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

Document Status	Final
Part of AUTOSAR Release	4.2.2

Document Change History

Release	Changed by	Description
4.2.2	AUTOSAR Release Management	• "‘Coordination algorithm"’ and "‘Coordinated shutdown"’ redefined • Make the CarWakeup feature available • Debugging support marked as obsolete • Editorial changes
4.2.1	AUTOSAR Release Management	• Corrections on the requirement tracing • Clarification at use of callback versus callout • Editorial changes
4.1.3	AUTOSAR Release Management	• Rework of wakeup and abortion of coordinated shutdown • Rework of coordination of nested sub-busses
4.1.2	AUTOSAR Release Managment	• Remove DEM usage • Correct multiplicity and dependency of configuration parameter • Corrections on RemoteSleepIndication feature • Corrections on MainFunction and coordinated shutdown • Formal correction on REQ Tags • Editorial changes • Removed chapter(s) on change documentation

4.1.1	AUTOSAR Administration	• Introduction J1939Nm • Merged and corrected calculation of delay timer for Coordination Algorithm • Correction of parametrization and Services for Coordinator Synchonization Algorithm • Moved Nm_Passive_Mode_Enabled Parameter back to global container
4.0.3	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
4.0.1	AUTOSAR Administration	• Release check added • DET Error Code for false Pointer added • ChannelID harmonized in COM-Stack • Nm-State-changes in Userdata via NmIf
3.1.4	AUTOSAR Administration	• Remove explicit support for OSEK NM from specification • NM Coordinator functionality reworked (chapter 7.2 and 7.2.4) • Debugging functionality added • Link time configuration variant introduced • Legal disclaimer revised
3.1.1	AUTOSAR Administration	Legal disclaimer revised
3.0.1	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 **Network Management Interface** module.

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

Additionally, this document describes the 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 **Network Management Interface** implementation can either support only Basic functionality or both Basic functionality and NM Coordinator functionality.

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

2 Acronyms and Abbreviations

The glossary below includes acronyms and abbreviations and terms relevant to the Network Management Interface module that are not included in the [1, AUTOSAR glossary].

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

Terms:	Definition:
Bus-Sleep Mode	Network mode where all interconnected communication controllers are in the sleep mode.
NM-Channel	Logical channel associated with the NM-cluster
NM-Cluster	Set of NM nodes coordinated with the use of the NM algorithm.
NM-Coordinator	A functionality of the Nm which allows coordination of network sleep for multiple NM Channels.
NM-Message	Packet of information exchanged for purposes of the NM algorithm.
NM-Timeout	Timeout in the NM algorithm that initiates transition into Bus-Sleep Mode.
NM User Data	Supplementary application specific piece of data that is attached to every NM message sent on the bus.
Node Identifier	Node address information exchanged for purposes of the NM algorithm.
Node Identifier List	List of Node Identifiers recognized by the NM algorithm.
Bus	Physical communication medium to which a NM node/ecu is connected to.
network	Entity of all NM nodes/ecus which are connected to the same bus.
channel	Logical bus to which the NM node/ecu is connected to.
Coordinated shutdown	Shutdown of two or more busses in a way that their shutdown is finished coinciding.
Coordination algorithm	Initiation of coordinated shutdown in case all conditions are met.

3 Related documentation

3.1 Input documents

[1] Glossary
AUTOSAR_TR_Glossary

[2] General Specification of Basic Software Modules
AUTOSAR_SWS_BSWGeneral

[3] Specification of CAN Network Management
AUTOSAR_SWS_CANNetworkManagement

[4] Specification of FlexRay Network Management
AUTOSAR_SWS_FlexRayNetworkManagement

[5] Specification of LIN Network Management
AUTOSAR_SWS_LINNetworkManagement

[6] Specification of UDP Network Management
AUTOSAR_SWS_UDPNetworkManagement

[7] Specification of Network Management for SAE J1939
AUTOSAR_SWS_SAEJ1939NetworkManagement

[8] General Requirements on Basic Software Modules
AUTOSAR_SRS_BSWGeneral

[9] Requirements on Network Management
AUTOSAR_SRS_NetworkManagement

3.2 Related specification

AUTOSAR provides a General Specification on Basic Software modules [2, SWS BSW General], which is also valid for the Generic Network Management Interface.

Thus, the specification SWS BSW General shall be considered as additional and required specification for the Generic Network Management Interface.

4 Constraints and assumptions

4.1 Limitations

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

Figure 4.1 shows a typical example of the AUTOSAR NM stack.

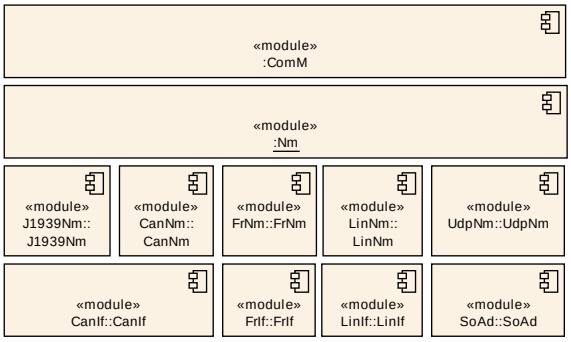

Figure 4.1: Nm stack modules

4.2 Specific limitations of the current release

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

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

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

4.3 Applicability to automotive domains

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

5 Dependencies to other modules

5.1 Interfaces to modules

Figure 5.1 shows the interfaces provided to and required from other modules in the AUTOSAR BSW.

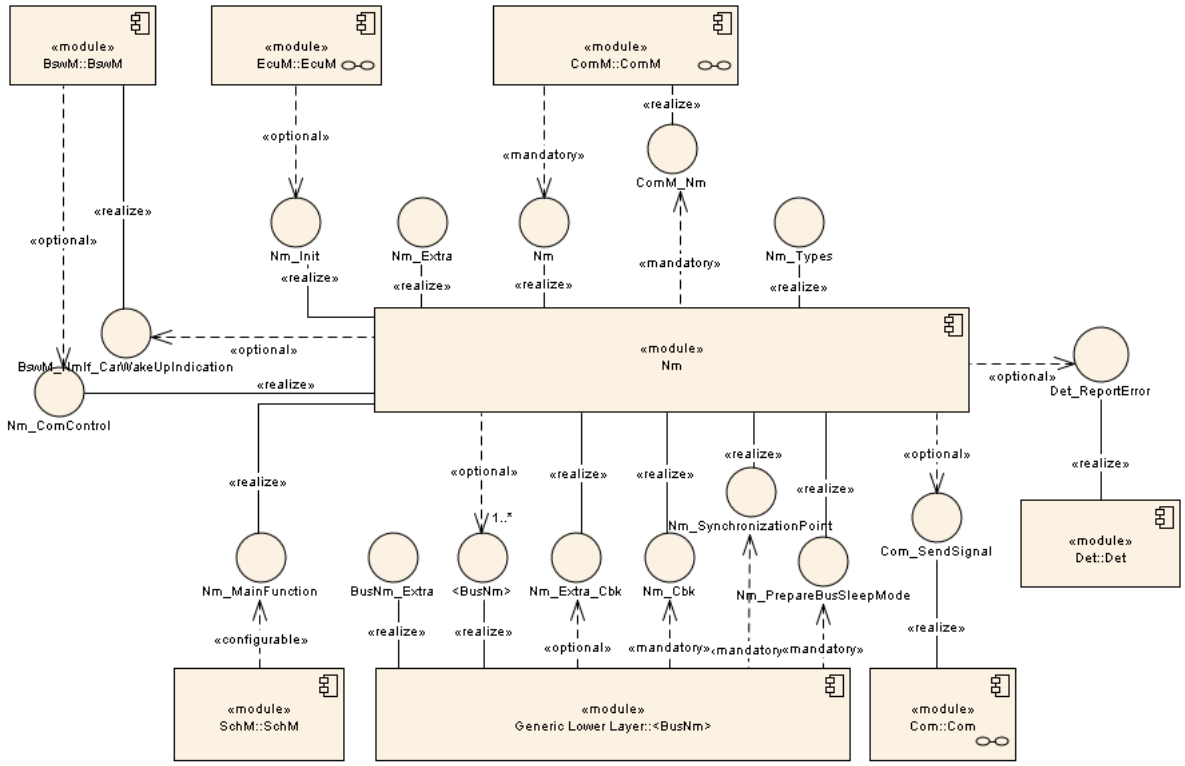

Figure 5.1: Nm's interfaces to other modules

5.1.1 ComM, CanNm, J1939Nm, FrNm, 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 ([3, **CanNm**])
- FlexRay Network Management ([4, **FrNm**])
- LIN Network Management ([5, **LinNm**])
- Ethernet Network Management ([6, **UdpNm**]).
- J1939 Network Management ([7, **J1939Nm**]).

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

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

5.1.2 Error handling modules (DET)

Nm reports development errors to the Development Error Tracer (**DET**) according to [SWS_Nm_00232].

5.1.3 BSW Scheduler

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

5.2 File structure

5.2.1 Code file structure

[SWS_Nm_00247] ⌈ The code file structure shall not be defined within this specification completely. At this point it shall be pointed out that the code-file structure shall include the following files named:

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

⌋ *(SRS_BSW_00159, SRS_BSW_00345, SRS_BSW_00419)*

5.2.2 Header file structure

[SWS_Nm_00123] ⌈ 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).

⌋(*SRS_BSW_00159*, *SRS_BSW_00345*, *SRS_BSW_00381*, *SRS_BSW_00419*)

[SWS_Nm_00124] ⌈ The following header files shall be included by the Nm Interface module:

- Std_Types.h (for AUTOSAR standard types)
  Note: Platform_Types.h (for platform specific types) and Compiler.h (for compiler specific language extensions) are indirectly included via AUTOSAR standard types.
- Nm_MemMap.h (for memory abstraction)
- SchM_Nm.h (for interfaces with the BSW Scheduler)
- ComM_Nm.h (for Communication Manager callback functions)
- BswM_Nm.h (If the BswM is used for CarWakeup-functionalitiy)
- <cdd>.h (If a CDD is used for CarWakeup-functionalitiy. The name of the CDD is generic.)

⌋(*SRS_BSW_00348*, *SRS_BSW_00353*, *SRS_BSW_00357*, *SRS_BSW_00381*, *SRS_BSW_00384*, *SRS_BSW_00412*)

[SWS_Nm_00243] ⌈ The Nm Interface shall optionally include the header file of DET (depending on the pre-processor switch NmDevErrorDetect, see ECUC_Nm_00203).

- Det.h for service of the Development Error Tracer.

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

6 Requirements traceability

The following tables references the requirements specified in [8] as well as [9] and links to the fulfillment of these.

Requirement	Description	Satisfied by
[SRS_BSW_00003]	All software modules shall provide version and identification information	[SWS_Nm_00044]
[SRS_BSW_00004]	All Basic SW Modules shall perform a pre-processor check of the versions of all imported include files	[SWS_Nm_00999]
[SRS_BSW_00005]	Modules of the μC Abstraction Layer (MCAL) may not have hard coded horizontal interfaces	[SWS_Nm_00999]
[SRS_BSW_00006]	The source code of software modules above the μC Abstraction Layer (MCAL) shall not be processor and compiler dependent.	[SWS_Nm_00999]
[SRS_BSW_00007]	All Basic SW Modules written in C language shall conform to the MISRA C 2004 Standard.	[SWS_Nm_00999]
[SRS_BSW_00009]	All Basic SW Modules shall be documented according to a common standard.	[SWS_Nm_00999]
[SRS_BSW_00010]	The memory consumption of all Basic SW Modules shall be documented for a defined configuration for all supported platforms.	[SWS_Nm_00999]
[SRS_BSW_00101]	The Basic Software Module shall be able to initialize variables and hardware in a separate initialization function	[SWS_Nm_00030] [SWS_Nm_00127] [SWS_Nm_00128] [SWS_Nm_00129] [SWS_Nm_00131] [SWS_Nm_00133] [SWS_Nm_00135] [SWS_Nm_00137] [SWS_Nm_00139] [SWS_Nm_00141] [SWS_Nm_00143] [SWS_Nm_00145] [SWS_Nm_00147] [SWS_Nm_00149] [SWS_Nm_00151]

Requirement	Description	Satisfied by
[SRS_BSW_00158]	All modules of the AUTOSAR Basic Software shall strictly separate configuration from implementation	[SWS_Nm_00999]
[SRS_BSW_00159]	All modules of the AUTOSAR Basic Software shall support a tool based configuration	[SWS_Nm_00123] [SWS_Nm_00247]
[SRS_BSW_00160]	Configuration files of AUTOSAR Basic SW module shall be readable for human beings	[SWS_Nm_00999]
[SRS_BSW_00161]	The AUTOSAR Basic Software shall provide a microcontroller abstraction layer which provides a standardized interface to higher software layers	[SWS_Nm_00999]
[SRS_BSW_00162]	The AUTOSAR Basic Software shall provide a hardware abstraction layer	[SWS_Nm_00999]
[SRS_BSW_00164]	The Implementation of interrupt service routines shall be done by the Operating System, complex drivers or modules	[SWS_Nm_00999]
[SRS_BSW_00167]	All AUTOSAR Basic Software Modules shall provide configuration rules and constraints to enable plausibility checks	[SWS_Nm_00999]
[SRS_BSW_00168]	SW components shall be tested by a function defined in a common API in the Basis-SW	[SWS_Nm_00999]
[SRS_BSW_00170]	The AUTOSAR SW Components shall provide information about their dependency from faults, signal qualities, driver demands	[SWS_Nm_00999]

Requirement	Description	Satisfied by
[SRS_BSW_00171]	Optional functionality of a Basic-SW component that is not required in the ECU shall be configurable at pre-compile-time	[SWS_Nm_00243]
[SRS_BSW_00172]	The scheduling strategy that is built inside the Basic Software Modules shall be compatible with the strategy used in the system	[SWS_Nm_00999]
[SRS_BSW_00300]	All AUTOSAR Basic Software Modules shall be identified by an unambiguous name	[SWS_Nm_00999]
[SRS_BSW_00301]	All AUTOSAR Basic Software Modules shall only import the necessary information	[SWS_Nm_00117] [SWS_Nm_00243]
[SRS_BSW_00302]	All AUTOSAR Basic Software Modules shall only export information needed by other modules	[SWS_Nm_00999]
[SRS_BSW_00304]	All AUTOSAR Basic Software Modules shall use the following data types instead of native C data types	[SWS_Nm_00999]
[SRS_BSW_00305]	Data types naming convention	[SWS_Nm_00999]
[SRS_BSW_00306]	AUTOSAR Basic Software Modules shall be compiler and platform independent	[SWS_Nm_00999]
[SRS_BSW_00307]	Global variables naming convention	[SWS_Nm_00999]
[SRS_BSW_00308]	AUTOSAR Basic Software Modules shall not define global data in their header files, but in the C file	[SWS_Nm_00999]
[SRS_BSW_00309]	All AUTOSAR Basic Software Modules shall indicate all global data with read-only purposes by explicitly assigning the const keyword	[SWS_Nm_00999]

Requirement	Description	Satisfied by
[SRS_BSW_00310]	API naming convention	[SWS_Nm_00999]
[SRS_BSW_00312]	Shared code shall be reentrant	[SWS_Nm_00999]
[SRS_BSW_00314]	All internal driver modules shall separate the interrupt frame definition from the service routine	[SWS_Nm_00999]
[SRS_BSW_00318]	Each AUTOSAR Basic Software Module file shall provide version numbers in the header file	[SWS_Nm_00999]
[SRS_BSW_00321]	The version numbers of AUTOSAR Basic Software Modules shall be enumerated according specific rules	[SWS_Nm_00999]
[SRS_BSW_00323]	All AUTOSAR Basic Software Modules shall check passed API parameters for validity	[SWS_Nm_00233]
[SRS_BSW_00325]	The runtime of interrupt service routines and functions that are running in interrupt context shall be kept short	[SWS_Nm_00999]
[SRS_BSW_00327]	Error values naming convention	[SWS_Nm_00232]
[SRS_BSW_00328]	All AUTOSAR Basic Software Modules shall avoid the duplication of code	[SWS_Nm_00999]
[SRS_BSW_00330]	It shall be allowed to use macros instead of functions where source code is used and runtime is critical	[SWS_Nm_00091]
[SRS_BSW_00331]	All Basic Software Modules shall strictly separate error and status information	[SWS_Nm_00999]
[SRS_BSW_00333]	For each callback function it shall be specified if it is called from interrupt context or not	[SWS_Nm_00028]

Requirement	Description	Satisfied by
[SRS_BSW_00334]	All Basic Software Modules shall provide an XML file that contains the meta data	[SWS_Nm_00999]
[SRS_BSW_00335]	Status values naming convention	[SWS_Nm_00999]
[SRS_BSW_00336]	Basic SW module shall be able to shutdown	[SWS_Nm_00999]
[SRS_BSW_00337]	Classification of development errors	[SWS_Nm_00232]
[SRS_BSW_00339]	Reporting of production relevant error status	[SWS_Nm_00999]
[SRS_BSW_00341]	Module documentation shall contains all needed informations	[SWS_Nm_00999]
[SRS_BSW_00342]	It shall be possible to create an AUTOSAR ECU out of modules provided as source code and modules provided as object code, even mixed	[SWS_Nm_00999]
[SRS_BSW_00343]	The unit of time for specification and configuration of Basic SW modules shall be preferably in physical time unit	[SWS_Nm_00999]
[SRS_BSW_00344]	BSW Modules shall support link-time configuration	[SWS_Nm_00030] [SWS_Nm_00195]
[SRS_BSW_00345]	BSW Modules shall support pre-compile configuration	[SWS_Nm_00120] [SWS_Nm_00123] [SWS_Nm_00247]
[SRS_BSW_00346]	All AUTOSAR Basic Software Modules shall provide at least a basic set of module files	[SWS_Nm_00999]
[SRS_BSW_00347]	A Naming seperation of different instances of BSW drivers shall be in place	[SWS_Nm_00999]
[SRS_BSW_00348]	All AUTOSAR standard types and constants shall be placed and organized in a standard type header file	[SWS_Nm_00124]

Requirement	Description	Satisfied by
[SRS_BSW_00350]	All AUTOSAR Basic Software Modules shall apply a specific naming rule for enabling/disabling the detection and reporting of development errors	[SWS_Nm_00999]
[SRS_BSW_00351]	Encapsulation of compiler specific methods to map objects	[SWS_Nm_00999]
[SRS_BSW_00353]	All integer type definitions of target and compiler specific scope shall be placed and organized in a single type header	[SWS_Nm_00124]
[SRS_BSW_00357]	For success/failure of an API call a standard return type shall be defined	[SWS_Nm_00124]
[SRS_BSW_00358]	The return type of init() functions implemented by AUTOSAR Basic Software Modules shall be void	[SWS_Nm_00030]
[SRS_BSW_00359]	All AUTOSAR Basic Software Modules callback functions shall avoid return types other than void if possible	[SWS_Nm_00112] [SWS_Nm_00114] [SWS_Nm_00154] [SWS_Nm_00156] [SWS_Nm_00159] [SWS_Nm_00162] [SWS_Nm_00192] [SWS_Nm_00193] [SWS_Nm_00194] [SWS_Nm_00230] [SWS_Nm_00234] [SWS_Nm_00250] [SWS_Nm_00254] [SWS_Nm_00272]
[SRS_BSW_00360]	AUTOSAR Basic Software Modules callback functions are allowed to have parameters	[SWS_Nm_00999]
[SRS_BSW_00361]	All mappings of not standardized keywords of compiler specific scope shall be placed and organized in a compiler specific type and keyword header	[SWS_Nm_00999]

Requirement	Description	Satisfied by
[SRS_BSW_00369]	All AUTOSAR Basic Software Modules shall not return specific development error codes via the API	[SWS_Nm_00233]
[SRS_BSW_00371]	The passing of function pointers as API parameter is forbidden for all AUTOSAR Basic Software Modules	[SWS_Nm_00999]
[SRS_BSW_00373]	The main processing function of each AUTOSAR Basic Software Module shall be named according the defined convention	[SWS_Nm_00020]
[SRS_BSW_00374]	All Basic Software Modules shall provide a readable module vendor identification	[SWS_Nm_00999]
[SRS_BSW_00375]	Basic Software Modules shall report wake-up reasons	[SWS_Nm_00999]
[SRS_BSW_00377]	A Basic Software Module can return a module specific types	[SWS_Nm_00999]
[SRS_BSW_00378]	AUTOSAR shall provide a boolean type	[SWS_Nm_00999]
[SRS_BSW_00379]	All software modules shall provide a module identifier in the header file and in the module XML description file.	[SWS_Nm_00999]
[SRS_BSW_00380]	Configuration parameters being stored in memory shall be placed into separate c-files	[SWS_Nm_00999]
[SRS_BSW_00381]	The pre-compile time parameters shall be placed into a separate configuration header file	[SWS_Nm_00123] [SWS_Nm_00124]
[SRS_BSW_00383]	The Basic Software Module specifications shall specify which other configuration files from other modules they use at least in the description	[SWS_Nm_00999]

Requirement	Description	Satisfied by
[SRS_BSW_00384]	The Basic Software Module specifications shall specify at least in the description which other modules they require	[SWS_Nm_00124] [SWS_Nm_00243]
[SRS_BSW_00385]	List possible error notifications	[SWS_Nm_00232]
[SRS_BSW_00386]	The BSW shall specify the configuration for detecting an error	[SWS_Nm_00232] [SWS_Nm_00233]
[SRS_BSW_00388]	Containers shall be used to group configuration parameters that are defined for the same object	[SWS_Nm_00999]
[SRS_BSW_00389]	Containers shall have names	[SWS_Nm_00999]
[SRS_BSW_00390]	Parameter content shall be unique within the module	[SWS_Nm_00999]
[SRS_BSW_00392]	Parameters shall have a type	[SWS_Nm_00999]
[SRS_BSW_00393]	Parameters shall have a range	[SWS_Nm_00999]
[SRS_BSW_00394]	The Basic Software Module specifications shall specify the scope of the configuration parameters	[SWS_Nm_00999]
[SRS_BSW_00395]	The Basic Software Module specifications shall list all configuration parameter dependencies	[SWS_Nm_00999]
[SRS_BSW_00396]	The Basic Software Module specifications shall specify the supported configuration classes for changing values and multiplicities for each parameter/container	[SWS_Nm_00120] [SWS_Nm_00195]
[SRS_BSW_00397]	The configuration parameters in pre-compile time are fixed before compilation starts	[SWS_Nm_00999]

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

Requirement	Description	Satisfied by
[SRS_BSW_00408]	All AUTOSAR Basic Software Modules configuration parameters shall be named according to a specific naming rule	[SWS_Nm_00999]
[SRS_BSW_00409]	All production code error ID symbols are defined by the Dem module and shall be retrieved by the other BSW modules from Dem configuration	[SWS_Nm_00999]
[SRS_BSW_00410]	Compiler switches shall have defined values	[SWS_Nm_00999]
[SRS_BSW_00411]	All AUTOSAR Basic Software Modules shall apply a naming rule for enabling/disabling the existence of the API	[SWS_Nm_00999]
[SRS_BSW_00412]	References to c-configuration parameters shall be placed into a separate h-file	[SWS_Nm_00124]
[SRS_BSW_00413]	An index-based accessing of the instances of BSW modules shall be done	[SWS_Nm_00999]
[SRS_BSW_00414]	Init functions shall have a pointer to a configuration structure as single parameter	[SWS_Nm_00030] [SWS_Nm_00282] [SWS_Nm_00283]
[SRS_BSW_00415]	Interfaces which are provided exclusively for one module shall be separated into a dedicated header file	[SWS_Nm_00999]
[SRS_BSW_00416]	The sequence of modules to be initialized shall be configurable	[SWS_Nm_00127] [SWS_Nm_00128] [SWS_Nm_00129] [SWS_Nm_00131] [SWS_Nm_00133] [SWS_Nm_00135] [SWS_Nm_00137] [SWS_Nm_00139] [SWS_Nm_00141] [SWS_Nm_00143] [SWS_Nm_00145] [SWS_Nm_00147] [SWS_Nm_00149] [SWS_Nm_00151] [SWS_Nm_00999]
[SRS_BSW_00417]	Software which is not part of the SW-C shall report error events only after the DEM is fully operational.	[SWS_Nm_00999]

Requirement	Description	Satisfied by
[SRS_BSW_00419]	If a pre-compile time configuration parameter is implemented as "const" it should be placed into a separate c-file	[SWS_Nm_00123] [SWS_Nm_00247]
[SRS_BSW_00422]	Pre-de-bouncing of error status information is done within the DEM	[SWS_Nm_00999]
[SRS_BSW_00423]	BSW modules with AUTOSAR interfaces shall be describable with the means of the SW-C Template	[SWS_Nm_00999]
[SRS_BSW_00424]	BSW module main processing functions shall not be allowed to enter a wait state	[SWS_Nm_00118] [SWS_Nm_00999]
[SRS_BSW_00425]	The BSW module description template shall provide means to model the defined trigger conditions of schedulable objects	[SWS_Nm_00118]
[SRS_BSW_00426]	BSW Modules shall ensure data consistency of data which is shared between BSW modules	[SWS_Nm_00999]
[SRS_BSW_00427]	ISR functions shall be defined and documented in the BSW module description template	[SWS_Nm_00999]
[SRS_BSW_00428]	A BSW module shall state if its main processing function(s) has to be executed in a specific order or sequence	[SWS_Nm_00999]
[SRS_BSW_00429]	BSW modules shall be only allowed to use OS objects and/or related OS services	[SWS_Nm_00999]
[SRS_BSW_00432]	Modules should have separate main processing functions for read/receive and write/transmit data path	[SWS_Nm_00999]

Requirement	Description	Satisfied by
[SRS_BSW_00433]	Main processing functions are only allowed to be called from task bodies provided by the BSW Scheduler	[SWS_Nm_00999]
[SRS_BSW_00437]	Memory mapping shall provide the possibility to define RAM segments which are not to be initialized during startup	[SWS_Nm_00999]
[SRS_BSW_00438]	Configuration data shall be defined in a structure	[SWS_Nm_00999]
[SRS_BSW_00439]	Enable BSW modules to handle interrupts	[SWS_Nm_00999]
[SRS_BSW_00440]	The callback function invocation by the BSW module shall follow the signature provided by RTE to invoke servers via Rte_Call API	[SWS_Nm_00999]
[SRS_BSW_00441]	Naming convention for type, macro and function	[SWS_Nm_00999]
[SRS_BSW_00442]	{OBSOLETE} The AUTOSAR architecture shall support standardized debugging and tracing features	[SWS_Nm_00240]
[SRS_BSW_00447]	Standardizing Include file structure of BSW Modules Implementing Autosar Service	[SWS_Nm_00999]
[SRS_BSW_00448]	Module SWS shall not contain requirements from Other Modules	[SWS_Nm_00999]
[SRS_BSW_00449]	BSW Service APIs used by Autosar Application Software shall return a Std_ReturnType	[SWS_Nm_00999]
[SRS_BSW_00450]	A Main function of a un-initialized module shall return immediately	[SWS_Nm_00121]
[SRS_BSW_00451]	Hardware registers shall be protected if concurrent access to these registers occur	[SWS_Nm_00999]

Requirement	Description	Satisfied by
[SRS_BSW_00452]	Classification of runtime errors	[SWS_Nm_00999]
[SRS_BSW_00453]	BSW Modules shall be harmonized	[SWS_Nm_00999]
[SRS_BSW_00454]	An alternative interface without a parameter of category DATA_REFERENCE shall be available.	[SWS_Nm_00999]
[SRS_BSW_00456]	- A Header file shall be defined in order to harmonize BSW Modules	[SWS_Nm_00999]
[SRS_BSW_00457]	- Callback functions of Application software components shall be invoked by the Basis SW	[SWS_Nm_00999]
[SRS_BSW_00458]	Classification of production errors	[SWS_Nm_00999]
[SRS_BSW_00459]	It shall be possible to concurrently execute a service offered by a BSW module in different partitions	[SWS_Nm_00999]
[SRS_BSW_00460]	Reentrancy Levels	[SWS_Nm_00999]
[SRS_BSW_00461]	Modules called by generic modules shall satisfy all interfaces requested by the generic module	[SWS_Nm_00999]
[SRS_BSW_00462]	All Standardized Autosar Interfaces shall have unique requirement Id / number	[SWS_Nm_00999]
[SRS_BSW_00463]	Naming convention of callout prototypes	[SWS_Nm_00999]
[SRS_BSW_00464]	File names shall be considered case sensitive regardless of the filesystem in which they are used	[SWS_Nm_00999]
[SRS_BSW_00465]	It shall not be allowed to name any two files so that they only differ by the cases of their letters	[SWS_Nm_00999]
[SRS_BSW_00466]	Classification of extended production errors	[SWS_Nm_00999]

Requirement	Description	Satisfied by
[SRS_BSW_00467]	The init / deinit services shall only be called by BswM or EcuM	[SWS_Nm_00999]
[SRS_BSW_00469]	Fault detection and healing of production errors and extended production errors	[SWS_Nm_00999]
[SRS_BSW_00470]	Execution frequency of production error detection	[SWS_Nm_00999]
[SRS_BSW_00471]	Do not cause dead-locks on detection of production errors - the ability to heal from previously detected production errors	[SWS_Nm_00999]
[SRS_BSW_00472]	Avoid detection of two production errors with the same root cause.	[SWS_Nm_00999]
[SRS_BSW_00473]	Classification of transient faults	[SWS_Nm_00999]
[SRS_Nm_00043]	NM shall not prohibit bus traffic with NM not being initialized	[SWS_Nm_00999]
[SRS_Nm_00044]	The NM shall be applicable to different types of communication systems which are in the scope of Autosar and support a bus sleep mode.	[SWS_Nm_00051] [SWS_Nm_00172] [SWS_Nm_00274] [SWS_Nm_00276]
[SRS_Nm_00045]	NM has to provide services to coordinate shutdown of NM-clusters independently of each other	[SWS_Nm_00167] [SWS_Nm_00168]
[SRS_Nm_00046]	It shall be possible to trigger the startup of all Nodes at any Point in Time.	[SWS_Nm_00031] [SWS_Nm_00032]
[SRS_Nm_00047]	NM shall provide a service to request to keep the bus awake and a service to cancel this request.	[SWS_Nm_00032] [SWS_Nm_00034] [SWS_Nm_00171]

Requirement	Description	Satisfied by
[SRS_Nm_00048]	NM shall put the communication controller into sleep mode if there is no bus communication	[SWS_Nm_00046]
[SRS_Nm_00050]	The NM shall provide the current state of NM	[SWS_Nm_00043] [SWS_Nm_00114] [SWS_Nm_00275]
[SRS_Nm_00051]	NM shall inform application when NM state changes occur.	[SWS_Nm_00031] [SWS_Nm_00032] [SWS_Nm_00046] [SWS_Nm_00156] [SWS_Nm_00158] [SWS_Nm_00159] [SWS_Nm_00161] [SWS_Nm_00162] [SWS_Nm_00163] [SWS_Nm_00249]
[SRS_Nm_00052]	The NM interface shall signal to the application that all other ECUs are ready to sleep.	[SWS_Nm_00192] [SWS_Nm_00999]
[SRS_Nm_00053]	NM on a node which is or become bus unavailable shall have a deterministic Behavior	[SWS_Nm_00999]
[SRS_Nm_00054]	There shall be a deterministic time from the point where all nodes agree to go to bus sleep to the point where bus is switched off.	[SWS_Nm_00999]
[SRS_Nm_00137]	NM shall perform communication system error handling for errors that have impact on the NM behavior.	[SWS_Nm_00999]
[SRS_Nm_00142]	NM shall guarantee an upper limit for the bus load generated by NM itself.	[SWS_Nm_00999]
[SRS_Nm_00143]	The bus load caused by NM shall be predictable.	[SWS_Nm_00999]
[SRS_Nm_00144]	NM shall support communication clusters of up to 64 ECUs	[SWS_Nm_00999]
[SRS_Nm_00145]	On a properly configured node, NM shall tolerate a loss of a predefined number of NM messages	[SWS_Nm_00999]

Requirement	Description	Satisfied by
[SRS_Nm_00146]	The NM shall tolerate a time jitter of NM messages in one or more ECUs	[SWS_Nm_00999]
[SRS_Nm_00147]	The NM algorithm shall be processor independent.	[SWS_Nm_00999]
[SRS_Nm_00148]	The specification and implementation shall be split-up into a communication system independent and communication system dependent parts	[SWS_Nm_00999]
[SRS_Nm_00149]	The timing of NM shall be configurable.	[SWS_Nm_00175] [SWS_Nm_00281] [SWS_Nm_00284]
[SRS_Nm_00150]	Specific functions of the Network Management shall be statically configurable at pre-compile time	[SWS_Nm_00055] [SWS_Nm_00130] [SWS_Nm_00132] [SWS_Nm_00134] [SWS_Nm_00136] [SWS_Nm_00138] [SWS_Nm_00140] [SWS_Nm_00142] [SWS_Nm_00144] [SWS_Nm_00146] [SWS_Nm_00148] [SWS_Nm_00150] [SWS_Nm_00164] [SWS_Nm_00165] [SWS_Nm_00166] [SWS_Nm_00231] [SWS_Nm_00241] [SWS_Nm_00251] [SWS_Nm_00255] [SWS_Nm_00273] [SWS_Nm_00277] [SWS_Nm_00278] [SWS_Nm_00279]
[SRS_Nm_00151]	The Network Management algorithm shall allow any node to integrate into an already running NM cluster	[SWS_Nm_00031]
[SRS_Nm_00153]	The Network Management shall optionally provide a possibility to detect present nodes	[SWS_Nm_00038] [SWS_Nm_00230]
[SRS_Nm_00154]	The Network Management API shall be independent from the communication bus	[SWS_Nm_00006] [SWS_Nm_00012] [SWS_Nm_00276]
[SRS_Nm_02503]	The NM API shall optionally give the possibility to send user data	[SWS_Nm_00035] [SWS_Nm_00250] [SWS_Nm_00252] [SWS_Nm_00285]
[SRS_Nm_02504]	The NM API shall optionally give the possibility to get user data	[SWS_Nm_00036]

Requirement	Description	Satisfied by
[SRS_Nm_02505]	The NM shall optionally set the local node identifier to the NM-message	[SWS_Nm_00039]
[SRS_Nm_02506]	The NM API shall give the possibility to read the source node identifier of the sender	[SWS_Nm_00037]
[SRS_Nm_02508]	Every node shall have associated with it a node identifier that is unique in the NM-cluster	[SWS_Nm_00040]
[SRS_Nm_02509]	The NM interface shall signal to the application that at least one other ECUs is not ready to sleep anymore.	[SWS_Nm_00193] [SWS_Nm_00999]
[SRS_Nm_02510]	For CAN NM it shall be optionally possible to immediately transmit the confirmation	[SWS_Nm_00999]
[SRS_Nm_02511]	It shall be possible to configure the Network Management of a node in Cluster Shutdown	[SWS_Nm_00168] [SWS_Nm_00228]
[SRS_Nm_02512]	The NM shall give the possibility to enable or disable the network management related communication configured for an active NM node	[SWS_Nm_00033] [SWS_Nm_00034]
[SRS_Nm_02513]	NM shall provide functionality which enables upper layers to control the sleep mode.	[SWS_Nm_00006] [SWS_Nm_00012] [SWS_Nm_00031] [SWS_Nm_00032] [SWS_Nm_00033] [SWS_Nm_00042] [SWS_Nm_00154] [SWS_Nm_00155]
[SRS_Nm_02514]	It shall be possible to group networks into NM Coordination Clusters	[SWS_Nm_00001] [SWS_Nm_00002] [SWS_Nm_00003] [SWS_Nm_00168] [SWS_Nm_00173]
[SRS_Nm_02515]	NM shall offer a generic possibility to run other NMs than the AUTOSAR-NMs	[SWS_Nm_00051] [SWS_Nm_00119] [SWS_Nm_00166] [SWS_Nm_00276]

Requirement	Description	Satisfied by
[SRS_Nm_02516]	All AUTOSAR NM instances shall support the NM Coordinator functionality including Bus synchronization on demand	[SWS_Nm_00169] [SWS_Nm_00171] [SWS_Nm_00174] [SWS_Nm_00175] [SWS_Nm_00176] [SWS_Nm_00177] [SWS_Nm_00194] [SWS_Nm_00284]
[SRS_Nm_02517]	<Bus>Nm shall support Partial Networking on CAN, FlexRay and Ethernet	[SWS_Nm_00999]
[SRS_Nm_02518]	Bus>Nm shall be able to distinguish between between NM Message	[SWS_Nm_00999]
[SRS_Nm_02519]	The NM Control Bit Vector shall contain a PNI (Partial Network Information) bit.	[SWS_Nm_00999]
[SRS_Nm_02520]	<Bus>Nm shall evaluate the PNI bit in the NM message	[SWS_Nm_00999]
[SRS_Nm_02521]	<Bus>Nm shall set the PNI bit for requesting Partial Network functionality	[SWS_Nm_00999]
[SRS_Nm_02522]	<Bus>Nm shall calculate the combined partial network request status EIRA	[SWS_Nm_00999]
[SRS_Nm_02523]	<Bus>Nm shall calculate the status of the external partial network requests ERA	[SWS_Nm_00999]
[SRS_Nm_02524]	<Bus>Nm shall communicate EIRA and ERA requests to the upper layers using virtual PDUs	[SWS_Nm_00999]
[SRS_Nm_02525]	<Bus>Nm shall support channel-specific configuration for ERA	[SWS_Nm_00999]
[SRS_Nm_02526]	<Bus>Nm shall support a global configuration for EIRA over all channels	[SWS_Nm_00999]
[SRS_Nm_02527]	CanNm shall implement a filter algorithm dropping all NM messages that are not relevant for the ECU	[SWS_Nm_00999]

Requirement	Description	Satisfied by
[SRS_Nm_02528]	CanNm shall provide a service which allows for spontaneous sending of NM messages.	[SWS_Nm_00999]
[SRS_Nm_02529]	If partial networking is used, the ECU shall secure that the first message on the bus is the wakeup frame.	[SWS_Nm_00999]
[SRS_Nm_02530]	CanIf shall provide an optional channel-specific TX filter	[SWS_Nm_00999]
[SRS_Nm_02531]	CanIf shall provide the possibility to initiate clear and check wake-up flags in the transceiver	[SWS_Nm_00999]
[SRS_Nm_02532]	When full communication is requested, CanSm shall enable pass mode on the CanIf TX filter	[SWS_Nm_00999]
[SRS_Nm_02533]	CanSm shall provide the possibility to initiate clear and check wake-up flags in the transceiver	[SWS_Nm_00999]
[SRS_Nm_02534]	CanSm shall support a validPN shutdown sequence	[SWS_Nm_00999]
[SRS_Nm_02535]	NM coordination on Nested Sub-Buses	[SWS_Nm_00254] [SWS_Nm_00256] [SWS_Nm_00257] [SWS_Nm_00259] [SWS_Nm_00261] [SWS_Nm_00262] [SWS_Nm_00267] [SWS_Nm_00271] [SWS_Nm_00272] [SWS_Nm_00280]
[SRS_Nm_02536]	NM shall provide an interface which triggers the transition to the Network Mode without keeping the network awake	[SWS_Nm_00031] [SWS_Nm_00119] [SWS_Nm_00245]
[SRS_Nm_02537]	No description	[SWS_Nm_00181] [SWS_Nm_00182] [SWS_Nm_00183] [SWS_Nm_00185] [SWS_Nm_00235] [SWS_Nm_00236] [SWS_Nm_00267]

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, J1939Nm, FrNm, LinNm and UdpNm) and the Communication Manager module (ComM).

Note: The Nm does not provide interface functions beyond those specified in this document. The Nm will provide an interface to the ComM, that does not contain specific knowledge about the type of the underlying busses, and that nevertheless is sufficient to accomplish the necessary network management functions. The algorithm handled by the Nm is bus independent.

Note: It is also required that other service layer modules access network management functions exclusively via Nm and that no bypasses to bus specific NM functions exist

[SWS_Nm_00006] ⌈ The Nm shall convert generic function calls from the ComM to bus specific functions of the bus specific NM layer. ⌋*(SRS_Nm_00154, SRS_Nm_02513)*

[SWS_Nm_00012] ⌈ The Nm shall convert callback functions called by the bus specific NM layers to generic callbacks to the ComM. ⌋*(SRS_Nm_00154, SRS_Nm_02513)*

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

7.2 NM Coordinator functionality

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

An ECU using an `NM` that actively performs the *NM Coordinator functionality* is commonly referred to as an `NM Coordinator`. However, in this specification this term is synonymous with the *NM Coordinator functionality* when used in requirements.

Note: Consider that certain bus types have different nomenclature on the terms Network, Channel, Cluster.

Note: 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 (see [SWS_Nm_00118]) are made. The NM Coordinator may use this to calculate the timeout status of internal timers.

7.2.1 Applicability of the NM Coordinator functionality

[SWS_Nm_00001] ⌈ The coordination algorithm shall be able to handle a topology where several coordinated busses are connected to one NM Coordinator. ⌋*(SRS_Nm_02514)*

[SWS_Nm_00256] ⌈ The NM-Coordinator shall support two or more NM-Coordinators connected to the same NM Cluster. ⌋*(SRS_Nm_02535)*

[SWS_Nm_00051] ⌈ The NM Coordinator shall be able to coordinate busses running the official AUTOSAR bus specific NMs as well as all other generic bus NMs implementing the required functionality, callbacks and interfaces as specified in subsection 7.4.2. ⌋*(SRS_Nm_00044, SRS_Nm_02515)*

Note: Coordinator Support for **J1939Nm** is not needed as the **J1939Nm** does not support shutdown handling.

[SWS_Nm_00055] ⌈ The NM Interface configuration shall provide the parameter NmCoordinatorSupportEnabled to define if the support of the *NM Coordinator functionality* is present or not. ⌋*(SRS_Nm_00150)*

[SWS_Nm_00167] ⌈ It shall be possible to configure multiple NM coordination clusters that shall be coordinated independently. ⌋*(SRS_Nm_00045)*

[SWS_Nm_00168] ⌈ Each bus shall belong to zero or one NM coordination cluster. ⌋*(SRS_Nm_00045, SRS_Nm_02511, SRS_Nm_02514)*

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

[SWS_Nm_00169] ⌈ Shutdown shall only be coordinated on the presently awake networks of a coordination cluster. Networks that are already in "bus-sleep mode" shall still be monitored but not coordinated. ⌋*(SRS_Nm_02516)*

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

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

7.2.2 Keeping coordinated busses alive

[SWS_Nm_00002] ⌈ As long as the node implementing the NM Coordinator is not ready to go to sleep on at least one of the busses in a coordination cluster (i.e. that it has actively requested the network), the NM Coordinator shall ensure that the network is requested on all currently active busses in that coordination cluster. ⌋*(SRS_Nm_02514)*

[SWS_Nm_00003] ⌈ As long as at least one bus in the coordination cluster is not ready to sleep (i.e. because another node than the NM Coordinator is requesting that bus), the NM Coordinator shall still ensure that the network is requested on all currently active busses in that coordination cluster even if the local ECU itself is ready to go to sleep on all busses of that coordination cluster. ⌋*(SRS_Nm_02514)*

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

Rationale: The **Nm** requests the network on a bus by calling the bus specific NM function BusNm_NetworkRequest().

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

The two requirements [SWS_Nm_00002] and [SWS_Nm_00003] above can be summarized as follows: as long as at least one node (including the node implementing the NM Coordinator) keeps any of the busses in the coordination cluster awake, the NM Coordinator shall keep all busses of that coordination cluster awake.

[SWS_Nm_00228] ⌈ If a bus of a coordination cluster has the parameter NmChannelSleepMaster set to TRUE, the NM Coordinator shall consider that bus ready to sleep at all times and shall not await an invocation of Nm_RemoteSleepIndication() from that bus before starting shutdown of that network. ⌋*(SRS_Nm_02511)*

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

7.2.3 Shutdown of coordinated busses

The level of synchronization achievable is dependent on the configuration. See subsection 7.2.5, Figure 7.1 shows an overview of the coordination algorithm. As described in Section 7.2.1, the coordination algorithm and coordinated shutdown shall be applied independently per NM coordination cluster.

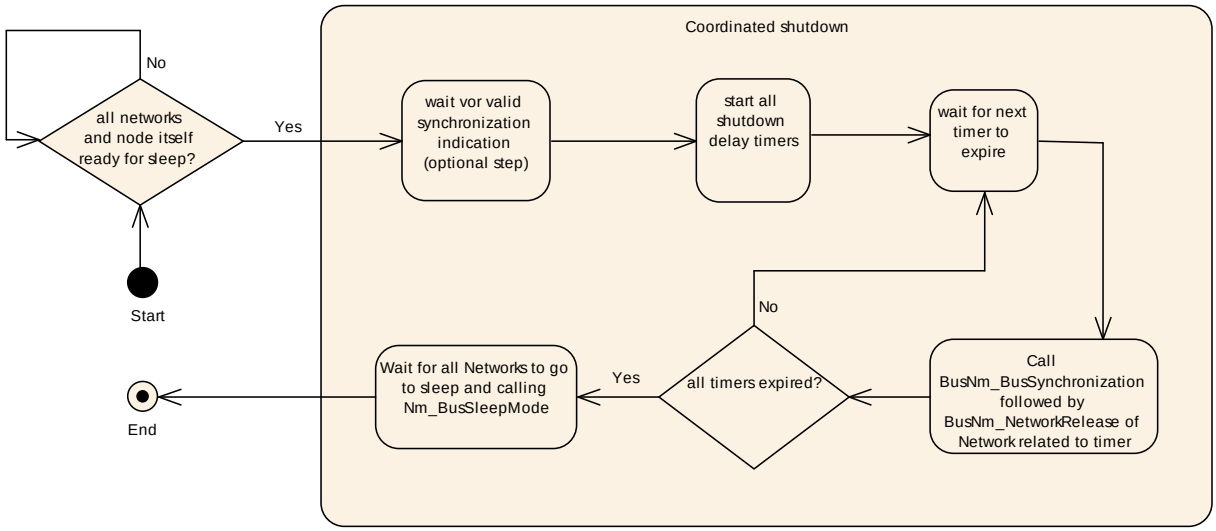

Figure 7.1: Overview of the coordination algorithm with the coordinated shutdown as part of it

Note: There is no limitation where the actions performed by the coordination algorithm shall take place.
This can be done either by the Nm main function (Nm_MainFunction()) or module indication / callbacks.

[SWS_Nm_00171] ⌈ When all networks of a coordination cluster are either ready to go to sleep or already in ”bus-sleep mode“ the NM Coordinator shall start the coordinated shutdown on all awake networks. The NM Coordinator shall evaluate continuously if the coordinated shutdown can be started. ⌋*(SRS_Nm_00047, SRS_Nm_02516)*

Rationale: Evaluation of shutdown conditions can be also done in other API calls then the main function. The evaluation can be segmented then to check only the specific conditions affected by the API calls there, hence it is not necessary to re-evaluate all conditions in every main processing period and every API call.

[SWS_Nm_00172] ⌈ If the configuration parameter NmSynchronizingNetwork is TRUE for any of the busses in a coordination cluster, the coordination shutdown shall be delayed until a network that is configured as synchronizing network for this coordination cluster invoked Nm_SynchronizationPoint(). ⌋*(SRS_Nm_00044)*

Rationale: If one or more of the networks in the NM coordination clusters is cyclic (such as FlexRay), a higher level of synchronized shutdown will be achieved if the algorithm is synchronized with one of the included cyclic networks. If configured so, the

shutdown timers for all coordinated networks will not be started until the synchronizing network has called the Nm_SynchronizationPoint().

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

[SWS_Nm_00173] ⌈ If not all conditions to start the `coordinated shutdown` have been met, or if the `coordinated shutdown` has already been started (but not aborted), calls to Nm_SynchronizationPoint() shall be ignored. ⌋*(SRS_Nm_02514)*

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

[SWS_Nm_00174] ⌈ If the configuration parameter NmSynchronizingNetwork is `FALSE` for all of the presently awake busses in a coordination cluster, the timers shall be started after all shutdown conditions have been met, without waiting for a call to `Nm_SynchronizationPoint`(). (see also [SWS_Nm_00172]). ⌋*(SRS_Nm_02516)*

[SWS_Nm_00175] ⌈ When the `coordinated shutdown` is started, a shutdown delay timer shall be activated for each currently awake channel in the coordination cluster. Each timer shall be configured with the shutdown delay timer calculated for that channel using the NmGlobalCoordinatorTime and subtracting the shutdown time of the specific channel **TSHUTDOWN_CHANNEL**. ⌋*(SRS_Nm_00149, SRS_Nm_02516)*

[SWS_Nm_00284] ⌈ If the NmGlobalCoordinatorTime is zero the shutdown delay timer of all channels shall also be zero. ⌋*(SRS_Nm_00149, SRS_Nm_02516)*

Note: The **TSHUTDOWN_CHANNEL** can be calculated as described in subsection 7.2.5 or with following formulas:
CanNm: Ready Sleep Time + Prepare BusSleep Time
FrNm: Ready Sleep Time, e.g.: (FrNmReadySleepCnt+1) * FrNmRepetitionCycle * ”Duration of one Flexray Cycle“
GenericNm: NmGenericBusNmShutdownTime

[SWS_Nm_00176] ⌈ When a shutdown timer expires for a network, **Nm** shall release the network by calling the BusNm_RequestBusSynchronization() followed by BusNm_NetworkRelease(). ⌋*(SRS_Nm_02516)*

[SWS_Nm_00177] ⌈ **Nm** shall keep track of all networks that have been released but have not yet reported ”bus-sleep mode“. If the shutdown is aborted, these networks shall still be considered active networks. (See Section subsection 7.3.3). ⌋*(SRS_Nm_02516)*

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

7.2.4 Coordination of nested sub-busses

To support the coordination of nested sub-busses the Nm-Coordinators need be configured to build up a coordination hierarchy. The top most `NM Coordinator` has only actively coordinated channels (NMActiveCoordinator == `TRUE`) per coordination cluster. This `NM Coordinator` has to initiate the `coordinated shutdown` for all other coordinators. An nested `NM Coordinator` receive his shutdown indication information from his passively configured channel (NmActiveCoordinator == `FALSE`) and provides this information to following NM Coordinators via his actively coordinated channels (NMActiveCoordinator == `TRUE`).

The Figure 7.2 will explain this as an example.

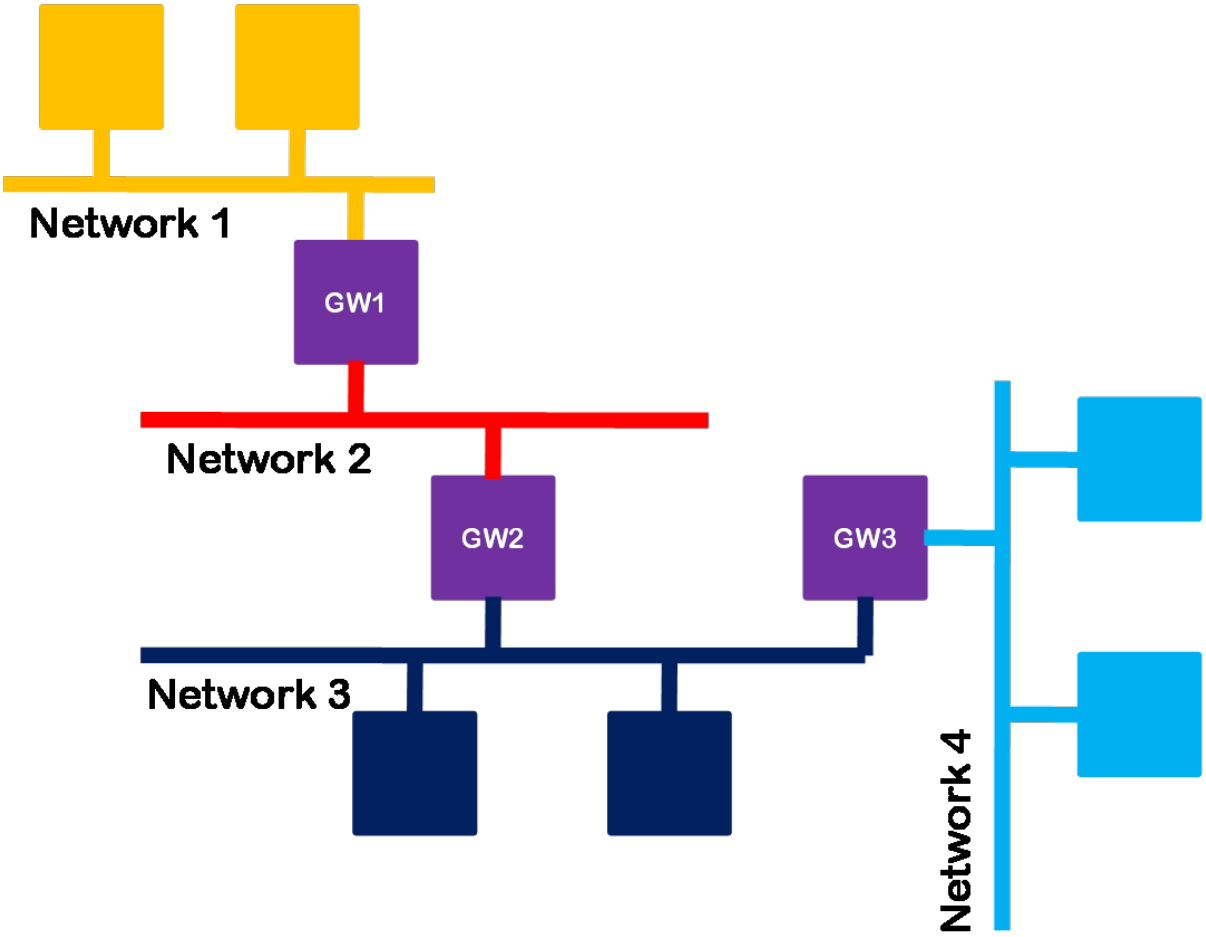

Figure 7.2: Use Case Nested Gateways

The exemplary topology shown in Figure 7.2 has the following coordination approach. GW 1 have configured the channel onto Network 1 and Network 2 as actively coordinating channels. Where GW 2 is configured with Network 2 connection as passively coordinated channel, but with actively coordinated channel on Network 3. GW 3 than needs to be configured on Network 3 as passively coordinated channel but as actively coordinated channel for his connection to the Network 4.

[SWS_Nm_00280] ⌈ The functionality of coordinating nested sub busses shall be available if the NmCoordinatorSyncSupport parameter is set to TRUE. ⌋*(SRS_Nm_02535)*

Note: All requirements within this chapter are valid "per Nm Coordination Cluster" (see [SWS_Nm_00167]).

The NMActiveCoordinator parameter indicates, if an NM Coordinator behaves on this channel in actively manner
(Actively coordinated channel) [NMActiveCoordinator = TRUE]
or behave in a passively manner
(Passively coordinated channel) [NMActiveCoordinator = FALSE].

[SWS_Nm_00257] ⌈ On its passively coordinated channels a NM-Coordinator shall send Nm messages only if the node has a network management request pending or a connected network which is coordinated actively by that NM Coordinator is not ready to sleep. ⌋*(SRS_Nm_02535)*

Rationale: This prevents that 2 NM Coordinators at the same channel, send NM messages when they are ready to sleep and therefore keep the bus awake. Without this mechanism it would not be possible to detect if there is at least one other node active.

[SWS_Nm_00259] ⌈ The NM Coordinator shall set the *NMcoordinatorSleepReady* bit in the NM message via <BusNm>_SetSleepReadyBit(Nm_Channel, value) to the value 1 at his actively coordinated channels,
IF
all nodes of the NM Coordination cluster are ready to sleep (RemoteSleepIndication)
AND
IF NmSynchronizingNetwork is enabled a Nm_SynchronizationPoint() call has been received on the corresponding channel
AND
all channels of this NM Coordination cluster are configured as NMActiveCoordinator == TRUE. ⌋*(SRS_Nm_02535)*

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

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

Rationale: Nodes which contain passively coordinated channels do not need a synchronization point as they are synchronized by the sleep ready bit of their active coordinator already.

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

[SWS_Nm_00261] ⌈ If Nm_CoordReadyToSleepIndication() is received on a passively coordinated channel the NmCoordinator shall set the *NMCoordinatorSleepReady* bit to SET (1) via API call to <BusNm>_SetSleepReadyBit(Nm_Channel,value) on all actively coordinated channels. ⌋*(SRS_Nm_02535)*

[SWS_Nm_00271] ⌈ If Nm_CoordReadyToSleepCancellation() is received on a passively coordinated channel the NmCoordinator shall set the *NMCoordinatorSleepReady* bit to `UNSET (0)` via API call to <BusNm>_SetSleepReadyBit(Nm_Channel,value) on all actively coordinated channels. ⌋*(SRS_Nm_02535)*

Note: On its passively coordinated channel a `NM Coordinator` would not set the *Sleep Ready* bit ever (via **<busNm>** function call) but forward a received status change of *Sleep ready* bit onto its actively coordinated channels.

Note: On its actively coordinated channel(s) a `NM Coordinator` a call of Nm_CoordReadyToSleepIndication() and Nm_CoordReadyToSleepCancellation()is not expected.

[SWS_Nm_00262] ⌈ The `NM Coordinator` shall start `coordinated shutdown` after the *Sleep Ready* Bit with `SET` status has been requested. ⌋*(SRS_Nm_02535)*

[SWS_Nm_00281] ⌈ NmGlobalCoordinatorTime shall be set at least to the maximum time needed to shut down all Networks coordinated. ⌋*(SRS_Nm_00149)*

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

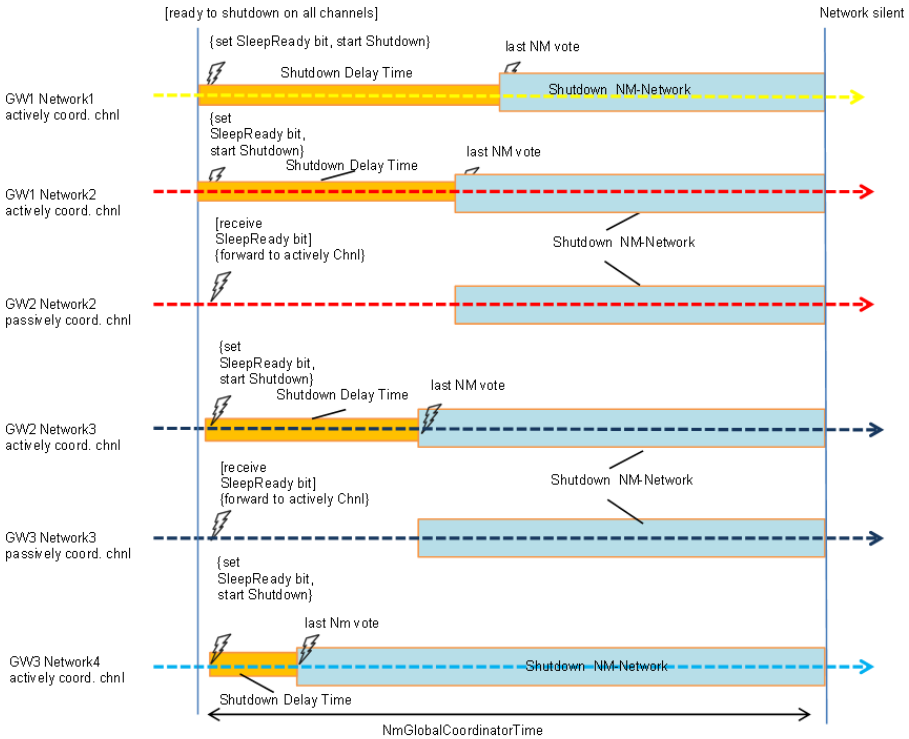

Figure 7.3: Shutdown with Nm_GlobalCoordinatorTime

[SWS_Nm_00267] ⌈ NM Coordinator shall set the *NMCoordinatorSleepReady* bit to UNSET (0) via API call to <BusNm>_SetSleepReadyBit(Nm_Channel,value) on all actively coordinated channels if the coordinated shutdown has been aborted for any reason. ⌋*(SRS_Nm_02535, SRS_Nm_02537)*

Note: Details about aborted shutdown can be found in subsection 7.3.3.

7.2.5 Calculation of shutdown timers

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

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

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

7.2.6 Synchronization Use Case 1 - Synchronous command

This Use Case focuses on how to synchronize the point in time where the different networks are released.
This results in the fastest possible total shutdown of all networks, but with the downside that the networks will not enter ”bus-sleep mode“ at the same time.

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

Since the Use Case does not consider any cyclic behavior of the networks, the synchronization parameter NmSynchronizingNetwork shall be set to FALSE for all networks and no **bus specific NM** shall be configured to invoke the Nm_SynchronizationPoint()

callback.
To achieve the fastest possible shutdown, the shutdown timer parameter NmGlobalCoordinatorTime needs to be set to `0.0`.

7.2.7 Synchronization Use Case 2 - Synchronous initiation

This Use Case is an extension of Use Case 1, but here consideration is taken to the fact that for some networks the request to release the network will only be acted upon at specific points in time. This Use Case will command a simultaneous shutdown like in Use Case 1, but will wait until a point in time suitable for the synchronizing network.

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

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

7.2.8 Synchronization Use Case 3 - Synchronous network sleep

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

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

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

To calculate the shutdown timer **TSHUTDOWN_CHANNEL** of each network, specific knowledge of each networks timing behavior must be obtained.

For all networks, **TSHUTDOWN_CHANNEL** must be calculated, this is the minimum time it will take the network to enter ”bus-sleep mode“. For non-cyclic networks (such as CAN), the time shall be measured from the point in time when the network is released until it enters ”bus-sleep mode“. For cyclic networks (such as FlexRay) the time shall also include the full range from the synchronization indication made just before the network is released. For Generic **BusNms** the time is given by the configuration parameter NmGenericBusNmShutdownTime.

For the synchronizing network, **TSYNCHRONIZATION_INDICATION** must be determined. This is the time between any two consecutive calls made by that **bus specific NM** to Nm_SynchronizationPoint().

The NmGlobalCoordinatorTime shall be the total time that is needed for the `coordination algorithm`. This includes the shutdown time of nested sub-busses. Start with setting NmGlobalCoordinatorTime to the same value as **TSHUTDOWN_CHANNEL** for the synchronizing network. If the **TSHUTDOWN_CHANNEL** for any other network is greater than NmGlobalCoordinatorTime, extend NmGlobalCoordinatorTime with **TSYNCHRONIZATION_INDICATION** repeatedly until NmGlobalCoordinatorTime is equal to, or larger than any **TSHUTDOWN_CHANNEL**.

The shutdown delay timer for each network shall be calculated as NmGlobalCoordinatorTime - **TSHUTDOWN_CHANNEL** for that network.

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

7.2.8.1 Examples

In the first case (Figure 7.4), the synchronizing network holds the largest **TSHUTDOWN_CHANNEL**, which will therefore equal the NmGlobalCoordinatorTime. For the synchronizing network, the shutdown delay timer will be NmGlobalCoordinatorTime - **TSHUTDOWN_CHANNEL**, which is zero, but then a small amount of time is added to make sure that the Nm will wait to release the network between the two synchronization points.
For the Non-cyclic network, the shutdown delay timer will simply be NmGlobalCoordinatorTime - **TSHUTDOWN_CHANNEL**.

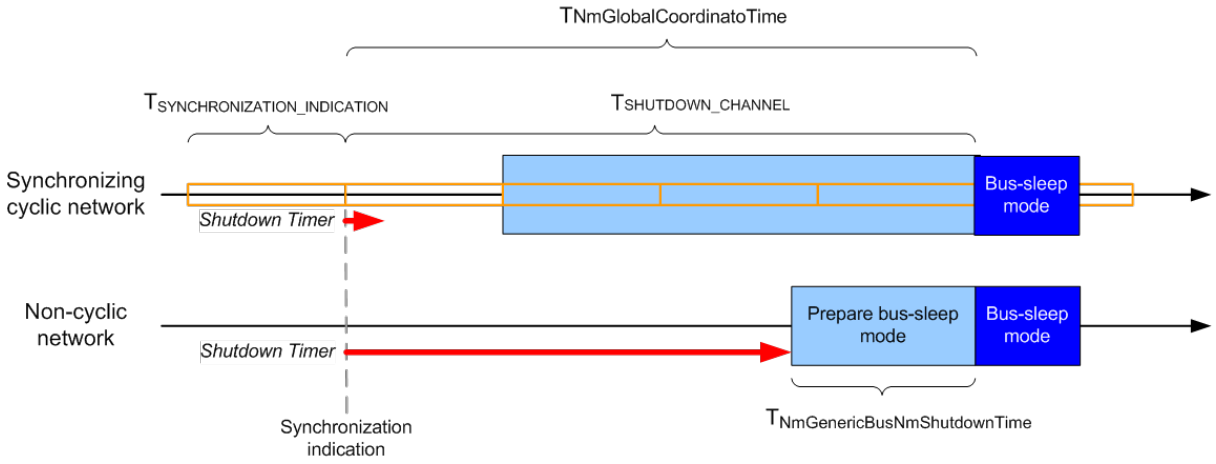

Figure 7.4: Timing example one

In the second case (Figure 7.5), the non-cyclic network takes very long time to shut down and therefore holds the largest **TSHUTDOWN_CHANNEL**. The NmGlobalCoordinatorTime has now been obtained by taking the synchronizing network's (slightly shorter) **TSHUTDOWN_CHANNEL** adding **TSYNCHRONIZATION_INDICATION** once to this value.

For the synchronizing network, the shutdown timer will be NmGlobalCoordinatorTime - **TSHUTDOWN_CHANNEL**, with a small amount of time added to make sure that the **Nm** will wait to release the network between the two synchronization points.
For the Non-cyclic network, the shutdown timer will simply be NmGlobalCoordinator-Time - **TSHUTDOWN_CHANNEL**.

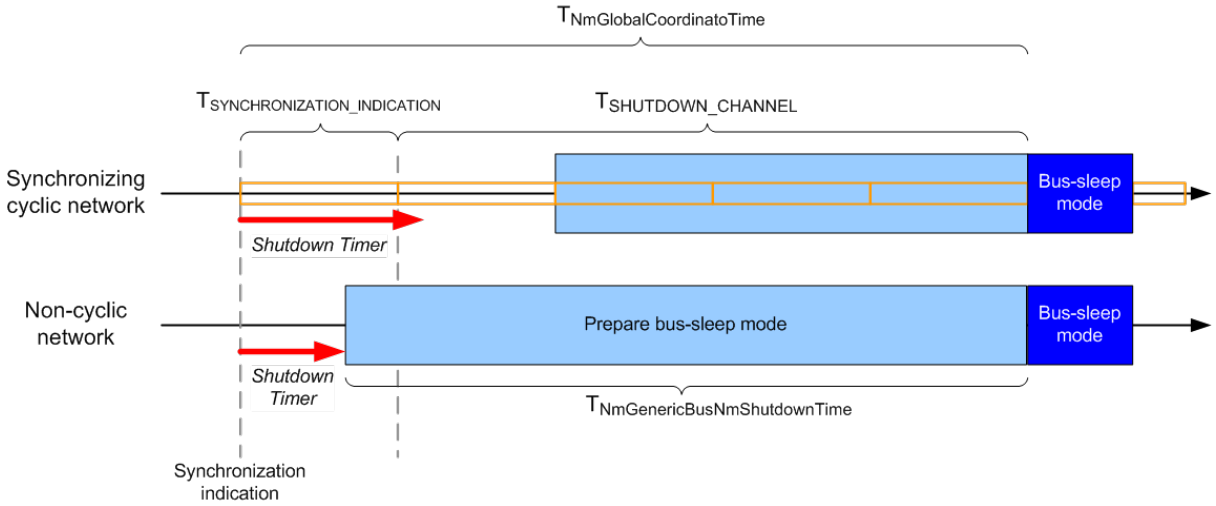

Figure 7.5: Timing example two

7.3 Wakeup and abortion of the coordinated shutdown

Nm is not responsible for normal wakeup of the node or the networks this will be done by the COM Manager (**ComM**).

7.3.1 External network wakeup

For both *Basic functionality* and *NM Coordination functionality*, **Nm** will forward wakeup indications from the networks (indicated by the bus specific NMs calling the callback Nm_NetworkStartIndication()) to the **ComM** by calling ComM_Nm_NetworkStartIndication(). **ComM** will then call Nm_PassiveStartUp, which will be forwarded by Nm to the corresponding interface of the bus specific NM.
Processing of wake-up events for channels in bus-sleep (related to transceiver and controller state) will be handled by **EcuM** and **ComM**. No interaction of the **Nm** apply here. **Nm** will get the network request from **ComM** as statet above, depending on the wake-up validation and the respective communication needs.

[SWS_Nm_00245] ⌈ If the **ComM** calls Nm_PassiveStartUp() for a network that is part of a coordinated cluster of networks, the **Nm** coordinator functionality shall treat this call as if the **ComM** had called Nm_NetworkRequest(). 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. ⌋*(SRS_Nm_02536)*

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

7.3.2 Coordinated wakeup

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

7.3.3 Abortion of the coordinated shutdown

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

1. A node on a network that has not yet been released and is still in 'network mode' starts requesting the network again. This will be detected by the **bus specific NM** which will inform **Nm** by calling Nm_RemoteSleepCancellation().
2. A node on a network that has already been released and has indicated "prepare bus-sleep mode" but not "bus-sleep mode" starts requesting the network again. This will be detected by the **bus specific NM** that will automatically change state to "network mode" and inform **Nm** by calling Nm_NetworkMode().
3. The **ComM** requests the network on any of the networks in the `NM Coordination Cluster`.
4. The coordinator which actively coordinates this network sends Nm message with cleared Ready-Sleep Bit. This will be detected by the Bus spec NM (only on passively coordinated channels) and forwarded to the NM by calling Nm_CoordReadyToSleepCancellation().

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

[SWS_Nm_00181] ⌈ The `coordinated shutdown` shall be aborted if any network in that `NM Coordination Cluster`,

- indicates Nm_RemoteSleepCancellation() or
- indicates Nm_NetworkMode() or
- indicates Nm_CoordReadyToSleepCancellation()
- or the **ComM** request one of the networks with Nm_NetworkRequest()or Nm_PassiveStartUp().

⌋*(SRS_Nm_02537)*

Note: Nm_NetworkStartIndication() is not a trigger to abort the `coordinated shutdown`, as this is handled by the upper layer.

[SWS_Nm_00182] ⌈ If the `coordinated shutdown` is aborted, `NM Coordinator` shall call ComM_Nm_RestartIndication() for all networks that already indicated "bus sleep". ⌋*(SRS_Nm_02537)*

Rationale: Since **Nm** cannot take decision to wake networks on its own, this must be decided by **ComM** just as in the (external) wakeup case.

[SWS_Nm_00183] ⌈ If the `coordinated shutdown` is aborted, `NM Coordinator` shall request the network from the **<busNm's>** for the networks that have not indicated "bus sleep". ⌋*(SRS_Nm_02537)*

[SWS_Nm_00185] ⌈ If the `coordination algorithm` has been aborted, all conditions that guard the initiation of the `coordinated shutdown` shall be evaluated again. ⌋*(SRS_Nm_02537)*

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

[SWS_Nm_00235] ⌈ If a coordinated shutdown has been aborted and **Nm** receives E_NOT_OK on a <BusNm>_NetworkRequest(), that network shall not be considered awake when the conditions for initiating a coordinated shutdown are evaluated again. ⌋*(SRS_Nm_02537)*

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

[SWS_Nm_00236] ⌈ If a coordinated shutdown has been initiated and **Nm** receives E_NOT_OK on a <BusNm>_NetworkRelease(), the shutdown shall be immediately aborted. For all networks that have not entered ”bus-sleep mode“, **Nm** shall request the networks. This includes the network that indicated an error for <BusNm>_NetworkRelease(). As soon as this has been done, the conditions for initiating coordinated shutdown can be evaluated again. This applies also to networks that were not actively participating in the current coordinated shutdown. ⌋*(SRS_Nm_02537)*

Rationale: If a network cannot be released, it shall immediately be requested again to synchronize the states between the NM Coordinator in the **Nm** and the **<BusNm>**. The coordinated shutdown will eventually be initiated again as long as the problem with the **<BusNm>** persists. It is up to the **<BusNm>** to report any problems directly to the **DEM** and/or **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 gives an overview of the API calls that are 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 requirements of the interfaces and the configuration parameters for enabling/disabling the API’s, refer to chapter 8.

7.4.1 Prerequisites for basic functionality

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

The *Basic functionality* provides the following API calls to the ComM:

- Nm_NetworkRequest() - [SWS_Nm_00032]
- Nm_NetworkRelease() - [SWS_Nm_00046]
- Nm_PassiveStartUp() - [SWS_Nm_00031]

Note: This implies that the **bus specific NM** provides the corresponding functions <BusNm>_NetworkRequest(), <BusNm>_NetworkRelease() and <BusNm>_PassiveStartUp().

The *Basic functionality* forwards the following API callbacks to the **ComM**:

- Nm_NetworkStartIndication() - [SWS_Nm_00154]
- Nm_NetworkMode() - [SWS_Nm_00156]
- Nm_BusSleepMode() - [SWS_Nm_00162]
- Nm_PrepareBusSleepMode() - [SWS_Nm_00159]

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

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

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

- Nm_DisableCommunication() - [SWS_Nm_00033]
- Nm_EnableCommunication() - [SWS_Nm_00034]
- Nm_SetUserData() - [SWS_Nm_00035]
- Nm_GetUserData() - [SWS_Nm_00036]
- Nm_GetPduData() - [SWS_Nm_00037]
- Nm_RepeatMessageRequest() - [SWS_Nm_00038]
- Nm_GetNodeIdentifier() - [SWS_Nm_00039]

- Nm_GetLocalNodeIdentifier() - [SWS_Nm_00040]
- Nm_CheckRemoteSleepIndication() - [SWS_Nm_00042]
- Nm_GetState() - [SWS_Nm_00043]

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

7.4.2 Prerequisites for NM Coordinator functionality

The `coordination algorithm` makes use of the following interfaces of the **bus specific NM**:

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

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

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

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

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

The `coordination algorithm` requires that the following callbacks of the **Nm** can be invoked by the **bus specific NM**:

- Nm_NetworkStartIndication() - [SWS_Nm_00154]
- Nm_NetworkMode() - [SWS_Nm_00156]
- Nm_BusSleepMode() - [SWS_Nm_00162]
- Nm_PrepareBusSleepMode() - [SWS_Nm_00159]
- Nm_RemoteSleepIndication() - [SWS_Nm_00192]

- Nm_RemoteSleepCancellation() - [SWS_Nm_00193]

- Nm_SynchronizationPoint() - [SWS_Nm_00194]

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

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

Note: A **bus specific NM** which is included in a coordination cluster must monitor its bus to identify when all other nodes on the network is ready to go to sleep. When this occurs, the **bus specific NM** shall call the callback Nm_RemoteSleepIndication() of **Nm**. (See [SWS_Nm_00192]).

Note: After a **bus specific NM** which is included in a coordination cluster has signaled to **Nm** that all other nodes on the network is ready to go to sleep (See [SWS_Nm_00192]), it must continue monitoring its bus to identify if any node starts requesting the network again, implying that the bus is no longer ready to go to sleep. When this occurs, the **bus specific NM** shall call the callback Nm_RemoteSleepCancellation() of **Nm**. (See [SWS_Nm_00193]).

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

Rationale: The Nm_SynchronizationPoint() shall be called by the **bus specific NM** in order to inform the coordination algorithm of a suitable point in time to initiate the coordinated shutdown. For cyclic networks this is typically at cycle boundaries. For non-cyclic networks this must be defined by other means. Each *NM Coordination Cluster* can be configured to make use of synchronization indications or not (See [SWS_Nm_00172]), and if they are used, the coordination algorithm filters indications and only acts on indications from networks that are configured as synchronizing networks.

Note: Please note for implementation of <bus>Nm: Cyclic networks invoke the Nm_SynchronizationPoint() repeatedly when no other nodes request the network. The invocation is typically made at boundaries in the **bus specific NM** protocol when changes in the NM voting will occur.

It is assumed that any call to BusNm_ReleaseNetwork() made between two of these Nm_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 NM**s. Since **Nm** is only a pass-through interface layer regarding features that are not used by the *NM Coordinator functionality*, enabling these in **Nm**'s configuration will in many cases only enable the pass-through of the controlling API functions and the callback indications from the bus specific layers.

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

7.5 Additional Functionality

7.5.1 Nm_CarWakeUpIndication

[SWS_Nm_00252] ⌈ If the <bus>Nm calls Nm_CarWakeUpIndication and NmCarWakeUpCallout is defined, the NM Interface shall call the callout function defined by NmCarWakeUpCallout with nmNetworkHandle as parameter. ⌋*(SRS_Nm_02503)*

[SWS_Nm_00285] ⌈ If the <bus>Nm calls Nm_CarWakeUpIndication and NmCarWakeUpCallout is not defined, the NM Interface shall call the function BswM_CarWakeUpIndication with nmNetworkHandle as parameter. ⌋*(SRS_Nm_02503)*

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

Note: The callout will be declared as specified within SWS_BSW_00039 and SWS_BSW_00135.

7.5.2 Nm_StateChangeNotification

[SWS_Nm_00249] ⌈ 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)

⌋(*SRS_Nm_00051*)

7.6 Error classification

[SWS_Nm_00232] ⌈ The **Nm** shall be able to detect the following errors and exceptions depending on its configuration:

Type of 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`

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

7.7 Error detection

For details refer to the chapter 7.3 “Error Detection” in [2, SWS_BSWGeneral].

7.8 Error notification

[SWS_Nm_00233] ⌈ 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. ⌋(*SRS_BSW_00323, SRS_BSW_00369, SRS_BSW_00386*)

Note: The handling of NULL-pointers is specified within [2, SWS_BSWGeneral], see SWS_BSW_00212.

7.9 Debugging

[SWS_Nm_00240] OBSOLETE ⌈ If the configuration switch NmCoordinatorSupportEnabled is set, the internal states of the NM Coordinator shall be available for debugging. ⌋(*SRS_BSW_00442*)

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.

[SWS_Nm_00117] ⌈

Module	*Imported Type*
Com	Com_SignalIdType
ComStack_Types	NetworkHandleType
Std_Types	Std_ReturnType Std_VersionInfoType

Table 8.1: Nm_ImportedTypes

⌋(*SRS_BSW_00301*)

8.2 Type definitions

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

8.2.1 Nm_ModeType

[SWS_Nm_00274] ⌈

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.	

Table 8.2: Nm_ModeType

⌋(*SRS_Nm_00044*)

8.2.2 Nm_StateType

[SWS_Nm_00275] ⌈

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)
	NM_STATE_OFFLINE	Offline State (7)
Description:	States of the network management state machine.	

Table 8.3: Nm_StateType

⌋(*SRS_Nm_00050*)

8.2.3 Nm_BusNmType

[SWS_Nm_00276] ⌈

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
	NM_BUSNM_J1939NM	SAE J1939 NM type (address claiming)
Description:	BusNm Type	

Table 8.4: Nm_BusNmType

⌋(*SRS_Nm_00044*, *SRS_Nm_00154*, *SRS_Nm_02515*)

8.2.4 Nm_ConfigType

[SWS_Nm_00282] ⌈

Name:	Nm_ConfigType		
Type:	Structure		
Range:	implementation specific		–
Description:	Configuration data structure of the Nm module.		

Table 8.5: Nm_ConfigType

⌋(*SRS_BSW_00414*)

8.3 Function definitions

8.3.1 Standard services provided by NM Interface

8.3.1.1 Nm_Init

[SWS_Nm_00030] ⌈

Service name:	Nm_Init	
Syntax:	`void Nm_Init( const Nm_ConfigType* ConfigPtr )`	
Service ID[hex]:	0x00	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	ConfigPtr	Pointer to the selected configuration set.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Initializes the NM Interface.	

Table 8.6: Nm_Init

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

[SWS_Nm_00127] ⌈ Caveats of `Nm_Init`: This service function has to be called after the initialization of the respective bus interface. ⌋(*SRS_BSW_00101*, *SRS_BSW_00416*)

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

Note: The Configuration pointer `ConfigPtr` is currently not used and shall therefore be set NULL_PTR value.

8.3.1.2 Nm_PassiveStartUp

[SWS_Nm_00031] ⌈

Service name:	Nm_PassiveStartUp

Syntax:	`Std_ReturnType Nm_PassiveStartUp( NetworkHandleType NetworkHandle )`	
Service ID[hex]:	0x01	
Sync/Async:	Asynchronous	
Reentrancy:	Non-reentrant for the same NetworkHandle, reentrant otherwise	
Parameters (in):	NetworkHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: No error 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).	

Table 8.7: Nm_PassiveStartUp

⌋(*SRS_Nm_00046*, *SRS_Nm_00051*, *SRS_Nm_00151*, *SRS_Nm_02513*, *SRS_Nm_02536*)

[SWS_Nm_00128] ⌈ Caveats of `Nm_PassiveStartUp`: The **<BusNm>** and the **Nm** itself are initialized correctly. ⌋(*SRS_BSW_00101*, *SRS_BSW_00416*)

8.3.1.3 Nm_NetworkRequest

[SWS_Nm_00032] ⌈

Service name:	Nm_NetworkRequest	
Syntax:	`Std_ReturnType Nm_NetworkRequest( NetworkHandleType NetworkHandle )`	
Service ID[hex]:	0x02	
Sync/Async:	Asynchronous	
Reentrancy:	Non-reentrant for the same NetworkHandle, reentrant otherwise	
Parameters (in):	NetworkHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: No error 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).

Table 8.8: Nm_NetworkRequest

⌋(*SRS_Nm_00046*, *SRS_Nm_00047*, *SRS_Nm_00051*, *SRS_Nm_02513*)

[SWS_Nm_00129] ⌈ Caveats of `Nm_NetworkRequest`: The **<BusNm>** and the **Nm** itself are initialized correctly. ⌋(*SRS_BSW_00101*, *SRS_BSW_00416*)

[SWS_Nm_00130] ⌈ Configuration of `Nm_NetworkRequest`: This function is only available if NmPassiveModeEnabled is set to `FALSE`. ⌋(*SRS_Nm_00150*)

8.3.1.4 Nm_NetworkRelease

[SWS_Nm_00046] ⌈

Service name:	Nm_NetworkRelease	
Syntax:	`Std_ReturnType Nm_NetworkRelease( NetworkHandleType NetworkHandle )`	
Service ID[hex]:	0x03	
Sync/Async:	Asynchronous	
Reentrancy:	Non-reentrant for the same NetworkHandle, reentrant otherwise	
Parameters (in):	NetworkHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: No error 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).	

Table 8.9: Nm_NetworkRelease

⌋(*SRS_Nm_00048*, *SRS_Nm_00051*)

[SWS_Nm_00131] ⌈ Caveats of `Nm_NetworkRelease`: The **<BusNm>** and the **Nm** itself are initialized correctly. ⌋(*SRS_BSW_00101*, *SRS_BSW_00416*)

[SWS_Nm_00132] ⌈ Configuration of `Nm_NetworkRelease`: This function is only available if NmPassiveModeEnabled is set to `FALSE`. ⌋(*SRS_Nm_00150*)

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

[SWS_Nm_00033] ⌈

Service name:	Nm_DisableCommunication	
Syntax:	`Std_ReturnType Nm_DisableCommunication( NetworkHandleType NetworkHandle )`	
Service ID[hex]:	0x04	
Sync/Async:	Asynchronous	
Reentrancy:	Non-reentrant for the same NetworkHandle, reentrant otherwise	
Parameters (in):	NetworkHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: No error 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).	

Table 8.10: Nm_DisableCommunication

⌋*(SRS_Nm_02513, SRS_Nm_02512)*

[SWS_Nm_00133] ⌈ Caveats of `Nm_DisableCommunication`: The **<BusNm>** and the **Nm** itself are initialized correctly. ⌋*(SRS_BSW_00101, SRS_BSW_00416)*

[SWS_Nm_00134] ⌈ Configuration of `Nm_DisableCommunication`: This function is only available if NmComControlEnabled is set to `TRUE`. ⌋*(SRS_Nm_00150)*

8.3.2.2 Nm_EnableCommunication

[SWS_Nm_00034] ⌈

Service name:	Nm_EnableCommunication
Syntax:	`Std_ReturnType Nm_EnableCommunication( NetworkHandleType NetworkHandle )`

Service ID[hex]:	0x05	
Sync/Async:	Asynchronous	
Reentrancy:	Non-reentrant for the same NetworkHandle, reentrant otherwise	
Parameters (in):	NetworkHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: No error 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).	

Table 8.11: Nm_EnableCommunication

⌋(*SRS_Nm_00047*, *SRS_Nm_02512*)

[SWS_Nm_00135] ⌈ Caveats of `Nm_EnableCommunication`: The **<BusNm>** and the **Nm** itself are initialized correctly. ⌋(*SRS_BSW_00101*, *SRS_BSW_00416*)

[SWS_Nm_00136] ⌈ Configuration of `Nm_EnableCommunication`: This function is only available if NmComControlEnabled is set to `TRUE`. ⌋(*SRS_Nm_00150*)

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

[SWS_Nm_00035] ⌈

Service name:	Nm_SetUserData	
Syntax:	`Std_ReturnType Nm_SetUserData( NetworkHandleType NetworkHandle, const uint8* nmUserDataPtr )`	
Service ID[hex]:	0x06	
Sync/Async:	Synchronous	
Reentrancy:	Non-reentrant for the same NetworkHandle, reentrant otherwise	
Parameters (in):	NetworkHandle nmUserDataPtr	Identification of the NM-channel 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).	

Table 8.12: Nm_SetUserData

⌋(*SRS_Nm_02503*)

[SWS_Nm_00137] ⌈ Caveats of `Nm_SetUserData`: The **<BusNm>** and the **Nm** itself are initialized correctly. ⌋(*SRS_BSW_00101, SRS_BSW_00416*)

[SWS_Nm_00138] ⌈ Configuration of `Nm_SetUserData`: This function is only available if NmUserDataEnabled is set to `TRUE` and NmPassiveModeEnabled is set to `FALSE`. ⌋(*SRS_Nm_00150*)

[SWS_Nm_00241] ⌈ Configuration of `Nm_SetUserData`: If NmComUserDataSupport is `TRUE` the API `Nm_SetUserData` shall not be available. ⌋(*SRS_Nm_00150*)

8.3.3.2 Nm_GetUserData

[SWS_Nm_00036] ⌈

Service name:	Nm_GetUserData	
Syntax:	`Std_ReturnType Nm_GetUserData( NetworkHandleType NetworkHandle, uint8* nmUserDataPtr )`	
Service ID[hex]:	0x07	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	NetworkHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	nmUserDataPtr	Pointer where user data out of the last successfully received NM message shall be copied to
Return value:	Std_ReturnType	E_OK: No error 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).	

Table 8.13: Nm_GetUserData

⌋(*SRS_Nm_02504*)

[SWS_Nm_00139] ⌈ Caveats of `Nm_GetUserData`: The **<BusNm>** and the **Nm** itself are initialized correctly. ⌋(*SRS_BSW_00101, SRS_BSW_00416*)

[SWS_Nm_00140] ⌈ Configuration of `Nm_GetUserData`: This function is only available if NmUserDataEnabled is set to `TRUE`. ⌋(*SRS_Nm_00150*)

8.3.3.3 Nm_GetPduData

[SWS_Nm_00037] ⌈

Service name:	Nm_GetPduData	
Syntax:	`Std_ReturnType Nm_GetPduData( NetworkHandleType NetworkHandle, uint8* nmPduData )`	
Service ID[hex]:	0x08	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	NetworkHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	nmPduData	Pointer where NM PDU shall be copied to.
Return value:	Std_ReturnType	E_OK: No error 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).	

Table 8.14: Nm_GetPduData

⌋(*SRS_Nm_02506*)

[SWS_Nm_00141] ⌈ Caveats of `Nm_GetPduData`: The **<BusNm>** and the **Nm** itself are initialized correctly. ⌋(*SRS_BSW_00101, SRS_BSW_00416*)

[SWS_Nm_00142] ⌈ Configuration of `Nm_GetPduData`: This function is only available if NmNodeIdEnabled or NmUserDataEnabled is set to `TRUE`. ⌋(*SRS_Nm_00150*)

8.3.3.4 Nm_RepeatMessageRequest

[SWS_Nm_00038] ⌈

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

Table 8.15: Nm_RepeatMessageRequest

⌋(*SRS_Nm_00153*)

[SWS_Nm_00143] ⌈ Caveats of `Nm_RepeatMessageRequest`: The **<BusNm>** and the **Nm** itself are initialized correctly. ⌋(*SRS_BSW_00101, SRS_BSW_00416*)

[SWS_Nm_00144] ⌈ Configuration of `Nm_RepeatMessageRequest`: This function is only available if NmNodeDetectionEnabled is set to `TRUE`. ⌋(*SRS_Nm_00150*)

8.3.3.5 Nm_GetNodeIdentifier

[SWS_Nm_00039] ⌈

Service name:	Nm_GetNodeIdentifier	
Syntax:	`Std_ReturnType Nm_GetNodeIdentifier( NetworkHandleType NetworkHandle, uint8* nmNodeIdPtr )`	
Service ID[hex]:	0x0a	
Sync/Async:	Synchronous	
Reentrancy:	Non-reentrant for the same NetworkHandle, reentrant otherwise	
Parameters (in):	NetworkHandle	Identification of the NM-channel

Parameters (inout):	None	
Parameters (out):	nmNodeIdPtr	Pointer where node identifier out of the last successfully received NM-message shall be copied to
Return value:	Std_ReturnType	E_OK: No error 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).	

Table 8.16: Nm_GetNodeIdentifier

⌋(*SRS_Nm_02505*)

[SWS_Nm_00145] ⌈ Caveats of `Nm_GetNodeIdentifier`: The **<BusNm>** and the **Nm** itself are initialized correctly. ⌋(*SRS_BSW_00101, SRS_BSW_00416*)

[SWS_Nm_00146] ⌈ Configuration of `Nm_GetNodeIdentifier`: This function is only available if NmNodeIdEnabled is set to `TRUE`. ⌋(*SRS_Nm_00150*)

8.3.3.6 Nm_GetLocalNodeIdentifier

[SWS_Nm_00040] ⌈

Service name:	Nm_GetLocalNodeIdentifier	
Syntax:	`Std_ReturnType Nm_GetLocalNodeIdentifier( NetworkHandleType NetworkHandle, uint8* nmNodeIdPtr )`	
Service ID[hex]:	0x0b	
Sync/Async:	Synchronous	
Reentrancy:	Non-reentrant for the same NetworkHandle, reentrant otherwise	
Parameters (in):	NetworkHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	nmNodeIdPtr	Pointer where node identifier of the local node shall be copied to
Return value:	Std_ReturnType	E_OK: No error 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).

Table 8.17: Nm_GetLocalNodeIdentifier

⌋(*SRS_Nm_02508*)

[SWS_Nm_00147] ⌈ Caveats of `Nm_GetLoclaNodeIdentifier`: The **<BusNm>** and the **Nm** itself are initialized correctly. ⌋(*SRS_BSW_00101*, *SRS_BSW_00416*)

[SWS_Nm_00148] ⌈ Configuration of `Nm_GetLocalNodeIdentifier`: This function is only available if NmNodeIdEnabled is set to `TRUE`. ⌋(*SRS_Nm_00150*)

8.3.3.7 Nm_CheckRemoteSleepIndication

[SWS_Nm_00042] ⌈

Service name:	Nm_CheckRemoteSleepIndication	
Syntax:	`Std_ReturnType Nm_CheckRemoteSleepIndication( NetworkHandleType nmNetworkHandle, boolean* nmRemoteSleepIndPtr )`	
Service ID[hex]:	0x0d	
Sync/Async:	Synchronous	
Reentrancy:	Non-reentrant for the same NetworkHandle, reentrant otherwise	
Parameters (in):	nmNetworkHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	nmRemoteSleepIndPtr	Pointer where check result of remote sleep indication shall be copied to
Return value:	Std_ReturnType	E_OK: No error 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).	

Table 8.18: Nm_CheckRemoteSleepIndication

⌋(*SRS_Nm_02513*)

[SWS_Nm_00149] ⌈ Caveats of `Nm_CheckRemoteSleepIndication`: The **<BusNm>** and the **Nm** itself are initialized correctly. ⌋(*SRS_BSW_00101*, *SRS_BSW_00416*)

[SWS_Nm_00150] ⌈ Configuration of `Nm_CheckRemoteSleepIndication`: This function is only available if NmRemoteSleepIndEnabled is set to `TRUE`. ⌋*(SRS_Nm_00150)*

8.3.3.8 Nm_GetState

[SWS_Nm_00043] ⌈

Service name:	Nm_GetState	
Syntax:	`Std_ReturnType Nm_GetState( NetworkHandleType nmNetworkHandle, Nm_StateType* nmStatePtr, Nm_ModeType* nmModePtr )`	
Service ID[hex]:	0x0e	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	nmNetworkHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	nmStatePtr nmModePtr	Pointer where state of the network management shall be copied to 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).	

Table 8.19: Nm_GetState

⌋*(SRS_Nm_00050)*

[SWS_Nm_00151] ⌈ Caveats of `Nm_GetState`: The **<BusNm>** and the **Nm** itself are initialized correctly. ⌋*(SRS_BSW_00101, SRS_BSW_00416)*

8.3.3.9 Nm_GetVersionInfo

[SWS_Nm_00044] ⌈

Service name:	Nm_GetVersionInfo
Syntax:	`void Nm_GetVersionInfo( Std_VersionInfoType* nmVerInfoPtr )`
Service ID[hex]:	0x0f

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

Table 8.20: Nm_GetVersionInfo

⌋(*SRS_BSW_00003*, *SRS_BSW_00407*)

8.4 Call-back notifications

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

[SWS_Nm_00028] ⌈ All callbacks of the Nm shall assume that they can run either in task or in interrupt context. ⌋(*SRS_BSW_00333*)

8.4.1 Standard Call-back notifications

8.4.1.1 Nm_NetworkStartIndication

[SWS_Nm_00154] ⌈

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

Table 8.21: Nm_NetworkStartIndication

⌋(*SRS_BSW_00359*, *SRS_Nm_02513*)

[SWS_Nm_00155] ⌈ The indication through callback function `Nm_NetworkStartIndication`: shall be forwarded to **ComM** by calling the ComM_Nm_NetworkStartIndication. ⌋(*SRS_Nm_02513*)

8.4.1.2 Nm_NetworkMode

[SWS_Nm_00156] ⌈

Service name:	Nm_NetworkMode	
Syntax:	`void Nm_NetworkMode( NetworkHandleType nmNetworkHandle )`	
Service ID[hex]:	0x12	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant	
Parameters (in):	nmNetworkHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Notification that the network management has entered Network Mode.	

Table 8.22: Nm_NetworkMode

⌋(*SRS_BSW_00359*, *SRS_Nm_00051*)

[SWS_Nm_00158] ⌈ The indication through callback function `Nm_NetworkMode`: shall be forwarded to **ComM** by calling the ComM_Nm_NetworkMode. ⌋(*SRS_Nm_00051*)

8.4.1.3 Nm_BusSleepMode

[SWS_Nm_00162] ⌈

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

Table 8.23: Nm_BusSleepMode

⌋(*SRS_BSW_00359*, *SRS_Nm_00051*)

[SWS_Nm_00163] ⌈ The indication through callback function `Nm_BusSleepMode`: shall be forwarded to **ComM** by calling the ComM_Nm_BusSleepMode. ⌋(*SRS_Nm_00051*)

8.4.1.4 Nm_PrepareBusSleepMode

[SWS_Nm_00159] ⌈

Service name:	Nm_PrepareBusSleepMode	
Syntax:	`void Nm_PrepareBusSleepMode( NetworkHandleType nmNetworkHandle )`	
Service ID[hex]:	0x13	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant	
Parameters (in):	nmNetworkHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Notification that the network management has entered Prepare Bus-Sleep Mode.	

Table 8.24: Nm_PrepareBusSleepMode

⌋(*SRS_BSW_00359*, *SRS_Nm_00051*)

[SWS_Nm_00161] ⌈ The indication through callback function `Nm_PrepareBusSleepMode`: shall be forwarded to **ComM** by calling ComM_Nm_PrepareBusSleepMode. ⌋(*SRS_Nm_00051*)

8.4.1.5 Nm_RemoteSleepIndication

[SWS_Nm_00192] ⌈

Service name:	Nm_RemoteSleepIndication	
Syntax:	`void Nm_RemoteSleepIndication( NetworkHandleType nmNetworkHandle )`	
Service ID[hex]:	0x17	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant	
Parameters (in):	nmNetworkHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	

Description:	Notification that the network management has detected that all other nodes on the network are ready to enter Bus-Sleep Mode.

Table 8.25: Nm_RemoteSleepIndication

⌋(*SRS_BSW_00359*, *SRS_Nm_00052*)

[SWS_Nm_00277] ⌈ Configuration of `Nm_RemoteSleepIndication`: This function is only available if NmRemoteSleepIndEnabled is set to `TRUE`. ⌋(*SRS_Nm_00150*)

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

Note: When *NM Coordinator functionality* is disabled `Nm_RemoteSleepIndication()` can be a empty function.

8.4.1.6 Nm_RemoteSleepCancellation

[SWS_Nm_00193] ⌈

Service name:	Nm_RemoteSleepCancellation	
Syntax:	`void Nm_RemoteSleepCancellation( NetworkHandleType nmNetworkHandle )`	
Service ID[hex]:	0x18	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant	
Parameters (in):	nmNetworkHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Notification that the network management has detected that not all other nodes on the network are longer ready to enter Bus-Sleep Mode.	

Table 8.26: Nm_RemoteSleepCancellation

⌋(*SRS_BSW_00359*, *SRS_Nm_02509*)

[SWS_Nm_00278] ⌈ Configuration of `Nm_RemoteSleepCancellation`: This function is only available if NmRemoteSleepIndEnabled is set to `TRUE`. ⌋(*SRS_Nm_00150*)

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

Note: When *NM Coordinator functionality* is disabled `Nm_RemoteSleepCancellation()` can be a empty function.

8.4.1.7 Nm_SynchronizationPoint

[SWS_Nm_00194] ⌈

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

Table 8.27: Nm_SynchronizationPoint

⌋(*SRS_BSW_00359*, *SRS_Nm_02516*)

The notification that this is a suitable point in time to initiate the `coordinated shutdown` is only needed for internal purposes of the NM Coordinator.

8.4.1.8 Nm_CoordReadyToSleepIndication

[SWS_Nm_00254] ⌈

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

Table 8.28: Nm_CoordReadyToSleepIndication

⌋(*SRS_BSW_00359*, *SRS_Nm_02535*)

[SWS_Nm_00255] ⌈ Configuration of `Nm_CoordReadyToSleepIndication`: Optional

If NmCoordinatorSyncSupport is set to TRUE , the Nm shall provide the API Nm_CoordReadyToSleepIndication. ⌋(*SRS_Nm_00150*)

8.4.1.9 Nm_CoordReadyToSleepCancellation

[SWS_Nm_00272] ⌈

Service name:	Nm_CoordReadyToSleepCancellation	
Syntax:	`void Nm_CoordReadyToSleepCancellation( NetworkHandleType nmChannelHandle )`	
Service ID[hex]:	0x1f	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	nmChannelHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Cancels an indication, when the NM Coordinator Sleep Ready bit in the Control Bit Vector is set back to 0.	

Table 8.29: Nm_CoordReadyToSleepCancellation

⌋(*SRS_BSW_00359, SRS_Nm_02535*)

[SWS_Nm_00273] ⌈ Configuration of Nm_CoordReadyToSleepCancellation: Optional
If NmCoordinatorSyncSupport is set to TRUE , the Nm shall provide the API Nm_CoordReadyToSleepCancellation. ⌋(*SRS_Nm_00150*)

8.4.2 Extra Call-back notifications

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

8.4.2.1 Nm_PduRxIndication

[SWS_Nm_00112] ⌈

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

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

Table 8.30: Nm_PduRxIndication

⌋(*SRS_BSW_00359*)

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

[SWS_Nm_00164] ⌈ Configuration of `Nm_PduRxIndication`: This function is only available if `NmPduRxIndicationEnabled` is set to `TRUE`. ⌋(*SRS_Nm_00150*)

8.4.2.2 Nm_StateChangeNotification

[SWS_Nm_00114] ⌈

Service name:	Nm_StateChangeNotification	
Syntax:	`void Nm_StateChangeNotification( NetworkHandleType nmNetworkHandle, Nm_StateType nmPreviousState, Nm_StateType nmCurrentState )`	
Service ID[hex]:	0x16	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant	
Parameters (in):	nmNetworkHandle nmPreviousState nmCurrentState	Identification of the NM-channel Previous state of the NM-channel 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.	

Table 8.31: Nm_StateChangeNotification

⌋(*SRS_BSW_00359*, *SRS_Nm_00050*)

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

[SWS_Nm_00165] ⌈ Configuration of `Nm_StateChangeNotification`: This function is only available if `NmStateChangeIndEnabled` is set to TRUE. ⌋*(SRS_Nm_00150)*

8.4.2.3 Nm_RepeatMessageIndication

[SWS_Nm_00230] ⌈

Service name:	Nm_RepeatMessageIndication	
Syntax:	`void Nm_RepeatMessageIndication( NetworkHandleType nmNetworkHandle )`	
Service ID[hex]:	0x1a	
Sync/Async:	Asynchronous	
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.	

Table 8.32: Nm_RepeatMessageIndication

⌋*(SRS_BSW_00359, SRS_Nm_00153)*

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.

[SWS_Nm_00231] ⌈ Configuration of `Nm_RepeatMessageIndication`: This function is only available if `NmRepeatMsgIndEnabled` is set to TRUE. ⌋*(SRS_Nm_00150)*

8.4.2.4 Nm_TxTimeoutException

[SWS_Nm_00234] ⌈

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

Table 8.33: Nm_TxTimeoutException

⌋(*SRS_BSW_00359*)

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

8.4.2.5 Nm_CarWakeUpIndication

[SWS_Nm_00250] ⌈

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

Table 8.34: Nm_CarWakeUpIndication

⌋(*SRS_BSW_00359, SRS_Nm_02503*)

[SWS_Nm_00251] ⌈ Configuration of `Nm_CarWakeUpIndication`: Optional
If NmCarWakeUpRxEnabled is `TRUE`, The Nm shall provide the API Nm_CarWakeUpIndication(). ⌋(*SRS_Nm_00150*)

8.5 Scheduled functions

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

[SWS_Nm_00020] ⌈ A scheduled main function shall only contain logic related to the *NM Coordinator functionality*. ⌋(*SRS_BSW_00373*)

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

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

8.5.1 Nm_MainFunction

[SWS_Nm_00118] ⌈

Service name:	Nm_MainFunction
Syntax:	`void Nm_MainFunction( void )`
Service ID[hex]:	0x10
Description:	This function implements the processes of the NM Interface, which need a fix cyclic scheduling.

Table 8.35: Nm_MainFunction

⌋*(SRS_BSW_00424, SRS_BSW_00425)*

[SWS_Nm_00279] ⌈ If NmCoordinatorSupportEnabled is set to TRUE, the Nm_MainFunction() API shall be available. ⌋*(SRS_Nm_00150)*

8.6 Expected interfaces

This chapter lists all interfaces required from other modules.

8.6.1 Mandatory Interfaces

This chapter lists all interfaces required from other modules.

[SWS_Nm_00119] ⌈

API function	***Description***
`<BusNm>_PassiveStartUp`	Passive startup of the NM. It triggers the transition from Bus-Sleep Mode to the Network Mode without requesting the network.
`ComM_Nm_BusSleepMode`	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_PrepareBusSleep Mode	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.

Table 8.36: Nm Mandatory Interfaces

⌋(*SRS_Nm_02515*, *SRS_Nm_02536*)

8.6.2 Optional Interfaces

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

[SWS_Nm_00166] ⌈

API function	*Description*
<BusNm>_CheckRemoteSleep Indication	Check if remote sleep indication takes place or not.
<BusNm>_DisableCommunica-tion	Disable the NM PDU transmission ability.
<BusNm>_EnableCommunica-tion	Enable the NM PDU transmission ability.
<BusNm>_GetLocalNodeIden-tifier	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>_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>_RepeatMessageRe-quest	Request a Repeat Message Request to be transmitted next on the bus.
<BusNm>_RequestBusSynchro-nization	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.
BswM_NmIf_CarWakeUpIndication	Function called by NmIf to indicate a CarWakeup.
Com_SendSignal	The service Com_SendSignal updates the signal object identified by SignalId with the signal referenced by the SignalDataPtr parameter.
Det_ReportError	Service to report development errors.

Table 8.37: Nm Optional Interfaces

⌋(*SRS_Nm_00150*, *SRS_Nm_02515*)

8.6.3 Configurable Interfaces

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

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

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

9 Sequence diagrams

9.1 Basic functionality

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

9.2 Seq of NM Coordinator functionality

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

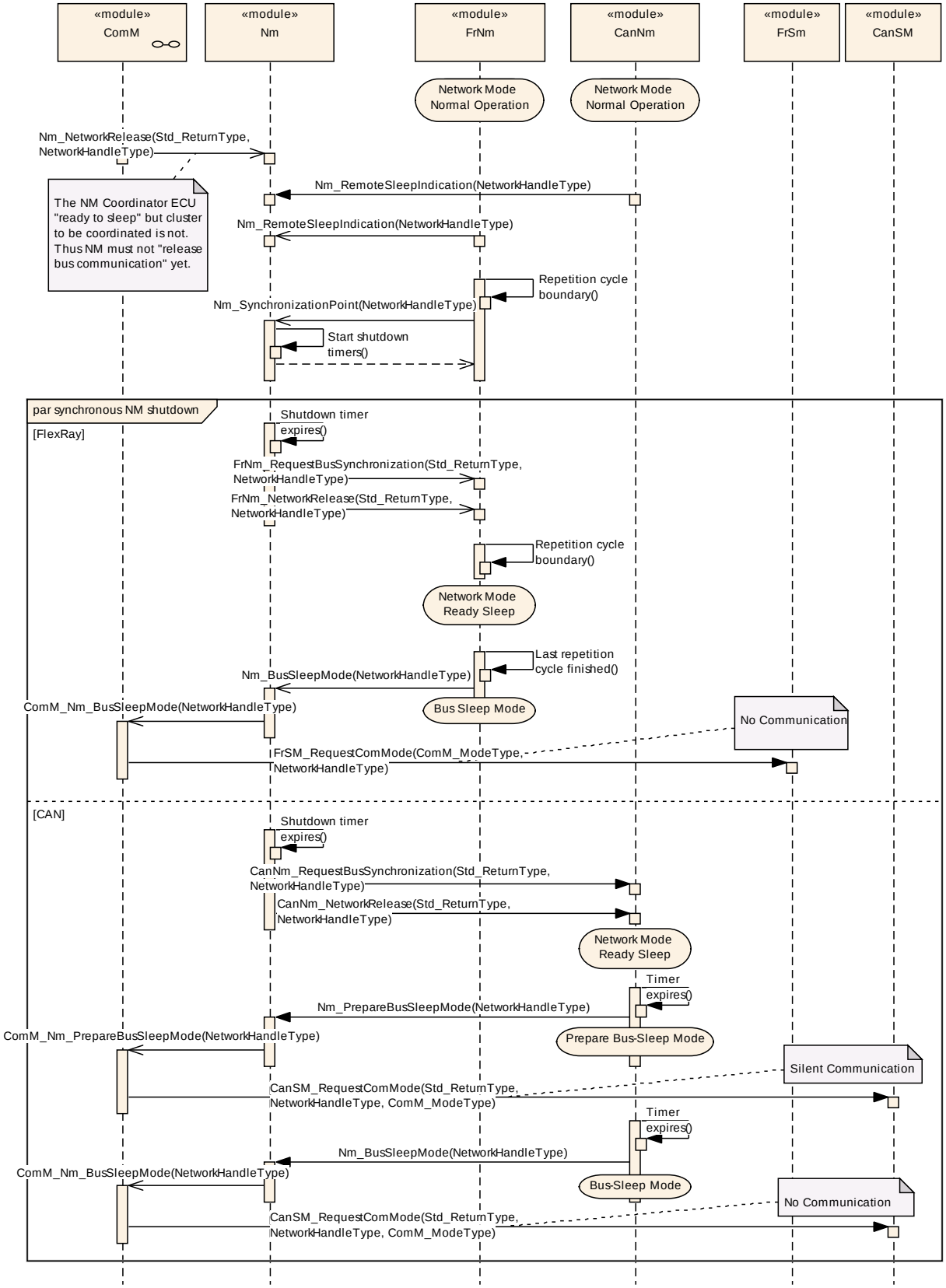

Figure 9.1: Nm Coordination

10 Configuration specification

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

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

10.1 How to read this chapter

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

10.2 Variants

10.2.1 VARIANT-PRE-COMPILE

[SWS_Nm_00120] ⌈ All configuration parameters are configurable at “Pre-compile time”.
Use case: Source code optimizations ⌋*(SRS_BSW_00345, SRS_BSW_00396)*

10.2.2 VARIANT-LINK-TIME

[SWS_Nm_00195] ⌈ 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 ⌋*(SRS_BSW_00344, SRS_BSW_00396)*

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 chapter 8.
Note that the behavior and configuration of Nm is closely dependent on the behavior and configuration of the different bus specific NM modules used.

10.3.1 Nm

Module Name	Nm	
Module Description	The Generic Network Management Interface module	
Post-Build Variant Support	false	
Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
NmChannelConfig	1..*	This container contains the configuration (parameters) of the bus channel(s). The channel parameter shall be harmonized within the whole communication stack.
NmGlobalConfig	1	This container contains all global configuration parameters of the Nm Interface.

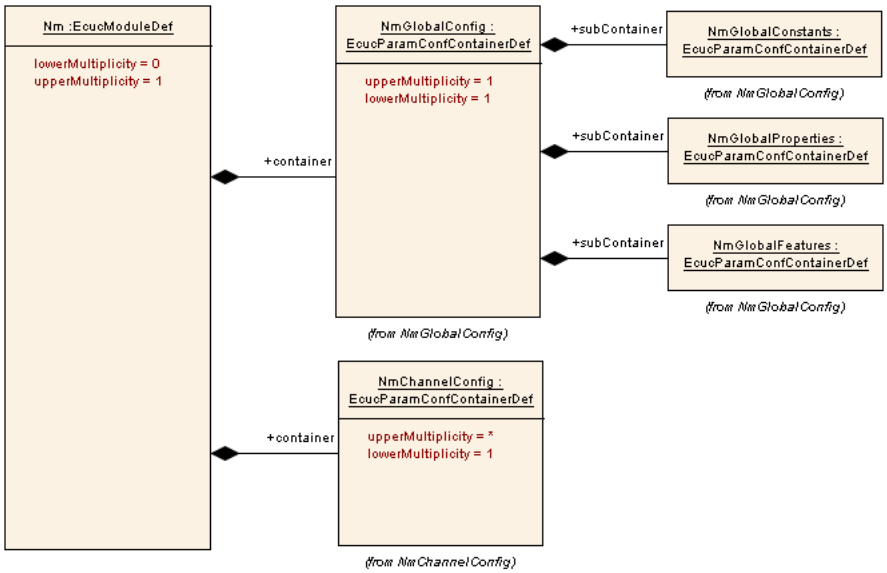

Figure 10.1: Nm configuration container overview

10.4 Global configurable parameters

10.4.1 NmGlobalConfig

NmGlobalConfig

SWS Item	[ECUC_Nm_00196]
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	

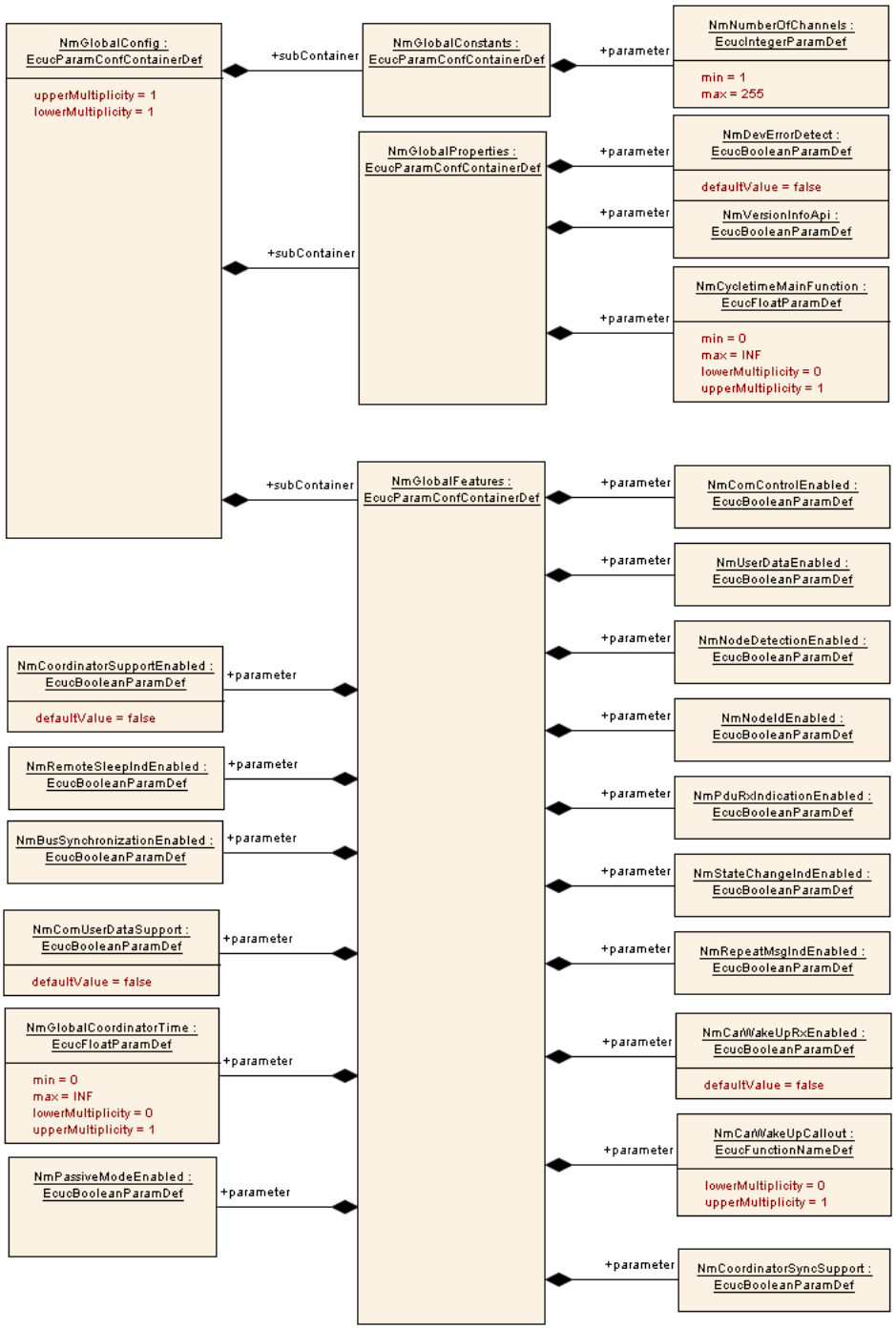

Figure 10.2: NmGlobalConfig overview

10.4.2 NmGlobalConstants

NmGlobalConstants

SWS Item	[ECUC_Nm_00198]

Container Name	NmGlobalConstants
Description	
Configuration Parameters	

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

No Included Containers

10.4.3 NmGlobalProperties

NmGlobalProperties

SWS Item	[ECUC_Nm_00199]
Container Name	NmGlobalProperties
Description	
Configuration Parameters	

Name	NmCycletimeMainFunction [ECUC_Nm_00205]		
Description	The period between successive calls to the Main Function of the NM Interface in seconds.		
Multiplicity	0..1		
Type	EcucFloatParamDef		
Range	0 .. INF		
Default Value			
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	–	

Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	–	
Scope / Dependency	scope: local dependency: If NmCoordinatorSupportEnabled is set to TRUE, then the NmCycletimeMainFunction shall be configured.		

Name	NmDevErrorDetect [ECUC_Nm_00203]		
Description	Switches the Default Error Tracer (Det) detection and notification ON or OFF. • true: enabled (ON). • false: disabled (OFF).		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value	false		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

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

No Included Containers

10.4.4 NmGlobalFeatures

NmGlobalFeatures

SWS Item	[ECUC_Nm_00200]
Container Name	NmGlobalFeatures

Description	
Configuration Parameters	

Name	NmBusSynchronizationEnabled [ECUC_Nm_00208]		
Description	Pre-processor switch for enabling bus synchronization support of the <BusNm>s. This feature is required for NM Coordinator nodes only.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local dependency: This parameter must be enabled if NmCoordinatorSupportEnabled is enabled.		

Name	NmCarWakeUpCallout [ECUC_Nm_00234]		
Description	Name of the callout function to be called if Nm_CarWakeUpIndication() is called. If this parameter is not configured, the NmIf will call BswM_CarWakeUpIndication.		
Multiplicity	0..1		
Type	EcucFunctionNameDef		
Default Value			
Regular Expression			
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	–	
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	–	
Scope / Dependency	scope: local dependency: only available if NmCarWakeUpRxEnabled == TRUE		

Name	NmCarWakeUpRxEnabled [ECUC_Nm_00235]
Description	Enables or disables CWU detection. FALSE - CarWakeUp not supported TRUE - CarWakeUp supported
Multiplicity	1
Type	EcucBooleanParamDef
Default Value	false

Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	–	
Scope / Dependency	scope: local		

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

Name	NmComUserDataSupport [ECUC_Nm_00230]		
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		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	NmCoordinatorSupportEnabled [ECUC_Nm_00206]		
Description	Pre-processor switch for enabling NM Coordinator support.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value	false		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	

Scope / Dependency	scope: local dependency: Only valid if NmRemoteSleepIndEnabled AND NmNumberOfChannels > 1

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

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

Name	NmNodeDetectionEnabled [ECUC_Nm_00212]
Description	Pre-processor switch for enabling the Node Detection feature.
Multiplicity	1
Type	EcucBooleanParamDef
Default Value	
Post-Build Variant Value	false

Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local dependency: Only valid if NmNodeIdEnabled is set to TRUE		

Name	NmNodeIdEnabled [ECUC_Nm_00213]		
Description	Pre-processor switch for enabling transmission of the source node identifier in NM messages.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	NmPassiveModeEnabled [ECUC_Nm_00209]		
Description	Pre-processor switch for enabling support of Passive Mode of the <BusNm>s.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local dependency: NmPassiveModeEnable == ComMNmPassiveModeEnable		

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

Name	NmRemoteSleepIndEnabled [ECUC_Nm_00207]		
Description	Pre-processor switch for enabling Remote Sleep Indication support. This feature is required for a Gateway or Nm Coordinator functionality.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local dependency: It must not be enabled if NmPassiveModeEnabled is enabled.		

Name	NmRepeatMsgIndEnabled [ECUC_Nm_00229]		
Description	Pre-processor switch for enabling the Repeat Message Bit Indication.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	NmStateChangeIndEnabled [ECUC_Nm_00215]		
Description	Pre-processor switch for enabling the Network Management state change notification.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	NmUserDataEnabled [ECUC_Nm_00211]
Description	Pre-processor switch for enabling User Data support.
Multiplicity	1
Type	EcucBooleanParamDef
Default Value	

Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

No Included Containers

10.5 Channel configurable parameters

10.5.1 NmChannelConfig

NmChannelConfig

SWS Item	[ECUC_Nm_00197]
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	

Name	NmActiveCoordinator [ECUC_Nm_00236]		
Description	This parameter indicates whether a NM channel - part of a Nm Coordination cluster - will be coordinated actively (NmActiveCoordinator = TRUE) or passively (NmActiveCoordinator = FALSE).		
Multiplicity	0..1		
Type	EcucBooleanParamDef		
Default Value			
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	

Scope / Dependency	scope: local dependency: If the NmCoordinatorSyncSupport is set to true this feature is available. Only one channel per Coordination cluster can have NmActiveCoordinator = FALSE. This parameter is mandatory if this channel belongs to a Coordination cluster (see ECUC_Nm_00221).

Name	NmChannelSleepMaster [ECUC_Nm_00227]		
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 calls 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 call has been made to Nm_RemoteSleepCancellation.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	–	
Scope / Dependency	scope: local dependency: If the parameter NmCoordClusterIndex is not defined, this parameter is not valid.		

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

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

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

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

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

Name	NmSynchronizingNetwork [ECUC_Nm_00223]		
Description	If this parameter is true, then this network is a synchronizing network for the NM coordination cluster which it belongs to. The network is expected to call Nm_SynchronizationPoint() at regular intervals.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	–	
Scope / Dependency	scope: local dependency: If the parameter NmCoordClusterIndex is not defined, this parameter is not valid. Only one network can be configured as synchronizing network (NmSynchronizingNetwork = TRUE) per coordination cluster (same NmCoordClusterIndex value per channel). NmSynchronizingNetwork can only be set to true if NmActiveCoordinator is true for all networks which have the same NmCoordClusterIndex.		

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

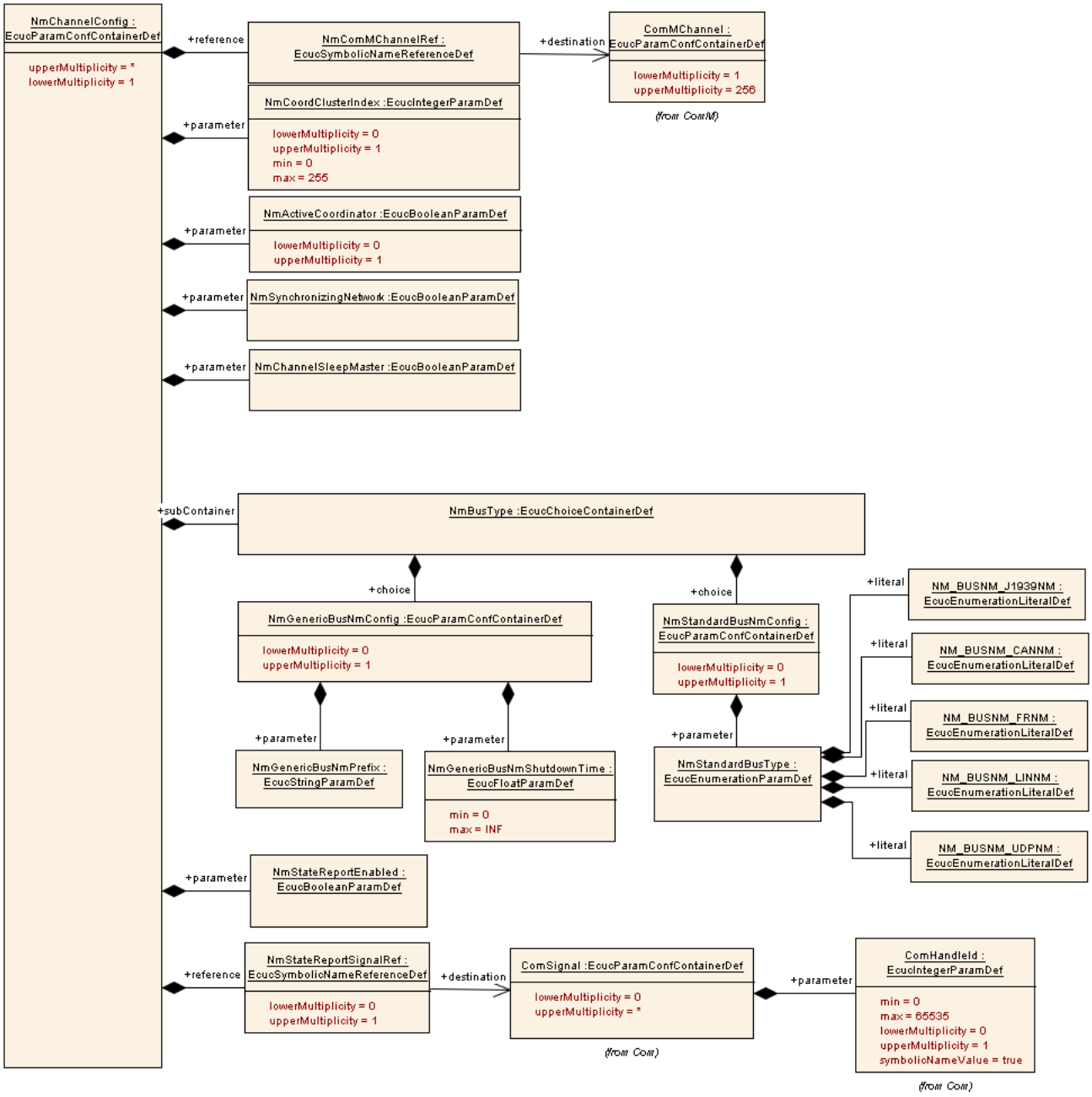

Figure 10.3: NmChannelConfig overview

10.5.2 NmBusType

NmBusType

SWS Item	[ECUC_Nm_00218]
Container Name	NmBusType
Description	
Configuration Parameters	

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

10.5.3 NmGenericBusNmConfig

NmGenericBusNmConfig

SWS Item	[ECUC_Nm_00225]
Container Name	NmGenericBusNmConfig
Description	
Configuration Parameters	

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

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

No Included Containers

10.5.4 NmStandardBusNmConfig

NmStandardBusNmConfig

SWS Item	[ECUC_Nm_00226]
Container Name	NmStandardBusNmConfig
Description	
Configuration Parameters	

Name	NmStandardBusType [ECUC_Nm_00220]		
Description	Identifies the bus type of the channel for standard AUTOSAR <BusNm>s and is used to determine which set of API calls to be called by Nm for the <BusNm>s. Note: The Ethernet bus' NM is UdpNm !		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	NM_BUSNM_CANNM	CAN bus	
	NM_BUSNM_FRNM	FlexRay bus	
	NM_BUSNM_J1939NM	J1939 bus (address claiming)	
	NM_BUSNM_LINNM	LIN bus	
	NM_BUSNM_UDPNM	Ethernet bus (using UDP)	
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	–	
Scope / Dependency	scope: local		

No Included Containers

10.6 Published Information

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

A Not applicable requirements

[SWS_Nm_00999] Not applicable requirements ⌈ These requirements are not applicable to this specification. ⌋*(SRS_Nm_00043, SRS_Nm_00052, SRS_Nm_00053, SRS_Nm_00054, SRS_Nm_00137, SRS_Nm_00142, SRS_Nm_00143, SRS_Nm_00144, SRS_Nm_00145, SRS_Nm_00146, SRS_Nm_00147, SRS_Nm_00148, SRS_Nm_02509, SRS_Nm_02510, SRS_Nm_02517, SRS_Nm_02518, SRS_Nm_02519, SRS_Nm_02520, SRS_Nm_02521, SRS_Nm_02522, SRS_Nm_02523, SRS_Nm_02524, SRS_Nm_02525, SRS_Nm_02526, SRS_Nm_02527, SRS_Nm_02528, SRS_Nm_02529, SRS_Nm_02530, SRS_Nm_02531, SRS_Nm_02532, SRS_Nm_02533, SRS_Nm_02534, SRS_BSW_00004, SRS_BSW_00005, SRS_BSW_00006, SRS_BSW_00007, SRS_BSW_00009, SRS_BSW_00010, SRS_BSW_00158, SRS_BSW_00160, SRS_BSW_00161, SRS_BSW_00162, SRS_BSW_00164, SRS_BSW_00167, SRS_BSW_00168, SRS_BSW_00170, SRS_BSW_00172, SRS_BSW_00300, SRS_BSW_00302, SRS_BSW_00304, SRS_BSW_00305, SRS_BSW_00306, SRS_BSW_00307, SRS_BSW_00308, SRS_BSW_00309, SRS_BSW_00310, SRS_BSW_00312, SRS_BSW_00314, SRS_BSW_00318, SRS_BSW_00321, SRS_BSW_00325, SRS_BSW_00328, SRS_BSW_00331, SRS_BSW_00334, SRS_BSW_00335, SRS_BSW_00336, SRS_BSW_00339, SRS_BSW_00341, SRS_BSW_00342, SRS_BSW_00343, SRS_BSW_00346, SRS_BSW_00347, SRS_BSW_00350, SRS_BSW_00351, SRS_BSW_00360, SRS_BSW_00361, SRS_BSW_00371, SRS_BSW_00374, SRS_BSW_00375, SRS_BSW_00377, SRS_BSW_00378, SRS_BSW_00379, SRS_BSW_00380, SRS_BSW_00383, SRS_BSW_00388, SRS_BSW_00389, SRS_BSW_00390, SRS_BSW_00392, SRS_BSW_00393, SRS_BSW_00394, SRS_BSW_00395, SRS_BSW_00397, SRS_BSW_00398, SRS_BSW_00399, SRS_BSW_00400, SRS_BSW_00401, SRS_BSW_00402, SRS_BSW_00403, SRS_BSW_00404, SRS_BSW_00406, SRS_BSW_00408, SRS_BSW_00409, SRS_BSW_00410, SRS_BSW_00411, SRS_BSW_00413, SRS_BSW_00415, SRS_BSW_00416, SRS_BSW_00417, SRS_BSW_00422, SRS_BSW_00423, SRS_BSW_00424, SRS_BSW_00426, SRS_BSW_00427, SRS_BSW_00428, SRS_BSW_00429, SRS_BSW_00432, SRS_BSW_00433, SRS_BSW_00437, SRS_BSW_00438, SRS_BSW_00439, SRS_BSW_00440, SRS_BSW_00441, SRS_BSW_00447, SRS_BSW_00448, SRS_BSW_00449, SRS_BSW_00451, SRS_BSW_00452, SRS_BSW_00453, SRS_BSW_00454, SRS_BSW_00456, SRS_BSW_00457, SRS_BSW_00458, SRS_BSW_00459, SRS_BSW_00460, SRS_BSW_00461, SRS_BSW_00462, SRS_BSW_00463, SRS_BSW_00464, SRS_BSW_00465, SRS_BSW_00466, SRS_BSW_00467, SRS_BSW_00469, SRS_BSW_00470, SRS_BSW_00471, SRS_BSW_00472, SRS_BSW_00473)*