EnableCommunication ( const Nm_ChannelHandleType nmChannelHandle );` | |
| ***Service ID:*** | `0x0D` | |
| ***Sync/Async:*** | Asynchronous | |
| ***Reentrancy:*** | Reentrant (but not for the same NM-Channel) | |
| ***Parameters (in):*** | `nmChannelHandle` | Identification of the NM-channel |
| ***Parameters (out):*** | `None` | N/A |
| ***Return value:*** | `NM_E_OK` | No error |
| | `NM_E_NOT_OK` | Enabling of NM PDU transmission ability has failed. |
| | `NM_E_NOT_EXECUTED` | Enabling of NM PDU transmission ability is not executed. |
| ***Description:*** | Enable the NM PDU transmission ability due to a ISO14229 Communication Control (28hex) service | |
| ***Caveats:*** | CanNm and CNm itself are initialized correctly. | |
| ***Configuration:*** | Optional (Only available if `CNM_COM_CONTROL_ENABLED` is defined) | |

### 8.3.7 CNm_SetUserData

| | |
|---|---|
| ***Service name:*** | CNm_SetUserData |
| ***Syntax:*** | `Nm_ReturnType CNm_SetUserData ( const Nm_ChannelHandleType nmChannelHandle, const uint8* const nmUserDataPtr );` |

| | | |
|---|---|---|
| ***Service ID:*** | 0x04 | |
| ***Sync/Async:*** | Synchronous | |
| ***Reentrancy:*** | Non-reentrant | |
| ***Parameters (in):*** | nmChannelHandle | Identification of the NM-channel |
| | nmUserDataPtr | Pointer where the user data for the next transmitted NM message shall be copied from. |
| | -- | -- |
| ***Parameters (out):*** | None | N/A |
| ***Return value:*** | NM_E_OK | No error |
| | NM_E_NOT_OK | Setting of user data has failed |
| ***Description:*** | Set user data for NM messages transmitted next on the bus.<br>For that purpose CanNm_SetUserData shall be called. | |
| ***Caveats:*** | CanNm and CNm itself are initialized correctly. | |
| ***Configuration:*** | Optional (Only available if CNM_USER_DATA_ENABLED is defined and CNM_PASSIVE_MODE_ENABLED is not defined) | |

### 8.3.8 CNm_GetUserData

| | | |
|---|---|---|
| ***Service name:*** | CNm_GetUserData | |
| ***Syntax:*** | Nm_ReturnType **CNm_GetUserData**<br>(<br>  const Nm_ChannelHandleType nmChannelHandle,<br>  uint8* const nmUserDataPtr<br>); | |
| ***Service ID:*** | 0x05 | |
| ***Sync/Async:*** | Synchronous | |
| ***Reentrancy:*** | Reentrant | |
| ***Parameters (in):*** | nmChannelHandle | Identification of the NM-channel |
| | -- | -- |
| | -- | -- |
| ***Parameters (out):*** | nmUserDataPtr | Pointer where user data out of the most recently received NM message shall be copied to. |
| | -- | -- |
| ***Return value:*** | NM_E_OK | No error |
| | NM_E_NOT_OK | Getting of user data has failed |
| ***Description:*** | Get user data out of the most recently received NM message.<br>For that purpose CanNm_GetUserData shall be called. | |
| ***Caveats:*** | CanNm and CNm itself are initialized correctly. | |
| ***Configuration:*** | Optional (Only available if CNM_USER_DATA_ENABLED is defined). | |

### 8.3.9 CNm_GetNodeIdentifier

| | |
|---|---|
| ***Service name:*** | CNm_GetNodeIdentifier |
| ***Syntax:*** | Nm_ReturnType **CNm_GetNodeIdentifier**<br>(<br>  const Nm_ChannelHandleType nmChannelHandle,<br>  uint8 * const nmNodeIdPtr<br>); |

| | | |
|---|---|---|
| ***Service ID:*** | `0x06` | |
| ***Sync/Async:*** | Synchronous | |
| ***Reentrancy:*** | Reentrant | |
| ***Parameters (in):*** | `nmChannelHandle` | Identification of the NM-channel |
| | -- | -- |
| | -- | -- |
| ***Parameters (out):*** | `nmNodeIdPtr` | Pointer where node identifier out of the most recently received NM PDU shall be copied to |
| ***Return value:*** | `NM_E_OK` | No error |
| | `NM_E_NOT_OK` | Getting of the node identifier out of the most recently received NM PDU has failed |
| ***Description:*** | Get node identifier out of the most recently received NM PDU.<br>For that purpose `CanNm_GetNodeIdentifier` shall be called. | |
| ***Caveats:*** | CanNm and CNm itself are initialized correctly. | |
| ***Configuration:*** | Optional (Only available if `CNM_NODE_ID_ENABLED` is defined). | |

### 8.3.10 CNm_GetLocalNodeIdentifier

| | | |
|---|---|---|
| ***Service name:*** | CNm_GetLocalNodeIdentifier | |
| ***Syntax:*** | `Nm_ReturnType` **`CNm_GetLocalNodeIdentifier`**<br>`(`<br>`  const Nm_ChannelHandleType nmChannelHandle,`<br>`  uint8 * const nmNodeIdPtr`<br>`);` | |
| ***Service ID:*** | `0x07` | |
| ***Sync/Async:*** | Synchronous | |
| ***Reentrancy:*** | Reentrant | |
| ***Parameters (in):*** | `nmChannelHandle` | Identification of the NM-channel |
| | -- | -- |
| | -- | -- |
| ***Parameters (out):*** | `nmNodeIdPtr` | Pointer where node identifier of the local node shall be copied to |
| ***Return value:*** | `NM_E_OK` | No error |
| | `NM_E_NOT_OK` | Getting of the node identifier of the local node has failed |
| ***Description:*** | Get node identifier configured for the local node.<br>For that purpose `CanNm_GetLocalNodeIdentifier` shall be called. | |
| ***Caveats:*** | CanNm and CNm itself are initialized correctly. | |
| ***Configuration:*** | Optional (Only available if `CNM_NODE_ID_ENABLED` is defined). | |

### 8.3.11 CNm_RepeatMessageRequest

| | |
|---|---|
| ***Service name:*** | CNm_RepeatMessageRequest |
| ***Syntax:*** | `Nm_ReturnType` **`CNm_RepeatMessageRequest`**<br>`(`<br>`  const Nm_ChannelHandleType nmChannelHandle`<br>`);` |
| ***Service ID:*** | `0x08` |
| ***Sync/Async:*** | Asynchronous |
| ***Reentrancy:*** | Reentrant (but not for the same NM-Channel) |

| | | |
|---|---|---|
| ***Parameters (in):*** | `nmChannelHandle` | Identification of the NM-channel |
| | -- | -- |
| | -- | -- |
| ***Parameters (out):*** | `None` | N/A |
| ***Return value:*** | `NM_E_OK` | No error |
| | `NM_E_NOT_OK` | Setting of Repeat Message Request Bit has failed |
| | `NM_E_NOT_EXECUTED` | Repeat Message Request is currently not executed. |
| ***Description:*** | Set Repeat Message Request Bit for NM messages transmitted next on the bus. For that purpose `CanNm_SetRepeatMessageBit` shall be called. | |
| ***Caveats:*** | CanNm and CNm itself are initialized correctly. | |
| ***Configuration:*** | Optional (Only available if `CNM_NODE_DETECTION_ENABLED` is defined) | |

### 8.3.12 CNm_GetPduData

| | | |
|---|---|---|
| ***Service name:*** | CNm_GetPduData | |
| ***Syntax:*** | `Nm_ReturnType CNm_GetPduData ( const Nm_ChannelHandleType nmChannelHandle, uint8 * const nmPduDataPtr );` | |
| ***Service ID:*** | `0x0A` | |
| ***Sync/Async:*** | Synchronous | |
| ***Reentrancy:*** | Reentrant | |
| ***Parameters (in):*** | `nmChannelHandle` | Identification of the NM-channel |
| | -- | -- |
| ***Parameters (out):*** | `nmPduDataPtr` | Pointer where NM PDU shall be copied to. |
| | -- | -- |
| | -- | -- |
| ***Return value:*** | `NM_E_OK` | No error |
| | `NM_E_NOT_OK` | Getting of NM PDU data has failed |
| ***Description:*** | Get the whole PDU data out of the most recently received NM message. For that purpose `CanNm_GetPduData` shall be called. | |
| ***Caveats:*** | CanNm and CNm itself are initialized correctly. | |
| ***Configuration:*** | Optional (Only available if `CNM_NODE_ID_ENABLED` or `CNM_NODE_DETECTION_ENABLED` or `CNM_USER_DATA_ENABLED` is defined). | |

### 8.3.13 CNm_GetState

| | |
|---|---|
| ***Service name:*** | CNm_GetState |
| ***Syntax:*** | `Nm_ReturnType CNm_GetState ( const Nm_ChannelHandleType nmChannelHandle, Nm_StateType * const nmStatePtr, Nm_ModeType * const nmModePtr, );` |
| ***Service ID:*** | `0x0B` |
| ***Sync/Async:*** | Synchronous |
| ***Reentrancy:*** | Reentrant |

| | | |
|---|---|---|
| ***Parameters (in):*** | `nmChannelHandle` | Identification of the NM-channel |
| | -- | -- |
| | -- | -- |
| ***Parameters (out):*** | `nmStatePtr` | Pointer where state of the network management shall be copied to. |
| | `nmModePtr` | Pointer where the mode of the network management shall be copied to. |
| ***Return value:*** | `NM_E_OK` | No error |
| | `NM_E_NOT_OK` | Getting of NM state has failed |
| ***Description:*** | Returns the state and the mode of the network management. | |
| ***Caveats:*** | CanNm and CNm itself are initialized correctly. | |
| ***Configuration:*** | Mandatory | |

### 8.3.14 CNm_GetVersionInfo

| | |
|---|---|
| ***Service name:*** | CNm_GetVersionInfo |
| ***Syntax:*** | `void CNm_GetVersionInfo ( Std_VersionInfoType * versioninfo );` |
| ***Service ID:*** | 0xF1 |
| ***Sync/Async:*** | Synchronous |
| ***Reentrancy:*** | Reentrant |
| ***Parameters (in):*** | `none` |
| ***Parameters (out):*** | `versioninfo` Pointer to where to store the version information of this module. |
| ***Return value:*** | none |
| ***Description:*** | This service returns the version information of this module. The version information includes:<br>- Module Id<br>- Vendor Id<br>- Vendor specific version numbers (BSW00407).<br><br>Note:<br>This function can be called even if the CanNm is not initialized. |
| ***Caveats:*** | -- |
| ***Configuration:*** | Optional (only available if `CNM_VERSION_INFO_API` is defined) |

### 8.3.15 CNm_RequestBusSynchronization

| | |
|---|---|
| ***Service name:*** | CNm_RequestBusSynchronization |
| ***Syntax:*** | `Nm_ReturnType CNm_RequestBusSynchronization ( const Nm_ChannelHandleType mmChannelHandle );` |
| ***Service ID:*** | 0xC0 |
| ***Sync/Async:*** | Asynchronous |
| ***Reentrancy:*** | Non-Reentrant |
| ***Parameters (in):*** | `nmChannelHandle` Identification of the NM-channel |

| | | |
|---|---|---|
| | -- | -- |
| | -- | -- |
| ***Parameters (out):*** | None | N/A |
| ***Return value:*** | NM_E_OK | No error |
| | NM_E_NOT_OK | Requesting of bus synchronization has failed |
| | NM_E_NOT_EXECUTED | Bus synchronization is currently not executed. |
| ***Description:*** | Request bus synchronization. | |
| ***Caveats:*** | CanNm and CNm itself are initialized correctly. | |
| ***Configuration:*** | Optional (Only available if CNM_BUS_SYNCHRONIZATION_ENABLED is defined) and CNM_PASSIVE_NODE_ENABLED is not defined. | |

### 8.3.16 CNm_CheckRemoteSleepIndication

| | | |
|---|---|---|
| ***Service name:*** | CNm_CheckRemoteSleepIndication | |
| ***Syntax:*** | Nm_ReturnType **CNm_CheckRemoteSleepIndication**<br>(<br>  const Nm_ChannelHandleType nmChannelHandle,<br>  boolean * const nmRemoteSleepIndPtr<br>);<br> | |
| ***Service ID:*** | 0xD0 | |
| ***Sync/Async:*** | Synchronous | |
| ***Reentrancy:*** | Reentrant (but not for the same NM-Channel) | |
| ***Parameters (in):*** | nmChannelHandle | Identification of the NM-channel |
| | -- | -- |
| | -- | -- |
| ***Parameters (out):*** | NmRemoteSleepIndPtr | Pointer where check result of remote sleep indication shall be copied to |
| ***Return value:*** | NM_E_OK | No error |
| | NM_E_NOT_OK | Checking of remote sleep indication bits has failed |
| | NM_E_NOT_EXECUTED | Checking of Remote Sleep Indication is currently not executed. |
| ***Description:*** | Check if remote sleep indication takes place or not. | |
| ***Caveats:*** | CanNm and CNm itself are initialized correctly. | |
| ***Configuration:*** | Optional (Only available if CNM_REMOTE_SLEEP_INDICATION_ENABLED is defined) | |

## 8.4 Scheduled functions: Nm Services provided for operating system

### 8.4.1 CNm_TxConfirmation

| | |
|---|---|
| ***Service name:*** | CNm_TxConfirmation |
| ***Syntax:*** | void **CNm_TxConfirmation**<br>(<br>  const Nm_ChannelHandleType nmChannelHandle<br>);<br> |
| ***Service ID:*** | 0xE0 |
| ***Sync/Async:*** | Synchronous |

| | | |
|---|---|---|
| ***Reentrancy:*** | Reentrant (but not for the same NM-Channel) | |
| ***Parameters (in):*** | `nmChannelHandle` | Identification of the NM-channel |
| | -- | -- |
| | -- | -- |
| ***Parameters (out):*** | `None` | N/A |
| ***Return value:*** | `None` | N/A |
| | -- | -- |
| ***Description:*** | Notify CAN Generic NM that a NM message has been successfully transmitted. | |
| ***Caveats:*** | CanNm and CNm itself are initialized correctly. | |
| ***Configuration:*** | Optional (Only available if `CNM_PASSIVE_MODE_ENABLED` is not defined) | |

### 8.4.2 CNm_RxIndication

| | |
|---|---|
| ***Service name:*** | CNm_RxIndication |
| ***Syntax:*** | `void CNm_RxIndication ( const Nm_ChannelHandleType nmChannelHandle );` |
| ***Service ID:*** | `0xE1` |
| ***Sync/Async:*** | Synchronous |
| ***Reentrancy:*** | Reentrant (but not for the same NM-Channel) |
| ***Parameters (in):*** | `nmChannelHandle` Identification of the NM-channel |
| | -- -- |
| | -- -- |
| ***Parameters (out):*** | `None` N/A |
| ***Return value:*** | `None` N/A |
| | -- -- |
| ***Description:*** | Notify CAN Generic NM that a NM message has been received. |
| ***Caveats:*** | CanNm and CNm itself are initialized correctly. |
| ***Configuration:*** | Mandatory |

### 8.4.3 CNm_RepeatMessageBitIndication

| | |
|---|---|
| ***Service name:*** | CNm_RepeatMessageBitIndication |
| ***Syntax:*** | `void CNm_RepeatMessageBitIndication ( const Nm_ChannelHandleType nmChannelHandle );` |
| ***Service ID:*** | `0xE2` |
| ***Sync/Async:*** | Synchronous |
| ***Reentrancy:*** | Reentrant (but not for the same NM-Channel) |
| ***Parameters (in):*** | `nmChannelHandle` Identification of the NM-channel |
| | -- -- |
| | -- -- |
| ***Parameters (out):*** | `None` N/A |
| ***Return value:*** | `None` N/A |
| | -- -- |

| | |
|---|---|
| ***Description:*** | Notify CAN Generic NM that a NM message has been received where a Repeat Message Bit is set. |
| ***Caveats:*** | CanNm and CNm itself are initialized correctly. |
| ***Configuration:*** | Optional (Only available if `CNM_NODE_DETECTION_ENABLED` is defined and `CNM_PASSIVE_MODE_ENABLED` is not defined).<br><br>Note: Repeat Message Bit reception is not indicated on a passive node since a passive node cannot request node detection actively nor respond to a node detection request. |

### 8.4.4 CNm_TxTimeoutException

| | | |
|---|---|---|
| ***Service name:*** | CNm_TxTimeoutException | |
| ***Syntax:*** | `void CNm_TxTimeoutException ( const Nm_ChannelHandleType nmChannelHandle );` | |
| ***Service ID:*** | `0xE3` | |
| ***Sync/Async:*** | Synchronous | |
| ***Reentrancy:*** | Reentrant (but not for the same NM-Channel) | |
| ***Parameters (in):*** | `nmChannelHandle` | Identification of the NM-channel |
| | -- | -- |
| | -- | -- |
| ***Parameters (out):*** | `None` | N/A |
| ***Return value:*** | `None` | N/A |
| | -- | -- |
| ***Description:*** | Notify CAN Generic NM that the NM PDU has not been transmitted successfully for `CANNM_MSG_TIMEOUT_TIME`. | |
| ***Caveats:*** | CanNm and CNm itself are initialized correctly. | |
| ***Configuration:*** | Optional (Only available if `CNM_PASSIVE_MODE_ENABLED` is not defined) | |

Note: This function is superfluous and should be removed in AUTOSAR Phase II.

## 8.5 Scheduled functions: Nm Services provided for operating system

### 8.5.1 CNm_MainFunction_<CNmClstIdx>

| | | |
|---|---|---|
| ***Service name:*** | CNm_MainFunction_<CNmClstIdx> | |
| ***Syntax:*** | `void CNm_MainFunction_<CNmClstIdx> ( void );` | |
| ***Service ID:*** | `0xF0` | |
| ***Sync/Async:*** | Synchronous | |
| ***Reentrancy:*** | Non-Reentrant | |
| ***Parameters (in):*** | None | N/A |
| | -- | -- |

| | | |
|---|---|---|
| | -- | -- |
| ***Parameters (out):*** | None | N/A |
| ***Return value:*** | None | N/A |
| | -- | -- |
| ***Description:*** | Main function of CAN Generic NM.<br>There is one dedicated NM Main Function for each CAN NM cluster.<br>The API names are therefore:<br>• CNm_MainFunction_0() for CAN NM cluster associated with NM channel 0,<br>• CNm_MainFunction_1() for CAN NM cluster associated with NM channel 1, etc. | |
| ***Caveats:*** | None | |
| ***Configuration:*** | Mandatory | |

## 8.6 Expected interfaces: Services called by the CNm

### 8.6.1 Mandatory Interfaces

This chapter presents interfaces required to fulfill mandatory NM functionality.

#### 8.6.1.1 CanNm API

| *API function* | *Module* | *Description* |
|---|---|---|
| `void`<br>**`CanNm_Init`**<br>`(`<br>`const CanNm_ConfigType * const nmConfigPtr`<br>`);` | CanNm | Initialize the CAN specific adaptation layer of AUTOSAR NM for CAN. |

#### 8.6.1.2 Nm Interface CBK

| *CBK function* | *Module* | *Description* |
|---|---|---|
| `void` **`Nm_NetworkStartIndication`**<br>`(`<br>`const Nm_ChannelHandleType`<br>`);` | NM Interface | Notification that a NM PDU has been received in the Bus-Sleep Mode, what indicates that some nodes in the network have already entered the Network Mode. The callback function shall start the network management state machine. |
| `void` **`Nm_NetworkMode`**<br>`(`<br>`const Nm_ChannelHandleType`<br>`);` | NM Interface | Notification that the network management has entered Network Mode. The callback function shall enable transmission of application messages. |
| `void` **`Nm_PrepareBusSleepMode`**<br>`(`<br>`const Nm_ChannelHandleType`<br>`);` | NM Interface | Notification that the network management has entered Prepare Bus-Sleep Mode. The callback function shall disable transmission of application messages. |
| `void` **`Nm_BusSleepMode`**<br>`(`<br>`const Nm_ChannelHandleType`<br>`);` | NM Interface | 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. |

#### 8.6.1.3 Dem API

| *CBK function* | *Module* | *Description* |
|---|---|---|
| `Std_ReturnType` **`Dem_ReportErrorStatus`**<br>`(`<br>`  Dem_EventIdType  EventId,`<br>`  Dem_EventStatusType  EventStatus`<br>`)` | Dem | Service for reporting Errors during start up and normal operation to the DEM. |

### 8.6.2 Optional Interfaces

This chapter presents interfaces required to fulfill optional NM functionality.

#### 8.6.2.1 CanNm API

| *API function* | *Module* | *Description* |
|---|---|---|
| `Nm_ReturnType`<br>**`CanNm_TriggerTransmission`**<br>`(`<br>`  const Nm_ChannelHandleType`<br>`);` | CanNm | Trigger a spontaneous transmission of a NM PDU independently of the normal periodic transmission.<br>*Configuration parameter:*<br>**`CNM_PASSIVE_MODE_ENABLED`** |
| `Nm_ReturnType`<br>**`CanNm_StartTransmission`**<br>`(`<br>`  const Nm_ChannelHandleType`<br>`);` | CanNm | Start transmission of periodic NM messages.<br>*Configuration parameter:*<br>**`CNM_PASSIVE_MODE_ENABLED`** |
| `Nm_ReturnType`<br>**`CanNm_StopTransmission`**<br>`(`<br>`  const Nm_ChannelHandleType`<br>`);` | CanNm | Stop transmission of periodic NM messages.<br>*Configuration parameter:*<br>**`CNM_PASSIVE_MODE_ENABLED`** |
| `Nm_ReturnType`<br>**`CanNm_StartBusLoadReduction`**<br>`(`<br>`  const Nm_ChannelHandleType`<br>`);` | CanNm | Start bus load reduction mechanism for periodic NM messages.<br>*Configuration parameter:*<br>**`CNM_BUS_LOAD_REDUCTION_ENABLED`** |
| `Nm_ReturnType`<br>**`CanNm_StopBusLoadReduction`**<br>`(`<br>`  const Nm_ChannelHandleType`<br>`);` | CanNm | Stop bus load reduction mechanism for periodic NM messages.<br>*Configuration parameter:*<br>**`CNM_BUS_LOAD_REDUCTION_ENABLED`** |
| `Nm_ReturnType` **`CanNm_SetUserData`**<br>`(`<br>`  const Nm_ChannelHandleType,`<br>`  const uint8*`<br>`);` | CanNm | Set user data for NM messages transmitted next on the bus.<br>*Configuration parameter:*<br>**`CNM_USER_DATA_ENABLED`**<br>**`CNM_PASSIVE_MODE_ENABLED`** |
| `Nm_ReturnType` **`CanNm_GetUserData`**<br>`(`<br>`  const Nm_ChannelHandleType,`<br>`  uint8* const`<br>`);` | CanNm | Get user data out of the most recently received NM message.<br>*Configuration parameter:*<br>**`CNM_USER_DATA_ENABLED`** |
| `Nm_ReturnType` | CanNm | Get Node Identifier out of the most |

| | | |
|---|---|---|
| **CanNm_GetNodeIdentifier**<br>(<br>  const Nm_ChannelHandleType,<br>  uint8 * const<br>); | | recently received NM message.<br>*Configuration parameter:*<br>**CNM_NODE_ID_ENABLED** |
| Nm_ReturnType<br>**CanNm_GetLocalNodeIdentifier**<br>(<br>  const Nm_ChannelHandleType,<br>  uint8 * const<br>); | CanNm | Get Node Identifier of the local node.<br>*Configuration parameter:*<br>**CNM_NODE_ID_ENABLED** |
| Nm_ReturnType<br>**CanNm_SetRepeatMessageBit**<br>(<br>  const Nm_ChannelHandleType,<br>  const boolean nmRepeatMessageBit<br>); | CanNm | Set Repeat Message Request Bit for NM messages transmitted next on the bus.<br>*Configuration parameter:*<br>**CNM_NODE_DETECTION_ENABLED** |
| Nm_ReturnType **CanNm_GetPduData**<br>(<br>  const Nm_ChannelHandleType,<br>  uint8 * const<br>); | CanNm | Get NM PDU Data out of the most recently received NM message.<br>*Configuration parameter:*<br>**CNM_NODE_ID_ENABLED**<br>**CNM_NODE_DETECTION_ENABLED**<br>**CNM_USER_DATA_ENABLED** |

### 8.6.2.2 Nm Interface CBK

| *CBK function* | *Module* | *Description* |
|---|---|---|
| void **Nm_RemoteSleepIndication**<br>(<br>  const Nm_ChannelHandleType<br>); | NM Interface | Notification that the network management has detected that all other nodes are ready to sleep. The NM gateway shall check if the Bus is still required.<br>*Configuration parameter:*<br>**CNM_REMOTE_SLEEP_IND_ENABLED** |
| void **Nm_RemoteSleepCancelation**<br>(<br>  const Nm_ChannelHandleType<br>); | NM Interface | Notification that the network management has detected that no more all other nodes are ready to sleep. The NM gateway shall check if the Bus is again required.<br>*Configuration parameter:*<br>**CNM_REMOTE_SLEEP_IND_ENABLED** |
| void **Nm_PduRxIndication**<br>(<br>  const Nm_ChannelHandleType<br>); | NM Interface | Notification that a NM message has been received.<br>*Configuration parameter:*<br>**CNM_PDU_RX_INDICATION_ENABLED** |
| **Nm_StateChangeNotification**<br>(<br>  const Nm_ChannelHandleType,<br>  const Nm_StateType nmPreviousState,<br>  const Nm_StateType nmCurrentState<br>) | NM Interface | Notification that the CAN Generic NM state has changed.<br>*Configuration parameter:*<br>**CNM_STATE_CHANGE_IND_ENABLED** |

### 8.6.2.3 Det API

| *CBK function* | *Module* | *Description* |
|---|---|---|
| void **Det_ReportError** | Det | Service for reporting of development |

| | | |
|---|---|---|
| (<br>uint16 ModuleId,<br>uint8 InstanceId,<br>uint8 ApiId,<br>uint8 ErrorId<br>) | | errors in the development mode.<br>*Configuration parameter:*<br>**CNM_DEV_ERROR_DETECT** |

## 8.7 Parameter check

**CNM131:** If detection of development errors is enabled by CNM_DEV_ERROR_DETECT (configuration parameter), then for all CAN Generic NM API services validity check of input parameters shall be made.
Exception: The NULL Pointer check of input parameters shall not be done for **CNm_Init**.

**CNM132:** Parameter type checking shall be made at compile time; if types do not fit the compilation process shall be stopped and respective compilation warnings or errors shall be returned as far as supported by the compiler.

**CNM133:** Parameter value check (for parameters of the constant value) shall be made at configuration time; if the value is invalid, the configuration process shall be stopped and respective configuration error shall be reported .

**CNM134:** Parameter value check (for parameters of the variable value) shall be made at execution time; if the value is invalid, execution of a service shall be rejected and respective development error shall be reported .

## 8.8 Version check

**CNM135:** NM shall check at pre-compile time if the version numbers of C- and H-files are identical.

## 8.9 UML State chart diagram

The Figure 8-2 shows an UML state diagram with respect to the API specification. Mode change related transitions are denoted in green, error handling related transitions in red and optional node detection related transitions in blue. Additionally it is assumed that optional busload reduction functionality is enabled.

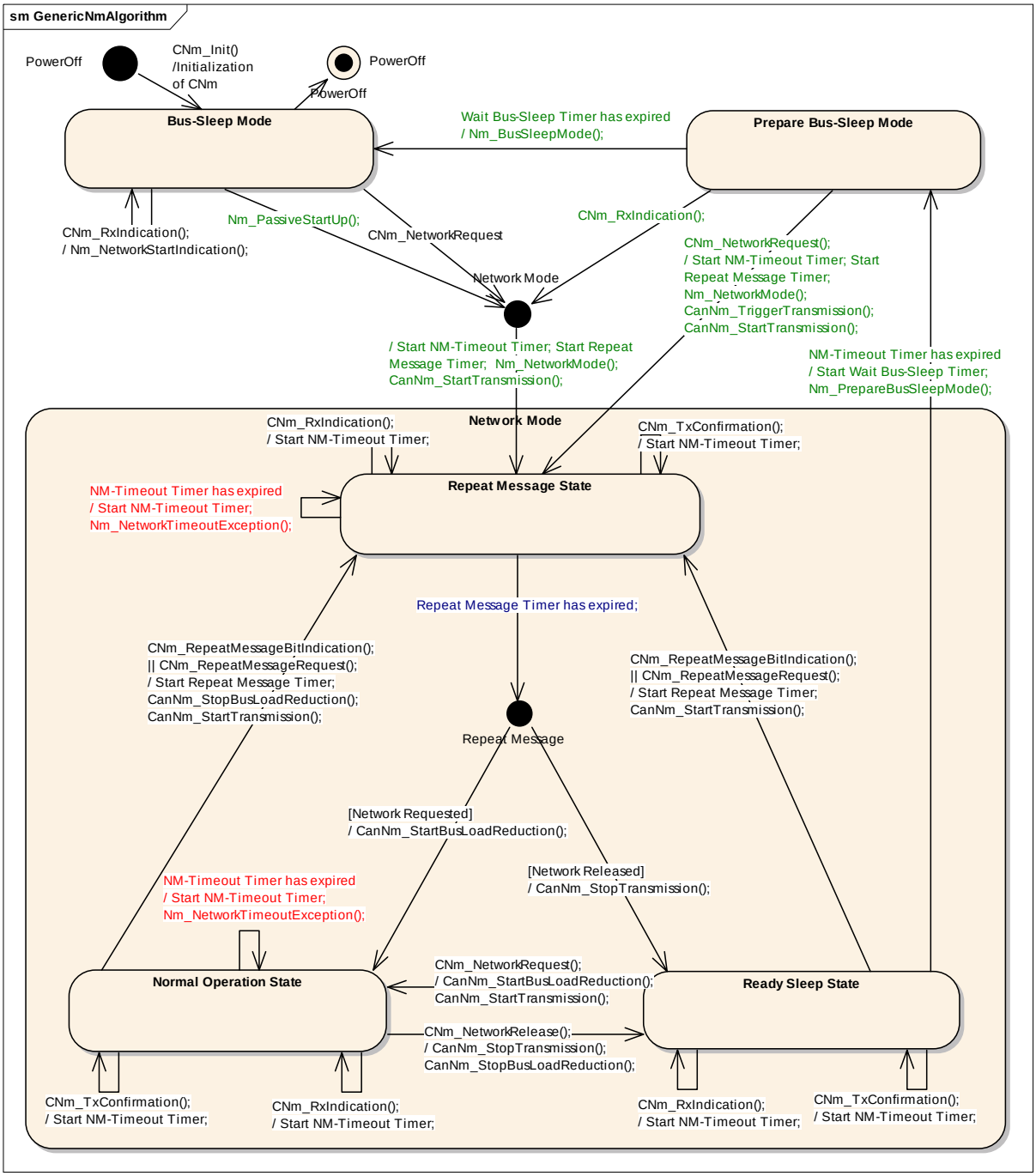
sm GenericNmAlgorithm
PowerOff
CNm_Init()
/Initialization
of CNm
PowerOff
PowerOff
Bus-Sleep Mode
Wait Bus-Sleep Timer has expired
/ Nm_BusSleepMode();
Prepare Bus-Sleep Mode
CNm_RxIndication();
/ Nm_NetworkStartIndication();
Nm_PassiveStartUp();
CNm_NetworkRequest
CNm_RxIndication();
CNm_NetworkRequest();
/ Start NM-Timeout Timer; Start
Repeat Message Timer;
Nm_NetworkMode();
CanNm_TriggerTransmission();
CanNm_StartTransmission();
Network Mode
/ Start NM-Timeout Timer; Start Repeat
Message Timer; Nm_NetworkMode();
CanNm_StartTransmission();
NM-Timeout Timer has expired
/ Start Wait Bus-Sleep Timer;
Nm_PrepareBusSleepMode();
Network Mode
CNm_RxIndication();
/ Start NM-Timeout Timer;
CNm_TxConfirmation();
/ Start NM-Timeout Timer;
NM-Timeout Timer has expired
/ Start NM-Timeout Timer;
Nm_NetworkTimeoutException();
Repeat Message State
Repeat Message Timer has expired;
CNm_RepeatMessageBitIndication();
|| CNm_RepeatMessageRequest();
/ Start Repeat Message Timer;
CanNm_StopBusLoadReduction();
CanNm_StartTransmission();
CNm_RepeatMessageBitIndication();
|| CNm_RepeatMessageRequest();
/ Start Repeat Message Timer;
CanNm_StartTransmission();
Repeat Message
[Network Requested]
/ CanNm_StartBusLoadReduction();
[Network Released]
/ CanNm_StopTransmission();
NM-Timeout Timer has expired
/ Start NM-Timeout Timer;
Nm_NetworkTimeoutException();
Normal Operation State
CNm_NetworkRequest();
/ CanNm_StartBusLoadReduction();
CanNm_StartTransmission();
Ready Sleep State
CNm_NetworkRelease();
/ CanNm_StopTransmission();
CanNm_StopBusLoadReduction();
CNm_TxConfirmation();
/ Start NM-Timeout Timer;
CNm_RxIndication();
/ Start NM-Timeout Timer;
CNm_RxIndication();
/ Start NM-Timeout Timer;
CNm_TxConfirmation();
/ Start NM-Timeout Timer;

**Figure 8-2**

# 9 Sequence diagrams

## 9.1 Use Case 01 – Initialization

Sequence in Figure 9-1 shows how to initialize the CAN Generic Network Management.

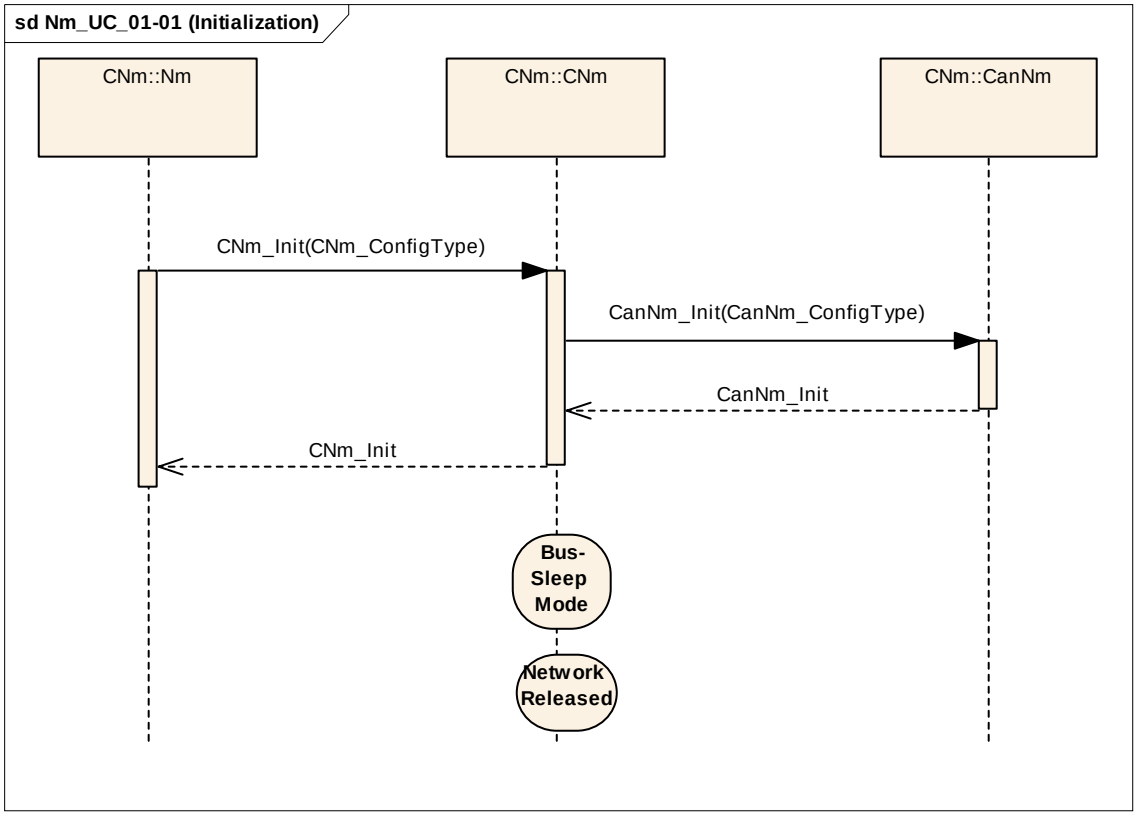


**Figure 9-1**

## 9.2 Use Case 02 – Regular Passive Startup

Sequence in Figure 9-2 shows the regular passive startup of the CAN Generic Network Management.

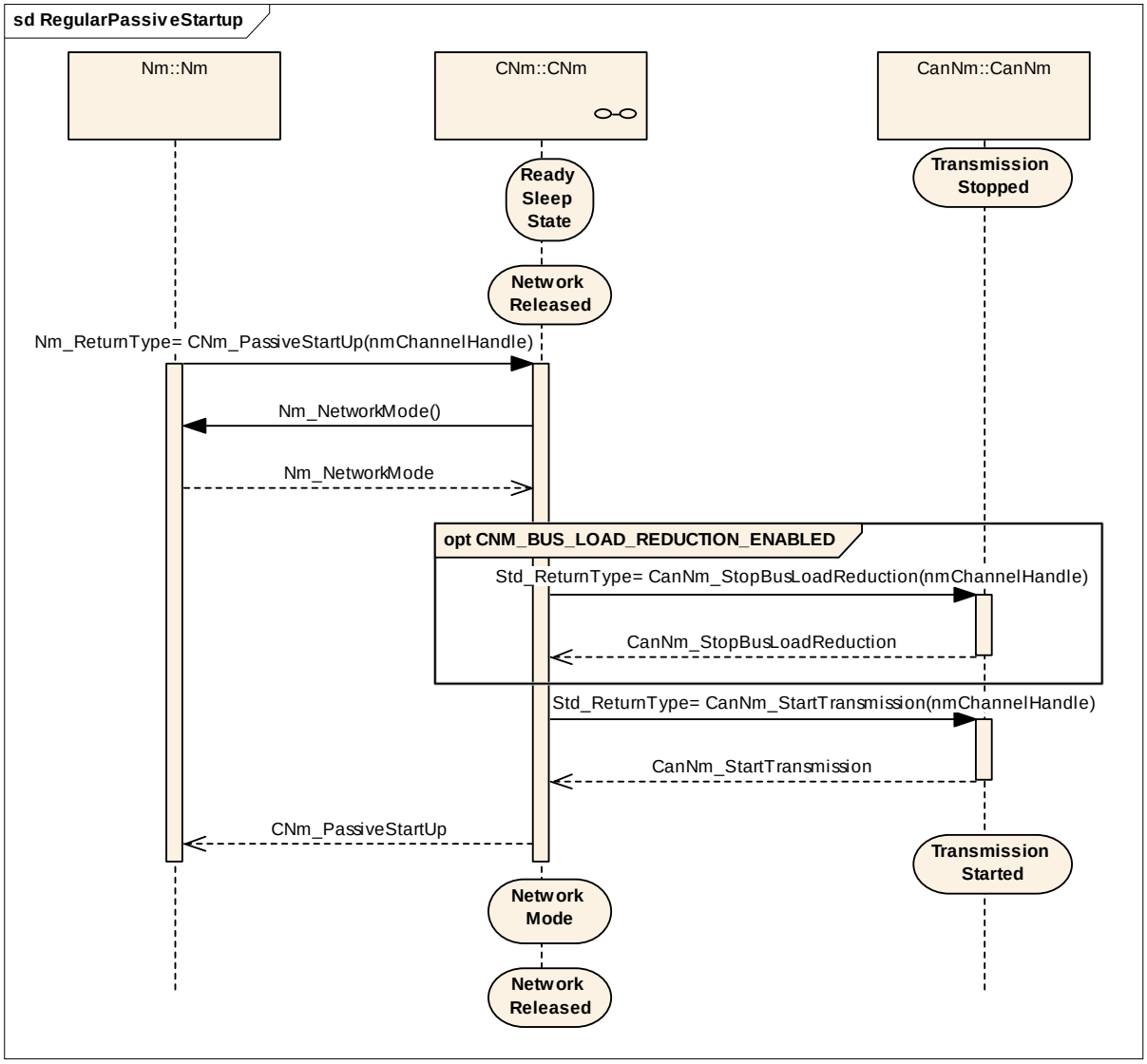
sd RegularPassiveStartup
Nm::Nm
CNm::CNm
CanNm::CanNm
Ready Sleep State
Transmission Stopped
Network Released
Nm_ReturnType= CNm_PassiveStartUp(nmChannelHandle)
Nm_NetworkMode()
Nm_NetworkMode
opt CNM_BUS_LOAD_REDUCTION_ENABLED
Std_ReturnType= CanNm_StopBusLoadReduction(nmChannelHandle)
CanNm_StopBusLoadReduction
Std_ReturnType= CanNm_StartTransmission(nmChannelHandle)
CanNm_StartTransmission
CNm_PassiveStartUp
Transmission Started
Network Mode
Network Released

**Figure 9-2**

## 9.3 Use Case 03 – Passive Startup during shutdown

Sequence in figure 9-3 shows the passive startup during shutdown of the CAN Generic Network Management. It is valid if a NM PDU is received in the time between Bus-Sleep Mode Notification from NM to ComM and switching off the bus transceivers by ComM.

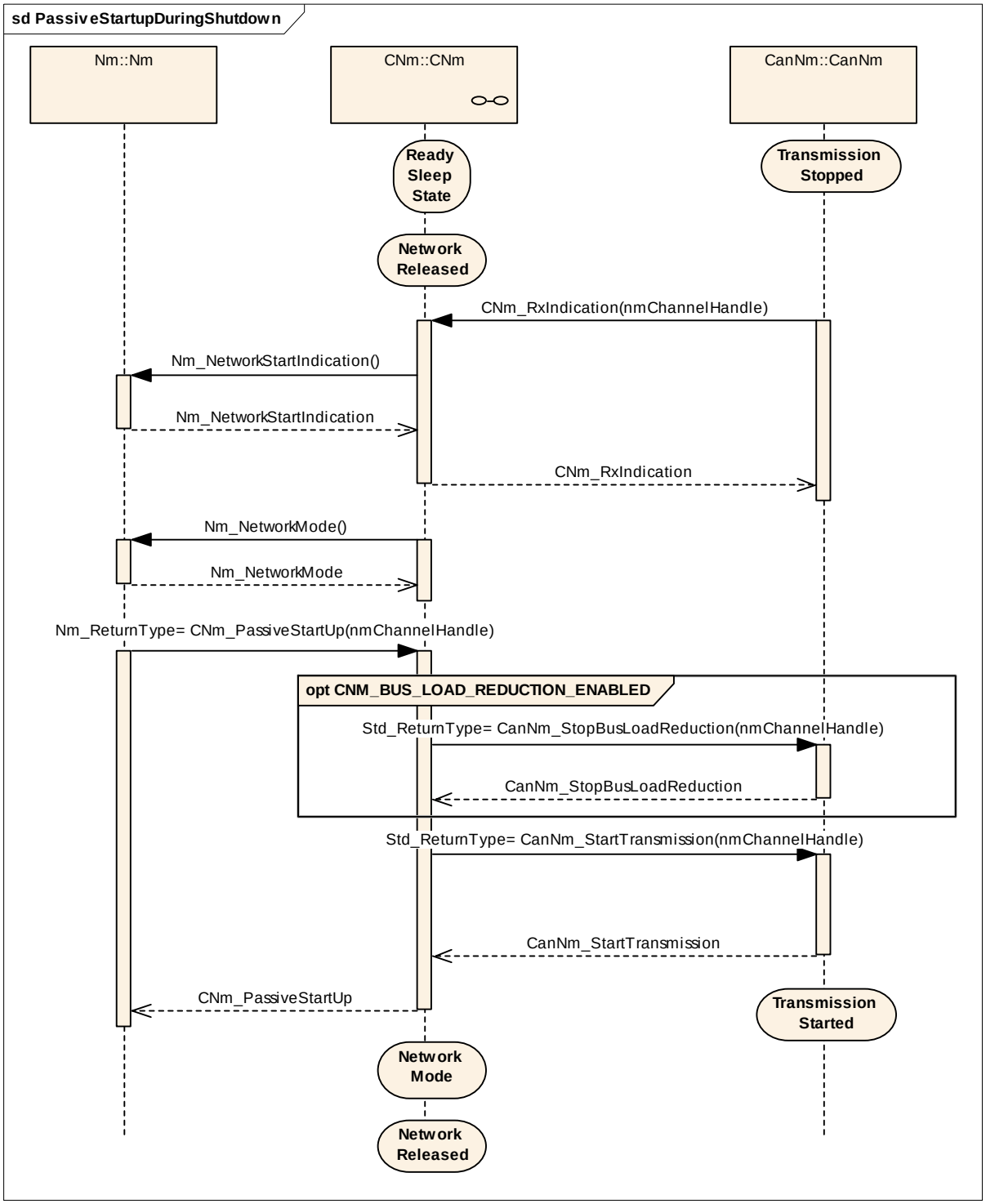


**Figure 9-3**

## 9.4 Use Case 04 – Normal operation

Sequence in Figure 9-4 shows how to request or release the network.

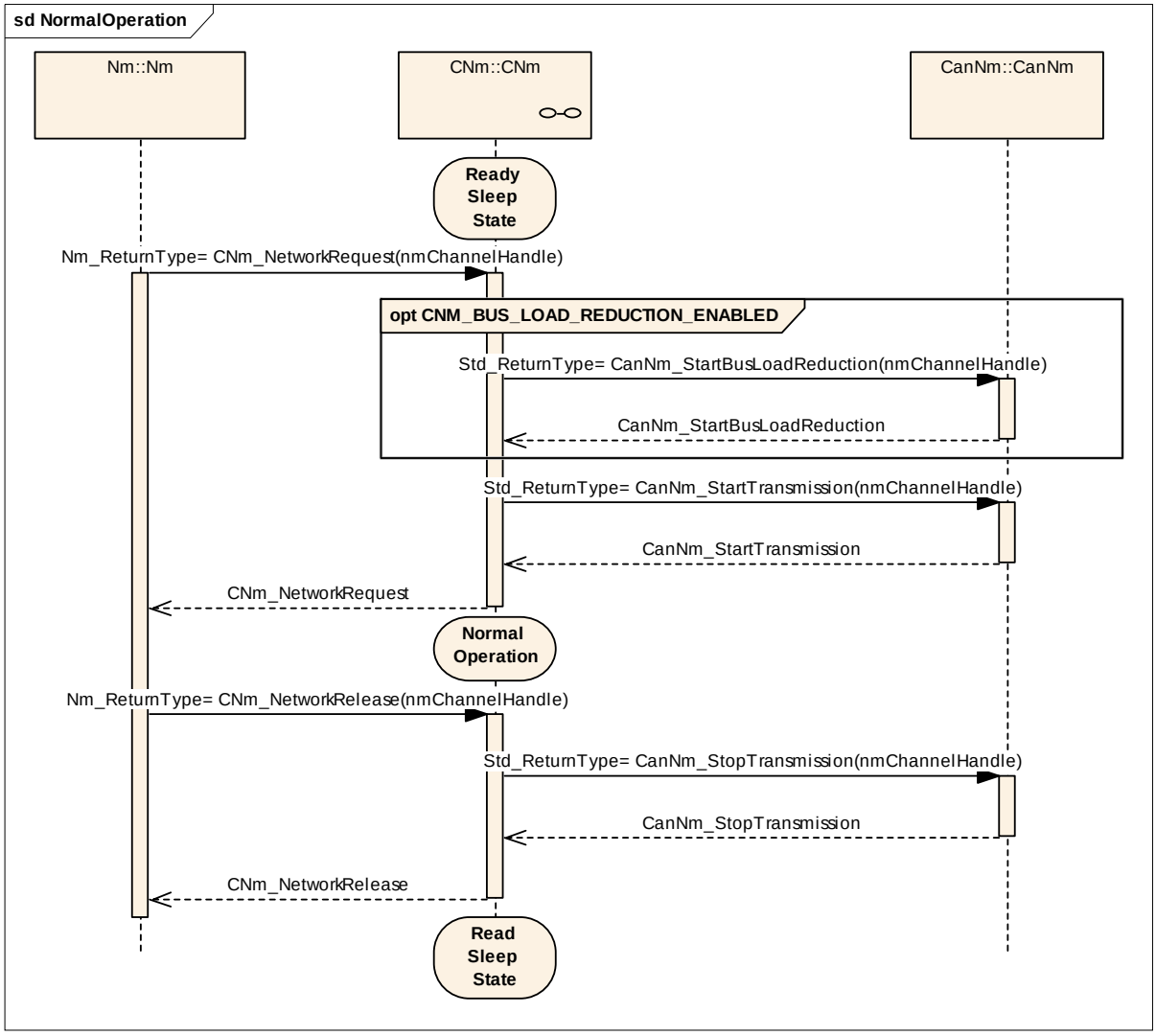


**Figure 9-4**

## 9.5 Use Case 05 – Information services

Sequence in Figure 9-5 shows how to get or respectively set the CAN Generic Network Management related data, i.e. user data, node identifier list, local node identifier and the current state of the coordination algorithm.

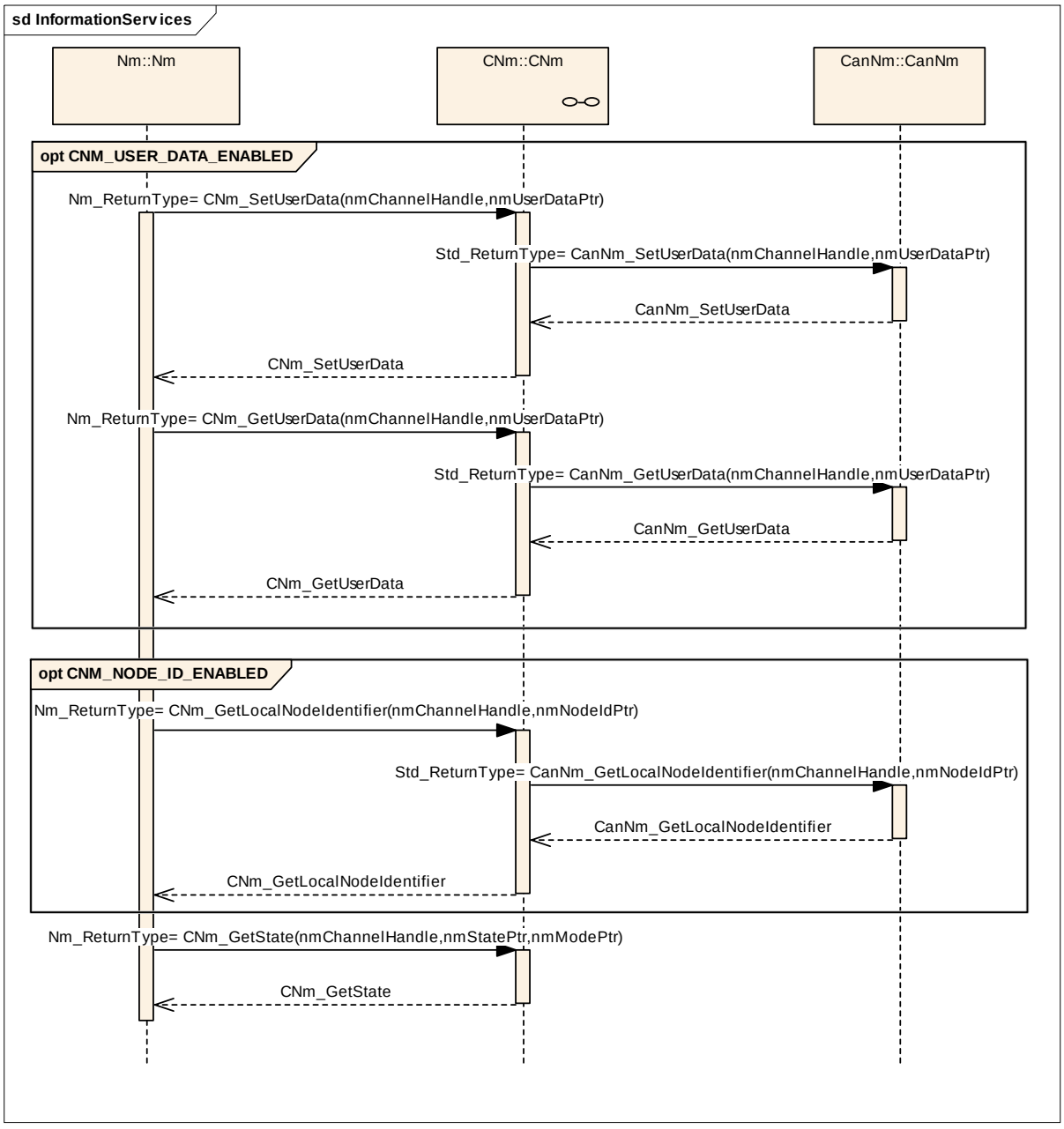


**Figure 9-5**

## 9.6 Use Case 06 – Repeat Message Bit Indication

Sequence in Figure 9-6 shows how to respond to the Repeat Message Bit Indication.

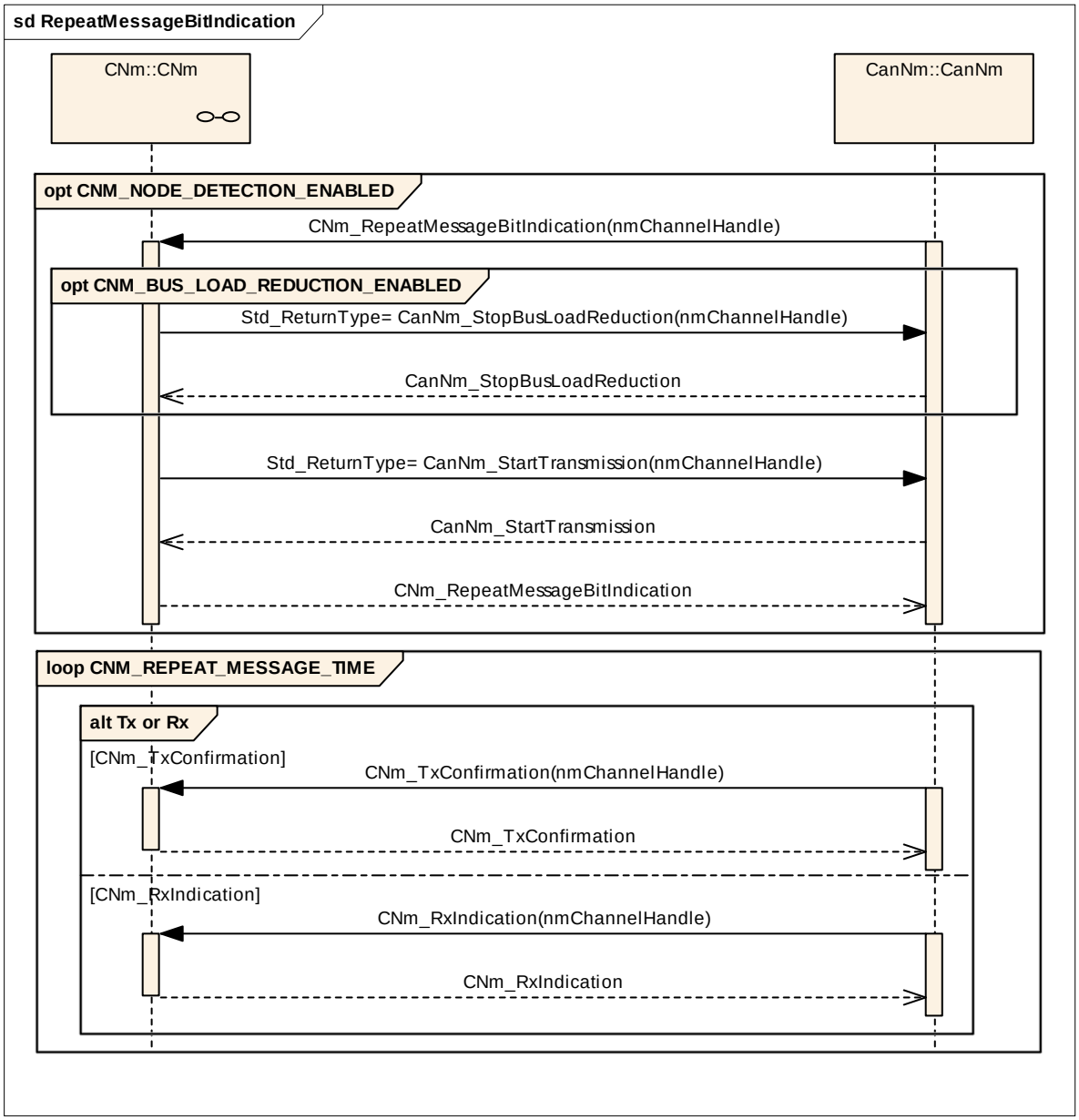


**Figure 9-6**

# 10 Configuration specification

Configuration specification

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

**CNM136:** Both static and runtime configuration parameters shall be located outside the source code of the module .

In general, this chapter defines configuration parameters and their clustering into containers. In order to support the specification Chapter 10.1 describes fundamentals. It also specifies a template (table) you shall use for the parameter specification. Chapter 10.2 specifies the structure (containers) and the parameters of the module <Module Name>. Chapter 10.3 specifies published information of the module <Module Name>.

## 10.1 How to read this chapter

In addition to this section, it is highly recommended to read the documents:

- AUTOSAR Layered Software Architecture
- AUTOSAR ECU Configuration Specification
  This document describes the AUTOSAR configuration methodology and the AUTOSAR configuration meta model in detail.

The following is only a short survey of the topic and it will not replace the ECU Configuration Specification document.

### 10.1.1 Configuration and configuration parameters

Configuration parameters define the variability of the generic part(s) of an implementation of a module. This means that only generic or configurable module implementation can be adapted to the environment (software/hardware) in use during system and/or ECU configuration.

The configuration of parameters can be achieved at different times during the software process: before compile time, before link time or after build time. In the following, the term “configuration class” (of a parameter) shall be used in order to refer to a specific configuration point in time.

### 10.1.2 Variants

Variants describe sets of configuration parameters. E.g., variant 1: only pre-compile time configuration parameters; variant 2: mix of pre-compile- and post build time-

configuration parameters. In one variant a parameter can only be of one configuration class.

### 10.1.3 Containers

Containers structure the set of configuration parameters. This means:

- *all* configuration parameters are kept in containers.
- (sub-) containers can reference (sub-) containers. It is possible to assign a multiplicity to these references. The multiplicity then defines the possible number of instances of the contained parameters.

### 10.1.4 Specification template for configuration parameters

The following tables consist of three sections:

- the general section
- the configuration parameter section
- the section of included/referenced containers

Pre-compile time - specifies whether the configuration parameter shall be of configuration class *Pre-compile time* or not

| ***Label*** | ***Description*** |
|---|---|
| x | The configuration parameter shall be of configuration class *Pre-compile time*. |
| -- | The configuration parameter shall never be of configuration class *Pre-compile time*. |

Link time - specifies whether the configuration parameter shall be of configuration class *Link time* or not

| ***Label*** | ***Description*** |
|---|---|
| x | The configuration parameter shall be of configuration class *Link time*. |
| -- | The configuration parameter shall never be of configuration class *Link time*. |

Post Build - specifies whether the configuration parameter shall be of configuration class *Post Build* or not

| ***Label*** | ***Description*** |
|---|---|
| x | The configuration parameter shall be of configuration class *Post Build* and no specific implementation is required. |
| L | *Loadable* - the configuration parameter shall be of configuration class *Post Build* and only one configuration parameter set resides in the ECU. |
| M | *Multiple* - the configuration parameter shall be of configuration class *Post Build* and is selected out of a set of multiple parameters by passing a dedicated pointer to the init function of the module. |
| -- | The configuration parameter shall never be of configuration class *Post Build*. |

## 10.2 Containers and configuration parameters

The configuration parameters as defined in this chapter are used to create a data model for an AUTOSAR tool chain. The realization in the code is implementation specific.

### 10.2.1 Variants

Variant 1: All configuration parameters configurable at pre-compile time.
Use case: Source code optimizations

Variant 2: All configuration parameters of the container **`CNm_GlobalConfig`** related to enable or disable an optional feature shall be configurable at pre-compile time; the remaining configuration parameters shall be configurable at link time.
Use case: Object code.

Variant 3: The parameters contained in **`CNm_ChannelConfig`** are configurable at post-build time. The parameters contained in **`CNm_GlobalConfig`** are configurable at pre-compile time
Use case: ECU configuration can be flashed (L) and selected during initialization phase (M).

### 10.2.2 CNm_GlobalConfig

CNM143: The Global Scope specifies configuration parameters that shall be defined in the module's configuration header file **`CNm_Cfg.h`**.

| | |
|---|---|
| ***SWS Item*** | -- |
| ***Container Name*** | CNm_GlobalConfig |
| ***Description*** | This container contains all global configuration parameters of CAN Generic NM configured from the NM Module perspective. |
| ***Configuration Parameters*** | |

| | | | |
|---|---|---|---|
| ***Name*** | **`CNM_DEV_ERROR_DETECT`** | | |
| ***Description*** | Pre-processor switch for enabling development error detection support. | | |
| ***Type*** | Boolean | | |
| ***Unit*** | -- | | |
| ***Range*** | TRUE | Option is enabled | |
| | FALSE | Option is disabled | |
| ***Configuration Class*** | ***Pre-compile*** | x | Variant 1, 2, 3 |
| | ***Link time*** | -- | |
| | ***Post Build*** | -- | |
| ***Scope*** | Module | | |
| ***Dependency*** | None | | |

| | |
|---|---|
| ***Name*** | **`CNM_VERSION_INFO_API`** |
| ***Description*** | Pre-processor switch for enabling version info API support. |
| ***Type*** | Boolean |
| ***Unit*** | -- |

| | | | |
|---|---|---|---|
| ***Range*** | TRUE | Option is enabled | |
| | FALSE | Option is disabled | |
| ***Configuration Class*** | ***Pre-compile*** | x | Variant 1, 2, 3 |
| | ***Link time*** | -- | |
| | ***Post Build*** | -- | |
| ***Scope*** | Module | | |
| ***Dependency*** | None | | |

| | | | |
|---|---|---|---|
| ***Name*** | **CNM_USER_DATA_ENABLED** | | |
| ***Description*** | Pre-processor switch for enabling user data support. | | |
| ***Type*** | Boolean | | |
| ***Unit*** | -- | | |
| ***Range*** | TRUE | Option is enabled | |
| | FALSE | Option is disabled | |
| ***Configuration Class*** | ***Pre-compile*** | x | Variant 1, 2, 3 |
| | ***Link time*** | -- | |
| | ***Post Build*** | -- | |
| ***Scope*** | Module | | |
| ***Dependency*** | This parameter shall be derived from `NM_USER_DATA_ENABLED.` | | |

| | | | |
|---|---|---|---|
| ***Name*** | **CNM_REMOTE_SLEEP_IND_ENABLED** | | |
| ***Description*** | Pre-processor switch for enabling remote sleep indication support. | | |
| ***Type*** | Boolean | | |
| ***Unit*** | -- | | |
| ***Range*** | TRUE | Option is enabled | |
| | FALSE | Option is disabled | |
| ***Configuration Class*** | ***Pre-compile*** | x | Variant 1, 2, 3 |
| | ***Link time*** | -- | |
| | ***Post Build*** | -- | |
| ***Scope*** | Module | | |
| ***Dependency*** | This feature is required for gateway nodes only.<br>It must not be defined if **CNM_PASSIVE_MODE_ENABLED** is defined.<br>This parameter shall be derived from<br>`NM_REMOTE_SLEEP_IND_ENABLED.` | | |

| | | | |
|---|---|---|---|
| ***Name*** | **CNM_BUS_SYNCHRONIZATION_ENABLED** | | |
| ***Description*** | Pre-processor switch for enabling bus synchronization support. | | |
| ***Type*** | Boolean | | |
| ***Unit*** | -- | | |
| ***Range*** | TRUE | Option is enabled | |
| | FALSE | Option is disabled | |
| ***Configuration Class*** | ***Pre-compile*** | x | Variant 1, 2, 3 |
| | ***Link time*** | -- | |
| | ***Post Build*** | -- | |
| ***Scope*** | Module | | |
| ***Dependency*** | This feature is required for gateway nodes only.<br>It must not be defined if **CNM_PASSIVE_MODE_ENABLED** is defined.<br>This parameter shall be derived from<br>`NM_BUS_SYNCHRONIZATION_ENABLED.` | | |

| | | | |
|---|---|---|---|
| ***Name*** | **CNM_NODE_DETECTION_ENABLED** | | |
| ***Description*** | Pre-processor switch for enabling the node detection support. | | |
| ***Type*** | Boolean | | |
| ***Unit*** | -- | | |
| ***Range*** | TRUE | Option is enabled | |
| | FALSE | Option is disabled | |
| ***Configuration Class*** | ***Pre-compile*** | x | Variant 1, 2, 3 |
| | ***Link time*** | -- | |

| | | | |
|---|---|---|---|
| | ***Post Build*** | -- | |
| ***Scope*** | Module | | |
| ***Dependency*** | This parameter shall be derived from `NM_NODE_DETECTION_ENABLED`.<br>This parameter shall only be enabled if `CNM_NODE_ID_ENABLED` is defined. | | |

| | | | |
|---|---|---|---|
| ***Name*** | **`CNM_BUS_LOAD_REDUCTION_ENABLED`** | | |
| ***Description*** | Pre-processor switch for enabling busload reduction support. | | |
| ***Type*** | Boolean | | |
| ***Unit*** | -- | | |
| ***Range*** | TRUE | Option is enabled | |
| | FALSE | Option is disabled | |
| ***Configuration Class*** | ***Pre-compile*** | x | Variant 1, 2, 3 |
| | ***Link time*** | -- | |
| | ***Post Build*** | -- | |
| ***Scope*** | Module | | |
| ***Dependency*** | Must not be defined if **`CNM_PASSIVE_MODE_ENABLED`** is defined.<br>This parameter shall be derived from `NM_PASSIVE_MODE_ENABLED`. | | |

| | | | |
|---|---|---|---|
| ***Name*** | **`CNM_NODE_ID_ENABLED`** | | |
| ***Description*** | Pre-processor switch for enabling the source node identifier. | | |
| ***Type*** | Boolean | | |
| ***Unit*** | -- | | |
| ***Range*** | TRUE | Option is enabled | |
| | FALSE | Option is disabled | |
| ***Configuration Class*** | ***Pre-compile*** | x | Variant 1, 2, 3 |
| | ***Link time*** | -- | |
| | ***Post Build*** | -- | |
| ***Scope*** | Module | | |
| ***Dependency*** | None | | |

| | | | |
|---|---|---|---|
| ***Name*** | **`CNM_IMMEDIATE_RESTART_ENABLED`** | | |
| ***Description*** | Pre-processor switch for enabling the asynchronous transmission of a NM PDU upon bus-communication request in Prepare-Bus-Sleep mode.. | | |
| ***Type*** | Boolean | | |
| ***Unit*** | -- | | |
| ***Range*** | TRUE | Option is enabled | |
| | FALSE | Option is disabled | |
| ***Configuration Class*** | ***Pre-compile*** | x | Variant 1, 2, 3 |
| | ***Link time*** | -- | |
| | ***Post Build*** | -- | |
| ***Scope*** | Module | | |
| ***Dependency*** | Must not be defined if **`CNM_PASSIVE_MODE_ENABLED`** is defined. | | |

| | | | |
|---|---|---|---|
| ***Name*** | **`CNM_PASSIVE_MODE_ENABLED`** | | |
| ***Description*** | Pre-processor switch for enabling support of the Passive Mode. | | |
| ***Type*** | Boolean | | |
| ***Unit*** | -- | | |
| ***Range*** | TRUE | Option is enabled | |
| | FALSE | Option is disabled | |
| ***Configuration Class*** | ***Pre-compile*** | x | Variant 1, 2, 3 |
| | ***Link time*** | -- | |
| | ***Post Build*** | -- | |
| ***Scope*** | Module | | |
| ***Dependency*** | This parameter shall be derived from `NM_PASSIVE_NODE_ENABLED`. | | |
| ***Name*** | **`CNM_PDU_RX_INDICATION_ENABLED`** | | |

| | | | |
|---|---|---|---|
| ***Description*** | Pre-processor switch for enabling the PDU Rx Indication. | | |
| ***Type*** | Boolean | | |
| ***Unit*** | -- | | |
| ***Range*** | TRUE | Option is enabled | |
| | FALSE | Option is disabled | |
| ***Configuration Class*** | ***Pre-compile*** | x | Variant 1, 2, 3 |
| | ***Link time*** | -- | |
| | ***Post Build*** | -- | |
| ***Scope*** | Module | | |
| ***Dependency*** | This parameter shall be derived from `NM_PDU_RX_INDICATION_ENABLED`. | | |

| | | | |
|---|---|---|---|
| ***Name*** | **`CNM_STATE_CHANGE_IND_ENABLED`** | | |
| ***Description*** | Pre-processor switch for enabling the CAN Generic NM state change notification. | | |
| ***Type*** | Boolean | | |
| ***Unit*** | -- | | |
| ***Range*** | TRUE | Option is enabled | |
| | FALSE | Option is disabled | |
| ***Configuration Class*** | ***Pre-compile*** | x | Variant 1, 2, 3 |
| | ***Link time*** | -- | |
| | ***Post Build*** | -- | |
| ***Scope*** | Module | | |
| ***Dependency*** | This parameter shall be derived from `NM_STATE_CHANGE_ID_ENABLED`. | | |

| | | | |
|---|---|---|---|
| ***Name*** | **`CNM_COM_CONTROL_ENABLED`** | | |
| ***Description*** | Pre-processor switch for enabling the Communication Control support. | | |
| ***Type*** | Boolean | | |
| ***Unit*** | -- | | |
| ***Range*** | TRUE | Option is enabled | |
| | FALSE | Option is disabled | |
| ***Configuration Class*** | ***Pre-compile*** | x | Variant 1, 2, 3 |
| | ***Link time*** | -- | |
| | ***Post Build*** | -- | |
| ***Scope*** | Module | | |
| ***Dependency*** | This parameter shall be derived from `NM_COM_CONTROL_ENABLED`. | | |

| | | | |
|---|---|---|---|
| ***Name*** | **`CNM_NUMBER_OF_CHANNELS`** | | |
| ***Description*** | Number of NM channels allowed within one ECU. | | |
| ***Type*** | Integer | | |
| ***Unit*** | `--` | | |
| ***Range*** | `1..255` | | |
| ***Configuration Class*** | ***Pre-compile*** | x | Variant 1, 2, 3 |
| | ***Link time*** | -- | |
| | ***Post Build*** | -- | |
| ***Scope*** | Module | | |
| ***Dependency*** | None | | |

| | | | |
|---|---|---|---|
| ***Name*** | **`CNM_CANNM_CONFIG_PTR`** | | |
| ***Description*** | Pointer to configuration of CanNm. | | |
| ***Type*** | Integer | | |
| ***Unit*** | `--` | | |
| ***Range*** | `uint32` | | |
| ***Configuration Class*** | ***Pre-compile*** | -- | |
| | ***Link time*** | -- | |
| | ***Post Build*** | M;L | Variant 1, 2, 3 |
| ***Scope*** | Module | | |

| | |
|---|---|
| ***Dependency*** | None |

| ***Included Containers*** | | | |
|---|---|---|---|
| ***Container Name*** | ***Multiplicity*** | ***Scope*** | ***Dependency*** |
| CNm_ChannelConfig | 1..* | Instance | CNm_ChannelConfig |

### 10.2.3 CNm_ChannelConfig

**CNM137:** The container CNm_ChannelConfig specifies configuration parameters that shall be located in a data structure of type **`CNm_ConfigType`** .

**CNM138:** Runtime configurable parameters listed below shall be configurable for each NM-cluster separately.

| | |
|---|---|
| ***SWS Item*** | -- |
| ***Container Name*** | CNm_ChannelConfig |
| ***Description*** | This container contains the channel specific configuration parameter of the CNm. |
| ***Configuration Parameters*** | |

| | | | |
|---|---|---|---|
| ***Name*** | **`CNM_CHANNEL_ID`** | | |
| ***Description*** | Channel identifier configured for the respective AUTOSAR NM cluster. | | |
| ***Type*** | Integer | | |
| ***Unit*** | `--` | | |
| ***Range*** | `0..254` | | |
| ***Configuration Class*** | ***Pre-compile*** | x | Variant 1 |
| | ***Link time*** | x | Variant 2, 3 |
| | ***Post Build*** | -- | |
| ***Scope*** | Instance | | |
| ***Dependency*** | It is used by referring to the respective NM channel handle. It must be unique for each AUTOSAR NM cluster within one ECU.<br>This parameter shall be derived from `NM_CHANNEL_ID.` | | |

| | | | |
|---|---|---|---|
| ***Name*** | **`CNM_INSTANCE_ID`** | | |
| ***Description*** | Instance identifier configured for the respective AUTOSAR NM cluster.<br>It is defined as follows:<br>most significant byte = NM_MODULE_ID,<br>least significant byte = NM_CHANNEL_ID | | |
| ***Type*** | Integer | | |
| ***Unit*** | `--` | | |
| ***Range*** | `1E00h..1EFEh` | | |
| ***Configuration Class*** | ***Pre-compile*** | x | Variant 1 |
| | ***Link time*** | x | Variant 2, 3 |
| | ***Post Build*** | -- | |
| ***Scope*** | Instance | | |
| ***Dependency*** | It is used by reporting of development errors to DET. It must be unique for each AUTOSAR NM cluster within one ECU.<br>This parameter shall be derived from `NM_INSTANCE_ID.` | | |
| ***Name*** | **`CNM_CHANNEL_ACTIVE`** | | |
| ***Description*** | It determines if the respective NM channel is active or not. | | |

| | | | |
|---|---|---|---|
| | Indicates whether a particular NM-channel shall be initialized (**TRUE**) or not (**FALSE**). | | |
| ***Type*** | Boolean | | |
| ***Unit*** | -- | | |
| ***Range*** | TRUE | NM not active on the channel | |
| | FALSE | NM active on the channel | |
| ***Configuration Class*** | ***Pre-compile*** | x | Variant 1 |
| | ***Link time*** | x | Variant 2, 3 |
| | ***Post Build*** | -- | |
| ***Scope*** | Instance | | |
| ***Dependency*** | None | | |

| | | | |
|---|---|---|---|
| ***Name*** | **CNM_BUS_LOAD_REDUCTION_ACTIVE** | | |
| ***Description*** | It determines if bus load reduction for the respective NM channel is active or not. | | |
| ***Type*** | Boolean | | |
| ***Unit*** | -- | | |
| ***Range*** | TRUE | Bus load reduction active on channel | |
| | FALSE | Bus load reduction inactive on channel | |
| ***Configuration Class*** | ***Pre-compile*** | x | Variant 1 |
| | ***Link time*** | x | Variant 2, 3 |
| | ***Post Build*** | -- | |
| ***Scope*** | Instance | | |
| ***Dependency*** | If CNM_BUS_LOAD_REDUCTION_ACTIVE is set to TRUE CNM_BUS_LOAD_REDUCTION_ENABLED must be set to TRUE. This parameter shall only be available if CNM_BUS_LOAD_REDUCTION_ENABLED is set to TRUE. | | |

| | | | |
|---|---|---|---|
| ***Name*** | **CNM_TIMEOUT_TIME** | | |
| ***Description*** | Network Timeout for NM PDUs.<br>It denotes the time [ms] how long the NM shall stay in the Network Mode before transition into Prepare Bus-Sleep Mode shall take place. | | |
| ***Type*** | Integer | | |
| ***Unit*** | [ms] | | |
| ***Range*** | 1..65535 | | |
| ***Configuration Class*** | ***Pre-compile*** | x | Variant 1 |
| | ***Link time*** | x | Variant 2, 3 |
| | ***Post Build*** | -- | |
| ***Scope*** | Instance | | |
| ***Dependency*** | It shall be equal for all nodes in the cluster.<br>It shall be greater than CANNM_MSG_CYCLE_TIME.<br>Typically it should be equal to: n × CANNM_ MSG_CYCLE_TIME, where n denotes the number of NM PDU cycle times in the Ready Sleep State before transition into the Bus-Sleep Mode is initiated.<br>The value of n decremented by one determines the amount of lost NM PDUs that can be tolerated by the coordination algorithm. | | |

| | | | |
|---|---|---|---|
| ***Name*** | **CNM_WAIT_BUS_SLEEP_TIME** | | |
| ***Description*** | Timeout for bus calm down phase.<br>It denotes the time [ms] how long the NM shall stay in the Prepare Bus-Sleep Mode before transition into Bus-Sleep Mode shall take place. | | |
| ***Type*** | Integer | | |
| ***Unit*** | [ms] | | |
| ***Range*** | 1..65535 | | |
| ***Configuration Class*** | ***Pre-compile*** | x | Variant 1 |
| | ***Link time*** | x | Variant 2, 3 |
| | ***Post Build*** | -- | |
| ***Scope*** | Instance | | |

| | |
|---|---|
| ***Dependency*** | It shall be equal for all nodes in the cluster.<br>It shall be long enough to make all Tx-buffer empty.<br>The value 0 denotes that no Prepare Bus-Sleep State is configured.<br>It means that Prepare Bus-Sleep State is transient what implicates that it is left immediately after entrance and in result if NM-Timeout Timer expires the Bus-Sleep Mode is entered. |

| | | | |
|---|---|---|---|
| ***Name*** | **CNM_REPEAT_MESSAGE_TIME** | | |
| ***Description*** | Timeout for Repeat Message State.<br>It defines the time [ms] how long the NM shall stay in the Repeat Message State. | | |
| ***Type*** | Integer | | |
| ***Unit*** | [ms] | | |
| ***Range*** | 0..65535 | | |
| ***Configuration Class*** | ***Pre-compile*** | x | Variant 1 |
| | ***Link time*** | x | Variant 2, 3 |
| | ***Post Build*** | -- | |
| ***Scope*** | Instance | | |
| ***Dependency*** | Typically it should be equal to: n × CANNM_ MSG_CYCLE_TIME, where n denotes the number of NM PDUs that are normally sent in the Repeat Message State.<br>The value of n decremented by one determines the amount of lost NM PDUs that can be tolerated by the node detection procedure.<br>The value 0 denotes that no Repeat Message State is configured.<br>It means that Repeat Message State is transient what implicates that it is left immediately after entrance and in result no start-up stability is guaranteed and no node detection procedure is possible. | | |

| | | | |
|---|---|---|---|
| ***Name*** | **CNM_REMOTE_SLEEP_IND_TIME** | | |
| ***Description*** | Timeout for Remote Sleep Indication.<br>It defines the time [ms] how long it shall take to recognize that all other nodes are ready to sleep. | | |
| ***Type*** | Integer | | |
| ***Unit*** | [ms] | | |
| ***Range*** | 1..65535 | | |
| ***Configuration Class*** | ***Pre-compile*** | x | Variant 1 |
| | ***Link time*** | x | Variant 2, 3 |
| | ***Post Build*** | -- | |
| ***Scope*** | Instance | | |
| ***Dependency*** | Typically it should be equal to: n × CANNM_ MSG_CYCLE_TIME, where n denotes the number of NM PDUs that are normally sent before Remote Sleep Indication is detected.<br>The value of n decremented by one determines the amount of lost NM PDUs that can be tolerated by the Remote Sleep Indication procedure.<br>The value 0 denotes that no Remote Sleep Indication functionality is configured. | | |

| ***Included Containers*** | | |
|---|---|---|
| ***Container Name*** | ***Multiplicity*** | ***Scope / Dependency*** |
| None | | |

## 10.3 Published parameters

Published information contains data defined by the implementer of the SW module that does not change when the module is adapted (i.e. configured) to the actual HW/SW environment. It thus contains version and manufacturer information.

CNM139: : The following table specifies configuration parameters that shall be published in the module's header file.

| ***SWS Item*** | CNM139 | |
|---|---|---|
| ***Information elements*** | | |
| ***Information element name*** | ***Type / Range*** | ***Information element description*** |
| `CNM_VENDOR_ID` | `#define / uint16` | Vendor ID of the dedicated implementation of this module according to the AUTOSAR vendor list |
| `CNM_MODULE_ID` | `#define / 0x1E` | Module ID of this module from Module List |
| `CNM_AR_MAJOR_VERSION` | `#define / uint8` | Major version number of AUTOSAR specification on which the appropriate implementation is based on. |
| `CNM_AR_MINOR_VERSION` | `#define / uint8` | Minor version number of AUTOSAR specification on which the appropriate implementation is based on. |
| `CNM_AR_PATCH_VERSION` | `#define / uint8` | Patch level version number of AUTOSAR specification on which the appropriate implementation is based on. |
| `CNM_SW_MAJOR_VERSION` | `#define / uint8` | Major version number of the vendor specific implementation of the module. The numbering is vendor specific. |
| `CNM_SW_MINOR_VERSION` | `#define / uint8` | Minor version number of the vendor specific implementation of the module. The numbering is vendor specific. |
| `CNM_SW_PATCH_VERSION` | `#define / uint8` | Patch level version number of the vendor specific implementation of the module. The numbering is vendor specific. |

## 10.4 Examples

### 10.4.1 Example timing behavior for NM PDUs

Assume an example network of three nodes 1, 2, 3 (Figure 10-1). Nodes specific cycle offsets are equal respectively to t1 < t2 < t3 < T. NM cycle time is equal to T (Figure 10-2).
NM PDUs sent on the bus within the Repeat Message State are presented in the Figure 10-3, and within the Normal Operation / Ready Sleep State in Figure 10-4. Each dot in Figure 10-4denotes restart of the NM-Timeout Timer.

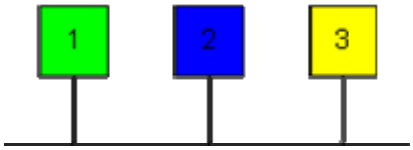


**Figure 10-1**

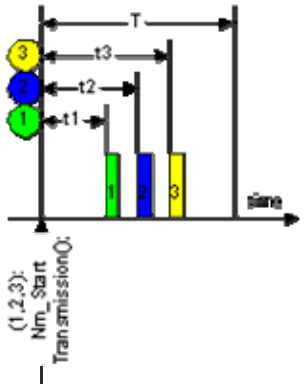


**Figure 10-2**

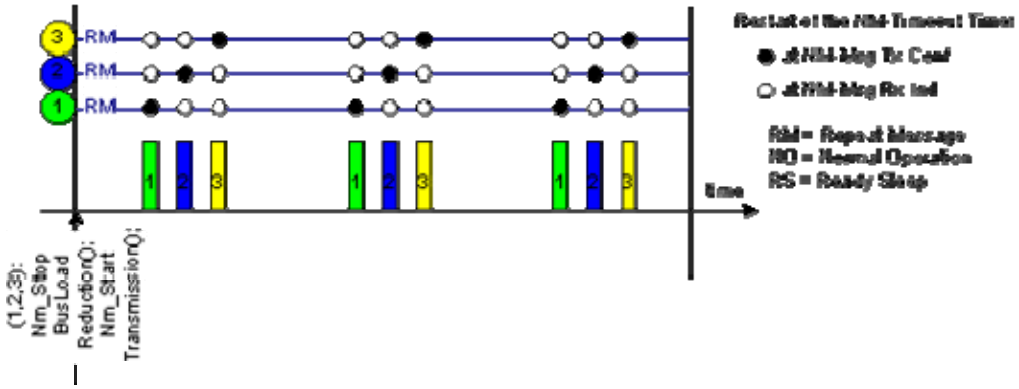


**Figure 10-3**

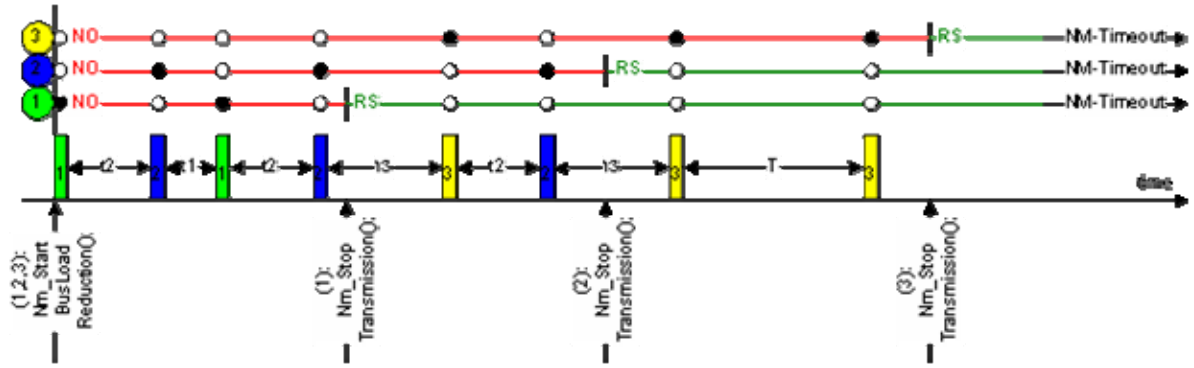


**Figure 10-4**

# 11 Changes to Release 1

## 11.1 Deleted SWS Items

| *SWS Item* | *Rationale* |
|---|---|
| NM003 | CAN Generic NM is no more applicable to multiple bus systems |
| NM007 | #8965 |
| NM033 | #14310 |
| NM036 | #13302 |
| NM037 | #8965 |
| NM041 | Requirement obsolete with respect to the NM functionality |
| NM043 | Detection of Present Nodes removed |
| NM044 | Detection of Present Nodes removed |
| NM046 | Detection of Present Nodes removed |
| NM053 | #14310 |
| NM059 | Requirement obsolete with respect to the NM functionality |
| NM083 | #8965 |
| NM084 | #8965 |
| NM098 | #8965 |
| NM106 | #8965 |
| NM108 | #13307 |
| NM110 | #14313 |
| NM111 | #13307 |
| NM115 | #8965 |
| NM116 | Requirement obsolete with respect to the NM functionality |
| NM117 | #8965 |
| NM118 | Detection of Present Nodes removed |
| NM120 | Detection of Present Nodes removed |
| NM121 | Detection of Present Nodes removed |
| NM122 | Detection of Present Nodes removed |
| NM123 | #13307 |
| NM124 | Detection of Present Nodes removed |
| NM126 | Detection of Present Nodes removed |
| NM127 | Detection of Present Nodes removed |
| NM130 | #13307 |
| NM136 | #13307 |
| NM137 | #13307 |
| NM140 | Detection of Present Nodes removed |
| NM142 | #13307 |
| NM143 | #13307 |
| NM144 | #13307 |
| NM147 | Configuration parameter for Control Bit Vector removed |
| NM148 | #13307 |
| NM149 | Write access to Control Bit Vector removed |
| NM150 | Read access to Control Bit Vector removed |
| NM151 | Control Bit Vector Indication removed |
| NM152 | Control Bit Vector Indication removed |
| NM154 | BusNm does no longer exist |
| NM155 | BusNm does no longer exist |
| NM161 | Removed according to review sheet |
| NM162 | #13307 |
| NM163 | Detection of Present Nodes removed |
| NM164 | Detection of Present Nodes removed |
| NM165 | Detection of Present Nodes removed |
| NM166 | Detection of Present Nodes removed |
| NM167 | Detection of Present Nodes removed |

| | |
|---|---|
| NM170 | #14310 |
| NM174 | #14310 |
| NM177 | Removed according to face to face meeting decision |
| NM178 | #14310 |
| NM180 | Removed according to face to face meeting decision |
| NM181 | Removed according to face to face meeting decision |
| NM188 | #6770 |
| NM190 | #14155 |
| NM196 | #8965 |
| NM197 | Requirement obsolete with respect to the new release |

## 11.2 Replaced SWS Items

| *SWS Item of Release 1* | *replaced by SWS Item* | *Rationale* |
|---|---|---|
| NM001 | CNM001 | Change of module prefix |
| NM002 | CNM002 | Change of module prefix |
| NM004 | CNM003 | Change of module prefix |
| NM005 | CNM004 | Change of module prefix |
| NM006 | CNM005 | Change of module prefix |
| NM008 | CNM018 | Change of module prefix |
| NM009 | CNM028 | Change of module prefix |
| NM010 | CNM024 | Change of module prefix |
| NM013 | CNM029 | Change of module prefix |
| NM014 | CNM030 | Change of module prefix |
| NM016 | CNM034 | Change of module prefix |
| NM018 | CNM038 | Change of module prefix |
| NM019 | CNM041 | Change of module prefix |
| NM022 | CNM060 | Change of module prefix |
| NM023 | CNM042 | Change of module prefix |
| NM024 | CNM010 | Change of module prefix |
| NM034 | CNM057 | Change of module prefix |
| NM035 | CNM062 | Change of module prefix |
| NM039 | CNM064 | Change of module prefix |
| NM040 | CNM069 | Change of module prefix |
| NM042 | CNM074 | Change of module prefix |
| NM047 | CNM072 | Change of module prefix |
| NM072 | CNM100 | Change of module prefix |
| NM073 | CNM097 | Change of module prefix |
| NM074 | CNM099 | Change of module prefix |
| NM078 | CNM131 | Change of module prefix |
| NM079 | CNM137 | Change of module prefix |
| NM080 | CNM139 | Change of module prefix |
| NM081 | CNM143 | Change of module prefix |
| NM082 | CNM136 | Change of module prefix |
| NM085 | CNM015 | Change of module prefix |
| NM086 | CNM013 | Change of module prefix |
| NM088 | CNM021 | Change of module prefix |
| NM089 | CNM022 | Change of module prefix |
| NM090 | CNM023 | Change of module prefix |
| NM091 | CNM084 | Change of module prefix |
| NM092 | CNM027 | Change of module prefix |
| NM093 | CNM014 | Change of module prefix |
| NM094 | CNM016 | Change of module prefix |
| NM095 | CNM017 | Change of module prefix |
| NM096 | CNM031 | Change of module prefix |

| | | |
|---|---|---|
| NM097 | CNM032 | Change of module prefix |
| NM097 | CNM083 | Change of module prefix |
| NM099 | CNM033 | Change of module prefix |
| NM100 | CNM037 | Change of module prefix |
| NM102 | CNM039 | Change of module prefix |
| NM105 | CNM040 | Change of module prefix |
| NM107 | CNM043 | Change of module prefix |
| NM113 | CNM058 | Change of module prefix |
| NM114 | CNM059 | Change of module prefix |
| NM115 | CNM061 | Change of module prefix |
| NM117 | CNM075 | Change of module prefix |
| NM128 | CNM080 | Change of module prefix |
| NM129 | CNM081 | Change of module prefix |
| NM075 | CNM130 | Change of module prefix |
| NM131 | CNM132 | Change of module prefix |
| NM132 | CNM133 | Change of module prefix |
| NM133 | CNM134 | Change of module prefix |
| NM134 | CNM049 | Change of module prefix |
| NM138 | CNM011 | Change of module prefix |
| NM139 | CNM012 | Change of module prefix |
| NM141 | CNM138 | Change of module prefix |
| NM145 | CNM073 | Change of module prefix |
| NM146 | CNM079 | Change of module prefix |
| NM153 | CNM089 | Change of module prefix |
| NM168 | CNM090 | Change of module prefix |
| NM171 | CNM088 | Change of module prefix |
| NM172 | CNM070 | Change of module prefix |
| NM173 | CNM077 | Change of module prefix |
| NM175 | CNM036 | Change of module prefix |
| NM176 | CNM019 | Change of module prefix |
| NM179 | CNM025 | Change of module prefix |
| NM182 | CNM035 | Change of module prefix |
| NM183 | CNM044 | Change of module prefix |
| NM184 | CNM045 | Change of module prefix |
| NM185 | CNM046 | Change of module prefix |
| NM186 | CNM047 | Change of module prefix |
| NM187 | CNM050 | Change of module prefix |
| NM189 | CNM068 | Change of module prefix |
| NM191 | CNM048 | Change of module prefix |
| NM192 | CNM082 | Change of module prefix |
| NM193 | CNM087 | Change of module prefix |
| NM194 | CNM095 | Change of module prefix |
| NM195 | CNM103 | Change of module prefix |
| NM196 | CNM104 | Change of module prefix |
| NM198 | CNM007 | Change of module prefix |
| NM199 | CNM008 | Change of module prefix |
| NM200 | CNM101 | Change of module prefix |
| NM201 | CNM102 | Change of module prefix |
| NM202 | CNM076 | Change of module prefix |
| NM203 | CNM078 | Change of module prefix |
| NM214 | CNM006 | Change of module prefix |
| NM215 | CNM096 | Change of module prefix |
| NM216 | CNM098 | Change of module prefix |
| NM217 | CNM065 | Change of module prefix |
| NM221 | CNM055 | Change of module prefix |
| NM222 | CNM056 | Change of module prefix |
| NM223 | CNM135 | Change of module prefix |
| NM224 | CNM009 | Change of module prefix |

| | | |
|---|---|---|
| NM240 | CNM051 | Change of module prefix |
| NM241 | CNM085 | Change of module prefix |

## 11.3 Changed SWS Items

| *SWS Item* | *Rationale* |
|---|---|
| CNM001 | #14312 |
| CNM002 | Update of module prefix<br>#14312 |
| CNM003 | Editorial change |
| CNM004 | Editorial change |
| CNM005 | Editorial change |
| CNM010 | Condition for data consistency added. Update of module prefix (#132919).<br>Update of CNm_GetState (#10297).<br>#14190 |
| CNM011 | Editorial change |
| CNM013 | Editorial change |
| CNM015 | Update of module prefix (#13291) |
| CNM016 | #14312 |
| CNM017 | #14312 |
| CNM018 | Note deleted since FlexRay is not supported anymore<br>#14312 |
| CNM021 | Update of module prefix (#13291) |
| CNM022 | #6770 |
| CNM023 | #6770 |
| CNM024 | #14322, #14312 |
| CNM028 | Description refined<br>#13539 |
| CNM029 | #14322, #14312 |
| CNM031 | #6770 |
| CNM032 | Description refined |
| CNM033 | Update of module prefix (#13291) |
| CNM034 | Update of module prefix (#13291) |
| CNM035 | Condition “as fast as possible” removed since not testable<br>BusNm_ replaced by CanNm_ and configuration parameter added<br>#6770, #14153 |
| CNM036 | #6770 |
| CNM037 | #14312 |
| CNM038 | Editorial change (#13323). Update of module prefix (#13291)<br>#14191 |
| CNM039 | Condition for avoiding unexpected notification added.<br>Update of module prefix (#13291). |
| CNM040 | Update of module prefix (#13291).<br>#14312 |
| CNM041 | #8956, #6770 |
| CNM042 | Update of module prefix (#13291). #6770. #14322. |
| CNM043 | Update of module prefix (#13291). #6770. #14322. |
| CNM044 | BusNm_ replaced by CanNm_. Service is optional.<br>#14192, #14312 |
| CNM045 | BusNm_ replaced by CanNm_. Service is optional.<br>#14193, #14312 |
| CNM046 | BusNm_ replaced by CanNm_. Service is optional.<br>#14194, #14312, #16619 |
| CNM047 | NM message data updated<br>#14312 |
| CNM048 | Service marked as optional. Update of module prefix (#13291). |

| | |
|---|---|
| | #13220, #14312 |
| CNM049 | Service marked as optional. Update of module prefix (#13291). |
| CNM050 | NM message data updated. As optional marked.<br>#14195 |
| CNM057 | Update of module prefix (#13291). |
| CNM058 | #13301 |
| CNM059 | BusNm_ replaced by CanNm_. #13301 |
| CNM062 | Editorial change |
| CNM064 | Update of module prefix (#13291). |
| CNM068 | Update of module prefix (#13291) and API (#6770). |
| CNM069 | Editorial change |
| CNM070 | NM_NODE_DETECTION_LOCK_TIME removed.<br>Update of module prefix (#13291). |
| CNM072 | Configuration parameter NM_MAX_NUMBER_OF_NODES deleted.<br>Editorial change.<br>#14316 |
| CNM074 | Update of module prefix (#13291). #14312 |
| CNM075 | Update of module prefix (#13291). #14312 |
| CNM077 | Update of module prefix (#13291). |
| CNM079 | Editorial change. Update of module prefix (#13291). |
| CNM080 | Editorial change. Update of module prefix (#13291). #14312 |
| CNM081 | Editorial change. Update of module prefix (#13291). #13220. #14312 |
| CNM082 | Update of module prefix (#13291). |
| CNM083 | Editorial change |
| CNM084 | Description refined and bus specifics removed |
| CNM087 | Control Bit Vector support removed. Editorial change. |
| CNM088 | Update of module prefix (#13291). |
| CNM089 | Update of module prefix (#13291). #14312 |
| CNM090 | Editorial change<br>#14158 |
| CNM095 | Update of module prefix (#13291). |
| CNM100 | Types of error list extended |
| CNM130 | Editorial change. Update of module prefix (#13291). |
| CNM131 | Module prefix updated 8(#13291).<br>#14166 |
| CNM137 | SWS template adaptation – configuration specification |
| CNM138 | SWS template adaptation – configuration specification |
| CNM139 | Type added<br>#14301 |
| CNM143 | SWS template adaptation – configuration specification |

## 11.4 Added SWS Items

| *SWS Item* | *Rationale* |
|---|---|
| CNM006 | SWS template adaptation - File structure added |
| CNM007 | SWS template adaptation - File structure added |
| CNM008 | SWS template adaptation - File structure added |
| CNM009 | SWS template adaptation - File structure added |
| CNM051 | Configuration of NM message length<br>#14312 |
| CNM055 | Initialization refined |
| CNM056 | Initialization refined |
| CNM065 | Selectable post-built time configuration defined |
| CNM076 | Exception handling for Remote Sleep Indication added<br>Note has been removed (#13566). |
| CNM078 | Exception handling for Remote Sleep Indication added |

| | |
|---|---|
| CNM085 | Configuration parameter for Passive Mode |
| CNM096 | Error classification refined |
| CNM098 | Error classification refined<br>BusNm_ replaced by CanNm_ |
| CNM101 | Error handling refined |
| CNM102 | Error reporting to the DET refined<br>Invalid channel handle value shall be passed to DET |
| CNM111 | CNm_GetPduData added (#13449) |
| CNM135 | Version check added<br>#14167 |
| CNM141 | Configuration parameter added |
| CNM142 | Passive Mode added |
| CNM144 | State transition from Bus-Sleep Mode to Network Mode |
| CNM146 | Write access to Repeat Message Request Bit added (#8948, #13333) |
| CNM147 | Configuration parameter for Repeat Message Request Bit added (#8948, #13333)<br>#14152 |
| CNM148 | CNm_GetPduData added (#13449)<br>#14152 |
| CNM149 | #13449<br>#14199 |
| CNM150 | #13449 |
| CNM151 | State change notification added |
| CNM152 | State change notification added |
| CNM153 | #14152, #13539 |
| CNM154 | #14152 |
| CNM155 | #14152, #13539 |
| CNM156 | #14152 |
| CNM157 | #14152 |
| CNM158 | #14152 |
| CNM159 | #13539 |
| CNM160 | #13539 |
| CNM161 | #13539 |
| CNM162 | #13539 |
| CNM163 | #13539 |
| CNM164 | #13539 |
| CNM165 | #13539 |
| CNM166 | #13539 |
| CNM167 | #13539 |
| CNM168 | #13539 |
| CNM169 | #13539 |
| CNM170 | #13539 |
| CNM171 | #13539 |
| CNM172 | #13539 |
| CNM173 | #13539 |
| CNM174 | #13539 |
| CNM175 | #14322 |