MICROSAR LIN Driver

Technical Reference

LIN Driver Core

Version 6.2.0

Authors	Friedrich Kiesel, Bastian Molkenthin, Lutz Pflüger
Status	Released

Document Information

History

Author	Date	Version	Remarks
Friedrich Kiesel	2012-07-18	4.00.00	AUTOSAR 4.0.3
Friedrich Kiesel	2012-11-22	4.01.00	Added Polling Mode
Friedrich Kiesel	2012-12-06	4.01.01	Some minor corrections
Bastian Molkenthin	2013-04-11	4.02.00	Added Post-Build-Loadable
Lutz Pflüger	2013-10-28	5.00.00	Added Predefined Runtime Measurement Remove LIN_PROD_ERROR_DETECT
Lutz Pflüger	2014-11-04	6.00.00	Add POST-BUILD-SELECTABLE, Lin_WakeupInternal(), LinIf_WakeupConfirmation and numbers type of Lin_GetVersionInfo
Lutz Pflüger	2014-11-14	6.00.01	Delete Chapter 4.3
Lutz Pflüger	2015-02-04	6.00.02	Rename
Lutz Pflüger	2015-02-17	6.01.00	Add Lin_GetRxPin Macro description Fixing Lin_InitMemory and Lin_Init API description Change defines of Service ID's
Lutz Pflüger	2015-10-06	6.01.01	Add interrupt bits integration hint, Chapter 4.4
Lutz Pflüger	2015-12-17	6.02.00	Update Architecture Overview, Remove Lin_GetRxPin functionality, Change Chapter 3.3

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_SWS_LINDriver.pdf	2.2.0
[2]	AUTOSAR	AUTOSAR_SWS_DevelopmentErrorTracer.pdf	3.2.0
[3]	AUTOSAR	AUTOSAR_SWS_DiagnosticEventManager.pdf	4.2.0
[4]	AUTOSAR	AUTOSAR_BasicSoftwareModules.pdf	V1.0.0
[5]	Vector	TechnicalReference_Lin_[Controller_name].pdf	

Scope of the Document

This technical reference describes the general use of the LIN driver basis software. All aspects which are LIN controller specific are described in a separate document [5], 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.

Caution
This symbol calls your attention to warnings.

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
4.0.0	AUTOSAR 4
4.1.0	Support for Post-Build-Loadable
5.0.0	Predefined Runtime Measurement points (RTM)
6.0.0	Lin_WakeupInternal, LinIf_WakeupConfirmation and POST-BUILD-SELECTABLE

Table 1-1 Component history

2. Introduction

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

Supported AUTOSAR Release*:	4	
Supported Configuration Variants:	pre-compile	
Vendor ID:	LIN_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	LIN_MODULE_ID	82 decimal (according to ref. [4])

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

2.1 Architecture Overview

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

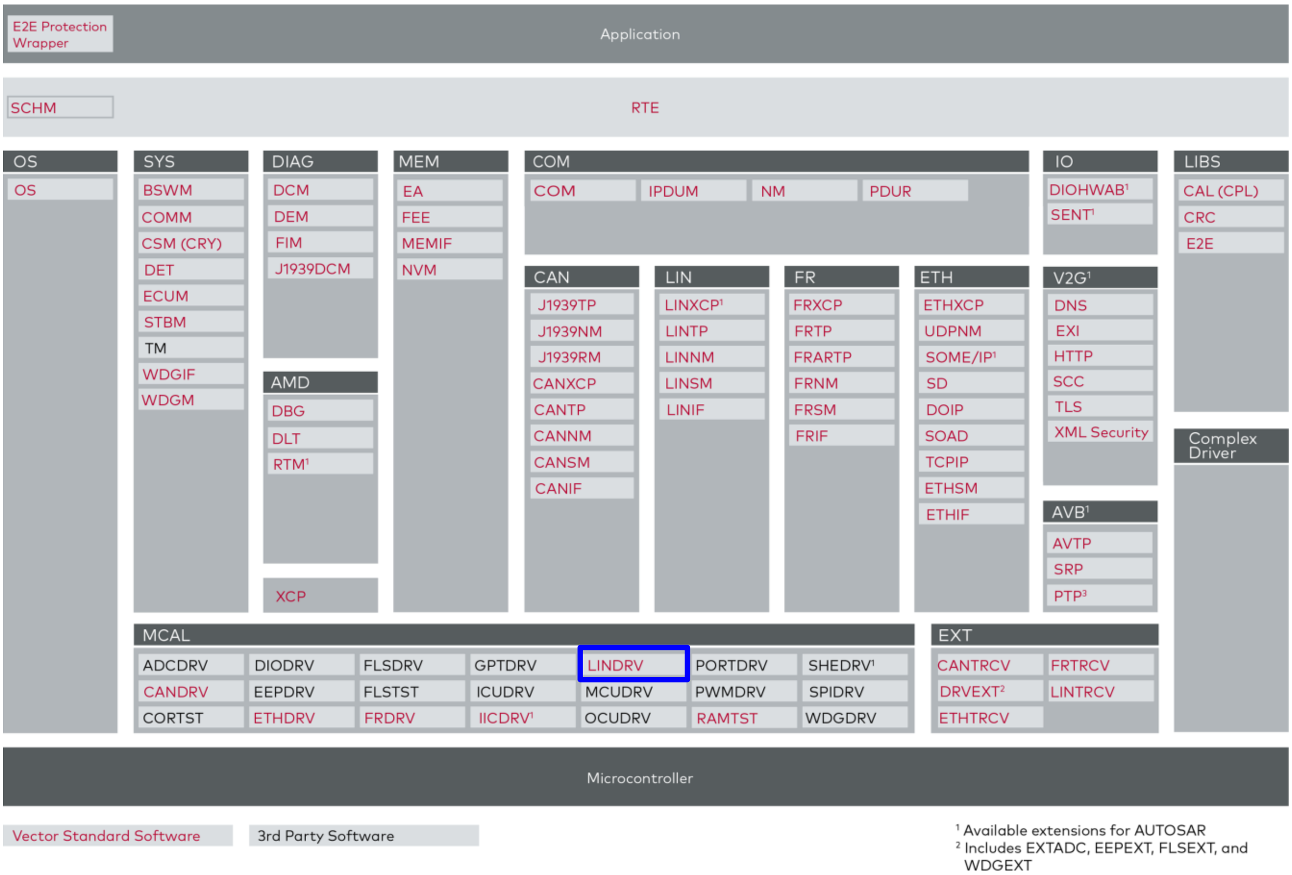

Figure 2-1 AUTOSAR 4.1 Architecture Overview

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

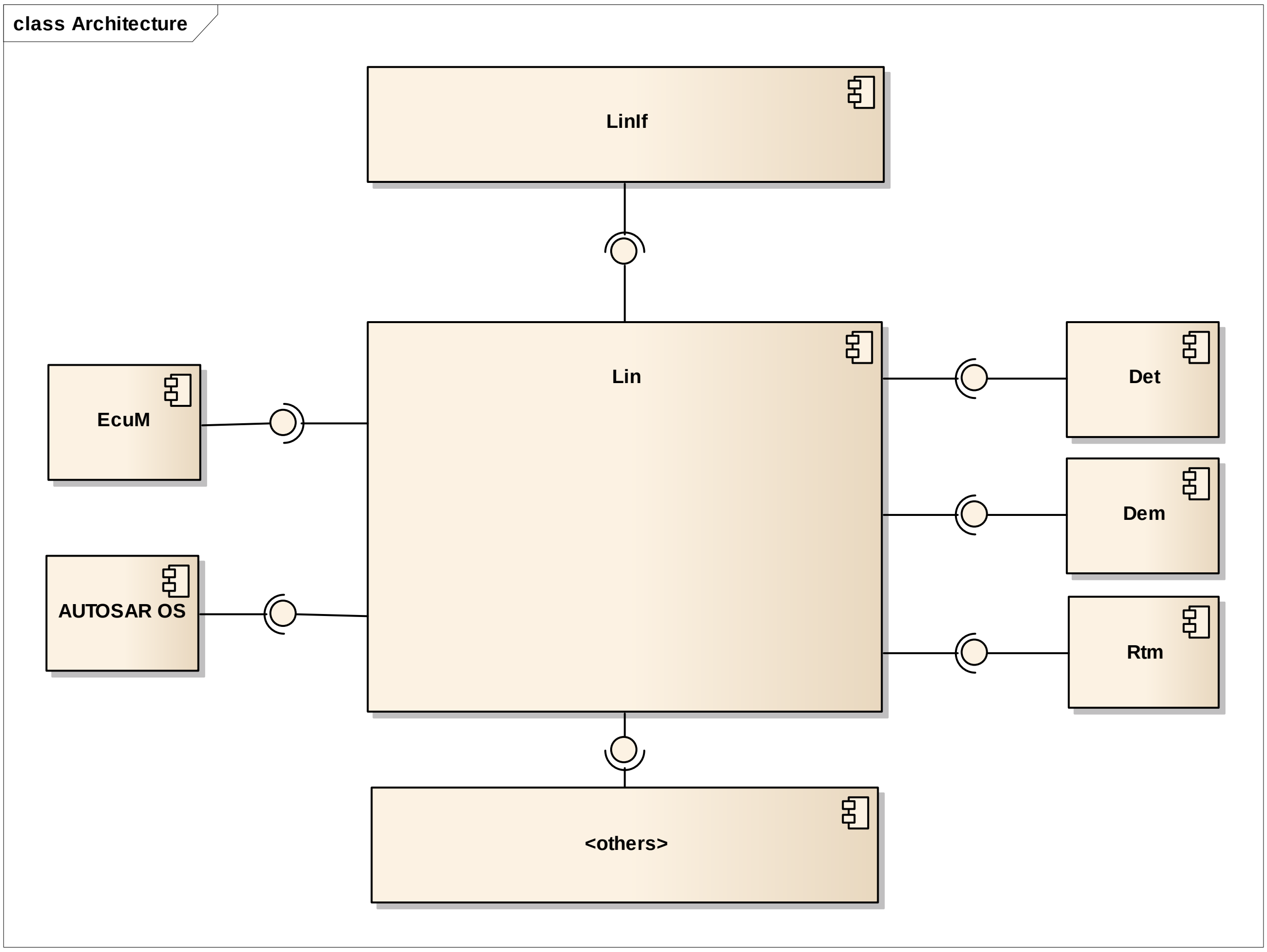

Figure 2-2 Interfaces to adjacent modules of the LIN

3. Functional Description

3.1 Features

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

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 LIN functionality beyond the AUTOSAR standard. The corresponding features are listed in the table

> Table 3-3 Features provided beyond the AUTOSAR standard

Please see also the hardware specific documentation [5] for further information about hardware specific additions or limitations.

The following features specified in [1] are supported:

Supported AUTOSAR Standard Conform Features
LIN Driver initialization
LIN frame transmission
LIN response reception
Mode change: sleep (normal and internal), wakeup (with notification) including reporting to EcuM and LinIf
Status reporting
Development error detection and reporting to DET
Production error detection and reporting to DEM/DET
Multi channel support
Version reporting

Table 3-1 Supported AUTOSAR standard conform features

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

Not Supported AUTOSAR Standard Conform Features
Multiple LIN Driver

Table 3-2 Not supported AUTOSAR standard conform features

The following features are provided beyond the AUTOSAR standard:

Features Provided Beyond The AUTOSAR Standard
Predefined Runtime Measurement points (RTM)

Table 3-3 Features provided beyond the AUTOSAR standard

The MICROSAR LIN Driver provides service functions for initialization, operation mode change and operation mode detection of the used LIN hardware. Service functions and callback functions are provided to detect wakeup by bus events and report them to the upper layer components.

3.2 Initialization

After power on the LIN hardware has to be initialized. Therefore the LIN Driver provides two service functions.

The function Lin_InitMemory() initializes all initialized variables of the LIN Driver. This variables need to be initialized before Lin_Init() is called. This function has only to be called after power on or reset before any other function in case initialized variables are not set after power on or reset (i.e. by the startup code). The function Lin_Init() initializes the channel independent states and channel independent LIN hardware registers. Also all LIN Driver channels which are selected by the generation tool are initialized by calling Lin_Init().

3.3 Wake up handling

Wakeup frame handling is only applicable in state sleep. A wakeup frame can be transmitted by calling the function Lin_Wakeup(). When calling Lin_WakeupInternal() no wakeup frame are transmitted. If a wakeup frame is received the EcuM is informed by calling the function EcuM_CheckWakeupEvent().

3.4 Sleep handling

Sleep mode frame handling is only applicable in state wake. A sleep mode frame can be transmitted by calling the function Lin_GoToSleep(). When calling Lin_GoToSleepInternal() the LIN Driver enters sleep mode without transmitting a sleep mode frame.

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 `LIN_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 LIN ID is 82.

The reported service IDs identify the services which are described in 5.3. The following table presents the service IDs and the related services:

Service ID	Service
0x00	Lin_Init
0x01	Lin_GetVersionInfo
0x02	Lin_InitChannel
0x03	Lin_DeInitChannel
0x04	Lin_SendFrame
0x06	Lin_GoToSleep
0x07	Lin_Wakeup
0x08	Lin_GetStatus
0x09	Lin_GoToSleepInternal
0x0A	Lin_WakeupValidation
0x0B	Lin_WakeupInternal
0x90	Lin_Interrupt

Table 3-4 Service IDs

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

Error Code		Description
0x00	LIN_E_UNINIT	API service used without module initialization.
0x02	LIN_E_INVALID_CHANNEL	API service used with an invalid or inactive channel parameter.
0x03	LIN_E_INVALID_POINTER	API service called with invalid configuration pointer
0x04	LIN_E_STATE_TRANSITION	Invalid state transition for the current state
0x05	LIN_E_PARAM_POINTER	API service called with a NULL pointer
0x10	LIN_E_TIMEOUT	Timeout caused by hardware error

Table 3-5 Errors reported to DET

3.5.1.1 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 and can be en-/disabled by means of en-/disabling the DET reporting via the parameter `LIN_DEV_ERROR_DETECT`.

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

Check Service	Config	versioninfo	Channel	PduInfoPtr	Lin_SduPtr
Lin_Init()	■				
Lin_GetVersionInfo()		■			
Lin_SendFrame()			■	■	
Lin_GoToSleep()			■		
Lin_Wakeup()			■		
Lin_WakeupInternal()			■		
Lin_GetStatus()			■		■
Lin_GoToSleepInternal()			■		
Lin_CheckWakeup ()			■		

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

3.5.2 Production Code Error Reporting

The only error reported from LIN Driver to DEM is described in the following table:

Error Code	Description
LIN_E_TIMEOUT	Timeout caused by hardware error

Table 3-7 Errors reported to DEM

Caution
By default, the LIN_E_TIMEOUT error is not reported!

Note
The availability of the LIN_E_TIMEOUT error depends on hardware. This means some hardware platforms don't report the LIN_E_TIMEOUT error.

If production error reporting is enabled the error is reported to the DEM using the service `Dem_ReportErrorStatus()` as specified in [3].

To enable the error reporting of LIN_E_TIMEOUT create a sub container 'LinDemEventParameterRefs' in the 'LinGlobalConfig' container (right click 'LinGlobalConfig' on DaVinci Configurator) and select a valid target reference for 'E TIMEOUT'. For disabling delete the 'LinDemEventParameterRefs' container.

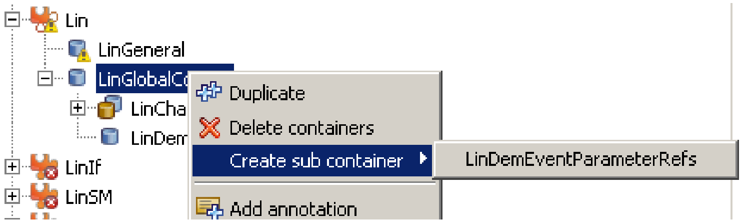

Figure 3-1 Creating LindemEventParameterRefs on DaVinci Configurator

If the error reporting of LIN_E_TIMEOUT is enabled it wouldn't be reported to DET.

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()`.

3.6 Predefined Runtime Measurement points

If enabled runtime measurement points are added to the BSW module code and are added to the module configuration of MICROSAR RTM. The available measurement points can be seen and configured in the RTM module configuration. Availability of the MICROSAR RTM module is a prerequisite to make use of the runtime measurement functionality. Runtime measurement points are possible on following functions:

Function	Id
`Lin_Init()`	RtmConf_RtmMeasurementPoint_Lin_Init
`Lin_Interrupt()`	RtmConf_RtmMeasurementPoint_Lin_Interrupt

Table 3-8 RTM points identifier.

Caution
Runtime measurement should be disabled in production code as measurement points may inflict additional runtime.

4. Integration

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

4.1 Scope of Delivery

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

4.1.1 Static Files

File Name	Description
Lin.h	Header containing the interface of the LIN Driver.
Lin.c	C code containing the functionality of the LIN Driver. This file is either delivered as source code or as library. If delivered as library the name is Lin.* with the compiler specific library extension.
Lin_Irq.c	This module contains the implementation of interrupt functions. This file is always delivered as source code.
Lin_GeneralTypes.h	Header allowing access to the commonly used type definitions of the LIN cluster.
Lin_Types.h	Header containing type definitions of the LIN Driver.

Table 4-1 Static files

4.1.2 Dynamic Files

The dynamic files are generated by the configuration tool [config tool].

File Name	Description
Lin_Cfg.h	Generated header adapting the LIN Driver to project requirements.
Lin_DrvGeneralTypes.h	Generated driver type definitions.
Lin_Lcfg.c	Generated C code containing tables with link time variables (RAM/ROM).
Lin_PBcfg.c	Generated C code containing tables with post build variables (ROM).

Table 4-2 Generated files

4.2 Include Structure

Following the include structure of the MICROSAR LIN Driver is given. The including of MemMap.h is not shown.

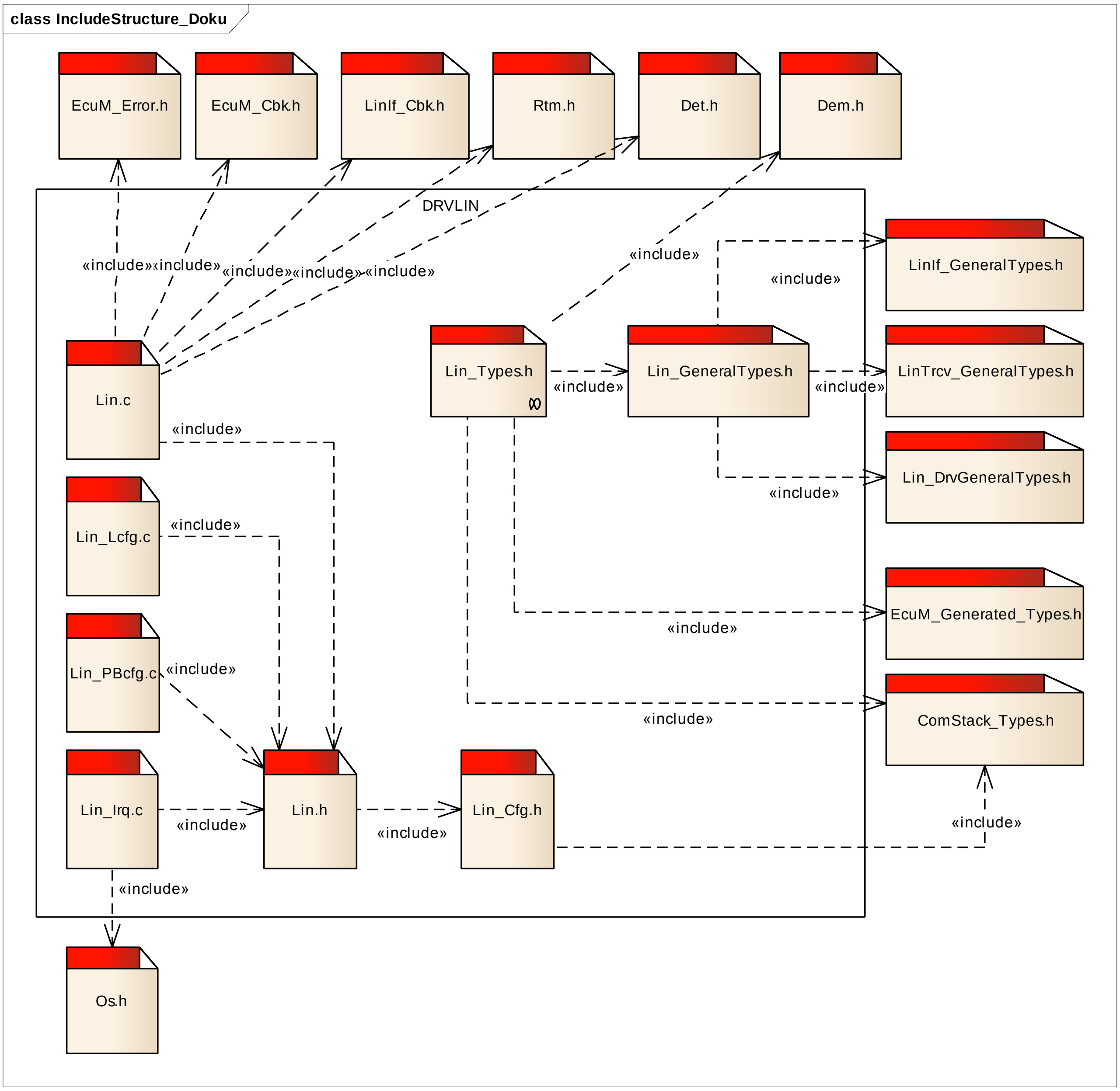

Figure 4-1 Include structure

Optional includes	Description
Dem.h	Dem.h is only included if reporting to DEM is enabled (LIN_E_TIMEOUT_TYPE_DEM is defined).
Det.h	Det.h is only included if reporting to DET is enabled (LIN_DEV_ERROR_DETECT == STD_ON).
Os.h	This header file is included if ‘Category 2’ is selected in the configuration tool.
Rtm.h	Rtm.h is only included if Runtime Measurement points are enabled (LIN_RUNTIME_MEASUREMENT_SUPPORT == STD_ON).

Table 4-3 Optional includes

4.3 Critical Sections

The MICROSAR LIN Driver is either running in interrupt context or is called from LinIf. The LinIf already prevents from interruption by means of exclusive areas. Thus no exclusive area handling is done within the LIN Driver.

4.4 Interrupt bits

The interrupt enable bits which aren’t located in the register address space of the LIN peripheral hardware module are generally not set by the LIN driver. The user must ensure that the bits are set.

5. API Description

For an interfaces overview please see Figure 2-2.

5.1 Type Definitions

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

Type Name	C-Type	Description	Value Range
Lin_u8PtrType	uint8	Pointer to a uint8 variable. Use for 'uint8**' definition in Lin_GetStatus	not applicable

Table 5-1 Type definitions

5.2 Interrupt Service Routines provided by LIN

Please see the LIN controller specific documentation for details.

5.3 Services provided by LIN

5.3.1 Lin_Interrupt

Prototype	
`void **Lin_Interrupt** (uint8 ChannelConfigIdx)`	
Parameter	
ChannelConfigIdx	ChannelConfig array index of the hardware channel (provided from ComStackLib).
Return code	
void	-
Functional Description	
Interrupt processing function.	
Particularities and Limitations	
The service ID of this function is LIN_SID_INTERRUPT_ID	
Call Context	
Called by Lin Driver	

Table 5-2 Lin_Interrupt

5.3.2 Lin_InitMemory

Prototype
`void **Lin_InitMemory** (void)`

Parameter	
-	
Return code	
void	-
Functional Description	
Sets the module state to uninitialized.	
Particularities and Limitations	
Function must be called in case LIN_VAR_ZERO_INIT variables are not initialized with 0 after reset (i.e. by startup code). This service function has to be called before Lin_Init() function.	
Call Context	
Called by upper layer.	

Table 5-3 Lin_InitMemory

5.3.3 Lin_Init

Prototype	
void **Lin_Init** (const Lin_ConfigType *Config)	
Parameter	
Config	Pointer to a selected configuration structure
Return code	
void	-
Functional Description	
Initializes the LIN module channel hardware and sets the state to initialize.	
Particularities and Limitations	
This service function has to be called before any other LIN driver function. The service ID of this function is LIN_SID_INIT_ID	
Call Context	
Called by upper layer.	

Table 5-4 Lin_Init

5.3.4 Lin_GetVersionInfo

Prototype	
void **Lin_GetVersionInfo** (Std_VersionInfoType *versioninfo)	
Parameter	
versioninfo	Pointer to where to store the version information of this module.
Return code	
void	-

Functional Description	
This service returns version information as decimal, vendor ID and AUTOSAR module ID of the component.	
Particularities and Limitations	
This function shall be pre compile time configurable On/Off by the configuration parameter: LIN_VERSION_INFO_API The service ID of this function is LIN_SID_GETVERSIONINFO_ID	
Call Context	
Called by upper layer.	

Table 5-5 Lin_GetVersionInfo

5.3.5 Lin_SendFrame

Prototype	
`Std_ReturnType` **`Lin_SendFrame`** `(uint8 Channel, Lin_PduType *PduInfoPtr)`	
Parameter	
Channel	LIN channel to be addressed
PduInfoPtr	Pointer to PDU containing the PID, Checksum model, Response type, Dl and SDU data pointer
Return code	
Std_ReturnType	E_OK: send command has been accepted E_NOT_OK: send command has not been accepted, development or production error occurred
Functional Description	
The function Lin_SendFrame generates a LIN frame on the addressed LIN channel.	
Particularities and Limitations	
The service ID of this function is LIN_SID_SENDFRAME_ID	
Call Context	
Called by upper layer.	

Table 5-6 Lin_SendFrame

5.3.6 Lin_GoToSleep

Prototype	
`Std_ReturnType` **`Lin_GoToSleep`** `(uint8 Channel)`	
Parameter	
Channel	LIN channel to be addressed
Return code	
Std_ReturnType	E_OK: Sleep command has been accepted E_NOT_OK: Sleep command has not been accepted, development or production error occurred
Functional Description	
The function Lin_GoToSleep transmits a goto-sleep-command on the addressed LIN channel.	

Particularities and Limitations
If supported by HW the LIN hardware unit maybe set to reduced power operation mode. The service ID of this function is LIN_SID_GOTOSLEEP_ID
Call Context
Called by upper layer.

Table 5-7 Lin_GoToSleep

5.3.7 Lin_GoToSleepInternal

Prototype	
`Std_ReturnType` **`Lin_GoToSleepInternal`** `(uint8 Channel)`	
Parameter	
Channel	LIN channel to be addressed
Return code	
Std_ReturnType	E_OK: Sleep command has been accepted E_NOT_OK: Sleep command has not been accepted, development or production error occurred
Functional Description	
Same function as Lin_GoToSleep but without sending a go-to-sleep-command on the bus.	
Particularities and Limitations	
The service ID of this function is LIN_SID_GOTOSLEEPINTERNAL_ID	
Call Context	
Called by upper layer.	

Table 5-8 Lin_GoToSleepInternal

5.3.8 Lin_Wakeup

Prototype	
`Std_ReturnType` **`Lin_Wakeup`** `(uint8 Channel)`	
Parameter	
Channel	LIN channel to be addressed
Return code	
Std_ReturnType	E_OK: Wake-up request has been accepted E_NOT_OK: Wake-up request has not been accepted, development or production error occurred
Functional Description	
Sends a wakeup frame on the on the addressed LIN channel.	
Particularities and Limitations	
The service ID of this function is LIN_SID_WAKEUP_ID	
Call Context	
Called by upper layer.	

Table 5-9 Lin_Wakeup

5.3.9 Lin_WakeupInternal

Prototype	
Std_ReturnType **Lin_WakeupInternal** (uint8 Channel)	
Parameter	
Channel	LIN channel to be addressed
Return code	
Std_ReturnType	E_OK: Wake-up request has been accepted E_NOT_OK: Wake-up request has not been accepted, development or production error occurred
Functional Description	
Sets the channel state to LIN_CH_OPERATIONAL without generating a wake up pulse.	
Particularities and Limitations	
The service ID of this function is LIN_SID_WAKEUPINTERNAL_ID	
Call Context	
Called by upper layer.	

Table 5-10 Lin_WakeupInternal

5.3.10 Lin_CheckWakeup

Prototype	
Std_ReturnType **Lin_CheckWakeup** (uint8 Channel)	
Parameter	
Channel	LIN channel to be addressed
Return code	
Std_ReturnType	E_OK: No error has occurred during execution of the API E_NOT_OK: An error has occurred during execution of the API
Functional Description	
After a wake up caused by LIN bus transceiver or LIN driver the function Lin_CheckWakeup will be called by the LIN Interface module to identify the corresponding LIN channel.	
Particularities and Limitations	
The service ID of this function is LIN_SID_CHECKWAKEUP_ID	
Call Context	
Called by upper layer.	

Table 5-11 Lin_CheckWakeup

5.3.11 Lin_GetStatus

Prototype	
Lin_StatusType **Lin_GetStatus** (uint8 Channel, Lin_u8PtrType *Lin_SduPtr)	
Parameter	
Channel	LIN channel to be addressed

Lin_SduPtr	Pointer to pointer to shadow buffer or memory mapped LIN Hardware receive buffer
Return code	
Lin_StatusType	Lin_StatusType: Information about the current message state.
Functional Description	
The function Lin_GetStatus shall return the current transmission, reception or operation status of the LIN driver.	
Particularities and Limitations	
The service ID of this function is LIN_SID_GETSTATUS_ID	
Call Context	
Called by upper layer.	

Table 5-12 Lin_GetStatus

5.4 Services used by LIN

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

Component	API
DET	Det_ReportError()
DEM	Dem_ReportErrorStatus()

EcuM	EcuM_CheckWakeup()
	EcuM_SetWakeupEvent()

RTM	Rtm_Start()
	Rtm_Stop()

LinIf	LinIf_WakeupConfirmation()

Table 5-13 Services used by the LIN

6. Configuration

6.1 Configuration Variants

The LIN supports the configuration variants

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

The configuration classes of the LIN parameters depend on the supported configuration variants. For their definitions please see the DrvLin_[HwPlatform]_bswmd.arxml file.

7. AUTOSAR Standard Compliance

7.1 Deviations

7.1.1 Deviations within API

API	Deviation	Reason
Lin_InitMemory	Additional API	Some compiler / startup codes do not support initialization of variables. This can now also be done by calling this function before calling the initialization function.

Table 7-1 Deviations within API

7.1.2 Deviations within features

Feature	Deviation	Reason
-	-	-

Table 7-2 Deviations within features

7.2 Additions/ Extensions

7.2.1 Memory initialization

To have an independent memory initialization for this BSW module the additional function Lin_InitMemory() was added. This must be called before normal initialization if initialized variables are not initialized during startup phase.

7.2.2 Additional header file ‘Lin_Types.h’

An additional header file has been introduced that contains the definitions of all types, which need to be known outside of the LIN Driver (i.e. the generated configuration files of the LIN Driver) and are not defined in Lin_GeneralTypes.h.

So the header ‘Lin_Types.h’ is included by the specified LIN Driver headers, no adaptations to other modules are necessary.

7.2.3 Polling Mode

In some uses cases it is not possible to use interrupts for the communication stack. If "Enable Polling Mode" is enabled, no LIN interrupts are called anymore. This feature is not available for all hardware platforms.

With polling mode enabled, the interrupt flags are polled periodically by the former interrupt function. The former interrupt function has to be called periodically with short cycle time by the application. Please note that calling this function with a cycle longer than 8 bit times may result in loss of bus events.

7.3 Limitations

7.3.1 Controller

Please refer to the hardware specific documentation [5] for further information about supported controllers.

7.3.2 Compiler

Please refer to the hardware specific documentation [5] for further information about supported compilers.

8. Glossary and Abbreviations

8.1 Glossary

Term	Description
Buffer	A buffer in a memory area normally in the RAM. It is an area, that the application has reserved for data storage.
Channel	A channel defines the assignment (1:1) between a physical communication interface and a physical layer on which different modules are connected to (either CAN or LIN). 1 channel consists of 1..X network(s).
Component	CAN Driver, Network Management ... are software COMPONENTS in contrast to the expression module, which describes an ECU.
EAD	Embedded Architecture Designer; generation tool for MICROSAR components
Electronic Control Unit	Also known as ECU. Small embedded computer system consisting of at least one CPU and corresponding periphery which is placed in one housing.
Interrupt	Processor-specific event which can interrupt the execution of a current program section.
Interrupt service routine	The function used for direct processing of an interrupt.
Post-build	This type of configuration is possible after building the software module or the ECU software. The software may either receive parameters of its configuration during the download of the complete ECU software resulting from the linkage of the code, or it may receive its configuration file that can be downloaded to the ECU separately, avoiding a re-compilation and re-build of the ECU software modules. In order to make the post-build time re-configuration possible, the re-configurable parameters shall be stored at a known memory location of ECU storage area.
Sleep mode	An activity in which a node enters a state of maximum protocol inactivity with the ability to detect a wake up signal.
Transceiver	A transceiver adapts the physical layer to the communication interface.

Table 8-1 Glossary

8.2 Abbreviations

Term	Description
	Automotive Open System Architecture
API	Application Program Interface, for OSEK: The description of the user interface to the operating system, communications and network management functions.
AUTOSAR	Automotive Open System Architecture

BSW	Basic Software
CAN	Controller Area Network protocol originally defined for use as a communication network for control applications in vehicles.
DEM	Diagnostic Event Manager
DET	Development Error Tracer
EAD	Embedded Architecture Designer
ECU	Electronic Control Unit
F9	undefined
HIS	Hersteller Initiative Software
HW	Hardware
ID	Identifier (e.g. Identifier of a CAN message)
ISR	Interrupt Service Routine
LIN	Local Interconnect Network
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
PDU	undefined
PPort	Provide Port
RPort	Require Port
RTE	Runtime Environment
SDU	undefined
SPI	SPI Driver
SRS	Software Requirement Specification
SWC	Software Component
SWS	Software Specification
VI	undefined
RTM	Runtime Measurement

Table 8-2 Abbreviations

9. Contact

Visit our website for more information on

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

www.vector.com