MICROSAR LIN Network Management

Technical Reference

Nm_AsrNmLin

Version 2.01.00

Authors	Lutz Pflüger, Bastian Molkenthin
Status	Released

Document Information

History

Author	Date	Version	Remarks
Lutz Pflüger	2013-06-19	1.00.00	Initial version
Lutz Pflüger	2013-10-30	2.00.00	Change Workaround, R8 release
Bastian Molkenthin	2015-06-16	2.01.00	Added critical sections

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_SWS_LINNetworkManagement.pdf	2.0.0
[2]	AUTOSAR	AUTOSAR_SWS_DevelopmentErrorTracer.pdf	3.2.0
[3]	AUTOSAR	AUTOSAR_SWS_DiagnosticEventManager.pdf	4.2.0
[4]	AUTOSAR	AUTOSAR_TR_BSWModuleList.pdf	1.6.0
[5]	AUTOSAR	AUTOSAR_SWS_RTE.pdf	3.2.0

Scope of the Document

This technical reference describes the general use of the LIN Network Management basic software. All aspects which are LIN controller specific are described in a separate document, which is also part of the delivery.

Caution
We have configured the programs in accordance with your specifications in the questionnaire. Whereas the programs do support other configurations than the one specified in your questionnaire, Vector's release of the programs delivered to your company is expressly restricted to the configuration you have specified in the questionnaire.

Contents

Illustrations

Tables

1 Component History

The component history gives an overview over the important milestones that are supported in the different versions of the component.

Component Version	New Features
1.00.00	Creation of component

Table 1-1 Component history

2 Introduction

This document describes the functionality, API and configuration of the AUTOSAR BSW module LINNM as specified in [1]. Also the integration of the LIN Network Management into the AUTOSAR stack is covered by this document.

The integration of the FlexRay Network Management, CAN Network Management and the UDP Network Management are not covered by this document.

Please note that in this document the term Application is not used strictly for the user software but also for any higher software layer, like e.g. the AUTOSAR Network Management Interface. Therefore, Application refers to any of the software components using the LIN NM.

For further information please also refer to the AUTOSAR SWS specifications, referenced at the beginning of this document in Table: ‘Reference Documents’.

Supported AUTOSAR Release*:	4	
Supported Configuration Variants:	pre-compile	
Vendor ID:	LINNM_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	LINNM_MODULE_ID	63 decimal (according to ref. [4])

* For the precise AUTOSAR Release 4.x please see the release specific documentation.

2.1 Naming Conventions

The names of the service function provided by the LIN NM always start with a prefix that denominates the software component where the service is located. E.g. a service that starts with ‘`LinNm_`’ is implemented within the LIN NM.

Naming conventions	
Nm_	Services of NM Interface
LinNm_	Services of LIN NM
Det_	Services of Development Error Tracer

Table 2-1 Naming Conventions

2.2 Architecture Overview

The following figure shows where the LINNM is located in the AUTOSAR architecture.

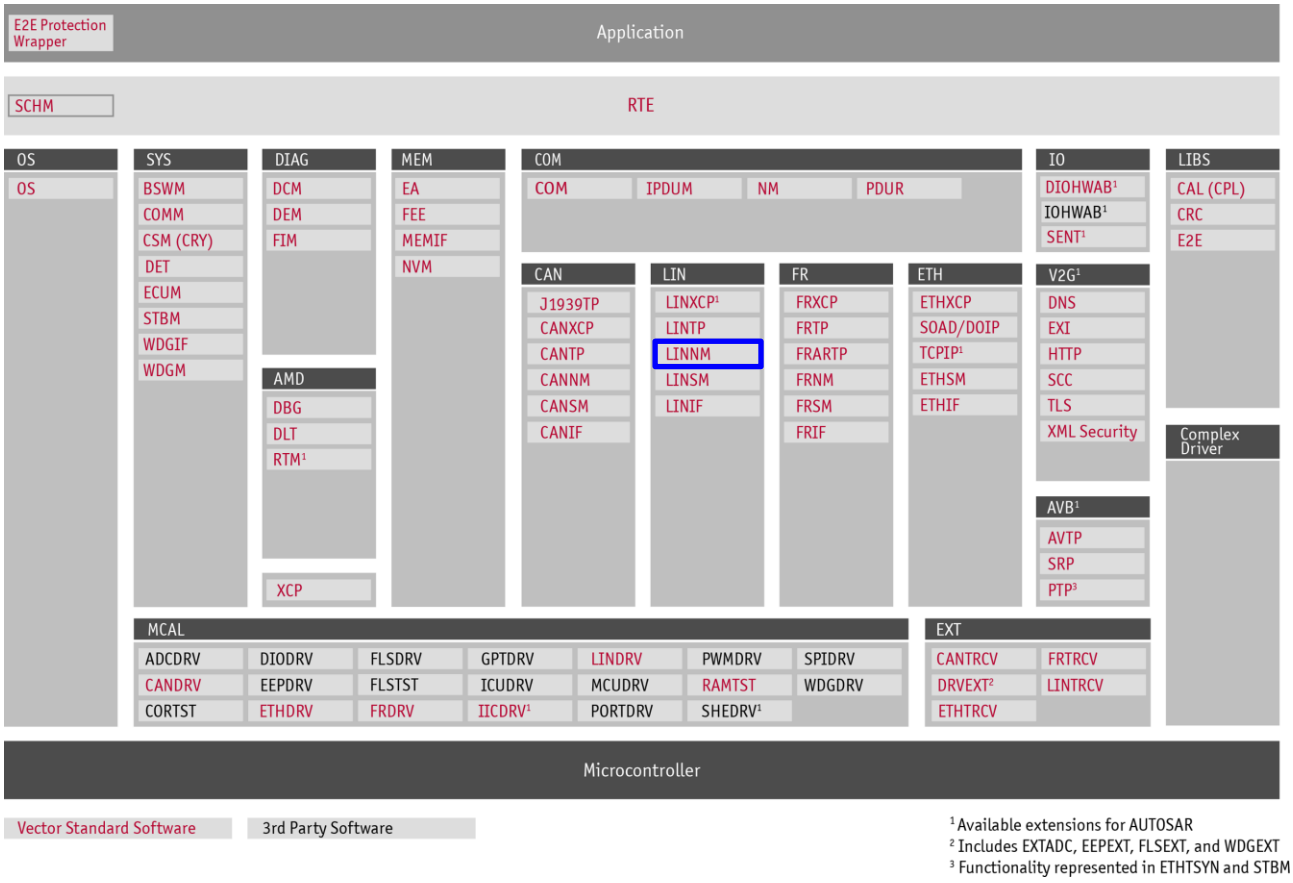


Figure 2-1 AUTOSAR 4.x Architecture Overview

The next figure shows the interfaces to adjacent modules of the LINNM. These interfaces are described in chapter 4.3.

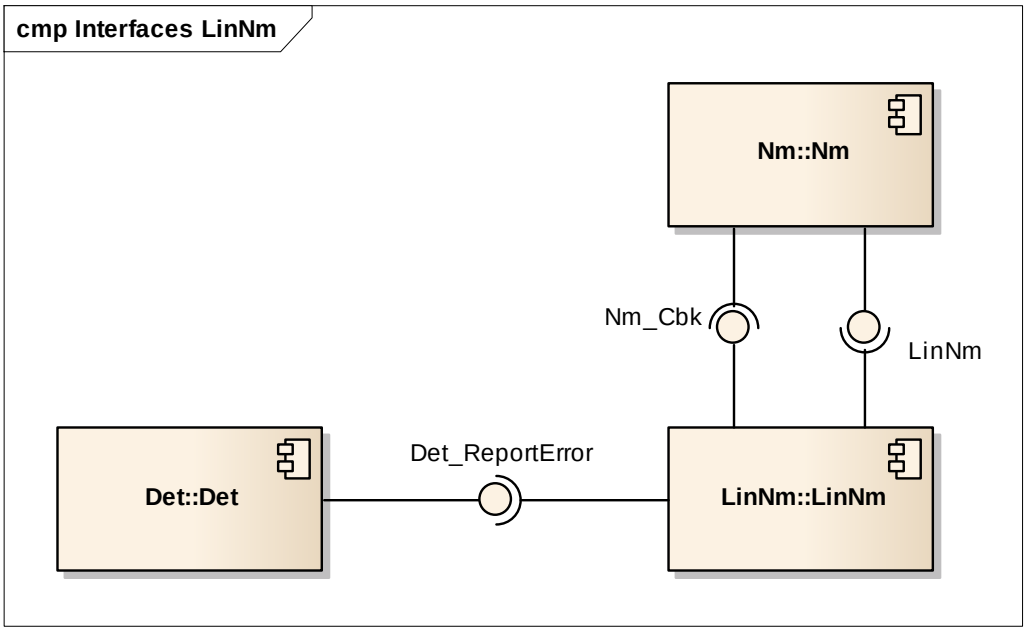


Figure 2-2 Interfaces to adjacent modules of the LINNM

3 Functional Description

3.1 Features

The LIN Network Management coordination algorithm is based on a basic state machine. No special network management messages (like CAN, FlexRay or UDP) are necessary on LIN because it is a master/slave network. Therefore the coordination algorithm is very simple and no interface to LinIf is needed.

The features listed in the following tables cover the complete functionality specified for the LINNM.

The AUTOSAR standard functionality is specified in [1], the corresponding features are listed in the tables

> Table 3-1 Supported AUTOSAR standard conform features

> Table 3-2 Not supported AUTOSAR standard conform features

Vector Informatik provides further LINNM functionality beyond the AUTOSAR standard. The corresponding features are listed in the table

> Table 3-3 Features provided beyond the AUTOSAR standard

The following features specified in [1] are supported:

Supported AUTOSAR Standard Conform Features
Basic state machine

Table 3-1 Supported AUTOSAR standard conform features

3.1.1 Deviations against AUTOSAR 4.0.3

The following features specified in [1] are not supported:

Not Supported AUTOSAR Standard Conform Features
Provision of LinNm_Cfg.c

Table 3-2 Not supported AUTOSAR standard conform features

Other deviations are provided as follows.

3.1.1.1 RAM Initialization

If RAM is not implicitly initialized at start-up, the function `LinNm_InitMemory` has to be called.

3.1.2 Additions/ Extensions

The following features are provided beyond the AUTOSAR standard:

Features Provided Beyond The AUTOSAR Standard
Memory Initialization

Table 3-3 Features provided beyond the AUTOSAR standard

3.1.2.1 Memory Initialization

AUTOSAR expects the start-up code to automatically initialize RAM. Not every startup code of embedded targets reinitializes all variables correctly it is possible that the state of a variable may not be initialized, as expected. To avoid this problem the Vector AUTOSAR NM provides additional functions to initialize the relevant variables of the LIN NM.

Refer also to chapter 5.4.1 ‘LinNm_InitMemory’.

3.2 Network Management Mechanism

As described above the AUTOSAR LINNM is based on a basic state machine.

The following figure shows the state diagram of the LIN NM. The events are calls of LIN NM functions by the application.

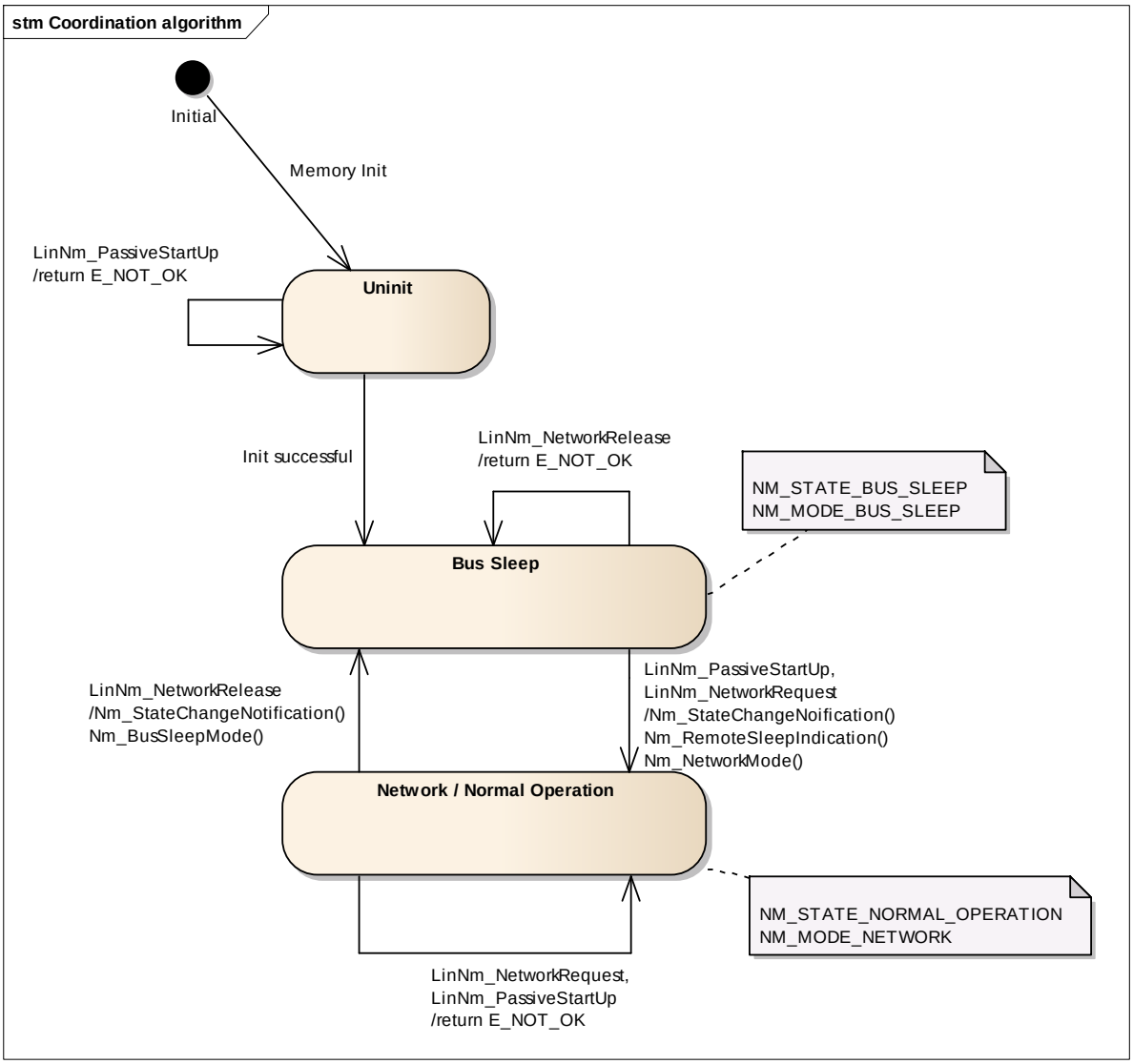


Figure 3-1 Basic state machine of LIN NM

Note
No network messages are send over LIN and therefore no state transition are triggered on occurrence of NM messages on the basic state machine. The basic state machine is only triggered from the NM.

Caution

- If the Passive Mode (`LINNM_PASSIVE_MODE_ENABLED`) enabled, the network release function `LinNm_NetworkRelease` does not exist and there is no possibility to leave the “Network/Normal Operation” state.
- The same behavior happens if `LinNm_NetworkRelease` is never called, e.g. if ComM calls `Nm_PassiveStartUp` after EcuM indicated a Passive Wake-up.

This can avoid the ECU from sleep! To prevent this, see Workaround.

Workaround:

Never use LINNM in Passive Mode (`LINNM_PASSIVE_MODE_ENABLED`) and apply one of the following workarounds:

- Do not use LINNM, use NM Variant LIGHT or NONE in ComM channel configuration instead.

or if LINNM is used apply one of the following workarounds:

- Make sure that the LinNm channel is added to a NM Coordination Cluster in the Nm module.

otherwise

- Ensure that if NM Coordinator is not used for LIN NM that:
  - Synchronous Wake Up in ComM is disabled
  - a ComM user to the LIN channel in the ComM module is assigned
  - if communication is needed the Application must requests COMM_FULL_COMMUNICATION via ComM user
  - if no more communication is needed the Application must request COMM_NO_COMMUNICATION to ensure that ComM calls `Nm_NetworkRelease`
  - If a wake-up event on the LIN channel occurs, then COMM_FULL_COMMUNICATION needs to be requested for the corresponding ComM user. This can be accomplished for instance by using a BswM rule. This rule has to contain the condition of having the state of the wake-up source for the LIN channel being set to ECUM_WKSTATUS_VALIDATED. If this mode change happens, the ComM user shall request COMM_FULL_COMMUNICATION.

3.3 Initialization

Before the LIN NM can be used it has to be initialized by the application. The initialization has to be carried out before any other functionality of the LIN NM is executed. It shall take place prior to initialization of the NM Interface.

Also refer to chapter 5.4.1 ‘LinNm_InitMemory’.

Caution
The LIN NM assumes that some variables are initialized with zero at start-up. If the embedded target does not initialize RAM within the start-up code the function ‘`LinNm_InitMemory`’ has to be called during start-up and before the initialization is performed. Refer also to chapter 3.1.2.2 ‘Memory Initialization’.

3.4 Error Handling

3.4.1 Development Error Reporting

By default, development errors are reported to the DET using the service `Det_ReportError()` as specified in [2], if development error reporting is enabled (i.e. pre-compile parameter `LINNM_DEV_ERROR_DETECT==STD_ON`).

If another module is used for development error reporting, the function prototype for reporting the error can be configured by the integrator, but must have the same signature as the service `Det_ReportError()`.

The reported LINNM ID is 63.

The reported service IDs identify the services which are described in 5.4. The following table presents the service IDs and the related services:

Service ID	Service
0x01	LinNm_PassiveStartUp
0x02	LinNm_NetworkRequest
0x03	LinNm_NetworkRelease
0x0E	LinNm_GetState
0xF1	LinNm_GetVersionInfo

Table 3-4 Service IDs

The errors reported to DET are described in the following table:

Error Code		Description
`0x01`	`LINNM_E_NO_INIT`	API service used without module initialization.
`0x02`	`LINNM_E_INVALID_CHANNEL`	API service used with wrong channel handle.
`0x12`	`LINNM_E_PARAM_POINTER`	API service used with null pointer parameter.

Table 3-5 Errors reported to DET

3.4.1.1 Parameter Checking

AUTOSAR requires that API functions check the validity of their parameters. The checks in Table 3-6 are internal parameter checks of the API functions. These checks are for development error reporting. The Parameter `LINNM_DEV_ERROR_DETECT` dis-/ enables the call of Det_ReportError() for all checks globally.

The following table shows which parameter checks are performed on which services:

Service \ Check	LINNM_E_NO_INIT	LINNM_E_INVALID_CHANNEL	LINNM_E_PARAM_POINTER
LinNm_PassiveStartUp	■	■	
LinNm_NetworkRequest	■	■	
LinNm_NetworkRelease	■	■	
LinNm_GetState	■	■	■
LinNm_GetVersionInfo			■

Table 3-6 Development Error Reporting: Assignment of checks to services

3.4.2 Production Code Error Reporting

By default, production code related errors are reported to the DEM using the service `Dem_ReportErrorStatus()` as specified in [3], if production error reporting is enabled.

If another module is used for production code error reporting, the function prototype for reporting the error can be configured by the integrator, but must have the same signature as the service `Dem_ReportErrorStatus()`.

The errors reported to DEM are described in the following table:

Error Code	Description
N/A	Currently no DEM errors are specified

Table 3-7 Errors reported to DEM

4 Integration

This chapter gives necessary information for the integration of the MICROSAR LINNM into an application environment of an ECU.

4.1 Scope of Delivery

The delivery of the LINNM contains the files which are described in the chapters 4.1.1 and 4.1.2:

4.1.1 Static Files

File Name	Source Code Delivery	Object Code Delivery	Description
`LinNm.c`	■		Source code of LIN NM. The user **must not** change this file!
`LinNm.h`	■	■	API of LIN NM. The user **must not** change this file!
`LinNm_Cbk.h`	■	■	API of LIN NM callback functions. The user **must not** change this file!

Table 4-1 Static files

Do not edit manually
The static files listed above must not be edited by the user!

4.1.2 Dynamic Files

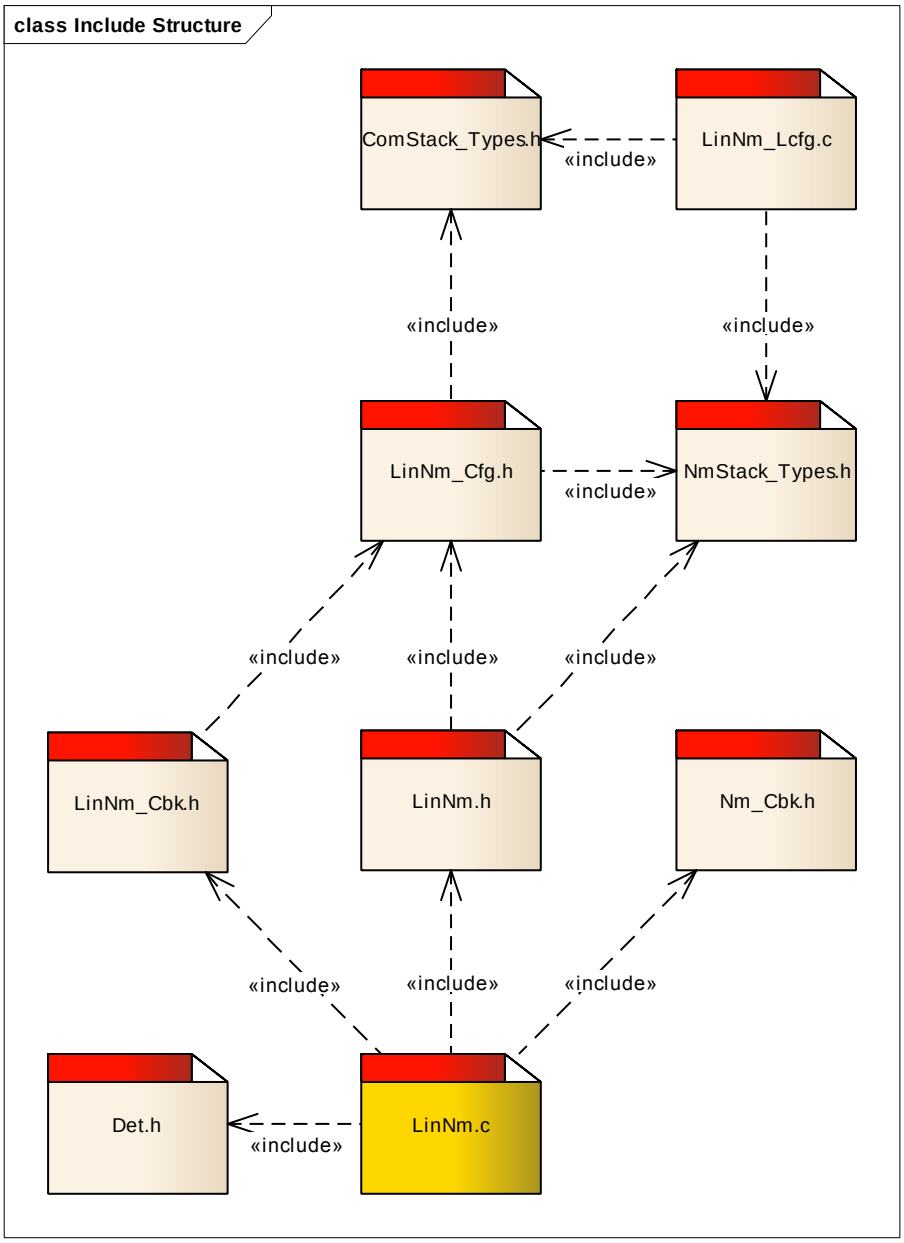
The dynamic files are generated by the configuration tool DaVinci Configurator.

File Name	Description
`LinNm_Cfg.h`	Configuration header file for LIN NM. The user **must not** change this file!
`LinNm_Lcfg.c`	Link-time variant Configuration source file. The user **must not** change this file!

Table 4-2 Generated files

Do not edit manually
The dynamic files listed above must not be edited by the user! They should be generated with the configuration tool to guarantee valid parameters.

4.2 Include Structure



4.3 Critical sections

Critical sections are handled by the BSW Scheduler (SchM), see [5]. They are automatically configured by the DaVinci Configurator. User interaction is only necessary by updating the internal behavior using the solving action in DaVinci Configurator. It is signaled as a warning in the validation tab.

The LINNM calls the following function when entering a critical section:

```
SchM_Enter_LinNm_LINNM_EXCLUSIVE_AREA_0()
```

When the critical section is left, the following function is called by LINNM:

```
SchM_Exit_LinNm_LINNM_EXCLUSIVE_AREA_0()
```

4.4 Critical section codes

To ensure data consistency, code sections inside an exclusive area must not be interrupted. Therefore the critical section code must lead to corresponding interrupt locks, as described below:

LINNM_EXCLUSIVE_AREA_0

> Must only lock interrupts if the API functions LinNm_PassiveStartup, LinNm_NetworkRequest, LinNm_NetworkRelease and LinNm_GetState can interrupt each other. This is e.g. not the case if all functions are called from the same task context.

5 API Description

For an interfaces overview please see Figure 2-2.

5.1 Data Types

The software module LIN NM uses the standard AUTOSAR data types that are defined within `Std_Types.h` and the platform specific data types that are defined within `Platform_Types.h` and the Communication Stack Types defined within `ComStack_Types.h`. Furthermore the standard AUTOSAR NM Stack Types defined within `NmStack_Types.h` are used.

5.2 Type Definitions

The types defined by the LINNM are described in this chapter.

Type Name	Type	Description
N/A	-	-

Table 5-1 Type definitions

5.3 Global Constants

5.3.1 AUTOSAR Specification Version

The version of AUTOSAR specification on which the appropriate implementation is based on is provided by three BCD coded defines:

Name	Type	Description
`LINNM_AR_RELEASE_MAJOR_VERSION`	BCD	Contains the major specification version number.
`LINNM_AR_RELEASE_MINOR_VERSION`	BCD	Contains the minor specification version number.
`LINNM_AR_RELEASE_REVISION_VERSION`	BCD	Contains the patch level specification version number.

Table 5-2 Specification Version API Data

5.3.2 Component Versions

The source code versions of LIN NM are provided by three BCD coded macros (and additionally as constants):

Name	Type	Description
`LINNM_SW_MAJOR_VERSION (LinNm_MainVersion)`	BCD	Contains the major component version number.
`LINNM_SW_MINOR_VERSION (LinNm_SubVersion)`	BCD	Contains the minor component version number.
`LINNM_SW_PATCH_VERSION (LinNm_ReleaseVersion)`	BCD	Contains the patch level component version number.

Table 5-3 Component Version API Data

These constants are declared as external and can be read by the application at any time.

5.3.3 Vendor and Module ID

LIN NM provides the vendor identifier according to AUTOSAR as defines:

Name	Type	Description	Value
LINNM_VENDOR_ID	-	Vendor ID according to AUTOSAR.	30
LINNM_MODULE_ID	-	Module ID according to AUTOSAR.	63

Table 5-4 Vendor/Module ID

5.4 Services provided by LINNM

5.4.1 LinNm_InitMemory

Prototype	
`void LinNm_InitMemory (void)`	
Parameter	
`void`	none
Return Code	
`void`	none
Functional Description	
Initialize Memory, so that expected start values are set	
Particularities and Limitations	
none	
Pre-Conditions	
Interrupts are disabled	
Call Context	
Called from Application	

Table 5-5 LinNm_InitMemory

5.4.2 LinNm_Init

Prototype	
`void LinNm_Init (void)`	
Parameter	
`void`	none
Return Code	
`void`	none
Functional Description	
Initialization of the LIN Network Management its internal state machine. By default the NM starts in the Bus-Sleep Mode.	
Particularities and Limitations	
Called by application (EcuM)	
Pre-Conditions	
> Interrupts must be disabled > Before call of any NM service (except LinNm_InitMemory())	
Call Context	
System Startup, non-reentrant	

Table 5-6 LinNm_Init

5.4.3 LinNm_PassiveStartUp

Prototype	
`Std_ReturnType` **`LinNm_PassiveStartUp`** `(const NetworkHandleType nmChannelHandle)`	
Parameter	
`nmChannelHandle`	Channel parameters
Return Code	
Std_ReturnType	E_OK - No error E_NOT_OK - Start of network management has failed
Functional Description	
Starts the NM from the Bus Sleep Mode and triggers transition to the Network Mode (Repeat Message State) This service have no effect if the current state is not equal to Sleep Mode. In that case E_NOT_OK is returned	
Particularities and Limitations	
Called from NM Interface	
Pre-Conditions	
NM is initialized	
Call Context	
Function could be called from interrupt level or from task level, Reentrant	

Table 5-7 LinNm_PassiveStartUp

Caution
Do not call this function if Passive Mode is enabled or the application does not call the `LinNm_NetworkRelease` function. **This can avoid the ECU from sleep**! Refer also to chapter 3.2 'Network Management Mechanism'

5.4.4 LinNm_NetworkRequest

Prototype	
`Std_ReturnType` **`LinNm_NetworkRequest`** `(const NetworkHandleType nmChannelHandle)`	
Parameter	
`nmChannelHandle`	Channel parameters
Return Code	
Std_ReturnType	E_OK - No error E_NOT_OK - Requesting bus-communication has failed
Functional Description	
Request bus-communication.	
Particularities and Limitations	
Called from NM Interface	

Pre-Conditions
NM is initialized
Call Context
Function could be called from interrupt level or from task level, Reentrant

Table 5-8 LinNm_NetworkRequest

5.4.5 LinNm_NetworkRelease

Prototype	
`Std_ReturnType` **`LinNm_NetworkRelease`** `(const NetworkHandleType nmChannelHandle)`	
Parameter	
`nmChannelHandle`	Channel parameters
Return Code	
Std_ReturnType	E_OK - No error E_NOT_OK - Releasing bus-communication has failed
Functional Description	
Release bus-communication	
Particularities and Limitations	
Called from NM Interface	
Pre-Conditions	
NM is initialized	
Call Context	
Function could be called from interrupt level or from task level, Reentrant	

Table 5-9 LinNm_NetworkRelease

5.4.6 LinNm_GetState

Prototype	
`Std_ReturnType` **`LinNm_GetState`** `(const NetworkHandleType nmChannelHandle, Nm_StateType * const nmStatePtr, Nm_ModeType * const nmModePtr)`	
Parameter	
`nmChannelHandle`	Channel parameters
`nmStatePtr`	Pointer where the state of the Network Management shall be copied to
`nmModePtr`	Pointer where the mode of the Network Management shall be copied to
Return Code	
Std_ReturnType	E_OK - No error E_NOT_OK - Getting the NM state has failed
Functional Description	
Return current state and mode of the network management	

Particularities and Limitations
Called from NM Interface
Pre-Conditions
NM is initialized
Call Context
Function could be called from interrupt level or from task level, Reentrant

Table 5-10 LinNm_GetState

5.4.7 LinNm_GetVersionInfo

Prototype	
`void LinNm_GetVersionInfo (Std_VersionInfoType *versioninfo)`	
Parameter	
`versioninfo`	Pointer to store the version information to
Return Code	
-	-
Functional Description	
Return Version Info of this Module	
Particularities and Limitations	
Called from Application	
Pre-Conditions	
none	
Call Context	
Function could be called from interrupt level or from task level, Reentrant	

Table 5-11 LinNm_GetVersionInfo

5.4.8 Empty functions

The following functions are provided as empty functions if the corresponding features are activated, since their original purpose cannot be fulfilled due to the limited NM algorithm.

5.4.8.1 LinNm_SetUserData

Prototype	
`Std_ReturnType **LinNm_SetUserData** (const NetworkHandleType NetworkHandle, const uint8 * const nmUserDataPtr)`	
Parameter	
`NetworkHandle`	-
`nmUserDataPtr`	-
Return Code	
Std_ReturnType	E_OK - No error
Functional Description	
Empty function to be complaint with NM specifications.	
Particularities and Limitations	
-	
Pre-Conditions	
-	
Call Context	
-	

Table 5-12 LinNm_SetUserData

5.4.8.2 LinNm_GetUserData

Prototype	
`Std_ReturnType **LinNm_GetUserData** (const NetworkHandleType NetworkHandle, uint8 * const nmUserDataPtr)`	
Parameter	
`NetworkHandle`	-
`nmUserDataPtr`	-
Return Code	
Std_ReturnType	E_OK - No error
Functional Description	
Empty function to be complaint with NM specifications.	
Particularities and Limitations	
-	
Pre-Conditions	
-	
Call Context	
-	

Table 5-13 LinNm_GetUserData

5.4.8.3 LinNm_GetNodeIdentifier

Prototype	
`Std_ReturnType` **`LinNm_GetNodeIdentifier`** `(const NetworkHandleType NetworkHandle, uint8 * const nmNodeIdPtr)`	
Parameter	
`NetworkHandle`	-
`nmNodeIdPtr`	-
Return Code	
Std_ReturnType	E_OK - No error
Functional Description	
Empty function to be complaint with NM specifications.	
Particularities and Limitations	
-	
Pre-Conditions	
-	
Call Context	
-	

Table 5-14 LinNm_GetNodeIdentifier

5.4.8.4 LinNm_GetLocalNodeIdentifier

Prototype	
`Std_ReturnType` **`LinNm_GetLocalNodeIdentifier`** `(const NetworkHandleType NetworkHandle, uint8 * const nmNodeIdPtr)`	
Parameter	
`NetworkHandle`	-
`nmNodeIdPtr`	-
Return Code	
Std_ReturnType	E_OK - No error
Functional Description	
Empty function to be complaint with NM specifications.	
Particularities and Limitations	
-	
Pre-Conditions	
-	
Call Context	
-	

Table 5-15 LinNm_GetLocalNodeIdentifier

5.4.8.5 LinNm_RepeatMessageRequest

Prototype	
`Std_ReturnType` **`LinNm_RepeatMessageRequest`** `(const NetworkHandleType NetworkHandle)`	
Parameter	
`NetworkHandle`	-
Return Code	
Std_ReturnType	E_OK - No error
Functional Description	
Empty function to be complaint with NM specifications.	
Particularities and Limitations	
-	
Pre-Conditions	
-	
Call Context	
-	

Table 5-16 LinNm_RepeatMessageRequest

5.4.8.6 LinNm_GetPduData

Prototype	
`Std_ReturnType` **`LinNm_GetPduData`** `(const NetworkHandleType NetworkHandle, uint8 * const nmPduDataPtr)`	
Parameter	
`NetworkHandle`	-
`nmPduDataPtr`	-
Return Code	
Std_ReturnType	E_OK - No error
Functional Description	
Empty function to be complaint with NM specifications.	
Particularities and Limitations	
-	
Pre-Conditions	
-	
Call Context	
-	

Table 5-17 LinNm_GetPduData

5.4.8.7 LinNm_RequestBusSynchronization

Prototype	
`Std_ReturnType` **`LinNm_RequestBusSynchronization`** `(const NetworkHandleType nmChannelHandle)`	
Parameter	
`nmChannelHandle`	-
Return Code	
Std_ReturnType	E_OK - No error
Functional Description	
Empty function to be complaint with NM specifications.	
Particularities and Limitations	
-	
Pre-Conditions	
-	
Call Context	
-	

Table 5-18 LinNm_RequestBusSynchronization

5.4.8.8 LinNm_CheckRemoteSleepIndication

Prototype	
`Std_ReturnType` **`LinNm_CheckRemoteSleepIndication`** `(const NetworkHandleType nmChannelHandle, boolean * const nmRemoteSleepIndPtr)`	
Parameter	
`nmChannelHandle`	-
`nmRemoteSleepIndPtr`	-
Return Code	
Std_ReturnType	E_OK - No error
Functional Description	
Empty function to be complaint with NM specifications.	
Particularities and Limitations	
-	
Pre-Conditions	
-	
Call Context	
-	

Table 5-19 LinNm_CheckRemoteSleepIndication

5.4.8.9 LinNm_Transmit

Prototype	
`Std_ReturnType` **`LinNm_Transmit`** `(PduIdType LinTxPduId, const PduInfoType *PduInfoPtr)`	
Parameter	
`LinTxPduId`	-
`PduInfoPtr`	-
Return Code	
Std_ReturnType	E_NOT_OK - returns always
Functional Description	
Empty function to be complaint with NM specifications. Always return E_NOT_OK.	
Particularities and Limitations	
-	
Pre-Conditions	
-	
Call Context	
-	

Table 5-20 LinNm_Transmit

5.4.8.10 LinNm_SetSleepReadyBit

Prototype	
`Std_ReturnType` **`LinNm_SetSleepReadyBit`** `(const NetworkHandleType nmChannelHandle, const boolean nmSleepReadyBit)`	
Parameter	
`nmChannelHandle`	-
`nmSleepReadyBit`	-
Return Code	
Std_ReturnType	E_OK - No error
Functional Description	
Empty function to be complaint with NM specifications.	
Particularities and Limitations	
-	
Pre-Conditions	
-	
Call Context	
-	

Table 5-21 LinNm_SetSleepReadyBit

5.4.8.11 LinNm_EnableCommunication

Prototype	
`Std_ReturnType` **`LinNm_EnableCommunication`** `(const NetworkHandleType NetworkHandle)`	
Parameter	
`NetworkHandle`	-
Return Code	
Std_ReturnType	E_OK - No error
Functional Description	
Empty function to be complaint with NM specifications.	
Particularities and Limitations	
-	
Pre-Conditions	
-	
Call Context	
-	

Table 5-22 LinNm_EnableCommunication

5.4.8.12 LinNm_DisableCommunication

Prototype	
`Std_ReturnType` **`LinNm_DisableCommunication`** `(const NetworkHandleType NetworkHandle)`	
Parameter	
`NetworkHandle`	-
Return Code	
Std_ReturnType	E_OK - No error
Functional Description	
Empty function to be complaint with NM specifications.	
Particularities and Limitations	
-	
Pre-Conditions	
-	
Call Context	
-	

Table 5-23 LinNm_DisableCommunication

5.5 Services used by LINNM

In the following table services provided by other components, which are used by the LINNM are listed. For details about prototype and functionality refer to the documentation of the providing component.

Component	API
DET	Det_ReportError[1]
NM	Nm_BusSleepMode Nm_NetworkMode Nm_RemoteSleepIndication[2] Nm_StateChangeNotification[3]

Table 5-24 Services used by the LINNM

[1] Service only used if the feature ‘Dev Error Detect’ is enabled
[2] Service only used if the feature ‘Remote Sleep Ind Enabled’ is enabled.
[3] Service only used if the feature ‘State Change Ind Enabled’ is enabled.

6 Glossary and Abbreviations

6.1 Glossary

Term	Description
-	-

Table 6-1 Glossary

6.2 Abbreviations

Abbreviation	Description
API	**A**pplication **P**rogramming **I**nterface
AUTOSAR	**Aut**omotive **O**pen **S**ystem **Ar**chitecture
LIN	**L**ocal **I**nterconnect **N**etwork
CAN	**C**ontroller **A**rea **N**etwork
ComM	**C**ommunication **M**anager
UDP	**U**ser **D**atagram **P**rotocol
BSW	**B**asis **S**oft**w**are
DET	**D**evelopment **E**rror **T**racer
DEM	**D**iagnostic **E**vent **M**anager
ECU	**E**lectronic **C**ontrol **U**nit
NM	**N**etwork **M**anagement
RAM	**R**andom **A**ccess **M**emory
ROM	**R**ead **O**nly **M**emory
SRS	**S**ystem **R**equirements **S**pecification (used for AUTOSAR documents)
SWS	**S**oft**w**are **S**pecification (used for AUTOSAR documents)
HIS	**H**erstelleri**i**nitiative **S**oftware
LinIf	**LIN I**nter**f**ace
BCD	**B**inary **C**oded **D**ecimal
EcuM	**ECU** State **M**anager

Table 6-2 Abbreviations

7 Contact

Visit our website for more information on

- News
- Products
- Demo software
- Support
- Training data
- Addresses

www.vector.com