Document Title	Specification of CAN Network Management
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	013
Document Classification	Standard
Document Version	3.6.0
Document Status	Final
Part of Release	4.1
Revision	3

Document Change History

Date	Version	Changed by	Change Description
31.03.2014	3.6.0	AUTOSAR Release Management	• Rewording and improving Partial Networking Algorithm Requirements • Remote Sleep Indication Timeout handling corrected • Network Release handling during communication control clarified
31.10.2013	3.5.0	AUTOSAR Release Management	• Fixed Message Cycle Time Offset Handling • Corrected Active Wakeup Handling • Editorial changes • Removed chapter(s) on change documentation
12.02.2013	3.4.0	AUTOSAR Administration	• Partial Network Handling corrected • Coordinator Support improved • Start-up Handling from Prepare-Bus Sleep clarified
17.11.2011	3.3.0	AUTOSAR Administration	• Support for Partial Networking • Support for Car Wakeup • Immediate Transmission of NM-PDUs • Support of a coordinated shutdown with multiple connected gateways
19.10.2010	3.2.0	AUTOSAR Administration	• Changed Signature of RxIndication and TriggerTransmit • Faster NM wakeup
07.12.2009	3.1.0	AUTOSAR Administration	• Nm User Data accessible through PduR • Changed PDU handle ID exchange with CanIf • No more instance specific CanNm_MainFunction() APIs • Legal disclaimer revised

Document Change History

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

Disclaimer

This specification and the material contained in it, as released by AUTOSAR is for the purpose of information only. AUTOSAR and the companies that have contributed to it shall not be liable for any use of the specification.

The material contained in this specification is protected by copyright and other types of Intellectual Property Rights. The commercial exploitation of the material contained in this specification requires a license to such Intellectual Property Rights.

This specification may be utilized or reproduced without any modification, in any form or by any means, for informational purposes only.
For any other purpose, no part of the specification may be utilized or reproduced, in any form or by any means, without permission in writing from the publisher.

The AUTOSAR specifications have been developed for automotive applications only. They have neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Advice for users

AUTOSAR Specification Documents may contain exemplary items (exemplary reference models, "use cases", and/or references to exemplary technical solutions, devices, processes or software).

Any such exemplary items are contained in the Specification Documents for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such Specification Documents, nor any later documentation of AUTOSAR conformance of products actually implementing such exemplary items, imply that intellectual property rights covering such exemplary items are licensed under the same rules as applicable to the AUTOSAR Standard.

Table of Contents

1 Introduction and Functional Overview

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

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

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

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

2 Acronyms and abbreviations

Acronym/abbreviation:	***Description:***
API	Application Programming Interface
BSW	Basic Software
DET	Development Error Tracer
CanIf	Abbreviation for the CAN Interface
CanNm	Abbreviation for CAN Network Management
NM	Network Management
PDU	Protocol Data Unit
SDU	Service Data Unit

Term	***Description:***
“PDU transmission ability is disabled”	This means that the Network Management PDU transmission has been disabled by the service CanNm_DisableCommunication.
“Repeat Message Request Bit Indication”	CanNm_RxIndication finds the RptMsgRequest set in the Control Bit Vector of a received Network Management PDU.
“PN filter mask”	Vector of filter mask bytes defined by configuration container(s) CanNmPnFilterMaskByte

3 Related documentation

3.1 Input documents

[1] AUTOSAR Layered Software Architecture
AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf

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

[3] Requirements on Network Management
AUTOSAR_SRS_NetworkManagement.pdf

[4] Specification of CAN Interface
AUTOSAR_SWS_CANInterface.pdf

[5] Specification of FlexRay Network Management
AUTOSAR_SWS_FlexRayNetworkManagement.pdf

[6] Specification of Communication Stack Types
AUTOSAR_SWS_CommunicationStackTypes.pdf

[7] Specification of ECU Configuration
AUTOSAR_TPS_ECUConfiguration.pdf

[8] Specification of BSW Scheduler
AUTOSAR_SWS_BSW_Scheduler.pdf

[9] Specification of Generic Network Management Interface
AUTOSAR_SWS_NetworkManagementInterface.pdf

[10] Specification of Communication Manager
AUTOSAR_SWS_ComManager.pdf

[11] Specification of ECU State Manager
AUTOSAR_SWS_ECUStateManager.pdf

[12] Specification of Operating System
AUTOSAR_SWS_OS.pdf

[13] Specification of Diagnostic Event Manager
AUTOSAR_SWS_DiagnosticEventManager.pdf

[14] Specification of Development Error Tracer
AUTOSAR_SWS_DevelopmentErrorTracer.pdf

[15] Specification of Standard Types
AUTOSAR_SWS_StandardTypes.pdf

[16] Specification of Platform Types
AUTOSAR_SWS_PlatformTypes.pdf

[17] Specification of Compiler Abstraction
AUTOSAR_SWS_CompilerAbstraction.pdf

[18] Basic Software Module Description Template,
AUTOSAR_TPS_BSWModuleDescriptionTemplate.pdf

[19] List of Basic Software Modules
AUTOSAR_TR_BSWModuleList.pdf

[20] General Specification of Basic Software Modules
AUTOSAR_SWS_BSWGeneral.pdf

3.2 Related standards and norms

Not available.

3.3 Related specification

AUTOSAR provides a General Specification on Basic Software modules [20] (SWS BSW General), which is also valid for CAN Network Management.

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

4 Constraints and assumptions

4.1 Limitations

1. One channel of CanNm is associated with only one network management cluster in one network. One network management cluster can have only one channel of CanNm in one node.
2. One channel of CanNm is associated with only one network within the same ECU.
3. CanNm is only applicable for CAN systems.

The Figure 4-1 presents an AUTOSAR Network Management stack within an example ECU belonging to two CanNm clusters.

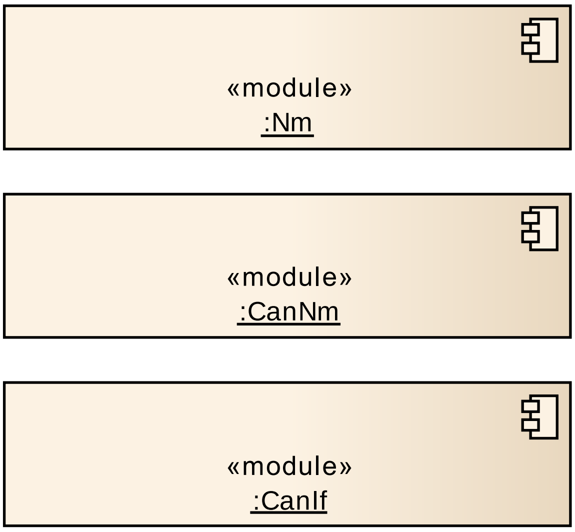


Figure 4-1 AUTOSAR NM Stack on CAN

The CanNm module is only applicable for CAN systems.

4.2 Applicability to car domains

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

5 Dependencies to other modules

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

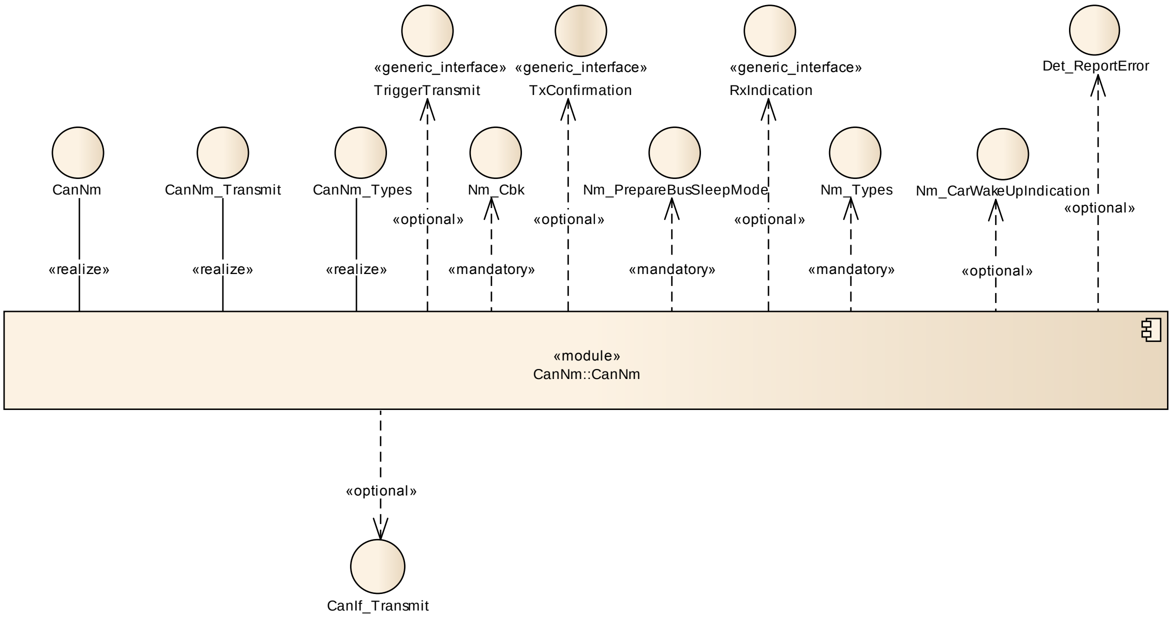


Figure 5-1 Dependencies to other modules

5.1 File Structure

5.1.1 Code File Structure

Please refer to the chapter 5.1.6 Code file structure in “SWS_BSWGeneral”

5.1.2 Header File Structure

Please refer to the chapter 5.1.7 Header file structure in “SWS_BSWGeneral”

[SWS_CanNm_00305]「`ComStack_Types.h` shall be included.

Note: The following header files are indirectly included by `ComStack_Types.h`

- `Std_Types.h` (for AUTOSAR standard types)
- `Platform_Types.h` (for platform specific types)
- `Compiler.h` (for compiler specific language extensions)」(SRS_BSW_00348, SRS_BSW_00353, SRS_BSW_00361, SRS_BSW_00301)

[SWS_CanNm_00307]「`Nm_Cbk.h` shall be included for CanNm specific callbacks of Generic Network Management Interface.」(SRS_BSW_00301)

[SWS_CanNm_00308]「`Det.h` shall be included for interfacing the DET if DET usage is configured.」(SRS_BSW_00301)

[SWS_CanNm_00309]「`NmStack_Types.h` shall be included for common network management types.」(BSW0 0301)

[SWS_CanNm_00312]「`CanIf.h` shall be included for interfacing the CanIf.」(SRS_BSW_00301)

[SWS_CanNm_00326]「`PduR_CanNm.h` shall be included if COM user data support is enabled.」()

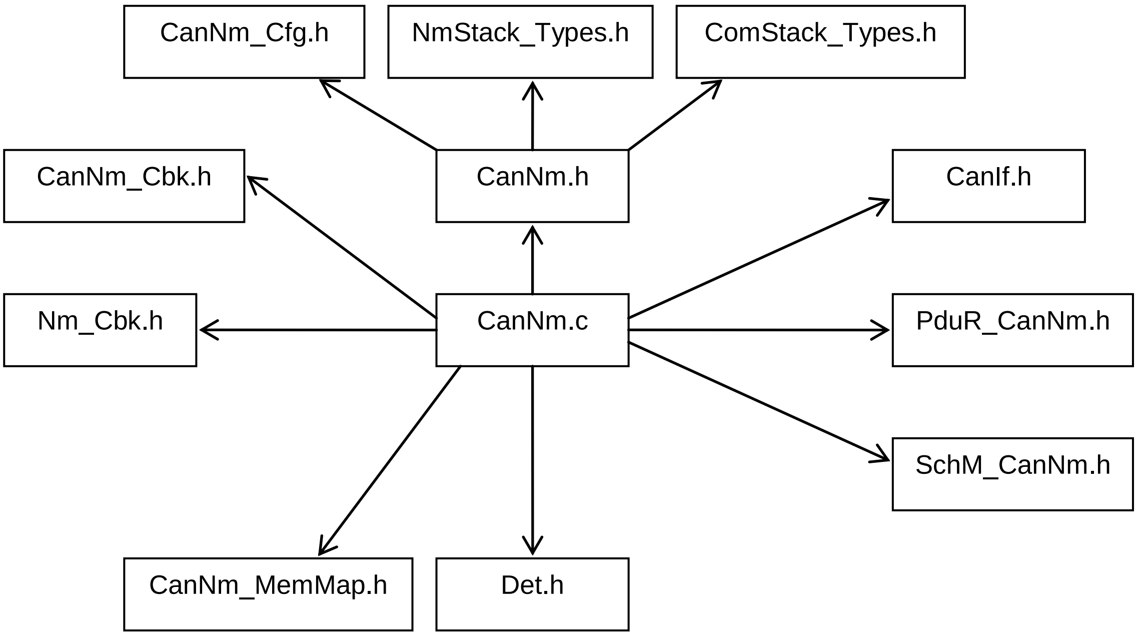


Figure 5-2 Header File Structure

6 Requirements traceability

Requirement	Description	Satisfied by
-	-	SWS_CanNm_00005
-	-	SWS_CanNm_00006
-	-	SWS_CanNm_00013
-	-	SWS_CanNm_00014
-	-	SWS_CanNm_00023
-	-	SWS_CanNm_00025
-	-	SWS_CanNm_00032
-	-	SWS_CanNm_00033
-	-	SWS_CanNm_00035
-	-	SWS_CanNm_00037
-	-	SWS_CanNm_00039
-	-	SWS_CanNm_00040
-	-	SWS_CanNm_00045
-	-	SWS_CanNm_00051
-	-	SWS_CanNm_00052
-	-	SWS_CanNm_00060
-	-	SWS_CanNm_00061
-	-	SWS_CanNm_00064
-	-	SWS_CanNm_00065
-	-	SWS_CanNm_00066
-	-	SWS_CanNm_00069
-	-	SWS_CanNm_00071
-	-	SWS_CanNm_00072
-	-	SWS_CanNm_00073
-	-	SWS_CanNm_00074
-	-	SWS_CanNm_00075
-	-	SWS_CanNm_00085
-	-	SWS_CanNm_00086
-	-	SWS_CanNm_00087
-	-	SWS_CanNm_00088
-	-	SWS_CanNm_00089
-	-	SWS_CanNm_00092
-	-	SWS_CanNm_00093
-	-	SWS_CanNm_00094
-	-	SWS_CanNm_00096
-	-	SWS_CanNm_00097

-	-	SWS_CanNm_00098
-	-	SWS_CanNm_00099
-	-	SWS_CanNm_00100
-	-	SWS_CanNm_00101
-	-	SWS_CanNm_00102
-	-	SWS_CanNm_00103
-	-	SWS_CanNm_00104
-	-	SWS_CanNm_00105
-	-	SWS_CanNm_00106
-	-	SWS_CanNm_00107
-	-	SWS_CanNm_00108
-	-	SWS_CanNm_00109
-	-	SWS_CanNm_00110
-	-	SWS_CanNm_00111
-	-	SWS_CanNm_00112
-	-	SWS_CanNm_00113
-	-	SWS_CanNm_00114
-	-	SWS_CanNm_00115
-	-	SWS_CanNm_00116
-	-	SWS_CanNm_00117
-	-	SWS_CanNm_00118
-	-	SWS_CanNm_00119
-	-	SWS_CanNm_00120
-	-	SWS_CanNm_00121
-	-	SWS_CanNm_00122
-	-	SWS_CanNm_00123
-	-	SWS_CanNm_00124
-	-	SWS_CanNm_00126
-	-	SWS_CanNm_00127
-	-	SWS_CanNm_00128
-	-	SWS_CanNm_00129
-	-	SWS_CanNm_00132
-	-	SWS_CanNm_00133
-	-	SWS_CanNm_00135
-	-	SWS_CanNm_00137
-	-	SWS_CanNm_00138
-	-	SWS_CanNm_00141
-	-	SWS_CanNm_00143
-	-	SWS_CanNm_00144

-	-	SWS_CanNm_00145
-	-	SWS_CanNm_00146
-	-	SWS_CanNm_00147
-	-	SWS_CanNm_00149
-	-	SWS_CanNm_00150
-	-	SWS_CanNm_00151
-	-	SWS_CanNm_00152
-	-	SWS_CanNm_00153
-	-	SWS_CanNm_00154
-	-	SWS_CanNm_00156
-	-	SWS_CanNm_00157
-	-	SWS_CanNm_00158
-	-	SWS_CanNm_00159
-	-	SWS_CanNm_00160
-	-	SWS_CanNm_00161
-	-	SWS_CanNm_00162
-	-	SWS_CanNm_00163
-	-	SWS_CanNm_00166
-	-	SWS_CanNm_00168
-	-	SWS_CanNm_00170
-	-	SWS_CanNm_00172
-	-	SWS_CanNm_00173
-	-	SWS_CanNm_00174
-	-	SWS_CanNm_00175
-	-	SWS_CanNm_00176
-	-	SWS_CanNm_00177
-	-	SWS_CanNm_00178
-	-	SWS_CanNm_00179
-	-	SWS_CanNm_00180
-	-	SWS_CanNm_00181
-	-	SWS_CanNm_00185
-	-	SWS_CanNm_00187
-	-	SWS_CanNm_00189
-	-	SWS_CanNm_00190
-	-	SWS_CanNm_00191
-	-	SWS_CanNm_00192
-	-	SWS_CanNm_00193
-	-	SWS_CanNm_00194
-	-	SWS_CanNm_00196

-	-	SWS_CanNm_00197
-	-	SWS_CanNm_00198
-	-	SWS_CanNm_00199
-	-	SWS_CanNm_00202
-	-	SWS_CanNm_00203
-	-	SWS_CanNm_00206
-	-	SWS_CanNm_00208
-	-	SWS_CanNm_00211
-	-	SWS_CanNm_00214
-	-	SWS_CanNm_00215
-	-	SWS_CanNm_00216
-	-	SWS_CanNm_00217
-	-	SWS_CanNm_00219
-	-	SWS_CanNm_00220
-	-	SWS_CanNm_00221
-	-	SWS_CanNm_00222
-	-	SWS_CanNm_00223
-	-	SWS_CanNm_00224
-	-	SWS_CanNm_00227
-	-	SWS_CanNm_00228
-	-	SWS_CanNm_00229
-	-	SWS_CanNm_00231
-	-	SWS_CanNm_00232
-	-	SWS_CanNm_00234
-	-	SWS_CanNm_00235
-	-	SWS_CanNm_00237
-	-	SWS_CanNm_00238
-	-	SWS_CanNm_00245
-	-	SWS_CanNm_00246
-	-	SWS_CanNm_00247
-	-	SWS_CanNm_00248
-	-	SWS_CanNm_00249
-	-	SWS_CanNm_00250
-	-	SWS_CanNm_00251
-	-	SWS_CanNm_00252
-	-	SWS_CanNm_00253
-	-	SWS_CanNm_00254
-	-	SWS_CanNm_00255
-	-	SWS_CanNm_00256

-	-	SWS_CanNm_00257
-	-	SWS_CanNm_00259
-	-	SWS_CanNm_00260
-	-	SWS_CanNm_00261
-	-	SWS_CanNm_00262
-	-	SWS_CanNm_00263
-	-	SWS_CanNm_00264
-	-	SWS_CanNm_00265
-	-	SWS_CanNm_00266
-	-	SWS_CanNm_00267
-	-	SWS_CanNm_00268
-	-	SWS_CanNm_00269
-	-	SWS_CanNm_00270
-	-	SWS_CanNm_00271
-	-	SWS_CanNm_00272
-	-	SWS_CanNm_00273
-	-	SWS_CanNm_00274
-	-	SWS_CanNm_00275
-	-	SWS_CanNm_00276
-	-	SWS_CanNm_00277
-	-	SWS_CanNm_00279
-	-	SWS_CanNm_00281
-	-	SWS_CanNm_00282
-	-	SWS_CanNm_00283
-	-	SWS_CanNm_00284
-	-	SWS_CanNm_00285
-	-	SWS_CanNm_00292
-	-	SWS_CanNm_00295
-	-	SWS_CanNm_00297
-	-	SWS_CanNm_00298
-	-	SWS_CanNm_00314
-	-	SWS_CanNm_00315
-	-	SWS_CanNm_00324
-	-	SWS_CanNm_00325
-	-	SWS_CanNm_00326
-	-	SWS_CanNm_00327
-	-	SWS_CanNm_00328
-	-	SWS_CanNm_00329
-	-	SWS_CanNm_00330

-	-	SWS_CanNm_00331
-	-	SWS_CanNm_00332
-	-	SWS_CanNm_00333
-	-	SWS_CanNm_00334
-	-	SWS_CanNm_00335
-	-	SWS_CanNm_00336
-	-	SWS_CanNm_00338
-	-	SWS_CanNm_00339
-	-	SWS_CanNm_00340
-	-	SWS_CanNm_00341
-	-	SWS_CanNm_00342
-	-	SWS_CanNm_00343
-	-	SWS_CanNm_00344
-	-	SWS_CanNm_00345
-	-	SWS_CanNm_00346
-	-	SWS_CanNm_00348
-	-	SWS_CanNm_00401
-	-	SWS_CanNm_00402
-	-	SWS_CanNm_00403
-	-	SWS_CanNm_00404
-	-	SWS_CanNm_00405
-	-	SWS_CanNm_00406
-	-	SWS_CanNm_00407
-	-	SWS_CanNm_00408
-	-	SWS_CanNm_00409
-	-	SWS_CanNm_00410
-	-	SWS_CanNm_00411
-	-	SWS_CanNm_00412
-	-	SWS_CanNm_00413
-	-	SWS_CanNm_00414
-	-	SWS_CanNm_00415
-	-	SWS_CanNm_00416
-	-	SWS_CanNm_00417
-	-	SWS_CanNm_00419
-	-	SWS_CanNm_00420
-	-	SWS_CanNm_00421
-	-	SWS_CanNm_00422
-	-	SWS_CanNm_00423
-	-	SWS_CanNm_00424

-	-	SWS_CanNm_00425
-	-	SWS_CanNm_00426
-	-	SWS_CanNm_00427
-	-	SWS_CanNm_00428
-	-	SWS_CanNm_00429
-	-	SWS_CanNm_00430
-	-	SWS_CanNm_00431
-	-	SWS_CanNm_00432
-	-	SWS_CanNm_00433
-	-	SWS_CanNm_00434
-	-	SWS_CanNm_00435
-	-	SWS_CanNm_00436
-	-	SWS_CanNm_00437
-	-	SWS_CanNm_00438
-	-	SWS_CanNm_00439
-	-	SWS_CanNm_00442
-	-	SWS_CanNm_00443
-	-	SWS_CanNm_00444
-	-	SWS_CanNm_00445
-	-	SWS_CanNm_00446
-	-	SWS_CanNm_00447
-	-	SWS_CanNm_00450
-	-	SWS_CanNm_00454
BSW00301	-	SWS_CanNm_00309
BSW00434	-	SWS_CanNm_00999
BSW136	-	SWS_CanNm_00999
BSW139	-	SWS_CanNm_00999
BSW140	-	SWS_CanNm_00999
SRS_BSW_00005	Modules of the æC Abstraction Layer (MCAL) may not have hard coded horizontal interfaces	SWS_CanNm_00999
SRS_BSW_00006	The source code of software modules above the æC Abstraction Layer (MCAL) shall not be processor and compiler dependent.	SWS_CanNm_00999
SRS_BSW_00010	The memory consumption of all Basic SW Modules shall be documented for a defined configuration for all supported platforms.	SWS_CanNm_00999
SRS_BSW_00160	Configuration files of AUTOSAR Basic SW module shall be readable for human beings	SWS_CanNm_00999
SRS_BSW_00161	The AUTOSAR Basic Software shall provide a microcontroller abstraction layer which provides a standardized interface to	SWS_CanNm_00999

	higher software layers	
SRS_BSW_00162	The AUTOSAR Basic Software shall provide a hardware abstraction layer	SWS_CanNm_00999
SRS_BSW_00164	The Implementation of interrupt service routines shall be done by the Operating System, complex drivers or modules	SWS_CanNm_00999
SRS_BSW_00168	SW components shall be tested by a function defined in a common API in the Basis-SW	SWS_CanNm_00999
SRS_BSW_00170	The AUTOSAR SW Components shall provide information about their dependency from faults, signal qualities, driver demands	SWS_CanNm_00999
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_CanNm_00999
SRS_BSW_00301	All AUTOSAR Basic Software Modules shall only import the necessary information	SWS_CanNm_00305, SWS_CanNm_00307, SWS_CanNm_00308, SWS_CanNm_00312
SRS_BSW_00305	Data types naming convention	SWS_CanNm_00999
SRS_BSW_00306	AUTOSAR Basic Software Modules shall be compiler and platform independent	SWS_CanNm_00999
SRS_BSW_00307	Global variables naming convention	SWS_CanNm_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_CanNm_00999
SRS_BSW_00312	Shared code shall be reentrant	SWS_CanNm_00999
SRS_BSW_00314	All internal driver modules shall separate the interrupt frame definition from the service routine	SWS_CanNm_00999
SRS_BSW_00321	The version numbers of AUTOSAR Basic Software Modules shall be enumerated according specific rules	SWS_CanNm_00999
SRS_BSW_00323	All AUTOSAR Basic Software Modules shall check passed API parameters for validity	SWS_CanNm_00244
SRS_BSW_00325	The runtime of interrupt service routines and functions that are running in interrupt context shall be kept short	SWS_CanNm_00999
SRS_BSW_00326	-	SWS_CanNm_00999
SRS_BSW_00328	All AUTOSAR Basic Software Modules shall avoid the duplication of code	SWS_CanNm_00999
SRS_BSW_00330	It shall be allowed to use macros instead of functions where source code is used and runtime is critical	SWS_CanNm_00999
SRS_BSW_00331	All Basic Software Modules shall strictly separate error and status information	SWS_CanNm_00999

SRS_BSW_00333	For each callback function it shall be specified if it is called from interrupt context or not	SWS_CanNm_00999
SRS_BSW_00334	All Basic Software Modules shall provide an XML file that contains the meta data	SWS_CanNm_00999
SRS_BSW_00335	Status values naming convention	SWS_CanNm_00999
SRS_BSW_00336	Basic SW module shall be able to shutdown	SWS_CanNm_00999
SRS_BSW_00341	Module documentation shall contains all needed informations	SWS_CanNm_00999
SRS_BSW_00347	A Naming seperation of different instances of BSW drivers shall be in place	SWS_CanNm_00999
SRS_BSW_00348	All AUTOSAR standard types and constants shall be placed and organized in a standard type header file	SWS_CanNm_00305
SRS_BSW_00353	All integer type definitions of target and compiler specific scope shall be placed and organized in a single type header	SWS_CanNm_00305
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_CanNm_00305
SRS_BSW_00375	Basic Software Modules shall report wake-up reasons	SWS_CanNm_00999
SRS_BSW_00377	A Basic Software Module can return a module specific types	SWS_CanNm_00999
SRS_BSW_00387	The Basic Software Module specifications shall specify how the callback function is to be implemented	SWS_CanNm_00999
SRS_BSW_00410	Compiler switches shall have defined values	SWS_CanNm_00999
SRS_BSW_00413	An index-based accessing of the instances of BSW modules shall be done	SWS_CanNm_00999
SRS_BSW_00415	Interfaces which are provided exclusively for one module shall be separated into a dedicated header file	SWS_CanNm_00999
SRS_BSW_00416	The sequence of modules to be initialized shall be configurable	SWS_CanNm_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_CanNm_00999
SRS_BSW_00423	BSW modules with AUTOSAR interfaces shall be describable with the means of the SW-C Template	SWS_CanNm_00999
SRS_BSW_00424	BSW module main processing functions shall not be allowed to enter a wait state	SWS_CanNm_00999
SRS_BSW_00425	The BSW module description template shall provide means to model the defined trigger conditions of schedulable objects	SWS_CanNm_00999
SRS_BSW_00426	BSW Modules shall ensure data	SWS_CanNm_00999

	consistency of data which is shared between BSW modules	
SRS_BSW_00427	ISR functions shall be defined and documented in the BSW module description template	SWS_CanNm_00999
SRS_BSW_00429	BSW modules shall be only allowed to use OS objects and/or related OS services	SWS_CanNm_00999
SRS_BSW_00432	Modules should have separate main processing functions for read/receive and write/transmit data path	SWS_CanNm_00999
SRS_Nm_00043	NM shall not prohibit bus traffic with NM not being initialized	SWS_CanNm_00999
SRS_Nm_00046	It shall be possible to trigger the startup of all Nodes at any Point in Time.	SWS_CanNm_00999
SRS_Nm_00048	NM shall put the communication controller into sleep mode if there is no bus communication	SWS_CanNm_00999
SRS_Nm_00050	The NM shall provide the current state of NM	SWS_CanNm_00999
SRS_Nm_00052	The NM interface shall signal to the application that all other ECUs are ready to sleep.	SWS_CanNm_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_CanNm_00999
SRS_Nm_00144	NM shall support communication clusters of up to 64 ECUs	SWS_CanNm_00999
SRS_Nm_00147	The NM algorithm shall be processor independent.	SWS_CanNm_00999
SRS_Nm_00151	The Network Management algorithm shall allow any node to integrate into an already running NM cluster	SWS_CanNm_00999
SRS_Nm_00153	The Network Management shall optionally provide a possibility to detect present nodes	SWS_CanNm_00999
SRS_Nm_00154	The Network Management API shall be independent from the communication bus	SWS_CanNm_00999
SRS_Nm_02509	The NM interface shall signal to the application that at least one other ECUs is not ready to sleep anymore.	SWS_CanNm_00999
SRS_Nm_02516	All AUTOSAR NM instances shall support the NM Coordinator functionality including Bus synchronization on demand	SWS_CanNm_00130, SWS_CanNm_00226, SWS_CanNm_00280

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

Requirement	***Satisfied by***
[SRS_BSW_00344] Reference to link-time configuration	Ok; see chapter 10.2
[SRS_BSW_00404] Reference to post build time configuration	Ok; see Chapter 10.2

[SRS_BSW_00405] Reference to multiple configuration sets	Ok; see Chapter 10.2
[SRS_BSW_00345] Pre-compile-time configuration	Ok; see SWS_CanNm_00302
[SRS_BSW_00159] Tool-based configuration	Ok; see Chapter 10.2
[SRS_BSW_00167] Static configuration checking	Ok; see Chapter 10.2
[SRS_BSW_00170] Data for reconfiguration of AUTOSAR SW-Components	N/a (CanNm is no SW-C)
[SRS_BSW_00171] Configurability of optional functionality	Ok; see Chapter 10.2
[SRS_BSW_00380] Separate C-Files for configuration parameters	Ok; see SWS_CanNm_00299, SWS_CanNm_00300, SWS_CanNm_00301
[SRS_BSW_00419] Separate C-Files for pre-compile time configuration parameters	Ok; see SWS_CanNm_00299
[SRS_BSW_00381] Separate configuration header file for pre-compile time parameters	Ok; see SWS_CanNm_00302
[SRS_BSW_00412] Separate H-File for configuration parameters	Ok; see SWS_CanNm_00302, SWS_CanNm_00303, SWS_CanNm_00304
[SRS_BSW_00383] List dependencies of configuration files	Ok; see SWS_CanNm_00313
[SRS_BSW_00384] List dependencies to other modules	Ok; see Figure 5-1
[SRS_BSW_00387] Specify the configuration class of callback function	n/a (Callback functions are not configurable)
[SRS_BSW_00388] Introduce containers	Ok; see Chapter 10.2
[SRS_BSW_00389] Containers shall have names	Ok; see Chapter 10.2
[SRS_BSW_00390] Parameter content shall be unique within the module	Ok; see Chapter 10.2
[SRS_BSW_00391] Parameter shall have unique names	Ok; see Chapter 10.2
[SRS_BSW_00392] Parameters shall have a type	Ok; see Chapter 10.2
[SRS_BSW_00393] Parameters shall have a range	Ok; see Chapter 10.2
[SRS_BSW_00394] Specify the scope of the parameters	Ok; see Chapter 10.2
[SRS_BSW_00395] List the required parameters (per parameter)	Ok; see Chapter 10.2
[SRS_BSW_00396] Configuration classes	Ok; see Chapter 10.2
[SRS_BSW_00397] Pre-compile-time parameters	Ok; see Chapter 10.2
[SRS_BSW_00398] Link-time parameters	Ok; see Chapter 10.2
[SRS_BSW_00399] Loadable Post-build time parameters	Ok; see Chapter 10.2
[SRS_BSW_00400] Selectable Post-build time parameters	Ok; see Chapter 10.2
[SRS_BSW_00402] Published information	Ok; see Chapter 10.3.5
[SRS_BSW_00375] Notification of wake-up reason	n/a (CanNm does not wake-up an ECU)
[SRS_BSW_00101] Initialization interface	Ok; see chapter 8.3.1
[SRS_BSW_00416] Sequence of Initialization	n/a (sequence is defined by ComM)
[SRS_BSW_00406] Check module initialization	Ok; see chapter 7.19
[SRS_BSW_00168] Diagnostic Interface of SW components	n/a (diagnostics for CanNm not required)
[SRS_BSW_00407] Function to read out published parameters	Ok; see chapter 10.3.5
[SRS_BSW_00423] Usage of SW-C template to describe BSW modules with AUTOSAR Interfaces	n/a (CanNm has no interface to the RTE)
[SRS_BSW_00424] BSW main processing function task allocation	n/a (CanNm scheduled function is called by the BSW scheduler)
[SRS_BSW_00425] Trigger conditions for schedulable objects	n/a (implementation specific)
[SRS_BSW_00426] Exclusive areas in BSW modules	n/a (implementation specific)

[SRS_BSW_00427] ISR description for BSW modules	n/a (implementation specific)
[SRS_BSW_00428] Execution order dependencies of main processing functions	Ok; see chapter 7.17
[SRS_BSW_00429] Restricted BSW OS functionality access	n/a (none of these services are used by the CanNm)
[BSW00431] The BSW Scheduler module implements task bodies	Ok; see chapter 7.17
[SRS_BSW_00432] Modules should have separate main processing functions for read/receive and write/transmit data path	n/a (transmission and reception is handled in CanNm_MainFunction)
[SRS_BSW_00433] Calling of main processing functions	Ok; see chapter 7.17
[BSW00434] The Schedule Module shall provide an API for exclusive areas	n/a (implementation specific)
[SRS_BSW_00336] Shutdown interface	n/a (no shutdown interface needed)
[SRS_BSW_00337] Classification of errors	Ok; see chapter 7.14
[SRS_BSW_00338] Detection and Reporting of development errors	Ok; see chapter 7.14 and 10.2
[SRS_BSW_00369] Do not return development error codes via API	Ok; see chapter 7.19
[SRS_BSW_00339] Reporting of production relevant error status	Ok; see chapter 7.14
[SRS_BSW_00417] Reporting of Error Events by Non-Basic Software	n/a (CanNm is no SW-C)
[SRS_BSW_00323] API parameter checking	Ok; see SWS_CanNm_00243, SWS_CanNm_00244
[SRS_BSW_00004] Version check	Ok;
[SRS_BSW_00409] Header files for production code error IDs	Ok; see CANNM207
[SRS_BSW_00385] List possible error notifications	Ok; see 7.14
[SRS_BSW_00386] Configuration for detecting an error	Ok: CANNM_DEV_ERROR_DETECT
[SRS_BSW_00161] Microcontroller abstraction	n/a (CanNm microcontroller independent)
[SRS_BSW_00162] ECU layout abstraction	n/a (CanNm is ECU hardware independent)
[SRS_BSW_00005] No hard coded horizontal interfaces within MCAL	n/a (CanNm is not part of the MCAL)
[SRS_BSW_00415] User dependent include files	n/a (not flexible with respect to future extensions)
[SRS_BSW_00164] Implementation of interrupt service routines	n/a (no ISR provided)
[SRS_BSW_00325] Runtime of interrupt service routines	n/a (no ISR provided)
[SRS_BSW_00326] Transition from ISRs to OS tasks	n/a (no ISR provided)
[SRS_BSW_00342] Usage of source code and object code	Ok; see chapter 10.2.1
[SRS_BSW_00343] Specification and configuration of time	Ok; see chapter 10.2
[SRS_BSW_00160] Human-readable configuration data	n/a (implementation specific)
[SRS_BSW_00007] HIS MISRA C	Ok; all implementation related information
[SRS_BSW_00300] Module naming convention	Ok; CanNm prefix is used
[SRS_BSW_00413] Accessing instances of BSW modules	n/a (implementation specific)
[SRS_BSW_00347] Naming separation of different instances of	n/a

BSW drivers	(implementation specific)
[SRS_BSW_00305] Self-defined data types naming convention	n/a (no self-defined data types used)
[SRS_BSW_00307] Global variables naming convention	n/a (no global variables specified)
[SRS_BSW_00310] API naming convention	Ok; see chapter 7.19
[SRS_BSW_00373] Main processing function naming convention	Ok; see chapter 8.6.1
[SRS_BSW_00327] Error values naming convention	Ok; see chapter 7.14
[SRS_BSW_00335] Status values naming convention	n/a (no status values exported)
[SRS_BSW_00350] Development error detection keyword	Ok; see chapter 8.8
[SRS_BSW_00408] Configuration parameter naming convention	Ok; see chapter 10.2
[SRS_BSW_00410] Compiler switches shall have defined values	n/a (CanNm is compiler independent)
[SRS_BSW_00411] Get version info keyword	Ok; see chapter 8.3.15
[SRS_BSW_00346] Basic set of module files	Ok; see SWS_CanNm_00299, SWS_CanNm_00300, SWS_CanNm_00301, SWS_CanNm_00302, SWS_CanNm_00303, SWS_CanNm_00304
[SRS_BSW_00158] Separation of configuration from implementation	Ok; see SWS_CanNm_00299, SWS_CanNm_00300, SWS_CanNm_00301, SWS_CanNm_00302, SWS_CanNm_00303, SWS_CanNm_00304
[SRS_BSW_00314] Separation of interrupt frames and service routines	n/a (CanNm doesn't have interrupt frame definitions)
[SRS_BSW_00370] Separation of callback interface from API	Ok; see SWS_CanNm_00303CANNM044
[SRS_BSW_00348] Standard type header	Ok; see SWS_CanNm_00305
[SRS_BSW_00353] Platform specific type header	Ok; see SWS_CanNm_00305
[SRS_BSW_00361] Compiler specific language extension header	Ok; see SWS_CanNm_00305
[SRS_BSW_00301] Limit imported information	Ok; see SWS_CanNm_00305, SWS_CanNm_00306, SWS_CanNm_00307, SWS_CanNm_00308, SWS_CanNm_00309, SWS_CanNm_00310 SWS_CanNm_00311, SWS_CanNm_00312, SWS_CanNm_00313
[SRS_BSW_00302] Limit exported information	Ok; see SWS_CanNm_00302 and chapter 7.19
[SRS_BSW_00328] Avoid duplication of code	n/a (implementation specific)
[SRS_BSW_00312] Shared code shall be reentrant	n/a (implementation specific)
[SRS_BSW_00006] Platform independency	n/a (CanNm is hardware independent)
[SRS_BSW_00357] Standard API return type	Ok; see chapter 7.19
[SRS_BSW_00377] Module specific API return types	n/a (CanNm doesn't define own types)
[SRS_BSW_00304] AUTOSAR integer data types	Ok; see chapter 7.19

[SRS_BSW_00355] Do not redefine AUTOSAR integer data types	Ok; see chapter 7.19
[SRS_BSW_00378] AUTOSAR boolean type	Ok; see chapter 7.19 and 10.2
[SRS_BSW_00306] Avoid direct use of compiler and platform specific keywords	n/a (implementation specific)
[SRS_BSW_00308] Definition of global data	Ok; see SWS_CanNm_00299, SWS_CanNm_00300, SWS_CanNm_00301
[SRS_BSW_00309] Global data with read-only constraint	n/a (implementation specific)
[SRS_BSW_00371] Do not pass function pointers via API	Ok; see chapter 7.19
[SRS_BSW_00358] Return type of init() functions	Ok; see chapter 8.3.1
[SRS_BSW_00414] Parameter of init function	Ok; see chapter 8.3.1
[SRS_BSW_00376] Return type and parameters of main processing functions	Ok; see chapter 8.6.1
[SRS_BSW_00359] Return type of callback functions	Ok; see chapter 8.4
[SRS_BSW_00360] Parameters of callback functions	Ok; see chapter 8.4
[SRS_BSW_00329] Avoidance of generic interfaces	Ok; see chapter 7.19
[SRS_BSW_00330] Usage of macros / inline functions instead of functions	n/a (implementation specific)
[SRS_BSW_00331] Separation of error and status values	n/a (CanNm doesn't provide status information)
[SRS_BSW_00009] Module User Documentation	Ok; see whole document
[SRS_BSW_00401] Documentation of multiple instances of configuration parameters	Ok; see chapter 10.2
[SRS_BSW_00172] Compatibility and documentation of scheduling strategy	n/a (implementation specific)
[SRS_BSW_00010] Memory resource documentation	n/a (implementation specific)
[SRS_BSW_00333] Documentation of callback function context	n/a (implementation specific)
[SRS_BSW_00374] Module vendor identification	Ok; see chapter 10.3.5
[SRS_BSW_00379] Module identification	Ok; see chapter 10.3.5
[SRS_BSW_00003] Version identification	Ok; see chapter 10.3.5
[SRS_BSW_00318] Format of module version numbers	Ok; see chapter 10.3.5
[SRS_BSW_00321] Enumeration of module version numbers	n/a (implementation specific)
[SRS_BSW_00341] Microcontroller compatibility documentation	n/a (CanNm is microcontroller independent)
[SRS_BSW_00334] Provision of XML file	n/a (implementation specific)
[SRS_Nm_02516] Bus Synchronization on demand	Ok; see SWS_CanNm_00226, SWS_CanNm_00130, SWS_CanNm_00280

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

Requirement	***Satisfied by***
[SRS_Nm_00150] Configuration of functionality	Ok; see chapter 10.2
[SRS_Nm_00151] Integration into running NM cluster	n/a (The CAN bus satisfies already this requirement)
[SRS_Nm_00043] Bus Traffic without NM Initialization	n/a (ComM is responsible to initialize the communication components)
[SRS_Nm_00044] Applicability to different types of	Ok; see chapter 4.2

communication systems	
[SRS_Nm_00045] NM-cluster Independent Shutdown Coordination	Ok; see chapter 7.19
[SRS_Nm_00046] Trigger of startup of all Nodes at any Point in Time	n/a (not in the responsibility of CanNm)
[SRS_Nm_00047] Bus Keep Awake Services	Ok; see chapter 8.3.6
[SRS_Nm_00048] Bus Sleep Mode	n/a (not in the responsibility of CanNm)
[SRS_Nm_00050] NM State Information	n/a (application can determine the Nm states using ComM API)
[SRS_Nm_00051] NM State Change Indication	Ok; see chapter 8.7.1
[SRS_Nm_00052] Notification that all other ECUs are ready to sleep	n/a (not in the responsibility of CanNm)
[SRS_Nm_02509] Notification that at least one other node is not ready to sleep anymore	n/a (not in the responsibility of CanNm)
[SRS_Nm_02503] Sending user data	Ok; see chapter 8.3.7
[SRS_Nm_02504] Receiving user data	Ok; see chapter 8.3.8
[SRS_Nm_00153] Detection of present nodes	n/a (not in the responsibility of CanNm)
[SRS_Nm_02508] Unambiguous node identification per bus	Ok; see chapter 8.3.11
[SRS_Nm_02505] Sending node identifier	Ok; see chapter 7.12
[SRS_Nm_02506] Receiving node identifier	Ok; see chapter 0
[SRS_Nm_02511] Configurable Role in Cluster Shutdown	Ok; see 7.8.3
[SRS_Nm_00053] Deterministic Behavior in Case of Bus Unavailability	Ok; see chapter 7.11
[SRS_Nm_00137] Communication system error handling	Ok; see chapter 7.11
[BSW136] Coordination of coupled networks	n/a (not in the scope of CanNm)
[BSW140] Compliance with OSEK NM on a gateway	n/a (not in the scope of CanNm)
[SRS_Nm_00054] Deterministic Time for Bus Sleep	n/a (not in the scope of CanNm)
[SRS_Nm_00142] Limitation of NM bus load	Ok; see chapter 7.7
[SRS_Nm_00143] Predictable NM bus load	Ok; see chapter 7.6.1
[SRS_Nm_00144] ECU cluster size	n/a (CanNm hasn't any restriction concerning cluster size)
[SRS_Nm_00145] Robustness against NM message losses	CanNm is robust against loss of NM messages for a time specified by CANNM_TIMEOUT_TIME.
[SRS_Nm_00146] Robustness against NM message jitter	CanNm is robust against jitter of NM messages for up to a time specified by CANNM_TIMEOUT_TIME.
[SRS_Nm_00147] Processor independent algorithm	n/a (not in the responsibility of CanNm)
[SRS_Nm_00149] Configurable Timing	Ok; see chapter 10.2
[SRS_Nm_00154] Bus independency of API	n/a (CanNm has to be bus dependent)
[SRS_Nm_00148] Separation of Communication system dependent parts	Ok; see whole document
[BSW139] Compliance with OSEK NM on one cluster	n/a (not in the scope of CanNm)
[SRS_Nm_02510] Immediate Transmission Confirmation	Ok; see chapter 8.4.1
[SRS_Nm_02512] CommunicationControl (28 hex) service support	Ok; SWS_CanNm_00215 and SWS_CanNm_00216

7 Functional specification

7.1 Coordination algorithm

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

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

If any node in the network management cluster requires bus-communication, it can wake-up the network management cluster from the Bus-Sleep Mode by transmitting Network Management PDUs. For more details concerning wakeup procedure itself please refer to [10].

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

[SWS_CanNm_00087] ⌈Every network node in a CanNm cluster shall transmit periodic Network Management PDUs as long as it requires bus-communication; otherwise it shall transmit no Network Management PDUs.⌋()

[SWS_CanNm_00088] ⌈If bus communication in a CanNm cluster is released and there are no Network Management PDUs on the bus for a configurable amount of time determined by `CANNM_TIMEOUT_TIME` + `CANNM_WAIT_BUS_SLEEP_TIME` (both configuration parameters) transition into the Bus-Sleep Mode shall be performed.⌋()

The overall state machine of the AUTOSAR CanNm algorithm can be defined as follows:

[SWS_CanNm_00089] ⌈The AUTOSAR CanNm state machine shall contain states, transitions and triggers required for the AUTOSAR CanNm algorithm seen from point of view of one single node in the network management cluster.⌋()

Note: State transitions have to be performed latest within the next main function.

Note: An UML state chart of the AUTOSAR CanNm state machine from point of view of one single node in the network management cluster can be found in detail in the API specification chapter 8).

7.2 Operational Modes

In the following chapter operational modes of the AUTOSAR CanNm algorithm are described in detail.

[SWS_CanNm_00092] ⌈The AUTOSAR CanNm shall contain three operational modes visible at the module's interface:

- Network Mode
- Prepare Bus-Sleep Mode
- Bus-Sleep Mode⌋()

[SWS_CanNm_00093] ⌈Changes of the AUTOSAR CanNm operational modes shall be notified to the upper layer by means of callback functions.⌋()

7.2.1 Network Mode

[SWS_CanNm_00094] ⌈The Network Mode shall consist of three internal states:

- Repeat Message State
- Normal Operation State
- Ready Sleep State⌋()

[SWS_CanNm_00314] ⌈When the Network Mode is entered from Bus-Sleep, by default, the CanNm module shall enter the Repeat Message State.⌋()

[SWS_CanNm_00315] ⌈When the Network Mode is entered from Prepare Bus-Sleep Mode, by default, the CanNm module shall enter the Repeat Message State.⌋()

[SWS_CanNm_00096] ⌈When the Network Mode is entered, the CanNm module shall start the NM-Timeout Timer.⌋()

[SWS_CanNm_00097] ⌈When the Network Mode is entered, CanNm shall notify the upper layer of the new current operational mode by calling the callback function `Nm_NetworkMode`.⌋()

[SWS_CanNm_00098] ⌈At successful reception of a Network Management PDU (call of `CanNm_RxIndication`) in the Network Mode, the CanNm module shall restart the NM-Timeout Timer.⌋()

[SWS_CanNm_00099] ⌈At successful transmission of a Network Management PDU (call of `CanNm_TxConfirmation`) in the Network Mode, the CanNm module shall restart the NM-Timeout Timer.⌋()

Note: If `CANNM_IMMEDIATE_TXCONF_ENABLED` is enabled it is assumed that each Network Management PDU transmission request results in a successful Network Management PDU transmission.

[SWS_CanNm_00206] ⌈The CAN NM module shall reset the NM-Timeout Timer every time it is started or restarted.⌋()

7.2.1.1 Repeat Message State

For nodes that are not in passive mode (refer to chapter 7.8.3) the Repeat Message State ensures, that any transition from Bus-Sleep or Prepare Bus-Sleep to the Network Mode becomes visible to the other nodes on the network. Additionally it ensures that any node stays active for a minimum amount of time. It can be used for detection of present nodes.

[SWS_CanNm_00100] ⌈When the Repeat Message State is entered from Bus-Sleep Mode, Prepare-Bus-Sleep Mode, Normal Operation State or Ready Sleep State, the CanNm module shall (re-)start transmission of Network Management PDUs unless passive mode is enabled and/or communication is disabled.⌋()

[SWS_CanNm_00101] ⌈When the NM-Timeout Timer expires in the Repeat Message State, the CanNm module shall start the NM-Timeout Timer.⌋()

[SWS_CanNm_00102] ⌈The network management state machine shall stay in the Repeat Message State for a configurable amount of time determined by the `CANNM_REPEAT_MESSAGE_TIME` (configuration parameter); after that time the CanNm module shall leave the Repeat Message State.⌋()

[SWS_CanNm_00103] ⌈When Repeat Message State is left and if the network has been requested (see SWS_CanNm_00104), the CanNm module shall enter the Normal Operation State.⌋()

[SWS_CanNm_00106] ⌈When Repeat Message State is left and if the network has been released (see SWS_CanNm_00105), the CanNm module shall enter the Ready Sleep State.⌋()

[SWS_CanNm_00107] ⌈When Repeat Message State is left and if the option CANNM_NODE_DETECTION_ENABLED is enabled, the CanNm module shall clear the Repeat Message Bit.⌋()

[SWS_CanNm_00137] ⌈If the service CanNm_RepeatMessageRequest is called in Repeat Message State, Prepare Bus-Sleep Mode or Bus-Sleep Mode, the CanNm module shall not execute the service and return E_NOT_OK.⌋()

7.2.1.2 Normal Operation State

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

[SWS_CanNm_00116] ⌈When the Normal Operation State is entered from Ready Sleep State, the CanNm module shall start transmission of Network Management PDUs.⌋()

Note: If passive mode is enabled or the Network Management PDU transmission ability has been disabled no NM PDUs are transmitted, therefore no action is required.

[SWS_CanNm_00117] ⌈When the NM-Timeout Timer expires in the Normal Operation State, the CanNm module shall start the NM-Timeout Timer.⌋()

[SWS_CanNm_00118] ⌈When the network is released and the current state is Normal Operation State, the CanNm module shall enter the Ready Sleep state (refer to SWS_CanNm_00105).⌋()

[SWS_CanNm_00119] ⌈At Repeat Message Request Bit Indication in the Normal Operation State, the CanNm module shall enter the Repeat Message State.⌋()

[SWS_CanNm_00120] ⌈At Repeat Message Request (CanNm_RepeatMessageRequest) in the Normal Operation State, the CanNm module shall enter the Repeat Message State.⌋()

[SWS_CanNm_00121] ⌈At Repeat Message Request (`CanNm_RepeatMessageRequest`) in the Normal Operation State the CanNm module shall set the Repeat Message Bit.⌋()

7.2.1.3 Ready Sleep State

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

[SWS_CanNm_00108] ⌈When the Ready Sleep State is entered from Repeat Message State or Normal Operation State, the CanNm module shall stop transmission of Network Management PDUs.⌋()

Note: If passive mode is enabled no NM PDUs are transmitted, therefore no action is required.

[SWS_CanNm_00109] ⌈When the NM-Timeout Timer expires in the Ready Sleep State, the CanNm module shall enter the Prepare Bus-Sleep Mode.⌋()

[SWS_CanNm_00110] ⌈When the network is requested and the current state is the Ready Sleep State, the CanNm module shall enter Normal Operation State (refer to SWS_CanNm_00104).⌋()

[SWS_CanNm_00111] ⌈At Repeat Message Request Bit Indication in the Ready Sleep State, the CanNm module shall enter the Repeat Message State.⌋()

[SWS_CanNm_00112] ⌈At Repeat Message Request (`CanNm_RepeatMessageRequest`) in the Ready Sleep State, the CanNm module shall enter the Repeat Message State.⌋()

[SWS_CanNm_00113] ⌈At Repeat Message Request (CanNm_RepeatMessageRequest) in Ready Sleep State the CanNm module shall set the Repeat Message Bit.⌋()

7.2.2 Prepare Bus-Sleep Mode

The purpose of the Prepare Bus-Sleep Mode is to ensure that all nodes have time to stop their network activity before the Bus-Sleep Mode is entered. In Prepare Bus-Sleep Mode the bus activity is calmed down (i.e. queued messages are transmitted in order to make all Tx-buffers empty) and finally there is no activity on the bus in the Prepare Bus-Sleep Mode.

[SWS_CanNm_00114] ⌈When Prepare Bus-Sleep Mode is entered, the CanNm module shall notify the upper layer by calling `Nm_PrepareBusSleepMode`.⌋()

[SWS_CanNm_00115] ⌈The CanNm module shall stay in the Prepare Bus-Sleep Mode for a configurable amount of time determined by the `CANNM_WAIT_BUS_SLEEP_TIME` (configuration parameter); after that time the Prepare Bus-Sleep Mode shall be left and the Bus-Sleep Mode shall be entered.⌋()

[SWS_CanNm_00124] ⌈At successful reception of a Network Management PDU in the Prepare Bus-Sleep Mode, the CanNm Module shall enter the Network Mode; by default the CanNm Module shall enter the Repeat Message State (refer to SWS_CanNm_00315).⌋()

[SWS_CanNm_00123] ⌈When the network is requested in the Prepare Bus-Sleep Mode, the CanNm module shall enter the Network Mode; by default the CanNm Module shall enter the Repeat Message State (refer to SWS_CanNm_00315).⌋()

[SWS_CanNm_00122] ⌈When the network has been requested in the Prepare Bus-Sleep Mode and the CanNm module has entered Network Mode and if `CANNM_IMMEDIATE_RESTART_ENABLED` (configuration parameter) is set to TRUE, the CanNm module shall transmit a Network Management PDU.⌋()

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

Note: If `CANNM_IMMEDIATE_RESTART_ENABLED` is set to TRUE and a wake-up line is used, a burst of Network Management PDUs occurs if all network nodes get a network request in Prepare Bus-Sleep Mode.

7.2.3 Bus-Sleep Mode

The purpose of the Bus-Sleep Mode is to reduce power consumption in the node when no messages are to be exchanged. The communication controller is switched into the sleep mode, respective wakeup mechanisms are activated and finally power consumption is reduced to the adequate level in the Bus-Sleep Mode.

If a configurable amount of time determined by the CANNM_TIMEOUT_TIME + CANNM_WAIT_BUS_SLEEP_TIME (both configuration parameters) is identically configured for all nodes in the network management cluster, all nodes in the network management cluster that are coordinated with use of the AUTOSAR NM algorithm perform the transition into the Bus-Sleep Mode at approximately the same time.

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

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

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

[SWS_CanNm_00126] ⌈When Bus-Sleep Mode is entered, except by default at initialization, the CanNm module shall notify the upper layer by calling the callback function Nm_BusSleepMode.⌋()

[SWS_CanNm_00127] ⌈When the CanNm module receives successfully Network Management PDU (call of CanNm_RxIndication) in the Bus-Sleep Mode, the CanNm module shall notify the upper layer by calling the callback function Nm_NetworkStartIndication.⌋()

[SWS_CanNm_00336] ⌈When the CanNm module successfully receives a Network Management PDU (call of CanNm_RxIndication) in the Bus-Sleep Mode, the CanNm module shall report the error CANNM_E_NET_START_IND if development error tracing is enabled (CANNM_DEV_ERROR_DETECT is set to TRUE).⌋()

Rationale: To avoid race conditions and state inconsistencies between Network and Mode Management, CanNm will not automatically perform the transition from Bus-Sleep Mode to Network Mode. CanNm will only inform the upper layers which have to make the wake-up decision. Network Management PDU reception in Bus-Sleep Mode must be handled depending on the current state of the ECU shutdown/startup process.

[SWS_CanNm_00128] ⌈If CanNm_PassiveStartUp is called in the Bus-Sleep Mode or Prepare Bus-Sleep Mode, the CanNm module shall enter the Network Mode; by default the CanNm module shall enter the Repeat Message State (refer to SWS_CanNm_00314 and SWS_CANNM_00315).⌋()

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

[SWS_CanNm_00129] ⌈When the network is requested in Bus-Sleep Mode, the CanNm module shall enter the Network Mode; by default the CanNm module shall enter the Repeat Message State (refer to SWS_CanNm_00314 and SWS_CanNm_00104).⌋()

7.3 Network states

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

[SWS_CanNm_00104] ⌈The function call `CanNm_NetworkRequest` shall request the network. I.e. the CanNm module shall change network state to 'requested'.⌋()

[SWS_CanNm_00105] ⌈The function call `CanNm_NetworkRelease` shall release the network. I.e. the CanNm module shall change network state to 'released'.⌋()

7.4 Initialization

[SWS_CanNm_00141] ⌈If the initialization of the CanNm module (`CanNm_Init`) is successful, the CanNm module shall set the Network Management State to `NM_STATE_BUS_SLEEP`.⌋()

Note: The CanNm module should be initialized after CanIf is initialized and before any other network management service is called.

[SWS_CanNm_00143] ⌈When initialized, by default, the CanNm module shall set the network state to 'released'⌋()

[SWS_CanNm_00144] ⌈When initialized, by default, the CanNm module shall enter the Bus-Sleep Mode.⌋()

[SWS_CanNm_00145] ⌈If AUTOSAR CanNm is not initialized the CanNm module shall not prohibit bus traffic.⌋()

[SWS_CanNm_00147] ⌈If `CanNm_PassiveStartUp` is called in the Network Mode, the CanNm module shall not execute this service and shall return `E_NOT_OK`.⌋()

[SWS_CanNm_00060] ⌈The function CanNm_Init shall select the active configuration set by means of a configuration pointer parameter being passed (see 8.3.1).⌋()

[SWS_CanNm_00061] ⌈After initialization the CanNm module shall stop the NM Message Tx Timeout Timer.⌋()

Note: The NM Message Tx Timeout Timer is not needed in case of `CANNM_IMMEDIATE_TXCONF_ENABLED` is set to TRUE or if `CANNM_PASSIVE_MODE_ENABLED` is set to TRUE.

[SWS_CanNm_00023] ⌈During initialization the CanNm module shall deactivate the bus load reduction.⌋()

[SWS_CanNm_00033] ⌈ After initialization the CanNm module shall stop the transmission of Network Management PDUs by stopping the Message Cycle Timer.⌋()

Note: If `CANNM_PASSIVE_MODE_ENABLED` is set to TRUE the CanNm Message Cycle is not needed, because no Network Management PDUs are transmitted by such nodes.

[SWS_CanNm_00039] ⌈If CanNm is not initialized a call of any CanNm function (except `CanNm_Init`) shall be rejected and `E_NOT_OK` shall be returned if the API has a return value. If development error detection is enabled it shall report `CANNM_E_NO_INIT` to the Development Error Tracer.⌋()

[SWS_CanNm_00025] ⌈During initialization the CanNm module shall set each byte of the user data to `0xFF`.⌋()

[SWS_CanNm_00085] ⌈During initialization the CanNm module shall set the Control Bit Vector to `0x00`.⌋()

7.5 Execution

7.5.1 Processor architecture

[SWS_CanNm_00146] ⌈The AUTOSAR CanNm algorithm shall be processor independent, which means; it shall not rely on any processor specific hardware support and thus shall be realizable on any processor architecture that is in the scope of AUTOSAR.⌋()

7.5.2 Timing parameters

[SWS_CanNm_00246] ⌈The configuration parameter `CANNM_TIMEOUT_TIME` shall determine the AUTOSAR CanNm timing parameter NM-Timeout Time.⌋()

[SWS_CanNm_00247] ⌈The configuration parameter `CANNM_REPEAT_MESSAGE_TIME` shall determine the AUTOSAR CanNm timing parameter Repeat Message Time.⌋()

[SWS_CanNm_00248] ⌈The configuration parameter `CANNM_WAIT_BUS_SLEEP_TIME` shall determine the AUTOSAR CanNm timing parameter Wait Bus-Sleep Time.⌋()

[SWS_CanNm_00249] ⌈The configuration parameter `CANNM_REMOTE_SLEEP_IND_TIME` shall determine the AUTOSAR CanNm timing parameter Remote Sleep Indication Time.⌋()

7.6 Communication Scheduling

7.6.1 Transmission

Note: The transmission mechanisms described in this chapter are only relevant if the Network Management PDU transmission ability is enabled.

[SWS_CanNm_00072] ⌈The Network Management PDUs transmission capability shall be configurable by means of `CANNM_PASSIVE_MODE_ENABLED` (see chapter 10.2).⌋()

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

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

[SWS_CanNm_00237] ⌈The CanNm module shall provide the periodic transmission mode. In this transmission mode the CanNm module shall send Network Management PDUs periodically.⌋()

[SWS_CanNm_00238] ⌈The CanNm module shall optionally provide the periodic transmission mode with bus load reduction. In this transmission mode the CanNm module shall transmit Network Management PDUs due to a specific algorithm.⌋()

The periodic transmission mode with bus load reduction ensures a reduced bus load.

Note: The periodic transmission mode is used in the "Repeat Message State" and "Normal Operation State" if the bus load reduction mechanism is disabled.
The periodic transmission mode with bus load reduction is only used, in the "Normal Operation State" if the bus load reduction mechanism is enabled.

[SWS_CanNm_00071] ⌈The immediate transmission confirmation mechanism shall be configurable by means of the `CANNM_IMMEDIATE_TXCONF_ENABLED` (see 10.2).⌋()

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

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

[SWS_CanNm_00005] ⌈If the Repeat Message State is not entered via `CanNm_NetworkRequest` OR `CanNmImmediateNmTransmissions` is zero the transmission of NM PDU shall be delayed by `CANNM_MSG_CYCLE_OFFSET` after entering the repeat message state.⌋()

[SWS_CanNm_00334] ⌈When entering the Repeat Message State from Bus Sleep Mode or Prepare Bus Sleep Mode because of `CanNm_NetworkRequest()` (active wakeup) and if `CanNmImmediateNmTransmissions` is greater zero, the NM PDUs shall be transmitted using `CanNmImmediateNmCycleTime` as cycle time. The transmission of the first NM PDU shall be triggered as soon as possible. After the transmission the Message Cycle Timer shall be reloaded with `CanNmImmediateNmCycleTime`. The `CanNmMsgCycleOffset` shall not be applied in this case.⌋()

[SWS_CanNm_00006]⌈If Normal Operation State is entered from Ready Sleep State the transmission of NM PDUs shall be started immediately.⌋()

[SWS_CanNm_00454]⌈If CanNmPnHandleMultipleNetworkRequests is set to TRUE CanNm_NetworkRequest shall trigger a state transition from Network Mode to Repeat Message state and the NM PDUs shall be transmitted using CanNmImmediateNmCycleTime as cycle time. The transmission of the first NM PDU shall be triggered as soon as possible. After the transmission the Message Cycle Timer shall be reloaded with CanNmImmediateNmCycleTime. The CanNmMsgCycleOffset shall not be applied in this case.⌋()

Note: CanNmImmediateNmTransmissions has to be greater zero in this case due to SWS_CanNm_00445

[SWS_CanNm_00335] ⌈The number of NM PDUs transmitted with the cycle time CanNmImmediateNmCycleTime is defined by CanNmImmediateNmTransmissions (counter). After all immediate NM PDUs have been transmitted the CanNm shall continue the transmission using the CanNmMsgCycleTime.⌋()

Note: While transmitting NM PDUs using the CanNmImmediateNmCycleTime no other Nm PDUs shall be transmitted (i.e. the CanNmMsgCycleTime transmission cycle is stopped).

[SWS_CanNm_00032] ⌈ If transmission of Network Management PDUs has been started and the CanNm Message Cycle Timer expires the CanNm module shall transmit a Network Management PDU by calling the CanIf function CanIf_Transmit.⌋()

Note: If the function call of CanIf_Transmit fails the Transmission Error handling described in chapter 7.11 informs the CanNm module.

[SWS_CanNm_00040] ⌈If the CanNm Message Cycle Timer expires the CanNm module shall restart with CANNM_MSG_CYCLE_TIME.⌋()

Note: If a Network Management PDU has been successfully transmitted the function CanNm_TxConfirmation shall be called by CanIf if CANNM_IMMEDIATE_TXCONF_ENABLED (configuration parameter) is disabled.

[SWS_CanNm_00051] ⌈If transmission of Network Management PDUs has been stopped the CanNm module shall cancel the Message Cycle Timer.⌋()

7.6.2 Reception

If a NM PDU has been successfully received, the CanIf module will call the callback function CanNm_RxIndication.

[SWS_CanNm_00035] ⌈On the call of the callback function CanNm_RxIndication, the CanNm module shall copy the data of the Network Management PDU referenced in the function parameter to an internal buffer.⌋()

7.7 Bus Load Reduction Mechanism

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

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

1. If a Network Management PDU is received the CanNm Message Cycle Timer is reloaded with the node specific timing parameter CANNM_MSG_REDUCED_TIME. The node specific time CANNM_MSG_REDUCED_TIME should be greater than ½ CANNM_MSG_CYCLE_TIME and less than CANNM_MSG_CYCLE_TIME**.**

2. If a Network Management PDU is been transmitted the CanNm Message Cycle Timer is reloaded with the network management cluster specific timing parameter CANNM_MSG_CYCLE_TIME.

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

The algorithm ensures that the bus load is limited to a maximum two Network Management PDUs per CANNM_MSG_CYCLE_TIME.

An example can be found in chapter 11.

[SWS_CanNm_00052] ⌈The bus load reduction mechanism shall be statically configurable by means of the CANNM_BUS_LOAD_REDUCTION_ENABLED parameter (see 10.2).⌋()

[SWS_CanNm_00156] ⌈When the Repeat Message State is entered from Bus-Sleep Mode, Prepare Bus-Sleep Mode, Normal Operation or Ready Sleep State the CanNm module shall deactivate the busload reduction.⌋()

[SWS_CanNm_00157] ⌈When the Normal Operation State is entered from Repeat Message State or Ready Sleep State and CANNM_BUS_LOAD_REDUCTION_ENABLED is TRUE the CanNm module shall activate the busload reduction.⌋()

[SWS_CanNm_00069] ⌈If the bus load reduction mechanism is globally enabled (CANNM_BUS_LOAD_REDUCTION_ENABLED is TRUE), for a particular channel activated and the function CanNm_RxIndication is called for this channel, the CanNm module shall restart the CanNm Message Cycle Timer with the node specific time CANNM_MSG_REDUCED_TIME.⌋()

7.8 Additional features

7.8.1 Detection of Remote Sleep Indication

The “Remote Sleep Indication” denotes a situation, where a node in Normal Operations States finds all other nodes in the cluster are ready to sleep (in Ready-Seep State). The node in Normal Operation State will still keep the bus awake.

[SWS_CanNm_00149] ⌈Detection of remote sleep indication shall be statically configurable with use of the CANNM_REMOTE_SLEEP_IND_ENABLED switch (configuration parameter).⌋()

[SWS_CanNm_00150] ⌈If the CanNm module receives no Network Management PDUs in the Normal Operation State for a configurable amount of time determined by CANNM_REMOTE_SLEEP_IND_TIME (configuration parameter), the CanNm module shall call the callback function Nm_RemoteSleepIndication.⌋()

With a call of Nm_RemoteSleepIndication CanNm notifies the module Nm that all nodes in the cluster are ready to sleep (the so-called ‘Remote Sleep Indication’).

[SWS_CanNm_00151] ⌈If Remote Sleep Indication has been previously detected and if a Network Management PDU is received in the Normal Operation State or Ready Sleep State again, the module CanNm shall call the callback function Nm_RemoteSleepCancellation.⌋()

[SWS_CanNm_00152] ⌈If Remote Sleep Indication has been previously detected and if Repeat Message State is entered from Normal Operation State or Ready

Sleep State, the module CanNm shall call the callback function `Nm_RemoteSleepCancellation`.⌋()

With a call of `Nm_RemoteSleepCancellation` CanNm notifies the module Nm that some nodes in the cluster are not ready to sleep anymore (the so-called 'Remote Sleep Cancellation').

[SWS_CanNm_00154] ⌈When the service `CanNm_CheckRemoteSleepIndication` is called and the state is Bus-Sleep Mode, Prepare Bus-Sleep Mode or Repeat Message State the CanNm module shall not execute the service and shall return `E_NOT_OK`.⌋()

7.8.2 User Data

[SWS_CanNm_00158] ⌈Support of NM user data shall be statically configurable with use of the `CANNM_USER_DATA_ENABLED` switch (configuration parameter).⌋()

[SWS_CanNm_00159] ⌈When `CanNm_SetUserData` is called the CanNm module shall set the Network Management user data for the Network Management PDUs transmitted next on the bus.⌋()

[SWS_CanNm_00160] ⌈When `CanNm_GetUserData` is called CanNm module shall return the Network Management user data of the most recently received Network Management PDU.⌋()

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

7.8.2.1 COM User Data

Alternatively to the usage of the CanNm APIs to set and get user data, CanNm may use the COM to retrieve its user data.

[SWS_CanNm_00327] ⌈If `CanNmComUserDataSupport` is enabled the API `CanNm_SetUserData` shall not be available.⌋()

[SWS_CanNm_00328] ⌈If `CanNmComUserDataSupport` is enabled the CanNm shall collect the NM User Data from the referenced NM I-PDU by calling `PduR_CanNmTriggerTransmit` and combine the user data with the further NM bytes each time before it requests the transmission of the corresponding NM PDU.⌋()

[SWS_CanNm_00450] ⌈If `CanNmComUserDataSupport` is enabled and `PduR_CanNmTriggerTransmit` returns E_NOT_OK, the NM shall use the last transmitted value for NmUserData.⌋()

Note: The transmission of outdated NM data can be avoided by not stopping the IPdu in COM used for NmUserData transmission.

[SWS_CanNm_00329] ⌈If `CanNmComUserDataSupport` is enabled the CanNm shall call `PduR_CanNmTxConfirmation` within the message transmission confirmation function `CanNm_TxConfirmation` called by the CanIf.⌋()

[SWS_CanNm_00332] ⌈If `CanNmComUserDataSupport` is enabled and the number of available user data bytes does not match to the length of the referenced I-PDU an error shall be reported at generation time.⌋()

7.8.3 Passive Mode

In the Passive Mode the node is only receiving Network Management PDUs but not transmitting any Network Management PDUs.

[SWS_CanNm_00161] ⌈Passive Mode shall be statically configurable with use of the `CANNM_PASSIVE_MODE_ENABLED` switch (configuration parameter).⌋()

[SWS_CanNm_00162] ⌈Passive Mode shall be statically configured for all channels within one ECU.⌋()

[SWS_CanNm_00163] ⌈If Passive Mode is used (configuration parameter `CANNM_PASSIVE_MODE_ENABLED`) the following configurations options shall be disabled:

- Bus Synchronization
  (configuration parameter `CANNM_BUS_SYNCHRONIZATION_ENABLED`)
- Bus Load Reduction
  (configuration parameter `CANNM_BUS_LOAD_REDUCTION_ENABLED`)
- Remote Sleep Indication
  (configuration parameter `CANNM_REMOTE_SLEEP_IND_ENABLED`)
- Node Detection
  (configuration parameter `CANNM_NODE_DETECTION_ENABLED`)⌋()

7.8.4 Network Management PDU Rx Indication

[SWS_CanNm_00037] ⌈On the call of the callback function `CanNm_RxIndication`, the CanNm module shall call the Nm callback function `Nm_PduRxIndication`, if and only if `CANNM_PDU_RX_INDICATION_ENABLED` (configuration parameter) is set to TRUE.⌋()

7.8.5 State change notification

[SWS_CanNm_00166] ⌈All changes of the AUTOSAR CanNm states shall be notified to the upper layer by calling `Nm_StateChangeNotification` if the callback `Nm_StateChangeNotification` is enabled (configuration parameter `CANNM_STATE_CHANGE_IND_ENABLED` is `TRUE`).⌋()

7.8.6 Communication Control

[SWS_CanNm_00168] ⌈Communication Control shall be statically configurable with use of the `CANNM_COM_CONTROL_ENABLED` switch (configuration parameter).⌋()

[SWS_CanNm_00170] ⌈If the service `CanNm_DisableCommunication` is called the CanNm module shall disable the Network Management PDU transmission ability.⌋()

Note: This behavior shall also be applied in Repeat Message State. Communication Control feature does not influence the duration of the Repeat Message State.

Note: The NM coordination algorithm cannot work correctly if NM PDU transmission ability is disabled. Therefore it has to be ensured that the ECU is not shutdown as long as the NM PDU transmission ability is disabled.
If `CanNm_NetworkRelease` is called and NM PDU transmission ability has been disabled, ECU will shut down. This ensures that ECU can shut down also in case of race conditions (e.g. diagnostic session left shortly before enabling communication) or a wrong usage of communication control.

[SWS_CanNm_00173] ⌈When the Network Management PDU transmission ability is disabled, the CanNm module shall stop the CanNm Message Cycle Timer in order to stop the transmission of Network Management PDUs.⌋()

[SWS_CanNm_00174] ⌈When the Network Management PDU transmission ability is disabled, the CanNm module shall stop the NM-Timeout Timer.⌋()

[SWS_CanNm_00175] ⌈When the Network Management PDU transmission ability is disabled,
the CanNm module shall stop the Remote Sleep Indication Timer.⌋()

[SWS_CanNm_00178] ⌈When the Network Management PDU transmission ability is enabled, the transmission of NM PDUs shall be started latest within the next NM main function.⌋()

[SWS_CanNm_00179] ⌈When the Network Management PDU transmission ability is enabled, the CanNm module shall restart the NM-Timeout Timer.⌋()

[SWS_CanNm_00180] ⌈When the Network Management PDU transmission ability is enabled, the CanNm module shall re-start the Remote Sleep Indication Timer.⌋()

[SWS_CanNm_00181] ⌈The service `CanNm_RequestBusSynchronization` shall return `E_NOT_OK` if the Network Management PDU transmission ability is disabled.⌋()

7.8.7 Coordinator Synchronization Support

When having more than one coordinator connected to the same bus a special bit in the CBV, the *NmCoordinatorSleepReady* bit is used to indicate that the main coordinator requests to start shutdown sequence. The main functionality of the algorithm is described in the Nm module.

[SWS_CanNm_00341] ⌈If CanNm has entered Network Mode or called `Nm_CoordReadyToSleepCancellation` before it shall notify the Nm by calling `Nm_CoordReadyToSleepIndication` on the first reception of a NM PDU with the *NmCoordinatorSleepReady* bit (see CBV) set to 1. ⌋()

[SWS_CanNm_00348] ⌈If CanNm called `Nm_CoordReadyToSleepIndication` and is still in Network Mode it shall notify the Nm by calling `Nm_CoordReadyToSleepCancellation` on the first reception of a NM PDU with the NmCoordinatorSleepReady bit (see CBV) set to 0.⌋()

[SWS_CanNm_00342] ⌈The *NmCoordinatorSleepReady* bit in the CBV shall be set by the API `CanNm_SetSleepReadyBit`.⌋()

[SWS_CanNm_00343] ⌈This feature is optional and only available if `CANNM_COORDINATOR_SYNC_SUPPORT` is set to `TRUE`.⌋()

7.9 Car Wakeup

[SWS_CanNm_00405] ⌈The position of the Car Wakeup bit in the NM-PDU is defined by the configuration parameters CanNmCarWakeUpBytePosition and CanNmCarWakeUpBitPosition.⌋()

7.9.1 Rx Path

[SWS_CanNm_00406] ⌈If the car wakeup bit within any received NM-PDU is 1, CanNmCarWakeUpRxEnabled is TRUE, and CanNmCarWakeUpFilterEnabled is FALSE CanNm shall call Nm_CarWakeUpIndication and perform the standard Rx indication handling.⌋()

[SWS_CanNm_00407] ⌈If CanNm_GetPduData is called in the context of Nm_CarWakeUpIndication,
CanNm shall return the PDU data of the PDU that causes the call of Nm_CarWakeUpIndication.⌋()

Note: This is required to enable the ECU to identify detail about the sender of the car wakeup request.

[SWS_CanNm_00408] ⌈If CanNmCarWakeUpFilterEnabled is TRUE, the car wakeup bit within any received NM-PDU is 1, CanNmCarWakeUpRxEnabled is TRUE and the Node ID in the received NM-PDU is equal to CanNmCarWakeUpFilterNodeId the CanNm module shall call Nm_CarWakeUpIndication and perform the standard Rx Indication handling.⌋()

Note: The car wakeup filter is necessary to realize sub gateways that only consider the car wakeup of the central Gateway to avoid wrong wakeups.

7.9.2 Tx Path

The transmission of the car wakeup bit shall be handled by the application using the NM user data mechanism provided by the CanNm module.

7.10 Partial Networking

7.10.1 Rx Handling of NM PDUs

[SWS_CanNm_00409] ⌈If the CanNmPnEnabled is FALSE, the CanNm shall not drop NM PDUs from further Rx Indication handling and the partial networking extensions shall be disabled.⌋()

[SWS_CanNm_00410] ⌈If `CanNmPnEnabled` is `TRUE`, the PNI bit in the received NM-PDU is 0 and `CanNmAllNmMessagesKeepAwake` is `TRUE`, the CanNm module shall not drop NM PDUs from further Rx Indication handling omitting the extensions for partial networking.⌋()

Note: This is required to enable the Gateway to stay awake on any kind of NM-PDU.

[SWS_CanNm_00411] ⌈If `CanNmPnEnabled` is `TRUE`, the PNI bit in the received NM-PDU is 0 and `CanNmAllNmMessagesKeepAwake` is `FALSE`, the CanNm module shall ignore the received NM-PDU.⌋()

[SWS_CanNm_00412] ⌈If `CanNmPnEnabled` is `TRUE` and the PNI bit in the received NM-PDU is 1, CanNm module shall process the Partial Networking Information of the NM-PDU as described in chapter 7.10.3 NM PDU Filter Algorithm.⌋()

7.10.2 Tx Handling of NM PDUs

[SWS_CanNm_00413] ⌈If `CanNmPnEnabled` is `TRUE` the CanNm module shall set the value of the transmitted PNI bit to 1.⌋()

Note: The usage of the CBV is mandatory in case Partial Networking is used.

[SWS_CanNm_00414] ⌈If `CanNmPnEnabled` is `FALSE` the CanNm module shall set the value of the transmitted PNI bit always to 0.⌋()

7.10.3 NM PDU Filter Algorithm

The intention of the NM-PDU filter algorithm is to drop all received NM-PDUs that are not relevant for the ECU. If there is no NM-PDU on the network, that is relevant for the receiving ECU, the NM Timeout Timer is no longer restarted and the CanNm module changes to Prepare Bus Sleep Mode during active bus communication.
In order to distinguish between NM-PDUs that are relevant for the ECU and PDUs that are not relevant, the CanNm evaluates the NM User Data that contains the PN requests provided by requesting ECU. Every bit of the PN request information represents one PN.
It is statically configured if the ECU (CanNm) is part of one specific partial network or not. The NM-PDUs are ignored if the ECU is not part of the requested partial networks.

[SWS_CanNm_00403] ⌈During initialization CanNm shall disable the NM-PDU filter algorithm on all networks where `CanNmPnEnabled` is `TRUE`. When the NM-PDU filter algorithm is disabled all received NM PDUs shall not be dropped from further Rx Indication Handling.⌋()

[SWS_CanNm_00404] ⌈If the CanSm calls CanNm_ConfirmPnAvailability the NM-PDU filter algorithm shall be enabled on the indicated channel.⌋()

Rationale: This is required to allow a malfunctioning PN transceiver to shut down synchronously with the remaining network.

Note: If the NM-PDU filter algorithm is not enabled (e.g. due to malfunctioning PN transceiver) the CanNm restarts the NM-Timeout Timer when receiving a NM-PDU. Therefore normal shutdown behavior is performed.

[SWS_CanNm_00415] ⌈The NM-PDU filter algorithm shall evaluate the bytes of the received NM-PDU defined by `CanNmPnInfoOffset` (in bytes) starting from byte 0 and `CanNmPnInfoLength` (in bytes). This range is called PN Info Range.⌋()

[SWS_CanNm_00416] ⌈Every bit of the PN Info Range represents one Partial Network. If the bit is set to 1 the Partial Network is requested. If the bit is set to 0 there is no request for this PN.⌋()

[SWS_CanNm_00417] ⌈The filter algorithm shall compare (bitwise AND) the received PN information with the PN filter mask to detect if a relevant PN is requested or not. Each bit of the PN filter mask shall have the following meaning:

0 The PN request is irrelevant for the ECU. The communication stack of the ECU is not kept awake if this bit is set in a received NM-PDU.

1 The PN request is relevant for the ECU. The communication stack of the ECU is kept awake if this bit is set in a received NM-PDU.⌋()

[SWS_CanNm_00419] ⌈If at least one relevant PN is requested in the received NM-PDU the PDU shall not be dropped from further Rx Indication handling.⌋()

[SWS_CanNm_00420] ⌈If no relevant PN is requested in the received NM-PDU and `CanNmAllNmMessagesKeepAwake` is `FALSE` the PDU shall be dropped from further processing.⌋()

[SWS_CanNm_00421] ⌈If no relevant PN is requested in the received NM-PDU and `CanNmAllNmMessagesKeepAwake` is `TRUE` the PDU shall not be dropped from further Rx Indication handling.⌋()

Note: This is required to enable the Gateway to stay awake on any kind of NM-PDU.

Example:

- CanNmPnInfoOffset = 4
- CanNmPnInfoLength = 2

Only Byte 4 and Byte 5 of the NM PDU contain PN information:

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
CBV	NID	User Data		PN Info		User Data	
0x40	0x00	0xFF	0xFF	0x12	0x8E	0xFF	0xFF

Figure 7-1 Example NM PDU containing PN information

For this example two CanNmPnFilterMaskBytes would be defined, e.g

- CanNmPnFilterMaskByteIndex = 0 with CanNmPnFilterMaskByteValue = 0x01
- CanNmPnFilterMaskByteIndex = 1 with CanNmPnFilterMaskByteValue = 0x97

The filter algorithm actions and result would then be:

Filter Mask Value (Byte)	Compared to received PN info	Resulting in
0x01 (Byte 0)	0x12 (NM PDU Byte 4)	0x00 (no relevant PN information)
0x97 (Byte 1)	0x8E (NM PDU Byte 5)	0x86 (relevant PN information)

Figure 7-2 Example PN Filter Algorithm

As one byte contains relevant information this NM PDU would not be dropped from further Rx Indication handling.

7.10.4 Aggregation of Internal and External Requested Partial Networks

Note: This feature is used by every ECU that has to switch I-PDU-Groups because of the activity of partial networks. (e.g. to prevent false timeouts) I-PDU-Groups shall be switched on if the corresponding PN is requested internally or externally. I-PDU-Groups shall be switched off not until all internal and external requests for the corresponding PN are released.
The logic for switching the IPDU-Groups is implemented by ComM. The CanNm only provides the information if a PN is requested or not. The COM module is used to transfer the data to the upper layers.
To switch the I-PDU-Groups synchronously on all direct connected ECUs, CanNm shall provide the information of a request change to the upper layer at (almost) the same time on every ECU. This is why the reset timer is restarted on every received and every sent NM PDU (see below).
The aggregated state of the internal/external requested PNs is called External Internal Requests Aggregated (EIRA).

[SWS_CanNm_00422] ⌈If `CanNmPnEiraCalcEnabled` is `FALSE` the CanNm module shall skip the aggregation of external and internal PN requests information.⌋()

[SWS_CanNm_00423] ⌈If `CanNmPnEiraCalcEnabled` is `TRUE` the CanNm module shall calculate the aggregation of external and internal PN requests over all CanNm channels where `CanNmPnEnabled` is `TRUE`.⌋()

[SWS_CanNm_00424] ⌈The EIRA shall have the size of `CanNmNmPnInfoLength` and shall be initialized with value 0 (not requested) for every external and internal PN request.⌋()

[SWS_CanNm_00425] ⌈The CanNm shall only consider the PN request bits that are relevant for the ECU (defined by PN filter mask). All other PN request bits are ignored. Thus the EIRA only contains those PN requests, which are relevant for the ECU.⌋()

[SWS_CanNm_00426] ⌈If a NM-PDU is received the CanNm shall set every requested and filtered (relevant) PN request bit within this received NM-PDU in the EIRA to 1.⌋()

[SWS_CanNm_00427] ⌈If a NM-PDU is send by the CanNm, the CanNm module shall set every PN request bit in the EIRA to 1 that has been requested by the PN request bits in the transmitted NM-PDU.⌋()

[SWS_CanNm_00428] ⌈The CanNm module shall provide an EIRA reset timer for every PN request bit together for all physical CAN channels.⌋()

Note: This means, only one timer is required to handle one PN on multiple connected physical channels. For example: only 8 EIRA reset timers are required to handle the requests of a Gateway with 6 physical channels and 8 partial networks.
This is possible because the switch of PN PDU-Groups is done global for the ECU and not dependent of the physical channel.

[SWS_CanNm_00429] ⌈If an NM-PDU is received the CanNm module shall restart the EIRA reset timer for every PN request bit that has been requested in the received NM-PDU with `CanNmPnResetTime`.⌋()

[SWS_CanNm_00430] ⌈If a NM-PDU is send by the CanNm, the CanNm module shall restart the EIRA reset timer for every PN request bit that has been requested in the NM PDU with `CanNmPnResetTime`.⌋()

Note: `CanNmPnResetTime` shall be configured to a value greater than `CanNmMsgCycleTime`. If `CanNmPnResetTime` is configured to a value smaller than

CanNmMsgCycleTime and only one ECU requests the PN, the request state toggles in the EIRA because request state is rested before the requesting ECU is able to send the next NM PDU.

Note: CanNmPnResetTime shall be configured to a value smaller than CanNmTimeoutTime to avoid that the timer could elapse after NM already changed to Prepare Bus Sleep.

[SWS_CanNm_00431] ⌈If one of the EIRA reset timers expires, the CanNm module shall set PN request bit for the corresponding PN in the EIRA to 0.⌋()

[SWS_CanNm_00432] ⌈If content of EIRA changes (any bit changes from 1 to 0 or from 0 to 1) because of a received or transmitted NM-PDU or the EIRA reset timer expiration, the CanNm shall inform the upper layers by calling PduR_CanNmRxIndication(). By means of the Rx Indication function the EIRA data shall be provided to the COM module.⌋()

7.10.5 Aggregation of External Requested Partial Networks

Note: This feature is used by the Gateways to collect only the external PN requests.
The external PN requests are mirrored back to the requesting bus and provided to other (required) physical channels of a central gateway. In case of a sub gateway the requests bit must not be mirrored back to the requesting physical channel in order to avoid static waking between central- and sub gateways. This logic shall be implemented by the ComM. The CanNm module provides the information if the PN is externally requested or not. The COM module is used for data transmission to the upper layer.
The aggregated state of the external requested PNs is called "External Requests Aggregated" (ERA).

[SWS_CanNm_00433] ⌈If CanNmPnEraCalcEnabled is FALSE the CanNm module shall skip the aggregation of external PN requests.⌋()

[SWS_CanNm_00434] ⌈If CanNmPnEraCalcEnabled is TRUE the CanNm module shall calculate the aggregation of external PN requests for each CanNm channel where CanNmPnEnabled is set to TRUE separately.⌋()

[SWS_CanNm_00435] ⌈The ERA module shall have a size of CanNmNmPnInfoLength and shall be initialized with value 0 (no PN requested).⌋()

[SWS_CanNm_00436] ⌈The CanNm shall only consider the PN request bits in the NM-PDU that are relevant for the ECU (defined by PN filter mask). All other PN request bits are ignored. Thus the ERA only contains those PN requests, which are relevant for the ECU.⌋()

[SWS_CanNm_00437] ⌈If a NM-PDU is received the CanNm module shall set every PN request bit in the ERA to 1 that has been requested by the PN request bits of the received NM-PDU.⌋()

[SWS_CanNm_00438] ⌈The CanNm module shall provide an ERA reset timer for every PN request bit for every physical CAN channel.⌋()

Note: This means, a separate timer is required to handle one PN on multiple physical channels.
For example: 48 ERA reset timers are required to handle the requests of a gateway with 6 physical channels and 8 partial networks. It is not possible to combine the reset timer like EIRA timers, because the external request mustn't be mirrored back to the requesting bus by a sub gateway. Thus it is required to detect the physical channel that is the source of the request bit.

[SWS_CanNm_00439] ⌈If a NM-PDU is received the CanNm module shall restart the ERA reset timer for every PN request bit that is requested in the NM-PDU with `CanNmPnResetTime`.⌋()

Note: `CanNmPnResetTime` shall be configured to a value greater than `CanNmMsgCycleTime`. If `CanNmPnResetTime` is configured to a value smaller than `CanNmMsgCycleTime` and only one ECU requests the PN, the request state toggles in the ERA because request state is rested before the requesting ECU is able to send the next NM-PDU.

Note: `CanNmPnResetTime` shall be configured to a value smaller than `CanNmTimeoutTime` to avoid that the timer could elapse after NM already changed to Prepare Bus Sleep.

[SWS_CanNm_00442] ⌈If one of the ERA reset timers expires, the CanNm module shall set the PN request bit of the corresponding PN in the ERA to 0.⌋()

[SWS_CanNm_00443] ⌈If content of ERA changes (any bit changes from 1 to 0 or from 0 to 1) because of a received NM-PDU or the ERA reset timer expiration the CanNm module shall inform the upper layers by calling `PduR_CanNmRxIndication()`. By means of the Rx Indication function the ERA data shall be provided to the COM module.⌋()

7.10.6 Spontaneous Transmission of NM PDUs via CanNm_NetworkRequest

[SWS_CanNm_00444] ⌈If CanNm_NetworkRequest is called, CanNmPnHandleMultipleNetworkRequest is set to TRUE and CanNm is in Ready Sleep State, Normal Operation State or Repeat Message State, CanNm shall change to or restart the Repeat Message State.⌋()

[SWS_CanNm_00445] ⌈If CanNmPnHandleMultipleNetworkRequests is set to TRUE the CanNm feature 'Immediate Transmission' is mandatory. It shall be ensured that CanNmImmediateNmTransmissions > 0 is given.⌋()

Note: The PN Control Module (e.G. ComM) is responsible to call CanNm_NetworkRequest if the PN request bits changes.

7.11 Transmission Error Handling

Depending on configuration the CanNm will evaluate the confirmation from the CanIf that a Network Management PDU has been successfully transmitted. Transmission Error Handling is a functionality that will monitor these confirmations and alarm the upper layers if a transmission confirmation is not received within a specific amount of time. The functionality works by restarting a timer after each request to transmit a Network Management PDU and if a confirmation has not been received before the timer expires, a callback of the Nm is invoked.

[SWS_CanNm_00073] ⌈If CANNM_PASSIVE_MODE_ENABLED is TRUE (see SWS_CanNm_00072) or CANNM_IMMEDIATE_TXCONF_ENABLED is TRUE the CanNm module shall deactivate the transmission error handling.⌋()

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

[SWS_CanNm_00064] ⌈The NM Message Tx Timeout Timer shall be started with CANNM_MSG_TIMEOUT_TIME when the transmission of a NM PDU is requested.⌋()

[SWS_CanNm_00065] ⌈The NM Message Tx Timeout Timer shall be stopped when CanNm_TxConfirmation is called by the CanIf.⌋()

[SWS_CanNm_00066] ⌈The function Nm_TxTimeoutException shall be called once when the NM Message Tx Timeout Timer expires.⌋()

[SWS_CanNm_00446] ⌈If `CanNmPnEnabled` is set to `TRUE` the function CanSM_TxTimeoutException shall be called once when the NM Message Tx Timeout Timer expires.⌋()

7.12 Network Management PDU Structure

The figure below shows the default format of the Network Management PDU:

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

Figure 7-3 Network Management PDU Default Format

[SWS_CanNm_00074] ⌈The location of the source node identifier shall be configurable by means of `CANNM_PDU_NID_POSITION` to Byte 0, Byte 1, or off (default: Byte 0).⌋()

Note: Setting the `CANNM_PDU_NID_POSITION` to off means that in the NM PDU no space is occupied by the source node identifier. Hence one more byte is available for user data.

[SWS_CanNm_00075] ⌈The location of the control Bit vector shall be configurable by means of `CANNM_PDU_CBV_POSITION` to Byte 0, Byte 1, or off (default: Byte 1).⌋()

Note: Setting the `CANNM_PDU_CBV_POSITION` to off means that in the NM PDU no space is occupied by the source node identifier. Hence one more byte is available for user data.

Note: The length of the Network Management PDU is defined by the PduLength parameter in the "global" ECUC module ([EcuC003_Conf], see Ecu Configuration specification). The difference between number of enabled system bytes and length is the amount of user data bytes.

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

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
CBV	Reserved	Partial Network Information Bit	Reserved	Active Wakeup Bit	NM Coordinator Sleep Ready Bit	Reserved R3.2 NM Coordinator ID (High Bit)	Reserved R3.2 NM Coordinator ID (Low Bit)	Repeat Message Request

Figure 7-4 Control Bit Vector

[SWS_CanNm_00045] ⌈The Control Bit Vector shall consist of

Bit 0: Repeat Message Request
  0: Repeat Message State not requested
  1: Repeat Message State requested
Bit 3: NM Coordinator Sleep Bit
  0: Start of synchronized shutdown is not requested by main coordinator
  1: Start of synchronized shutdown is requested by main coordinator
Bit 4 Active Wakeup Bit
  0: Node has not woken up the network (passive wakeup)
  1: Node has woken up the network (active Wakeup)
Bit 6 Partial Network Information Bit (PNI)
  0: NM PDU contains no Partial Network request information
  1: NM PDU contains Partial Network request information
Bit 1, 2, 5, 7 are reserved for future extensions
  0: Disabled / Reserved for future usage⌋()

Note: The Control Bit Vector is initialized with `0x00` during initialization (also refer to SWS_CanNm_00085).

[SWS_CanNm_00013] ⌈The CanNm module shall set the source node identifier with the configuration parameter `CANNM_NODE_ID` unless `CANNM_PDU_NID_POSITION` is set to off.⌋()

[SWS_CanNm_00135] ⌈Support of Repeat Message Request Bit and Repeat Message State Request shall be statically configurable with use of the `CANNM_NODE_DETECTION_ENABLED` switch (configuration parameter).⌋()

[SWS_CanNm_00138] ⌈The service call `CanNm_GetPduData` shall provide whole PDU data (Node ID, Control Bit Vector and User Data) of the most recently received Network Management PDU.⌋()

[SWS_CanNm_00401] ⌈If the CanNm performs a state change from Bus Sleep Mode or Prepare Bus Sleep Mode to Network Mode due to a call to `CanNm_NetworkRequest` (i.e. due to an active wakeup) and `CanNmActiveWakeupBitEnabled` is `TRUE`, the CanNm shall set the ActiveWakeupBit in the CBV.⌋()

[SWS_CanNm_00402] ⌈If the CanNm module leaves the Network Mode and `CanNmActiveWakeupBitEnabled` is `TRUE`, the CanNm module shall clear the ActiveWakeupBit in the CBV.⌋()

7.13 Functional requirements on CanNm API

[SWS_CanNm_00014] ⌈If the node detection functionality is enabled and if `CANNM_REPEAT_MSG_IND_ENABLED` is enabled, the CanNm module shall call the callback function `Nm_RepeatMessageIndication` upon each reception of the RepeatMessageRequest bit.⌋()

[SWS_CanNm_00086] ⌈If `CANNM_USER_DATA_ENABLED` is enabled but no user data bytes are available, the CanNm module shall raise an error during configuration or compilation time.⌋()

7.14 Error classification

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

Req. ID	*Type or error*	*Relevance*	*Related error code*	*Error Value*
CANNM316:	API service used without module initialization	Development	`CANNM_E_NO_INIT`	`0x01`
CANNM317:	API service called with wrong channel handle	Development	`CANNM_E_INVALID_CHANNEL`	`0x02`
CANNM318:	API service called with wrong PDU-ID	Development	`CANNM_E_INVALID_PDUID`	`0x03`
CANNM337:	Reception of NM PDUs in Bus-Sleep Mode.	Development	`CANNM_E_NET_START_IND`	`0x04`
CANNM319:	CanNm initialization has failed, e.g. selected configuration set doesn’t exist.	Development	`CANNM_E_INIT_FAILED`	`0x05`
CANNM321:	NM-Timeout Timer has abnormally expired outside of the Ready Sleep State; it may happen: (1) because of Bus-Off state, (2) if some ECU requests bus communication or node detection shortly before the NM-Timeout Timer expires so that a Network Management PDU can not be transmitted in time; this race condition applies to event-triggered systems	Development	`CANNM_E_NETWORK_TIMEOUT`	`0x11`
CANNM322:	Null pointer has been passed as an argument (Does not apply to function CanNm_Init)	Development	`CANNM_E_NULL_POINTER`	`0x12`

7.15 Error detection

For details refer to the chapter 7.3 “Error Detection” in *SWS_BSWGeneral.*

7.16 Error notification

[SWS_CanNm_00189] ⌈The CanNm module shall not return development errors by API functions; in case of a development error, the execution of the respective API function shall be aborted and E_NOT_OK shall be returned, if applicable.⌋()

[SWS_CanNm_00190] ⌈The CanNm module shall not return production errors by API functions; in case of a production error, the execution of the respective API function shall be aborted and E_NOT_OK shall be returned, if applicable.⌋()

Note: Currently no production errors are specified for the CAN NM.

[SWS_CanNm_00191] ⌈Each CanNm function that is not executed due to missing initialization of CanNm shall return E_NOT_OK to the calling function if development error detection is enabled (CANNM_DEV_ERROR_DETECT is set to TRUE).⌋()

[SWS_CanNm_00192] ⌈When a CanNm service with an invalid network handle is called, the called function shall not be executed and it shall return E_NOT_OK to the calling function if development error detection is enabled (CANNM_DEV_ERROR_DETECT is set to TRUE).⌋()

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

[SWS_CanNm_00292] ⌈When the NULL Pointer is passed as an argument to a CanNm service, the called function shall not be executed, but instead of that it shall report CANNM_E_NULL_POINTER to the Development Error Tracer and if possible, it shall return E_NOT_OK to the calling function if development error detection is enabled (CANNM_DEV_ERROR_DETECT is set to TRUE).⌋()

[SWS_CanNm_00193] ⌈When the NM-Timeout Timer expires in the Repeat Message State, the CanNm module shall report CANNM_E_NETWORK_TIMEOUT to the Development Error Tracer⌋()

[SWS_CanNm_00194] ⌈When the NM-Timeout Timer expires in the Normal Operation State, the CanNm module shall report CANNM_E_NETWORK_TIMEOUT to the Development Error Tracer ⌋()

7.17 Scheduling of the main function

For details refer to the chapter 8.5 “Scheduled functions” in *SWS_BSWGeneral.*

7.18 Application notes

7.18.1 Wakeup notification

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

7.18.2 Coordination of coupled networks

[SWS_CanNm_00185] ⌈Support of bus synchronization on demand shall be statically configurable with use of the `CANNM_BUS_SYNCHRONIZATION_ENABLED` switch (configuration parameter).⌋()

Since the shutdown of CanNm can be done at any time, the call of the API `Nm_SynchronizationPoint`(), which is specified in concept 065, is not supported.

7.18.3 Debugging Concept

For details refer to the chapter 7.1.17 “Debugging support” in *SWS_BSWGeneral.*

7.19 Summary of CanNm Timing Requirements

This section gives a summary of the CanNm timing requirements. Please note that this chapter is a summary only and does not replace or act as requirement. Moreover this section does not require any specific way of implementation

Type of timing	***Requirements***
Nm timeout related	SWS_CanNm_00061 SWS_CanNm_00096 SWS_CanNm_00098 SWS_CanNm_00099 SWS_CanNm_00101 SWS_CanNm_00109 SWS_CanNm_00117 SWS_CanNm_00174 SWS_CanNm_00179 SWS_CanNm_00193 SWS_CanNm_00194 SWS_CanNm_00206
Tx confirmation timeout related	SWS_CanNm_00064 SWS_CanNm_00065 SWS_CanNm_00066 CANNM067 CANNM068
NmPdu transmission related	SWS_CanNm_00005 SWS_CanNm_00032 SWS_CanNm_00040 SWS_CanNm_00051 SWS_CanNm_00061 SWS_CanNm_00069 CANNM169 SWS_CanNm_00173 SWS_CanNm_00178
Remote sleep indication related	SWS_CanNm_00175 SWS_CanNm_00180

8 API specification

[SWS_CanNm_00244] ⌈The CanNm module shall reject the execution of a service called with an invalid parameter and shall inform the DET.⌋(SRS_BSW_00323)

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

8.1 Imported Types

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

[SWS_CanNm_00245] ⌈

Module	***Imported Type***
ComStack_Types	NetworkHandleType
	PduIdType
	PduInfoType
Nm	Nm_ModeType
	Nm_StateType
Std_Types	Std_ReturnType
	Std_VersionInfoType

⌋()

8.2 Type Definitions

8.2.1 CanNm_ConfigType

[SWS_CanNm_00447]⌈

Name:	`CanNm_ConfigType`	
Type:	`Structure`	
Range:	`implementation specific`	--
Description:	This type shall contain at least all parameters that are post-build able according to chapter 10.	

⌋()

8.3 CanNm Functions called by the Nm

8.3.1 CanNm_Init

[SWS_CanNm_00208] ⌈

Service name:	CanNm_Init

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

Service ID[hex]:	0x00

Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	cannmConfigPtr	Pointer to a selected configuration structure
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Initialize the CanNm module.	

⌋()

[SWS_CanNm_00253] ⌈Caveats of `CanNm_Init`: The function `CanNm_Init` has to be called after initialization of the CanIf.⌋()

8.3.2 CanNm_PassiveStartUp

[SWS_CanNm_00211] ⌈

Service name:	CanNm_PassiveStartUp	

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

Service ID[hex]:	0x01	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant (but not for the same NM-Channel)	
Parameters (in):	nmChannelHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: No error E_NOT_OK: Passive startup of network management has failed
Description:	Passive startup of the AUTOSAR CAN NM. It triggers the transition from Bus-Sleep Mode or Prepare Bus Sleep Mode to the Network Mode in Repeat Message State. Caveats: CanNm is initialized correctly. Configuration: Mandatory	

⌋()

[SWS_CanNm_00254] ⌈Caveats of `CanNm_PassiveStartUp`: The CanNm module is initialized correctly.⌋()

8.3.3 CanNm_NetworkRequest

[SWS_CanNm_00213]?

Service name:	CanNm_NetworkRequest

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

Service ID[hex]:	0x02	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant (but not for the same NM-channel)	
Parameters (in):	nmChannelHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: No error E_NOT_OK: Requesting of network has failed
Description:	Request the network, since ECU needs to communicate on the bus.	

⌋()

[SWS_CanNm_00255] ⌈The function `CanNm_NetworkRequest` shall change the Network state to ‘requested’.⌋()

[SWS_CanNm_00256] ⌈Caveats of `CanNm_NetworkRequest`: The CanNm module is initialized correctly.⌋()

[SWS_CanNm_00257] ⌈Configuration of `CanNm_NetworkRequest`: Optional (Only available if `CANNM_PASSIVE_MODE_ENABLED` is not defined).⌋()

8.3.4 CanNm_NetworkRelease

[SWS_CanNm_00214] ⌈

Service name:	CanNm_NetworkRelease	

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

Service ID[hex]:	0x03	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant (but not for the same NM-Channel)	
Parameters (in):	nmChannelHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: No error E_NOT_OK: Releasing of network has failed
Description:	Release the network, since ECU doesn't have to communicate on the bus.	

⌋()

[SWS_CanNm_00259] ⌈Caveats of `CanNm_NetworkRelease`: The CanNm module is initialized correctly.⌋()

[SWS_CanNm_00260] ⌈Configuration of `CanNm_NetworkRelease`: Optional (Only available if `CANNM_PASSIVE_MODE_ENABLED` is not defined)⌋()

8.3.5 CanNm_DisableCommunication

[SWS_CanNm_00215] ⌈

Service name:	CanNm_DisableCommunication	

Syntax:	`Std_ReturnType CanNm_DisableCommunication(`
	`    const NetworkHandleType nmChannelHandle`
	`)`	

Service ID[hex]:	0x0c	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant (but not for the same NM-channel)	
Parameters (in):	nmChannelHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: No error E_NOT_OK: Disabling of NM PDU transmission ability has failed
Description:	Disable the NM PDU transmission ability due to a ISO14229 Communication Control (28hex) service	

⌋()

[SWS_CanNm_00261] ⌈Caveats of `CanNm_DisableCommunication`: The CanNm module is initialized correctly.⌋()

[SWS_CanNm_00262] ⌈Configuration of `CanNm_DisableCommunication`: Optional (Only available if `CANNM_COM_CONTROL_ENABLED` is set to TRUE)⌋()

[SWS_CanNm_00172] ⌈The service `CanNm_DisableCommunication` shall return `E_NOT_OK`, if the current mode is not Network Mode.⌋()

[SWS_CanNm_00298] ⌈If the module operates in passive mode (`CANNM_PASSIVE_MODE_ENABLED`) the service `CanNm_DisableCommunication` shall have no effects and shall directly return `E_NOT_OK`.⌋()

8.3.6 CanNm_EnableCommunication

[SWS_CanNm_00216] ⌈

Service name:	CanNm_EnableCommunication	

Syntax:	`Std_ReturnType CanNm_EnableCommunication(`
	`    const NetworkHandleType nmChannelHandle`
	`)`	

Service ID[hex]:	0x0d	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant (but not for the same NM-channel)	
Parameters (in):	nmChannelHandle	Identification of the NM-channel
Parameters (inout):	None	

Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: No error E_NOT_OK: Enabling of NM PDU transmission ability has failed
Description:	Enable the NM PDU transmission ability due to a ISO14229 Communication Control (28hex) service	

⌋()

[SWS_CanNm_00176] ⌈The service `CanNm_EnableCommunication` shall enable the Network Management PDU transmission ability if the Network Management PDU transmission ability is disabled.⌋()

[SWS_CanNm_00177] ⌈The service `CanNm_EnableCommunication` shall return `E_NOT_OK` if the Network Management PDU transmission ability is enabled.⌋()

[SWS_CanNm_00295] ⌈The service `CanNm_EnableCommunication` shall return `E_NOT_OK`, if the current mode is not Network Mode.⌋()

[SWS_CanNm_00263] ⌈Caveats of `CanNm_EnableCommunication`: The CanNm module is initialized correctly.⌋()

[SWS_CanNm_00264] ⌈Configuration of `CanNm_EnableCommunication`: Optional (Only available if `CANNM_COM_CONTROL_ENABLED` is set to `TRUE`).⌋()

[SWS_CanNm_00297] ⌈If the module operates in passive mode (`CANNM_PASSIVE_MODE_ENABLED`) the service `CanNm_EnableCommunication` shall have no effects and shall directly return `E_NOT_OK`.⌋()

8.3.7 CanNm_SetUserData

[SWS_CanNm_00217] ⌈

Service name:	CanNm_SetUserData	

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

Service ID[hex]:	0x04	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	nmChannelHandle	Identification of the NM-channel
	nmUserDataPtr	Pointer where the user data for the next transmitted NM PDU shall be copied from
Parameters (inout):	None	
Parameters (out):	None	

Return value:	Std_ReturnType	E_OK: No error E_NOT_OK: Setting of user data has failed
Description:	Set user data for NM PDUs transmitted next on the bus.	

⌋()

[SWS_CanNm_00265] ⌈Caveats of `CanNm_SetUserData`: The CanNm module is initialized correctly.⌋()

[SWS_CanNm_00266] ⌈Configuration of `CanNm_SetUserData`: Optional (Only available if `CANNM_USER_DATA_ENABLED` is set to `TRUE` and `CANNM_PASSIVE_MODE_ENABLED` is not defined)⌋()

8.3.8 CanNm_GetUserData

[SWS_CanNm_00218]?

Service name:	CanNm_GetUserData	

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

Service ID[hex]:	0x05	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	nmChannelHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	nmUserDataPtr	Pointer where user data out of the most recently received NM PDU shall be copied to
Return value:	Std_ReturnType	E_OK: No error E_NOT_OK: Getting of user data has failed
Description:	Get user data out of the most recently received NM PDU.	

⌋()

[SWS_CanNm_00267] ⌈Caveats of `CanNm_GetUserData`: The CanNm module is initialized correctly.⌋()

[SWS_CanNm_00268] ⌈Configuration of `CanNm_GetUserData`: Optional (Only available if `CANNM_USER_DATA_ENABLED` is set to `TRUE`).⌋()

8.3.9 CanNm_Transmit

[SWS_CanNm_00331] ⌈

Service name:	CanNm_Transmit

Syntax:	`Std_ReturnType CanNm_Transmit(`
	`    PduIdType CanNmTxPduId,`
	`    const PduInfoType* PduInfoPtr`

	)	
Service ID[hex]:	0x14	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	CanNmTxPduId	Handle of the CanNm user data PDU.
	PduInfoPtr	Pointer to a structure containing the pointer to the NM user data buffer and the related DLC.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Transmit request has been accepted E_NOT_OK: Transmit request has not been accepted (CanNm is not in RM or NO)
Description:	This function is used by the PduR to trigger a spontaneous transmission of an NM PDU with the provided NM User Data.	

⌋()

[SWS_CanNm_00330] ⌈If `CanNmComUserDataSupport` is enabled the CanNm implementation shall provide an API `CanNm_Transmit`.⌋()

[SWS_CanNm_00333] ⌈If CanNm is in RepeatMessage state or NormalOperation state and if `CanNm_Transmit()` is called CanNm shall request an additional transmission of the NM PDU with the current user data.⌋()

8.3.10 CanNm_GetNodeIdentifier

[SWS_CanNm_00219] ⌈

Service name:	CanNm_GetNodeIdentifier	

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

Service ID[hex]:	0x06	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	nmChannelHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	nmNodeIdPtr	Pointer where node identifier out of the most recently received NM PDU shall be copied to
Return value:	Std_ReturnType	E_OK: No error E_NOT_OK: Getting of the node identifier out of the most recently received NM PDU has failed
Description:	Get node identifier out of the most recently received NM PDU.	

⌋()

[SWS_CanNm_00132] ⌈The service call `CanNm_GetNodeIdentifier` shall provide the node identifier out of the most recently received Network Management PDU.⌋()

[SWS_CanNm_00269] ⌈Caveats of `CanNm_GetNodeIdentifier`: The CanNm module is initialized correctly.⌋()

[SWS_CanNm_00270] ⌈Configuration of `CanNm_GetNodeIdentifier`: Optional (Only available if `CANNM_NODE_ID_ENABLED` is set to TRUE).⌋()

8.3.11 CanNm_GetLocalNodeIdentifier

[SWS_CanNm_00220] ⌈

Service name:	CanNm_GetLocalNodeIdentifier	

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

Service ID[hex]:	0x07	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	nmChannelHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	nmNodeIdPtr	Pointer where node identifier of the local node shall be copied to
Return value:	Std_ReturnType	E_OK: No error E_NOT_OK: Getting of the node identifier of the local node has failed
Description:	Get node identifier configured for the local node.	

⌋()

[SWS_CanNm_00133] ⌈The service call `CanNm_GetLocalNodeIdentifier` shall provide the node identifier configured for the local host node.⌋()

[SWS_CanNm_00271] ⌈Caveats of `CanNm_GetLocalNodeIdentifier`: The CanNm module is initialized correctly.⌋()

[SWS_CanNm_00272] ⌈Configuration of `CanNm_GetLocalNodeIdentifier`: Optional (Only available if `CANNM_NODE_ID_ENABLED` is set to TRUE.).⌋()

8.3.12 CanNm_RepeatMessageRequest

[SWS_CanNm_00221] ⌈

Service name:	CanNm_RepeatMessageRequest

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

Service ID[hex]:	0x08	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant (but not for the same NM-channel)	
Parameters (in):	nmChannelHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: No error E_NOT_OK: Setting of Repeat Message Request Bit has failed
Description:	Set Repeat Message Request Bit for NM PDUs transmitted next on the bus.	

⌋()

[SWS_CanNm_00273] ⌈Caveats of `CanNm_RepeatMessageRequest`: The CanNm module is initialized correctly.⌋()

[SWS_CanNm_00274] ⌈Configuration of `CanNm_RepeatMessageRequest`: Optional (Only available if `CANNM_NODE_DETECTION_ENABLED` is set to `TRUE`).⌋()

8.3.13 CanNm_GetPduData

[SWS_CanNm_00222] ⌈

Service name:	CanNm_GetPduData	

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

Service ID[hex]:	0x0a	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	nmChannelHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	nmPduDataPtr	Pointer where NM PDU shall be copied to
Return value:	Std_ReturnType	E_OK: No error E_NOT_OK:Getting of NM PDU data has failed
Description:	Get the whole PDU data out of the most recently received NM PDU.	

⌋()

[SWS_CanNm_00275] ⌈Caveats of **`CanNm_GetPduData`**: The CanNm module is initialized correctly.⌋()

[SWS_CanNm_00276] ⌈Configuration of `CanNm_GetPduData`: Optional (Only available if `CANNM_NODE_DETECTION_ENABLED` or `CANNM_USER_DATA_ENABLED` or `CANNM_NODE_ID_ENABLED` is set to TRUE).⌋()

8.3.14 CanNm_GetState

[SWS_CanNm_00223] ⌈

Service name:	CanNm_GetState	

Syntax:	`Std_ReturnType CanNm_GetState(`
	`    const NetworkHandleType nmChannelHandle,`
	`    Nm_StateType* const nmStatePtr,`
	`    Nm_ModeType* const nmModePtr`
	`)`	

Service ID[hex]:	0x0b	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	nmChannelHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	nmStatePtr	Pointer where state of the network management shall be copied to
	nmModePtr	Pointer where the mode of the network management shall be copied to
Return value:	Std_ReturnType	E_OK: No error E_NOT_OK: Getting of NM state has failed
Description:	Returns the state and the mode of the network management.	

⌋()

[SWS_CanNm_00277] ⌈Caveats of `CanNm_GetState`: The CanNm module is initialized correctly.⌋()

8.3.15 CanNm_GetVersionInfo

[SWS_CanNm_00224] ⌈

Service name:	CanNm_GetVersionInfo	

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

Service ID[hex]:	0xf1	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	versioninfo	Pointer to where to store the version information of this module
Return value:	None	
Description:	This service returns the version information of this module.	

⌋()

8.3.16 CanNm_RequestBusSynchronization

[SWS_CanNm_00226] ⌈

Service name:	CanNm_RequestBusSynchronization	

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

Service ID[hex]:	0xc0	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	nmChannelHandle	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 synchronization has failed
Description:	Request bus synchronization.	

⌋(SRS_Nm_02516)

[SWS_CanNm_00279] ⌈Caveats of `CanNm_RequestBusSynchronization`: The CanNm module is initialized correctly.⌋()

[SWS_CanNm_00280] ⌈Configuration of `CanNm_RequestBusSynchronization`: Optional (Only available if `CANNM_BUS_SYNCHRONIZATION_ENABLED` is set to `TRUE`) and `CANNM_PASSIVE_MODE_ENABLED` is not defined.⌋(SRS_Nm_02516)

[SWS_CanNm_00130] ⌈The service call `CanNm_RequestBusSynchronization` shall trigger transmission of a single Network Management PDU if `CANNM_PASSIVE_MODE_ENABLED` (configuration parameter) is not defined.⌋(SRS_Nm_02516)

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

[SWS_CanNm_00187] ⌈If `CanNm_RequestBusSynchronization` is called in Bus-Sleep Mode and Prepare Bus-Sleep Mode the CanNm module shall not execute the service and shall return `E_NOT_OK`.⌋()

8.3.17 CanNm_CheckRemoteSleepIndication

[SWS_CanNm_00227] ⌈

Service name:	CanNm_CheckRemoteSleepIndication

Syntax:	`Std_ReturnType CanNm_CheckRemoteSleepIndication(`
	`    const NetworkHandleType nmChannelHandle,`
	`    boolean* const nmRemoteSleepIndPtr`
	`)`

Service ID[hex]:	0xd0	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant (but not for the same NM-channel)	
Parameters (in):	nmChannelHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	nmRemoteSleepIndPtr	Pointer where check result of remote sleep indication shall be copied to
Return value:	Std_ReturnType	E_OK: No error E_NOT_OK: Checking of remote sleep indication bits has failed
Description:	Check if remote sleep indication takes place or not.	

⌋()

[SWS_CanNm_00153] ⌈Service call `CanNm_CheckRemoteSleepIndication` shall provide the information about current status of Remote Sleep Indication (i.e. already detected or not).⌋()

[SWS_CanNm_00281] ⌈Caveats of `CanNm_CheckRemoteSleepIndication`: The CanNm module is initialized correctly.⌋()

[SWS_CanNm_00282] ⌈Configuration of `CanNm_CheckRemoteSleepIndication`: Optional (Only available if `CANNM_REMOTE_SLEEP_IND_ENABLED` is set to `TRUE`).⌋()

8.3.18 CanNm_SetSleepReadyBit

[SWS_CanNm_00338] ⌈

Service name:	CanNm_SetSleepReadyBit	

Syntax:	`Std_ReturnType CanNm_SetSleepReadyBit(`
	`    const NetworkHandleType nmChannelHandle,`
	`    const boolean nmSleepReadyBit`
	`)`	

Service ID[hex]:	0x17	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	nmChannelHandle	Identification of the NM-channel
	nmSleepReadyBit	Value written to ReadySleep Bit in CBV
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: No error E_NOT_OK: Writing of remote sleep indication bit has failed
Description:	Set the NM Coordinator Sleep Ready bit in the Control Bit Vector	

⌋()

[SWS_CanNm_00339] ⌈Caveats of `CanNm_SetSleepReadyBit`: The CanNm module is initialized correctly.⌋()

[SWS_CanNm_00340] ⌈Configuration of `CanNm_SetSleepReadyBit`: Optional (Only available if `CANNM_COORDINATOR_SYNC_SUPPORT` is set to `TRUE`).⌋()

8.4 CanNm functions called by the CanIf

8.4.1 CanNm_TxConfirmation

[SWS_CanNm_00228] ⌈

Service name:	CanNm_TxConfirmation	

Syntax:	`void CanNm_TxConfirmation(`
	`    PduIdType TxPduId`
	`)`	

Service ID[hex]:	0x40	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant for different PduIds. Non reentrant for the same PduId.	
Parameters (in):	TxPduId	ID of the I-PDU that has been transmitted.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	The lower layer communication interface module confirms the transmission of an I-PDU.	

⌋()

Note: The callback function `CanNm_TxConfirmation` is called by the CAN Interface and implemented by the CanNm module.

[SWS_CanNm_00229] ⌈The callback function `CanNm_TxConfirmation` shall inform the DET, if development error detection is enabled (`CANNM_DEV_ERROR_DETECT` is set to `TRUE`) and if function call has failed because of the following reasons:

- Invalid PDU ID (`CANNM_E_INVALID_PDUID`)
- CanNm was not initialized (`CANNM_E_NO_INIT`)⌋()

[SWS_CanNm_00283] ⌈Caveats of `CanNm_TxConfirmation`:

- The call context is either on interrupt level (interrupt mode) or on task level (polling mode). This callback service is re-entrant for multiple CAN controller usage.
- The CanNm module is initialized correctly.⌋()

[SWS_CanNm_00284] ⌈Configuration of CanNm_TxConfirmation: Optional (Only available if CANNM_IMMEDIATE_TXCONF_ENABLED is set to FALSE).⌋()

8.4.2 CanNm_RxIndication

[SWS_CanNm_00231] ⌈

Service name:	CanNm_RxIndication	

Syntax:	`void CanNm_RxIndication(`
	`    PduIdType RxPduId,`
	`    const PduInfoType* PduInfoPtr`
	`)`	

Service ID[hex]:	0x42	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant for different PduIds. Non reentrant for the same PduId.	
Parameters (in):	RxPduId	ID of the received I-PDU.
	PduInfoPtr	Contains the length (SduLength) of the received I-PDU and a pointer to a buffer (SduDataPtr) containing the I-PDU.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Indication of a received I-PDU from a lower layer communication interface module.	

⌋()

Note: The callback function CanNm_RxIndication called by the CAN Interface and implemented by the CanNm module. It is called in case of a receive indication event of the CAN Driver.

[SWS_CanNm_00232] ⌈The callback function CanNm_RxIndication shall inform the DET, if development error detection is enabled (CANNM_DEV_ERROR_DETECT is set to TRUE) and if function call has failed because of the following reasons:

- Invalid PDU ID (CANNM_E_INVALID_PDUID)
- CanNm was not initialized (CANNM_E_NO_INIT)
- PduInfoPtr or SduDataPtr equals NULL_PTR (CANNM_E_NULL_POINTER)⌋()

[SWS_CanNm_00285] ⌈Caveats of CanNm_RxIndication:

- Until this service returns the CAN Interface will not access canSduPtr. The canSduPtr is only valid and can be used by upper layers until the indication returns. CAN Interface guarantees that the number of configured bytes for this canNmRxPduId is valid. The call context is either on interrupt level (interrupt mode) or on task level (polling mode). This callback service is re-entrant for multiple CAN controller usage.
- The CanNm module is initialized correctly.⌋()

8.5 Called by CanSM

8.5.1 CanNm_ConfirmPnAvailability

[SWS_CanNm_00344] ⌈

Service name:	CanNm_ConfirmPnAvailability	

Syntax:	`void CanNm_ConfirmPnAvailability(`
	`    const NetworkHandleType nmChannelHandle`
	`)`	

Service ID[hex]:	0x16	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant (but not for the same NM-channel)	
Parameters (in):	nmChannelHandle	Identification of the NM-channel
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Enables the PN filter functionality on the indicated NM channel. Availability: The API is only available if CanNmPnEnabled is TRUE.	

⌋()

[SWS_CanNm_00345] ⌈Caveats of `CanNm_ConfirmPnAvailability`: The CanNm module is initialized correctly.⌋()

[SWS_CanNm_00346] ⌈Configuration of `CanNm_ConfirmPnAvailability`: Optional (Only available if `CANNM_PN_ENABLED` is set to `TRUE`).⌋()

8.6 Scheduled Functions

8.6.1 CanNm_MainFunction

[SWS_CanNm_00234] ⌈

Service name:	CanNm_MainFunction

Syntax:	`void CanNm_MainFunction(`
	`    void`
	`)`

Service ID[hex]:	0x13
Description:	Main function of the CanNm which processes the algorithm describes in that document.

⌋()

[SWS_CanNm_00235] ⌈The scheduled function `CanNm_MainFunction` shall inform the DET, if development error detection is enabled (`CANNM_DEV_ERROR_DETECT` is set to TRUE) and if function call has failed because of the following reasons:

- The CanNm module was not initialized (`CANNM_E_NO_INIT`).⌋()

8.7 Expected Interfaces

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

8.7.1 Mandatory Interfaces

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

[SWS_CanNm_00324] ⌈

API function	***Description***
Nm_BusSleepMode	Notification that the network management has entered Bus-Sleep Mode.
Nm_NetworkMode	Notification that the network management has entered Network Mode.
Nm_NetworkStartIndication	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.
Nm_PrepareBusSleepMode	Notification that the network management has entered Prepare Bus-Sleep Mode.

⌋()

8.7.2 Optional Interfaces

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

[SWS_CanNm_00325] ⌈

API function	***Description***
CanIf_CancelTransmit	This is a dummy method introduced for interface compatibility.
CanIf_Transmit	This service initiates a request for transmission of the CAN L-PDU specified by the CanTxSduId and CAN related data in the L-SDU structure.
CanSM_TxTimeoutException	This function shall notify the CanSM module, that the CanNm has detected for the affected partial CAN network a tx timeout exception, which shall be recovered within the respective network state machine of the CanSM module.
Det_ReportError	Service to report development errors.
Nm_CarWakeUpIndication	This function is called by a <Bus>Nm to indicate reception of a CWU request.
Nm_CoordReadyToSleepCancellation	Cancels an indication, when the NM Coordinator Sleep Ready bit in the Control Bit Vector is set back to 0.
Nm_CoordReadyToSleepIndication	Sets an indication, when the NM Coordinator Sleep Ready bit in the Control Bit Vector is set
Nm_PduRxIndication	Notification that a NM message has been received.
Nm_RemoteSleepCancellation	Notification that the network management has detected that not all other nodes on the network are longer ready to enter Bus-Sleep Mode.
Nm_RemoteSleepIndication	Notification that the network management has detected that all other nodes on the network are ready to enter Bus-Sleep Mode.
Nm_RepeatMessageIndication	Service to indicate that an NM message with set Repeat Message Request Bit has been received.
Nm_StateChangeNotification	Notification that the state of the lower layer <BusNm> has

	changed.
Nm_TxTimeoutException	Service to indicate that an attempt to send an NM message failed.
PduR_CanNmRxIndication	Indication of a received I-PDU from a lower layer communication interface module.
PduR_CanNmTriggerTransmit	Within this API, the upper layer module (called module) shall copy its data into the buffer provided by PduInfoPtr->SduDataPtr and update the length of the actual copied data in PduInfoPtr->SduLength.
PduR_CanNmTxConfirmation	The lower layer communication interface module confirms the transmission of an I-PDU.

⌋()

8.7.3 Configurable interfaces

Not applicable

8.7.4 Job End Notification

Not applicable

8.8 Parameter check

[SWS_CanNm_00196] ⌈If detection of development errors is enabled by `CANNM_DEV_ERROR_DETECT` (configuration parameter), then for all CanNm API services validity check of input parameters shall be made.
Exception: The `NULL` Pointer check of input parameters shall not be done for `CanNm_Init.`⌋()

[SWS_CanNm_00197] ⌈Parameter type checking shall be made at compile time;
if types do not fit the compilation process shall be stopped and respective compilation warnings or errors shall be returned as far as supported by the compiler.⌋()

[SWS_CanNm_00198] ⌈Parameter value check (for parameters of the constant value) shall be made at configuration time; if the value is invalid, the configuration process shall be stopped and respective configuration error shall be reported.⌋()

[SWS_CanNm_00199] ⌈Parameter value check (for parameters of the variable value) shall be made at execution time; if the value is invalid, execution of a service shall be rejected and respective development error shall be reported.⌋()

8.9 Version check

For details refer to the chapter 5.1.8 “Version Check” in *SWS_BSWGeneral.*

8.10 UML State chart diagram

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

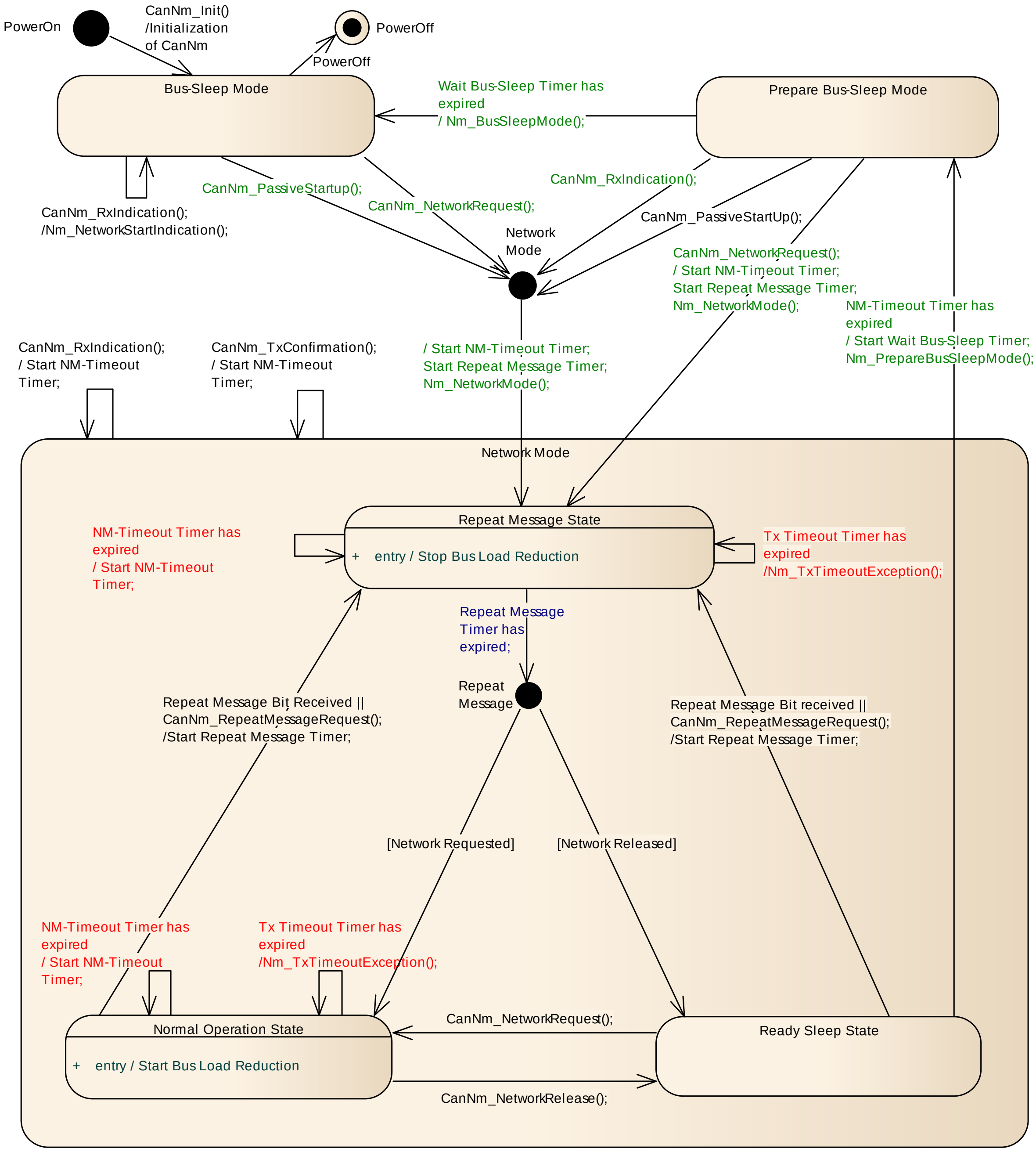
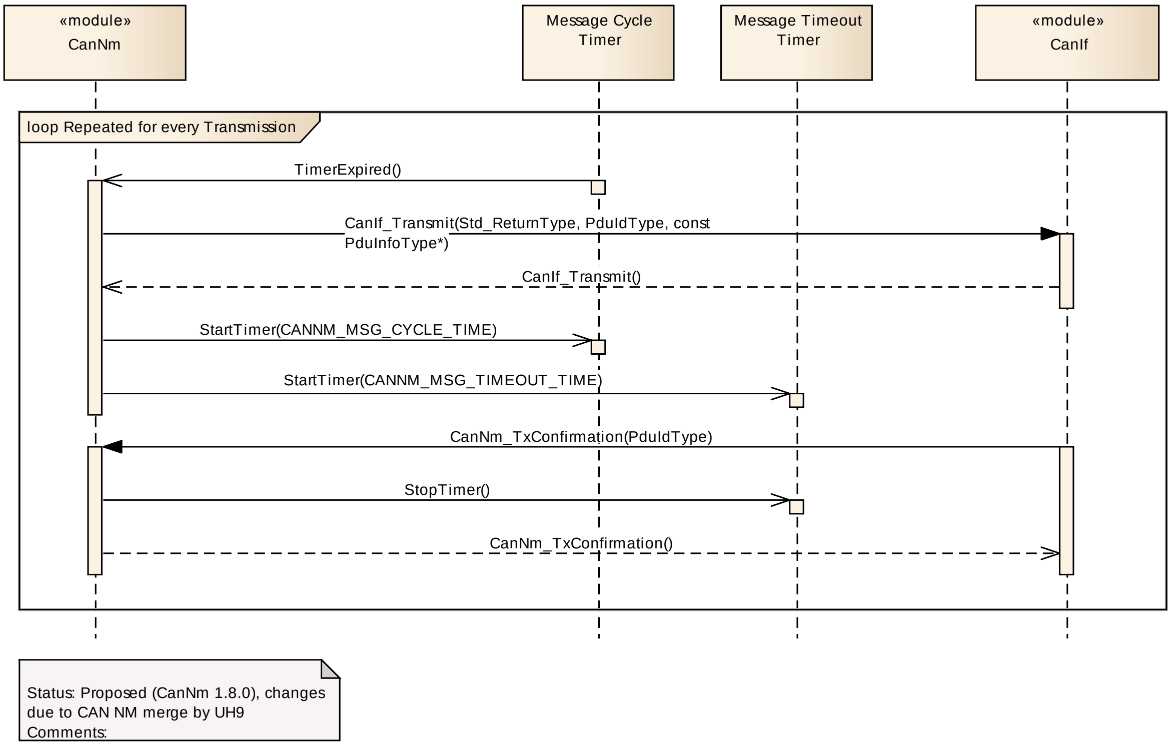


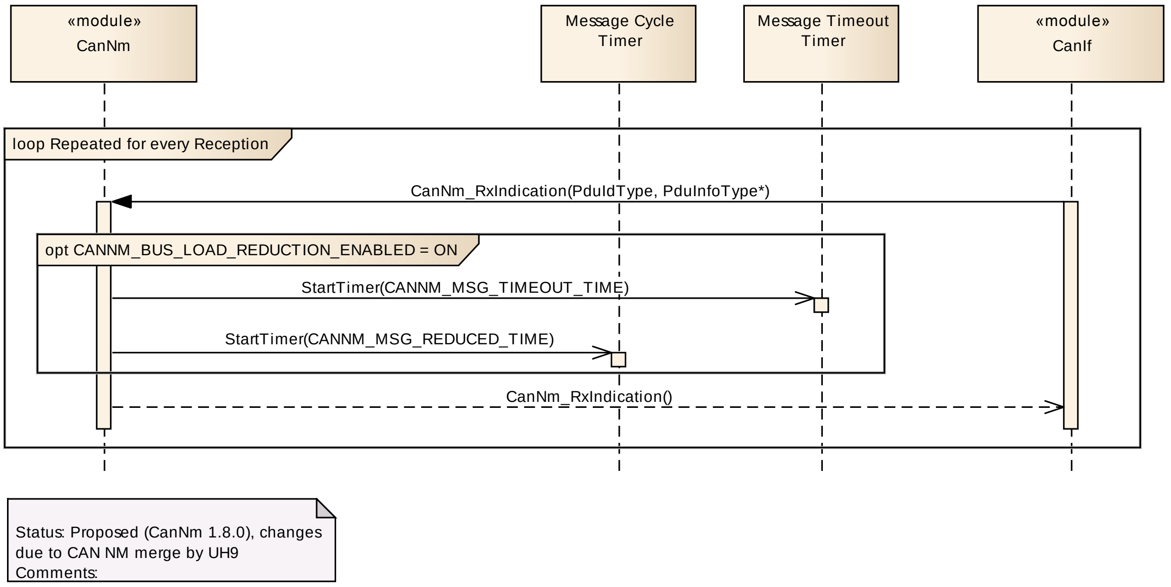
Figure 8-1 CanNm Algorithm

9 Sequence diagrams

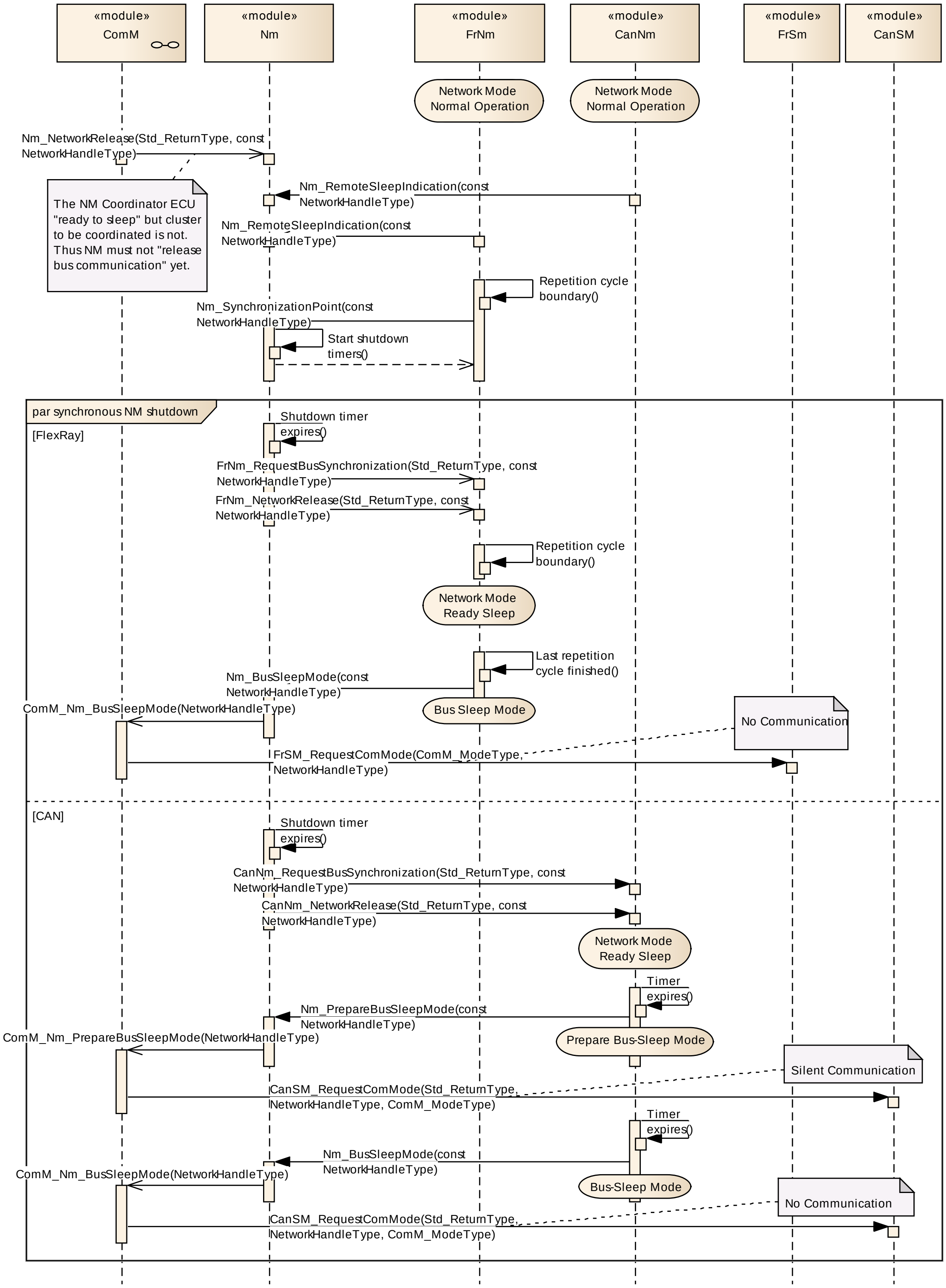
9.1 CanNm Transmission


«module»
CanNm
Message Cycle Timer
Message Timeout Timer
«module»
CanIf
loop Repeated for every Transmission
TimerExpired()
CanIf_Transmit(Std_ReturnType, PduIdType, const PduInfoType*)
CanIf_Transmit()
StartTimer(CANNM_MSG_CYCLE_TIME)
StartTimer(CANNM_MSG_TIMEOUT_TIME)
CanNm_TxConfirmation(PduIdType)
StopTimer()
CanNm_TxConfirmation()
Status: Proposed (CanNm 1.8.0), changes due to CAN NM merge by UH9
Comments:

9.2 CanNm Reception


«module»
CanNm
Message Cycle Timer
Message Timeout Timer
«module»
CanIf
loop Repeated for every Reception
CanNm_RxIndication(PduIdType, PduInfoType*)
opt CANNM_BUS_LOAD_REDUCTION_ENABLED = ON
StartTimer(CANNM_MSG_TIMEOUT_TIME)
StartTimer(CANNM_MSG_REDUCED_TIME)
CanNm_RxIndication()
Status: Proposed (CanNm 1.8.0), changes due to CAN NM merge by UH9
Comments:

9.3 Nm Coordination


«module»
ComM
«module»
Nm
«module»
FrNm
«module»
CanNm
«module»
FrSm
«module»
CanSM
Network Mode
Normal Operation
Network Mode
Normal Operation
Nm_NetworkRelease(Std_ReturnType, const
NetworkHandleType)
The NM Coordinator ECU "ready to sleep" but cluster to be coordinated is not. Thus NM must not "release bus communication" yet.
Nm_RemoteSleepIndication(const
NetworkHandleType)
Nm_RemoteSleepIndication(const
NetworkHandleType)
Repetition cycle
boundary()
Nm_SynchronizationPoint(const
NetworkHandleType)
Start shutdown
timers()
par synchronous NM shutdown
[FlexRay]
Shutdown timer
expires()
FrNm_RequestBusSynchronization(Std_ReturnType, const
NetworkHandleType)
FrNm_NetworkRelease(Std_ReturnType, const
NetworkHandleType)
Repetition cycle
boundary()
Network Mode
Ready Sleep
Last repetition
cycle finished()
Nm_BusSleepMode(const
NetworkHandleType)
ComM_Nm_BusSleepMode(NetworkHandleType)
Bus Sleep Mode
No Communication
FrSM_RequestComMode(ComM_ModeType,
NetworkHandleType)
[CAN]
Shutdown timer
expires()
CanNm_RequestBusSynchronization(Std_ReturnType, const
NetworkHandleType)
CanNm_NetworkRelease(Std_ReturnType, const
NetworkHandleType)
Network Mode
Ready Sleep
Timer
expires()
Nm_PrepareBusSleepMode(const
NetworkHandleType)
ComM_Nm_PrepareBusSleepMode(NetworkHandleType)
Prepare Bus-Sleep Mode
Silent Communication
CanSM_RequestComMode(Std_ReturnType,
NetworkHandleType, ComM_ModeType)
Timer
expires()
Nm_BusSleepMode(const
NetworkHandleType)
ComM_Nm_BusSleepMode(NetworkHandleType)
Bus-Sleep Mode
No Communication
CanSM_RequestComMode(Std_ReturnType,
NetworkHandleType, ComM_ModeType)

10 Configuration specification

In general, this chapter defines configuration parameters and their clustering into containers. In order to support the specification chapter 10.1 describes fundamentals. It also specifies a template (table) you shall use for the parameter specification. We intend to leave chapter 10.1 in the specification to guarantee comprehension.

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

Chapter 10.3 specifies published information of the module CanNm.

10.1 How to read this chapter

For details refer to the chapter 10.1 “Introduction to configuration specification” in *SWS_BSWGeneral.*

10.2 Containers and configuration parameters

The following chapters summarize all configuration parameters. The detailed meanings of the parameters are described in the chapters 7 and 7.19.

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

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

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

10.2.1 Variants

[SWS_CanNm_00250]⌈VARIANT-PRE-COMPILE: Only parameters with “Pre-compile time” configuration are allowed in this variant.⌋()

[SWS_CanNm_00251] ⌈VARIANT-LINK-TIME: Only parameters with “Pre-compile time” and “Link time” are allowed in this variant.⌋()

[SWS_CanNm_00252]⌈ VARIANT-POST-BUILD: Parameters with “Pre-compile time”, “Link time” and “Post-build time” are allowed in this variant. ⌋()

10.3 Containers and configuration parameters

This chapter describes the configuration container and parameters used for CanNm configuration.

10.3.1 CanNm Global Configuration Overview

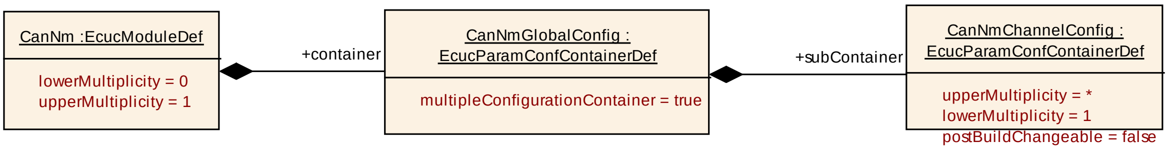

CanNm :EcucModuleDef
lowerMultiplicity = 0
upperMultiplicity = 1
+container
CanNmGlobalConfig : EcucParamConfContainerDef
multipleConfigurationContainer = true
+subContainer
CanNmChannelConfig : EcucParamConfContainerDef
upperMultiplicity = *
lowerMultiplicity = 1
postBuildChangeable = false

Figure 10-1 CanNm top level configuration overview

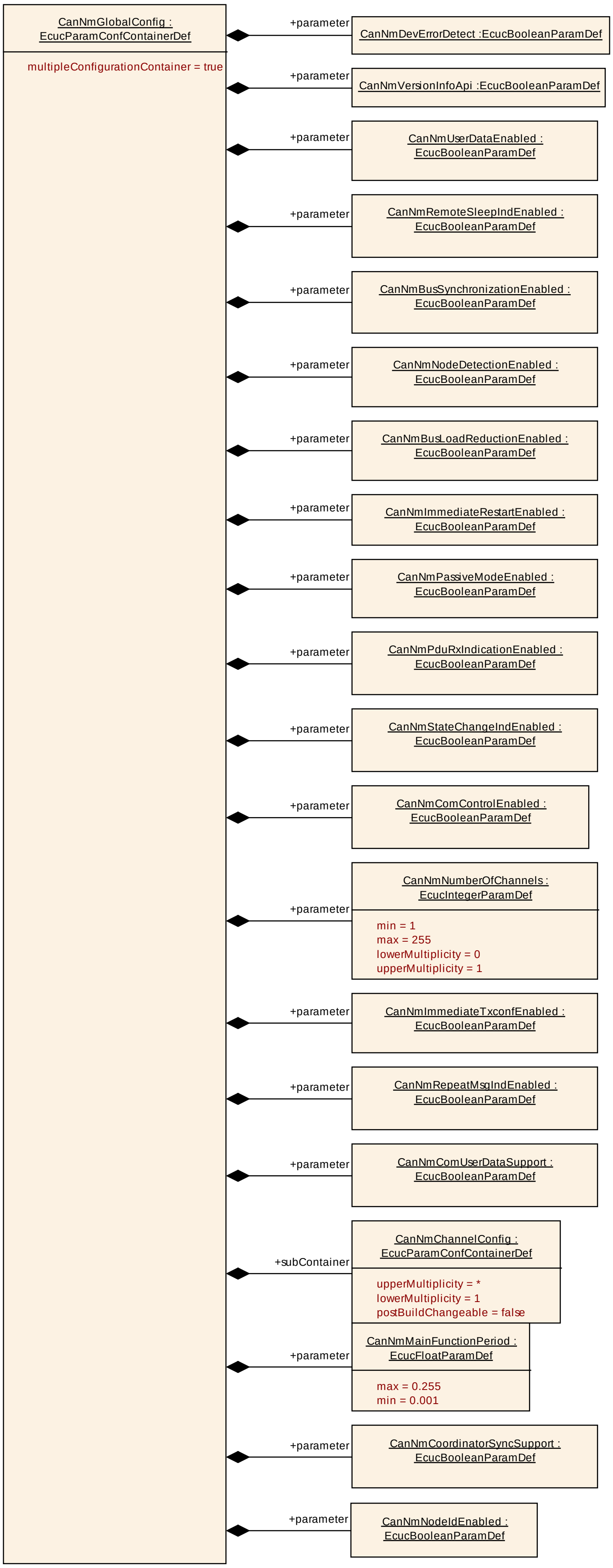

CanNmGlobalConfig :
EcucParamConfContainerDef
multipleConfigurationContainer = true
+parameter
CanNmDevErrorDetect :EcucBooleanParamDef
+parameter
CanNmVersionInfoApi :EcucBooleanParamDef
+parameter
CanNmUserDataEnabled :
EcucBooleanParamDef
+parameter
CanNmRemoteSleepIndEnabled :
EcucBooleanParamDef
+parameter
CanNmBusSynchronizationEnabled :
EcucBooleanParamDef
+parameter
CanNmNodeDetectionEnabled :
EcucBooleanParamDef
+parameter
CanNmBusLoadReductionEnabled :
EcucBooleanParamDef
+parameter
CanNmImmediateRestartEnabled :
EcucBooleanParamDef
+parameter
CanNmPassiveModeEnabled :
EcucBooleanParamDef
+parameter
CanNmPduRxIndicationEnabled :
EcucBooleanParamDef
+parameter
CanNmStateChangeIndEnabled :
EcucBooleanParamDef
+parameter
CanNmComControlEnabled :
EcucBooleanParamDef
+parameter
CanNmNumberOfChannels :
EcucIntegerParamDef
min = 1
max = 255
lowerMultiplicity = 0
upperMultiplicity = 1
+parameter
CanNmImmediateTxconfEnabled :
EcucBooleanParamDef
+parameter
CanNmRepeatMsgIndEnabled :
EcucBooleanParamDef
+parameter
CanNmComUserDataSupport :
EcucBooleanParamDef
+subContainer
CanNmChannelConfig :
EcucParamConfContainerDef
upperMultiplicity = *
lowerMultiplicity = 1
postBuildChangeable = false
+parameter
CanNmMainFunctionPeriod :
EcucFloatParamDef
max = 0.255
min = 0.001
+parameter
CanNmCoordinatorSyncSupport :
EcucBooleanParamDef
+parameter
CanNmNodeIdEnabled :
EcucBooleanParamDef

Figure 10-2 Parameters of CanNm global configuration

10.3.2 CanNmGlobalConfig

SWS Item	**ECUC_CanNm_00001 :**
Container Name	CanNmGlobalConfig{CanNm_GlobalConfig} [Multi Config Container]
Description	This container contains the global configuration parameter of the CanNm. The parameters and the parameters of the sub containers shall be mapped to the C data type CanNm_ConfigType (for parameters where it is possible) which is passed to the CanNm_Init function. This container is a MultipleConfigurationContainer (only for variant 3), i.e. this container and its sub-containers exit once per configuration set.
Configuration Parameters	

SWS Item	**ECUC_CanNm_00040 :**		
Name	CanNmBusLoadReductionEnabled {CANNM_BUS_LOAD_REDUCTION_ENABLED}		
Description	Pre-processor switch for enabling busload reduction support.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local dependency: CanNmBusLoadReductionEnabled = false if CanNmPassiveModeEnabled == true or CanNmPnEnabled == true for at least one CanNm Channel.		

SWS Item	**ECUC_CanNm_00006 :**		
Name	CanNmBusSynchronizationEnabled {CANNM_BUS_SYNCHRONIZATION_ENABLED}		
Description	Pre-processor switch for enabling bus synchronization support. This feature is required for gateway nodes only.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU dependency: calculationFormula = If (CanNmPassiveModeEnabled == False) then Equal(NmBusSynchronizationEnabled) else Equal(False)		

SWS Item	**ECUC_CanNm_00013 :**		
Name	CanNmComControlEnabled {CANNM_COM_CONTROL_ENABLED}		
Description	Pre-processor switch for enabling the Communication Control support.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU dependency: calculationformula = Equal(NmComControlEnabled)		

SWS Item	**ECUC_CanNm_00044 :**

Name	CanNmComUserDataSupport {CANNM_COM_USER_DATA_SUPPORT}		
Description	Enable/disable the user data support.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_CanNm_00080 :**		
Name	CanNmCoordinatorSyncSupport {CANNM_COORDINATOR_SYNC_SUPPORT}		
Description	Enables/disables the coordinator synchronisation support.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_CanNm_00002 :**		
Name	CanNmDevErrorDetect {CANNM_DEV_ERROR_DETECT}		
Description	Pre-processor switch for enabling development error detection support.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_CanNm_00009 :**		
Name	CanNmImmediateRestartEnabled {CANNM_IMMEDIATE_RESTART_ENABLED}		
Description	Pre-processor switch for enabling the asynchronous transmission of a NM PDU upon bus-communication request in Prepare-Bus-Sleep mode.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local dependency: Must not be defined if CanNmPassiveModeEnabled is defined.		

SWS Item	**ECUC_CanNm_00041 :**		
Name	CanNmImmediateTxconfEnabled {CANNM_IMMEDIATE_TXCONF_ENABLED}		
Description	Enable/disable the immediate tx confirmation.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants

	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU dependency: CanNmImmediateTxconfEnabled shall not be enabled if CanNmPasiveModeEnabled is enabled.		

SWS Item	**ECUC_CanNm_00032 :**		
Name	CanNmMainFunctionPeriod {CANNM_MAIN_FUNCTION_PERIOD}		
Description	Call cycle in seconds of CanNm_MainFunction.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	0.001 .. 0.255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_CanNm_00007 :**		
Name	CanNmNodeDetectionEnabled {CANNM_NODE_DETECTION_ENABLED}		
Description	Precompile time switch to enable the node detection feature.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU dependency: Only valid if CanNmNodeIdEnabled is set to TRUE		

SWS Item	**ECUC_CanNm_00083 :**		
Name	CanNmNodeIdEnabled {CANNM_NODE_ID_ENABLED}		
Description	Pre-processor switch for enabling the source node identifier.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU dependency: calculationFormula = Equal(NmNodeIdEnabled)		

SWS Item	**ECUC_CanNm_00014 : (Obsolete)**		
Name	CanNmNumberOfChannels {CANNM_NUMBER_OF_CHANNELS}		
Description	Number of Can NM channels allowed within one ECU. Please note that this parameter is deprecated and will be removed in future. **Tags:** atp.Status=obsolete atp.StatusRevisionBegin=4.1.1		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	1 .. 255		
Default value	--		

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

SWS Item	**ECUC_CanNm_00010 :**		
Name	CanNmPassiveModeEnabled {CANNM_PASSIVE_MODE_ENABLED}		
Description	Pre-processor switch for enabling support of the Passive Mode.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU dependency: calculationFormula = Equal(NmPassiveModeEnabled)		

SWS Item	**ECUC_CanNm_00011 :**		
Name	CanNmPduRxIndicationEnabled {CANNM_PDU_RX_INDICATION_ENABLED}		
Description	Pre-processor switch for enabling the PDU Rx Indication.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU dependency: calculationFormula = Equal(NmPduRxIndicationEnabled)		

SWS Item	**ECUC_CanNm_00070 :**		
Name	CanNmPnEiraCalcEnabled {CANNM_PN_EIRA_CALC_ENABLED}		
Description	Specifies if CanNm calculates the PN request information for internal an external requests. (EIRA) true: PN request are calculated false: PN request are not calculated		
Multiplicity	0..1		
Type	EcucBooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local dependency: only available if CanNmPnEnabled == true for at least one CanNm Channel		

SWS Item	**ECUC_CanNm_00059 :**		
Name	CanNmPnResetTime {CANNM_PN_RESET_TIME}		
Description	Specifies the runtime of the reset timer in seconds. This reset time is valid for the reset of PN requests in the EIRA and in the ERA. The value shall be the same for every channel. Thus it is a global config parameter.		
Multiplicity	0..1		
Type	EcucFloatParamDef		
Range	0.001 .. 65.535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-

			BUILD
	Post-build time	--	
Scope / Dependency	scope: local dependency: only available if CanNmPnEnabled == true for at least one CanNm Channel.		

SWS Item	**ECUC_CanNm_00055 :**		
Name	CanNmRemoteSleepIndEnabled {CANNM_REMOTE_SLEEP_IND_ENABLED}		
Description	Pre-processor switch for enabling remote sleep indication support. This feature is required for gateway nodes only.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local dependency: calculationFormula = If (CanNmPassiveModeEnabled == False) then Equal(NmRemoteSleepIndEnabled) else Equal(False)		

SWS Item	**ECUC_CanNm_00005 :**		
Name	CanNmRepeatMsgIndEnabled {CANNM_REPEAT_MSG_IND_ENABLED}		
Description	Enable/disable the notification that a RepeatMessageRequest bit has been received.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU dependency: calculationformula = Equal(NmRepeatMsgIndEnabled)		

SWS Item	**ECUC_CanNm_00012 :**		
Name	CanNmStateChangeIndEnabled {CANNM_STATE_CHANGE_IND_ENABLED}		
Description	Pre-processor switch for enabling the CAN NM state change notification.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU dependency: calculationFormula = Equal(NmStateChangeIdEnabled)		

SWS Item	**ECUC_CanNm_00004 :**		
Name	CanNmUserDataEnabled {CANNM_USER_DATA_ENABLED}		
Description	Pre-processor switch for enabling user data support.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	

Scope / Dependency	scope: ECU dependency: calculationFormula = Equal(NmUserDataEnabled)

SWS Item	**ECUC_CanNm_00003 :**		
Name	CanNmVersionInfoApi {CANNM_VERSION_INFO_API}		
Description	Pre-processor switch for enabling version info API support.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_CanNm_00072 :**		
Name	CanNmPnEiraRxNSduRef {CANNM_PN_EIRA_RX_NSDU_REF}		
Description	Reference to a Pdu in the COM-Stack. Only one SduRef is required for CanNm because the EIRA is the aggregation over all Can Channels.		
Multiplicity	0..1		
Type	Reference to [Pdu]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: local dependency: only available if CanNmPnEnabled == true for at least one CanNm Channel.		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CanNmChannelConfig	1..*	This container contains the channel specific configuration parameter of the CanNm.
CanNmPnInfo	0..1	PN information configuration

10.3.3 CanNm Channel Configuration Overview

[SWS_CanNm_00202] ⌈The container CanNmChannelConfig specifies configuration parameter that shall be located in a data structure of type `CanNm_ConfigType`.⌋()

[SWS_CanNm_00203] ⌈Runtime configurable parameters listed below shall be configurable for each network management cluster separately.⌋()

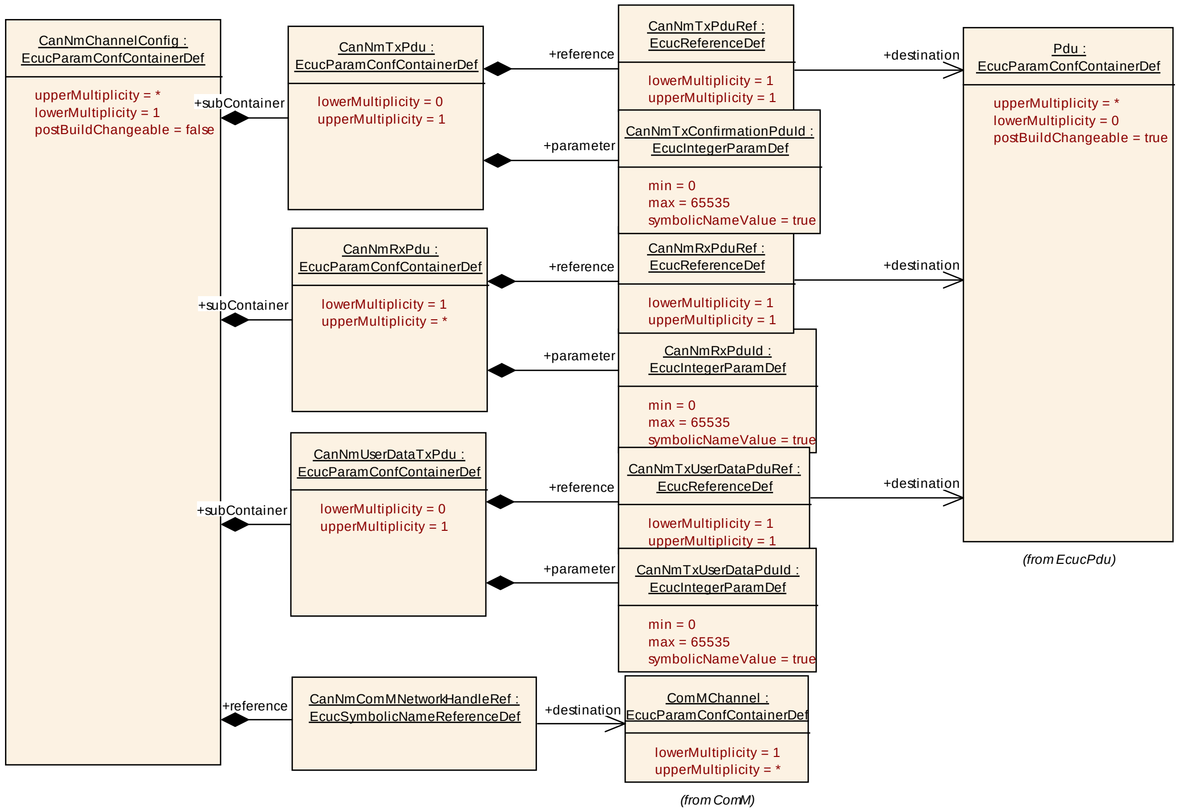


Figure 10-3 CanNm Channel Configuration Overview

10.3.4 CanNmChannelConfig

SWS Item	**ECUC_CanNm_00017 :**
Container Name	CanNmChannelConfig{CanNm_ChannelConfig}
Description	This container contains the channel specific configuration parameter of the CanNm. **Attributes:** postBuildChangeable=false
Configuration Parameters	

SWS Item	**ECUC_CanNm_00084 :**		
Name	CanNmActiveWakeupBitEnabled {CANNM_ACTIVE_WAKEUP_BIT_ENABLED}		
Description	Enables/Disables the handling of the Active Wakeup Bit in the CanNm module.		
Multiplicity	0..1		
Type	EcucBooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: local dependency: This parameter is only valid if CanNmPassiveModeEnabled is False.		

SWS Item	**ECUC_CanNm_00068 :**
Name	CanNmAllNmMessagesKeepAwake

Description	Specifies if CanNm drops irrelevant NM PDUs. false: Only NM PDUs with a PNI bit = true and containing a PN request for this ECU triggers the standard RX indication handling true: Every NM PDU triggers the standard RX indication handling		
Multiplicity	0..1		
Type	EcucBooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: local dependency: only available if CanNmPnEnabled == true		

SWS Item	**ECUC_CanNm_00042 :**		
Name	CanNmBusLoadReductionActive {CANNM_BUS_LOAD_REDUCTION_ACTIVE}		
Description	This parameter defines if bus load reduction for the respective NM channel is active or not.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: local dependency: Shall only be True if CanNmBusLoadReductionEnabled is True.		

SWS Item	**ECUC_CanNm_00075 :**		
Name	CanNmCarWakeUpBitPosition {CANNM_CAR_WAKE_UP_BIT_POSITION}		
Description	Specifies the Bit position of the CWU within the NM PDU.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 7		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: local dependency: only available if CanNmCarWakeUpRxEnabled == TRUE		

SWS Item	**ECUC_CanNm_00076 :**		
Name	CanNmCarWakeUpBytePosition {CANNM_CAR_WAKE_UP_BYTE_POSITION}		
Description	Specifies the Byte position of the CWU within the NM PDU.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	2 .. 7		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	

Scope / Dependency	scope: local dependency: only available if CanNmCarWakeUpRxEnabled == TRUE		

SWS Item	**ECUC_CanNm_00077 :**		
Name	CanNmCarWakeUpFilterEnabled {CANNM_CAR_WAKE_UP_FILTER_ENABLED}		
Description	If CWU filtering is supported, only the CWU bit within the NM PDU with source node identifier CanNmCarWakeUpFilterNodeId is considered as CWU request. FALSE - CWU filtering is not supported TRUE - CWU filtering is supported		
Multiplicity	0..1		
Type	EcucBooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: local dependency: only available if CanNmCarWakeUpRxEnabled == TRUE		

SWS Item	**ECUC_CanNm_00078 :**		
Name	CanNmCarWakeUpFilterNodeId {CANNM_CAR_WAKE_UP_FILTER_NODE_ID}		
Description	Source node identifier for CWU filtering. If CWU filtering is supported, only the CWU bit within the NM PDU with source node identifier CanNmCarWakeUpFilterNodeId is considered as CWU request.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: local dependency: only available if CanNmCarWakeUpRxEnabled == TRUE		

SWS Item	**ECUC_CanNm_00074 :**		
Name	CanNmCarWakeUpRxEnabled {CANNM_CAR_WAKE_UP_RX_ENABLED}		
Description	Enables or disables support of CarWakeUp bit evaluation in received NM PDUs. FALSE - CarWakeUp not supported TRUE - CarWakeUp supported		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_CanNm_00057 :**
Name	CanNmImmediateNmCycleTime {CANNM_IMMEDIATE_NM_CYCLETIME}
Description	Defines the immediate NM PDU cycle time in seconds which is used for CanNmImmediateNmTransmissions NM PDU transmissions.

Multiplicity	0..1		
Type	EcucFloatParamDef		
Range	0.001 .. 65.535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: local dependency: This parameter is only valid if CanNmImmediateNmTransmissions is greater one.		

SWS Item	**ECUC_CanNm_00056 :**		
Name	CanNmImmediateNmTransmissions {CANNM_IMMEDIATE_NM_TRANSMISSIONS}		
Description	Defines the number of immediate NM PDUs which shall be transmitted. If the value is zero no immediate NM PDUs are transmitted. The cycle time of immeditate NM PDUs is defined by CanNmImmediateNmCycleTime.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: local dependency: If CanNmImmediateRestartEnabled = true then CanNmImmediateNmTransmissions = 0		

SWS Item	**ECUC_CanNm_00029 :**		
Name	CanNmMsgCycleOffset {CANNM_MSG_CYCLE_OFFSET}		
Description	Time offset in the periodic transmission node. It determines the start delay of the transmission. Specified in seconds.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	0 .. 65.535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: local dependency: Parameter value < CanMsgCycleTime This parameter is only valid if CanNmPassiveModeEnabled is False.		

SWS Item	**ECUC_CanNm_00028 :**		
Name	CanNmMsgCycleTime {CANNM_MSG_CYCLE_TIME}		
Description	Period of a NM PDU in seconds. It determines the periodic rate in the "periodic transmission mode with bus load reduction" and is the basis for transmit scheduling in the "periodic transmission mode without bus load reduction".		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	0.001 .. 65.535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE

	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: ECU dependency: This parameter is only valid if CanNmPassiveModeEnabled is False.		

SWS Item	**ECUC_CanNm_00043 :**		
Name	CanNmMsgReducedTime {CANNM_MSG_REDUCED_TIME}		
Description	Node specific bus cycle time in the periodic transmission mode with bus load reduction. Specified in seconds.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	0.001 .. 65.535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: local dependency: 0,5 * CanNmMsgCycleTime ≤ CanNmMsgReducedTime < CanNmMsgCycleTime This parameter is only valid if CanNmBusLoadReductionEnabled == True and CanNmBusLoadReductionActive == True and CanNmPassiveModeEnabled == False Otherwise this parameter is notused.		

SWS Item	**ECUC_CanNm_00030 :**		
Name	CanNmMsgTimeoutTime {CANNM_MSG_TIMEOUT_TIME}		
Description	Transmission Timeout of NM PDU. If there is no transmission confirmation by the CAN Interface within this timeout, the CANNM module shall give an error notification.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	0.001 .. 65.535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: local dependency: CanNmMsgTimeoutTime < CanNmMsgCycleTime This parameter is only valid if CANNM_PASSIVE_MODE_ENABLED is disabled.		

SWS Item	**ECUC_CanNm_00031 :**		
Name	CanNmNodeId {CANNM_NODE_ID}		
Description	Node identifier of local node.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE

	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: local dependency: This parameter is only valid if CanNmNodeIdEnabled == True		

SWS Item	**ECUC_CanNm_00026 :**		
Name	CanNmPduCbvPosition {CANNM_PDU_CBV_POSITION}		
Description	Defines the position of the control bit vector within the NM PDU. The value of the parameter represents the location of the control bit vector in the NM PDU (CanNmPduByte0 means byte 0, CanNmPduByte1 means byte 1, CanNmPduOff means source node identifier is not part of the NM PDU) ImplementationType: CanNm_PduPositionType		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	CANNM_PDU_BYTE_0	Byte 0 is used	
	CANNM_PDU_BYTE_1	Byte 1 is used	
	CANNM_PDU_OFF	Control Bit Vector is not used	
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	

Scope / Dependency	scope: ECU
	dependency: CanNmPduNidPosition; If CanNmNodeDetectionEnabled == true then CanNmPduCbvPosition != CANNM_PDU_OFF
	
	if(CANNM_PDU_CBV_POSITION != CANNM_PDU_OFF && CANNM_PDU_NID_POSITION != CANNM_PDU_OFF) then CANNM_PDU_CBV_POSITION != CANNM_PDU_NID_POSITION
	
	if(CANNM_PDU_CBV_POSITION != CANNM_PDU_OFF && CANNM_PDU_NID_POSITION == CANNM_PDU_OFF) then CANNM_PDU_CBV_POSITION = CANNM_PDU_BYTE0		

SWS Item	**ECUC_CanNm_00025 :**		
Name	CanNmPduNidPosition {CANNM_PDU_NID_POSITION}		
Description	Defines the position of the source node identifier within the NM PDU. The value of the parameter represents the location of the source node identifier in the NM PDU (CanNMPduByte0 means byte 0, CanNmPduByte1 means byte 1, CanNmPduOff means source node identifier is not part of the NM PDU) ImplementationType: CanNm_PduPositionType		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	CANNM_PDU_BYTE_0	Byte 0 is used	
	CANNM_PDU_BYTE_1	Byte 1 is used	
	CANNM_PDU_OFF	Node Identification is not used	
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: ECU dependency: CanNmPduCbvPosition; If CanNmNodeIdEnabled == true then CanNmPduNidPosition != CANNM_PDU_OFF if(CANNM_PDU_NID_POSITION != CANNM_PDU_OFF && CANNM_PDU_CBV_POSITION != CANNM_PDU_OFF) then CANNM_PDU_NID_POSITION != CANNM_PDU_CBV_POSITION		

	if(CANNM_PDU_NID_POSITION != CANNM_PDU_OFF &&
	CANNM_PDU_CBV_POSITION == CANNM_PDU_OFF) then
	CANNM_PDU_NID_POSITION = CANNM_PDU_BYTE0

SWS Item	**ECUC_CanNm_00066 :**		
Name	CanNmPnEnabled {CANNM_PN_ENABLED}		
Description	Enables or disables support of partial networking. false: Partial networking Range not supported true: Partial networking supported		
Multiplicity	0..1		
Type	EcucBooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU dependency: If CanNmPnEnabled == TRUE then CanNmComUserDataSupport = TRUE		

SWS Item	**ECUC_CanNm_00067 :**		
Name	CanNmPnEraCalcEnabled {CANNM_PN_ERA_CALC_ENABLED}		
Description	Specifies if CanNm calculates the PN request information for external requests. (ERA) false: PN request are not calculated true: PN request are calculated		
Multiplicity	0..1		
Type	EcucBooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: local dependency: only available if CanNmPnEnabled == true		

SWS Item	**ECUC_CanNm_00073 :**		
Name	CanNmPnHandleMultipleNetworkRequests {CANNM_PN_HANDLE_MULTIPLE_NETWORK_REQUESTS}		
Description	Specifies if CanNm performs an additional transition from Network Mode to Repeat Message State (true) or not (false).		
Multiplicity	0..1		
Type	EcucBooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: local dependency: only available if CanNmPnEnabled == true		

SWS Item	**ECUC_CanNm_00023 :**
Name	CanNmRemoteSleepIndTime {CANNM_REMOTE_SLEEP_IND_TIME}
Description	Timeout for Remote Sleep Indication. It defines the time in seconds how long it shall take to recognize that all other nodes are ready to sleep.
Multiplicity	1
Type	EcucFloatParamDef

Range	0 .. 65.535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: local dependency: CanNmRemoteSleepIndTime ≥ CanNmMsgCycleTime		

SWS Item	**ECUC_CanNm_00022 :**		
Name	CanNmRepeatMessageTime {CANNM_REPEAT_MESSAGE_TIME}		
Description	Timeout for Repeat Message State. It defines the time in seconds how long the NM shall stay in the Repeat Message State.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	0 .. 65.535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: local dependency: CanNmRepeatMessageTime = n * CanNmMsgCycleTime; CanNmRepeatMessageTime > CanNmImmediateNmTransmissions * CanNmImmediateNmCycleTime Typically it should be equal to: n * CanNmMsgCycleTime, where n denotes the number of NM PDUs that are normally sent in the Repeat Message State. The value of n decremented by one determines the amount of lost NM PDUs that can be tolerated by the node detection procedure. The value 0 denotes that no Repeat Message State is configured. It means that Repeat Message State is transient what implicates that it is left immediately after entrance and in result no start-up stability is guaranteed and no node detection procedure is possible.		

SWS Item	**ECUC_CanNm_00020 :**		
Name	CanNmTimeoutTime {CANNM_TIMEOUT_TIME}		
Description	Network Timeout for NM PDUs. It denotes the time in seconds how long the NM shall stay in the Ready Sleep State before transition into the Prepare Bus-Sleep Mode is initiated.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	0.002 .. 65.535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: local dependency: CanNmTimeoutTime > CanNmMsgCycleTime It shall be equal for all nodes in the cluster. It shall be greater than CanNmMsgCycleTime.		

SWS Item	**ECUC_CanNm_00027 : (Obsolete)**

Name	CanNmUserDataLength {CANNM_USER_DATA_LENGTH}		
Description	Defines the length of the user data contained in the NM PDU Please note that this parameter is deprecated and will be removed in future. **Tags:** atp.Status=obsolete atp.StatusRevisionBegin=4.1.1		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 8		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: ECU dependency: CanNmPduLength, CanNmPduNidPosition, CanNmPduCbvPosition		

SWS Item	**ECUC_CanNm_00021 :**		
Name	CanNmWaitBusSleepTime {CANNM_WAIT_BUS_SLEEP_TIME}		
Description	Timeout for bus calm down phase. It denotes the time in seconds how long the NM shall stay in the Prepare Bus-Sleep Mode before transition into Bus-Sleep Mode shall take place.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	0.001 .. 65.535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: local dependency: It shall be equal for all nodes in the cluster. It shall be long enough to make all Tx-buffer empty.		

SWS Item	**ECUC_CanNm_00018 :**		
Name	CanNmComMNetworkHandleRef {CANNM_CHANNEL_ID}		
Description	This reference points to the unique channel defined by the ComMChannel and provides access to the unique channel index value in ComMChannelId.		
Multiplicity	1		
Type	Symbolic name reference to [ComMChannel]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_CanNm_00079 :**		
Name	CanNmPnEraRxNSduRef {CANNM_PN_ERA_RX_NSDU_REF}		
Description	Reference to a Pdu in the COM-Stack. The SduRef is required for every CanNm Channel, because ERA is reported per channel.		
Multiplicity	0..1		
Type	Reference to [Pdu]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-

			BUILD
	Post-build time	--	
Scope / Dependency	scope: local dependency: only available if CanNmPnEnabled == true for at least one CanNm Channel.		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CanNmRxPdu	1..*	This container is used to configure the Rx PDU properties that are used for the CanNm Channel.
CanNmTxPdu	0..1	This container contains the CanNmTxConfirmationPduId and the CanNmTxPduRef.
CanNmUserDataTxPdu	0..1	This optional container is used to configure the UserNm PDU. This container is only available if CanNmComUserDataSupport is enabled.

10.3.5 CanNmRxPdu

SWS Item	**ECUC_CanNm_00038 :**
Container Name	CanNmRxPdu
Description	This container is used to configure the Rx PDU properties that are used for the CanNm Channel.
Configuration Parameters	

SWS Item	**ECUC_CanNm_00054 :**		
Name	CanNmRxPduId {CANNM_RX_PDU_ID}		
Description	This parameter defines the Rx PDU ID of the CanIf L-PDU range that is associated with this CanNm channel.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_CanNm_00039 :**		
Name	CanNmRxPduRef		
Description	Reference to the global PDU that is used by this CanNm channel.		
Multiplicity	1		
Type	Reference to [Pdu]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers

10.3.6 CanNmTxPdu

SWS Item	**ECUC_CanNm_00036 :**
Container Name	CanNmTxPdu
Description	This container contains the CanNmTxConfirmationPduId and the CanNmTxPduRef.

Configuration Parameters

SWS Item	**ECUC_CanNm_00048 :**		
Name	CanNmTxConfirmationPduId {CANNM_TX_CONFIRMATION_PDU_ID}		
Description	Handle Id to be used by the Lower Layer to confirm the transmission of the CanNmTxPdu to the LowerLayer.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_CanNm_00037 :**		
Name	CanNmTxPduRef		
Description	The reference to the common PDU structure.		
Multiplicity	1		
Type	Reference to [Pdu]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers

10.3.7 CanNmUserDataTxPdu

SWS Item	**ECUC_CanNm_00045 :**
Container Name	CanNmUserDataTxPdu
Description	This optional container is used to configure the UserNm PDU. This container is only available if CanNmComUserDataSupport is enabled.
Configuration Parameters	

SWS Item	**ECUC_CanNm_00047 :**		
Name	CanNmTxUserDataPduId {CANNM_TX_USER_DATA_PDU_ID}		
Description	This parameter defines the Handle ID of the NM User Data I-PDU.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_CanNm_00046 :**		
Name	CanNmTxUserDataPduRef		
Description	Reference to the NM User Data I-PDU in the global PDU collection.		
Multiplicity	1		
Type	Reference to [Pdu]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE

	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers

10.3.8 CanNmPnInfo

SWS Item	**ECUC_CanNm_00071 :**
Container Name	CanNmPnInfo
Description	PN information configuration
Configuration Parameters	

SWS Item	**ECUC_CanNm_00061 :**		
Name	CanNmPnInfoLength		
Description	Specifies the length of the PN request information in the NM PDU.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 7		
Default value	1		
ConfigurationClass	*Pre-compile time*	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: local dependency: only available if CanNmPnEnabled == true for at least one CanNm Channel.		

SWS Item	**ECUC_CanNm_00060 :**		
Name	CanNmPnInfoOffset		
Description	Specifies the offset of the PN request information in the NM PDU.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 7		
Default value	1		
ConfigurationClass	*Pre-compile time*	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: local dependency: only available if CanNmPnEnabled == true for at least one CanNm Channel.		

Included Containers		
Container Name	*Multiplicity*	*Scope / Dependency*
CanNmPnFilterMaskByte	0..7	PN information configuration

10.3.9 CanNmPnFilterMaskByte

SWS Item	**ECUC_CanNm_00069 :**
Container Name	CanNmPnFilterMaskByte
Description	PN information configuration
Configuration Parameters	

SWS Item	**ECUC_CanNm_00063 :**

Name	CanNmPnFilterMaskByteIndex		
Description	Index of the filter mask byte. Specifies the position within the filter mask byte array.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 6		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: local dependency: only available if CanNmPnEnabled == true for at least one CanNm Channel; CanNmPnFilterMaskByteIndex < CanNmPnInfoLength		

SWS Item	**ECUC_CanNm_00064 :**		
Name	CanNmPnFilterMaskByteValue		
Description	Parameter to configure the filter mask byte.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	0		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local dependency: only available if CanNmPnEnabled == true for at least one CanNm Channel.		

No Included Containers

10.4 Published parameters

For details refer to the chapter 10.3 “Published Information” in *SWS_BSWGeneral.*

11 Examples

11.1 Example of periodic transmission mode with bus load reduction

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

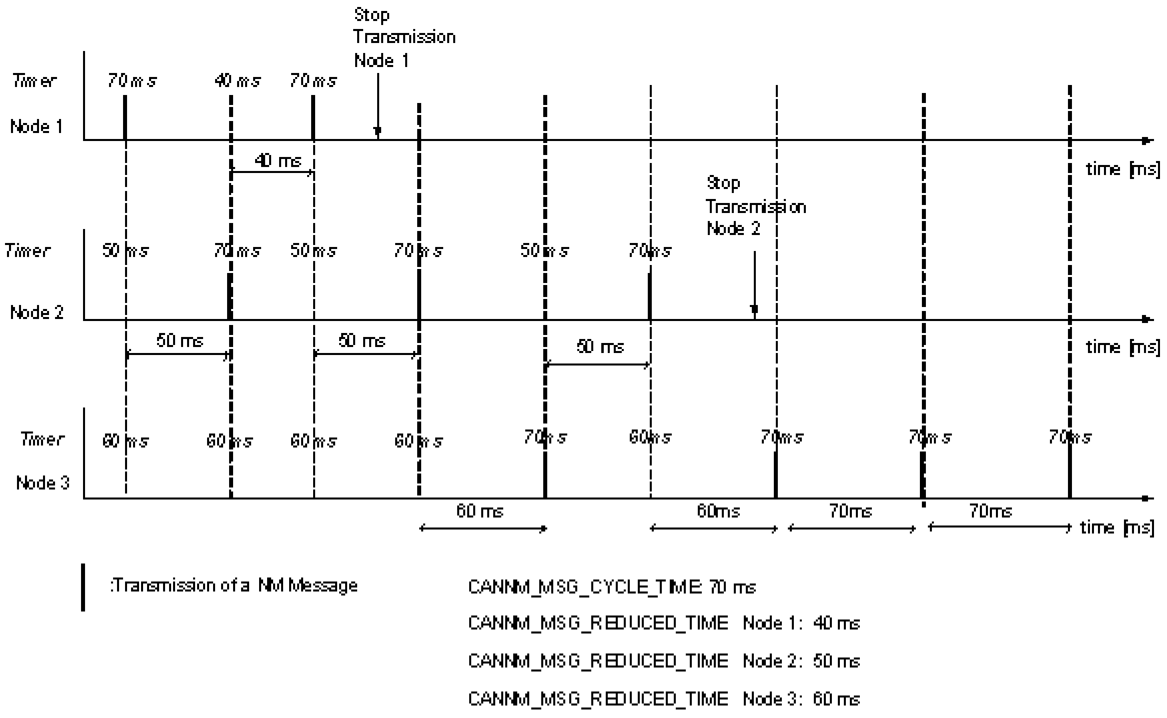


Figure 11-1Example for Bus Load Reduction

11.2Example timing behavior for Network Management PDUs

Assume an example network of three nodes 1, 2, 3 (see also Figure 11-2). Nodes specific cycle offsets are equal respectively to t1 < t2 < t3 < T. NM cycle time is equal to T (see Figure 11-3).

Network Management PDUs sent on the bus within the Repeat Message State are presented in the Figure 11-4, and within the Normal Operation / Ready Sleep State in Figure 11-5. Each dot in Figure 11-5 denotes restart of the NM-Timeout Timer.

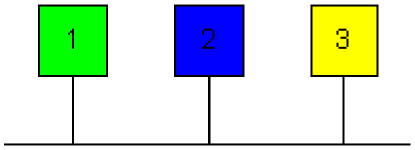


Figure 11-2 Example for 3 ECUs connected to a network

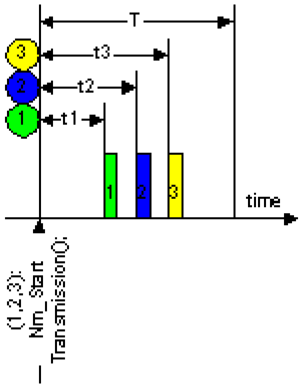


Figure 11-3 Example for NM Transmission Start of different ECUs

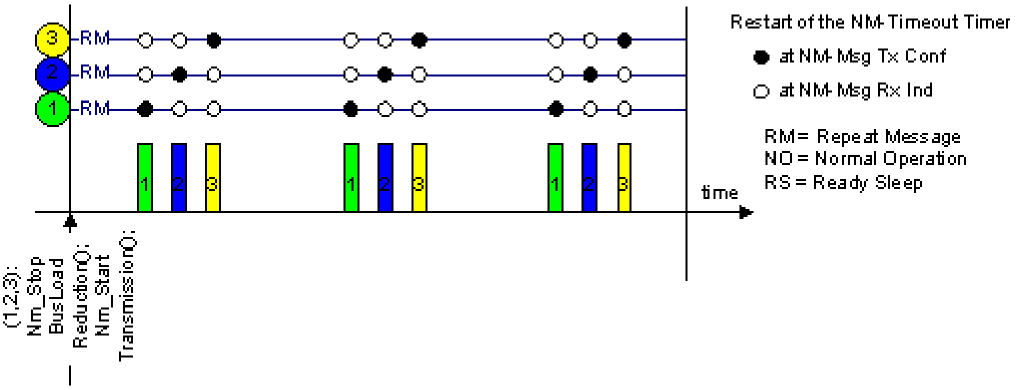


Figure 11-4 Example for NM Transmission Handling of multiple ECUs

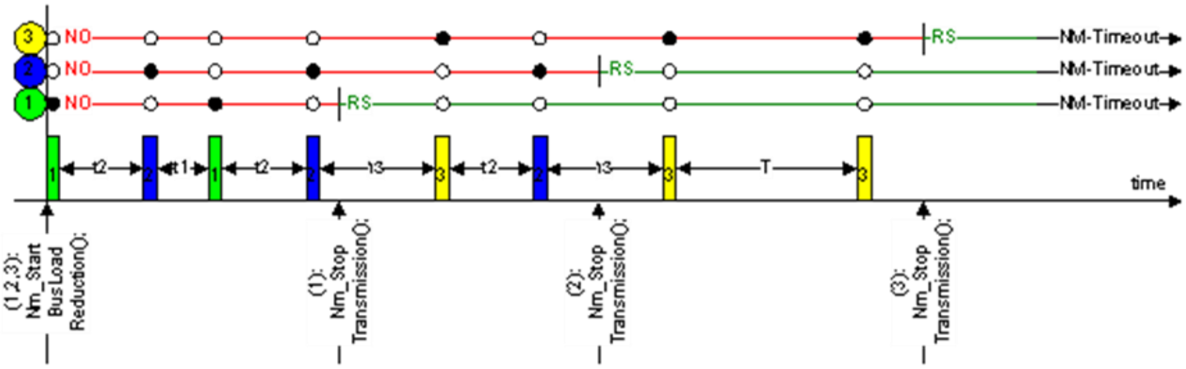


Figure 11-5 Example for NM Timeout Handling

12 Not applicable requirements

[SWS_CanNm_00999]「These requirements are not applicable to this specification.

」(SRS_BSW_00170, SRS_BSW_00387, SRS_BSW_00375, SRS_BSW_00416, SRS_BSW_00168, SRS_BSW_00423, SRS_BSW_00424, SRS_BSW_00425, SRS_BSW_00426, SRS_BSW_00427, SRS_BSW_00429, SRS_BSW_00432, BSW00434, SRS_BSW_00336, SRS_BSW_00417, SRS_BSW_00161, SRS_BSW_00162, SRS_BSW_00005, SRS_BSW_00415, SRS_BSW_00164, SRS_BSW_00325, SRS_BSW_00326, SRS_BSW_00160, SRS_BSW_00413, SRS_BSW_00347, SRS_BSW_00305, SRS_BSW_00307, SRS_BSW_00335, SRS_BSW_00410, SRS_BSW_00314, SRS_BSW_00328, SRS_BSW_00312, SRS_BSW_00006, SRS_BSW_00377, SRS_BSW_00306, SRS_BSW_00309, SRS_BSW_00330, SRS_BSW_00331, SRS_BSW_00172, SRS_BSW_00010, SRS_BSW_00333, SRS_BSW_00321, SRS_BSW_00341, SRS_BSW_00334, SRS_Nm_00151, SRS_Nm_00043, SRS_Nm_00046, SRS_Nm_00048, SRS_Nm_00050, SRS_Nm_00052, SRS_Nm_02509, SRS_Nm_00153, BSW136, BSW140, SRS_Nm_00054, SRS_Nm_00144, SRS_Nm_00147, SRS_Nm_00154, BSW139)