VECTOR

MICROSAR LIN State Manager

Technical Reference

Version 5.00.01

Authors	Mark A. Fingerle
Status	Released

Document Information

History

Author	Date	Version	Remarks
Marco Pfalzgraf	2012-07-25	1.00.00	Initial creation of document
Marco Pfalzgraf	2013-04-08	1.01.00	ESCAN00066496: Reworked chapter 7.1 ‘Deviations’ Deleted chapter ‘Compiler Abstraction and Memory Mapping’
Marco Pfalzgraf	2014-05-15	1.02.00	ESCAN00074303: Added description for ‘Full Communication Mode Request Repetition’ in chapter 3.1.6
Marco Pfalzgraf	2014-10-16	2.00.00	ESCAN00077623: Added Post-Build Selectable to supported feature table
Marco Pfalzgraf	2014-12-09	2.01.00	ESCAN00080029: Added Schedule Table End Notification
Marco Pfalzgraf	2015-04-14	3.00.00	ESCAN00082411: Adapted list of supported features FEAT-427 (SafeBSW): Adapted Table 3-5 Rework: Adapted Table 4-2; Adapted Figure 4-1
Mark A. Fingerle	2019-06-13	5.00.00	3.3 State machine, 3.1.2, 3.1.3, 3.1.6, 3.5.1 Service ID, Adapt 4.3 Critical Sections, LinSM_GotoSleepIndication, 5.2.6, 5.4.4, 5.4.5, 7.2.3 Node Type Slave Table 3-6 Development Error Reporting: Assignment of checks to services removed
Mark A. Fingerle	2020-05-13	5.00.01	ESCAN00106366, 7.1.1 FullComRequest during shut down, 7.1.7 Transceiver activation

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_LINStateManager.pdf	1.3.0
[2]	AUTOSAR	AUTOSAR_SWS_DevelopmentErrorTracer.pdf	3.2.0
[3]	AUTOSAR	AUTOSAR_TR_BSWModuleList.pdf	1.6.0
[4]	AUTOSAR	AUTOSAR_SWS_BSWModeManager.pdf	1.2.0
[5]	Vector	TechnicalReference_MICROSAR_IdentityManager.pdf	latest
[6]	Vector	TechnicalReference_Asr_LinIf.pdf	latest
[7]	Vector	TechnicalReference_Asr_BswM.pdf	latest

Scope of the Document

This technical reference describes the general use of the LinSM basis software.

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.

Caution
This symbol calls your attention to warnings.

Contents

1. Component History 7

2. Introduction 8
2.1 Architecture Overview 9

3. Functional Description 11
3.1 Features 11
3.1.1 Request Full Communication 12
3.1.2 Request No Communication 12
3.1.3 Go to Sleep Indication (Slave Node only) 12
3.1.4 Switching Schedule Tables (Master Node only) 12
3.1.5 Confirmation Timeout Handling 13
3.1.6 Full Communication Mode Request Repetition 13
3.1.7 MICROSAR Identity Manager using Post-Build Selectable 13
3.1.8 Schedule Table End Notification (Master Node only) 13
3.2 Initialization 14
3.3 States 14
3.4 Main Functions 15
3.5 Error Handling 15
3.5.1 Development Error Reporting 15
3.5.2 Production Code Error Reporting 17

4. Integration 18
4.1 Scope of Delivery 18
4.1.1 Static Files 18
4.1.2 Dynamic Files 18
4.2 Include Structure 19
4.3 Critical Sections 19

5. API Description 22
5.1 Type Definitions 22
5.2 Services provided by LinSM 22
5.2.1 LinSM_InitMemory 22
5.2.2 LinSM_Init 23
5.2.3 LinSM_MainFunction 23
5.2.4 LinSM_GetVersionInfo 24
5.2.5 LinSM_RequestComMode 24
5.2.6 LinSM_ScheduleRequest 25

5.2.7 LinSM_GetCurrentComMode 26
5.3 Services used by LinSM 26
5.4 Callback Functions 27
5.4.1 LinSM_WakeupConfirmation 27
5.4.2 LinSM_GotoSleepConfirmation 28
5.4.3 LinSM_GotoSleepIndication 28
5.4.4 LinSM_ScheduleRequestConfirmation 29
5.4.5 LinSM_ScheduleEndNotification 29

6. Configuration 30
6.1 Configuration Variants 30

7. AUTOSAR Standard Compliance 31
7.1 Deviations 31
7.1.1 Processing of Concurrent Requests 31
7.1.2 No Notification with same State or Schedule when Request Fails 31
7.1.3 Communication Requests not Handled on Task Level 32
7.1.4 Null Schedule is not Requested in Init Function 32
7.1.5 No Production Error Detection 32
7.1.6 ComM bus sleep event notification 32
7.1.7 Transceiver activation 32
7.2 Additions/ Extensions 32
7.2.1 Additional DET Error Codes 32
7.2.2 Memory Initialization 33
7.2.3 Node Type Slave 33

8. Glossary and Abbreviations 34
8.1 Glossary 34
8.2 Abbreviations 34

9. Contact 35

Illustrations

Figure 2-1 AUTOSAR 4.x Architecture Overview 9
Figure 2-2 Interfaces to adjacent modules of the LinSM 10
Figure 3-1 LinSM state machine 15
Figure 4-1 Include structure 19

Tables

Table 1-1 Component history 7
Table 3-1 Supported AUTOSAR standard conform features 11
Table 3-2 Not supported AUTOSAR standard conform features 11
Table 3-3 Features provided beyond the AUTOSAR standard 12
Table 3-4 Service IDs 16
Table 3-5 Errors reported to DET 17
Table 4-1 Static files 18
Table 4-2 Generated files 18
Table 5-1 Type definitions 22
Table 5-2 LinSM_InitMemory 22
Table 5-3 LinSM_Init 23
Table 5-4 LinSM_MainFunction 23
Table 5-5 LinSM_GetVersionInfo 24
Table 5-6 LinSM_RequestComMode 24
Table 5-7 LinSM_ScheduleRequest 25
Table 5-8 LinSM_GetCurrentComMode 26
Table 5-9 Services used by the LinSM 26
Table 5-10 LinSM_WakeupConfirmation 27
Table 5-11 LinSM_GotoSleepConfirmation 28
Table 5-12 LinSM_GotoSleepIndication 28
Table 5-13 LinSM_ScheduleRequestConfirmation 29
Table 5-14 LinSM_ScheduleEndNotification 29
Table 8-1 Glossary 34
Table 8-2 Abbreviations 34

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	The LinSM is implemented according to AUTOSAR release 4.0.3. In this first release only configuration variant pre-compile is supported.
2.01.00	Support of Full Communication Mode Request Repetition according to RfC57658.
3.00.00	Support of “MICROSAR Identity Manager using Post-Build Selectable” and “Post-Build Loadable”
3.01.00	Support Schedule Table End Notification

Table 1-1 Component history

2. Introduction

This document describes the functionality, API and configuration of the AUTOSAR BSW module LinSM as specified in [1].

Supported AUTOSAR Release*:	4	
Supported Configuration Variants:	pre-compile, post-build-loadable, post-build-selectable	
Vendor ID:	LINSM_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	LINSM_MODULE_ID	141 decimal (according to ref. [3])

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

The LinSM is responsible for the control flow of the LIN bus. Upper software layers can request the following services of the LinSM:

- Request and release communication on the LIN bus
- Switching schedule tables

Furthermore the following functionality is also in the scope of the LinSM:

- LinSM performs a timeout supervision of confirmations made by LinIf as response to go-to-sleep, wake-up and switch schedule table request.
- Upon successful mode changes and schedule switches upper layers (ComM and BswM) are notified by LinSM by corresponding callouts.
- The LIN transceiver mode is set by LinSM according to the current communication mode.

2.1 Architecture Overview

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

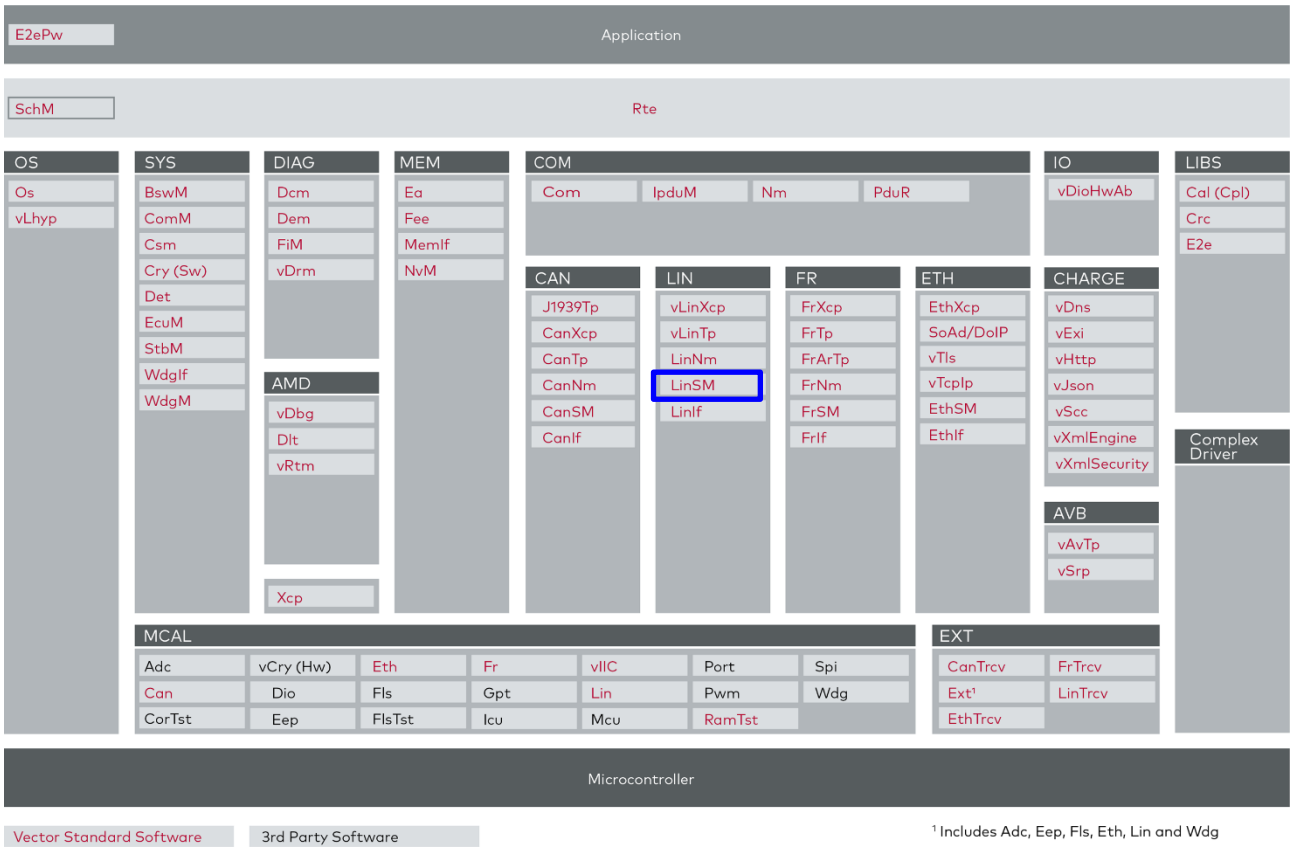

Figure 2-1 AUTOSAR 4.x Architecture Overview

The next figure shows the interfaces to adjacent modules of the LinSM. These interfaces are described in chapter 5.

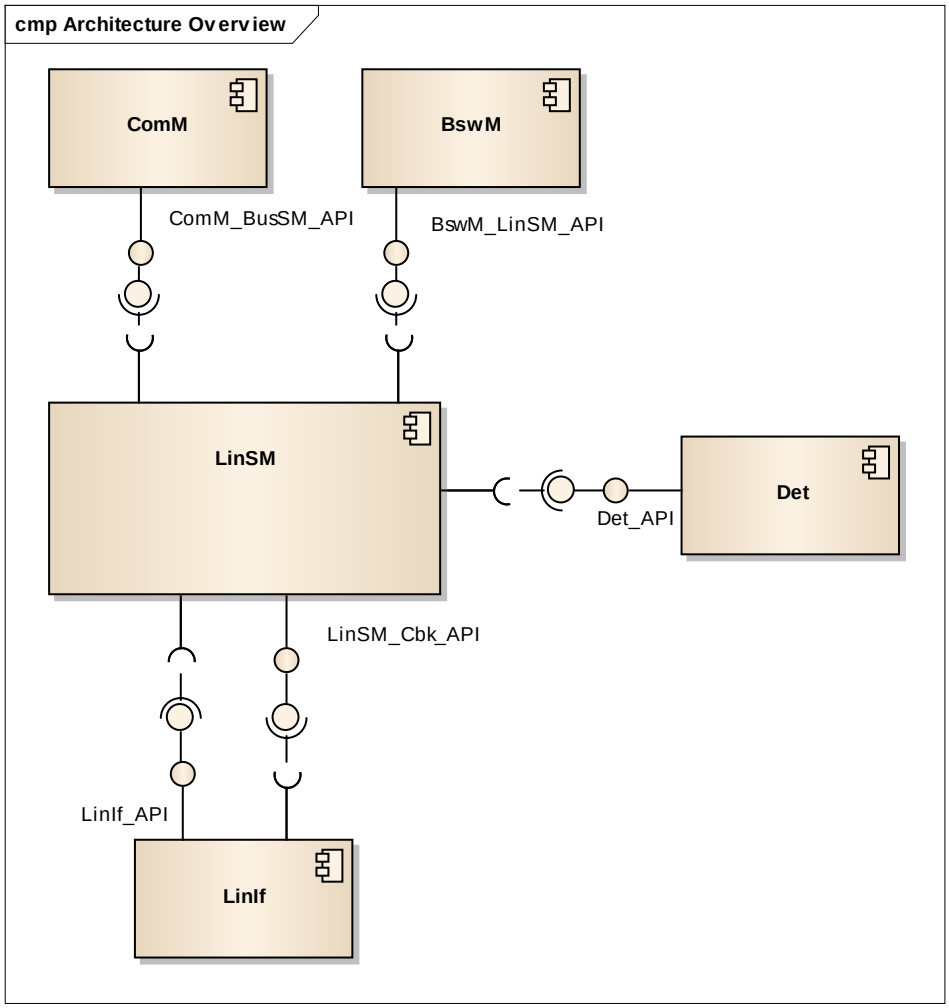

Figure 2-2 Interfaces to adjacent modules of the LinSM

Applications do not access the services of the BSW modules directly. They use the service ports provided by the BSW modules via the RTE. Currently the LinSM does not provide any service ports that can be used by applications.

3. Functional Description

3.1 Features

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

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

For further information of not supported features see also chapter 7.

Vector Informatik provides further LinSM 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
Request LIN bus communication (full or no communication)
LIN Interface confirmation timeout handling
LIN schedule table switching
Setting LIN transceiver mode according to communication mode
Development error detection
Handling multiple networks and drivers
Configuration variant “pre-compile”
MICROSAR Identity Manager using Post-Build Selectable
Post-Build Loadable

Table 3-1 Supported AUTOSAR standard conform features

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

Not Supported AUTOSAR Standard Conform Features
Configuration variants “link-time” and “post-build” are currently not supported

Table 3-2 Not supported AUTOSAR standard conform features

The following features are provided beyond the AUTOSAR standard:

Features Provided Beyond The AUTOSAR Standard
Schedule Table End Notification
LIN Slave support

Table 3-3 Features provided beyond the AUTOSAR standard

3.1.1 Request Full Communication

The LinSM provides the API function LinSM_RequestComMode() which allows the ComM to request full communication mode. The request works in an asynchronous way. It is forwarded to LIN interface by calling LinIf_Wakeup(). When the wake-up process is finished the LinSM is notified by LinIf via the callback function LinSM_WakeupConfirmation(). After successful confirmation the ComM and BswM are notified about the new communication state via the functions ComM_BusSM_ModeIndication() and BswM_LinSM_CurrentState().

3.1.2 Request No Communication

Master Node:

No communication is also requested by the ComM via the API function LinSM_RequestComMode(). The request works in an asynchronous way. It is forwarded to LIN Interface by calling LinIf_GotoSleep(). When the go-to-sleep process is finished the LinSM is notified by LinIf via the callback function LinSM_GotoSleepConfirmation(). After successful confirmation the ComM and BswM are notified about the new communication state via the functions ComM_BusSM_ModeIndication() and BswM_LinSM_CurrentState().

Slave Node:

Just stores the changed request. Shutdown is triggered via go to sleep indication (see 3.1.3).

3.1.3 Go to Sleep Indication (Slave Node only)

The go to sleep indication is called by the LinIf via the API function LinSM_GotoSleepIndication(). The LinIf will call this callback when the go to sleep command is received on the network or a bus idle timeout occurs. The indication is forwarded to LIN Interface by calling LinIf_GotoSleep(). When the requested communication mode by ComM module is COMM_NO_COMMUNICATION, the ComM will be notified of the bus sleep event by calling ComM_BusSM_BusSleepMode for the specified network. When the go-to-sleep process is finished the LinSM is notified by LinIf via the callback function LinSM_GotoSleepConfirmation(). After successful confirmation and if no communication is requested the ComM and BswM are notified about the new communication state via the functions ComM_BusSM_ModeIndication() and BswM_LinSM_CurrentState(). But if still full communication is requested the LinSM Slave Node reinitiates the wakeup (like in 3.1.1).

3.1.4 Switching Schedule Tables (Master Node only)

The LinSM provides the API function LinSM_ScheduleRequest() which allows the BswM to switch a LIN schedule table. A schedule request is only accepted in state LINSM_FULL_COM. The LinSM forwards the request to LIN Interface by calling LinIf_ScheduleRequest(). When the requested schedule table was activated the LinIf confirms the schedule switch via the LinSM callback function LinSM_ScheduleRequestConfirmation(). After successful confirmation the BswM is notified about the new schedule table via the function BswM_LinSM_CurrentSchedule().

3.1.5 Confirmation Timeout Handling

For each request that is processed by the LinSM (‘Request Full Communication’, ‘Request No Communication’ and ‘Switching Schedule Tables’), a timeout handling is performed. The LinSM expects each request to be successfully processed within a configurable time. This time can be configured as ‘Confirmation Timeout’ in the configuration tool for every LinSM channel. If no confirmation is made by the LinIf within the expected timeout, the LinSM will report the development error ‘LINSM_E_CONFIRMATION_TIMEOUT’ independently of the corresponding request. The following actions depend on the type of request that timed out:

- **Full Communication Request:** After confirmation timeout is detected, LinSM will repeat the call of LinIf_Wakeup().
- **No Communication Request:** After confirmation timeout is detected, LinSM will change its state immediately to LINSM_NO_COM and notifies BswM and ComM about no communication state.
- **Switch Schedule Table Request:** After confirmation timeout of a schedule table request, there are no further actions besides the DET error necessary. It is assumed that the schedule is not switched and therefore the BswM will not be notified.

3.1.6 Full Communication Mode Request Repetition

If a Full Communication request is not accepted by LinIf, i.e. the wake-up request to the LinIf was not successful, the LinSM will repeat the call of LinIf_Wakeup() for a configurable number of times. The wake-up request is considered as not successful if the LinSM_WakeupConfirmation is called with success parameter set to FALSE or when a wake-up confirmation timeout occurs.

The number of repetitions can be configured with the parameter LinSMModeRequestRepetitionMax. If the parameter does not exist or it is set to zero, no repetition will be performed.

Note that the wake-up request will be not successful due to integration issues only. In error case LinSM will report a DET error as described in chapter 3.5.1 ‘Development Error Reporting’.

If the maximum number of retries has been executed

- the Master Node will initiate a shut down like communication mode request no communication was received (see 3.1.2).
- the Slave Node stops any wakeup action until the configured LinSMSilenceAfterWakeupTimeout has elapsed. The node still reacts at the go to sleep indication. But even after no com has been reached, a new full communication request will be delayed, if the silence timer is still running.

3.1.7 MICROSAR Identity Manager using Post-Build Selectable

LinSM supports Post-Build Selectable. Please refer to [5] for details about this feature.

3.1.8 Schedule Table End Notification (Master Node only)

The LinSM supports the “schedule table end notification”. The feature can be activated in LinIf configuration by setting the parameter LinIfScheduleEndNotificationSupported.

If enabled the LinIf notifies the LinSM at the end of each schedule table. The LinSM just forwards this information to the BswM by calling the corresponding ScheduleEndNotification Request Port. The user can use this request port in BswM rules to trigger BswM actions upon the end of schedule tables. E.g. the schedule table end notification can be used to switch immediately to a new schedule when the current schedule has finished. For further information please also refer to [6] and [7]. API details can be found in chapter 5.4.5 ‘LinSM_ScheduleEndNotification’.

3.2 Initialization

Before calling any other functionality of the LinSM has to be initialized by calling LinSM_Init(). As the LinSM will not call service function of other modules during initialization it can be initialized before any other adjacent module (LinIf, ComM, BswM or Det) is initialized. Refer also to 5.2.2 ‘LinSM_Init’.

The LinSM assumes that some variables are initialized with certain values at start-up. As not all embedded targets support the initialization of RAM within start-up code the LinSM module provides the function LinSM_InitMemory(). This function has to be called during start-up and before LinSM_Init() is called. Refer also to chapter 7.2.2 ’Memory Initialization’. For API details refer to chapter 5.2.1 ‘LinSM_InitMemory’.

3.3 States

Figure 3-1 shows the LinSM internal state machine. The figure gives an overview about the internal behavior of the module.

On the first level there are two global states, LINSM_UNINIT and LINSM_INIT. These are only used, if Development Error Detection is enabled to check if a LinSM service function is called when the LinSM was not initialized before. For more information about initialization see chapter 3.1.6.

For each channel where the LinSM is active there are the states LINSM_NO_COM and LINSM_FULL_COM. After initialization each channel is set to LINSM_NO_COM state. The states are changed upon communication requests by the ComM. The requests are forwarded to the LinIf. After successful wake-up or sleep request the LinIf will notify the LinSM, which in turn changes its state and notifies the BswM and ComM about the new state. See also chapters 3.1.1 ‘Request Full Communication’ 3.1.2 ‘Request No Communication’ for more information how communication requests are handled.

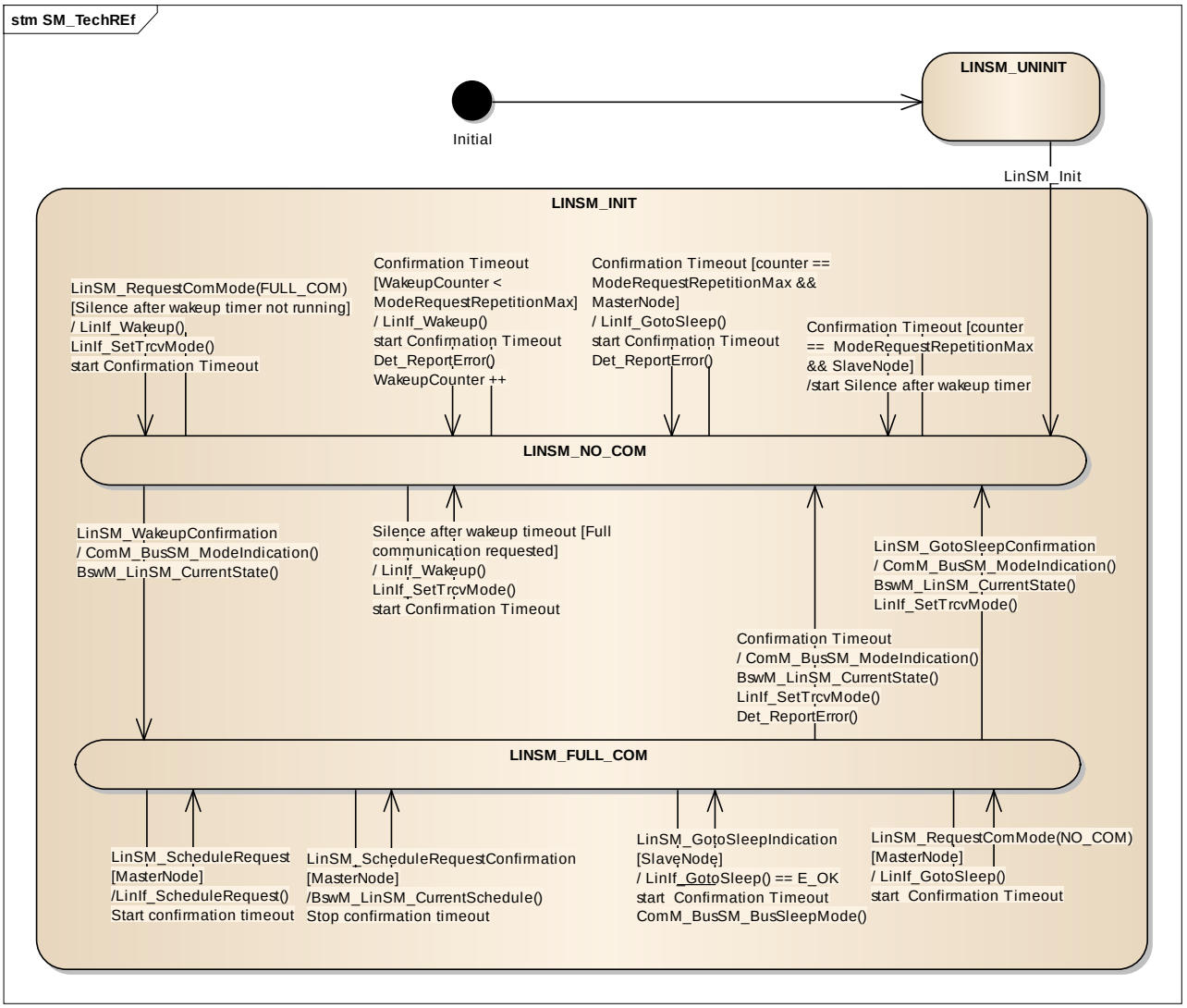

Figure 3-1 LinSM state machine

3.4 Main Functions

The LinSM implementation provides one main function. This main function has to be called cyclically on task level by the BSW Scheduler. The default cycle time is 10 milliseconds and can be configured in the configuration tool.

For API details refer to chapter 5.2.3 ‘LinSM_MainFunction’.

3.5 Error Handling

3.5.1 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 `LINSM_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 LinSM ID is 141.

The reported service IDs identify the services which are described in 5.2. The following table presents the service IDs and the related services:

Service ID	Service
0x01u	LinSM_Init
0x02u	LinSM_GetVersionInfo
0x03u	LinSM_GotoSleepIndication
0x10u	LinSM_ScheduleRequest
0x11u	LinSM_GetCurrentComMode
0x12u	LinSM_RequestComMode
0x20u	LinSM_ScheduleRequestConfirmation
0x21u	LinSM_WakeupConfirmation
0x22u	LinSM_GotoSleepConfirmation
0x23u	LinSM_ScheduleEndNotification
0x30u	LinSM_MainFunction
0x40u	LinSM_SwitchTransceiver[1]

Table 3-4 Service IDs

[1] LinSM_SwitchTransceiver is an internal function which is used within LinSM_MainFunction(), LinSM_RequestComMode() and LinSM_GotoSleepConfirmation().

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

Error Code		Description
0x00u	LINSM_E_UNINIT	API called before initialization.
0x0Au	LINSM_E_PARAM_CONFIG	LinSM_Init called with wrong configuration parameter.
0x20u	LINSM_E_NONEXISTENT_NETWORK	API called with invalid network handle.
0x30u	LINSM_E_PARAMETER	API called with invalid parameter.
0x40u	LINSM_E_PARAMETER_POINTER	API called with null-pointer as parameter.
0x50u	LINSM_E_CONFIRMATION_TIMEOUT	A LinIf confirmation timeout occurred (wake up, go to sleep or schedule request confirmation timeout).
0x60u	LINSM_E_INIT_FAILED	LinSM_Init is called while the module is already initialized.
0x61u	LINSM_E_GETTRCVMODE_FAILED	LinIf_GetTrcvMode() returned E_NOT_OK.
0x62u	LINSM_E_SETTRCVMODE_FAILED	LinIf_SetTrcvMode() returned E_NOT_OK.
0x63u	LINSM_E_NOT_IN_FULL_COM	LinSM was not in expected state LINSM_FULL_COM.
0x64u	LINSM_E_REQUEST_IN_PROCESS	Schedule switch requested when a communication request (no or full) was still in process.
0x65u	LINSM_E_MODE_ALREADY_ACTIVE	The requested mode (no or full com) was already active.
0x66u	LINSM_E_FULLCOM_NOT_REQUESTED	The wake-up confirmation function was called without a preceding full communication request.
0x67u	LINSM_E_NOCOM_NOT_REQUESTED	The go-to-sleep confirmation was called without a preceding no communication request.
0x68u	LINSM_E_TIMEOUT_IN_NO_COM	A confirmation timeout occurred in state LINSM_NO_COM when full communication was not requested.
0x69u	LINSM_E_WAKEUPCONF_NO_SUCCESS	LinIf reported a negative wake-up confirmation (success == FALSE).
0x6Au	LINSM_E_NODE_TYPE	API service called at a channel where the wrong node type (Master/Slave) is configured.

Table 3-5 Errors reported to DET

3.5.2 Production Code Error Reporting

There are currently no production errors reported by LinSM.

4. Integration

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

4.1 Scope of Delivery

The delivery of the LinSM contains the files which are described in the following.

4.1.1 Static Files

File Name	Description
LinSM.c	This is the source file of the LinSM.
LinSM.h	This is the header file of the LinSM which defines the services of the LinSM used by upper software layers.
LinSM_Cbk.h	Header file which includes the external declarations of the callback functions of the LinSM. File has to be included by LinIf which implements LinSM callback functions.
LinSM_Types.h	Header file which includes LinSM specific data types.

Table 4-1 Static files

4.1.2 Dynamic Files

The dynamic files are generated by the configuration tool DaVinci Configurator 5.

File Name	Description
LinSM_Cfg.h	Generated configuration header file that contains all pre-compile switches which en/disable LinSM features. It also contains declarations of all generated data.
LinSM_Cfg.c	Generated configuration source file that contains all pre-compile configurable data.
LinSM_PBcfg.h	Generated configuration header file that contains declarations of post-build changeable data.
LinSM_PBcfg.c	Generated configuration source file that contains all post-build changeable data.

Table 4-2 Generated files

4.2 Include Structure

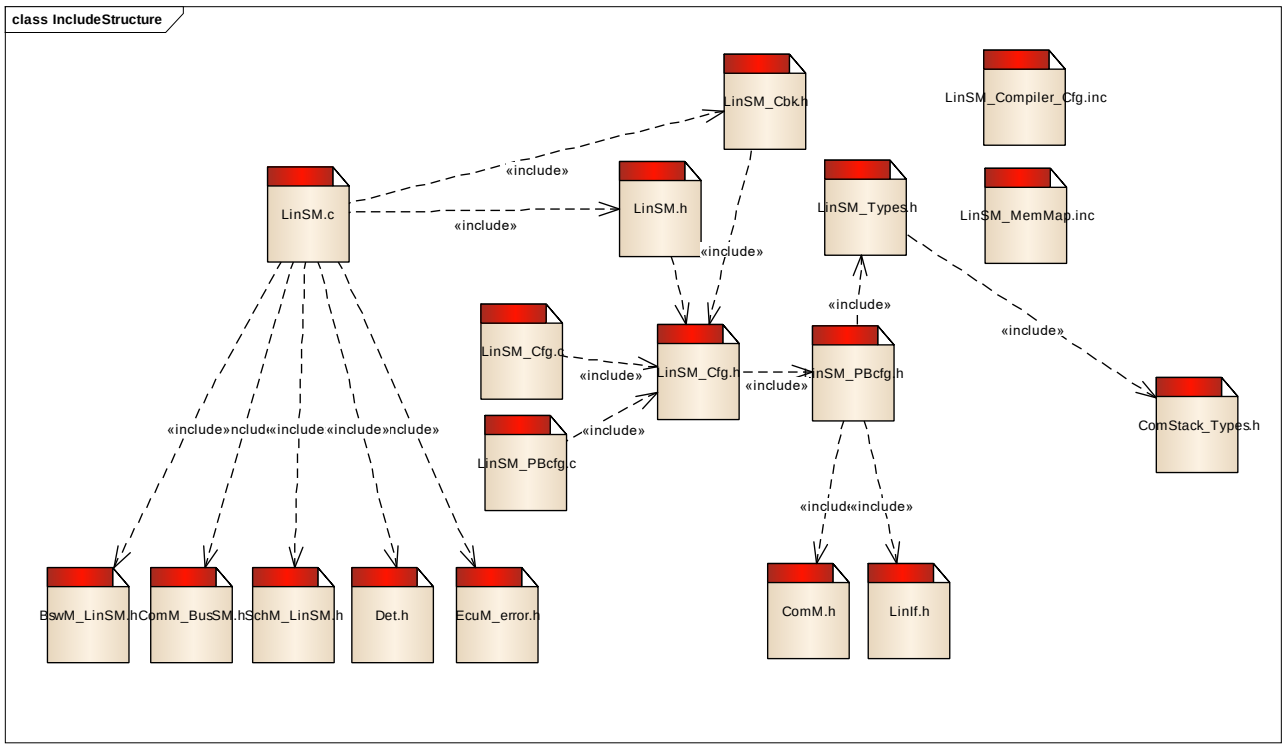

Figure 4-1 Include structure

4.3 Critical Sections

Normally the LinSM has no code sections which are executed on interrupt level, thus no interrupt locks are required. But if a full preemptive operating system is used and there are cyclic tasks that have different priority levels (tasks can interrupt each other), critical code sections have to be protected.

Another reason for critical code section protection is when the BswM action that requests schedule switches is processed in “immediate” mode. Please refer to [4] for more information about “immediate” and “deferred” operation of BswM.

The LinSM implements code sections which are called out of four different contexts:

- **LinSM task context:** Refer to chapter 5.2.3 ‘LinSM_MainFunction’
- **LinIf task context:** The LinIf_MainFunction calls the LinSM confirmation functions (LinSM_ScheduleRequestConfirmation, LinSM_WakeupConfirmation(), LinSM_GotoSleepConfirmation)
- **ComM task context:** The ComM_MainFunction calls the function LinSM_RequestComMode()
- **Context of BswM action BswMLinScheduleSwitch:** With this action the BswM is able to request schedule switches, i.e. BswM will call LinSM_ScheduleRequest when the action is executed. The action is either executed on BswM task or interrupt context. This depends on how the corresponding rule is configured to be processed “deferred” (task context) or “immediate” (interrupt context).

To ensure data consistency and a correct function of the LinSM these four code sections must not interrupt each other. Therefore the integrator has to configure the following exclusive areas in schedule manager:

LINSM_EXCLUSIVE_AREA_0: Must lock task interrupts if the function `LinSM_RequestComMode()` may be interrupted by any of the functions

- `LinSM_MainFunction()`
- `LinSM_GotoSleepIndication()`
- `LinSM_WakeupConfirmation()`
- `LinSM_GotoSleepConfirmation()`

LINSM_EXCLUSIVE_AREA_1: Must lock task interrupts if any of the functions `LinSM_WakeupConfirmation()`, `LinSM_GotoSleepConfirmation()` or `LinSM_ScheduleRequestConfirmation()` may be interrupted by any of the functions

- `LinSM_MainFunction()`
- `LinSM_RequestComMode()`
- `LinSM_GotoSleepIndication()`
- `LinSM_WakeupConfirmation()`
- `LinSM_GotoSleepConfirmation()`

LINSM_EXCLUSIVE_AREA_2: Must lock task interrupts if the function `LinSM_MainFunction()` may be interrupted by any of the functions

- `LinSM_RequestComMode()`
- `LinSM_GotoSleepIndication()`
- `LinSM_WakeupConfirmation()`
- `LinSM_GotoSleepConfirmation()`

LINSM_EXCLUSIVE_AREA_3: Must lock task interrupts if the function `LinSM_ScheduleRequest()` may be interrupted by any of the functions

- `LinSM_MainFunction()`
- `LinSM_RequestComMode()`
- `LinSM_GotoSleepIndication()`
- `LinSM_WakeupConfirmation()`

> `LinSM_GotoSleepConfirmation()`

LINSM_EXCLUSIVE_AREA_4: Must lock task interrupts if the function `LinSM_GotoSleepIndication()` may be interrupted by any of the functions

> `LinSM_MainFunction()`

> `LinSM_RequestComMode()`

> `LinSM_WakeupConfirmation()`

> `LinSM_GotoSleepConfirmation()`

It is recommended to use AUTOSAR OS ‘Resources’ for these exclusive areas to prevent priority inversion and dead-locks.

5. API Description

For an interfaces overview please see Figure 2-2.

5.1 Type Definitions

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

Type Name	C-Type	Description	Value Range
LinSM_ModeType	uint8	This is the type for the state parameter used for BswM_LinSM_CurrentState().	`LINSM_BSWM_FULL_COM` LinSM is in state LINSM_FULL_COM
			`LINSM_BSWM_NO_COM` LinSM is in state LINSM_NO_COM

Table 5-1 Type definitions

5.2 Services provided by LinSM

5.2.1 LinSM_InitMemory

Prototype	
`void` **`LinSM_InitMemory`** `(void)`	
Parameter	
N/A	
Return code	
void	N/A
Functional Description	
This function initializes the LinSM memory, the global state is set to LINSM_UNINIT.	
Particularities and Limitations	
This function has to be called only once during power on.	
Call Context	
System Startup	

Table 5-2 LinSM_InitMemory

5.2.2 LinSM_Init

Prototype	
`void LinSM_Init (const LinSM_ConfigType *ConfigPtr)`	
Parameter	
ConfigPtr	Pointer to post-build data structure
Return code	
void	N/A
Functional Description	
This function initializes the LinSM. All variables are set to default values. The LinSM global state is set to LINSM_INIT, the sub-state (communication state) is set to LINSM_NO_COM.	
Particularities and Limitations	
This function has to be called only once during power on after LinSM_InitMemory() was called.	
Call Context	
System Startup	

Table 5-3 LinSM_Init

5.2.3 LinSM_MainFunction

Prototype	
`void LinSM_MainFunction (void)`	
Parameter	
N/A	
Return code	
void	N/A
Functional Description	
Periodic function that handles the confirmation timeout handling of communication and schedule requests and the full communication repetition handling.	
Particularities and Limitations	
This function is usually called by the Basic Software Scheduler with the configured LinSM cycle time.	
Call Context	
LinSM Task Context	

Table 5-4 LinSM_MainFunction

5.2.4 LinSM_GetVersionInfo

Prototype	
`void LinSM_GetVersionInfo (Std_VersionInfoType *versioninfo)`	
Parameter	
versioninfo	Pointer where the version information of this module is copied to.
Return code	
void	N/A
Functional Description	
LinSM_GetVersionInfo() returns version information, vendor ID and AUTOSAR module ID of the component. The versions are BCD-coded.	
Particularities and Limitations	
Function is only available if enabled at pre-compile time (LINSM_VERSION_INFO_API == STD_ON).	
Call Context	
Task or interrupt context	

Table 5-5 LinSM_GetVersionInfo

5.2.5 LinSM_RequestComMode

Prototype	
`Std_ReturnType LinSM_RequestComMode (NetworkHandleType network, ComM_ModeType mode)`	
Parameter	
network	System channel handle
mode	Requested communication mode: Only COMM_FULL_COMMUNICATION or COMM_NO_COMMUNICATION is accepted.
Return code	
Std_ReturnType	E_OK - Request accepted E_NOT_OK - Not possible to perform the request, e.g. not initialized.
Functional Description	
This function is used by ComM to request a communication mode.	
Particularities and Limitations	
The mode switch will not be made instantly. The LinSM will notify the caller when mode transition is made.	
Call Context	
ComM Task context	

Table 5-6 LinSM_RequestComMode

5.2.6 LinSM_ScheduleRequest

Prototype	
`Std_ReturnType LinSM_ScheduleRequest (NetworkHandleType network, LinIf_SchHandleType schedule)`	
Parameter	
network	System channel handle
schedule	Index of the new schedule table
Return code	
Std_ReturnType	E_OK - Schedule table request has been accepted. E_NOT_OK - Schedule table switch request has not been accepted due to one of the following reasons: - LinSM has not been initialized - Referenced channel does not exist (identification is out of range) - Referenced schedule table does not exist (identification is out of range) - Sub-state is not LINSM_FULL_COM or a go-to-sleep request is pending
Functional Description	
The upper layer (BswM) requests a schedule table to be changed on the given LIN network.	
Particularities and Limitations	
Master only, i.e. the function is only available if on any channel the LinSMNodeType was set to MASTER. A schedule request is only accepted in state LINSM_FULL_COM.	
Call Context	
BswM Task context	

Table 5-7 LinSM_ScheduleRequest

5.2.7 LinSM_GetCurrentComMode

Prototype	
`Std_ReturnType LinSM_GetCurrentComMode (NetworkHandleType network, ComM_ModeType *mode)`	
Parameter	
network	System channel handle
mode	Pointer where the communication mode information is copied to.
Return code	
Std_ReturnType	E_OK - Ok E_NOT_OK - Not possible to perform the request, e.g. not initialized.
Functional Description	
Function to query the current communication mode. COMM_FULL_COMMUNICATION or COMM_NO_COMMUNICATION.	
Particularities and Limitations	
-	
Call Context	
Task or interrupt context	

Table 5-8 LinSM_GetCurrentComMode

5.3 Services used by LinSM

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

Component	API
DET	Det_ReportError
LinIf	LinIf_Wakeup
LinIf	LinIf_GotoSleep
LinIf	LinIf_ScheduleRequest
LinIf	LinIf_SetTrcvMode
LinIf	LinIf_GetTrcvMode
ComM	ComM_BusSM_ModeIndication
ComM	ComM_BusSM_BusSleepMode
BswM	BswM_LinSM_CurrentState
BswM	BswM_LinSM_CurrentSchedule
BswM	BswM_LinSM_ScheduleEndNotification

Table 5-9 Services used by the LinSM

5.4 Callback Functions

This chapter describes the callback functions that are implemented by the LinSM and can be invoked by other modules. The prototypes of the callback functions are provided in the header file `LinSM_Cbk.h` by the LinSM.

5.4.1 LinSM_WakeupConfirmation

Prototype	
`void LinSM_WakeupConfirmation ( NetworkHandleType network, boolean success)`	
Parameter	
network	System channel handle
success	True if wake-up was successfully sent, false otherwise.
Return code	
void	N/A
Functional Description	
The LinIf will call this callback with success == TRUE when the wake-up signal command has been sent successfully on the network. The LinSM stops the confirmation timeout timer and changes to full communication state.	
Particularities and Limitations	
-	
Call Context	
LinIf Task context	

Table 5-10 LinSM_WakeupConfirmation

5.4.2 LinSM_GotoSleepConfirmation

Prototype	
`void LinSM_GotoSleepConfirmation ( NetworkHandleType network, boolean success)`	
Parameter	
network	System channel handle
success	True if go-to-sleep was successfully sent, false otherwise.
Return code	
void	N/A
Functional Description	
The LinIf will call this callback when the go-to-sleep command has been sent successfully on the network. The LinSM stops the confirmation timeout timer, switches the transceiver to standby mode (if the transceiver handling is enabled) and changes to state LINSM_NO_COM.	
Particularities and Limitations	
-	
Call Context	
LinIf Task context	

Table 5-11 LinSM_GotoSleepConfirmation

5.4.3 LinSM_GotoSleepIndication

Prototype	
`void LinSM_GotoSleepIndication (NetworkHandleType network)`	
Parameter	
network [in]	System channel handle
Return code	
void	none
Functional Description	
Triggers the go-to-sleep. The LinIf will call this callback when the go to sleep command is received on the network or a bus idle timeout occurs. Only applicable for LIN slave nodes.	
Particularities and Limitations	
Slave only, i.e. the function is only available if on any channel the `LinSMNodeType` was set to SLAVE.	
Call context	
> TASK > This function is Non-Reentrant	

Table 5-12 LinSM_GotoSleepIndication

5.4.4 LinSM_ScheduleRequestConfirmation

Prototype	
`void LinSM_ScheduleRequestConfirmation (NetworkHandleType network, LinIf_SchHandleType schedule)`	
Parameter	
network	System channel handle
schedule	Index of the new active schedule table.
Return code	
void	N/A
Functional Description	
The LinIf module will call this callback when the new requested schedule table is active.	
Particularities and Limitations	
Master only, i.e. the function is only available if on any channel the LinSMNodeType was set to MASTER.	
Call Context	
LinIf Task context	

Table 5-13 LinSM_ScheduleRequestConfirmation

5.4.5 LinSM_ScheduleEndNotification

Prototype	
`void LinSM_ScheduleEndNotification (NetworkHandleType network, LinIf_SchHandleType schedule)`	
Parameter	
network	System channel handle
schedule	Index of the current schedule table.
Return code	
void	N/A
Functional Description	
The LinIf module will call this callback at the end of the current schedule table.	
Particularities and Limitations	
Master only, i.e. the callback is only available if on any channel the LinSMNodeType was set to MASTER and if the parameter LinIfScheduleEndNotificationSupported is enabled.	
Call context	
LinIf Task context	

Table 5-14 LinSM_ScheduleEndNotification

6. Configuration

6.1 Configuration Variants

The LinSM supports the following configuration variants

- VARIANT-PRE-COMPILE
- VARIANT-POST-BUILD-LOADABLE
- VARIANT-POST-BUILD-SELECTABLE
- VARIANT-POST-BUILD-LOADABLE-SELECTABLE

The configuration classes of the LinSM parameters depend on the supported configuration variants. For their definitions please see the LinSM_bswmd.arxml file.

7. AUTOSAR Standard Compliance

7.1 Deviations

7.1.1 Processing of Concurrent Requests

The AUTSOAR specification [1] states that requests shall be ignored if “another request is in process on the same network” (LINSM163, LINSM174). LinSM_RequestComMode() and LinSM_ScheduleRequest() shall directly return E_NOT_OK.

To achieve a common handling of communication requests in the ComM, the behavior of the LinSM was implemented consistently to other State Managers (CanSM and FrSM). So communication requests and schedule table requests are accepted even another request is still in process. The behavior for each request will be as follows:

If full communication is requested while a full communication request is still in process, LinSM_RequestComMode() will do nothing except return E_OK and waits until the last request will be confirmed by LinIf.

If full communication is requested while a no communication request is still in process, LinSM_RequestComMode() will notify NO_COM to ComM and BswM and call LinIf_Wakeup() and return E_OK. The LinIf will handle the concurrent request so that the last request will be performed and a corresponding LinSM confirmation function will be called.

If no communication is requested while a no communication request is still in process, LinSM_RequestComMode() will do nothing except return E_OK and waits until the last request will be confirmed by LinIf.

If no communication is requested while a full communication request is still in process, LinSM_RequestComMode() will call LinIf_GotoSleep() and return E_OK. The LinIf will handle the concurrent request so that the last request will be performed and a corresponding LinSM confirmation function will be called.

If a schedule table switch is requested while the last schedule table request is still in process, LinSM_ScheduleRequest() will call LinIf_ScheduleRequest() and return E_OK. The LinIf will start the last requested schedule table and will call LinSM_ScheduleRequestConfirmation() when the latest requested schedule will be switched.

Only if a schedule switch is requested while a no communication request is still in process LinSM_ScheduleRequest() will return E_NOT_OK and ignore the request. Additionally, a DET error “LINSM_E_REQUEST_IN_PROCESS” will be reported. In case a schedule table switch is requested while a full communication request is in process, the current state is not LINSM_FULL_COM and therefore E_NOT_OK is returned as specified in [1] (LINSM0211).

7.1.2 No Notification with same State or Schedule when Request Fails

To avoid unintended actions performed by BswM or ComM due to LinSM mode indications or schedule switch indications, the LinSM will not notify the BswM nor the ComM of unchanged state or schedule in case a communication or schedule request fails (e.g. due to a timeout or corresponding LinIf API returns E_NOT_OK). This means that

ComM_BusSM_ModeIndication, BswM_LinSM_CurrentState and BswM_LinSM_CurrentSchedule will never be called with the state or schedule that is already active. In particular the following requirements defined in [1] are not supported: LINSM046, LINSM170, LINSM177, LINSM0213, LINSM0214 and LINSM0215.

Exception here is the Full Communication request. If a LinIf wake-up request fails due to a negative confirmation or due to a confirmation timeout and maximum number of repetitions is reached, BswM as well as ComM are notified that LinSM is in state NO_COM state which is the same state before the request was made. As this happens due to integration issues only, a DET error will be reported in such a case.

7.1.3 Communication Requests not Handled on Task Level

According to [1] (LINSM0223) the LinSM shall store communication requests within LinSM_RequestComMode() and process the event on the next main function call.

This requirement is conflicting the requirements LINSM036 and LINSM047, that requires LinSM to call LinIf_Wakeup() and LinIf_GotoSleep() “directly” in LinSM_RequestComMode() “and not wait for next main function call”.

The MICROSAR LinSM implementation works according to the last mentioned requirements and forwards the requests directly in LinSM_RequestComMode() and not in the main function.

7.1.4 Null Schedule is not Requested in Init Function

According to [1] (LINSM0216) the LinSM “shall set the Null Schedule” within the LinSM_Init function. This requirement is not supported, because it is conflicted with LINSM151. Instead it is assumed that Null Schedule is the default in all relevant modules after initialization.

7.1.5 No Production Error Detection

As LinSM does not report any production errors the following requirements are not supported: LINSM011, LINSM051, LINSM056, LINSM058, LINSM085.

7.1.6 ComM bus sleep event notification

ComM_BusSM_BusSleepMode is always called even calling LinIf_GotoSleep() is answered with E_NOT_OK.

7.1.7 Transceiver activation

The transceiver is set to active when a full communication request is received, because for a successful wakeup the wakeup confirmation call is necessary which needs the LIN bus reception capabilities.

7.2 Additions/ Extensions

7.2.1 Additional DET Error Codes

There are additional DET error codes to those defined in [1] reported by the MICROSAR LinSM. Please refer to chapter 3.5.1 ‘Development Error Reporting’ for a complete list of defined error codes.

7.2.2 Memory Initialization

Not every start-up code of embedded targets and neither CANoe provides initialized RAM. So it might happen that the state of a variable that needs initialized RAM may not be set to the expected initial value. Therefore an explicit initialization of such variables has to be provided at start-up by calling the additional function LinSM_InitMemory().

For more information refer to chapter 3.2 ‘Initialization’.

7.2.3 Node Type Slave

The LinSM can be LIN master or LIN slave on more than one cluster. All references to (switching of) schedule tables do only apply to LIN master node, there are no schedule tables for LIN slave node. For slave nodes the startup (repetition) might be delayed by LinSMSilenceAfterWakeupTimeout (see 3.1.6). The shutdown for slave nodes is triggered by a new API (see 3.1.3)

8. Glossary and Abbreviations

8.1 Glossary

Term	Description
DaVinci Configurator	Generation tool for MICROSAR components

Table 8-1 Glossary

8.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
BswM	Basic Software Mode Manager
CanSM	CAN State Manager
ComM	Communication Manager
DET	Development Error Tracer
ECU	Electronic Control Unit
FrSM	FlexRay State Manager
ISR	Interrupt Service Routine
LinIf	LIN Interface
LinSM	LIN State Manager
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
RTE	Runtime Environment
SRS	Software Requirement Specification
SWC	Software Component
SWS	Software Specification

Table 8-2 Abbreviations

9. Contact

Visit our website for more information on

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

www.vector.com