Document Title	Specification of Network Management Interface
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	228
Document Classification	Standard

Document Version	3.0.0
Document Status	Final
Part of Release	4.0
Revision	3

Document Change History

Date	Version	Changed by	Change Description
01.12.2011	3.0.0	AUTOSAR Administration	• NmMultipleChannelsEnabled removed • Added Mandatory Interfaces provided by ComM to Chapter 8.6.1 • move NmPassiveMode • Enabled form global configuration to channel configuration • Removed Nm_ReturnType • Fixed some min and max values of FloatPAramDef configuration parameters • Added support of NmCarWakup-Feature • Added support of coordinated shutdown of nested sub-busses
15.10.2010	2.1.0	AUTOSAR Administration	• Release check added • DET Error Code for false Pointer added • ChannelID harmonized in COM-Stack • Nm-State-changes in Userdata via NmIf
30.11.2009	2.0.0	AUTOSAR Administration	• Remove explicit support for OSEK NM from specification • NM Coordinator functionality reworked (chapter 7.2 and 7.2.8) • Debugging functionality added • Link time configuration variant introduced • Legal disclaimer revised
23.06.2008	1.0.1	AUTOSAR Administration	Legal disclaimer revised
03.12.2007	1.0.0	AUTOSAR Administration	Initial release

Disclaimer

Advice for users

AUTOSAR specifications 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 specifications for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such specifications, 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, interfaces and configuration of the **Generic Network Management Interface** module.

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

Additionally, this document describes the interoperability between several networks connected to the same (coordinator) ECU that run AUTOSAR NM, where 'interoperability' means that these networks can be put to sleep synchronously. This is also referred to as *NM Coordinator functionality*.

Support of the *NM Coordinator functionality* is optional. A **Generic Network Management Interface** implementation can either support only *Basic functionality* or both *Basic functionality* and *NM Coordinator functionality*.

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

2 Acronyms and abbreviations

Abbreviation / Acronym:	Description:
CanIf	CAN Interface
CanNm	CAN NM
CC	Communication Controller
ComM	Communication Manager
EcuM	ECU State Manager
DEM	Diagnostic Event Manager
DET	Development Error Tracer
Nm	Generic Network Management Interface module *This is the abbreviation used for this module throughout this specification.*
NM	Network Management
OEM	Original Equipment Manufacturer
CBV	Control Bit Vector in NM-message

Term:	Definition:
Bus-Sleep Mode	Network mode where all interconnected communication controllers are in the sleep mode.
NM-Channel	Logical channel associated with the NM-cluster
NM-Cluster	Set of NM nodes coordinated with use of the NM algorithm
NM-Coordinator	A functionality of the Generic NM Interface which allows coordination of network sleep for multiple NM Channels.
active NM-Coordinator	The NM-Coordinator which synchronizes the shutdown of all other NM-Coordinators connected to the Network
passive NM-Coordinator	An NM_Coordinator which is going to shutdown synchronized by an active NM_Coordinator
NM-Message	Packet of information exchanged for purposes of the NM algorithm.
NM-Timeout	Timeout in the NM algorithm that initiates transition into Bus-Sleep Mode.
NM User Data	Supplementary application specific piece of data that is attached to every NM message sent on the bus.
Node Identifier	Node address information exchanged for purposes of the NM algorithm.
Node Identifier List	List of Node Identifiers recognized by the NM algorithm.
Bus	Physical communication medium to which a NM node/ecu is connected to.
network	Entity of all NM nodes/ecus which are connected to the same bus.
channel	Logical bus to which the NM node/ecu is connected to

3 Related documentation

3.1 Input documents

[1] Layered Software Architecture
AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf

[2] General Requirements on Basic Software Modules
AUTOSAR_SRS_BSWGeneral.pdf

[3] Requirements on Network Management
AUTOSAR_SRS_NetworkManagement.pdf

[4] Specification of CAN Network Management,
AUTOSAR_SWS_CANNetworkManagement.pdf

[5] Specification of FlexRay Network Management
AUTOSAR_SWS_FlexRayNetworkManagement.pdf

[6] Specification of Communication Manager
AUTOSAR_SWS_COMManager.pdf

[7] Specification of Development Error Tracer
AUTOSAR_SWS_DevelopmentErrorTracer.pdf

[8] Requirements on Basic Software Module Description Template
AUTOSAR_RS_BSWModuleDescriptionTemplate.pdf

[9] List of Basic Software Modules
AUTOSAR_TR_BSWModuleList

3.2 Related standards and norms

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

4 Constraints and assumptions

4.1 Limitations

1. The **Generic Network Management Interface** can only be applied to communication systems that support broadcast communication and 'bus-sleep mode'.
2. There will be only one instance of the **Generic Network Management Interface** layer for all NM-Clusters. This instance manages all channels where a NM is used.
3. The **Generic Network Management Interface** shall only include the common modes, definitions and return values of different bus specific NM layers.

Figure 1 shows a typical example of the AUTOSAR NM stack.

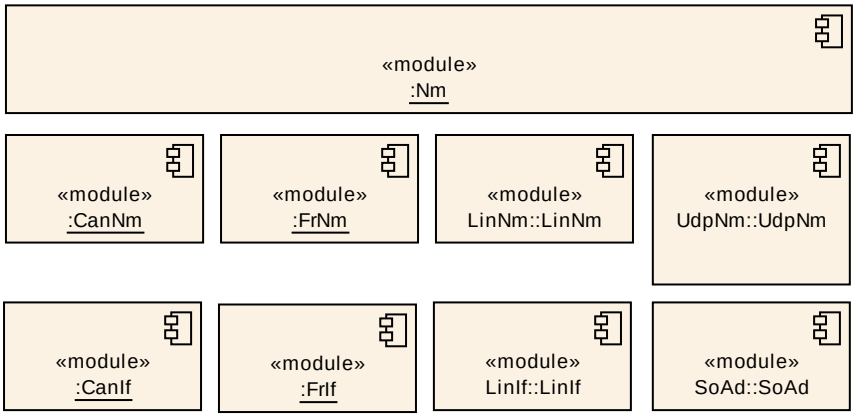

Figure 1 - NM stack modules

4.2 Specific limitations of the current release

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

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

Also; explicit support for OSEK NM has been completely removed from this specification as of AUTOSAR Release 4.0. OSEK NM can still be supported by extending the CanNm or by introducing a Complex Device Driver (CDD) on BusNm level as a generic BusNm. Supporting the OSEK NM through a CDD is not specified by AUTOSAR.

4.3 Applicability to car domains

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

5 Dependencies to other modules

5.1 Interfaces to modules

Figure 2 shows the interfaces provided to and required from other modules in the AUTOSAR BSW.

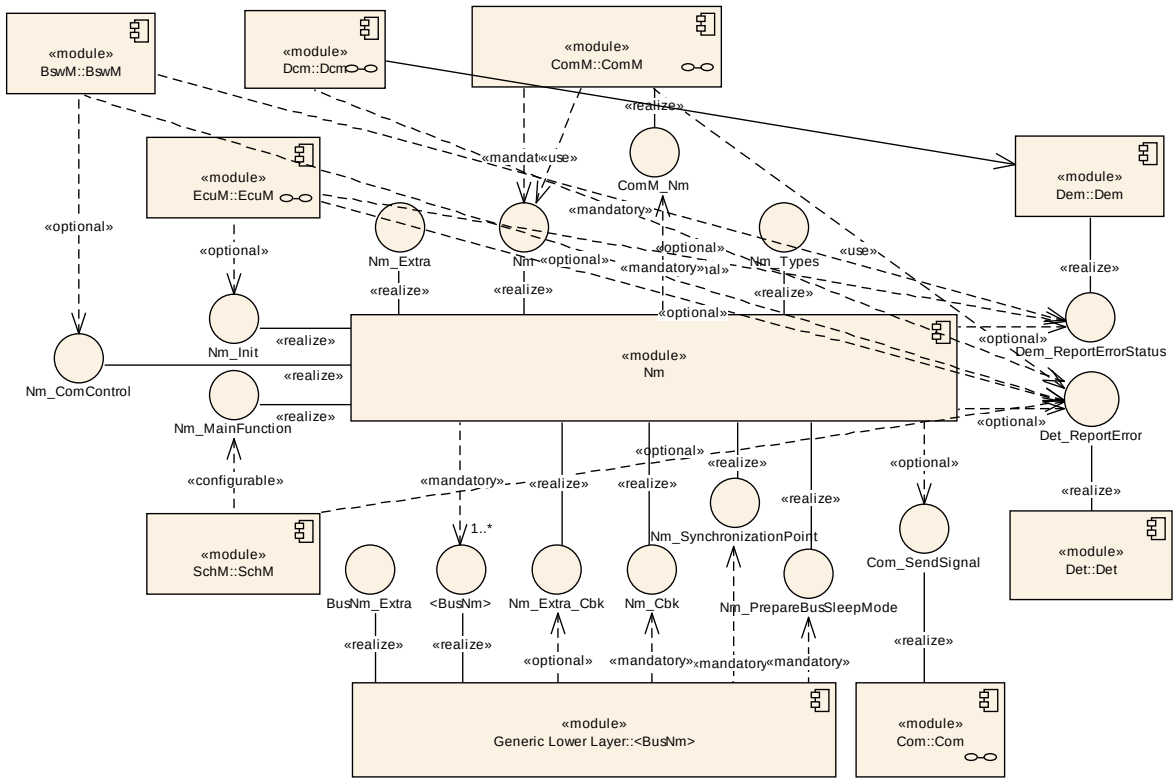

Figure 2 - Nm's interfaces to other modules

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

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

- CAN Network Management (**CanNm**)
- FlexRay Network Management (**FrNm**)
- LIN Network Management (**LinNm**)
- Ethernet Network Management (**UdpNm**).

With respect to *callbacks*, the **Nm** provides notification callbacks to the bus specific Network Management modules and calls the notification callbacks provided by the **ComM**.

In addition to the official AUTOSAR NM-modules above, Nm also support generic bus specific NM layers (**<BusNm>**). Any component which implements the required provided interfaces and uses the provided callback functions of Nm can be used as a bus specific NM. See Chapter 7.4 for the prerequisites for a generic bus specific NM.

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

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

5.1.2 Error handling modules (DET, DEM)

Nm will report development errors to the Development Error Tracer (**DET**) according to Nm025.

Nm will report production errors to the Diagnostic Event Manager (**DEM**) according to Nm026.

5.1.3 BSW Scheduler

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

5.2 File structure

5.2.1 Code file structure

[Nm247] ⌈ The code file structure shall not be defined within this specification completely. At this point it shall be pointed out that the code-file structure shall include the following files named:

- `Nm_Lcfg.c` (for link-time configurable parameters) ⌋()

5.2.2 Header file structure

[Nm123] ⌈ The Nm Interface module shall provide the following header files:

- `Nm.h` (for declaration of provided interface functions)
- `Nm_Cbk.h` (for declaration of provided call-back functions)

- `Nm_Cfg.h` (for pre-compile time configurable parameters)
- `NmStack_Types.h` (type definitions for the Nm Stack, see chapter Type definitions) 」(BSW00345, BSW159, BSW00380, BSW00419)

[Nm124] 「 The following header files will be included within the Nm Interface module:

- `Std_Types.h` (for AUTOSAR standard types)
  Note: `Platform_Types.h` (for platform specific types) and `Compiler.h` (for compiler specific language extensions) are indirectly included via AUTOSAR standard types.
- `MemMap.h` (for memory abstraction)
- `SchM_Nm.h` (for interfaces with the BSW Scheduler)
- `ComM_Nm.h` (for Communication Manager callback functions) 」(BSW00381, BSW00412, BSW00435, BSW00348, BSW00353)
- <cdd>.h For CarWakeup-functionalitiy a CDD is needed. The name of the CDD is generic.

[Nm242] 「 The Nm Interface shall optionally include the header file of **DEM** (depending on implementation of the NmIf)

- `Dem.h` for service of theDiagnostic Event Manager

***Rationale**: Production errors are not explicitly defined. This is up to the implementation of this module. If this module use the DEM functionality is shall include the provided header file from DEM. See also Nm125 and Nm026.* 」()

[Nm243] 「 The Nm Interface shall optionally include the header file of **DET** (depending on the pre-processor switch NmDevErrorDetect, see Nm203_Conf).

- `Det.h` for service of the Development Error Tracer 」()

6 Requirements traceability

Requirement	Satisfied by
-	Nm230
-	Nm255
-	Nm185
-	Nm176
-	Nm159
-	Nm187
-	Nm186
-	Nm164
-	Nm144
-	Nm134
-	Nm156
-	Nm262
-	Nm142
-	Nm139
-	Nm244
-	Nm241
-	Nm150
-	Nm234
-	Nm245
-	Nm161
-	Nm194
-	Nm130
-	Nm042
-	Nm112
-	Nm140
-	Nm260
-	Nm125
-	Nm173
-	Nm145
-	Nm172
-	Nm165
-	Nm136
-	Nm167

-	Nm253
-	Nm254
-	Nm233
-	Nm188
-	Nm143
-	Nm170
-	Nm127
-	Nm129
-	Nm151
-	Nm261
-	Nm133
-	Nm171
-	Nm126
-	Nm267
-	Nm177
-	Nm250
-	Nm146
-	Nm249
-	Nm162
-	Nm135
-	Nm128
-	Nm155
-	Nm168
-	Nm256
-	Nm235
-	Nm141
-	Nm002
-	Nm137
-	Nm149
-	Nm231
-	Nm183
-	Nm257
-	Nm192
-	Nm148
-	Nm258
-	Nm132
-	Nm182
-	Nm138

-	Nm266
-	Nm119
-	Nm228
-	Nm166
-	Nm265
-	Nm191
-	Nm147
-	Nm012
-	Nm003
-	Nm181
-	Nm001
-	Nm251
-	Nm240
-	Nm193
-	Nm114
-	Nm189
-	Nm169
-	Nm131
-	Nm248
-	Nm242
-	Nm252
-	Nm246
-	Nm236
-	Nm095
-	Nm247
-	Nm163
-	Nm051
-	Nm153
-	Nm158
-	Nm174
-	Nm190
-	Nm264
-	Nm006
-	Nm243
-	Nm152
BSW003	Nm044
BSW00301	Nm117
BSW00306	Nm999

BSW00307	Nm999
BSW00308	Nm999
BSW00309	Nm999
BSW00312	Nm999
BSW00314	Nm999
BSW00321	Nm999
BSW00323	Nm022
BSW00325	Nm999
BSW00326	Nm999
BSW00327	Nm232
BSW00328	Nm999
BSW00329	Nm999
BSW00330	Nm091
BSW00333	Nm028
BSW00334	Nm999
BSW00336	Nm999
BSW00339	Nm026
BSW00341	Nm999
BSW00344	Nm030
BSW00345	Nm123
BSW00347	Nm999
BSW00348	Nm124
BSW00350	Nm022
BSW00353	Nm124
BSW00355	Nm999
BSW00357	Nm999
BSW00358	Nm030
BSW00361	Nm999
BSW00371	Nm999
BSW00373	Nm020
BSW00375	Nm999
BSW00376	Nm118
BSW00380	Nm123
BSW00381	Nm124
BSW00385	Nm232
BSW00386	Nm022, Nm023
BSW00387	Nm091
BSW00399	Nm999

BSW004	Nm999
BSW00400	Nm999
BSW00404	Nm999
BSW00405	Nm030
BSW00407	Nm044
BSW00409	Nm999
BSW00411	Nm154
BSW00412	Nm124
BSW00413	Nm999
BSW00414	Nm030
BSW00416	Nm121
BSW00417	Nm999
BSW00419	Nm123
BSW00422	Nm999
BSW00423	Nm999
BSW00424	Nm118
BSW00425	Nm118
BSW00426	Nm999
BSW00427	Nm999
BSW00428	Nm999
BSW00429	Nm999
BSW00431	Nm999
BSW00432	Nm999
BSW00433	Nm999
BSW00434	Nm999
BSW00435	Nm124
BSW00436	Nm999
BSW00437	Nm999
BSW00438	Nm999
BSW00439	Nm999
BSW00440	Nm999
BSW005	Nm999
BSW006	Nm999
BSW007	Nm999
BSW009	Nm999
BSW010	Nm999
BSW02503	Nm035
BSW02504	Nm036

BSW02505	Nm039
BSW02506	Nm037
BSW02508	Nm040
BSW02509	Nm999
BSW02510	Nm999
BSW02511	Nm999
BSW02512	Nm034, Nm033
BSW02513	Nm031, Nm033, Nm032
BSW043	Nm999
BSW046	Nm031, Nm032
BSW047	Nm034, Nm032
BSW048	Nm046
BSW050	Nm043
BSW051	Nm031, Nm032, Nm046
BSW052	Nm999
BSW053	Nm999
BSW054	Nm999
BSW101	Nm030
BSW137	Nm999
BSW142	Nm999
BSW143	Nm999
BSW144	Nm999
BSW145	Nm999
BSW146	Nm999
BSW147	Nm999
BSW149	Nm175
BSW150	Nm055, Nm224, Nm025, Nm022
BSW151	Nm031
BSW153	Nm038
BSW159	Nm123
BSW161	Nm999
BSW162	Nm999
BSW164	Nm999
BSW168	Nm999
BSW170	Nm999
BSW172	Nm999

According to [2] General Requirements on Basic Software Modules.

Requirement	***Satisfied by***
Functional Requirements	
Configuration	
[BSW00344] Reference to link-time configuration	Nm030
[BSW00404] Reference to post build time configuration	n/a *(no post build parameters)*
[BSW00405] Reference to multiple configuration sets	Nm030
[BSW00345] Pre-compile-time configuration	Nm123
[BSW159] Tool-based configuration	Nm123
[BSW167] Static configuration checking	Chapter 10
[BSW171] Configurability of optional functionality	Chapter 10
[BSW170] Data for reconfiguration of AUTOSAR SW-Components	n/a *(not an SW-C)*
[BSW00380] Separate C-Files for configuration parameters	Nm123
[BSW00419] Separate C-Files for pre-compile time configuration parameters	Nm123
[BSW00381] Separate configuration header file for pre-compile time parameters	Nm124
[BSW00412] Separate H-File for configuration parameters	Nm124
[BSW00383] List dependencies of configuration files	Chapter 5
[BSW00384] List dependencies to other modules	Chapter 5
[BSW00387] Specify the configuration class of callback function	Nm091
[BSW00388] Introduce containers	Chapter 10
[BSW00389] Containers shall have names	Chapter 10
[BSW00390] Parameter content shall be unique within the module	Chapter 10
[BSW00391] Parameter shall have unique names	Chapter 10
[BSW00392] Parameters shall have a type	Chapter 10
[BSW00393] Parameters shall have a range	Chapter 10
[BSW00394] Specify the scope of the parameters	Chapter 10
[BSW00395] List the required parameters (per parameter)	Chapter 10
[BSW00396] Configuration classes	Chapter 10
[BSW00397] Pre-compile-time parameters	Chapter 10
[BSW00398] Link-time parameters	Chapter 10
[BSW00399] Loadable Post-build time parameters	n/a *(no post build parameters)*
[BSW00400] Selectable Post-build time parameters	n/a *(no post build parameters)*
[BSW00438]	n/a *(no post build parameters)*
[BSW00402] Published information	Chapter 10.6
Wake-Up	
[BSW00375] Notification of wake-up reason	n/a *(no wake-up interrupt handled)*
Initialization	
[BSW101] Initialization interface	Nm030
[BSW00416] Sequence of Initialization	Nm121
[BSW00406] Check module initialization	Errors defined in interfaces
[BSW00437] NoInit-Area in RAM	n/a *(implementation specific)*
Normal Operation	
[BSW168] Diagnostic Interface of SW components	n/a *(not an SW-C)*
[BSW00407] Function to read out published parameters	Nm044
[BSW00423] Usage of SW-C template to describe BSW modules with AUTOSAR Interfaces	n/a *(not an SW-C)*
[BSW00424] BSW main processing function task allocation	Nm118
[BSW00425] Trigger conditions for schedulable objects	Nm118
[BSW00426] Exclusive areas in BSW modules	n/a *(implementation specific)*
[BSW00427] ISR description for BSW modules	n/a *(no ISR functions)*
[BSW00428] Execution order dependencies of main processing functions	n/a *(no dependencies)*
[BSW00429] Restricted BSW OS functionality access	n/a *(implementation specific)*

[BSW00431] The BSW Scheduler module implements task bodies	n/a *(not in the scope of this specification)*
[BSW00432] Modules should have separate main processing functions for read/receive and write/transmit data path	n/a *(no read/receive or write/transmit operations)*
[BSW00433] Calling of main processing functions	n/a *(not in the scope of this specification)*
[BSW00434] The Schedule Module shall provide an API for exclusive areas	n/a *(implementation specific)*
Shutdown Operation	
[BSW00336] Shutdown interface	n/a *(no de-initialization function)*
Fault Operation and Error Detection	
[BSW00337] Classification of errors	Chapter 7.6
[BSW00338] Detection and Reporting of development errors	Chapter 0 and 7.8
[BSW00369] Do not return development error codes via API	Chapter 8
[BSW00339] Reporting of production relevant error status	Nm026
[BSW00422] Pre-de-bouncing of production relevant error status	n/a *(not in the scope of this specification)*
[BSW00417] Reporting of Error Events by Non-Basic Software	n/a *(not in the scope of this specification)*
[BSW00323] API parameter checking	Nm022
[BSW004] Version check	n/a *(implementation specific)*
[BSW00409] Header files for production code error IDs	n/a *(no production code errors)*
[BSW00385] List possible error notifications	Nm232
[BSW00386] Configuration for detecting an error	Nm022, Nm023
Non--functional Requirements	
Software Architecture Requirements	
[BSW161] Microcontroller abstraction	n/a *(not in the scope of this specification)*
[BSW162] ECU layout abstraction	n/a *(not in the scope of this specification)*
[BSW005] No hard coded horizontal interfaces within MCAL	n/a *(not in the scope of this specification)*
[BSW00415] User dependent include files	Chapter 5
Software Integration Requirements	
[BSW164] Implementation of interrupt service routines	n/a *(no interrupt routines)*
[BSW00325] Runtime of interrupt service routines	n/a *(no interrupt routines)*
[BSW00326] Transition from ISRs to OS tasks	n/a *(no interrupt routines)*
[BSW00342] Usage of source code and object code	Chapter 10
[BSW00343] Specification and configuration of time	Nm222_Conf
[BSW160] Human-readable configuration data	Chapter 10
Software Module Design Requirements	
Software quality	
[BSW007] HIS MISRA C	n/a *(implementation specific)*
Naming conventions	
[BSW00300] Module naming convention	Ok *(Nm)*
[BSW00413] Accessing instances of BSW modules	n/a *(no instantiation of NM Interface)*
[BSW00347] Naming separation of different instances of BSW drivers	n/a *(not BSW driver)*
[BSW00441] Enumeration literals and #define naming convention	Chapter 8.2, Nm220_Conf
[BSW00305] Self-defined data types naming convention	Chapter 8.2
[BSW00307] Global variables naming convention	n/a *(implementation specific)*
[BSW00310] API naming convention	Chapter 8
[BSW00373] Main processing function naming convention	Nm020
[BSW00327] Error values naming convention	Nm232
[BSW00335] Status values naming convention	Chapter 8.2
[BSW00350] Development error detection keyword	Nm022, Nm203_Conf
[BSW00408] Configuration parameter naming convention	Chapter 10

[BSW00410] Compiler switches shall have defined values	Chapter 10.2
[BSW00411] Get version info keyword	Nm153, Nm204_Conf
Module file structure	
[BSW00346] Basic set of module files	Chapter 5.2
[BSW158] Separation of configuration from implementation	Chapter 5.2
[BSW00314] Separation of interrupt frames and service routines	n/a *(no interrupt frames)*
[BSW00370] Separation of callback interface from API	Chapter 5.2
[BSW00435] Module Header File Structure for the Basic Software Scheduler	Nm124
[BSW00436] Module Header File Structure for the Basic Software Memory Mapping	n/a *(implementation specific)*
Standard header files	
[BSW00348] Standard type header	Nm124
[BSW00353] Platform specific type header	Nm124
[BSW00361] Compiler specific language extension header	n/a *(no extensions)*
[BSW00301] Limit imported information	Nm117
[BSW00302] Limit exported information	Chapter 5.1
[BSW00328] Avoid duplication of code	n/a *(implementation specific)*
[BSW00312] Shared code shall be reentrant	n/a *(implementation specific)*
[BSW006] Platform independency	n/a *(not in the scope of this specification)*
[BSW00439] Declaration of interrupt handlers and ISRs	n/a *(no ISR functions)*
Types and keywords	
[BSW00357] Standard API return type	Chapter 8
[BSW00377] Module specific API return types	n/a
[BSW00304] AUTOSAR integer data types	Chapter 8
[BSW00355] Do not redefine AUTOSAR integer data types	n/a *(No integer data types defined)*
[BSW00378] AUTOSAR boolean type	Chapter 8
[BSW00306] Avoid direct use of compiler and platform specific keywords	n/a *(implementation specific)*
Global data	
[BSW00308] Definition of global data	n/a *(No global data defined)*
[BSW00309] Global data with read-only constraint	n/a *(No global data defined)*
Interface and API	
[BSW00371] Do not pass function pointers via API	n/a *(No function pointers passed)*
[BSW00358] Return type of init() functions	Nm030
[BSW00414] Parameter of init function	Nm030
[BSW00376] Return type and parameters of main processing functions	Nm118
[BSW00359] Return type of callback functions	Chapter 8.4
[BSW00360] Parameters of callback functions	Chapter 8.4
[BSW00440] Function prototype for callback functions of AUTOSAR Services	n/a *(No RTE services)*
[BSW00329] Avoidance of generic interfaces	n/a *The NM Interface is rewritten to support Generic Interfaces to support new Bus Specific NM modules in the context of the Complex Device Driver concept.*
[BSW00330] Usage of macros / inline functions instead of functions	Nm091
[BSW00331] Separation of error and status values	Chapter 8.2
Software Documentation Requirements	
[BSW009] Module User Documentation	n/a *(implementation specific)*
[BSW00401] Documentation of multiple instances of configuration parameters	Chapter 10
[BSW172] Compatibility and documentation of scheduling strategy	n/a *(no internal scheduling policy)*
[BSW010] Memory resource documentation	n/a *(implementation specific)*

[BSW00333] Documentation of callback function context	Nm028
[BSW00374] Module vendor identification	Nm008
[BSW00379] Module identification	Nm008
[BSW003] Version identification	Nm008, Nm044
[BSW00318] Format of module version numbers	Nm008
[BSW00321] Enumeration of module version numbers	n/a *(implementation specific)*
[BSW00341] Microcontroller compatibility documentation	n/a *(implementation specific)*
[BSW00334] Provision of XML file	n/a *(implementation specific)*

According to [3] Requirements on Network Management.

Requirement	***Satisfied by***
Functional Requirements	
Configuration	
[BSW150] Configuration of functionality	Nm055, Nm224, Chapter 7.4.3, Nm022, Nm025, Chapter 10
Initialization	
[BSW151] Integration into running NM cluster	Nm031
[BSW043] Bus Traffic without NM Initialization	n/a *(not in the scope of this specification)*
Normal Operation	
[BSW044] Applicability to different types of communication systems	Chapter 5.1.1
[BSW02515] Compliance with non AUTOSAR-NMs	Chapter 5.1.1
[BSW045] NM-cluster Independent Shutdown Coordination	Chapter 7
[BSW02513] Control of NM	Nm031, Nm032, Nm033
[BSW046] Trigger of startup of all Nodes at any Point in Time	Nm031, Nm032
[BSW047] Bus Keep Awake Services	Nm032, Nm034
[BSW048] Bus Sleep Mode	Nm046
[BSW050] NM State Information	Nm043
[BSW051] NM State Change Indication	Chapter 7.4.1, Nm032, Nm046, Nm031
[BSW052] Notification that all other ECUs are ready to sleep	n/a *(Handled internally in Nm)*
[BSW02509] Notification that at least one other node is not ready to sleep anymore	n/a *(Handled internally in Nm)*
[BSW02503] Sending user data	Nm035
[BSW02504] Receiving user data	Nm036
[BSW153] Detection of present nodes	Nm038
[BSW02508] Unambiguous node identification per bus	Nm040
[BSW02505] Sending node identifier	Nm039
[BSW02506] Receiving node identifier	Nm037
[BSW02511] Configurable Role in Cluster Shutdown	n/a *(Bus specific, outside the scope of this specification)*
Fault Operation	
[BSW053] Deterministic Behavior in Case of Bus Unavailability	n/a *(Bus specific, outside the scope of this specification)*
[BSW137] Communication system error handling	n/a *(Bus specific, outside the scope of this specification)*
Gateway Operation	
[BSW02514] Coordination of coupled networks	Chapter 7.2
Non-Functional Requirements (Qualities)	
Timing Requirements	
[BSW054] Deterministic Time for Bus Sleep	n/a *(Bus specific, outside the scope of this specification)*
Resource Usage	
[BSW142] Limitation of NM bus load	n/a *(Bus specific, outside the scope of this specification)*
[BSW143] Predictable NM bus load	n/a *(Bus specific, outside the scope of*

	this specification)
[BSW144] ECU cluster size	n/a *(Bus specific, outside the scope of this specification)*
[BSW145] Robustness against NM message losses	n/a *(Bus specific, outside the scope of this specification)*
[BSW146] Robustness against NM message jitter	n/a *(Bus specific, outside the scope of this specification)*
[BSW147] Processor independent algorithm	n/a *(outside the scope of this specification)*
[BSW149] Configurable Timing	Nm175
Hardware independency	
[BSW154] Bus independency of API	Chapter 7.1 and 8.3
CAN Specific Requirements	
Resource Usage	
[BSW148] Separation of Communication system dependent parts	Chapter 5.1.1
Transmission Confirmation	
[BSW02510] Immediate Transmission Confirmation	n/a *(Bus specific, outside the scope of this specification)*
Diagnostic Service	
[BSW02512] CommunicationControl (28 hex) service support	Nm033, Nm034
FlexRay Specific Requirements	
n/a	

7 Functional specification

The NM Interface functionality consists of two parts:

- The *Base functionality* necessary to run, together with the bus specific NM modules, AUTOSAR NM on an ECU.
- The *NM Coordinator functionality* used by gateway ECUs to synchronously shut down one ore more busses.

7.1 Base functionality

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

[Nm095] ⌈ The **Nm** shall not provide interface functions beyond those specified in this document ⌋()

***Rationale**: The **Nm** should provide an interface to the **ComM**, that does not contain specific knowledge about the type of the underlying busses, and that nevertheless should be sufficient to accomplish the necessary network management functions. Because of this requirement, the algorithm handled by the **Nm** shall be bus independent.*
*Note: It is also required that other service layer modules access network management functions exclusively via **Nm** and that no bypasses to bus specific NM functions exist*

[Nm006] ⌈ The **Nm** shall convert generic function calls from the **ComM** to bus specific functions of the bus specific NM layer. ⌋()

[Nm012] ⌈ The **Nm** shall convert callback functions called by the bus specific NM layers to generic callbacks to the **ComM**. ⌋()

[Nm091] ⌈ The *Base functionality* of **Nm** may be implemented completely or partly using macros ⌋(BSW00387, BSW00330)

7.2 NM Coordinator functionality

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

An ECU using an NM that actively performs the NM Coordinator functionality is commonly referred to as an *NM Coordinator*. However, in this specification this term will be synonymous with the *NM Coordinator functionality* when used in requirements.
Note: *Consider that certain bus types have different nomenclature on the terms Network, Channel, Cluster.*

[Nm224] ⌈ If the *NM Coordinator functionality* is configured, the configuration parameter `NmCycletimeMainFunction` shall be configured with the cycle time of the rate at which two successive calls to the **Nm**'s main function (ref Nm118) is made. ⌋ (BSW150)

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

7.2.1 Applicability of the NM Coordinator functionality

[Nm001] ⌈ The *coordinator algorithm* shall be able to handle a topology where several coordinated busses are connected to one *NM Coordinator*. ⌋()

[Nm256] ⌈ The NM-Coordinator shall support two or more NM-Coordinators connected to the same NM Cluster.⌋()

[Nm051]: ⌈ The *NM Coordinator* shall be able to coordinate busses running the official AUTOSAR bus specific NMs (**CanNm**, **FrNm**, **LinNm** and **UdpNm**) as well as all other generic bus NMs implementing the required functionality, callbacks and interfaces as specified in Chapter 7.4.2. ⌋()

[Nm055]: ⌈ The NM Interface configuration shall provide the parameter `NmCoordinatorSupportEnabled` to define if the coordinator algorithm to support the NM Coordinator functionality is present or not. ⌋(BSW150)

[Nm167] ⌈ It shall be possible to configure multiple *NM coordination clusters* that shall be coordinated independently. ⌋()

[Nm168] ⌈ Each bus shall belong to zero or one *NM coordination cluster*. ⌋()

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

[Nm169] ⌈ Shutdown shall only be coordinated on the presently awake networks of a *coordination cluster*. Networks that are already in 'bus-sleep mode' shall still be monitored but not coordinated. ⌋()

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

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

[Nm244] ⌈ All NM networks configured to be part of a *coordinated cluster* of the *NM coordinator functionality* shall have the corresponding **Bus NM** configured to be able to actively send out NM messages (e.g. CANNM_PASSIVE_MODE_ENABLED = false).
As a result of this configuration restriction, all **BusNm** used by the coordinator functionality of the **Nm** module shall provide the API `<BusNm>_NetworkRequest()`. ⌋()

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

7.2.2 Keeping coordinated busses alive

[Nm002]:⌈ As long as the node implementing the *NM Coordinator* is not ready to go to sleep on at least one of the busses in a *coordination cluster* (i.e. that it has actively requested the network), the *NM Coordinator* shall ensure that the network is requested on all currently active busses in that coordination cluster. ⌋()

[Nm003]:⌈ As long as at least one bus in the *coordination cluster* is not ready to sleep (i.e. because another node than the *NM Coordinator* is requesting that bus), the *NM Coordinator* shall still ensure that the network is requested on all currently active busses in that *coordination cluster* even if the local ECU itself is ready to go to sleep on all busses of that *coordination cluster* ⌟()

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

***Rationale**: The **Nm** will request the network on a bus by calling the bus specific NMs function `BusNm_NetworkRequest()`.*

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

The two requirements Nm002 and Nm003 above can be summarized as follows: as long as at least one node (including the node implementing the *NM Coordinator*) keeps any of the busses in the *coordination cluster* awake, the *NM Coordinator* shall keep all busses of that coordination cluster awake.

[Nm228] 「 If a bus of a *coordination cluster* has the parameter `NmChannelSleepMaster` set to `TRUE`, the NM Coordinator shall consider that bus ready to sleep at all times and shall not await an invocation of `Nm_RemoteSleepIndication()` from that bus before starting shutdown of that network. 」()

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

[Nm257] 「Passive NM-Coordinator(s) send Nm messages only if the node has a network management request pending or a connected network which is coordinated active by that coordinator is not ready to sleep.」()

Rationale: This prevents that both, active NM-Coordinator and passive NM_Coordinator, send NM messages within CANNM_MSG_CYCLE_TIME when they are ready to sleep. Without this mechanism it would not be possible to detect if there is at least one other node active.

7.2.3 Shutdown of coordinated busses

The level of synchronization achievable is dependent on the configuration. See Chapter 7.2.4. Figure 3 shows an overview of the *coordination algorithm*. As described in Section 7.2.1, the *shutdown algorithm* shall be applied independently per *NM coordination cluster*.

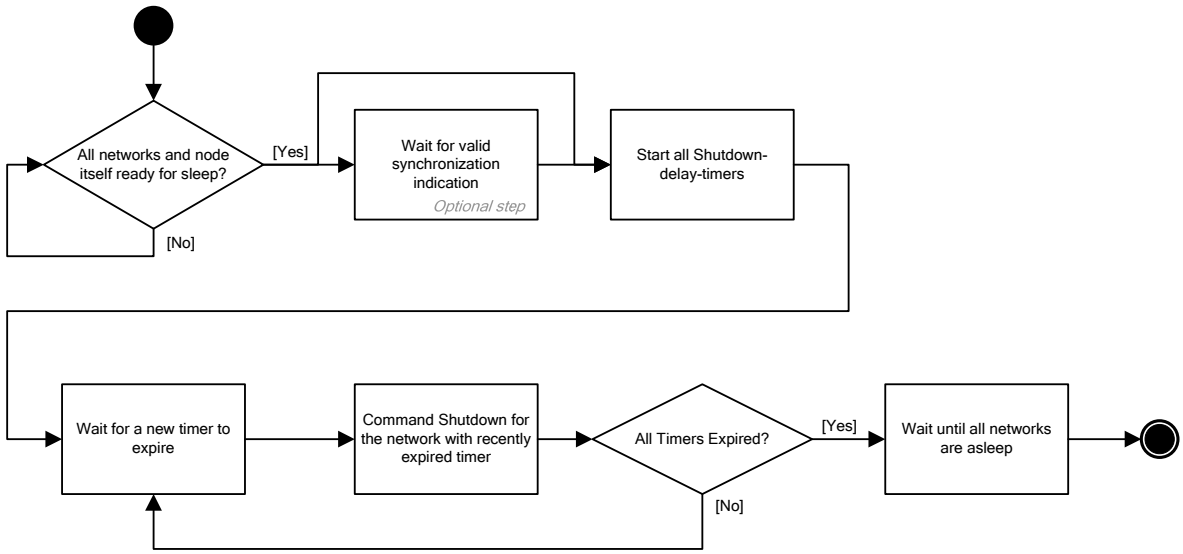

Figure 3 - Overview of and entry to the *coordination algorithm*

[Nm170] ⌈ The actions of the coordination algorithm shall not be limited to the main function of **Nm** (`Nm_MainFunction()`).
Rationale: *No limitation of the implementation.*
Note: *The Nm Coordinator algorithm is allowed to perform required actions directly on any indications/callbacks (except for evaluating timers). Timers should/could be initiated inside callbacks. The main function should perform timer-evaluation and trigger expired timers. The main function should perform coordination initiation evaluation and initiate timers if they are not already initiated.* ⌋()

[Nm171] ⌈ The *NM Coordinator* shall constantly evaluate whether all presently awake networks of a *coordination cluster* are ready to go to sleep. The *coordination algorithm* may not be started until all networks of the coordination cluster are either ready to go to sleep or are already in 'bus-sleep mode'. The evaluation shall be carried out in all API calls of the **Nm** including the main function. ⌋()

Rationale*: It is not necessary to re-evaluate all conditions in every main processing period and every API call. Since all events that can lead to a change in a networks status will result in API calls to* ***Nm*** *(either from* ***ComM*** *or from the bus specific NMs), it is only necessary to re-evaluate those specific conditions that are affected by that API call.*

[Nm172] ⌈ If the configuration parameter `NmSynchronizingNetwork` is `TRUE` for any of the busses in a *coordination cluster*, the *coordination algorithm* must wait for a valid synchronization indication before the shutdown timers can be started. The validity of a synchronization indication is done by evaluating if the bus specific NM which has invoked the `Nm_SynchronizationPoint()` is configured as a synchronizing network for the *coordination cluster* it is part of. ⌋()

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

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

[Nm173] ⌈ If not all conditions to start coordinated shutdown have been met, or if the coordination algorithm has already been started (but not aborted), calls to `Nm_SynchronizationPoint()` shall be ignored. ⌋()

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

[Nm174] ⌈ If the configuration parameter `NmSynchronizingNetwork` is `FALSE` for all of the presently awake busses in a *coordination cluster*, the *coordination algorithm* shall start the timers after all shutdown conditions have been met (see also Nm172). ⌋()

[Nm175] ⌈ When the coordination algorithm is started, a shutdown timer shall be activated for all currently awake networks in the *coordination cluster*. Each timer shall be configured with the default shutdown time for that network given by the configuration parameter `NmShutdownDelayTimer`. ⌋(BSW149)

[Nm176] ⌈ When a shutdown timer expires for a network, Nm shall release the network by calling the `BusNm_RequestBusSynchronization()` followed by `BusNm_NetworkRelease()`. ⌋()

[Nm177] ⌈ Nm shall keep track of all networks that have been released but have not yet reported 'bus-sleep mode'. If the shutdown is aborted, these networks shall still be considered active networks. (See Section 7.3.3). ⌋()

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

7.2.4 Calculation of shutdown timers

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

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

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

7.2.5 Synchronization Use Case 1 – Synchronous command

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

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

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

7.2.6 Synchronization Use Case 2 – Synchronous initiation

This Use Case is an extension of Use Case 1, but here consideration is taken to the fact that for some networks the request to release the network will only be acted

upon at specific points in time. This Use Case will command a simultaneous shutdown like in Use Case 1, but will wait until a point in time suitable for the synchronizing network.

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

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

7.2.7 Synchronization Use Case 3 – Synchronous network sleep

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

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

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

To calculate the shutdown timer parameter `NmShutdownDelayTimer` of each network, specific knowledge of each networks timing behavior must be obtained.

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

For the synchronizing network, $\mathbf{T_{SYNCHRONIZATION\_INDICATION}}$ must be determined. This is the time between any two consecutive calls made by that bus specific NM to `Nm_SynchronizationPoint()`.

The **$T_{TOTAL\_SHUTDOWN}$** shall be the total time that is needed for the coordination algorithm. Start with setting **$T_{TOTAL\_SHUTDOWN}$** to the same value as **$T_{SHUTDOWN}$** for the synchronizing network. If the **$T_{SHUTDOWN}$** for any other network is greater than **$T_{TOTAL\_SHUTDOWN}$**, extend **$T_{TOTAL\_SHUTDOWN}$** with **$T_{SYNCHRONIZATION\_INDICATION}$** repeatedly until **$T_{TOTAL\_SHUTDOWN}$** is equal to, or larger than any **$T_{SHUTDOWN}$**.

The shutdown timer parameter `NmShutdownDelayTimer` for each network shall be calculated as **$T_{TOTAL\_SHUTDOWN}$** - **$T_{SHUTDOWN}$** for that network.

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

7.2.7.1 Examples

In the first case (Figure 4), the synchronizing network holds the largest **$T_{SHUTDOWN}$**, which will therefore equal the **$T_{TOTAL\_SHUTDOWN}$**. For the synchronizing network, the shutdown timer will be **$T_{TOTAL\_SHUTDOWN}$** - **$T_{SHUTDOWN}$**, which is zero, but then a small amount of time is added to make sure that the Nm will wait to release the network between the two synchronization points.
For the Non-cyclic network, the shutdown timer will simply be **$T_{TOTAL\_SHUTDOWN}$** - **$T_{SHUTDOWN}$**.

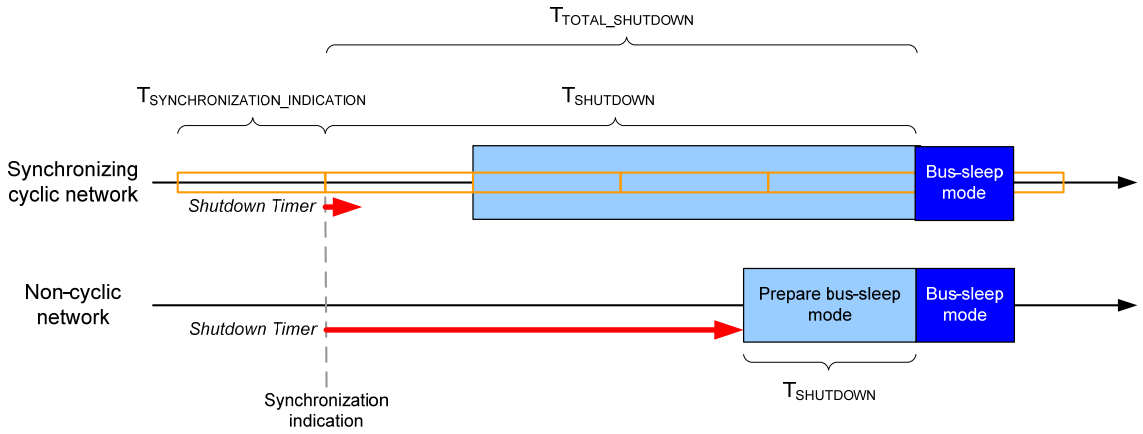

Figure 4 - Timing example one

In the second case (Figure 5), the non-cyclic network takes very long time to shut down and therefore holds the largest **$T_{SHUTDOWN}$**. The **$T_{TOTAL\_SHUTDOWN}$** has now been obtained by taking the synchronizing network's (slightly shorter) **$T_{SHUTDOWN}$** adding **$T_{SYNCHRONIZATION\_INDICATION}$** once to this value.

For the synchronizing network, the shutdown timer will be **$T_{TOTAL\_SHUTDOWN}$ - $T_{SHUTDOWN}$**, with a small amount of time added to make sure that the Nm will wait to release the network between the two synchronization points.
For the Non-cyclic network, the shutdown timer will simply be **$T_{TOTAL\_SHUTDOWN}$ - $T_{SHUTDOWN}$**.

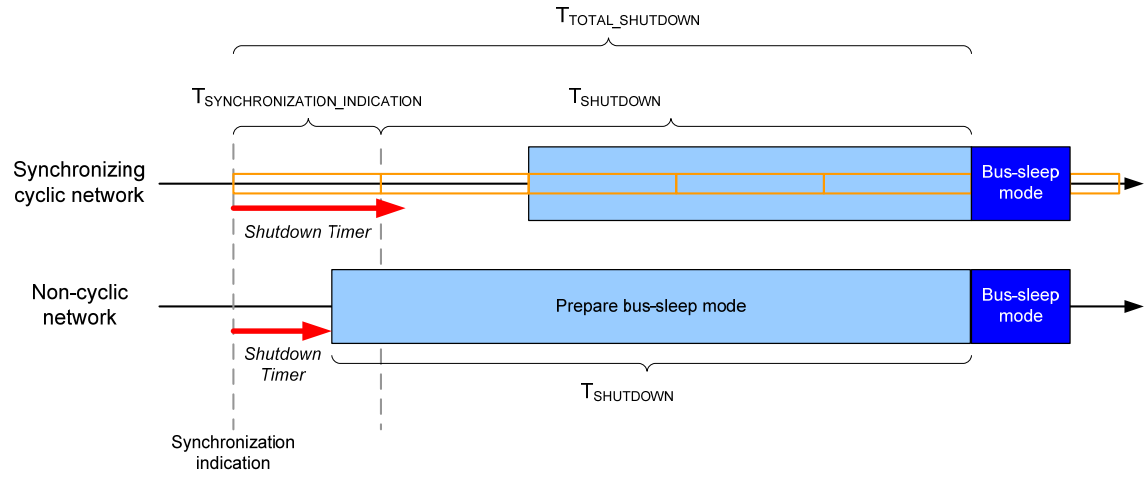

Figure 5 - Timing example two

7.2.8 Coordination of nested sub-busses

To support the coordination of nested sub-busses the Nm-Coordinator, which are linked together, have to be configured as an active or passive Nm-Coordinator. Defining the active Nm-Coordinator Network by Network allows an unlimited topology of Networks, assuming that there is always one active Nm-Coordinator on one Network.

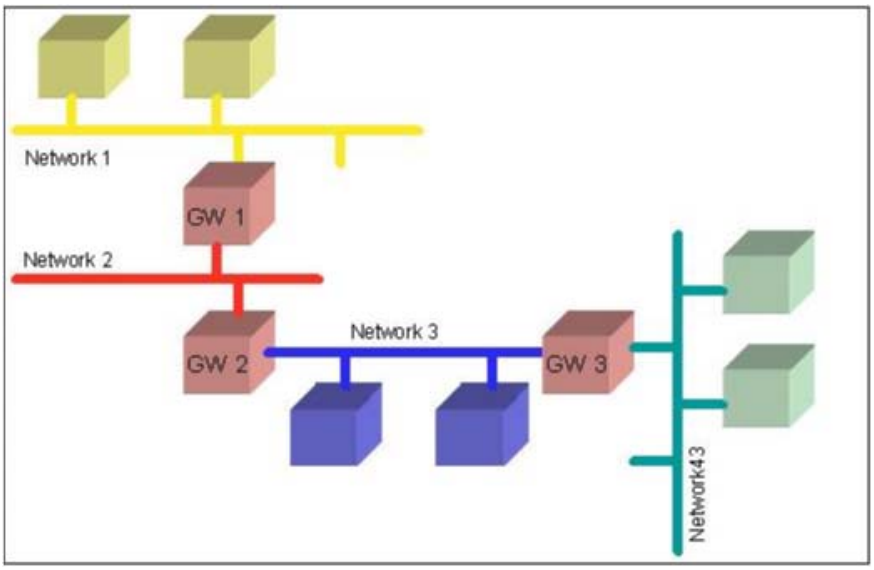

Figure 6: Use Case Nested Gateways

The topology shown in Figure 6 may have following coordination approach. GW 1 can be active Nm-Coordinator on Network 1 and 2, where GW 2 is passive Nm-Coordinator on Network 2 but active Nm-Coordinator on Network 3. GW 3 than needs to be passive Nm-Coordinator on Network 3 but active Nm-Coordinator on Network43. This Topology is only an example of one possible topology shut down coordinated by **Nm** implemented after following requirements.

[Nm258]「The NMactiveCoordinator parameter shall indicate, if an NM Coordinator is an active Nm-Coordinator(NMactiveCoordinator = TRUE) or a passive Nm-Coordinator. This parameter needs to be available per NM channel.」()

Nm259「The active NM Coordinator shall set the *NMcoordinatorSleepReady* bit in the NM message via `<BusNm>_SetSleepReadyBit(Nm_Channel, value)` to the value 1, if all nodes of the NM cluster are ready to sleep.」()

Note: *for Position of Coordinator Bits in CBV see Table 1.*

[Nm260]「A passive NM Coordinator must not set the NMcoordinatorSleepReady bit ever on its passively coordinated NM channel but it has to forward a received bit on its actively coordinated channels.」()

[Nm261]「A passive NM Coordinator gets Indication of the Sleep Ready Bit set by `Nm_CoordReadyToSleepIndication` called by the <Bus>Nm.」()

Table 1: Control Bit Vector(CBV)

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 1	Reserved	Reserved	Reserved	Reserved	NM Coordinator Sleep Ready	Reserved R3.2 NM Coordinator ID (High Bit)	Reserved R3.2 NM Coordinator ID (Low Bit)	Repeat Message Request

[Nm262]「The active and passive NM Coordinators shall start a coordination delay timer after the NMcoordinatorSleepReady bit in the CBV has been set by the active NM Coordinator.」()

Nm263「The NM Coordinator shall start the *shutdown network* sequence, once the coordination delay timer has expired.」()

Note: *This can be either waiting for synchronization point or starting shutdown timers* immediately, *depending on the configuration of the coordinator. (see [Nm172])*

Note*: : the coordination delay timer defines the time left before the NM cluster starts to shut down its network thus the shutdown network sequence of the nested bus shall be finished before the active NM Coordinator has shut down the Network to ensure a synchronous shutdown. See Figure 7:Shutdown with Nm_CoordDelayTimer.*

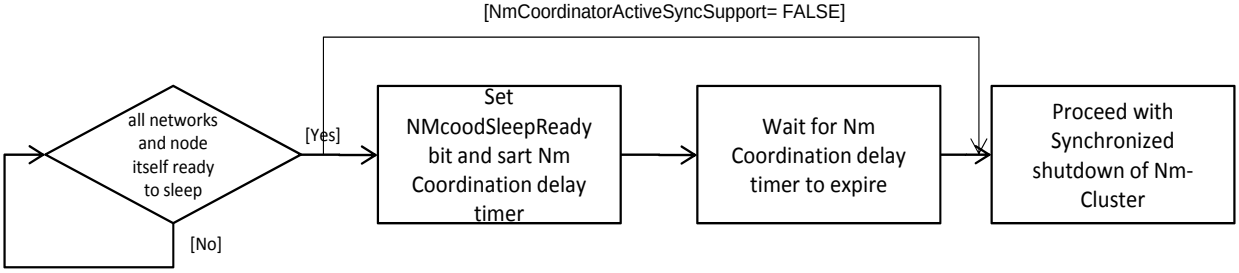

Figure 8: Setting the sleep ready bit

[**Nm264**]「The value of the coordination delay timer needs to be calculated from Nm_GlobalCoordinatorTime.」()

Note: *The coordination delay timer shall be calculated following: Nm_GlobalCoordinatorTime – minimum Nm-Cluster shutdowntime which the coordinated Nm_channel belongs to.(see Figure 7)*

[Nm265]「Nm_GlobalCoordinatorTime defines the maximum time needed to shut down all Networks coordinated. 」()

Note*:This includes all nested connections.(for example see Figure 7)*

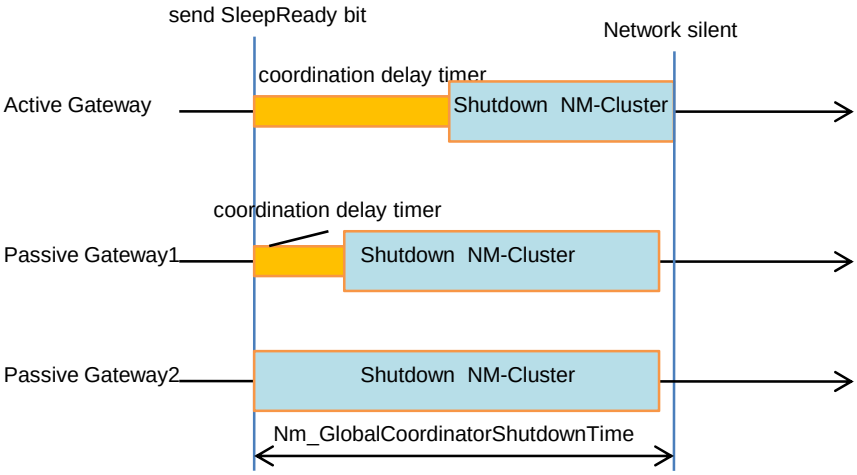

Figure 7: Shutdown with Nm_GlobalCoordinatorTime

[Nm266]「The active and passive NM Coordinator shall reset the coordination delay timer, if the network shutdown is aborted by an application or another network node requesting the network 」()

[Nm267]「If the network shutdown has been aborted for any reason, the NM Coordinator shall set the Sleep Ready bit back to the value 0 via `<BusNm>_SetSleepReadyBit(Nm_Channel, value)` on every actively coordinated channel.」()

7.3 Wakeup and aborted shutdown

Nm is not responsible for normal wakeup of the node or the networks; this is delegated to the COM Manager (**ComM**).

7.3.1 Normal wakeup

For both *Basic functionality* and *NM Coordination functionality*, **Nm** will forward wakeup indications from the networks (indicated by the bus specific NMs calling the callback `Nm_NetworkStartIndication()`) and forward this to the **ComM** by calling `ComM_Nm_NetworkStartIndication()`. **ComM** will then decide which action to take and will either call the `Nm_NetworkRequest()` or `Nm_PassiveStartup()`, which will be forwarded by **Nm** to the corresponding interface of the bus specific NM.

[Nm245] ⌈ If the **ComM** calls `Nm_PassiveStartup()` for a network that is part of a *coordinated cluster* of networks, the **Nm** coordinator functionality shall treat this call as if the **ComM** had called `Nm_NetworkRequest()`. The **Nm** shall forward a call of `<BusNm>_NetworkRequest()` to the lower layer. Accordingly, the network shall be counted as requested by the **NM** coordinator. ⌋()

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

7.3.2 Coordinated wakeup

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

7.3.3 Aborted shutdown

If the *NM Coordination functionality* is activated and coordinated shutdown has been initiated on an *NM Coordination Cluster*, dependent on the *coordinator algorithm* configuration it might take time before each included bus is actually released. If any node on one of the coordinated buses changes its state and starts requesting the network before all networks are released, race conditions can occur in the *coordination algorithm*. This can happen in four ways:

1. A node on a network that has not yet been released and is still in 'network mode' starts requesting the network again. This will be detected by the bus specific NM which will inform **Nm** by calling `Nm_RemoteSleepCancellation()`.

2. A node on a network that has already been released and has indicated 'prepare bus-sleep mode' but not 'bus-sleep mode' starts requesting the network again. This will be detected by the bus specific NM that will automatically change state to 'network mode' and inform **Nm** by calling `Nm_NetworkMode()`.

3. A node on a network that has already been released and has indicated 'bus-sleep mode' wakes up. This will be detected by the bus specific NM just as in the normal case. The bus specific NM will not change state automatically but will inform **Nm** by calling `Nm_NetworkStartIndication()`.

4. The **ComM** requests the network on any of the networks in the *NM Coordination Cluster*.

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

[Nm181] ⌈ The coordinated shutdown shall be aborted if any network in that *NM Coordination Cluster*,
- indicates `Nm_RemoteSleepCancelation()`
- indicates `Nm_NetworkMode()`
- indicates `Nm_NetworkStartIndication()`

or the **ComM** request one of the networks with `Nm_NetworkRequest()`or `Nm_PassiveStartup()` ⌋()

[Nm182] ⌈ If the shutdown is aborted, NM Coordinator shall call `ComM_Nm_RestartIndication()` for all networks that already indicated 'bus sleep'. ⌋()

***Rationale**: Since **Nm** cannot take decision to wake networks on its own, this must be forwarded to **ComM** just as in the normal wakeup case.*

[Nm183] ⌈ If the coordinated shutdown is aborted, NM Coordinator shall request the network from the <busNm's> for the network that have not indicated 'bus sleep'. ⌋()

[Nm185] ⌈ If a coordinated shutdown has been aborted, all conditions that guards the initiation of the coordinated shutdown shall be evaluated again ⌋()

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

[Nm235] ⌈ If a coordinated shutdown has been aborted and **Nm** receives `NM_E_NOT_OK` on a `<BusNm>_NetworkRequest()`, that network shall not be considered awake when the conditions for imitating shutdown are evaluated again. ⌋()

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

[Nm236] ⌈ If a coordinated shutdown has been initiated and **Nm** receives `NM_E_NOT_OK` on a `<BusNm>_NetworkRelease()`, the shutdown shall be immediately aborted. For all networks that have not entered 'bus-sleep mode', **Nm** shall request

the networks. This includes the network that indicated an error for `<BusNm>_NetworkRelease()`. As soon as this has been done, the conditions for initiating coordinated shutdown can be evaluated again. This applies also to networks that were not actively participating in the current coordinated shutdown. ⌋()

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

7.4 Prerequisites of bus specific Network Management modules

This chapter will give an overview of the API calls that is used for the *Basic functionality* and the *NM Coordination functionality* as well as information on the expected behavior of the bus specific NM for both functionalities.

For specific requirement of the interfaces, refer to Chapter 8.

7.4.1 Prerequisites for Basic functionality

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

The *Basic functionality* will provide the following API calls to the **ComM**:

- `Nm_NetworkRequest()` – Nm032
- `Nm_NetworkRelease()` – Nm046
- `Nm_PassiveStartup()` – Nm031

***Note**: This implies that the bus specific NM will provide the corresponding functions `<BusNm>_NetworkRequest()`, `<BusNm>_NetworkRelease()` and `<BusNm>_PassiveStartup()`.*

The Basic f*unctionality* will forward the following API callbacks to the **ComM**:

- `Nm_NetworkStartIndication()` – Nm154
- `Nm_NetworkMode()` – Nm156
- `Nm_BusSleepMode()` – Nm162
- `Nm_PrepareBusSleepMode()` – Nm159

***Note**: This implies that the **ComM** will provide the corresponding callback functions `ComM_Nm_NetworkStartIndication()`, `ComM_Nm_NetworkMode()`, `ComM_Nm_BusSleepMode()` and `ComM_Nm_PrepareBusSleepMode()`.*

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

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

- `Nm_DisableCommunication()` – Nm033
- `Nm_EnableCommunication()` – Nm034
- `Nm_SetUserData()` – Nm035
- `Nm_GetUserData()` – Nm036
- `Nm_GetPduData()` – Nm037
- `Nm_RepeatMessageRequest()` – Nm038
- `Nm_GetNodeIdentifier()` – Nm039
- `Nm_GetLocalNodeIdentifier()` – Nm040
- `Nm_CheckRemoteSleepIndication()` – Nm042
- `Nm_GetState()` – Nm043

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

7.4.2 Prerequisites for NM Coordinator functionality

The *coordination algorithm* will make use of the following interfaces of the bus specific NM:

- `<BusNm>_NetworkRequest()` – Nm119
- `<BusNm>_NetworkRelease()` – Nm119
- `<BusNm>_RequestBusSynchronization()` – Nm119
- `<BusNm>_CheckRemoteSleepIndication()` – Nm119

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

***Rationale**: The `BusNm_CheckRemoteSleepIndication()` is never explicitly mentioned in the coordination algorithm. Its use is dependent on the implementation.*

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

- `Nm_NetworkStartIndication()` – Nm154
- `Nm_NetworkMode()` – Nm156
- `Nm_BusSleepMode()` – Nm162
- `Nm_PrepareBusSleepMode()` – Nm159

- `Nm_RemoteSleepIndication()` – Nm192
- `Nm_RemoteSleepCancellation()` – Nm193

- `Nm_SynchronizationPoint()` – Nm194

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

***Note**: The `Nm_RemoteSleepIndication()` and `Nm_RemoteSleepCancellation()` are used by the coordination algorithm to determine when all conditions for initiating the coordinated shutdown are met. The indication will be called by the bus specific NM when it detects that all other nodes on the network (except for itself) is ready to go to 'bus-sleep mode'. Some implementations will also make use of the API call `BusNm_CheckRemoteSleepIndication()`.*

[Nm186] ⌈ A bus specific NM which is included in a *coordination cluster* must monitor its bus to identify when all other nodes on the network is ready to go to sleep. When this occurs, the bus specific NM shall call the callback `Nm_RemoteSleepIndication()` of **Nm**. (See Nm192). ⌋()

[Nm187] ⌈ After a bus specific NM which is included in a *coordination cluster* has signaled to **Nm** that all other nodes on the network is ready to go to sleep (See Nm192), it must continue monitoring its bus to identify if any node starts requesting the network again, implying that the bus is no longer ready to go to sleep. When this occurs, the bus specific NM shall call the callback `Nm_RemoteSleepCancellation()` of **Nm**. (See Nm193). ⌋()

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

***Rationale**: The `Nm_SynchronizationPoint()` shall be called by the bus specific NM in order to inform the* coordination algorithm *of a suitable point in time to initiate the coordinated shutdown. For cyclic networks this is typically at cycle boundaries. For non-cyclic networks this must be defined by other means. Each* NM Coordination Cluster *can be configured to make use of synchronization indications or not (See Nm172), and if they are used, the* coordination algorithm *will filter indications and only act on indications from networks that are configured as* synchronizing networks*.*

[Nm188] ⌈ Cyclic networks shall invoke the `Nm_SynchronizationPoint()` repeatedly when no other nodes request the network. The invocation shall typically be made at boundaries in the bus specific NM protocol when changes in the NM voting will occur. ⌋()

It shall be assumed that any call to `BusNm_ReleaseNetwork()` made between two of these `Nm_SynchronizationPoints()` will be acted upon at the same point in time as the next `Nm_SynchronizationPoint()` would have been invoked.

***Rationale**: The synchronization indication shall start when `Nm_RemoteSleepIndication()` has been notified and continue until either the network has been released (`BusNm_NetworkRelease()`) or the `Nm_RemoteSleepCancellation()` is called.*

7.4.3 Configuration of global parameters for bus specific networks

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

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

7.5 Additional Functionality

7.5.1 Nm_CarWakeUpIndication

[Nm252] 「If the <bus>Nm calls Nm_CarWakeUpIndication, the NM Interface shall call the callback function defined by NmCarWakeUpCallback with nmNetworkHandle as parameter. 」()

[Nm253] 「The application, called by NmCarWakeUpCallback**,** is responsible to manage the Car Wake Up (CWU) request and distribute the Request to other Nm channels by setting the CWU bit in its own Nm message. This application has to drop the CWU request if the request is not repeated within a specific time. 」()

7.6 Error classification

[Nm125] 「 Values for production code Event Ids are assigned externally by the configuration of the Dem. They are published in the file Dem_IntErrId.h and included via Dem.h. 」()

[Nm126] 「Development error values are of type uint8. 」()

[Nm232] :「
The Nm shall be able to detect the following errors and exceptions depending on its configuration (development/production):

Type or error	*Relevance*	*Related error code*	*Value [hex]*
API service used without Nm interface initialization	Development	NM_E_UNINIT	0x00
API Service called with wrong parameter but not with NULL-pointer	Development	NM_E_HANDLE_UNDEF	0x01
`API service called with a NULL pointer`	Development	NM_E_PARAM_POINTER	0x02

」(BSW00385, BSW00327)

7.7 Error detection

[Nm022] 「 The detection of development errors is configurable (*ON* / *OFF*) at pre-compile time. The configuration switch `NmDevErrorDetect` (see Chapter 10) shall activate or deactivate the detection of all development errors. 」(BSW00323, BSW00386, BSW00350, BSW150)

[Nm023] 「 If the `NmDevErrorDetect` switch is enabled, API parameter checking shall be enabled. 」(BSW00386)

7.8 Error notification

[Nm025] 「 Detected development errors shall be reported to the `Det_ReportError` service of the **Development Error Tracer** (**DET**) if the pre-processor switch `NmDevErrorDetect` is set, except if the respective service is implemented as macro (see Nm091). 」(BSW150)

***Note**: Even if the **Nm** is implemented as a macro, it is still allowed but not required, to implement error reporting to DET if possible.*

[Nm026] 「 Production errors shall be reported to **Diagnostic Event Manager** (**DEM**) if and only if they originate from the *NM Coordinator* functionality. Otherwise, it is assumed that the underlying bus specific NM module (e.g. **FrNm**.) has already reported to **DEM**. 」(BSW00339)

[Nm233] ⌈
If the pre-processor switch `NmDevErrorDetect` is set, all function calls containing a `NetworkHandleType` parameter shall raise the error `NM_E_HANDLE_UNDEF` if the network parameter is not a configured network handle. ⌋()

[Nm248] ⌈ In case an API is called with NULL-pointer as parameter,
the API service shall return immediately without any further action, beside reporting `NM_E_PARAM_POINTER`. ⌋()

7.9 Debugging

[Nm189] ⌈ Each variable that shall be accessible by AUTOSAR Debugging, shall be defined as global variable. ⌋()

[Nm190] ⌈ All type definitions of variables that shall be debugged shall be accessible by the header file Nm.h. ⌋()

[Nm191] ⌈ The declaration of variables in the header file shall be such, that it is possible to calculate the size of the variables by C-"sizeof". ⌋()

[Nm240] ⌈ If the configuration switch `NmCoordinatorSupportEnabled` is set, the internal states of the NM coordinator shall be available for debugging. ⌋()

***Rationale**: The internal state of the Nm coordinator indicate the state of the coordinated networks.*
***Caveat**:The representation of this states is implementation specific.*

8 API specification

8.1 Imported types

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

[Nm117] ⌈

Module	*Imported Type*
Com	Com_SignalIdType
ComStack_Types	NetworkHandleType
Dem	Dem_EventIdType
	Dem_EventStatusType
Std_Types	Std_ReturnType
	Std_VersionInfoType

⌋(BSW00301)

8.2 Type definitions

The following NM Stack types are specified and shall be defined in NmStack_types.h:

8.2.1 Nm_ModeType

Name:	Nm_ModeType	
Type:	Enumeration	
Range:	NM_MODE_BUS_SLEEP	Bus-Sleep Mode
	NM_MODE_PREPARE_BUS_SLEEP	Prepare-Bus Sleep Mode
	NM_MODE_SYNCHRONIZE	Synchronize Mode
	NM_MODE_NETWORK	Network Mode
Description:	Operational modes of the network management.	

8.2.2 Nm_StateType

Name:	Nm_StateType	
Type:	Enumeration	
Range:	NM_STATE_UNINIT	Uninitialized State (0)
	NM_STATE_BUS_SLEEP	Bus-Sleep State (1)
	NM_STATE_PREPARE_BUS_SLEEP	Prepare-Bus State (2)
	NM_STATE_READY_SLEEP	Ready Sleep State (3)
	NM_STATE_NORMAL_OPERATION	Normal Operation State (4)

	NM_STATE_REPEAT_MESSAGE	Repeat Message State (5)
	NM_STATE_SYNCHRONIZE	Synchronize State (6)
Description:	States of the network management state machine.	

8.2.3 Nm_BusNmType

Name:	Nm_BusNmType	
Type:	Enumeration	
Range:	NM_BUSNM_CANNM	CAN NM type
	NM_BUSNM_FRNM	FR NM type
	NM_BUSNM_LINNM	LIN NM type
	NM_BUSNM_UDPNM	UDP NM type
	NM_BUSNM_GENERICNM	Generic NM type
	NM_BUSNM_UNDEF	NM type undefined; it shall be defined as FFh
Description:	BusNm Type	

8.3 Function definitions

8.3.1 Standard services provided by NM Interface

8.3.1.1 Nm_Init

[Nm030] 「

Service name:	Nm_Init

Syntax:	`void Nm_Init(`
	`  void`
	`)`

Service ID[hex]:	0x00
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	Initializes the NM Interface.

」(BSW00344, BSW00405, BSW101, BSW00358, BSW00414)

[Nm127] 「 Caveats of `Nm_Init`: This service function has to be called after the initialization of the respective bus interface. 」()

8.3.1.2 Nm_PassiveStartUp

[Nm031] ⌈

Service name:	Nm_PassiveStartUp	
Syntax:	`Std_ReturnType Nm_PassiveStartUp( const NetworkHandleType NetworkHandle )`	
Service ID[hex]:	0x01	
Sync/Async:	Synchronous	
Reentrancy:	Non-reentrant for the same NetworkHandle, reentrant otherwise	
Parameters (in):	NetworkHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: No error E_NOT_OK: Passive start of network management has failed NetworkHandle does not exist (development only) Module not yet initialized (development only)
Description:	This function calls the <BusNm>_PassiveStartUp function (e.g. CanNm_PassiveStartUp function is called if channel is configured as CAN).	

⌋(BSW151, BSW02513, BSW046, BSW051)

[Nm128] ⌈ Caveats of `Nm_PassiveStartUp`: The **<BusNm>** and the **Nm** itself are initialized correctly. ⌋()

8.3.1.3 Nm_NetworkRequest

[Nm032] ⌈

Service name:	Nm_NetworkRequest	
Syntax:	`Std_ReturnType Nm_NetworkRequest( const NetworkHandleType NetworkHandle )`	
Service ID[hex]:	0x02	
Sync/Async:	Synchronous	
Reentrancy:	Non-reentrant for the same NetworkHandle, reentrant otherwise	
Parameters (in):	NetworkHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: No error E_NOT_OK: Requesting of bus communication has failed NetworkHandle does not exist (development only) Module not yet initialized (development only)
Description:	This function calls the <BusNm>_NetworkRequest (e.g. CanNm_NetworkRequest function is called if channel is configured as CAN).	

」(BSW02513, BSW046, BSW047, BSW051)

[Nm129] 「 Caveats of `Nm_NetworkRequest`: The **<BusNm>** and the **Nm** itself are initialized correctly. 」()

[Nm130] 「 Configuration of `Nm_NetworkRequest`: This function is only available if `NmPassiveModeEnabled` is set to `FALSE`. 」()

8.3.1.4 Nm_NetworkRelease

[Nm046] 「

Service name:	Nm_NetworkRelease	
Syntax:	`Std_ReturnType Nm_NetworkRelease( const NetworkHandleType NetworkHandle )`	
Service ID[hex]:	0x03	
Sync/Async:	Synchronous	
Reentrancy:	Non-reentrant for the same NetworkHandle, reentrant otherwise	
Parameters (in):	NetworkHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: No error E_NOT_OK: Releasing of bus communication has failed NetworkHandle does not exist (development only) Module not yet initialized (development only)
Description:	This function calls the <BusNm>_NetworkRelease bus specific function (e.g. CanNm_NetworkRelease function is called if channel is configured as CAN).	

」(BSW048, BSW051)

[Nm131] 「 Caveats of `Nm_NetworkRelease`: The **<BusNm>** and the **Nm** itself are initialized correctly. 」()

[Nm132] 「 Configuration of `Nm_NetworkRelease`: This function is only available if `NmPassiveModeEnabled` is set to `FALSE`. 」()

8.3.2 Communication control services provided by NM Interface

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

8.3.2.1 Nm_DisableCommunication

[Nm033] ⌈

Service name:	Nm_DisableCommunication	
Syntax:	`Std_ReturnType Nm_DisableCommunication( const NetworkHandleType NetworkHandle )`	
Service ID[hex]:	0x04	
Sync/Async:	Synchronous	
Reentrancy:	Non-reentrant for the same NetworkHandle, reentrant otherwise	
Parameters (in):	NetworkHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: No error E_NOT_OK: Disabling of NM PDU transmission ability has failed. NetworkHandle does not exist (development only) Module not yet initialized (development only)
Description:	Disables the NM PDU transmission ability. For that purpose <BusNm>_DisableCommunication shall be called (e.g. CanNm_DisableCommunication function is called if channel is configured as CAN).	

⌋(BSW02513, BSW02512)

[Nm133] ⌈ Caveats of `Nm_DisableCommunication`: The **<BusNm>** and the **Nm** itself are initialized correctly. ⌋()

[Nm134] ⌈ Configuration of `Nm_DisableCommunication`: This function is only available if `NmComControlEnabled` is set to `TRUE`. ⌋()

8.3.2.2 Nm_EnableCommunication

[Nm034] ⌈

Service name:	Nm_EnableCommunication	
Syntax:	`Std_ReturnType Nm_EnableCommunication( const NetworkHandleType NetworkHandle )`	
Service ID[hex]:	0x05	
Sync/Async:	Synchronous	
Reentrancy:	Non-reentrant for the same NetworkHandle, reentrant otherwise	
Parameters (in):	NetworkHandle	Identification of the NM-channel
Parameters (inout):	None	

Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: No error E_NOT_OK: Enabling of NM PDU transmission ability has failed. NetworkHandle does not exist (development only) Module not yet initialized (development only)
Description:	Enables the NM PDU transmission ability. For that purpose <BusNm>_EnableCommunication shall be called (e.g. CanNm_EnableCommunication function is called if channel is configured as CAN).	

」(BSW047, BSW02512)

[Nm135] 「 Caveats of `Nm_EnableCommunication`: The **<BusNm>** and the **Nm** itself are initialized correctly. 」()

[Nm136] 「 Configuration of `Nm_EnableCommunication`: This function is only available if `NmComControlEnabled` is set to `TRUE`. 」()

8.3.3 Extra services provided by NM Interface

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

8.3.3.1 Nm_SetUserData

[Nm035] 「

Service name:	Nm_SetUserData	

Syntax:	`Std_ReturnType Nm_SetUserData(`
	`  const NetworkHandleType NetworkHandle,`
	`  const uint8 * const nmUserDataPtr`
	`)`	

Service ID[hex]:	0x06	
Sync/Async:	Synchronous	
Reentrancy:	Non-reentrant for the same NetworkHandle, reentrant otherwise	
Parameters (in):	NetworkHandle	Identification of the NM-channel
	nmUserDataPtr	User data for the next transmitted NM message
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: No error E_NOT_OK: Setting of user data has failed NetworkHandle does not exist (development only) Module not yet initialized (development only)
Description:	Set user data for NM messages transmitted next on the bus. For that purpose <BusNm>_SetUserData shall be called (e.g.	

	CanNm_SetUserData function is called if channel is configured as CAN).

」(BSW02503)

[Nm137] 「 Caveats of `Nm_SetUserData`: The **<BusNm>** and the **Nm** itself are initialized correctly. 」()

[Nm138] 「 Configuration of `Nm_SetUserData`: This function is only available if `NmUserDataEnabled` is set to `TRUE` and `NmPassiveModeEnabled` is set to `FALSE`. 」()

[Nm241] 「 Configuration of `Nm_SetUserData`: If `NmComUserDataSupport` is `True` the API `Nm_SetUserData` shall not be available. 」()

8.3.3.2 Nm_GetUserData

[Nm036] 「

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

」(BSW02504)

[Nm139] 「 Caveats of `Nm_GetUserData`: The **<BusNm>** and the **Nm** itself are initialized correctly. 」()

[Nm140] 「 Configuration of `Nm_GetUserData`: This function is only available if `NmUserDataEnabled` is set to `TRUE`. 」()

8.3.3.3 Nm_GetPduData

[Nm037] ⌈

Service name:	Nm_GetPduData	

Syntax:	`Std_ReturnType Nm_GetPduData(`
	`  const NetworkHandleType NetworkHandle,`
	`  uint8 * const nmPduData`
	`)`	

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

⌋(BSW02506)

[Nm141] ⌈ Caveats of `Nm_GetPduData`: The **<BusNm>** and the **Nm** itself are initialized correctly. ⌋()

[Nm142] ⌈ Configuration of `Nm_GetPduData`: This function is only available if `NmNodeIdEnabled` or `NmUserDataEnabled` is set to `TRUE`. ⌋()

8.3.3.4 Nm_RepeatMessageRequest

[Nm038] ⌈

Service name:	Nm_RepeatMessageRequest	

Syntax:	`Std_ReturnType Nm_RepeatMessageRequest(`
	`  const NetworkHandleType NetworkHandle`
	`)`	

Service ID[hex]:	0x09	
Sync/Async:	Synchronous	
Reentrancy:	Non-reentrant for the same NetworkHandle, reentrant otherwise	
Parameters (in):	NetworkHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: No error

		E_NOT_OK: Setting of Repeat Message Request Bit has failed NetworkHandle does not exist (development only) Module not yet initialized (development only)
Description:	Set Repeat Message Request Bit for NM messages transmitted next on the bus. For that purpose <BusNm>_RepeatMessageRequest shall be called (e.g. CanNm_RepeatMessageRequest function is called if channel is configured as CAN). This will force all nodes on the bus to transmit NM messages so that they can be identified.	

」(BSW153)

[Nm143] 「 Caveats of `Nm_RepeatMessageRequest`: **<BusNm>** and **Nm** itself are initialized correctly. 」()

[Nm144] 「 Configuration of `Nm_RepeatMessageRequest`: This function is only available if `NmNodeDetectionEnabled` is `TRUE`. 」()

8.3.3.5 Nm_GetNodeIdentifier

[Nm039] 「

Service name:	Nm_GetNodeIdentifier	

Syntax:	`Std_ReturnType Nm_GetNodeIdentifier(`
	`  const NetworkHandleType NetworkHandle,`
	`  uint8 * const nmNodeIdPtr`
	`)`	

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

」(BSW02505)

[Nm145] 「 Caveats of `Nm_GetNodeIdentifier`: The **<BusNm>** and the **Nm** itself are initialized correctly. 」()

[Nm146] ⌈ Configuration of `Nm_GetNodeIdentifier`: This function is only available if `NmNodeIdEnabled` is set to `TRUE`. ⌋()

8.3.3.6 Nm_GetLocalNodeIdentifier

[Nm040] ⌈

Service name:	Nm_GetLocalNodeIdentifier	
Syntax:	`Std_ReturnType Nm_GetLocalNodeIdentifier( const NetworkHandleType NetworkHandle, uint8 * const nmNodeIdPtr )`	
Service ID[hex]:	0x0b	
Sync/Async:	Synchronous	
Reentrancy:	Non-reentrant for the same NetworkHandle, reentrant otherwise	
Parameters (in):	NetworkHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	nmNodeIdPtr	Pointer where node identifier of the local node shall be copied to
Return value:	Std_ReturnType	E_OK: No error E_NOT_OK: Getting of the node identifier of the local node has failed NetworkHandle does not exist (development only) Module not yet initialized (development only)
Description:	Get node identifier configured for the local node. For that purpose <BusNm>_GetLocalNodeIdentifier shall be called (e.g. CanNm_GetLocalNodeIdentifier function is called if channel is configured as CAN).	

⌋(BSW02508)

[Nm147] ⌈ Caveats of `Nm_GetLocalNodeIdentifier`: The **<BusNm>** and the **Nm** itself are initialized correctly. ⌋()

[Nm148] ⌈ Configuration of `Nm_GetLocalNodeIdentifier`: This function is only available if `NmNodeIdEnabled` is set to `TRUE`. ⌋()

8.3.3.7 Nm_CheckRemoteSleepIndication

[Nm042] ⌈

Service name:	Nm_CheckRemoteSleepIndication
Syntax:	`Std_ReturnType Nm_CheckRemoteSleepIndication( const NetworkHandleType nmNetworkHandle,`

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

」()

[Nm149] 「 Caveats of `Nm_CheckRemoteSleepIndication`: The **<BusNm>** and the **Nm** itself are initialized correctly. 」()

[Nm150] 「 Configuration of `Nm_CheckRemoteSleepIndication`: This function is only available if `NmRemoteSleepIndEnabled` is set to `TRUE`. 」()

8.3.3.8 Nm_GetState

[Nm043] 「

Service name:	Nm_GetState	

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

Service ID[hex]:	0x0e	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	nmNetworkHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	nmStatePtr	Pointer where state of the network management shall be copied to
	nmModePtr	Pointer to the location where the mode of the network management shall be copied to
Return value:	Std_ReturnType	E_OK: No error

		E_NOT_OK: Getting of NM state has failed NetworkHandle does not exist (development only) Module not yet initialized (development only)
Description:	Returns the state of the network management. The function <BusNm>_GetState shall be called (e.g. CanNm_GetState function is called if channel is configured as CAN).	

」(BSW050)

[Nm151] 「 Caveats of `Nm_GetState`: The **<BusNm>** and the **Nm** itself are initialized correctly. 」()

8.3.3.9 Nm_GetVersionInfo

[Nm044] 「

Service name:	Nm_GetVersionInfo	

Syntax:	`void Nm_GetVersionInfo(`
	`  Std_VersionInfoType* nmVerInfoPtr`
	`)`	

Service ID[hex]:	0x0f	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	nmVerInfoPtr	Pointer to where to store the version information of this module.
Return value:	None	
Description:	This service returns the version information of this module.	

」(BSW00407, BSW003)

[Nm152] 「 The function `Nm_GetVersionInfo` shall return 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. 」()

[Nm153] 「 Configuration of `Nm_GetVersionInfo`: This function is only available if `NmVersionInfoApi` is set to `TRUE`. 」()

8.4 Call-back notifications

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

[Nm028] ⌈ All callbacks of the **Nm** shall assume that they can run either in task or in interrupt context. ⌋(BSW00333)

8.4.1 Standard Call-back notifications

8.4.1.1 Nm_NetworkStartIndication

[Nm154] ⌈

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

⌋(BSW00411)

[Nm155] ⌈ The indication through callback function `Nm_NetworkStartIndication` shall be forwarded to **ComM** by calling the `ComM_Nm_NetworkStartIndication`. ⌋()

8.4.1.2 Nm_NetworkMode

[Nm156] ⌈

Service name:	Nm_NetworkMode
Syntax:	`void Nm_NetworkMode( const NetworkHandleType nmNetworkHandle )`
Service ID[hex]:	0x12

Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	nmNetworkHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Notification that the network management has entered Network Mode.	

」()

[Nm158] 「 The indication through callback function `Nm_NetworkMode` shall be forwarded to **ComM** by calling the `ComM_Nm_NetworkMode`. 」()

8.4.1.3 Nm_BusSleepMode

[Nm162] 「

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

」()

[Nm163] 「 The indication through callback function `Nm_BusSleepMode` shall be forwarded to **ComM** by calling the `ComM_Nm_BusSleepMode`. 」()

8.4.1.4 Nm_PrepareBusSleepMode

[Nm159] 「

Service name:	Nm_PrepareBusSleepMode
Syntax:	`void Nm_PrepareBusSleepMode( const NetworkHandleType nmNetworkHandle )`
Service ID[hex]:	0x13
Sync/Async:	Synchronous

Reentrancy:	Reentrant	
Parameters (in):	nmNetworkHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Notification that the network management has entered Prepare Bus-Sleep Mode.	

」()

[Nm161] 「 The indication through callback function `Nm_PrepareBusSleepMode` shall be forwarded to **ComM** by calling the `ComM_Nm_PrepareBusSleepMode`. 」()

8.4.1.5 Nm_RemoteSleepIndication

[Nm192] 「

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

」()

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

8.4.1.6 Nm_RemoteSleepCancellation

[Nm193] 「

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

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

」()

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

8.4.1.7 Nm_SynchronizationPoint

[Nm194] :「

Service name:	Nm_SynchronizationPoint	
Syntax:	`void Nm_SynchronizationPoint( const NetworkHandleType nmNetworkHandle )`	
Service ID[hex]:	0x19	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	nmNetworkHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Notification to the NM Coordinator functionality that this is a suitable point in time to initiate the coordination algorithm on.	

」()

The notification that this is a suitable point in time to initiate the coordination algorithm on is only needed for internal purposes of the *NM Coordinator*.

8.4.2 Extra Call-back notifications

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

8.4.2.1 Nm_PduRxIndication

[Nm112] 「

Service name:	Nm_PduRxIndication
Syntax:	`void Nm_PduRxIndication( const NetworkHandleType nmNetworkHandle`

	)	
Service ID[hex]:	0x15	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	nmNetworkHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Notification that a NM message has been received.	

」()

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

[Nm164] 「 Configuration of `Nm_PduRxIndication`: This function is only available if `NmPduRxIndicationEnabled` is set to `TRUE`. 」()

8.4.2.2 Nm_StateChangeNotification

[Nm114] 「

Service name:	Nm_StateChangeNotification	
Syntax:	`void Nm_StateChangeNotification( const NetworkHandleType nmNetworkHandle, const Nm_StateType nmPreviousState, const Nm_StateType nmCurrentState )`	
Service ID[hex]:	0x16	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	nmNetworkHandle	Identification of the NM-channel
	nmPreviousState	Previous state of the NM-channel
	nmCurrentState	Current (new) state of the NM-channel
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Notification that the state of the lower layer <BusNm> has changed.	

」()

The notification that the state of the bus-specific NM has changed is only needed for OEM specific extensions of the *NM Coordinator*.

[Nm165] 「 Configuration of `Nm_StateChangeNotification`: This function is only available if `NmStateChangeIndEnabled` is set to `TRUE`. 」()

[Nm249] ⌈ When `NmStateReportEnabled` is set to `TRUE`, `Nm_StateChangeNotification` shall call `Com_SendSignal(uint8, Com_SignalIdType, const void*)` with `NmStateReportSignalRef` as `Com_signalIdType`. `NmStateReportSignalRef` points to a 6 bit signal, called Network Management State (NMS). The NMS needs to be configuered in Com. The NMS shall be set to the value according to the following table:

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

⌋()

8.4.2.3 Nm_RepeatMessageIndication

[Nm230]: ⌈

Service name:	Nm_RepeatMessageIndication	
Syntax:	`void Nm_RepeatMessageIndication( const NetworkHandleType nmNetworkHandle )`	
Service ID[hex]:	0x1a	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	nmNetworkHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Service to indicate that an NM message with set Repeat Message Request Bit has been received.	

⌋()

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

[Nm231]: ⌈ Configuration of `Nm_RepeatMessageIndication`: This function is only available if NmRepeatMsgIndEnabled is set to `TRUE`. ⌋()

8.4.2.4 Nm_TxTimeoutException

[Nm234] ⌈

Service name:	Nm_TxTimeoutException	
Syntax:	`void Nm_TxTimeoutException( const NetworkHandleType nmNetworkHandle )`	
Service ID[hex]:	0x1b	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	nmNetworkHandle	--
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Service to indicate that an attempt to send an NM message failed.	

⌋()

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

8.4.2.5 Nm_CarWakeUpIndication

[Nm250] ⌈

Service name:	Nm_CarWakeUpIndication	
Syntax:	`void Nm_CarWakeUpIndication( const NetworkHandleType nmChannelHandle )`	
Service ID[hex]:	0x1d	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	nmChannelHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	This function is called by a <Bus>Nm to indicate reception of a CWU request.	

()

[Nm251] ⌈Configuration of Nm_CarWakeUpIndication: Optional

If `NmCarWakeUpRxEnabled` is `TRUE`, The Nm shall provide the API Nm_CarWakeUpIndication (). ⌋()

8.4.2.6 Nm_CoordReadyToSleepIndication

[Nm254]「

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

()

[Nm255]「Configuration of Nm_CoordReadyToSleepIndication: Optional

If `NmCoordinatorSyncSupport` is set to `TRUE` , the Nm shall provide the API Nm_CoordReadyToSleepIndication.」()

8.5 Scheduled functions

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

[Nm020] 「 A scheduled main function shall only contain logic related to the *NM Coordinator functionality*. 」(BSW00373)

[Nm121] 「 In case the main function is called before the Nm has been initialized, the main function shall immediately return without yielding an error. 」(BSW00416)

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

8.5.1 Nm_MainFunction

[Nm118]「

Service name:	Nm_MainFunction
Syntax:	`void Nm_MainFunction( void )`
Service ID[hex]:	0x10

Timing:	FIXED_CYCLIC
Description:	This function implements the processes of the NM Interface, which need a fix cyclic scheduling.

」(BSW00424, BSW00425, BSW00376)

8.6 Expected Interfaces

This chapter lists all interfaces required from other modules.

8.6.1 Mandatory Interfaces

This chapter lists all interfaces required from other modules.

[Nm119] 「

API function	***Description***
<BusNm>_CheckRemoteSleepIndication	Check if remote sleep indication takes place or not.
<BusNm>_DisableCommunication	Disable the NM PDU transmission ability.
<BusNm>_EnableCommunication	Enable the NM PDU transmission ability.
<BusNm>_GetLocalNodeIdentifier	Get node identifier configured for the local node.
<BusNm>_GetNodeIdentifier	Get node identifier out of the last successfully received NM-message.
<BusNm>_GetPduData	Pointer where NM PDU shall be copied to.
<BusNm>_GetState	Returns the state and the mode of the network management.
<BusNm>_GetUserData	Get user data out of the last successfully received NM message.
<BusNm>_GetVersionInfo	This service returns the version information of this module.
<BusNm>_NetworkRelease	Release the network, since ECU doesn't have to communicate on the bus.
<BusNm>_NetworkRequest	Request the network, since ECU needs to communicate on the bus.
<BusNm>_PassiveStartup	Passive startup of the NM. It triggers the transition from Bus-Sleep Mode to the Network Mode without requesting the network.
<BusNm>_RepeatMessageRequest	Request a Repeat Message Request to be transmitted next on the bus.
<BusNm>_RequestBusSynchronization	Request bus synchronization.
<BusNm>_SetSleepReadyBit	Set the NM Coordinator Sleep Ready bit in the Control Bit Vector
<BusNm>_SetUserData	Set user data for NM messages transmitted next on the bus.
ComM_Nm_BusSleepMode	Notification that the network management has entered Bus-Sleep Mode. This callback function should perform a transition of the hardware and transceiver to bus-sleep mode.
ComM_Nm_NetworkMode	Notification that the network management has entered Network Mode.
ComM_Nm_NetworkStartIndication	Indication that a NM-message has been received in the Bus Sleep Mode, what indicates that some nodes in the network have already entered the Network Mode.

ComM_Nm_PrepareBusSleepMode	Notification that the network management has entered Prepare Bus-Sleep Mode. Reentrancy: Reentrant (but not for the same NM-Channel)
ComM_Nm_RestartIndication	If NmIf has started to shut down the coordinated busses, AND not all coordinated busses have indicated bus sleep state, AND on at least on one of the coordinated busses NM is restarted, THEN the NM Interface shall call the callback function ComM_Nm_RestartIndication with the nmNetworkHandle of the channels which have already indicated bus sleep state.

」()

8.6.2 Optional Interfaces

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

[Nm166] 「

API function	*Description*
Com_SendSignal	The service Com_SendSignal updates the signal object identified by SignalId with the signal referenced by the SignalDataPtr parameter.
Dem_ReportErrorStatus	Queues the reported events from the BSW modules (API is only used by BSW modules). The interface has an asynchronous behavior, because the processing of the event is done within the Dem main function.
Det_ReportError	Service to report development errors.

」()

8.6.3 Configurable Interfaces

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

This chapter is not applicable since the Nm does not expect any configurable interfaces other than those included to support generic lower layer bus NMs.

8.7 Version Check

[Nm246] 「

The Nm module shall perform Inter Module Checks to avoid integration of incompatible files.
The imported included files shall be checked by preprocessing directives.

The following version numbers shall be verified:

- <MODULENAME>_AR_RELEASE_MAJOR_VERSION
- <MODULENAME>_AR_RELEASE_MINOR_VERSION

」()
If the values are not identical to the expected values, an error shall be reported.

9 Sequence diagrams

9.1 Basic functionality

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

9.2 NM Coordinator functionality

Figure 6 shows the sequence diagram for the shutdown of network of the *NM Coordinator* functionality.

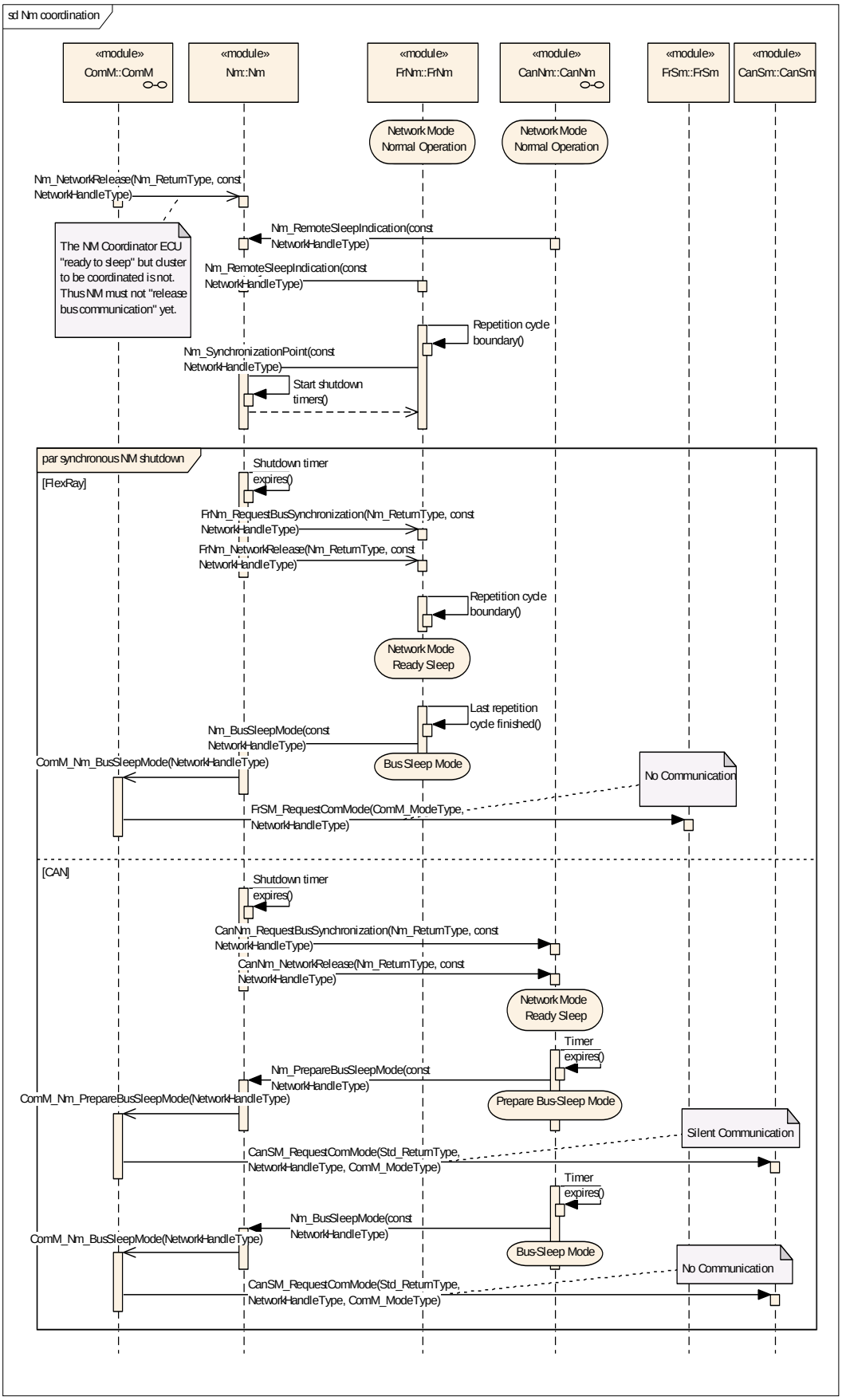

Figure 6 - Basic functionality sequence diagram

10 Configuration specification

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

In general, this chapter defines configuration parameters and their clustering into containers. Chapter 10.1 describes fundamentals. Chapter 10.2 specifies the variants used for configuration of the **Nm**. Chapter 10.3, 10.4 and 10.5 specifies the structure (containers) and the parameters of the **Nm**. Chapter 10.6 specifies published information of the **Nm**.

10.1 How to read this chapter

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

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

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

10.1.1 Configuration and configuration parameters

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

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

10.1.2 Variants

Variants describe sets of configuration parameters. E.g., variant 1: only pre-compile time configuration parameters, variant 2: mix of pre-compile- and post build time-configuration parameters. In one variant, a parameter can only be of one configuration class.

10.1.3 Containers

Containers structure the set of configuration parameters. This means:

- *all* configuration parameters are kept in containers.

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

10.1.4 Specification template for configuration parameters

The following tables consist of three sections:

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

10.1.4.1 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*.

10.1.4.2 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*.

10.1.4.3 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 Variants

10.2.1 VARIANT-PRE-COMPILE

[Nm120] ⌈ All configuration parameters are configurable at “Pre-compile time”.
Use case: Source code optimizations ⌋()

10.2.2 VARIANT-LINK-TIME

[Nm195] ⌈ All configuration parameters of the container `NmGlobalConfig` related to enable or disable an optional feature shall be configurable at “Pre-compile time”; the remaining configuration parameters shall be configured at “Link time”.
Use case: Object code libraries ⌋()

10.2.3 VARIANT-POST-BUILD

Not supported

10.3 Configuration parameters

The following Chapters summarize all configuration parameters for the Nm. The detailed meanings of most parameters are described in Chapter 7 and 8.

Note that the behavior and configuration of Nm is closely dependent on the behavior and configuration of the different bus specific NM modules used.

10.3.1 Nm

Module Name	*Nm*
Module Description	The Generic Network Management Interface module

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

10.4 Global configurable parameters

10.4.1 NmGlobalConfig

SWS Item	**Nm196_Conf :**
Container Name	NmGlobalConfig
Description	This container contains all global configuration parameters of the Nm Interface.
Configuration Parameters	

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

10.4.2 NmGlobalConstants

SWS Item	**Nm198_Conf :**
Container Name	NmGlobalConstants
Description	--
Configuration Parameters	

SWS Item	**Nm201_Conf :**		
Name	NmNumberOfChannels		
Description	Number of NM channels allowed within one ECU.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

No Included Containers

10.4.3 NmGlobalProperties

SWS Item	**Nm199_Conf :**
Container Name	NmGlobalProperties
Description	--
Configuration Parameters	

SWS Item	**Nm205_Conf :**
Name	NmCycletimeMainFunction
Description	The period between successive calls to the Main Function of the NM

	Interface in seconds.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	`0 .. INF`		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: Module dependency: Only valid if NmCoordinatorSupportEnabled is set to TRUE. If coordinator support is enabled and NmCycletimeMainFunction is not set, it shall be assumed that the specific implementation needs no main function.		

SWS Item	**Nm203_Conf :**		
Name	NmDevErrorDetect		
Description	Pre-processor switch for enabling development error detection and notification.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**Nm204_Conf :**		
Name	NmVersionInfoApi		
Description	Pre-processor switch for enabling Version Info API support.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

No Included Containers

10.4.4 NmGlobalFeatures

SWS Item	**Nm200_Conf :**
Container Name	NmGlobalFeatures
Description	--
Configuration Parameters	

SWS Item	**Nm208_Conf :**
Name	NmBusSynchronizationEnabled
Description	Pre-processor switch for enabling bus synchronization support of the <BusNm>s. This feature is required for NM Coordinator nodes only.
Multiplicity	1
Type	EcucBooleanParamDef

Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module dependency: It must not be enabled if NmPassiveModeEnabled is enabled.		

SWS Item	**Nm234_Conf :**		
Name	NmCarWakeUpCallback {NM_CAR_WAKE_UP_CALLBACK}		
Description	Name of the callback function to be called if Nm_CarWakeUpIndication() is called.		
Multiplicity	0..1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: Module dependency: only available if NmCarWakeUpRxEnabled == TRUE		

SWS Item	**Nm235_Conf :**		
Name	NmCarWakeUpRxEnabled		
Description	Enables or disables CWU detection. FALSE - CarWakeUp not supported TRUE - CarWakeUp supported		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**Nm210_Conf :**		
Name	NmComControlEnabled		
Description	Pre-processor switch for enabling the Communication Control support.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**Nm230_Conf :**
Name	NmComUserDataSupport
Description	Enable/Disable setting of NMUserData via SW-C. If NmComUserDataSupport is enabled the API Nm_SetUserData shall not be available.
Multiplicity	1
Type	EcucBooleanParamDef
Default value	false

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

SWS Item	**Nm206_Conf :**		
Name	NmCoordinatorSupportEnabled		
Description	Pre-processor switch for enabling NM Coordinator support.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module dependency: Only valid if NmRemoteSleepIndEnabled AND NmNumberOfChannels > 1		

SWS Item	**Nm237_Conf :**		
Name	NmGlobalCoordinatorTime		
Description	This parameter defines the maximum shutdown time of a connected and coordinated NM-Cluster. Note:This includes nested connections.		
Multiplicity	0..1		
Type	EcucFloatParamDef		
Range	`0 .. INF`		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Channel dependency: If the parameter NmBusSynchronizationEnabled is set to true this feature is available.		

SWS Item	**Nm212_Conf :**		
Name	NmNodeDetectionEnabled {NM_NODE_DETECTION_ENABLED}		
Description	Pre-processor switch for enabling the Node Detection feature.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module dependency: Only valid if NmNodeIdEnabled is set to TRUE		

SWS Item	**Nm213_Conf :**		
Name	NmNodeIdEnabled		
Description	Pre-processor switch for enabling transmission of the source node identifier in NM messages.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	

Scope / Dependency	scope: Module

SWS Item	**Nm214_Conf :**		
Name	NmPduRxIndicationEnabled		
Description	Pre-processor switch for enabling the PDU Rx Indication.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**Nm207_Conf :**		
Name	NmRemoteSleepIndEnabled		
Description	Pre-processor switch for enabling Remote Sleep Indication support. This feature is required for NM Coordinator nodes only.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module dependency: It must not be enabled if NmPassiveModeEnabled is enabled.		

SWS Item	**Nm229_Conf :**		
Name	NmRepeatMsgIndEnabled		
Description	Pre-processor switch for enabling the Repeat Message Bit Indication.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**Nm215_Conf :**		
Name	NmStateChangeIndEnabled		
Description	Pre-processor switch for enabling the Network Management state change notification.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**Nm211_Conf :**
Name	NmUserDataEnabled
Description	Pre-processor switch for enabling User Data support.
Multiplicity	1
Type	EcucBooleanParamDef

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

No Included Containers

10.5 Channel configurable parameters

10.5.1 NmChannelConfig

SWS Item	**Nm197_Conf :**
Container Name	NmChannelConfig
Description	This container contains the configuration (parameters) of the bus channel(s). The channel parameter shall be harmonized within the whole communication stack.
Configuration Parameters	

SWS Item	**Nm236_Conf :**		
Name	NmActiveCoordinator		
Description	This parameter indicates whether a NM Coordinator is an active gateway (NmActiveCoordinator = TRUE) or a passive.		
Multiplicity	0..1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Channel dependency: If the parameter NmBusSynchronizationEnabled is set to true this feature is available.		

SWS Item	**Nm216_Conf :**		
Name	NmChannelId		
Description	This parameter holds the unique channel index value. The value shall be the same as the ComMChannelId of the ComMChannel referenced by NmComMChannelRef. Implementation Type: NetworkHandleType		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: ECU dependency: Shall be harmonized with channel IDs of the whole communication stack.		

SWS Item	**Nm227_Conf :**		
Name	NmChannelSleepMaster		
Description	This parameter shall be set to indicate if the sleep of this network can be absolutely decided by the local node only and that no other nodes can oppose that decision. If this parameter is set to TRUE, the Nm shall assume that the channel is always ready to go to sleep and that no callouts to Nm_RemoteSleepIndication or Nm_RemoteSleepCancellation will be made from the <BusNm> representing this channel. If this parameter is set to FALSE, the Nm shall not assume that the network is ready to sleep until a callout has been made to Nm_RemoteSleepCancellation.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: Channel dependency: If the parameter NmCoordClusterIndex is not defined, this parameter is not valid.		

SWS Item	**Nm221_Conf :**		
Name	NmCoordClusterIndex		
Description	If this parameter is undefined for a channel, the corresponding bus does not belong to an NM coordination cluster.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: Channel		

SWS Item	**Nm209_Conf :**		
Name	NmPassiveModeEnabled		
Description	Pre-processor switch for enabling support of Passive Mode of the <BusNm>s.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**Nm222_Conf :**		
Name	NmShutdownDelayTimer		
Description	This parameter defines the time in seconds which the NM Coordination algorithm shall delay the release of this channel with.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	`0 .. INF`		
Default value	--		

ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: Channel dependency: If the parameter NmCoordClusterIndex is not defined, this parameter is not valid.		

SWS Item	**Nm231_Conf :**		
Name	NmStateReportEnabled		
Description	Specifies if the NMS shall be set for the corresponding network. false: No NMS shall be set true: The NMS shall be set		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module dependency: only available if NM_STATE_CHANGE_IND_ENABLED is TRUE and <Bus>NmComUserDataSupport is configured		

SWS Item	**Nm223_Conf :**		
Name	NmSynchronizingNetwork		
Description	If this parameter is true, then this network is a synchronizing network for the NM coordination cluster which it belongs to. The network is expected to call Nm_SynchronizationPoint() at regular intervals.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: Channel dependency: If the parameter NmCoordClusterIndex is not defined, this parameter is not valid.		

SWS Item	**Nm217_Conf :**		
Name	NmComMChannelRef		
Description	Reference to the corresponding ComM Channel.		
Multiplicity	1		
Type	Reference to [ComMChannel]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**Nm232_Conf :**		
Name	NmStateReportSignalRef		
Description	Reference to the signal for setting the NMS by calling Com_SendSignal for the respective channel.		
Multiplicity	0..1		
Type	Reference to [ComSignal]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	

Scope / Dependency	scope: Module dependency: Signal must be configured in COM. Only available if NmStateReportEnabled == true

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
NmBusType	1	--

10.5.2 NmBusType

SWS Item	**Nm218_Conf :**
Choice container Name	NmBusType
Description	--

Container Choices		
Container Name	***Multiplicity***	***Scope / Dependency***
NmGenericBusNmConfig	1	--
NmStandardBusNmConfig	1	--

10.5.3 NmGenericBusNmConfig

SWS Item	**Nm225_Conf :**
Container Name	NmGenericBusNmConfig
Description	--
Configuration Parameters	

SWS Item	**Nm219_Conf :**		
Name	NmGenericBusNmPrefix		
Description	The prefix which identifies the generic <BusNm>. This will be used to determine the API name to be called by Nm for the provided interfaces of the <BusNm>. This string will used for the module prefix before the "_" character in the API call name.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: ECU		

No Included Containers

10.5.4 NmStandardBusNmConfig

SWS Item	**Nm226_Conf :**
Container Name	NmStandardBusNmConfig
Description	--
Configuration Parameters	

SWS Item	**Nm220_Conf :**		
Name	NmStandardBusType		
Description	Identifies the bus type of the channel for standard AUTOSAR <BusNm>s and is used to determine which set of API calls to be called by Nm for the <BusNm>s. Note: The Ethernet bus' NM is UdpNm !		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	NM_BUSNM_CANNM	CAN bus	
	NM_BUSNM_FRNM	FlexRay bus	
	NM_BUSNM_LINNM	LIN bus	
	NM_BUSNM_UDPNM	Ethernet bus (using UDP)	
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: ECU		

No Included Containers

10.6 Published Information

[Nm001_PI] 「 The standardized common published parameters as required by BSW00402 in the General Requirements on Basic Software Modules [2] shall be published within the header file of this module and need to be provided in the BSW Module Description. The according module abbreviation can be found in the List of Basic Software Modules [9]. 」()

Additional module-specific published parameters are listed below if applicable.

11 Changes during SWS Improvements by Technical Office for set 2

11.1 Deleted SWS Items

SWS Item	*Rationale*

11.2 Replaced SWS Items

SWS Item of Release 1	*replaced by SWS Item*	*Rationale*
Nm104	Nm154, Nm155	requirement made atomic
Nm105	Nm156, Nm157, Nm158	requirement made atomic
Nm106	Nm159, Nm160, Nm161	requirement made atomic
Nm107	Nm162, Nm163	requirement made atomic

11.3 Changed SWS Items

SWS Item	*Rationale*
Nm004	Explanation in footnote separated from requirement
Nm022	Changed to standard text

11.4 Added SWS Items

SWS Item	*Rationale*
Nm122	Standard requirement for code file structure
Nm123	Requirement for header file structure identified
Nm124	Requirement for header file structure identified
Nm125	Standard requirement for error classification
Nm126	Standard requirement for error classification
Nm127	Requirement ID for caveats
Nm128	Requirement ID for caveats
Nm129	Requirement ID for caveats
Nm130	Requirement ID for configuration
Nm131	Requirement ID for caveats
Nm132	Requirement ID for configuration
Nm133	Requirement ID for caveats
Nm134	Requirement ID for configuration
Nm135	Requirement ID for caveats
Nm136	Requirement ID for configuration
Nm137	Requirement ID for caveats
Nm138	Requirement ID for configuration
Nm139	Requirement ID for caveats
Nm140	Requirement ID for configuration
Nm141	Requirement ID for caveats
Nm142	Requirement ID for configuration

Nm143	Requirement ID for caveats
Nm144	Requirement ID for configuration
Nm145	Requirement ID for caveats
Nm146	Requirement ID for configuration
Nm147	Requirement ID for caveats
Nm148	Requirement ID for configuration
Nm149	Requirement ID for caveats
Nm150	Requirement ID for configuration
Nm151	Requirement ID for caveats
Nm152	Separated Requirement for Nm_GetVersionInfo
Nm153	Requirement ID for configuration
Nm164	Requirement ID for configuration
Nm165	Requirement ID for configuration
Nm166	Requirement ID for table containing expected optional interfaces

12 Changes to Release 3

12.1 Deleted SWS Items

SWS Item	*Rationale*
Nm053, Nm056,Nm049	OSEK NM is no longer explicitly supported.
Nm157, Nm160	Generic NM Interface cannot set requirement on what the bus specific networks behaviour.
Nm179, Nm180	Never introduced because of configuration changes.
Nm083	Redundant since requirement Nm123 already specifies which parameters that shall go into which file.

12.2 Replaced SWS Items

SWS Item of Release 1	*replaced by SWS Item*	*Rationale*
Nm004	Nm167 - Nm191	*NM Coordination functionality* is completely rewritten. This requirement is obsolete and replaced with various new requirements.
Nm115	Nm169	New requirement that better fits the new *NM Coordination functionality*.
Nm116	Nm181, Nm182, Nm183	The aborted shutdown scenario is completely rewritten.
Nm122	Nm247	Nm_Lcfg.c (for link-time configurable parameters) needed. Nm122 stated no given c-file structure at all.

12.3 Changed SWS Items

SWS Item	*Rationale*
Nm095	Requirement was previously not testable.
Nm091	Rewritten to increase clarity of requirement.
Nm051	Rewritten to explicitly support official AUTOSAR NMs and generic interfaces.
Nm002	Rewritten to remove implicit OSEK support and to support coordination clusters.
Nm138	`Nm_SetUserData` wrongfully depended on `NmPassiveModeEnabled`.
Nm025	Modified to not exclude development error reporting for macro implementations.
Nm123	Took out Nm_Lcfg.c here since it is no h-file
Nm124	added)
Nm152	removed Instance ID from GetVersionId structure

12.4 Added SWS Items

SWS Item	*Rationale*
Nm167 – Nm191	Completely rewritten the *NM Coordination functionality*.
Nm192 – Nm194	Missing and new callback functions.
Nm195	Added Variant 2 – Pre-compile time of NmGlobalConfig
Nm196_Conf –	Added requirement items to Configuration Parameters

Nm223_Conf, Nm225_Conf – Nm226_Conf	
Nm224	Added requirement to populate the NmCycleTimeMainFunction.
Nm227_Conf – Nm228	Added new configuration parameter to support Master-networks and corresponding requirement.
Nm229_Conf	Added new missing configuration parameter NmRepeatMsgIndEnabled.
Nm230 – Nm231	Added missing call-back API Nm_RepeatMessageIndication
Nm232 – Nm233	Added requirements to report errors when undefined network handle is passed as a parameter.
Nm234	Added missing call-back API Nm_TxTimeoutException
Nm240	Add requirement for Debugging
Nm241	Add requirement for NmComUserDataSupport
Nm242	Add requirement for optional usage of DEM header file
Nm243	Add requirement for optional usage of DET header file
Nm244	Add requirement for Coordination cluster configuration
Nm245	Add requirement for handling passiv started networks, part of a coordination cluster
Nm001_PI	Rework of Published Information
Nm248	Error code for NULL-Pointer as parameter
Nm250 – Nm267	Support of Coordination of nested sub-busses

13 Not applicable requirements

[Nm999]「These requirements are not applicable to this specification.」(BSW00404, BSW170, BSW00399, BSW00400, BSW00438, BSW00375, BSW00437, BSW168, BSW00423, BSW00426, BSW00427, BSW00428, BSW00429, BSW00431, BSW00432, BSW00433, BSW00434, BSW00336, BSW00422, BSW00417, BSW004, BSW00409, BSW161, BSW162, BSW005, BSW164, BSW00325, BSW00326, BSW007, BSW00413, BSW00347, BSW00307, BSW00314, BSW00436, BSW00361, BSW00328, BSW00312, BSW006, BSW00439, BSW00357, BSW00355, BSW00306, BSW00308, BSW00309, BSW00371, BSW00440, BSW00329, BSW009, BSW172, BSW010, BSW00321, BSW00341, BSW00334, BSW043, BSW052, BSW02509, BSW02511, BSW053, BSW137, BSW054, BSW142, BSW143, BSW144, BSW145, BSW146, BSW147, BSW02510)