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

Document Change History

Date	Version	Changed by	Change Description
31.01.07	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

Release Notes

Errata and known deficiencies

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

Disclaimer

Advice to users of AUTOSAR Specification Documents:

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

Table of Contents

1 Introduction and Functional Overview

The CAN Network Management (CanNm) function provides an adaptation between CAN Generic Network Management (CNm) and the CAN Interface (CanIf) module. For a general understanding of the AUTOSAR Network Management functionality please read [4] and [10].

The CanNm covers the following functionalities:

- Periodic transmission and reception of CAN NM PDUs
- Encoding and decoding of CAN NM PDUs
- Transmission Error Handling
- Notification of the CAN Generic Network Management regarding transmission and reception of CAN NM PDUs

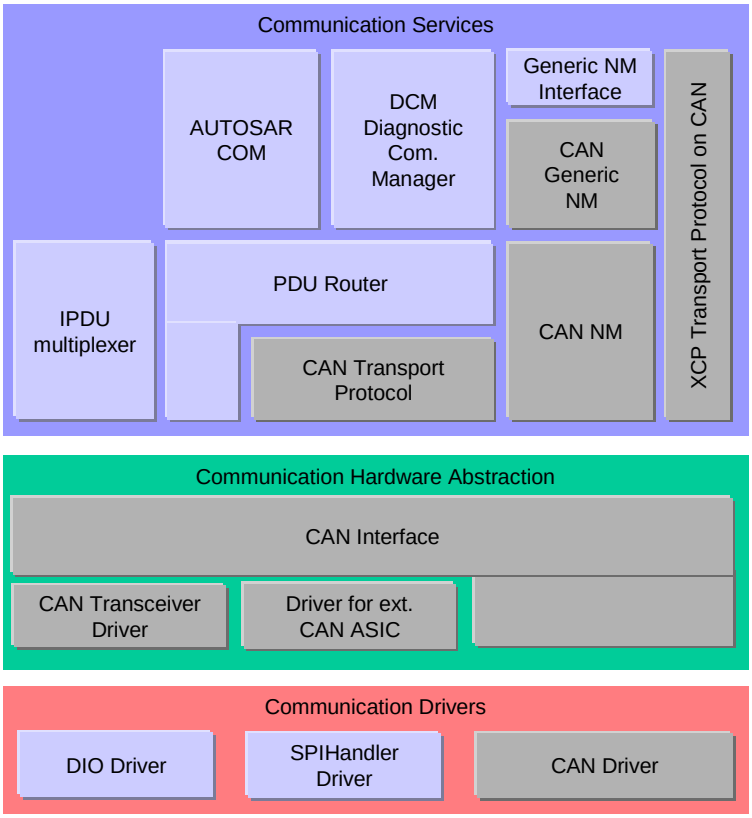


Figure 1: Software Architecture CAN Communication Stack

2 Acronyms and abbreviations

Acronym:	*Description:*
API	Application Programming Interface
NM	Network Management

Abbreviation:	*Description:*
CanNm	Abbreviation for the CAN specific Network Management
BusNm	Generic name for any kind of the bus specific NM (i.e. CanNm, FrNm)
PDU	Protocol Data Unit
CNm	Abbreviation for the CAN Generic Network Management
FrNm	Abbreviation for the FlexRay Network Management
CanIf	Abbreviation for the Can Driver Interface
BSW	Basic Software
SDU	Service Data Unit
DET	Development Error Tracer
DEM	Diagnostics Event Manager

3 Related documentation

3.1 Input documents

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

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

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

[4] AUTOSAR Specification of CAN Generic Network Management
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_Generic_NM.pdf

[5] AUTOSAR Specification of CAN Interface
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_CAN_Interface.pdf

[6] AUTOSAR Specification of FlexRay Network Management
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_FlexRay_NM.pdf

[7] AUTOSAR Specification of Communication Stack Types
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_ComStackTypes.pdf

[8] AUTOSAR Specification of ECU Configuration, WP 4.1.1.2 deliverable, Version 0.02 https://svn.autosar.org/repos/10Releases/
AUTOSAR_ECU_Configuration.pdf

[9] AUTOSAR Specification of BSW Scheduler, WP 1.1.2 deliverable, Version 0.14
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_BSW_Scheduler.pdf

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

3.2 Related standards and norms

Not available.

4 Constraints and assumptions

4.1 Limitations

This module is only applicable for CAN systems.

4.2 Applicability to car domains

The CanNm is applicable to any Car Domain.

5 Dependencies to other modules

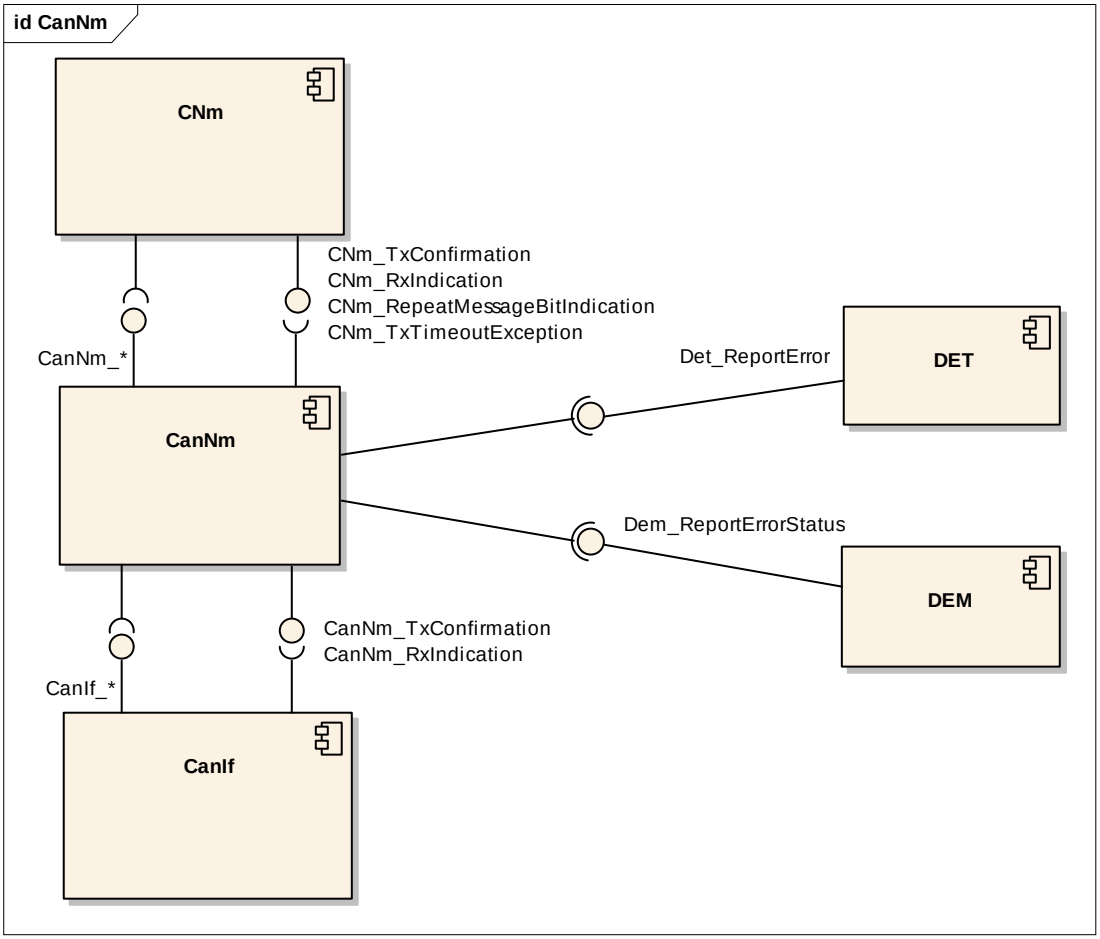


Figure 2 Dependencies to other modules

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)
- `CNm_Cbk.h` (for callback functions declaration of CNm)
- `Det.h` (for interface of DET – optional, included only if Det is configured)
- `Dem.h` (for interface of DEM)
- `Nm.h` (for common NM types)
- `SchM_CanNm.h` (for services of the Basic Software Scheduler)

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

- `Det_Cfg.h` (for configuration of DET – optional, included only if DET is configured)
- `Dem_Cfg.h` (for configuration of DEM)
- `CanIf_Cfg.h` (for the CAN PDU Ids)
- `CNm_Cfg.h` (for the derived configuration items from the CNm)

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 2
[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 10.3
[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 8.3.12
[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 7.8
[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 7.8
[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 7.8
[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 7.7
[BSW00338] Detection and Reporting of development errors	Ok; see chapter 7.7 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 7.7
[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; see 8.3.12
[BSW00409] Header files for production code error IDs	Ok; see CANNM082
[BSW00385] List possible error notifications	Ok; see 7.7
[BSW00386] Configuration for detecting an error	Ok; see chapter 10.2.3 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	n/a

drivers	(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 7.7
[BSW00335] Status values naming convention	n/a (no status values exported)
[BSW00350] Development error detection keyword	Ok; see chapter 10.2.3
[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 8.3.12
[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 configuration parameters	Ok; see chapter 10.2
[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 10.3
[BSW00379] Module identification	Ok; see chapter 10.3
[BSW003] Version identification	Ok; see chapter 10.3
[BSW00318] Format of module version numbers	Ok; see chapter 10.3
[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 8.3.2 and 8.3.3
[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 8.3.6
[BSW02504] Receiving user data	Ok; see chapter 8.3.7
[BSW153] Detection of present nodes	n/a (not in the responsibility of CanNm)
[BSW02508] Unambiguous node identification per bus	Ok; see chapter 8.3.8
[BSW02505] Sending node identifier	Ok; see chapter 7.5
[BSW02506] Receiving node identifier	Ok; see chapter 8.3.8
[BSW02511] Configurable Role in Cluster Shutdown	Ok; see 10.2.3
[BSW053] Deterministic Behavior in Case of Bus Unavailability	Ok; see chapter 7.4
[BSW137] Communication system error handling	Ok; see chapter 7.4
[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 8.3.4 and 8.3.5
[BSW143] Predictable NM bus load	Ok; see chapter 7.2.1
[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 10.2.3
[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

The CanNm covers the following functionalities:

- Periodic transmission and reception of CAN NM PDUs
- Encoding and decoding of CAN NM PDUs
- Transmission Error Handling
- Notification of the CAN Generic Network Management regarding transmission and reception of CAN NM PDUs

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

7.1 Initialization

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

Note:
`CanNm_Init` is called by `CNm_Init` (see [4]).

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

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

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

Note:
The CanNm 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 the user data bytes shall be set to `0xFF`.

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

7.2 Communication Scheduling

7.2.1 Transmission

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.

CANNM024: The transmission shall be started by the function `CanNm_StartTransmission` and stopped by the function `CanNm_StopTransmission`.

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 by the CNm 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, i.e. activated, by the CNm 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 to inform the CNm about a successful transmission 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: If start of transmission is requested the CanNm Message Cycle Timer shall be started with `CANNM_MSG_CYCLE_OFFSET`.

Note: This mechanism prevents bursts of NM messages.

CANNM032: If transmission is requested 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 CNm is informed by the Transmission Error handling described in chapter 7.4.

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.

CANNM036: If a NM message has been successfully transmitted the CNm function `CNm_TxConfirmation` shall be called (triggered by the function `CanNm_TxConfirmation`).

Note:
Requirement CANNM034 and CANNM036 are only valid if the simplified transmission confirmation mechanism is disabled. If it is enabled CANNM070 has to be considered instead.

CANNM070: If `CANNM_IMMEDIATE_TXCONF_ENABLED` is enabled the CNm function `CNm_TxConfirmation` shall be immediately called after a transmission of a NM message is requested (by means of the CanIf function `CanIf_Transmit`).

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

CANNM002: If the bus load reduction mechanism is enabled, the transmission mode shall be dynamically selectable during runtime by the functions `CanNm_StartBusLoadReduction` and `CanNm_StopBusLoadReduction`.

CANNM051: If stop of transmission is requested the CanNm Message Cycle Timer shall be canceled.

7.2.2 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 CNm function `CNm_RxIndication` shall be called (triggered by the function `CanNm_RxIndication`).

CANNM038: If a NM message has been successfully received with the RepeatMessageRequest bit (see 7.5) set the CNm function `CNm_RepeatMessageBitIndication` shall be called (triggered by `CanNm_RxIndication`) unless `CANNM_PDU_CBV_POSITION` is set to off.

Note:
CANNM038 is also valid if bus load reduction is active. The reaction of the CNm when `CNm_RepeatMessageIndication` is called is described in [4].

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`.

7.3 Bus Load Reduction Mechanism

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` has to be 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.

See also example in chapter 11.

7.4 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 CanNm Timeout Timer shall be started with `CANNM_MSG_TIMEOUT_TIME` when `CanNm_StartTransmission` is called.

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

CANNM066: The function `CNm_TxTimeoutException` provided by the CNm shall be called only once when the CanNm Timeout Timer expires.

Note:
It shall be avoided that `CNm_TxTimeoutException` is called periodically when the CanNm Timeout Timer has expired.

CANNM067: The CanNm 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 CanNm Timeout Timer shall be stopped when `CanNm_StopTransmission` is called.

7.5 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 3 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 4 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

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

7.6 Functional requirements on CanNm API

CANNM014: If the node detection functionality is enabled, the RepeatMessageRequest bit shall be set and cleared by the function `CanNm_SetRepeatMessageBit` (see also 8.3.10).

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.

7.7 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

CANNM019: Development errors shall be reported to the DET.

CANNM020: Production errors shall be reported to the DEM.

7.8 Scheduling of the main function

CANNM077: The CanNm_MainFunction_<Instance Id> functions shall be scheduled by the BSW scheduler (see [9]).

Note:
The order how CanNm_MainFunction and CNm_MainFunction are called shall be functional irrelevant.

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**.**

8.1 Imported Types

- The data type `PduIdType` is defined in `ComStack_Types.h`
- The data type `Std_ReturnType` and `Std_VersionInfoType` are defined in `Std_Types.h`
- `Nm_ChannelHandleType` is defined in `Nm.h`

8.2 Type Definitions

8.2.1 CanNm_PduPositionType

Type:	`enum CanNm_PduPositionType`	
Range:	`CANNM_PDU_BYTE_0 = 0x00`	PDU field is located at Byte 0
	`CANNM_PDU_BYTE_1 = 0x01`	PDU field is located at Byte 1
	`CANNM_PDU_OFF = 0xff`	PDU field is not used.
Description:	Type for configuring the PDU.	

8.2.2 CanNm_ConfigType

This type shall contain the parameters of the container CanNm_GlobalConfig and its sub containers (see 10.2.4).

8.3 CanNm Functions called by the CNm

8.3.1 CanNm_Init

Service name:	CanNm_Init	
Syntax:	`void CanNm_Init ( const CanNm_ConfigType * const nmConfigPtr )`	
Service ID:	0x01	
Sync/Async:	Sync	
Reentrancy:	No	
Parameters (in):	`nmConfigPtr`	Pointer to a selected configuration structure
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_StartTransmission

Service name:	CanNm_StartTransmission	

Syntax:	`Std_ReturnType` **`CanNm_StartTransmission`**
	`(`
	`  const Nm_ChannelHandleType nmChannelHandle`
	`)`	

Service ID:	`0x02`	
Sync/Async:	Sync	
Reentrancy:	Yes (but not within the same channel)	
Parameters (in):	`nmChannelHandle`	Identification of the physical channel
Parameters (out):	`None`	--
Return value:	`E_OK`	Function call was successful
	`E_NOT_OK`	Function call failed
Description:	Start transmission of NM messages 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 `nmChannelHandle` shall be used as the instance id.	
Caveats:	The CanIf and the CanNm modules are initialized correctly.	
Configuration:	Optional (only available if `CANNM_PASSIVE_MODE_ENABLED` is disabled)	

8.3.3 CanNm_StopTransmission

Service name:	CanNm_StopTransmission	

Syntax:	`Std_ReturnType` **`CanNm_StopTransmission`**
	`(`
	`  const Nm_ChannelHandleType nmChannelHandle`
	`)`	

Service ID:	0x03	
Sync/Async:	Sync	
Reentrancy:	Yes (but not within the same channel)	
Parameters (in):	`nmChannelHandle`	Identification of the physical channel
	--	--
	--	--

Parameters (out):	None	-
Return value:	E_OK	Function call was successful
	E_NOT_OK	Function call failed
Description:	Stops transmission of NM messages 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 nmChannelHandle shall be used as the instance id.	
Caveats:	The CanIf and the CanNm modules are initialized correctly.	
Configuration:	Optional (only available if CANNM_PASSIVE_MODE_ENABLED is disabled)	

8.3.4 CanNm_StartBusLoadReduction

Service name:	CanNm_StartBusLoadReduction	
Syntax:	Std_ReturnType **CanNm_StartBusLoadReduction** (const Nm_ChannelHandleType nmChannelHandle)	
Service ID:	0x04	
Sync/Async:	Sync	
Reentrancy:	Yes (but not within the same channel)	
Parameters (in):	nmChannelHandle	Identification of the physical channel
Parameters (out):	None	--
Return value:	E_OK	Function call was successful
	E_NOT_OK	Function call failed
Description:	Activates the CanNm bus load reduction mode. 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 nmChannelHandle shall be used as the instance id.	
Caveats:	The CanIf and the CanNm modules are initialized correctly. The busload reduction mechanism is statically enabled. The call of the API does not start the transmission of messages.	
Configuration:	Optional (only available if CANNM_BUS_LOAD_REDUCTION_ENABLED is enabled)	

8.3.5 CanNm_StopBusLoadReduction

Service name:	CanNm_StopBusLoadReduction
Syntax:	Std_ReturnType **CanNm_StopBusLoadReduction** (const Nm_ChannelHandleType nmChannelHandle)
Service ID:	0x05
Sync/Async:	Sync

Reentrancy:	Yes (but not within the same channel)	
Parameters (in):	`nmChannelHandle`	Identification of the physical channel
Parameters (out):	`None`	--
Return value:	`E_OK`	Function call was successful
	`E_NOT_OK`	Function call failed
Description:	Deactivates the CanNm bus load reduction mode. 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 `nmChannelHandle` shall be used as the instance id.	
Caveats:	The CanIf and the CanNm modules are initialized correctly. The busload reduction mechanism is enabled. The function does not stop the transmission of the NM messages.	
Configuration:	Optional (only available if `CANNM_BUS_LOAD_REDUCTION_ENABLED` is enabled)	

8.3.6 CanNm_SetUserData

Service name:	CanNm_SetUserData	
Syntax:	`Std_ReturnType CanNm_SetUserData ( const Nm_ChannelHandleType nmChannelHandle, const uint8 * nmUserDataPtr )`	
Service ID:	0x06	
Sync/Async:	Sync	
Reentrancy:	No	
Parameters (in):	`nmChannelHandle`	Identification of the physical channel
	`nmUserDataPtr`	Pointer to user data which is transmitted in the next sent NM message
Parameters (out):	`None`	--
Return value:	`E_OK`	Function call was successful
	`E_NOT_OK`	Function call failed
Description:	Set user data transmitted for the next sent NM message. 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 `nmChannelHandle` shall be used as the instance id.	
Caveats:	The CanIf and the CanNm modules are initialized correctly.	
Configuration:	Optional (only available if `CANNM_USER_DATA_ENABLED` is enabled and `CANNM_PASSIVE_MODE_ENABLED` is disabled)	

8.3.7 CanNm_GetUserData

Service name:	CanNm_GetUserData

Syntax:	`Std_ReturnType` **`CanNm_GetUserData`**
	`(`
	`  const Nm_ChannelHandleType nmChannelHandle,`
	`  uint8 * nmUserDataPtr`
	`)`	

Service ID:	0x07	
Sync/Async:	Sync	
Reentrancy:	Yes	
Parameters (in):	`nmChannelHandle`	Identification of the physical channel
Parameters (out):	`nmUserDataPtr`	Pointer where user data out of the last sucessfully received NM message shall be copied to
Return value:	`E_OK`	Function call was successful
	`E_NOT_OK`	Function call failed
Description:	Get user data out of the last successfully received NM message. 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 `nmChannelHandle` shall be used as the instance id.	
Caveats:	The CanIf and the CanNm modules are initialized correctly. Any NM Message previously received.	
Configuration:	Optional (only available if `CANNM_USER_DATA_ENABLED` is enabled)	

8.3.8 CanNm_GetNodeIdentifier

Service name:	CanNm_GetNodeIdentifier	

Syntax:	`Std_ReturnType` **`CanNm_GetNodeIdentifier`**
	`(`
	`  const Nm_ChannelHandleType nmChannelHandle,`
	`  uint8 * nmNodeIdPtr`
	`)`	

Service ID:	0x08	
Sync/Async:	Sync	
Reentrancy:	Yes	
Parameters (in):	`nmChannelHandle`	Identification of the physical channel
	--	--
Parameters (out):	`nmNodeIdPtr`	Pointer where node identifier shall be copied to
Return value:	`E_OK`	Function call was successful
	`E_NOT_OK`	Function call failed
Description:	Get the node identifier of the most recently received NM message.	
Caveats:	The CanIf and the CanNm modules are initialized correctly. 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 `nmChannelHandle` shall be used as the instance id.	
Configuration:	Optional (only available if `CANNM_PDU_NID_POSITION` is not set to off or	

	CANNM_NODE_DETECTION_ENABLED is enabled)

8.3.9 CanNm_GetLocalNodeIdentifier

Service name:	CanNm_GetLocalNodeIdentifier	
Syntax:	`Std_ReturnType CanNm_GetLocalNodeIdentifier ( const Nm_ChannelHandleType nmChannelHandle, uint8 * nmNodeIdPtr )`	
Service ID:	0x09	
Sync/Async:	Sync	
Reentrancy:	Yes	
Parameters (in):	nmChannelHandle	Identification of the physical channel
Parameters (out):	nmNodeIdPtr	Pointer where node identifier of the local node shall be copied to
Return value:	E_OK	Function call was successful
	E_NOT_OK	Function call failed
Description:	Get Node Identifier of the local node. 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 nmChannelHandle shall be used as the instance id.	
Caveats:	The CanIf and the CanNm modules are initialized correctly.	
Configuration:	Optional (only available if CANNM_PDU_NID_POSITION is not set to off or CANNM_NODE_DETECTION_ENABLED is enabled)	

8.3.10 CanNm_SetRepeatMessageBit

Service name:	CanNm_SetRepeatMessageBit	
Syntax:	`Std_ReturnType CanNm_SetRepeatMessageBit ( const Nm_ChannelHandleType nmChannelHandle, const boolean nmRepeatMessageBit )`	
Service ID:	0x0a	
Sync/Async:	Sync	
Reentrancy:	No	
Parameters (in):	nmChannelHandle	Identification of the physical channel
	nmRepeatMessageBit	Repeat Message Bit value
Parameters (out):	None	--
Return value:	E_OK	Function call was successful
	E_NOT_OK	Function call failed
Description:	Set Repeat Message Bit	
Caveats:	The CanIf and the CanNm modules are initialized correctly. 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 `nmChannelHandle` shall be used as the instance id.
Configuration:	Optional (only available if `CANNM_NODE_DETECT_ENABLED` is enabled)

8.3.11 CanNm_TriggerTransmission

Service name:	CanNm_TriggerTransmission	
Syntax:	`Std_ReturnType CanNm_TriggerTransmission ( const Nm_ChannelHandleType nmChannelHandle )`	
Service ID:	0x0e	
Sync/Async:	Sync	
Reentrancy:	No	
Parameters (in):	`nmChannelHandle`	Identification of the physical channel
Parameters (out):	`None`	--
Return value:	`E_OK`	Function call was successful
	`E_NOT_OK`	Function call failed
Description:	Trigger an immediate transmission of a NM-message independently of the normal cyclic transmission. Timeout handling shall be done as described in 7.4. 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) Inform the DEM if the call of the CanIf function CanIf_Transmit has failed (CANNM_E_CANIF_TRANSMIT_ERROR) If an error has to be indicated to the DET the value of `nmChannelHandle` shall be used as the instance id.	
Caveats:	The CanIf and the CanNm modules are initialized correctly.	
Configuration:	Optional (only available if `CANNM_PASSIVE_MODE_ENABLED` is disabled)	

8.3.12 CanNm_GetVersionInfo

Service name:	CanNm_ GetVersionInfo	
Syntax:	`void CanNm_GetVersionInfo ( Std_VersionInfoType *versionInfo )`	
Service ID:	0x14	
Sync/Async:	Synchronous	
Reentrancy:	Yes	
Parameters (in):	`none`	--
Parameters (out):	`versionInfo`	Pointer to where to store the version information of this module.

Return value:	none	--
Description:	This service returns the version information of this module. The version information includes: - Module Id - Vendor Id - Vendor specific version numbers (BSW00407). Hint: If source code for caller and callee of this function is available this function should be realized as a macro. The macro should be defined in the modules header file. Note: This function can be called even if the CanNm is not initialized.	
Caveats:	--	
Configuration:	Optional (only available if `CANNM_VERSION_INFO_API` is defined)	

8.3.13 CanNm_GetPduData

Service name:	CanNm_GetPduData	

Syntax:	`Std_ReturnType` **`CanNm_GetPduData`**
	`(`
	`const Nm_ChannelHandleType nmChannelHandle,`
	`uint8 * nmPduData`
	`)`	

Service ID:	0x15	
Sync/Async:	Synchronous	
Reentrancy:	No	
Parameters (in):	`nmChannelHandle`	Identification of the physical channel
Parameters (out):	`nmPduData`	Pointer where the PDU data shall be copied to
Return value:	`E_OK`	Function call was successful
	`E_NOT_OK`	Function call failed
Description:	Get the whole PDU data out of the most recently received NM message.	
Caveats:	The CanIf and the CanNm modules are initialized correctly.	
Configuration:	Optional (only available if `CANNM_NODE_DETECTION_ENABLED` is enabled and `CANNM_PDU_LENGTH > 0`)	

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:	0x0f	
Sync/Async:	Sync	
Reentrancy:	Yes (but not within the same channel)	
Parameters (in):	`canNmTxPduId`	Identification of the network through PDU-ID
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_PASSIVE_MODE_ENABLED is disabled and CANNM_IMMEDIATE_TXCONF_ENABLED is disabled)	

8.4.2 CanNm_RxIndication

Service name:	CanNm_RxIndication	
Syntax:	void **CanNm_RxIndication** (PduIdType canNmRxPduId, const unit8 *canSduPtr)	
Service ID:	0x10	
Sync/Async:	Sync	
Reentrancy:	Yes (but not within the same channel)	
Parameters (in):	canNmRxPduId	Identification of the network through PDU-ID
	canSduPtr	Pointer to received SDU
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 canSduPtr. The `canSduPtr` 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>`
	`(`
	`  void`
	`)`

Service ID:	0x13
Sync/Async:	Synchronous
Reentrancy:	Non-Reentrant
Parameters (in):	None --
Parameters (out):	None --
Return value:	None --
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.

API function	*Module*	*Description*
CNm_RxIndication	CNm	Service to indicate that a NM message has been successfully received.
Dem_ReportErrorStatus	Dem	Service to indicate a production error

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)*
Det_ReportError	Det	Development error notification	CANNM_DEV_ERROR_DETECT
CanIf_Transmit	CanIf	Service to request a transmission of a CAN message	CANNM_PASSIVE_MODE_ENABLED
CNm_TxConfirmation	CNm	Service to indicate that a NM message has been successfully transmitted.	CANNM_PASSIVE_MODE_ENABLED CANNM_IMMEDIATE_TXCONF_ENABLED
CNm_RepeatMessageBitIndication	CNm	Service to indicate that a NM message with set Repeat Message Request Bit has been received.	CANNM_PDU_CBV_POSITION
CNm_TxTimeoutException	CNm	Service to indicate that the CanNm Timeout Timer has been expired.	CANNM_PASSIVE_MODE_ENABLED CANNM_IMMEDIATE_TXCONF_ENABLED

8.6.3 Configurable interfaces

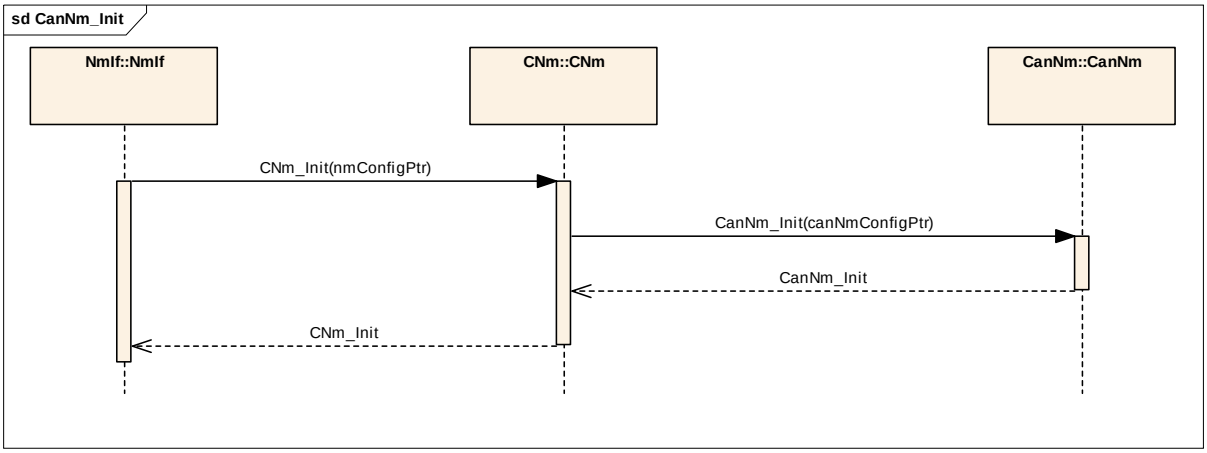
Not applicable

8.6.4 Job End Notification

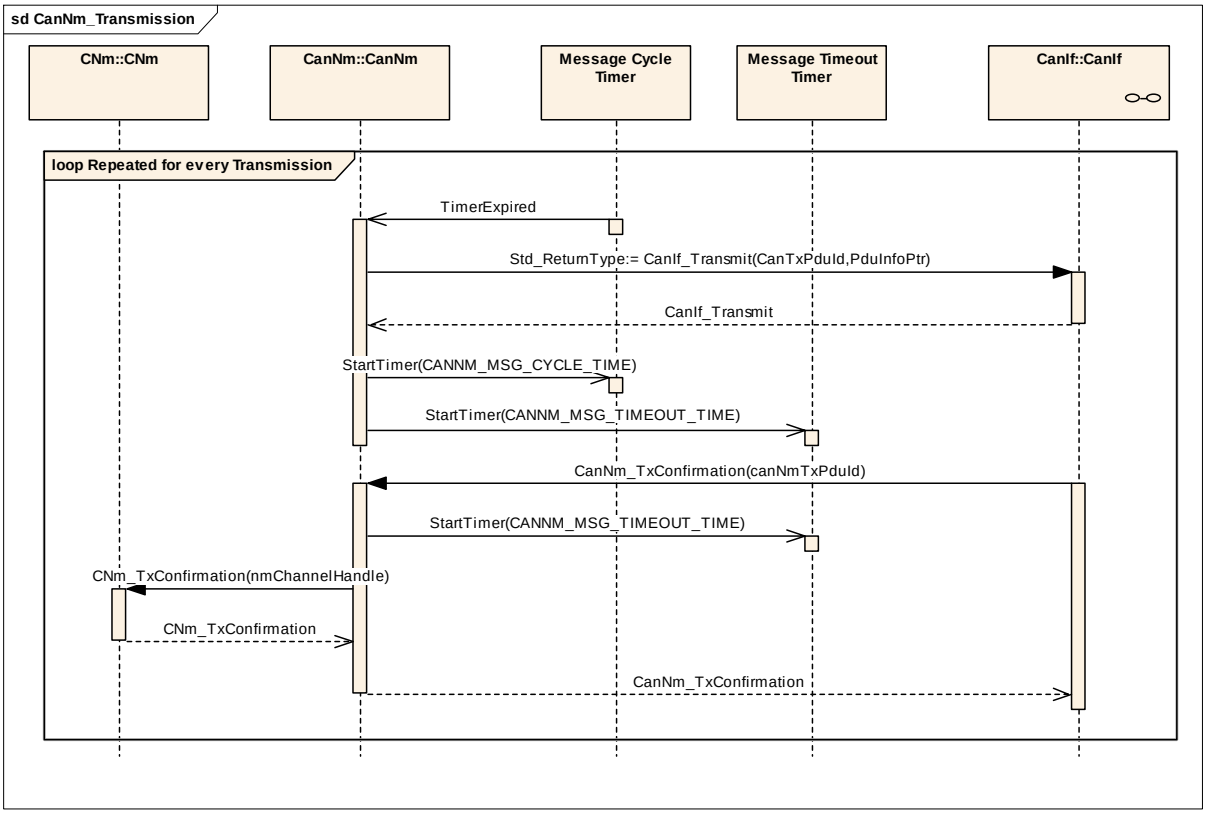
Not applicable

9 Sequence diagrams

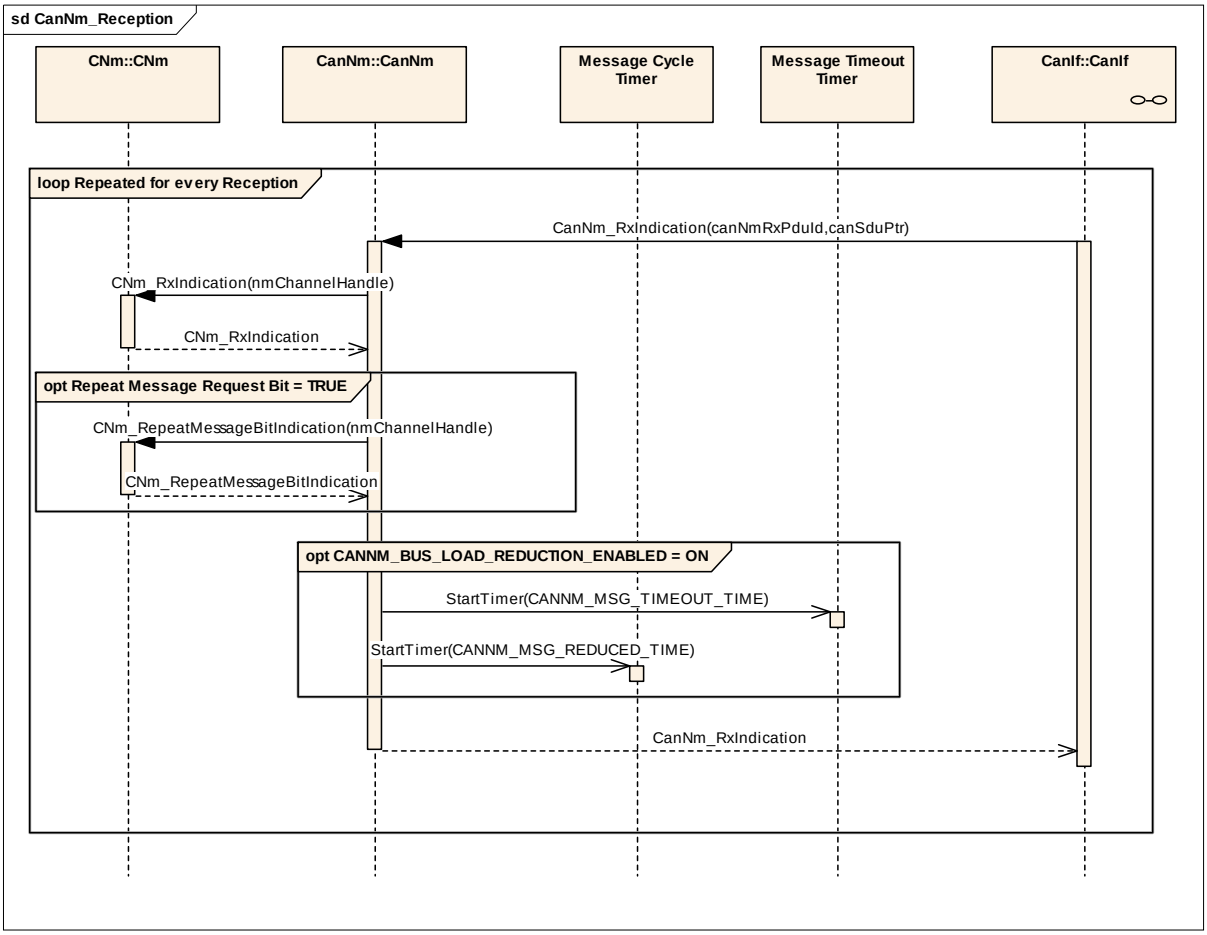
9.1 CanNm Initialization



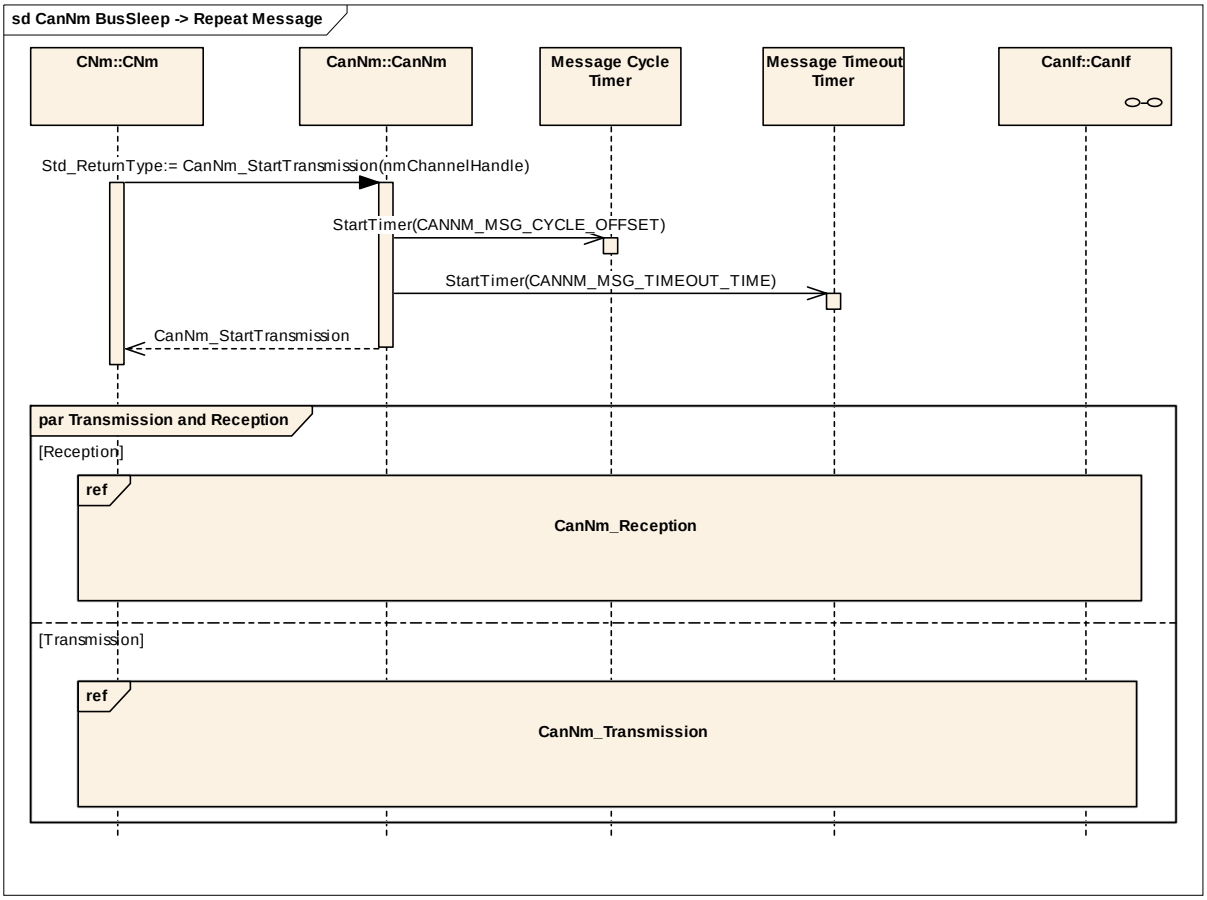
9.2 CanNm Transmission



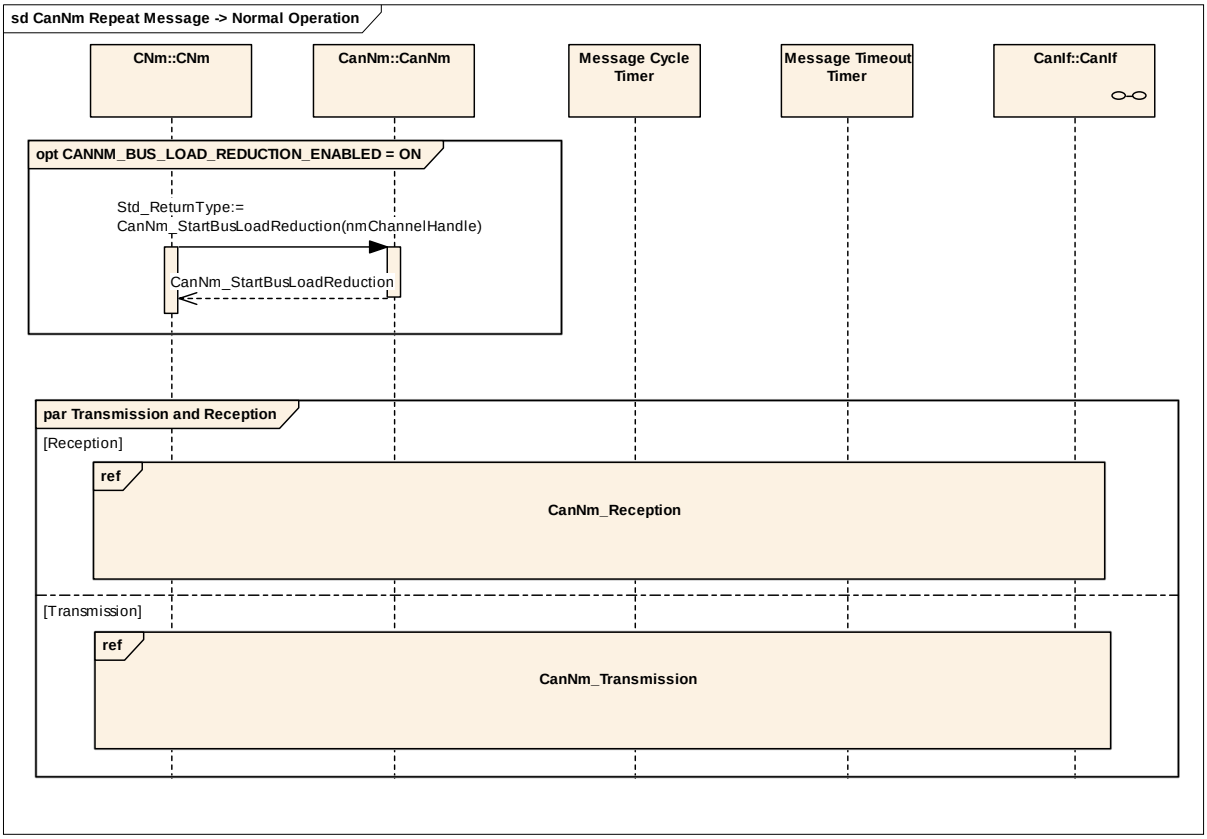
9.3 CanNm Reception



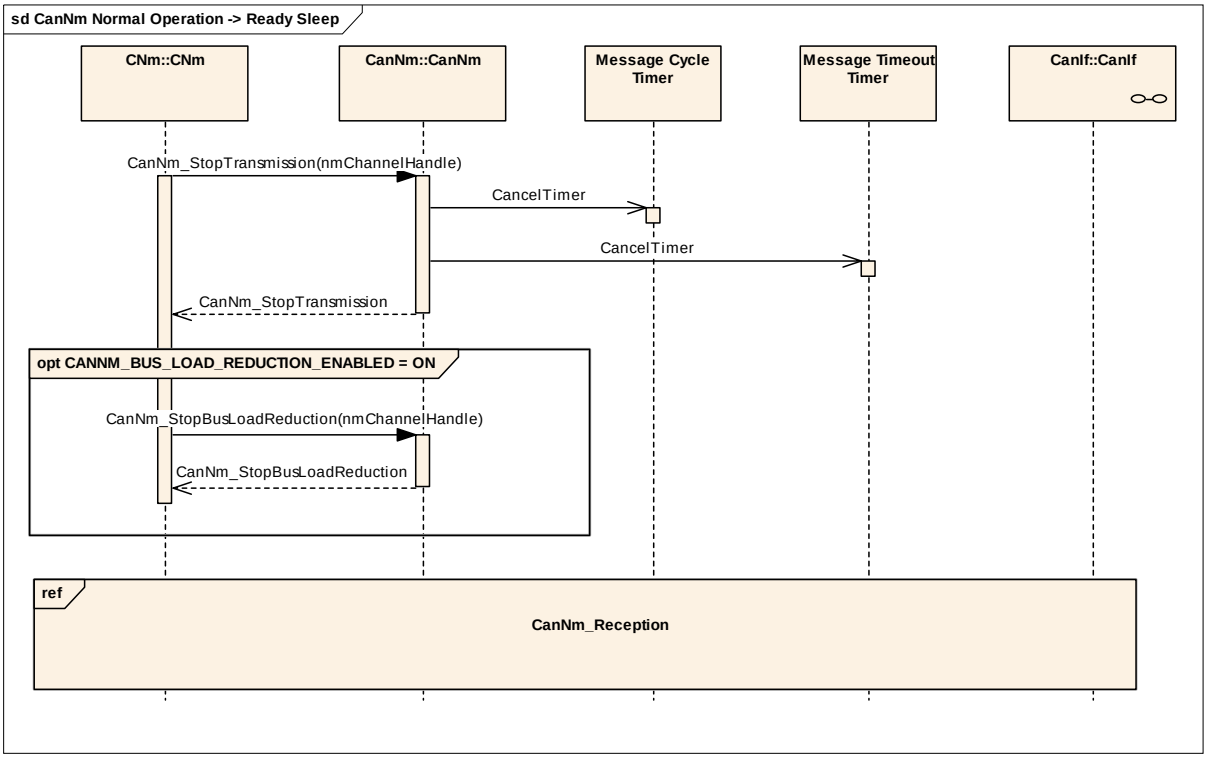
9.4 Transition from Bus Sleep to Repeat Message State



9.5 Transition from Repeat Message to Normal Operation State



9.6 Transition from Normal Operation to Ready Sleep State



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 [8]
  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 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.

CANNM084: Compiler switches (if used in the implementation) shall be compared with defined values.

Rationale:
Simple checks if a compiler switch is defined shall not be used.

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 containing in the container `CanNm_Featureconfig` and the `CANNM_NUMBER_OF_CHANNELS` parameter shall be configurable at pre-compile time. The remaining parameters of `CanNm_GlobalConfig` and `CanNm_ChannelConfig` shall be configurable at link time.
Use case: Object code libraries

Variant 3: The parameters contained in `CanNm_FeatureConfig` and the `CANNM_NUMBER_OF_CHANNELS` parameter is configurable at pre-compile time. The remaining parameters contained in `CanNm_GlobalConfig` and `CanNm_ChannelConfig` are configurable at post-build time.
Use case: ECU configuration can be flashed (L) or the valid configuration is selected during initialization (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.

10.2.2 Configuration Overview

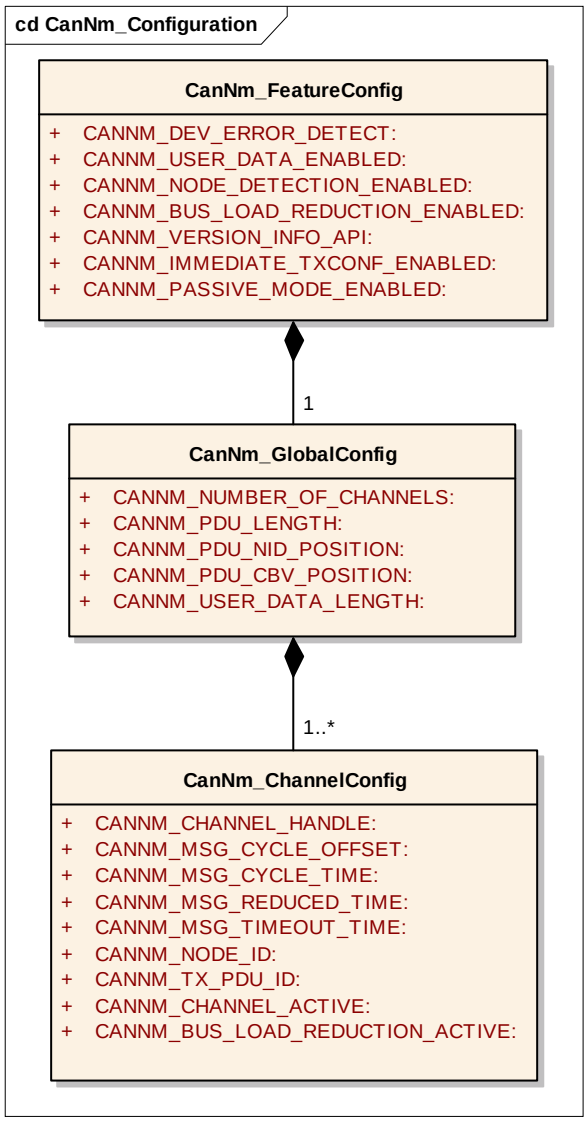


Figure 5: CanNm configuration containers

10.2.3 CanNm_FeatureConfig

SWS Item	--
Container Name	**`CanNm_FeatureConfig`**
Description	This container contains parameter which enables or disables optional features of the CanNm.
Configuration Parameters	

Name	**`CANNM_DEV_ERROR_DETECT`**	
Description	Enable/disable the development error detection.	
Type	`Boolean`	
Unit	--	
Range	TRUE, FALSE	TRUE means enabled

		FALSE means disabled	
	--	--	
Configuration Class	***Pre-compile***	x	All variants
	Link time	--	
	Post Build	--	
Scope	ECU		
Dependency	--		

Name	**CANNM_NODE_DETECTION_ENABLED**		
Description	Enable/disable the node detection functionality.		
Type	Boolean		
Unit	--		
Range	TRUE, FALSE	TRUE means enabled FALSE means disabled	
	--	--	
Configuration Class	***Pre-compile***	x	All variants
	Link time	--	--
	Post Build	--	
Scope	ECU		
Dependency	• This parameter should be derived from NM_NODE_DETECTION_ENABLED. • Enabling of this parameter makes only sense if CANNM_PASSIVE_MODE_ENABLED is disabled		

Name	**CANNM_USER_DATA_ENABLED**		
Description	Enable/disable the user data support.		
Type	Boolean		
Unit	--		
Range	TRUE, FALSE	TRUE means enabled FALSE means disabled	
	--	--	
Configuration Class	***Pre-compile***	x	All variants
	Link time	--	--
	Post Build	--	--
Scope	ECU		
Dependency	This parameter should be derived from NM_USER_DATA_ENABLED.		

Name	**CANNM_BUS_LOAD_REDUCTION_ENABLED**		
Description	Enable/disable the bus load reduction mechanism		
Type	Boolean		
Unit	--		
Range	TRUE, FALSE	TRUE means enabled FALSE means disabled	
	--	--	
Configuration Class	***Pre-compile***	x	All variants
	Link time	--	--
	Post Build	--	--
Scope	ECU		
Dependency	• This parameter should be derived from NM_BUS_LOAD_REDUCTION_ENABLED. • Enabling of this parameter makes only sense if CANNM_PASSIVE_MODE_ENABLED is disabled		

Name	**CANNM_VERSION_INFO_API**	
Description	Enable/disable the function **CanNm_GetVersionInfo** (see 8.3.12)	
Type	Boolean	
Unit	--	
Range	TRUE, FALSE	TRUE means enabled

		FALSE means disabled	
	--	--	
Configuration Class	***Pre-compile***	x	All variants
	Link time	--	--
	Post Build	--	--
Scope	ECU		
Dependency	None		

Name	**CANNM_IMMEDIATE_TXCONF_ENABLED**		
Description	Enable/disable the immediate tx confirmation, i.e. transmission confirmation to CNm is raised at the transmission request.		
Type	Boolean		
Unit	--		
Range	TRUE, FALSE	TRUE means enabled FALSE means disabled	
	--	--	
Configuration Class	***Pre-compile***	X	All variants
	Link time	--	--
	Post Build	--	--
Scope	ECU		
Dependency	Enabling of this parameter makes only sense if CANNM_PASSIVE_MODE_ENABLED is disabled		

Name	**CANNM_PASSIVE_MODE_ENABLED**		
Description	Enable/disable the passive mode, i.e. nodes which have this switch enabled must not send NM messages, but they shall receive and process NM messages.		
Type	Boolean		
Unit	--		
Range	TRUE, FALSE	TRUE means enabled FALSE means disabled	
	--	--	
Configuration Class	***Pre-compile***	X	All variants
	Link time	--	--
	Post Build	--	--
Scope	ECU		
Dependency	This parameter shall be derived from NM_PASSIVE_MODE_ENABLED defined in the NM Interface.		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CanNm_GlobalConfig	1	Instance / --

10.2.4 CanNm_GlobalConfig

SWS Item	--
Container Name	**`CanNm_GlobalConfig`**
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	

Name	**`CANNM_NUMBER_OF_CHANNELS`**		
Description	Number of CanNm channels		
Type	`const uint8`		
Unit	--		
Range	1..0xff	--	
	--	--	
Configuration Class	***Pre-compile***	x	All variants
	Link time	--	--
	Post Build	--	--
Scope	ECU		
Dependency	None		

Name	**`CANNM_PDU_LENGTH`**		
Description	Defines the length of the NM PDU.		
Type	`const uint8`		
Unit	--		
Range	*0..0x08*	--	
	--	--	
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	x	Variant 2
	Post Build	M,L	Variant 3
Scope	ECU		
Dependency	Valid values are within the range 0 <= CANNM_PDU_LENGTH <=8.		

Name	**`CANNM_PDU_NID_POSITION`**		
Description	Defines the position of the source node identifier within the NM PDU		
Type	`const CanNm_PduPositionType`		
Unit	--		
Range	`CANNM_PDU_BYTE_0`, `CANNM_PDU_BYTE_1`, `CANNM_PDU_OFF`	--	
	--	--	
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	x	Variant 2
	Post Build	M,L	Variant 3
Scope	ECU		
Dependency	• The value of the parameter represents the location of the source node identifier in the NM PDU (`CANNM_PDU_BYTE_0` means byte 0, `CANNM_PDU_BYTE_1` means byte 1, `CANNM_PDU_OFF` means source node identifier is not part of the NM PDU) • see also `CANNM_PDU_CBV_POSITION`		

Name	**CANNM_PDU_CBV_POSITION**		
Description	Defines the position of the control bit vector within the NM PDU		
Type	const CanNm_PduPositionType		
Unit	--		
Range	CANNM_PDU_BYTE_0, CANNM_PDU_BYTE_1, CANNM_PDU_OFF	--	
	--	--	
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	x	Variant 2
	Post Build	M,L	Variant 3
Scope	ECU		
Dependency	• The value of the parameter represents the location of the control bit vector in the NM PDU (CANNM_PDU_BYTE_0 means byte 0, CANNM_PDU_BYTE_1 means byte 1, CANNM_PDU_OFF means source node identifier is not part of the NM PDU) • see also CANNM_PDU_NID_POSITION		

Name	**CANNM_USER_DATA_LENGTH**		
Description	Defines the length of the user data contained in the NM PDU		
Type	const uint8		
Unit	--		
Range	*0..0x08*	--	
	--	--	
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	x	Variant 2
	Post Build	M,L	Variant 3
Scope	ECU		
Dependency	• The difference between CANNM_PDU_LENGTH and applied standardized bytes (source node identifier and control bit vector) within the NM PDU. • Valid values are 0x00..0x08.		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CanNm_ChannelConfig	1..*	Instance / --

10.2.5 CanNm_ChannelConfig

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

Name	**CANNM_CHANNEL_HANDLE**		
Description	Channel identifier configured for the respective instance of the NM.		
Type	const uint8		
Unit	--		
Range	0..*	--	
	--	--	
	Pre-compile	x	Variant 1
	Link time	x	Variant 2

	Post Build	M, L	Variant 3
Scope	instance		
Dependency	The `CANNM_CHANNEL_HANDLE` shall be encoded in the `canNmRxPduId` parameter which is passed to `CanNm_RxIndication()` function called by the CanIf. The `CANNM_CHANNEL_HANDLE` shall be encoded in the `canNmTxPduId` which is passed to `CanNm_TxConfirmation()` function called by the CanIf.		

Name	**`CANNM_MSG_CYCLE_TIME`**		
Description	Period of a NM-message. 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".		
Type	`const uint16`		
Unit	ms		
Range	0..*	--	
	--	--	
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	x	Variant 2
	Post Build	M, L	Variant 3
Scope	network		
Dependency	• `NM_TIMEOUT_TIME = n * CANNM_MSG_CYCLE_TIME` • This parameter is only valid if `CANNM_PASSIVE_MODE_ENABLED` is disabled		

Name	**`CANNM_MSG_CYCLE_OFFSET`**		
Description	Time offset in the periodic transmission node. It determines the start delay of the transmission.		
Type	`const uint16`		
Unit	ms		
Range	0..*	--	
	--	--	
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	x	Variant 2
	Post Build	M, L	Variant 3
Scope	instance		
Dependency	• <`CANNM_MSG_CYCLE_TIME` • This parameter is only valid if `CANNM_PASSIVE_MODE_ENABLED` is disabled		

Name	**`CANNM_MSG_REDUCED_TIME`**		
Description	Node specific bus cycle time in the periodic transmission mode with bus load reduction.		
Type	`const uint16`		
Unit	ms		
Range	`0..*`		
	--	--	
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	x	Variant 2
	Post Build	M, L	Variant 3
Scope	instance		
Dependency	• 0.5*CANNM_MSG_CYCLE_TIME <= CANNM_MSG_REDUCED_TIME < CANNM_MSG_CYCLE_TIME • Should be only present if pre-compile switch `CANNM_BUS_LOAD_REDUCTION_ENABLED` is defined • Value should be only processed if runtime configurable parameter		

	CANNM_BUS_LOAD_REDUCTION_ACTIVE is enabled • This parameter is only valid if CANNM_PASSIVE_MODE_ENABLED is disabled

Name	**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.		
Type	const uint16		
Unit	ms		
Range	0..*		
	--	--	
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	x	Variant 2
	Post Build	M, L	Variant 3
Scope	network		
Dependency	• This parameter is only valid if CANNM_PASSIVE_MODE_ENABLED is disabled • CANNM_MSG_TIMEOUT_TIME should be a multiple of CANNM_MSG_CYCLE_TIME		

Name	**CANNM_NODE_ID**		
Description	Node identifier of local node		
Type	const uint8		
Unit	--		
Range	0..*		
	--	--	
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	x	Variant 2
	Post Build	M, L	Variant 3
Scope	Instance		
Dependency	This parameter is only valid if CANNM_PASSIVE_NODE_ENABLED is set to OFF and CANNM_NODE_DETECTION_ENABLED is set to ON		

Name	**CANNM_TX_PDU_ID**		
Description	L-PDU handle of the NM PDU to be transmitted by **CanIf_Transmit** and passed to **CanNm_TxConfirmation** by the CanIf. This handle specifies the corresponding CAN frame ID and implicitly the CAN driver instance as well as the corresponding CAN controller device.		
Type	const PduIdType		
Unit	--		
Range	0..*		
	--	--	
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	x	Variant 2
	Post Build	M, L	Variant 3
Scope	Instance		
Dependency	• This parameter is only relevant if **CANNM_PASSIVE_MODE_ENABLED** is disabled. • If **CANNM_PASSIVE_MODE_ENABLED** is enabled no Tx PDU Id shall be configured in the CanIf.		

Name	**CANNM_CHANNEL_ACTIVE**
Description	Indicates whether a particular NM-channel shall be initialized (**TRUE**) or not (**FALSE**).
Type	const boolean
Unit	--

Range	TRUE	NM-Channel is enabled, i.e. shall be initialized	
	FALSE	NM-Channel is disabled, i.e. shall be not initialized	
Configuration Class	***Pre-compile***	x	Variant 1 and 3
	Link time	x	Variant 2
	Post Build	--	--
Scope	instance		
Dependency	none		

Name	**CANNM_BUS_LOAD_REDUCTION_ACTIVE**		
Description	Indicates whether the bus load reduction mechanism shall be enabled (**TRUE**) or disabled (**FALSE**) for a particular NM-channel.		
Type	const boolean		
Unit	--		
Range	TRUE	Bus load reduction mechanism shall be enabled for a particular channel	
	FALSE	Bus load reduction mechanism shall be not enabled for a particular channel	
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	x	Variant 2
	Post Build	M, L	Variant 3
Scope	instance		
Dependency	• This parameter should be only valid if the feature parameter **CANNM_BUS_LOAD_REDUCTION_ENABLED** is enabled. • This parameter is only valid if CANNM_PASSIVE_MODE_ENABLED is disabled		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
n/a	n/a	n/a

10.3 Published parameters

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

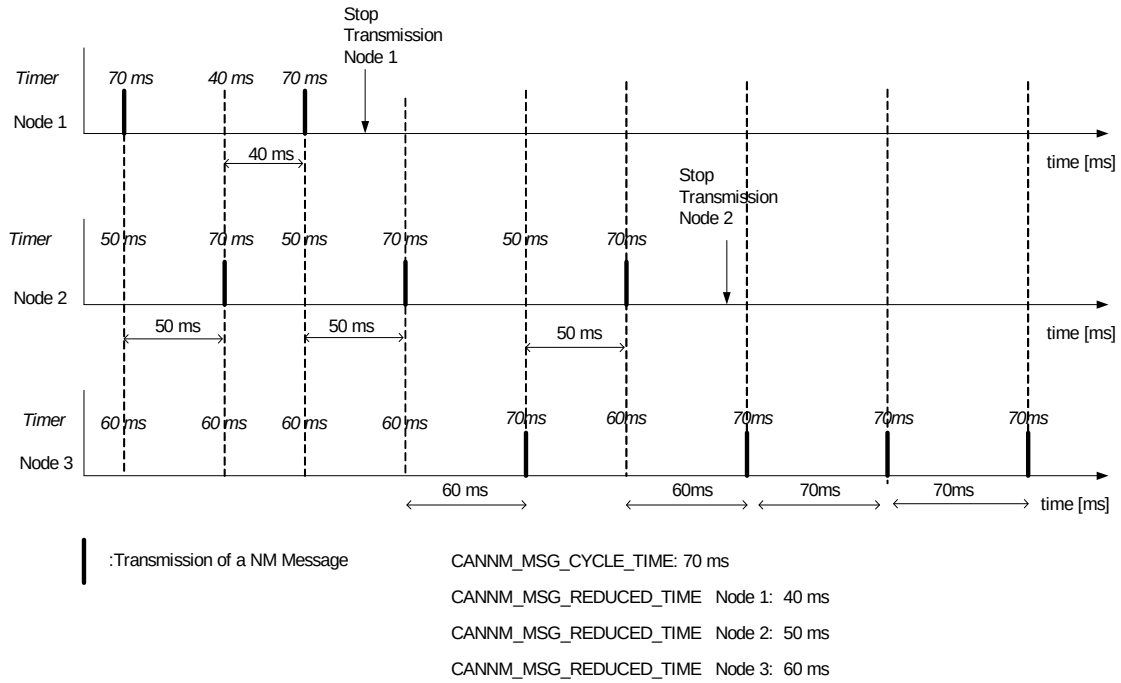
SWS Item		
Information elements		
Information element name	***Type / Range***	***Information element description***
CANNM_VENDOR_ID	#define /uint16	Vendor ID of the dedicated implementation of this module according to the AUTOSAR vendor list

CANNM_MODULE_ID	#define / 0x1F	Module ID of this module from Module List
CANNM_SW_MAJOR_VERSION	#define/ uint8	Major version number of the vendor specific implementation of the module. The numbering is vendor specific.
CANNM_SW_MINOR_VERSION	#define/ uint8	Minor version number of the vendor specific implementation of the module. The numbering is vendor specific.
CANNM_SW_PATCH_VERSION	#define/ uint8	Patch level version number of the vendor specific implementation of the module. The numbering is vendor specific.
CANNM_AR_MAJOR_VERSION	#define/ uint8	Major version number of the AUTOSAR specification of the module. The numbering is vendor specific.
CANNM_AR_MINOR_VERSION	#define/ uint8	Minor version number of the AUTOSAR specification of the module. The numbering is vendor specific.
CANNM_AR_PATCH_VERSION	#define/ uint8	Patch level version number of the AUTOSAR specification of the module. The numbering is vendor specific.

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.



12 Changes to Release 1

12.1 Deleted SWS Items

SWS Item	*Rationale*
CANNM004	Redundant requirement, already fulfilled by CANNM040 and CANNM032
CANNM006	Redundant requirement
CANNM007	Redundant requirement
CANNM027	Control Bit Vector feature removed

12.2 Replaced SWS Items

SWS Item of Release 1	*replaced by SWS Item*	*Rationale*
CANNM008	CANNM064, CANNM065, CANNM066, CANNM067, CANNM068	CANNM008 was not precise enough

12.3 Changed SWS Items

SWS Item	*Rationale*
CANNM023	unambiguously rephrased
CANNM024	unambiguously rephrased
CANNM005	unambiguously rephrased
CANNM013	Changed CANNM_SOURCE_ID to CANNM_NODE_ID
CANNM015	Rephrased
CANNM031	Rephrased
CANNM025	Extended with the reserved data part
CANNM001	Parts of requirements moved to a note because already requested by CNm
CANNM036	mMnor changes
CANNM002	Minor changes
CANNM014	Rephrased to cover also CanNm_GetRepeatMessageBit
CANNM023	Minor changes

12.4 Added SWS Items

SWS Item	*Rationale*
CANNM031	To be compliant with FrNm
CANNM032	Make clear when a NM message shall be transmitted
CANNM033	Extends CANNM023 from release 1
CANNM034	To be compliant with FrNm
CANNM035	To be compliant with FrNm
CANNM036	To be compliant with FrNm
CANNM037	To be compliant with FrNm
CANNM038	To be compliant with FrNm
CANNM039	Fulfill review comment
CANNM040	Specify timer handling in a more detailed way

CANNM041	To be compliant with FrNm
CANNM044	Apply new SWS template structure
CANNM045	Fulfill review comment #18 (AUTOSAR_Can_NM_Specification_Review_Rel2.xls)
CANNM047	Fulfill comments from WP4.2.2.1.2 meeting (04./05.07.2005)
CANNM048	Fulfill comments from WP4.2.2.1.2 meeting (04./05.07.2005)
CANNM050	Requested by BSW147
CANNM051	Missing in release 1
CANNM052	Missing in release 1
CANNM053	Missing in release 1
CANNM060	Release 1 doesn't support multiple configuration sets
CANNM061	Missing in release 1
CANNM064	Replaces CANNM008 to be more precise
CANNM065	Replaces CANNM008 to be more precise
CANNM066	Replaces CANNM008 to be more precise
CANNM067	Replaces CANNM008 to be more precise
CANNM068	Replaces CANNM008 to be more precise
CANNM069	Simplified chapter 7
CANNM071	Rework needed for R2.1
CANNM072	Rework needed for R2.1
CANNM070	Rework needed for R2.1
CANNM073	Rework needed for R2.1
CANNM074	Rework needed for R2.1
CANNM075	Rework needed for R2.1
CANNM076	Rework needed for R2.1
CANNM077	Rework needed for R2.1
CANNM085	Missing in release 1
CANNM086	Needed for R2.1