Document Title	Specification of CAN Network Management
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	013
Document Classification	Standard

Document Version	3.2.0
Document Status	Final
Part of Release	3.1
Revision	5

Document Change History

Date	Version	Changed by	Change Description
15.09.2010	3.2.0	AUTOSAR Administration	• Harmonization of CanNm_RxIndication signature • Legal disclaimer revised
28.01.2010	3.1.0	AUTOSAR Administration	• Improved configuration and interoperation of CanNm and CanIf • Legal disclaimer revised
23.06.2008	3.0.1	AUTOSAR Administration	• Legal disclaimer revised
18.12.2007	3.0.0	AUTOSAR Administration	• Merge CAN NM and Generic NM • Document meta information extended • Small layout adaptations made
31.01.07	2.0.0	AUTOSAR Administration	• Post build and link-time configuration variant introduced • Configurable NMPDU format introduced • Passive mode introduced • Legal disclaimer revised • Release Notes added • “Advice for users” revised • “Revision Information” added
30.06.2005	1.0.0	AUTOSAR Administration	Initial Release

Disclaimer

Advice for users

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 Network Management (CanNm).

The AUTOSAR CAN Network Management is a hardware independent protocol that can only be used on CAN (for limitations refer to chapter 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.

The CAN Network Management (CanNm) function provides an adaptation between Network Management Interface (Nm) and CAN Interface (CanIf) module. For a general understanding of the AUTOSAR Network Management functionality please refer to [9].

2 Acronyms and abbreviations

Acronym:	*Description:*
API	Application Programming Interface
NM	Network Management
"PDU transmission ability is disabled"	This means that the NM message transmission has been disabled by the optional service CanNm_DisableCommunication.

Abbreviation:	*Description:*
CanNm	Abbreviation for CAN Network Management
PDU	Protocol Data Unit
CanIf	Abbreviation for the CAN Interface
BSW	Basic Software
SDU	Service Data Unit
DET	Development Error Tracer
DEM	Diagnostic Event Manager

3 Related documentation

3.1 Input documents

[1] AUTOSAR Layered Software Architecture
AUTOSAR_LayeredSoftwareArchitecture.pdf

[2] AUTOSAR General Requirements on Basic Software Modules
AUTOSAR_SRS_General.pdf

[3] AUTOSAR Requirements on Network Management
AUTOSAR_SRS_NM.pdf

[4] AUTOSAR Specification of CAN Interface
AUTOSAR_SWS_CAN_Interface.pdf

[5] AUTOSAR Specification of FlexRay Network Management
AUTOSAR_SWS_FlexRay_NM.pdf

[6] AUTOSAR Specification of Communication Stack Types
AUTOSAR_SWS_ComStackTypes.pdf

[7] AUTOSAR Specification of ECU Configuration
AUTOSAR_ECU_Configuration.pdf

[8] AUTOSAR Specification of BSW Scheduler,
AUTOSAR_SWS_BSW_Scheduler.pdf

[9] AUTOSAR Specification of Generic Network Management Interface
AUTOSAR_SWS_NMInterface.pdf

[10] Specification of Communication Manager
AUTOSAR_SWS_ComManager.pdf

[11] Specification of ECU State Manager
AUTOSAR_SWS_ECU_StateManager.pdf

[12] Specification of Operating System
AUTOSAR_SWS_OS.pdf

[13] Specification of Diagnostics Event Manager
AUTOSAR_SWS_DEM.pdf

[14] Specification of Development Error Tracer
AUTOSAR_SWS_DET.pdf

[15] Specification of Standard Types
AUTOSAR_SWS_StandardTypes.pdf

[16] Specification of Platform Types
AUTOSAR_SWS_PlatformTypes.pdf

[17] Specification of Compiler Abstraction
AUTOSAR_SWS_CompilerAbstraction.pdf

[18] AUTOSAR Basic Software Module Description Template,
AUTOSAR_BSW_Module_Description.pdf

3.2 Related standards and norms

Not available.

4 Constraints and assumptions

4.1 Limitations

1. One instance of CanNm is associated with only one NM-Cluster in one network. One NM-Cluster can have only one instance of CanNm in one node.
2. One instance of CanNm is associated with only one network within the same ECU.
3. CanNm is only applicable for CAN systems.

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

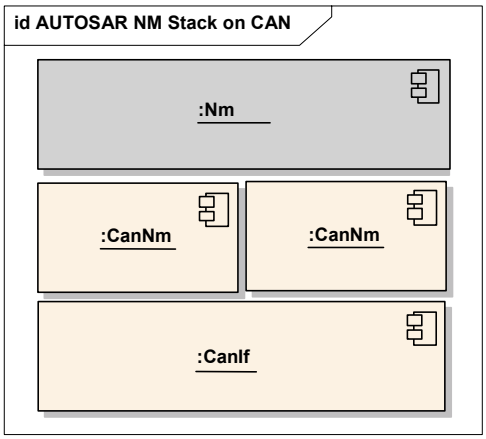


Figure 4-1

This module is only applicable for CAN systems.

CANNM131: The AUTOSAR CAN NM algorithm shall support up to 64 nodes per NM-Cluster.

Note: The AUTOSAR CAN 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 NM coordination algorithm.

4.2 Applicability to car domains

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

5 Dependencies to other modules

CAN Network Management (CanNm) uses services of CAN Interface (CanIf) and provides services to the Generic Network Management Interface (Nm).

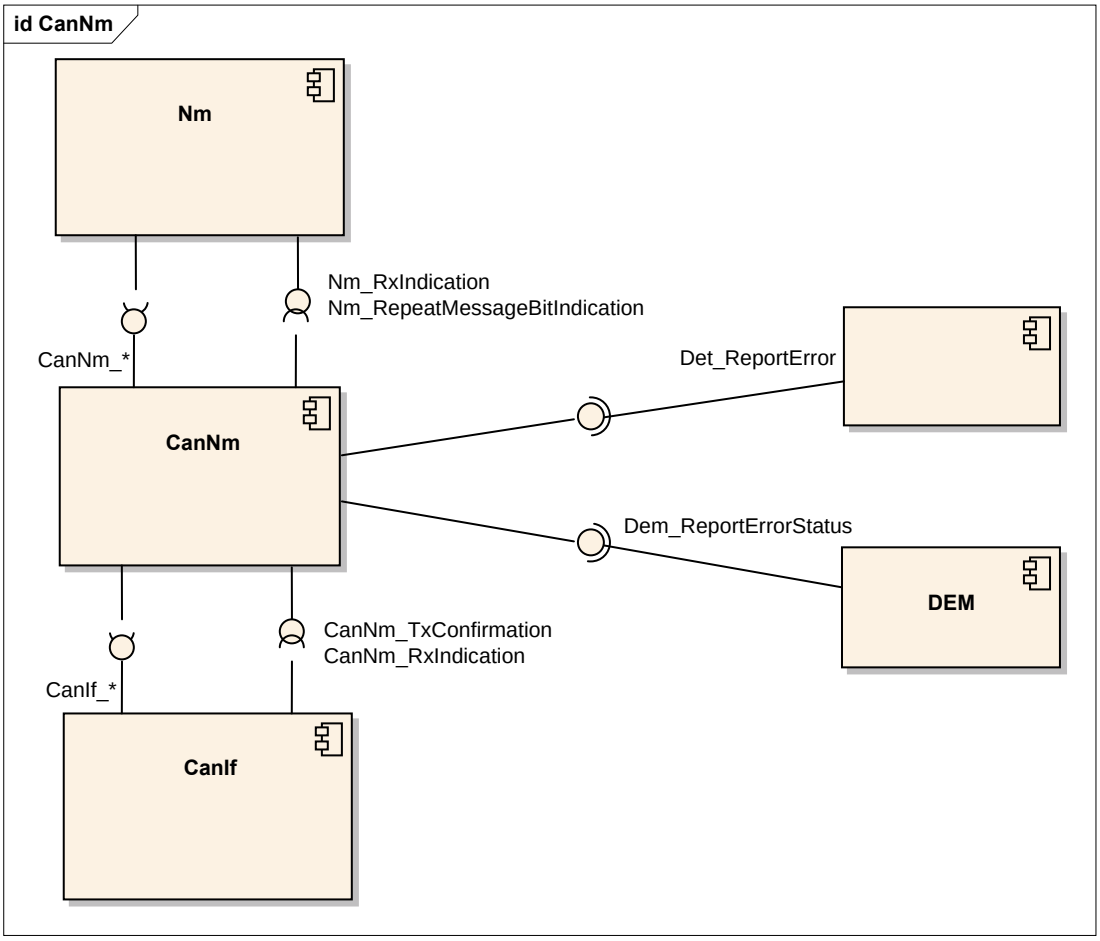


Figure 5-1 Dependencies to other modules

5.1 File Structure

5.1.1 Code File Structure

CANNM081: The following C-files shall be provided by the CanNm module.

- `CanNm.c` (for implementation of provided functionality)
- `CanNm_Cfg.c` (for pre-compile time configurable parameters)
- `CanNm_Lcfg.c` (for link time configurable parameters)
- `CanNm_PBcfg.c` (for post build time configurable parameters)

5.1.2 Header File Structure

CANNM044: The following H-files shall be provided by the CanNm module.

- `CanNm.h` (for declaration of provided interface functions)
- `CanNm_Cbk.h` (for declaration of provided call-back functions)
- `CanNm_Cfg.h` (for pre-compile time configurable parameters)

CANNM082: The following H-files shall be included by the CanNm module.

- `ComStack_Types.h`
  Note: The following header files are indirectly included by `ComStack_Types.h`
  - `Std_Types.h` (for AUTOSAR standard types)
  - `Platform_Types.h` (for platform specific types)
  - `Compiler.h` (for compiler specific language extensions)
- `CanNm.h` (for declaration of provided interface functions)
- `Nm_Cbk.h` (for CAN NM specific callbacks of Generic Generic Network Management Interface)
- `Det.h` (for interface of DET – optional, included only if DET is configured)
- `Dem.h` (for interface of DEM)
- `NmStack_Types.h` (for common NM types)
- `SchM_CanNm.h` (for services of the Basic Software Scheduler)
- `MemMap.h` (for Memory Mapping)

CANNM083: The following header files containing configuration data shall be included within the CanNm module.

- `CanIf_Cfg.h` (for the CAN PDU Ids)
- `Nm_Cfg.h` (for the derived configuration items from Nm)

6 Requirements traceability

Document: AUTOSAR General Requirements on Basic Software Modules [2]

Requirement	***Satisfied by***
[BSW00344] Reference to link-time configuration	Ok; see chapter 10.2
[BSW00404] Reference to post build time configuration	Ok; see Chapter 10.2
[BSW00405] Reference to multiple configuration sets	Ok; see Chapter 10.2
[BSW00345] Pre-compile-time configuration	Ok; see CANNM044
[BSW159] Tool-based configuration	Ok; see Chapter 10.2
[BSW167] Static configuration checking	Ok; see Chapter 10.2
[BSW170] Data for reconfiguration of AUTOSAR SW-Components	N/a (CanNm is no SW-C)
[BSW171] Configurability of optional functionality	Ok; see Chapter 10.2
[BSW00380] Separate C-Files for configuration parameters	Ok; see CANNM044
[BSW00419] Separate C-Files for pre-compile time configuration parameters	Ok; see CANNM081
[BSW00381] Separate configuration header file for pre-compile time parameters	Ok; see CANNM044
[BSW00412] Separate H-File for configuration parameters	Ok; see CANNM044
[BSW00383] List dependencies of configuration files	Ok; see CANNM083
[BSW00384] List dependencies to other modules	Ok; see Figure 5-1
[BSW00387] Specify the configuration class of callback function	n/a (Callback functions are not configurable)
[BSW00388] Introduce containers	Ok; see Chapter 10.2
[BSW00389] Containers shall have names	Ok; see Chapter 10.2
[BSW00390] Parameter content shall be unique within the module	Ok; see Chapter 10.2
[BSW00391] Parameter shall have unique names	Ok; see Chapter 10.2
[BSW00392] Parameters shall have a type	Ok; see Chapter 10.2
[BSW00393] Parameters shall have a range	Ok; see Chapter 10.2
[BSW00394] Specify the scope of the parameters	Ok; see Chapter 10.2
[BSW00395] List the required parameters (per parameter)	Ok; see Chapter 10.2
[BSW00396] Configuration classes	Ok; see Chapter 10.2
[BSW00397] Pre-compile-time parameters	Ok; see Chapter 10.2
[BSW00398] Link-time parameters	Ok; see Chapter 10.2
[BSW00399] Loadable Post-build time parameters	Ok; see Chapter 10.2
[BSW00400] Selectable Post-build time parameters	Ok; see Chapter 10.2
[BSW00402] Published information	Ok; see Chapter 10.4
[BSW00375] Notification of wake-up reason	n/a (CanNm does not wake-up an ECU)
[BSW101] Initialization interface	Ok; see chapter 8.3.1
[BSW00416] Sequence of Initialization	n/a (sequence is defined by ComM)
[BSW00406] Check module initialization	Ok; see chapter 8
[BSW168] Diagnostic Interface of SW components	n/a (diagnostics for CanNm not required)
[BSW00407] Function to read out published parameters	Ok; see chapter 10.4
[BSW00423] Usage of SW-C template to describe BSW modules with AUTOSAR Interfaces	n/a (CanNm has no interface to the RTE)
[BSW00424] BSW main processing function task allocation	n/a (CanNm scheduled function is called by the BSW scheduler)
[BSW00425] Trigger conditions for schedulable objects	n/a

	(implementation specific)
[BSW00426] Exclusive areas in BSW modules	n/a (implementation specific)
[BSW00427] ISR description for BSW modules	n/a (implementation specific)
[BSW00428] Execution order dependencies of main processing functions	Ok; see chapter 7.13
[BSW00429] Restricted BSW OS functionality access	n/a (none of these services are used by the CanNm)
[BSW00431] The BSW Scheduler module implements task bodies	Ok; see chapter 7.13
[BSW00432] Modules should have separate main processing functions for read/receive and write/transmit data path	n/a (transmission and reception is handled in CanNm_MainFunction)
[BSW00433] Calling of main processing functions	Ok; see chapter 7.13
[BSW00434] The Schedule Module shall provide an API for exclusive areas	n/a (implementation specific)
[BSW00336] Shutdown interface	n/a (no shutdown interface needed)
[BSW00337] Classification of errors	Ok; see chapter 7.12
[BSW00338] Detection and Reporting of development errors	Ok; see chapter 7.12 and 10.2
[BSW00369] Do not return development error codes via API	Ok; see chapter 8
[BSW00339] Reporting of production relevant error status	Ok; see chapter 7.12
[BSW00417] Reporting of Error Events by Non-Basic Software	n/a (CanNm is no SW-C)
[BSW00323] API parameter checking	Ok; see CANNM016
[BSW004] Version check	Ok;
[BSW00409] Header files for production code error IDs	Ok; see CANNM082
[BSW00385] List possible error notifications	Ok; see 7.12
[BSW00386] Configuration for detecting an error	Ok: `CANNM_DEV_ERROR_DETECT`
[BSW161] Microcontroller abstraction	n/a (CanNm microcontroller independent)
[BSW162] ECU layout abstraction	n/a (CanNm is ECU hardware independent)
[BSW005] No hard coded horizontal interfaces within MCAL	n/a (CanNm is not part of the MCAL)
[BSW00415] User dependent include files	n/a (not flexible with respect to future extensions)
[BSW164] Implementation of interrupt service routines	n/a (no ISR provided)
[BSW00325] Runtime of interrupt service routines	n/a (no ISR provided)
[BSW00326] Transition from ISRs to OS tasks	n/a (no ISR provided)
[BSW00342] Usage of source code and object code	Ok; see chapter 10.2.1
[BSW00343] Specification and configuration of time	Ok; see chapter 10.2
[BSW160] Human-readable configuration data	n/a (implementation specific)
[BSW007] HIS MISRA C	Ok; all implementation related information
[BSW00300] Module naming convention	Ok; CanNm prefix is used
[BSW00413] Accessing instances of BSW modules	n/a (implementation specific)
[BSW00347] Naming separation of different instances of BSW drivers	n/a (implementation specific)

[BSW00305] Self-defined data types naming convention	n/a (no self-defined data types used)
[BSW00307] Global variables naming convention	n/a (no global variables specified)
[BSW00310] API naming convention	Ok; see chapter 8
[BSW00373] Main processing function naming convention	Ok; see chapter 8.5.1
[BSW00327] Error values naming convention	Ok; see chapter 7.12
[BSW00335] Status values naming convention	n/a (no status values exported)
[BSW00350] Development error detection keyword	Ok; see chapter 8.7
[BSW00408] Configuration parameter naming convention	Ok; see chapter 10.2
[BSW00410] Compiler switches shall have defined values	Ok; see CANNM084
[BSW00411] Get version info keyword	Ok; see chapter 8.3.14
[BSW00346] Basic set of module files	Ok; see CANNM081 and CANNM044
[BSW158] Separation of configuration from implementation	Ok; see CANNM081 and CANNM044
[BSW00314] Separation of interrupt frames and service routines	n/a (CanNm doesn't have interrupt frame definitions)
[BSW00370] Separation of callback interface from API	Ok; see CANNM044
[BSW00348] Standard type header	Ok; see CANNM082
[BSW00353] Platform specific type header	Ok; see CANNM082
[BSW00361] Compiler specific language extension header	Ok; see CANNM082
[BSW00301] Limit imported information	Ok; see CANNM082 and CANNM083
[BSW00302] Limit exported information	Ok; see CANNM044 and chapter 8
[BSW00328] Avoid duplication of code	n/a (implementation specific)
[BSW00312] Shared code shall be reentrant	n/a (implementation specific)
[BSW006] Platform independency	n/a (CanNm is hardware independent)
[BSW00357] Standard API return type	Ok; see chapter 8
[BSW00377] Module specific API return types	n/a (CanNm doesn't define own types)
[BSW00304] AUTOSAR integer data types	Ok; see chapter 8
[BSW00355] Do not redefine AUTOSAR integer data types	Ok; see chapter 8
[BSW00378] AUTOSAR boolean type	Ok; see chapter 8 and 10.2
[BSW00306] Avoid direct use of compiler and platform specific keywords	n/a (implementation specific)
[BSW00308] Definition of global data	Ok; see CANNM081
[BSW00309] Global data with read-only constraint	n/a (implementation specific)
[BSW00371] Do not pass function pointers via API	Ok; see chapter 8
[BSW00358] Return type of init() functions	Ok; see chapter 8.3.1
[BSW00414] Parameter of init function	Ok; see chapter 8.3.1
[BSW00376] Return type and parameters of main processing functions	Ok; see chapter 8.5.1
[BSW00359] Return type of callback functions	Ok; see chapter 8.4
[BSW00360] Parameters of callback functions	Ok; see chapter 8.4
[BSW00329] Avoidance of generic interfaces	Ok; see chapter 8
[BSW00330] Usage of macros / inline functions instead of functions	n/a (implementation specific)
[BSW00331] Separation of error and status values	n/a (CanNm doesn't provide status information)
[BSW009] Module User Documentation	Ok; see whole document
[BSW00401] Documentation of multiple instances of	Ok; see chapter 10.2

configuration parameters	
[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 chapter 10.4
[BSW00379] Module identification	Ok; see chapter 10.4
[BSW003] Version identification	Ok; see chapter 10.4
[BSW00318] Format of module version numbers	Ok; see chapter 10.4
[BSW00321] Enumeration of module version numbers	n/a (implementation specific)
[BSW00341] Microcontroller compatibility documentation	n/a (CanNm is microcontroller independent)
[BSW00334] Provision of XML file	n/a (implementation specific)

Document: AUTOSAR Requirements on Basic Software, Module NM [3]

Requirement	***Satisfied by***
[BSW150] Configuration of functionality	Ok; see chapter 10.2
[BSW151] Integration into running NM cluster	n/a (The CAN bus satisfies already this requirement)
[BSW043] Bus Traffic without NM Initialization	n/a (ComM is responsible to initialize the communication components)
[BSW044] Applicability to different types of communication systems	Ok; see chapter 4.2
[BSW045] NM-cluster Independent Shutdown Coordination	Ok; see chapter 8
[BSW046] Trigger of startup of all Nodes at any Point in Time	n/a (not in the responsibility of CanNm)
[BSW047] Bus Keep Awake Services	Ok; see chapter 8.3.6
[BSW048] Bus Sleep Mode	n/a (not in the responsibility of CanNm)
[BSW050] NM State Information	n/a (CanNm has no states)
[BSW051] NM State Change Indication	n/a (CanNm has no states)
[BSW052] Notification that all other ECUs are ready to sleep	n/a (not in the responsibility of CanNm)
[BSW02509] Notification that at least one other node is not ready to sleep anymore	n/a (not in the responsibility of CanNm)
[BSW02503] Sending user data	Ok; see chapter 8.3.7
[BSW02504] Receiving user data	Ok; see chapter 8.3.8
[BSW153] Detection of present nodes	n/a (not in the responsibility of CanNm)
[BSW02508] Unambiguous node identification per bus	Ok; see chapter 8.3.10
[BSW02505] Sending node identifier	Ok; see chapter 7.10
[BSW02506] Receiving node identifier	Ok; see chapter 8.3.9
[BSW02511] Configurable Role in Cluster Shutdown	Ok; see 7.8.3
[BSW053] Deterministic Behavior in Case of Bus Unavailability	Ok; see chapter 7.9
[BSW137] Communication system error handling	Ok; see chapter 7.9
[BSW136] Coordination of coupled networks	n/a (not in the scope of CanNm)
[BSW140] Compliance with OSEK NM on a gateway	n/a

	(not in the scope of CanNm)
[BSW054] Deterministic Time for Bus Sleep	n/a (not in the scope of CanNm)
[BSW142] Limitation of NM bus load	Ok; see chapter 7.7
[BSW143] Predictable NM bus load	Ok; see chapter 7.6.1
[BSW144] ECU cluster size	n/a (CanNm hasn't any restriction concerning cluster size)
[BSW145] Robustness against NM message losses	n/a (not in the scope of CanNm)
[BSW146] Robustness against NM message jitter	n/a (not in the scope of CanNm)
[BSW147] Processor independent algorithm	Ok; see CANNM050
[BSW149] Configurable Timing	Ok; see chapter 10.2
[BSW154] Bus independency of API	n/a (CanNm has to be bus dependent)
[BSW148] Separation of Communication system dependent parts	Ok; see whole document
[BSW139] Compliance with OSEK NM on one cluster	n/a (not in the scope of CanNm)
[BSW02510] Immediate Transmission Confirmation	Ok; see chapter 8.4.1
[BSW02512] CommunicationControl (28 hex) service support	n/a (This feature is not supported currently; because of time constraints it is postponed to the next major release of AUTOSAR)

7 Functional specification

CANNM050: The described algorithms in this specification shall be processor independent.

7.1 Coordination algorithm

The AUTOSAR CAN 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 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 a 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 Software Specifcations of the Mode Management.

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

CANNM087: Every network node shall transmit periodic NM PDUs as long as it requires bus-communication; otherwise it shall transmit no NM PDUs.

CANNM088: If bus communication is released and there are no NM PDUs on the bus for a configurable amount of time determined by **`CANNM_TIMEOUT_TIME`** + **`CANNM_WAIT_BUS_SLEEP_TIME`** (both configuration parameters) transition into the Bus-Sleep Mode shall be performed.

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

CANNM089: The AUTOSAR CAN NM state machine shall contain states, transitions and triggers required for the AUTOSAR CAN NM coordination algorithm seen from point of view of one single node in the NM-cluster.

CANNM090: Transitions in the AUTOSAR CAN 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 NM state machine from point of view of one single node in the NM-cluster. All services

called by AUTOSAR CAN 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 NM state machine in sense of an UML state chart see API specification.

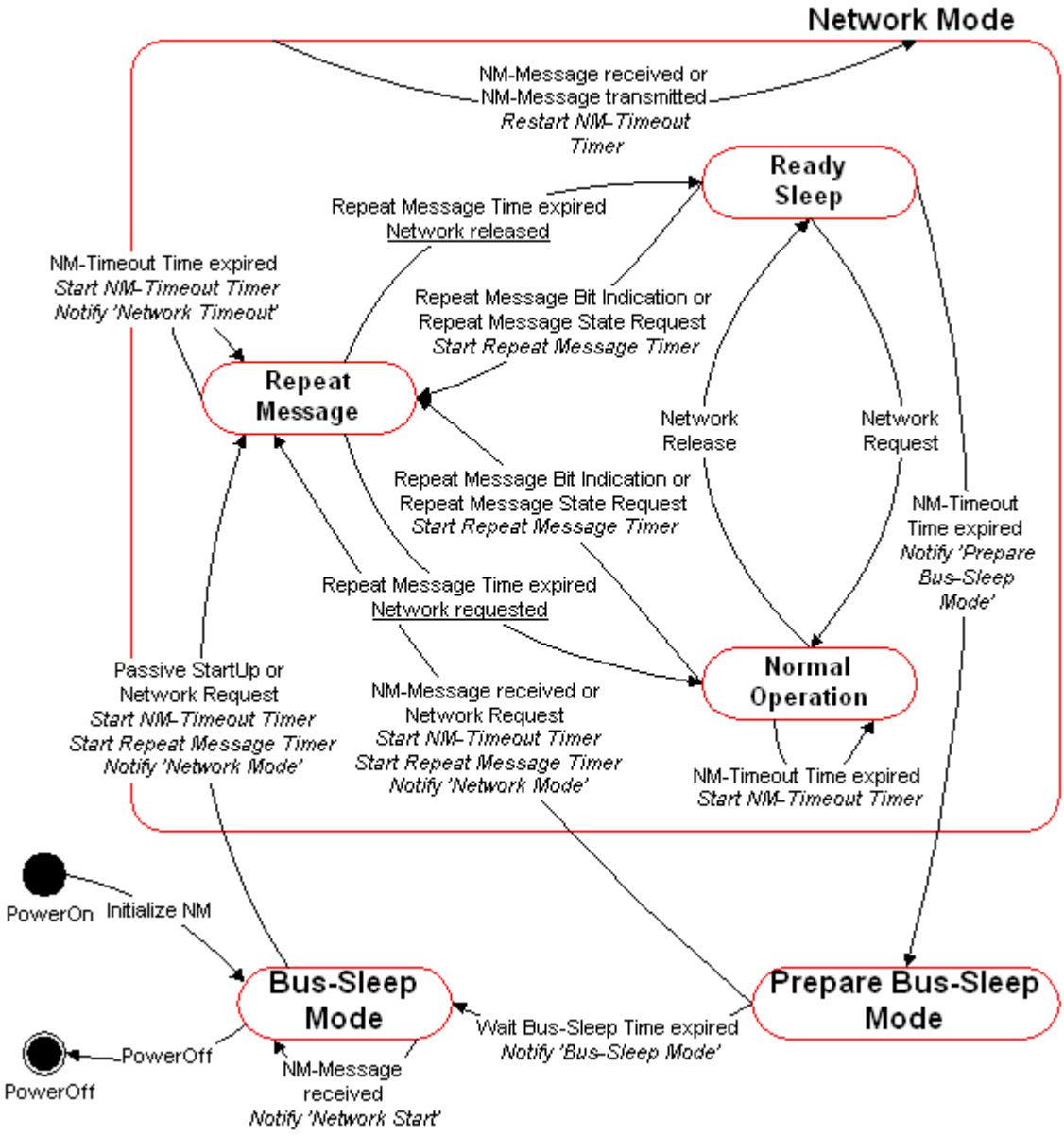


Figure 7-1

7.2 Operational Modes

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

CANNM091: The service call **`CanNm_GetState`** shall provide information about the current state and the current mode of the NM state machine.

CANNM092: The AUTOSAR CAN NM shall consist of three operational modes:
- Network Mode
- Prepare Bus-Sleep Mode
- Bus-Sleep Mode

CANNM093: Changes of the AUTOSAR CAN NM operational modes shall be notified to the upper layer by means of callback functions.

7.2.1 Network Mode

CANNM094: The Network Mode shall consist of three internal states:
- Repeat Message State
- Normal Operation State
- Ready Sleep State

CANNM095: When the Network Mode is entered from Bus-Sleep Mode or Prepare Bus-Sleep Mode, by default, the Repeat Message State shall be entered.

CANNM096: When the Network Mode is entered, the NM-Timeout Timer shall be started.

CANNM097: When the Network Mode is entered, the NM shall notify the upper layer by calling **`Nm_NetworkMode.`**

CANNM098: At successful reception of a NM PDU in the Network Mode, the NM-Timeout Timer shall be restarted.

CANNM099: At successful transmission of a NM PDU in the Network Mode, the NM-Timeout Timer shall be restarted.

CANNM206: The NM-Timeout Timer shall be reset every time it is started or restarted.

7.2.1.1 Repeat Message State

For nodes that are not in passive mode (refer to chapter 7.8.3) 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.

CANNM100: When the Repeat Message State is entered from Bus-Sleep Mode, Prepare-Bus-Sleep Mode, Normal Operation State or Ready Sleep State, the CanNm module shall start transmission of Network Management PDUs unless passive mode is enabled and/or communication is disabled.

CANNM101: When the NM-Timeout Timer expires in the Repeat Message State, the NM-Timeout Timer shall be restarted.

CANNM102: The NM shall stay in the Repeat Message State for a configurable amount of time determined by the **CANNM_REPEAT_MESSAGE_TIME** (configuration parameter); after that time the Repeat Message State shall be left.

CANNM103: When Repeat Message State is left, the Normal Operation State shall be entered, if the network has been requested (see CANNM104).

CANNM106: When Repeat Message State is left, the Ready Sleep State shall be entered, if the network has been released (see CANNM105).

CANNM107: When Repeat Message State is left and the option **CANNM_NODE_DETECTION_ENABLED** is enabled, the Repeat Message Bit shall be cleared.

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.

CANNM116: When the Normal Operation State is entered from Ready Sleep State, transmission of NM PDUs shall be started unless passive mode is enabled or the NM message transmission ability has been disabled.

CANNM117: When the NM-Timeout Timer expires in the Normal Operation State, the NM-Timeout Timer shall be restarted.

CANNM118: 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 (refer to CANNM105).

CANNM119: 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.

CANNM120: At Repeat Message Request (**CanNm_RepeatMessageRequest**) in the Normal Operation State, the Normal Operation State shall be left and the Repeat Message State shall be entered.

CANNM121: At Repeat Message Request in Normal Operation State the Repeat Message Bit shall be set.

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.

CANNM108: 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.

CANNM109: 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.

CANNM110: 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 (refer to CANNM104).

CANNM111: 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.

CANNM112: At Repeat Message Request (**`CanNm_RepeatMessageRequest`**) in the Ready Sleep State, the Ready Sleep State shall be left and the Repeat Message State shall be entered.

CANNM113: At Repeat Message Request in Ready Sleep State the Repeat Message Bit shall be set.

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.

CANNM114: When Prepare Bus-Sleep Mode is entered, the NM shall notify the upper layer by calling **`Nm_PrepareBusSleepMode`**.

CANNM115: The NM shall stay in the Prepare Bus-Sleep Mode for a configurable amount of time determined by the **`CANNM_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.

CANNM122: When the network has been requested in the Prepare Bus-Sleep Mode and Network Mode has been entered, a NM Message shall be transmitted if **`CANNM_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 **CANNM_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.

CANNM123: 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 (refer to CANNM095).

CANNM124: 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 (refer to CANNM095).

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.

CANNM125: If a configurable amount of time determined by the **CANNM_TIMEOUT_TIME** + **CANNM_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 **CANNM_TIMEOUT_TIME** and **CANNM_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 **CANNM_TIMEOUT_TIME** + **CANNM_WAIT_BUS_SLEEP_TIME** (both configuration parameters).

CANNM126: 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.

CANNM127: 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.

CANNM128: If **CanNm_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.

CANNM129: 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 (refer to CANNM095 and CANNM104).

7.3 Network states

Network states (i.e. ‘requested’ and ‘released’) are two additional states of the AUTOSAR CAN NM state machine that exist in parallel to the state machine. 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.

CANNM104: The optional service call **CanNm_NetworkRequest** shall request the network. I.e. network state shall be changed to ‘requested’.

CANNM105: The optional service call **CanNm_NetworkRelease** shall release the network. I.e. network state shall be changed to ‘released’.

7.4 Initialization

CANNM140: After reset the Network Management state shall be set to `NM_STATE_UNINIT`.

CANNM141: After successful initialization the Network Management state shall be set to `NM_STATE_BUS_SLEEP`, otherwise it shall be set to `NM_STATE_UNINIT`.

CANNM041: The function `CanNm_Init` shall initialize the CanNm.

CANNM142: The NM shall be initialized after CanIf is initialized and before any other NM service is called.

CANNM143: If initialized, by default, the network state shall be set to ‘released’

CANNM144: If initialized, by default, the Bus-Sleep Mode shall be entered.

CANNM145: Bus traffic should not be prohibited if AUTOSAR CAN NM is not initialized.

CANNM147: If `CanNm_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.

CANNM060: The function CanNm_Init shall select the active configuration set by means of a configuration pointer parameter being passed (see 8.3.1).

CANNM061: After initialization the CanNm Message Cycle Timer and NM Message Tx Timeout Timer shall be stopped.

Note:
No timer (CanNm Message Cycle Timer and NM Message Tx Timeout Timer) is needed if CANNM_PASSIVE_MODE_ENABLED is enabled, because no NM messages are transmitted by such nodes.

Note:
The NM Message Tx Timeout Timer is not needed in case of CANNM_IMMEDIATE_TXCONF_ENABLED is enabled.

CANNM023: After initialization the bus load reduction shall be deactivated if bus load reduction is enabled.

CANNM033: After initialization the transmission of NM messages shall be stopped.

CANNM039: If CanNm is not initialized a call of any CanNm function shall be rejected with the respective error code.

CANNM025: After initialization each byte of the user data bytes shall be set to `0xFF`.

CANNM085: After initialization the Control Bit Vector shall be set to `0x00`.

CANNM204: The NM State shall be initialized with `NM_STATE_UNINIT` for all NM instances that are configured to be inactive (configuration parameter `CANNM_CHANNEL_ACTIVE`).

7.5 Execution

7.5.1 Processor architecture

CANNM146: The AUTOSAR CAN 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.

7.5.2 Timing parameters

CANNM148: Timing parameters necessary for the AUTOSAR CAN NM shall be determined by the following configuration parameters:

- NM-Timeout Time: **CANNM_TIMEOUT_TIME**
- Repeat Message Time: **CANNM_REPEAT_MESSAGE_TIME**
- Wait Bus-Sleep Time: **CANNM_WAIT_BUS_SLEEP_TIME**
- Remote Sleep Indication Time: **CANNM_REMOTE_SLEEP_IND_TIME** (optional)

7.6 Communication Scheduling

7.6.1 Transmission

Note:
The transmission mechanisms described in this chapter are only relevant if the NM message transmission ability is enabled.

CANNM072: The transmission of NM messages shall be configurable by means of CANNM_PASSIVE_MODE_ENABLED (see chapter 10.2).

Note:
Passive nodes don't transmit NM messages, i.e. they cannot actively influence the shut down decision, but they do receive NM message in order to be able to shut down synchronous.

Note:
The transmission mechanisms described in this chapter are only relevant if CANNM_PASSIVE_MODE_ENABLED is disabled.

CANNM001: Two transmission modes should be provided by the CanNm:

1. Periodic transmission mode (mandatory). In this transmission mode the CanNm sends periodically NM messages.

2. Periodic transmission mode with bus load reduction (optional). In this transmission mode the CanNm transmits NM messages due to a specific algorithm. It ensures a reduced bus load.

Note:

The periodic transmission mode is used in the "Repeat Message State" and "Normal Operation State" if the bus load reduction mechanism is disabled.
The periodic transmission mode with bus load reduction is only used, in the "Normal Operation State" if the bus load reduction mechanism is enabled.
CANNM071: The immediate transmission confirmation mechanism shall be configurable by means of the `CANNM_IMMEDIATE_TXCONF_ENABLED` (see 10.2).

Note:
The immediate transmission confirmation mechanism is used for systems which don't want to use the actual confirmation from the CanIf.

Rationale:
If the bus access is completely regulated through an offline system design tool, the actual transmit confirmation by CanIf can be regarded as redundant. Since the maximum arbitration time is assumed to be known it is acceptable to immediately raise the confirmation at the transmission request time.

CANNM005: When entering the Repeat Message State from Ready Sleep State or Bus Sleep State, the transmission of the first NM PDU shall be delayed by the time indicated by CANNM_MSG_CYCLE_OFFSET in order to avoid bursts of NM messages.

CANNM032: If transmission of NM PDUs has been started and the CanNm Message Cycle Timer expires a NM message shall be transmitted by the CanIf function CanIf_Transmit.

Note:
If the call of CanIf_Transmit fails the Nm is informed by the Transmission Error handling described in chapter 7.9.

CANNM040: If the CanNm Message Cycle Timer expires it shall be restarted with CANNM_MSG_CYCLE_TIME.

CANNM034: If a NM message has been successfully transmitted the function `CanNm_TxConfirmation` shall be called by CanIf if `CANNM_IMMEDIATE_TXCONF_ENABLED` (configuration parameter) is disabled.

CANNM052: The bus load reduction mechanism shall be statically configurable by means of the `CANNM_BUS_LOAD_REDUCTION_ENABLED` parameter (see 10.2).

CANNM051: If transmission of NM PDUs has been stopped the CanNm Message Cycle Timer shall be canceled.

CANNM130: The optional service call **`CanNm_RequestBusSynchronization`** shall trigger transmission of a single NM PDU if **`CANNM_PASSIVE_MODE_ENABLED`** (configuration parameter) is not defined.

Rationale: This service is typically used for supporting the NM gateway extensions.

7.6.2 Reception

CANNM035: If a NM message has been successfully received the function `CanNm_RxIndication` shall be called by CanIf. The NM message shall be copied to an internal buffer.

CANNM037: If a NM message has been successfully received the Nm function `Nm_PduRxIndication` shall be called (triggered by the function `CanNm_RxIndication`) if `CANNM_PDU_RX_INDICATION_ENABLED` (configuration parameter) is defined.

7.7 Bus Load Reduction Mechanism (optional)

The transmission period of NM messages is usually determined by the timing parameter `CANNM_MSG_CYCLE_TIME`. This parameter has to be equal for all NM nodes which belong to a NM cluster. Without any action this would lead to a bus load which depends on the amount of members of the NM cluster. Even if bursts are prevented through a node specific timing parameter called `CANNM_MSG_OFFSET_TIME` a mechanism is necessary which reduces the bus load independently of the size of the NM cluster.

In order to achieve that the following two aspects have to be considered:

1. If a NM message has been successfully received the CanNm Message Cycle Timer is reloaded with the node specific timing parameter `CANNM_MSG_REDUCED_TIME`.
   The node specific time `CANNM_MSG_REDUCED_TIME` shall be greater than ½ `CANNM_MSG_CYCLE_TIME` and less than `CANNM_MSG_CYCLE_TIME`.

2. If a NM message has been successfully transmitted the CanNm Message Cycle Timer is reloaded with the NM cluster specific timing parameter `CANNM_MSG_CYCLE_TIME`.

This leads to the following behavior:
Only the two nodes with the smallest `CANNM_MSG_REDUCED_TIME` time transmit alternating NM messages on the network. If one of the nodes stops transmission, the node with the next smallest `CANNM_MSG_REDUCED_TIME` time will start to transmit NM messages. If there is only one node on the network that requires bus communication, one NM message per `CANNM_MSG_CYCLE_TIME` is transmitted.

The algorithm ensures that the bus load is limited to maximum two NM messages per CANNM_MSG_CYCLE_TIME.

An example can be found in chapter 11.

CANNM155: Busload reduction shall be statically configurable with use of the **CANNM_BUS_LOAD_REDUCTION_ENABLED** switch (configuration parameter).

CANNM156: 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.

CANNM157: When the Normal Operation State is entered from Repeat Message State or Ready Sleep State the busload reduction shall be activated.

CANNM069: If the bus load reduction mechanism is globally enabled, for a particular channel activated and a NM message has been successfully received the CanNm Message Cycle Timer shall be restarted with the node specific time CANNM_MSG_REDUCED_TIME**.**

7.8 Additional features

7.8.1 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.

CANNM149: Detection of remote sleep indication shall be statically configurable with use of the **CANNM_REMOTE_SLEEP_IND_ENABLED** switch (configuration parameter).

CANNM150: If no NM PDUs are received in the Normal Operation State for a configurable amount of time determined by the **CANNM_REMOTE_SLEEP_IND_TIME** (configuration parameter), the NM shall notify the Generic Network Management Interface that all other nodes in the cluster are ready to sleep (the so-called ‘Remote Sleep Indication’) by calling **Nm_RemoteSleepIndication.**

CANNM151: If Remote Sleep Indication has been previously detected and if a NM PDU is received in the Normal Operation State or Ready Sleep State again, the NM shall notify the Generic Network Management Interface that some nodes in the cluster are not ready to sleep anymore (the so-called ‘Remote Sleep Cancellation’) by calling **Nm_RemoteSleepCancelation**.

CANNM152: If Remote Sleep Indication has been previously detected and if Repeat Message State is entered from Normal Operation State, the NM shall notify the Generic Network Management Interface that some nodes in the cluster are not ready to sleep anymore (the so-called ‘Remote Sleep Cancellation’) by calling **Nm_RemoteSleepCancelation**.

CANNM153: Service call **CanNm_CheckRemoteSleepIndication** shall provide the information about current status of Remote Sleep Indication (i.e. already detected or not).

CANNM154: 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.

7.8.2 User Data (optional)

CANNM158: Support of NM user data shall be statically configurable with use of the **CANNM_USER_DATA_ENABLED** switch (configuration parameter).

CANNM159: When **CanNm_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.

CANNM160: When **CanNm_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.

7.8.3 Passive Mode (optional)

In the Passive Mode the node is only receiving NM messages but not transmitting any NM Messages.

CANNM161: Passive Mode shall be statically configurable with use of the **CANNM_PASSIVE_MODE_ENABLED** switch (configuration parameter).

CANNM162: Passive Mode shall be statically configured consistent for all instances within one ECU.

CANNM163: If Passive Mode is used (configuration parameter **CANNM_PASSIVE_MODE_ENABLED**) the following options must not be used:

- Bus Synchronization
  (configuration parameter **CANNM_BUS_SYNCHRONIZATION_ENABLED**)
- Bus Load Reduction
  (configuration parameter **CANNM_BUS_LOAD_REDUCTION_ENABLED**)
- Remote Sleep Indication
  (configuration parameter **CANNM_REMOTE_SLEEP_IND_ENABLED**)
- Node Detection
  (configuration parameter **CANNM_NODE_DETECTION_ENABLED**)
- Communication Control
  (configuration parameter **CANNM_COM_CONTROL_ENABLED**)

7.8.4 NM PDU Rx Indication (optional)

CANNM164: 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.

CANNM165: The optional service **Nm_PduRxIndication** shall be statically configurable. It shall be available if **CANNM_PDU_RX_INDICATION_ENABLED** is defined.

7.8.5 State change notification (optional)

CANNM166: All changes of the AUTOSAR CAN NM states shall be notified to the upper layer by calling **Nm_StateChangeNotification**.

CANNM167: The optional service **Nm_StateChangeNotification** shall be statically configurable. It shall be available if **CANNM_STATE_CHANGE_IND_ENABLED** is defined.

7.8.6 Communication Control (optional)

CANNM168: Communication Control shall be statically configurable with use of the **CANNM_COM_CONTROL_ENABLED** switch (configuration parameter).

CANNM169: If initialized, by default, the network NM PDU transmission ability shall be enabled.

CANNM170: The optional service **CanNm_DisableCommunication** shall disable the NM PDU transmission ability.

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. ComM ensures that the ECU is not shutdown as long as the NM PDU transmission ability is disabled.

CANNM172: The optional service **CanNm_DisableCommunication** shall return **NM_E_NOT_EXECUTED**, if the current mode is not Network Mode.

CANNM173: When the NM PDU transmission ability is disabled, the transmission of NM PDUs shall be stopped.

CANNM174: When the NM PDU transmission ability is disabled, the NM-Timeout Timer shall be stopped.

CANNM175: When the Network Management PDU transmission ability is disabled, the CanNm module shall stop the Remote Sleep Indication Timer.

CANNM176: The optional service **CanNm_EnableCommunication** shall enable the NM PDU transmission ability if the NM PDU transmission ability is disabled.

CANNM177: The optional service **CanNm_EnableCommunication** shall return **NM_E_NOT_EXECUTED** if the NM PDU transmission ability is enabled.

CANNM178: When the NM PDU transmission ability is enabled, the transmission of NM PDUs shall be started.

CANNM179: When the NM PDU transmission ability is enabled, the NM-Timeout Timer shall be restarted.

CANNM180: When the Network Management PDU transmission ability is enabled, the CanNm module shall re-start the Remote Sleep Indication Timer.

CANNM181: The optional service **CanNm_RequestBusSynchronization** shall return **NM_E_NOT_EXECUTED** if the NM PDU transmission ability is disabled.

7.9 Transmission Error Handling

CANNM073: The transmission error handling shall be inactive if `CANNM_PASSIVE_MODE_ENABLED` is enabled (see CANNM72) or `CANNM_IMMEDIATE_TXCONF_ENABLED` is enabled.

Rationale:
Transmission error handling makes only sense if a node is allowed to transmit NM messages and the real confirmation from the CanIf is evaluated.

CANNM064: The NM Message Tx Timeout Timer shall be started with `CANNM_MSG_TIMEOUT_TIME` when the transmission of NM messages is started.

CANNM065: The NM Message Tx Timeout Timer shall be restarted with `CANNM_MSG_TIMEOUT_TIME` when `CanNm_TxConfirmation` is called by the CanIf.

CANNM205: The NM Message Tx Timeout Timer shall be restarted as soon as possible after each NM Message transmit request if `CANNM_IMMEDIATE_TXCONF_ENABLED` is defined.

CANNM066: The function `Nm_TxTimeoutException` shall be called only once when the NM Message Tx Timeout Timer expires.

CANNM067: The NM Message Tx Timeout Timer shall be restarted with `CANNM_MSG_TIMEOUT_TIME` when `CanNm_RxIndication` is called by the CanIf if bus load reduction is activated.

CANNM068: The NM Message Tx Timeout Timer shall be stopped when the transmission of NM messages is stopped.

7.10 Message Structure

The figure below shows the default format of the NM PDU:

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 7	User data 5							
Byte 6	User data 4							
Byte 5	User data 3							
Byte 4	User data 2							
Byte 3	User data 1							
Byte 2	User data 0							
Byte 1	Control Bit Vector							
Byte 0	Source Node Identifier							

Figure 7-2 NM PDU Default Format

CANNM074: The location of the source node identifier shall be configurable by means of `CANNM_PDU_NID_POSITION` to Byte 0, Byte 1, or off (default: Byte 0).

CANNM075: The location of the control Bit vector shall be configurable by means of `CANNM_PDU_CBV_POSITION` to Byte 0, Byte 1, or off (default: Byte 1).

CANNM076: The length of the NM PDU shall be configurable to any integer value between (and including) 0 and 8 by means of `CANNM_PDU_LENGTH`, the difference between applied standardized bytes and length is user data.

The figure below describes the format of the Control Bit Vector:

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 1	Res	Res	Res	Res	Res	Res	Res	RptMsgRequest

Figure 7-3 Control Bit Vector

CANNM045: The Control Bit Vector shall consist of

- Bit 0: Repeat Message Request
  0: Repeat Message State not requested
  1: Repeat Message State requested
- Bit 1..7 are reserved for future extensions
  0 : Disabled / Reserved for future usage

Note: The Control Bit Vector is initialized with 0x00 during initialization (also refer to CANNM085).

CANNM013: The source node identifier shall be set with the configuration parameter `CANNM_NODE_ID` unless `CANNM_PDU_NID_POSITION` is set to off.

CANNM132: The optional service call **`CanNm_GetNodeIdentifier`** shall provide the node identifier out of the most recently received NM PDU.

CANNM133: The optional service call **CanNm_GetLocalNodeIdentifier** shall provide the node identifier configured for the local host node.

CANNM135: Support of Repeat Message Request Bit and Repeat Message State Request shall be statically configurable with use of the **CANNM_NODE_DETECTION_ENABLED** switch (configuration parameter).

CANNM136: The optional service call **CanNm_RepeatMessageRequest** shall be statically configured with use of the **CANNM_NODE_DETECTION_ENABLED** switch (configuration parameter).

CANNM137: If the optional service **CanNm_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.

CANNM138: The optional service call **CanNm_GetPduData** shall provide whole PDU data (Node ID, Control Bit Vector and User Data) of the most recently received NM PDU.

CANNM139: The optional service **CanNm_GetPduData** shall be statically configurable. It shall be available if **CANNM_NODE_ID_ENABLED** or **CANNM_NODE_DETECTION_ENABLED** or **CANNM_USER_DATA_ENABLED** is defined.

7.11 Functional requirements on CanNm API

CANNM014: If the node detection functionality is enabled, the function `Nm_RepeatMessageIndication` shall be called upon every reception of the RepeatMessageRequest bit if **CANNM_REPEAT_MSG_IND_ENABLED** is enabled.

CANNM053: The user data handling shall be configurable by means of the `CANNM_USER_DATA_ENABLED` parameter (see 10.2).

CANNM015: If the user data handling is enabled, the user data shall be set by the function `CanNm_SetUserData`**.**

CANNM031: If the user data handling is enabled, the user data of the most recently successfully received NM message shall be read by `CanNm_GetUserData`**.**

CANNM086: If `CANNM_USER_DATA_ENABLED` is enabled and `CANNM_USER_DATA_LENGTH` is set to `0x00` an error during configuration or compilation time shall be raised.

7.12 Error classification

CANNM018: The following errors shall be detectable by the CanNm depending on its build version (development/production mode).

Type or error	*Relevance*	*Related error code*	*Error Value*
API service used without module initialization	Development	CANNM_E_NO_INIT	0x01
API service called with wrong channel handle	Development	CANNM_E_INVALID_CHANNEL	0x02
CanNm initialization has failed, e.g. selected configuration set doesn't exist	Production	CANNM_E_INIT_FAILED	Assigned by the DEM
Call of CanIf function CanIf_Transmit has failed	Production	CANNM_E_CANIF_TRANSMIT_ERROR	Assigned by the 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	Production	CANNM_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	CANNM_E_DEV_NETWORK_TIMEOUT	0x11
Null pointer has been passed as an argument (Does not apply to function CanNm_Init)	Development	NM_E_NULL_POINTER	0x12h

CANNM019: Development errors shall be reported to the DET.

CANNM188: Reporting of development errors shall be statically configurable with use of the **CANNM_DEV_ERROR_DETECT** switch (configuration parameter).

CANNM189: Development errors shall not be returned by API functions; in case of a development error, the respective API function will return NM_E_NOT_OK, if applicable.

CANNM020: Production errors shall be reported to the DEM.

CANNM190: Production errors shall not be returned by API functions; in case of a production error, the respective API function will return NM_E_NOT_OK, if applicable.

CANNM191: If not initialized, the NM shall reject every API service apart from **CanNm_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

CANNM192: When NM API service with an invalid network 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 the invalid network handle shall be passed to DET as instance ID) and it shall return **NM_E_NOT_OK** to the calling function.

Note: The network handle is invalid if it is different from allowed configured values.

CANNM193: When the NM-Timeout Timer expires in the Repeat Message State, the NM shall report **NM_E_NETWORK_TIMEOUT** to Diagnostic Event Manager

CANNM194: When the NM-Timeout Timer expires in the Normal Operation State, the NM shall report **NM_E_NETWORK_TIMEOUT** to Diagnostic Event Manager

7.13 Scheduling of the main function

CANNM077: The `CanNm_MainFunction_<Instance Id>` functions shall be scheduled by the BSW scheduler (see [8]).

7.14 Application notes

7.14.1 Wakeup notification

Wakeup notification is defined in detail in the ECU State Manager specification.

7.14.2 Coordination of coupled networks

CANNM184: The AUTOSAR CAN 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.

CANNM185: Support of bus synchronization on demand shall be statically configurable with use of the **CANNM_BUS_SYNCHRONIZATION_ENABLED** switch (configuration parameter).

CANNM186: Service call **CanNm_RequestBusSynchronization** shall trigger transmission of a single NM PDU independently of the normal periodic transmission.

CANNM187: If **CanNm_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.

7.14.3 Interoperability with direct OSEK NM

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

8 API specification

CANNM016: Parameter value check shall be provided only in “development mode”. The execution of a service shall be rejected and the API shall inform the DET**.**

CANNM195: AUTOSAR CAN NM API consists of services, which are CAN specific and can be called whenever they are required; each service apart from **CanNm_Init** refers to one NM channel only.

8.1 Imported Types

Header file	***Imported Type***
Nm_Types.h	Nm_StateType
	Nm_ModeType
	Nm_ReturnType
ComStack_Types.h	PduIdType
	NetworkHandleType
Dem_Types.h	Dem_EventIdType
FrNm_Types.h	NetworkHandleType
PrimitiveTypes.h	PduInfoType
Std_Types.h	Std_VersionInfoType
	Std_ReturnType

8.2 Type Definitions

8.2.1 CanNm_ConfigType

This type shall contain the parameters of the container CanNm_GlobalConfig and its sub containers.

8.3 CanNm Functions called by the Nm

8.3.1 CanNm_Init

Service name:	CanNm_Init	

Syntax:	`void CanNm_Init(`
	`  const CanNm_ConfigType* const cannmConfigPtr`
	`)`	

Service ID[hex]:	0x00	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	cannmConfigPtr	Pointer to a selected configuration structure
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	

Description:	Initialize the complete CanNm module, i.e. all channels which are activated (see also configuration parameter CANNM_CHANNEL_ACTIVE) at configuration time are initialized. If a NULL pointer is passed as an argument to this function the default configuration shall be used. If an error has to be indicated to the DET the value 0x00 shall be used as the instance id. Caveats: This function has to be called after initialization of the CanIf. Configuration: Mandatory

8.3.2 CanNm_PassiveStartUp

Service name:	CanNm_PassiveStartUp	

Syntax:	`Nm_ReturnType CanNm_PassiveStartUp(`
	`  const NetworkHandleType nmChannelHandle`
	`)`	

Service ID[hex]:	0x01	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant (but not for the same NM-Channel)	
Parameters (in):	nmChannelHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Nm_ReturnType	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 NM. It triggers the transition from Bus-Sleep Mode to the Network Mode in Repeat Message State. 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 is initialized correctly. Configuration: Mandatory	

8.3.3 CanNm_NetworkRequest

Service name:	CanNm_NetworkRequest	

Syntax:	`Nm_ReturnType CanNm_NetworkRequest(`
	`  const NetworkHandleType nmChannelHandle`
	`)`	

Service ID[hex]:	0x02	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant (but not for the same NM-channel)	
Parameters (in):	nmChannelHandle	Identification of the NM-channel
Parameters	None	

(inout):		
Parameters (out):	None	
Return value:	Nm_ReturnType	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 is initialized correctly. Configuration: Optional (Only available if CANNM_PASSIVE_MODE_ENABLED is not defined).	

8.3.4 CanNm_NetworkRelease

Service name:	CanNm_NetworkRelease	

Syntax:	`Nm_ReturnType CanNm_NetworkRelease(`
	`  const NetworkHandleType nmChannelHandle`
	`)`	

Service ID[hex]:	0x03	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant (but not for the same NM-Channel)	
Parameters (in):	nmChannelHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Nm_ReturnType	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 is initialized correctly. Configuration: Optional (Only available if CANNM_PASSIVE_MODE_ENABLED is not defined).	

8.3.5 CanNm_DisableCommunication

Service name:	CanNm_DisableCommunication	
Syntax:	`Nm_ReturnType CanNm_DisableCommunication( const NetworkHandleType nmChannelHandle )`	
Service ID[hex]:	0x0c	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant (but not for the same NM-channel)	
Parameters (in):	nmChannelHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Nm_ReturnType	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 is initialized correctly. Configuration: Optional (Only available if CANNM_COM_CONTROL_ENABLED is defined).	

8.3.6 CanNm_EnableCommunication

Service name:	CanNm_EnableCommunication	
Syntax:	`Nm_ReturnType CanNm_EnableCommunication( const NetworkHandleType nmChannelHandle )`	
Service ID[hex]:	0x0d	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant (but not for the same NM-channel)	
Parameters (in):	nmChannelHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Nm_ReturnType	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 is initialized correctly. Configuration: Optional (Only available if CANNM_COM_CONTROL_ENABLED is defined).	

8.3.7 CanNm_SetUserData

Service name:	CanNm_SetUserData	

Syntax:	`Nm_ReturnType CanNm_SetUserData(`
	`  const NetworkHandleType nmChannelHandle,`
	`  const uint8* const nmUserDataPtr`
	`)`	

Service ID[hex]:	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 (inout):	None	
Parameters (out):	None	
Return value:	Nm_ReturnType	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. Caveats: CanNm is initialized correctly. Configuration: Optional (Only available if CANNM_USER_DATA_ENABLED is defined and CANNM_PASSIVE_MODE_ENABLED is not defined).	

8.3.8 CanNm_GetUserData

Service name:	CanNm_GetUserData	

Syntax:	`Nm_ReturnType CanNm_GetUserData(`
	`  const NetworkHandleType nmChannelHandle,`
	`  uint8* const nmUserDataPtr`
	`)`	

Service ID[hex]:	0x05	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	nmChannelHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	nmUserDataPtr	Pointer where user data out of the most recently received NM message shall be copied to
Return value:	Nm_ReturnType	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. Caveats: CanNm is initialized correctly. Configuration: Optional (Only available if CANNM_USER_DATA_ENABLED is defined).	

8.3.9 CanNm_GetNodeIdentifier

Service name:	CanNm_GetNodeIdentifier	

Syntax:	`Nm_ReturnType CanNm_GetNodeIdentifier(`
	`  const NetworkHandleType nmChannelHandle,`
	`  uint8* const nmNodeIdPtr`
	`)`	

Service ID[hex]:	0x06	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	nmChannelHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	nmNodeIdPtr	Pointer where node identifier out of the most recently received NM PDU shall be copied to
Return value:	Nm_ReturnType	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. Caveats: CanNm is initialized correctly. Configuration: Optional (Only available if CANNM_NODE_ID_ENABLED is defined).	

8.3.10 CanNm_GetLocalNodeIdentifier

Service name:	CanNm_GetLocalNodeIdentifier	

Syntax:	`Nm_ReturnType CanNm_GetLocalNodeIdentifier(`
	`  const NetworkHandleType nmChannelHandle,`
	`  uint8* const nmNodeIdPtr`
	`)`	

Service ID[hex]:	0x07	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	nmChannelHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	nmNodeIdPtr	Pointer where node identifier of the local node shall be copied to
Return value:	Nm_ReturnType	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. Caveats: CanNm is initialized correctly. Configuration: Optional (Only available if CANNM_NODE_ID_ENABLED is defined).	

8.3.11 CanNm_RepeatMessageRequest

Service name:	CanNm_RepeatMessageRequest

Syntax:	`Nm_ReturnType CanNm_RepeatMessageRequest(`
	`  const NetworkHandleType nmChannelHandle`
	`)`	

Service ID[hex]:	0x08	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant (but not for the same NM-channel)	
Parameters (in):	nmChannelHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Nm_ReturnType	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. Caveats: CanNm is initialized correctly. Configuration: Optional (Only available if CANNM_NODE_DETECTION_ENABLED is defined).	

8.3.12 CanNm_GetPduData

Service name:	CanNm_GetPduData	

Syntax:	`Nm_ReturnType CanNm_GetPduData(`
	`  const NetworkHandleType nmChannelHandle,`
	`  uint8* const nmPduDataPtr`
	`)`	

Service ID[hex]:	0x0a	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	nmChannelHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	nmPduDataPtr	Pointer where NM PDU shall be copied to
Return value:	Nm_ReturnType	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. Caveats: CanNm is initialized correctly. Configuration: Optional (Only available if CANNM_NODE_ID_ENABLED or CANNM_NODE_DETECTION_ENABLED or CANNM_USER_DATA_ENABLED is defined).	

8.3.13 CanNm_GetState

Service name:	CanNm_GetState

Syntax:	`Nm_ReturnType CanNm_GetState(`
	`  const NetworkHandleType nmChannelHandle,`
	`  Nm_StateType* const nmStatePtr,`

	`Nm_ModeType* const nmModePtr`)	
Service ID[hex]:	0x0b	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	nmChannelHandle	Identification of the NM-channel
Parameters (inout):	None	
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_ReturnType	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 is initialized correctly. Configuration: Mandatory	

8.3.14 CanNm_GetVersionInfo

Service name:	CanNm_GetVersionInfo	

Syntax:	`void CanNm_GetVersionInfo(`
	`  Std_VersionInfoType* versioninfo`
	)	

Service ID[hex]:	0xf1	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	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: - Module Id - Vendor Id - Vendor specific version numbers (BSW00407). Note: This function can be called even if CanNm is not initialized. Configuration: Optional (only available if CANNM_VERSION_INFO_API is defined).	

8.3.15 CanNm_RequestBusSynchronization

Service name:	CanNm_RequestBusSynchronization

Syntax:	`Nm_ReturnType CanNm_RequestBusSynchronization(`
	`  const NetworkHandleType nmChannelHandle`
	)

Service ID[hex]:	0xc0	
Sync/Async:	Asynchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	nmChannelHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Nm_ReturnType	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 is initialized correctly. Configuration: Optional (Only available if CANNM_BUS_SYNCHRONIZATION_ENABLED is defined) and CANNM_PASSIVE_MODE_ENABLED is not defined.	

8.3.16 CanNm_CheckRemoteSleepIndication

Service name:	CanNm_CheckRemoteSleepIndication	
Syntax:	`Nm_ReturnType CanNm_CheckRemoteSleepIndication( const NetworkHandleType nmChannelHandle, boolean* const nmRemoteSleepIndPtr )`	
Service ID[hex]:	0xd0	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant (but not for the same NM-channel)	
Parameters (in):	nmChannelHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	nmRemoteSleepIndPtr	Pointer where check result of remote sleep indication shall be copied to
Return value:	Nm_ReturnType	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 is initialized correctly. Configuration: Optional (Only available if CANNM_REMOTE_SLEEP_INDICATION_ENABLED is defined).	

8.4 CanNm functions called by the CanIf

8.4.1 CanNm_TxConfirmation

Service name:	CanNm_TxConfirmation
Syntax:	`void CanNm_TxConfirmation(`

	PduIdType canNmTxPduId)
Service ID[hex]:	0x0f
Sync/Async:	Synchronous
Reentrancy:	Reentrant (but not within the same channel)
Parameters (in):	canNmTxPduId: Identification of the network through PDU-ID
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This service confirms a previous successfully processed CAN transmit request. This callback service is called by the CanIf and implemented by the CanNm. The value passed to CanNm via the API parameter canNmTxPduId shall refer to the NM channel handle, i.e. a mapping from PduId to NM channel handle is not necessary. Inform the DET (if enabled) if function call has failed because of the following reasons: - Invalid channel handle (CANNM_E_INVALID_CHANNEL) - CanNm was not initialized (CANNM_E_NO_INIT) If an error has to be indicated to the DET the value of NM channel handle shall be used as the instance id. Caveats: The call context is either on interrupt level (interrupt mode) or on task level (polling mode). This callback service is re-entrant for multiple CAN controller usage. The CanNm module is initialized correctly. Configuration: Optional (Only available if CANNM_IMMEDIATE_TXCONF_ENABLED is defined).

8.4.2 CanNm_RxIndication

Service name:	CanNm_RxIndication	
Syntax:	void CanNm_RxIndication(PduIdType canNmRxPduId, const PduInfoType* PduInfoPtr)	
Service ID[hex]:	0x10	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant (but not within the same channel)	
Parameters (in):	canNmRxPduId	Identification of the network through PDU-ID
	PduInfoPtr	Contains the length (SduLength) of the received I-PDU and a pointer to a buffer (SduDataPtr) containing the I-PDU.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	This service indicates a successful reception of a received NM message to the CanNm after passing all filters and validation checks. This callback service is called by the CAN Interface and implemented by the CanNm. It is called in case of a receive indication event (i.e. ISR is triggered) of the CAN driver.	

	The value passed to CanNm via the API parameter canNmRxPduId shall refer to the NM channel handle, i.e. a mapping from PduId to NM channel handle is not necessary. Inform the DET (if enabled) if function call has failed because of the following reasons: - Invalid channel handle (CANNM_E_INVALID_CHANNEL) - CanNm was not initialized (CANNM_E_NO_INIT) If an error has to be indicated to the DET the value of NM channel handle shall be used as the instance id. Caveats: Until this service returns the CAN Interface will not access PduInfoPtr. PduInfoPtr is only valid and can be used by upper layers until the indication returns. CAN Interface guarantees that the number of configured bytes for this canNmRxPduId is valid. The call context is either on interrupt level (interrupt mode) or on task level (polling mode). This callback service is re-entrant for multiple CAN controller usage. The CanNm module is initialized correctly. Configuration: Mandatory

8.5 Scheduled Functions

8.5.1 CanNm_MainFunction_<Instance Id>

Service name:	CanNm_MainFunction_<Instance_Id>

Syntax:	`void CanNm_MainFunction_<Instance_Id>(`
	
	`)`

Service ID[hex]:	0x13
Timing:	FIXED_CYCLIC
Description:	Main function of the CanNm which processes the algorithm describes in that document. e.g. CanNm_MainFunction_0() represents the CanNm instance for the CAN channel 0 CanNm_MainFunction_1() represents the CanNm instance for the CAN channel 1 ... Inform the DET (if enabled) if function call has failed because of the following reasons: - CanNm was not initialized (CANNM_E_INVALID_CHANNEL) If an error has to be indicated to the DET the <Instance Id> shall be used as the instance id. Caveats: CanNm is initialized correctly, i.e. the function shall be robust if one or more channels are not initialized. Configuration: Mandatory

8.6 Expected Interfaces

In this chapter all interfaces required from other modules are listed.

8.6.1 Mandatory Interfaces

This chapter defines all interfaces which are required to fulfill the core functionality of the module.

Required functions

8.6.1.1 Functions of Generic Network Management Interface

Callback function	*Module*	*Description*
`void Nm_NetworkStartIndication ( const NetworkHandleType );`	Generic Network Management 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 ( const NetworkHandleType );`	Generic Network Management Interface	Notification that the network management has entered Network Mode. The callback function shall enable transmission of application messages.
`Void Nm_PrepareBusSleepMode ( const NetworkHandleType );`	Generic Network Management Interface	Notification that the network management has entered Prepare Bus-Sleep Mode. The callback function shall disable transmission of application messages.
`Void Nm_BusSleepMode ( const NetworkHandleType );`	Generic Network Management 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.

8.6.1.2 Functions of Diagnostic Event Manager

API function	*Module*	*Description*
`Std_ReturnType Dem_ReportErrorStatus ( Dem_EventIdType EventId, Dem_EventStatusType EventStatus )`	DEM	Service for reporting Errors during start up and normal operation to the DEM.

8.6.2 Optional Interfaces

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

API function	*Module*	*Description*	*Configuration parameter (description see chapter 10)*
`CanIf_Transmit`	CanIf	Service to request a transmission of a CAN	`CANNM_PASSIVE_MODE_ENABLED`

		message	

8.6.2.1 Functions of Generic Network Management Interface

Callback function	*Module*	*Description*

`void` **`Nm_RemoteSleepIndication`**
`(`
`  const NetworkHandleType`
`);`
	Generic Network Management 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.
		Configuration parameter:
		`CANNM_REMOTE_SLEEP_IND_ENABLED`

`void` **`Nm_RemoteSleepCancelation`**
`(`
`  const NetworkHandleType`
`);`
	Generic Network Management 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.
		Configuration parameter:
		`CANNM_REMOTE_SLEEP_IND_ENABLED`

`void` **`Nm_PduRxIndication`**
`(`
`  const NetworkHandleType`
`);`
	Generic Network Management Interface	Notification that a NM message has been received.
		Configuration parameter:
		`CANNM_PDU_RX_INDICATION_ENABLED`

`Nm_StateChangeNotification`
`(`
`  const NetworkHandleType,`
`  const Nm_StateType nmPreviousState,`
`  const Nm_StateType nmCurrentState`
`)`
	Generic Network Management Interface	Notification that the CAN NM state has changed.
		Configuration parameter:
		`CANNM_STATE_CHANGE_IND_ENABLED`

`void` **`Nm_RepeatMessageIndication`**
`(`
`  const NetworkHandleType`
`);`
	Generic Network Management Interface	Notification that a RepeatMessageRequest bit has been received.
		Configuration parameter:
		`CANNM_REPEAT_MSG_IND_ENABLED`

`void Nm_TxTimeoutException`
`(`
`  const NetworkHandleType`
`);`
	Generic Network Management Interface	Notification that the transmission error handling has detected a transmission error.
		Configuration parameter:
		`CANNM_PASSIVE_MODE_ENABLED` is not defined and
		`CANNM_IMMEDIATE_TXCONF_ENABLED` is not defined.

8.6.2.2 Functions of Development Error Tracer

Callback function	*Module*	*Description*

`void` **`Det_ReportError`**
`(`
`  uint16 ModuleId,`
`  uint8  InstanceId,`
`  uint8  ApiId,`
`  uint8  ErrorId`
`)`
	DET	Service for reporting of development errors in the development mode.
		Configuration parameter:
		`CANNM_DEV_ERROR_DETECT`

8.6.3 Configurable interfaces

Not applicable

8.6.4 Job End Notification

Not applicable

8.7 Parameter check

CANNM196: If detection of development errors is enabled by `CANNM_DEV_ERROR_DETECT` (configuration parameter), then for all CAN NM API services validity check of input parameters shall be made.
Exception: The NULL Pointer check of input parameters shall not be done for **`CanNm_Init`**

CANNM197: 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.

CANNM198: 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.

CANNM199: 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

CANNM200: 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 following figure 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.

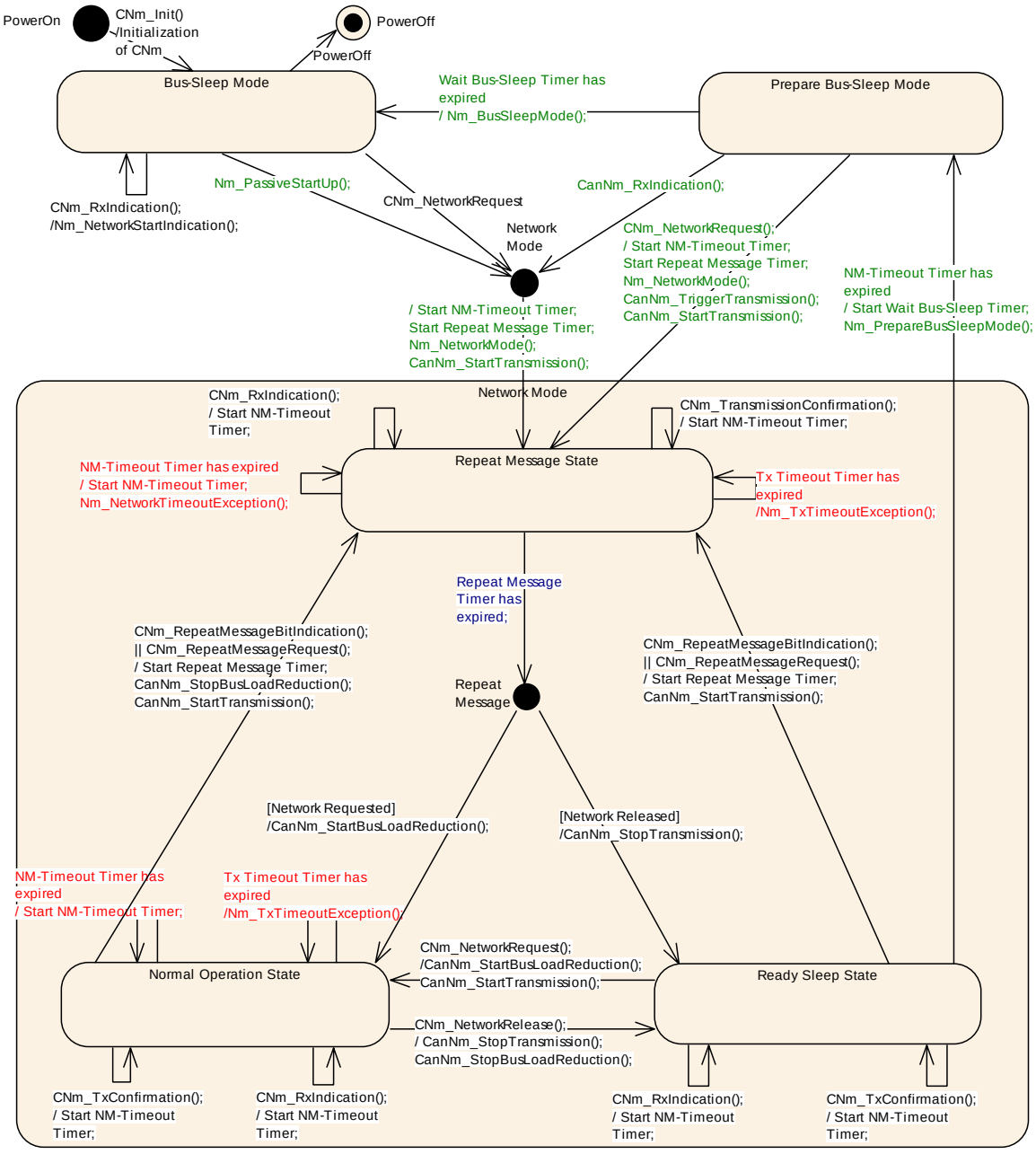
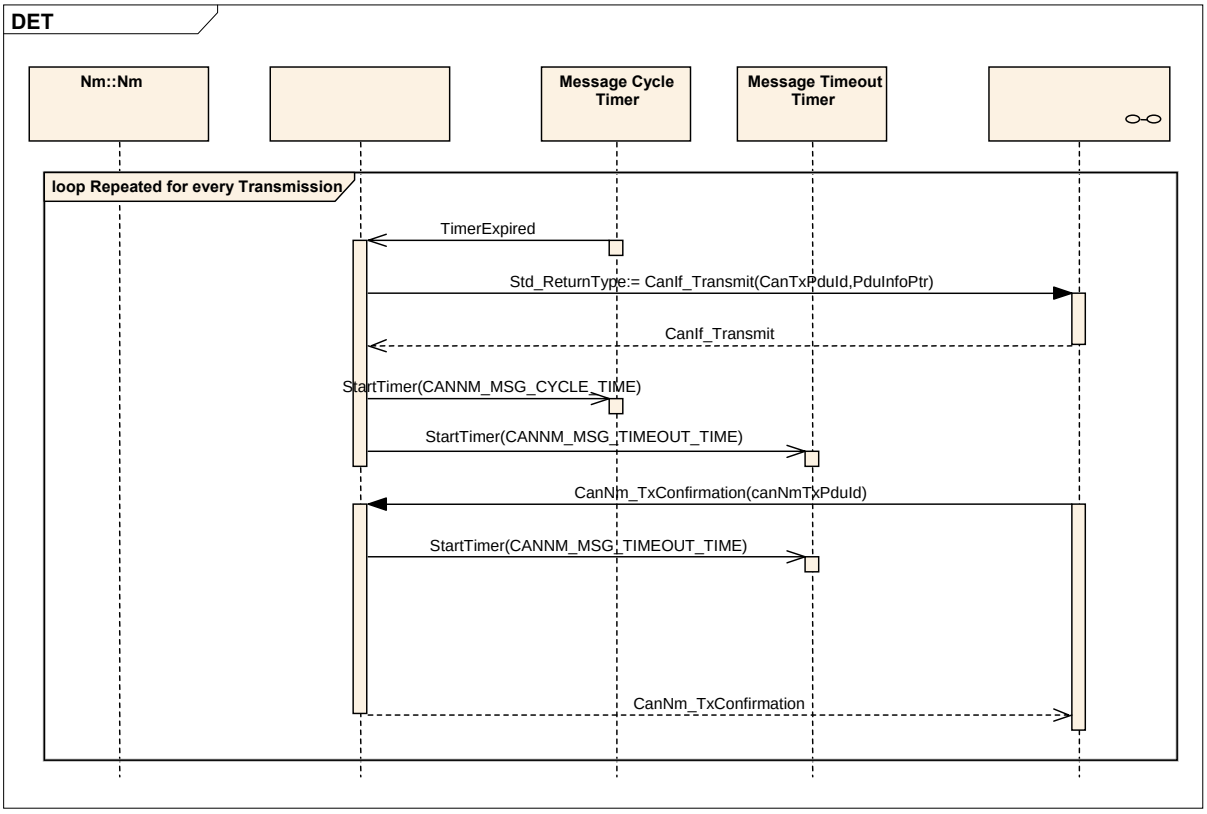


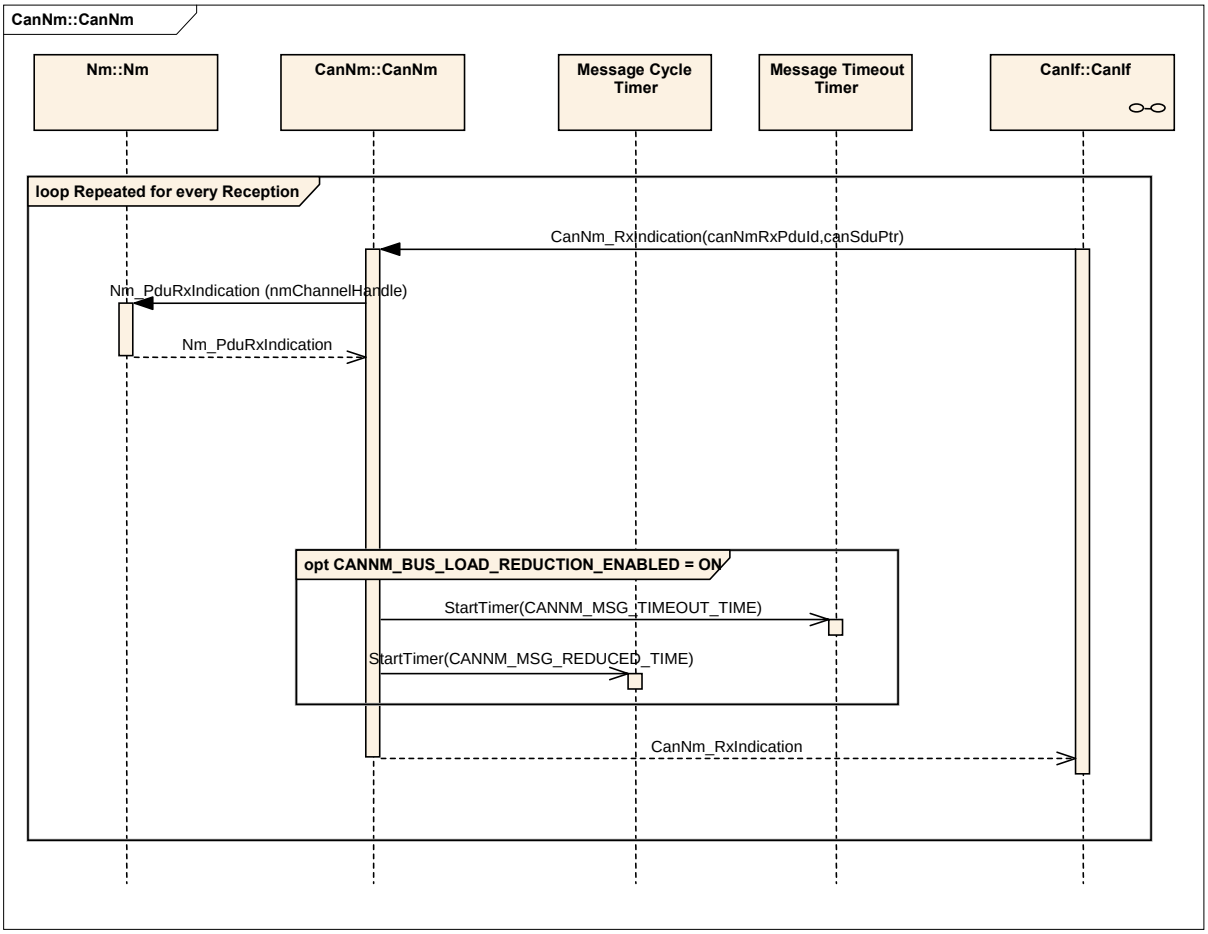
Figure 8-1

9 Sequence diagrams

9.1 CanNm Transmission



9.2 CanNm Reception



10 Configuration specification

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. We intend to leave chapter 10.1 in the specification to guarantee comprehension.

Chapter 10.2 specifies the structure (containers) and the parameters of the module CanNm.

Chapter 10.3 specifies published information of the module CanNm.

10.1 How to read this chapter

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

- AUTOSAR Layered Software Architecture [1]
- AUTOSAR ECU Configuration Specification [7]
  This document describes the AUTOSAR configuration methodology and the AUTOSAR configuration metamodel 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.

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.

The configuration parameters are divided in parameters which are used to enable features, parameters which affect all instances of the CanNm and parameters which affect the respective instances of the CanNm.

CANNM026: All configuration items shall be located outside the kernel of the module.

10.2.1 Variants

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

Variant 2: All configuration parameters of the container **`CanNm_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 **`CanNm_ChannelConfig`** are configurable at post-build time. The parameters contained in **`CanNm_GlobalConfig`** are configurable at pre-compile time
Use case: ECU configuration can be flashed (L) and selected during initialization phase (M).

Note:
The possibility to select a configuration (post-build time type L) is only explicitly mentioned for Variant 3, but from a technical perspective it is also possible to provide this configuration variant for variant 1 and 2.

10.3 Containers and configuration parameters

10.3.1 CanNm

Module Name	*CanNm*
Module Description	Configuration Parameters for the Can Nm module.

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CanNmGlobalConfig	1	This container contains the global configuration parameter of the CanNm. The parameters and the parameters of the sub containers shall be mapped to the C data type CanNm_ConfigType (for parameters where it is possible) which is passed to the CanNm_Init function. This container is a MultipleConfigurationContainer (only for variant 3), i.e. this container and its sub-containers exit once per configuration set.

10.3.2 CanNm_GlobalConfig General

CANNM201: The Global Scope specifies configuration parameter that shall be defined in the module's configuration header file **`CanNm_Cfg.h`**.

10.3.3 CanNmGlobalConfig

SWS Item	**--**
Container Name	CanNmGlobalConfig{CanNm_GlobalConfig} [Multi Config Container]
Description	This container contains the global configuration parameter of the CanNm. The parameters and the parameters of the sub containers shall be mapped to the C data type CanNm_ConfigType (for parameters where it is possible) which is passed to the CanNm_Init function. This container is a MultipleConfigurationContainer (only for variant 3), i.e. this container and its sub-containers exit once per configuration set.
Configuration Parameters	

SWS Item	**--**		
Name	CanNmBusLoadReductionEnabled {CANNM_BUS_LOAD_REDUCTION_ENABLED}		
Description	Pre-processor switch for enabling busload reduction support.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module dependency: Must not be defined if CanNmPassiveModeEnabled is defined.		

SWS Item	**--**
Name	CanNmBusSynchronizationEnabled {CANNM_BUS_SYNCHRONIZATION_ENABLED}
Description	Pre-processor switch for enabling bus synchronization support. This feature is required for gateway nodes only. calculationFormula = If (CanNmPassiveModeEnabled == False) then Equal(NmBusSynchronizationEnabled) else Equal(False)

Multiplicity	1		
Type	DerivedBooleanParamDef		
Default value	--		
calculationFormula	if (cannmpassivemodeenabled == false) then equal(nmbussynchronizationenabled) else equal(false)		
calculationLanguage	informal		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**--**		
Name	CanNmComControlEnabled {CANNM_COM_CONTROL_ENABLED}		
Description	Pre-processor switch for enabling the Communication Control support. calculationformula = Equal(NmComControlEnabled)		
Multiplicity	1		
Type	DerivedBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**--**		
Name	CanNmConfigPtr {CANNM_CANNM_CONFIG_PTR}		
Description	Pointer to configuration of CanNm.		
Multiplicity	1		
Type	IntegerParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**--**		
Name	CanNmDevErrorDetect {CANNM_DEV_ERROR_DETECT}		
Description	Pre-processor switch for enabling development error detection support.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**--**		
Name	CanNmImmediateRestartEnabled {CANNM_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.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	

Scope / Dependency	scope: Module dependency: Must not be defined if CanNmPassiveModeEnabled is defined.

SWS Item	**--**		
Name	CanNmImmediateTxconfEnabled {CANNM_IMMEDIATE_TXCONF_ENABLED}		
Description	Enable/disable the immediate tx confirmation.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module dependency: CanNmImmediateTxconfEnabled shall not be enabled if CanNmPasiveModeEnabled is enabled.		

SWS Item	**--**		
Name	CanNmNodeDetectionEnabled {CANNM_NODE_DETECTION_ENABLED}		
Description	Pre-processor switch for enabling the node detection support. calculationFormula = If(CanNmNodeIdEnabled == True) then Equal(NmNodeDetectionEnabled) else Equal(False)		
Multiplicity	1		
Type	DerivedBooleanParamDef		
Default value	--		
calculationFormula	if(cannmnodeidenabled == true) then equal(nmnodedetectionenabled) else equal(false)		
calculationLanguage	informal		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**--**		
Name	CanNmNodeIdEnabled {CANNM_NODE_ID_ENABLED}		
Description	Pre-processor switch for enabling the source node identifier. calculationFormula = Equal(NmNodeIdEnabled)		
Multiplicity	1		
Type	DerivedBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Network		

SWS Item	**--**		
Name	CanNmNumberOfChannels {CANNM_NUMBER_OF_CHANNELS}		
Description	Number of Can NM channels allowed within one ECU.		
Multiplicity	1		
Type	IntegerParamDef		
Range	1 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	

	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**--**		
Name	CanNmPassiveModeEnabled {CANNM_PASSIVE_MODE_ENABLED}		
Description	Pre-processor switch for enabling support of the Passive Mode. calculationFormula = Equal(NmPassiveModeEnabled)		
Multiplicity	1		
Type	DerivedBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**--**		
Name	CanNmPduRxIndicationEnabled {CANNM_PDU_RX_INDICATION_ENABLED}		
Description	Pre-processor switch for enabling the PDU Rx Indication. calculationFormula = Equal(NmPduRxIndicationEnabled)		
Multiplicity	1		
Type	DerivedBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**--**		
Name	CanNmRemoteSleepIndEnabled {CANNM_REMOTE_SLEEP_IND_ENABLED}		
Description	Pre-processor switch for enabling remote sleep indication support. This feature is required for gateway nodes only. calculationFormula = If (CanNmPassiveModeEnabled == False) then Equal(NmRemoteSleepIndEnabled) else Equal(False)		
Multiplicity	1		
Type	DerivedBooleanParamDef		
Default value	--		
calculationFormula	if (cannmpassivemodeenabled == false) then equal(nmremotesleepindenabled) else equal(false)		
calculationLanguage	informal		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**--**		
Name	CanNmRepeatMsgIndEnabled {CANNM_REPEAT_MSG_IND_ENABLED}		
Description	Enable/disable the notification that a RepeatMessageRequest bit has been received. calculationformula = Equal(NmRepeatMsgIndEnabled)		
Multiplicity	1		
Type	DerivedBooleanParamDef		
Default value	--		
calculationFormula	equal(nmrepeatmsgindenabled)		
calculationLanguage	informal		

ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**--**		
Name	CanNmStateChangeIndEnabled {CANNM_STATE_CHANGE_IND_ENABLED}		
Description	Pre-processor switch for enabling the CAN NM state change notification. calculationFormula = Equal(NmStateChangeIdEnabled)		
Multiplicity	1		
Type	DerivedBooleanParamDef		
Default value	--		
calculationFormula	equal(nmstatechangeidenabled)		
calculationLanguage	informal		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**--**		
Name	CanNmUserDataEnabled {CANNM_USER_DATA_ENABLED}		
Description	Pre-processor switch for enabling user data support. calculationFormula = Equal(NmUserDataEnabled)		
Multiplicity	1		
Type	DerivedBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Network		

SWS Item	**--**		
Name	CanNmVersionInfoApi {CANNM_VERSION_INFO_API}		
Description	Pre-processor switch for enabling version info API support.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CanNmChannelConfig	1..*	This container contains the channel specific configuration parameter of the CanNm.

10.3.4 CanNm_ChannelConfig general

CANNM202: The container CanNm_ChannelConfig specifies configuration parameter that shall be located in a data structure of type **`CanNm_ConfigType`**.

CANNM203: Runtime configurable parameters listed below shall be configurable for each NM-cluster separately.

10.3.5 CanNmChannelConfig

SWS Item	**--**
Container Name	CanNmChannelConfig{CanNm_ChannelConfig}
Description	This container contains the channel specific configuration parameter of the CanNm.
Configuration Parameters	

SWS Item	**--**		
Name	CanNmBusLoadReductionActive {CANNM_BUS_LOAD_REDUCTION_ACTIVE}		
Description	This parameter defines if bus load reduction for the respective NM channel is active or not.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: Instance dependency: Shall only be True if CanNmBusLoadReductionEnabled is True.		

SWS Item	**--**		
Name	CanNmChannelActive {CANNM_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). If this parameter is set to FALSE the respective NM instance shall not be used during runtime.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: Instance		

SWS Item	**--**		
Name	CanNmMainFunctionPeriod {CANNM_MAIN_FUNCTION_PERIOD}		
Description	Call cycle in seconds of CanNm_MainFunction_x for the respective instance.		
Multiplicity	1		
Type	FloatParamDef		
Range	0.0010 .. 0.255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: Instance		

SWS Item	**--**

Name	CanNmMsgCycleOffset {CANNM_MSG_CYCLE_OFFSET}		
Description	Time offset in the periodic transmission node. It determines the start delay of the transmission. Specified in seconds. This parameter is only valid if CanNmPassiveModeEnabled is False.		
Multiplicity	1		
Type	FloatParamDef		
Range	0.0 .. 65.535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: Instance dependency: Parameter value < CanMsgCycleTime		

SWS Item	**--**		
Name	CanNmMsgCycleTime {CANNM_MSG_CYCLE_TIME}		
Description	Period of a NM-message in seconds. It determines the periodic rate in the "periodic transmission mode with bus load reduction" and is the basis for transmit scheduling in the "periodic transmission mode without bus load reduction". This parameter is only valid if CanNmPassiveModeEnabled is False.		
Multiplicity	1		
Type	FloatParamDef		
Range	0.0010 .. 65.535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: Network dependency: NmTimeoutTime = n*CanNmMsgCycleTime		

SWS Item	**--**		
Name	CanNmMsgReducedtime {CANNM_MSG_REDUCED_TIME}		
Description	Node specific bus cycle time in the periodic transmission mode with bus load reduction. Specified in seconds. This parameter is only valid if CanNmBusLoadReductionEnabled == True and CanNmBusLoadReductionActive == True and CanNmPassiveModeEnabled == False Otherwise this parameter is not used.		
Multiplicity	1		
Type	FloatParamDef		
Range	0.0010 .. 65.535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: Instance dependency: 0,5 * CanNmMsgCycleTime <= CanNmMsgReducedTime < CanNmMsgCycleTime		

SWS Item	**--**
Name	CanNmMsgTimeoutTime {CANNM_MSG_TIMEOUT_TIME}
Description	Transmission Timeout of NM-message. If there is no transmission confirmation by the CAN Interface within this timeout, the CANNM module shall give an error notification.

	This parameter is only valid if CANNM_PASSIVE_MODE_ENABLED is disabled.		
Multiplicity	1		
Type	FloatParamDef		
Range	1.0 .. 65.535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: Instance dependency: CanNmMsgTimeoutTime = n*CanNmMsgCycleTime		

SWS Item	**--**		
Name	CanNmNodeId {CANNM_NODE_ID}		
Description	Node identifier of local node. This parameter is only valid if CanNmPassiveModeEnabled = False and CanNmNodeDetectionEnabled = True		
Multiplicity	1		
Type	IntegerParamDef		
Range	0 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: Instance		

SWS Item	**--**		
Name	CanNmPduCbvPosition {CANNM_PDU_CBV_POSITION}		

Description	Defines the position of the control bit vector within the NM PDU.
	The value of the parameter represents the location of the control bit vector in the NM PDU (CanNmPduByte0 means byte 0, CanNmPduByte1 means byte 1, CanNmPduOff means source node identifier is not part of the NM PDU)
	if(CANNM_PDU_CBV_POSITION != CANNM_PDU_OFF && CANNM_PDU_NID_POSITION != CANNM_PDU_OFF) then CANNM_PDU_CBV_POSITION != CANNM_PDU_NID_POSITION
	if(CANNM_PDU_CBV_POSITION != CANNM_PDU_OFF && CANNM_PDU_NID_POSITION == CANNM_PDU_OFF) then CANNM_PDU_CBV_POSITION = CANNM_PDU_BYTE0
	ImplementationType: CanNm_PduPositionType		

Multiplicity	1		
Type	EnumerationParamDef		
Range	CANNM_PDU_BYTE_0	Byte 0 is used	
	CANNM_PDU_BYTE_1	Byte 1 is used	
	CANNM_PDU_OFF	Control Bit Vector is not used	
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: ECU dependency: CanNmPduNidPosition		

SWS Item	**--**
Name	CanNmPduLength {CANNM_PDU_LENGTH}
Description	Defines the length of the NM PDU.
Multiplicity	1

Type	IntegerParamDef		
Range	0 .. 8		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**--**		
Name	CanNmPduNidPosition {CANNM_PDU_NID_POSITION}		

Description	Defines the position of the source node identifier within the NM PDU.
	The value of the parameter represents the location of the source node identifier in the NM PDU (CanNMPduByte0 means byte 0, CanNmPduByte1 means byte 1, CanNmPduOff means source node identifier is not part of the NM PDU)
	if(CANNM_PDU_NID_POSITION != CANNM_PDU_OFF && CANNM_PDU_CBV_POSITION != CANNM_PDU_OFF) then CANNM_PDU_NID_POSITION != CANNM_PDU_CBV_POSITION
	if(CANNM_PDU_NID_POSITION != CANNM_PDU_OFF && CANNM_PDU_CBV_POSITION == CANNM_PDU_OFF) then CANNM_PDU_NID_POSITION = CANNM_PDU_BYTE0
	ImplementationType: CanNm_PduPositionType		

Multiplicity	1		
Type	EnumerationParamDef		
Range	CANNM_PDU_BYTE_0	Byte 0 is used	
	CANNM_PDU_BYTE_1	Byte 1 is used	
	CANNM_PDU_OFF	Node Identification is not used	
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: Network dependency: CanNmPduCbvPosition		

SWS Item	**--**		
Name	CanNmRemoteSleepIndTime {CANNM_REMOTE_SLEEP_IND_TIME}		
Description	Timeout for Remote Sleep Indication. It defines the time in seconds how long it shall take to recognize that all other nodes are ready to sleep. Typically it should be equal to: n * CanNmMsgCycleTime, where n denotes the number of NM-Messages that are normally sent before Remote Sleep Indication is detected. The value of n decremented by one determines the amount of lost NM-Messages that can be tolerated by the Remote Sleep Indication procedure. The value 0 denotes that no Remote Sleep Indication functionality is configured.		
Multiplicity	1		
Type	FloatParamDef		
Range	0.0010 .. 65.535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: Instance dependency: CanNmRemoteSleepIndTime = n * CanNmMsgCycleTime		

SWS Item	**--**

Name	CanNmRepeatMessageTime {CANNM_REPEAT_MESSAGE_TIME}		
Description	Timeout for Repeat Message State. It defines the time in seconds how long the NM shall stay in the Repeat Message State. Typically it should be equal to: n * CanNmMsgCycleTime, where n denotes the number of NM-Messages that are normally sent in the Repeat Message State. The value of n decremented by one determines the amount of lost NM-Messages that can be tolerated by the node detection procedure. The value 0 denotes that no Repeat Message State is configured. 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.		
Multiplicity	1		
Type	FloatParamDef		
Range	0.0 .. 65.535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: Instance dependency: CanNmRepeatMessageTime = n * CanNmMsgCycleTime		

SWS Item	**--**		
Name	CanNmTimeoutTime {CANNM_TIMEOUT_TIME}		
Description	Network Timeout for NM-Messages. It denotes the time in seconds how long the NM shall stay in the Network Mode before transition into Prepare Bus-Sleep Mode shall take place. It shall be equal for all nodes in the cluster. It shall be greater than CanNmMsgCycleTime. Typically it should be equal to: n * CanNmMsgCycleTime, where n denotes the number of NM-Message cycle times in the Ready Sleep State before transition into the Bus-Sleep Mode is initiated. The value of n decremented by one determines the amount of lost NM-Messages that can be tolerated by the coordination algorithm.		
Multiplicity	1		
Type	FloatParamDef		
Range	0.0020 .. 65.535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: Instance dependency: CanNmTimeoutTime = n * CanNmMsgCycleTime		

SWS Item	**--**		
Name	CanNmUserDataLentgh {CANNM_USER_DATA_LENGTH}		
Description	Defines the length of the user data contained in the NM PDU		
Multiplicity	1		
Type	IntegerParamDef		
Range	0 .. 8		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: ECU dependency: CanNmPduLength, CanNmPduNidPosition,		

	CanNmPduCbvPosition

SWS Item	**--**		
Name	CanNmWaitBusSleepTime {CANNM_WAIT_BUS_SLEEP_TIME}		
Description	Timeout for bus calm down phase. It denotes the time in seconds how long the NM shall stay in the Prepare Bus-Sleep Mode before transition into Bus-Sleep Mode shall take place. It shall be equal for all nodes in the cluster. It shall be long enough to make all Tx-buffer empty.		
Multiplicity	1		
Type	FloatParamDef		
Range	0.0010 .. 65.535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: Instance		

SWS Item	**--**		
Name	CanNmChannelIdRef {CANNM_CHANNEL_ID}		
Description	Channel identifier configured for the respective AUTOSAR NM cluster. It is used by referring to the respective NM channel handle. It must be unique for each AUTOSAR NM cluster within one ECU.		
Multiplicity	1		
Type	Reference to NmChannelConfig		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: Instance		

SWS Item	**--**		
Name	CanNmNetworkHandle {CANNM_CHANNEL_HANDLE}		
Description	Channel identifier configured for the respective instance of the NM. The CanNmChannelHandle shall be encoded in the canNmRxPduId parameter which is passed to CanNm_RxIndication() function called by the CanIf.		
Multiplicity	1		
Type	Reference to CanStateManagerNetworks		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Instance		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CanNmRxPdu	1	This container is used to configure the Rx PDU Properties that are used for the CanNm Channel.
CanNmRxPdu	1	This container is used to configure the Rx PDU Properties that are used for the CanNm Channel.
CanNmTxPdu	1	This container contains the CanNmTxPduId and the CanNmTxPduRef.

10.3.6 CanNmRxPdu

SWS Item	**--**

Container Name	CanNmRxPdu
Description	This container is used to configure the Rx PDU Properties that are used for the CanNm Channel.
Configuration Parameters	

SWS Item	**--**		
Name	CanNmRxPduRef		
Description	Reference to the global PDU that is used by this CanNmChannel instance.		
Multiplicity	1		
Type	Reference to Pdu		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency			

No Included Containers

10.3.7 CanNmTxPdu

SWS Item	**--**
Container Name	CanNmTxPdu
Description	This container contains the CanNmTxPduId and the CanNmTxPduRef.
Configuration Parameters	

SWS Item	**--**		
Name	CanNmTxPduRef		
Description	The reference to the common PDU structure.		
Multiplicity	1		
Type	Reference to Pdu		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency			

No Included Containers

10.4 Published parameters

CANNM021: The following table specifies configuration parameters that shall be published in the module's header file.

The standard common published information like

vendorId (<Module>_VENDOR_ID),
moduleId (<Module>_MODULE_ID),
arMajorVersion (<Module>_AR_MAJOR_VERSION),
arMinorVersion (<Module>_ AR_MINOR_VERSION),
arPatchVersion (<Module>_ AR_PATCH_VERSION),
swMajorVersion (<Module>_SW_MAJOR_VERSION),
swMinorVersion (<Module>_ SW_MINOR_VERSION),
swPatchVersion (<Module>_ SW_PATCH_VERSION),

vendorApiInfix (<Module>_VENDOR_API_INFIX)

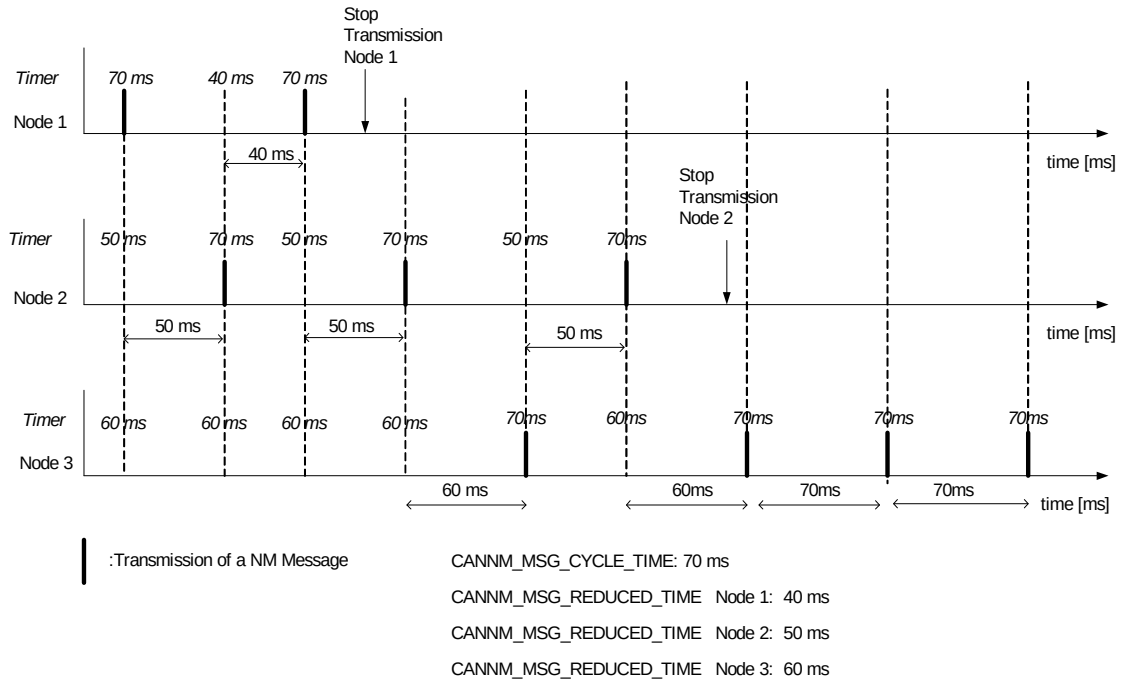
is provided in the BSW Module Description Template (see [18] Figure 4.1 and Figure 7.1).

Additional published parameters are listed below if applicable for this module.

11 Examples

11.1 Example of periodic transmission mode with bus load reduction

Three nodes are connected to the bus and are in “normal operation” state. The nodes (Node 1 and Node 2) with the smallest CANNM_MSG_REDUCED_TIME are sending alternating their NM messages. After a while node 1 goes into “ready sleep” state. Now node 2 and node 3 are sending alternating NM message. After a while also node 2 goes into “ready sleep” state. Since node 3 is the last node on the bus only node 3 is sending messages with CANNM_MSG_CYCLE_TIME.



11.1.1 Example timing □ehaviour for NM PDUs

Assume an example network of three nodes 1, 2, 3 (Figure 11-1). Nodes specific cycle offsets are equal respectively to t1 < t2 < t3 < T. NM cycle time is equal to T (Figure 11-2).
NM PDUs sent on the bus within the Repeat Message State are presented in the Figure 11-3, and within the Normal Operation / Ready Sleep State in Figure 11-4. Each dot in Figure 11-4denotes restart of the NM-Timeout Timer.

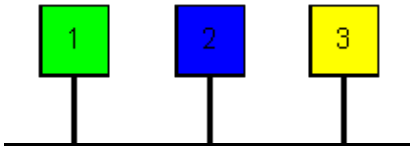


Figure 11-1

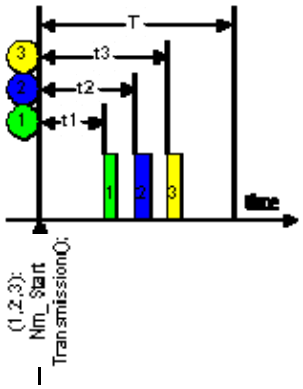


Figure 11-2

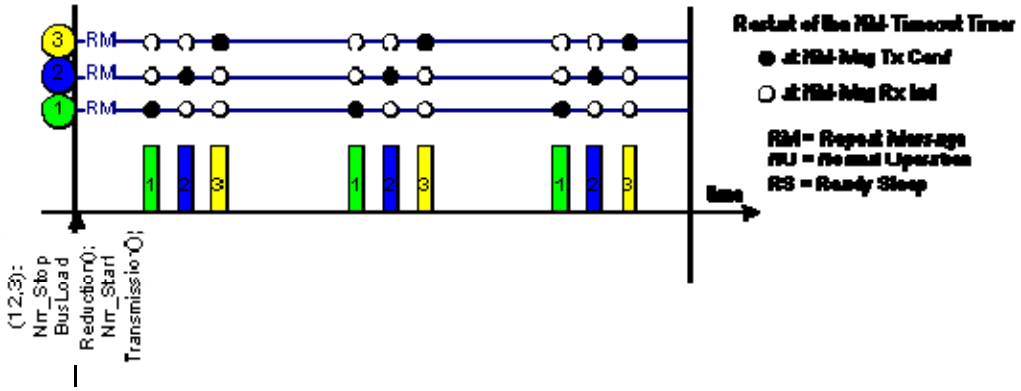


Figure 11-3

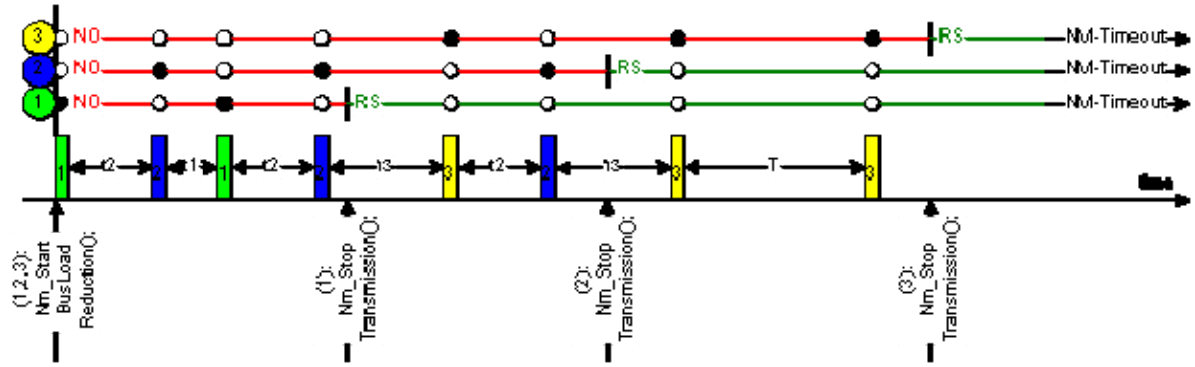


Figure 11-4

12 Changes to Release 2.1

12.1 Deleted SWS Items

None

12.2 Replaced SWS Items

SWS Item of Release 1	*replaced by SWS Item*	*Rationale*
CANNM036	CANNM132	#14202
CANNM002	CANNM157	#14202
CANNM070	CANNM189	#14202
CANNM024	CANNM201	#14202
CANNM038	CANNM147	#14202

12.3 Changed SWS Items

SWS Item	*Rationale*
CANNM034	#14202
CANNM037	#14202
CANNM014	#14165
CANNM061	‘CanNm Timeout Timer’ replaced by ‘NM Message Tx Timeout Timer’
CANNM064	‘CanNm Timeout Timer’ replaced by ‘NM Message Tx Timeout Timer’
CANNM065	‘CanNm Timeout Timer’ replaced by ‘NM Message Tx Timeout Timer’
CANNM066	‘CanNm Timeout Timer’ replaced by ‘NM Message Tx Timeout Timer’ Note removed
CANNM067	‘CanNm Timeout Timer’ replaced by ‘NM Message Tx Timeout Timer’
CANNM068	‘CanNm Timeout Timer’ replaced by ‘NM Message Tx Timeout Timer’
CANNM069	Requirement moved to chapter 7.7
CANNM077	Note removed
CANNM021	Requirement rephrased
CANNM082	#20622
CANNM005	#21564
CANNM032	#21564
CANNM051	#21564
CANNM041	#19745
CANNM005	#28543

12.4 Added SWS Items

SWS Item	*Rationale*
CANNM088..CANNM132	#14202
CANNM133..CANNM181	#14202
CANNM184..CANNM204	#14202
CANNM205	#21145
CANNM206	#19751