Document Title	Specification of LIN Driver
Document Category	Software specification (SWS)
Document UID	72
Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R26-11

Document Change History

Date	Release	Changed by	Description
2026-11-26	R26-11	AUTOSAR Release Management	• Removed GetVersionInfo API and parameters • Added for `Lin_MainFunction` supporting polling mode • Editorial Changes
2025-11-27	R25-11	AUTOSAR Release Management	• Clarification of an ongoing frame transmission and a new frame transmission is requested • Editorial Changes
2024-11-27	R24-11	AUTOSAR Release Management	• Clarification of configuration data structure for multicore support • Editorial Changes
2023-11-23	R23-11	AUTOSAR Release Management	• LIN_E_TIMEOUT removed as Production Error • Misleading note regarding Lin_CheckWakeup removed • Editorial Changes
2022-11-24	R22-11	AUTOSAR Release Management	• Clarification of Lin_CheckWakeup return values • Editorial changes
2021-11-25	R21-11	AUTOSAR Release Management	• Clarification of configuration parameter LinChannelWakeupSupport • Cleanup of Error classification chapter • Header file for EcuM_CheckWakeup changed
2020-11-30	R20-11	AUTOSAR Release Management	• Modified SWS_Lin_00266, Lin_GeneralTypes removed • Editorial change

△

2019-11-28	R19-11	AUTOSAR Release Management	• MCALMulticoreDistribution (CONC_639) • Changed Document Status from Final to published
2018-10-31	4.4.0	AUTOSAR Release Management	• LIN Slave support (CONC_634) • MCALMulticoreDistribution (CONC_639) as DRAFT • Replace references to LIN 2.1 by ISO 17987:2016 (with no functional modification) • Header file cleanup • Minor corrections / clarifications / editorial changes; For details please refer to the ChangeDocumentation
2017-12-08	4.3.1	AUTOSAR Release Management	• Minor corrections / clarifications / editorial changes • Resolve inconsistency on channel state upon initialization
2016-11-30	4.3.0	AUTOSAR Release Management	• Updated tracing information • Removed chapter 'Variants'
2015-07-31	4.2.2	AUTOSAR Release Management	• Chapter 6 "Requirements traceability" clean up • Reference to DET are named as "Default" Error Tracer instead of "Development" Error Tracer • Dependency on Module DET listed in Chapter 5 is linked to SWS_Lin_00048 instead of SWS_Lin_00052
2014-10-31	4.2.1	AUTOSAR Release Management	• Replaced SWS_Lin_00064 with SWS_Lin_00268

▽

△

2014-03-31	4.1.3	AUTOSAR Release Management	• Removed SWS_Lin_00243. • Modified SWS_Lin_00237, SWS_Lin_00058, SWS_Lin_00266, SWS_Lin_00255, SWS_Lin_00256, SWS_Lin_00258, SWS_Lin_00259, SWS_Lin_00260. • Updated Figure 7-1. • Removed references to SWS_Lin_00073 and SWS_Lin_00034 from chapter 6.
2013-10-31	4.1.2	AUTOSAR Release Management	• Removed outdated SWS_Lin_00109, SWS_Lin_00136 and SWS_Lin_00132. • Import of SWS_Lin_184 from R3.2.2 • Wake-up LIN Functionality updated • New API Lin_WakeupInternal added. See chapter 8.3.2.5 • Added the following type definition (with SWS item ID) to chapter 8: • Lin_FrameCsModelType • Lin_FrameDlType • Lin_FramePidType • Lin_FrameResponseType • Lin_PduType • Lin_StatusType • Editorial changes • Removed chapter(s) on change documentation
2013-03-15	4.1.1	AUTOSAR Administration	• Specified LIN_E_TIMEOUT as production error • Shifted all types used by other modules to Lin_GeneralTypes.h • Revised configuration container LinDemEventParamterRefs • Some minor updates
2011-12-22	4.0.3	AUTOSAR Administration	• Changed error reporting • Improved wake-up handling • Corrected call of Lin_Init

▽

△

2010-09-30	3.1.5	AUTOSAR Administration	• Introduce Lin_GeneralTypes.h • Add missing DET error code (NULL pointer error) • Remove instance ID from Lin_GetVersionInfo API • Correct naming of "WakeUp" to "Wakeup" • Further maintenance for R4.0.2: see chapter 12
2010-02-02	3.1.4	AUTOSAR Administration	• Support of advanced LIN controllers (combination of Lin_SendHeader and Lin_SendResponse to Lin_SendFrame) • Integrating LIN channel initialization in LIN module initialization • Further maintenance for R4.0: see chapter 11 • Legal disclaimer revised
2008-08-13	3.1.1	AUTOSAR Administration	• Legal disclaimer revised
2007-12-21	3.0.1	AUTOSAR Administration	• Editorial Changes • Tables generated in Chapter 8 and 10 • Document meta information extended • Small layout adaptations made
2007-01-24	2.1.15	AUTOSAR Administration	• Lin Transceiver Wake Up validation function added • Incorporate Feedback from Validator2 • Updated Chapter 10.2 according to the Specification of ECU Configuration Parameters • Legal disclaimer revised • Release Notes added • "Advice for users" revised • "Revision Information" added
2006-05-16	2.0	AUTOSAR Administration	• Initial release

Disclaimer

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

The material contained in this work is protected by copyright and other types of intellectual property rights. The commercial exploitation of the material contained in this work requires a license to such intellectual property rights.

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

The work has been developed for automotive applications only. It has neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Table of Contents

1 Introduction and functional overview 10
1.1 Scope . . . 10
1.2 Architectural overview . . . 10
2 Acronyms, abbreviations and glossary 12
2.1 Acronyms and abbreviations . . . 12
2.2 Glossary . . . 12
2.3 LIN hardware unit classification . . . 13
3 Related documentation 15
3.1 Input documents & related standards and norms . . . 15
3.2 Related specification . . . 15
4 Constraints and assumptions 16
4.1 Limitations . . . 16
4.1.1 Driver scope . . . 16
4.2 Applicability to car domains . . . 16
5 Dependencies to other modules 17
5.1 File structure . . . 17
5.1.1 Code file structure . . . 17
5.1.2 Header file structure . . . 18
6 Requirements Tracing 19
7 Functional specification 20
7.1 General Requirements . . . 20
7.2 LIN driver and Channel Initialization . . . 21
7.2.1 Background & Rationale . . . 21
7.2.2 Requirements . . . 21
7.2.3 State diagrams . . . 21
7.3 Frame processing . . . 23
7.3.1 Background & Rationale . . . 23
7.3.2 Requirements . . . 24
7.3.2.1 LIN Master specific . . . 24
7.3.2.2 LIN Slave specific . . . 25
7.3.2.3 Common . . . 26
7.3.3 Data Consistency . . . 27
7.3.3.1 Transmit Data Consistency: . . . 27
7.3.3.2 Receive Data Consistency: . . . 27
7.3.4 Data byte mapping . . . 28
7.4 Sleep and wake-up functionality . . . 28
7.4.1 Background & Rationale . . . 28
7.4.2 Requirements . . . 29

7.5 Error Classification . . . 29
7.5.1 Development Errors . . . 30
7.5.2 Runtime Errors . . . 30
7.5.3 Production Errors . . . 30
7.5.3.1 LIN_E_TIMEOUT . . . 30
7.6 Security Events . . . 31
8 API specification 32
8.1 Imported types . . . 32
8.2 Type definitions . . . 32
8.2.1 Lin_ConfigType . . . 32
8.2.2 Lin_FramePidType . . . 33
8.2.3 Lin_FrameCsModelType . . . 33
8.2.4 Lin_FrameResponseType . . . 33
8.2.5 Lin_FrameDlType . . . 34
8.2.6 Lin_PduType . . . 34
8.2.7 Lin_StatusType . . . 35
8.2.8 Lin_SlaveErrorType . . . 36
8.3 Function definitions . . . 36
8.3.1 Services affecting the complete LIN hardware unit . . . 36
8.3.1.1 Lin_Init . . . 36
8.3.1.2 Lin_CheckWakeup . . . 37
8.3.2 Services affecting a single LIN channel . . . 38
8.3.2.1 Lin_SendFrame . . . 38
8.3.2.2 Lin_GoToSleep . . . 40
8.3.2.3 Lin_GoToSleepInternal . . . 41
8.3.2.4 Lin_Wakeup . . . 42
8.3.2.5 Lin_WakeupInternal . . . 43
8.3.2.6 Lin_GetStatus . . . 44
8.4 Scheduled functions . . . 46
8.4.1 Lin_MainFunction . . . 46
8.5 Expected interfaces . . . 46
8.5.1 Mandatory interfaces . . . 46
8.5.2 Optional interfaces . . . 47
8.5.3 Callout interfaces . . . 47
8.5.4 Configurable interfaces . . . 48
9 Sequence diagrams 49
9.1 Receiving a LIN Frame . . . 49
9.1.1 LIN Master . . . 49
9.1.2 LIN Slave . . . 50

10 Configuration specification 51
10.1 How to read this chapter . . . 51
10.2 Containers and configuration parameters . . . 51
10.2.1 Lin . . . 52
10.2.2 LinGeneral . . . 53
10.2.3 LinChannel . . . 55
10.2.4 LinGlobalConfig . . . 58
10.2.5 LinDemEventParameterRefs . . . 58
10.3 Published Information . . . 59
A Not applicable requirements 60
B Change history of AUTOSAR traceable items 61
B.1 Traceable item history of this document according to AUTOSAR Release R26-11 . . . 61
B.1.1 Added Constraints in R26-11 . . . 61
B.1.2 Changed Constraints in R26-11 . . . 61
B.1.3 Deleted Constraints in R26-11 . . . 61
B.1.4 Added Specification Items in R26-11 . . . 61
B.1.5 Changed Specification Items in R26-11 . . . 61
B.1.6 Deleted Specification Items in R26-11 . . . 62
B.2 Traceable item history of this document according to AUTOSAR Release R25-11 . . . 62
B.2.1 Added Constraints in R25-11 . . . 62
B.2.2 Changed Constraints in R25-11 . . . 62
B.2.3 Deleted Constraints in R25-11 . . . 62
B.2.4 Added Specification Items in R25-11 . . . 62
B.2.5 Changed Specification Items in R25-11 . . . 62
B.2.6 Deleted Specification Items in R25-11 . . . 63
B.3 Traceable item history of this document according to AUTOSAR Release R24-11 . . . 63
B.3.1 Added Constraints in R24-11 . . . 63
B.3.2 Changed Constraints in R24-11 . . . 63
B.3.3 Deleted Constraints in R24-11 . . . 63
B.3.4 Added Specification Items in R24-11 . . . 63
B.3.5 Changed Specification Items in R24-11 . . . 63
B.3.6 Deleted Specification Items in R24-11 . . . 64
B.4 Traceable item history of this document according to AUTOSAR Release R23-11 . . . 64
B.4.1 Added Constraints in R23-11 . . . 64
B.4.2 Changed Constraints in R23-11 . . . 64
B.4.3 Deleted Constraints in R23-11 . . . 64
B.4.4 Added Specification Items in R23-11 . . . 65
B.4.5 Changed Specification Items in R23-11 . . . 68

B.4.6 Deleted Specification Items in R23-11 69

1 Introduction and functional overview

This specification specifies the functionality, API and the configuration of the AUTOSAR Basic Software module LIN driver.

1.1 Scope

The base for this document is the ISO 17987 specifications [1]. It is assumed that the reader is familiar with this specification. This document will not describe ISO 17987 LIN functionality again.

The LIN driver applies to ISO 17987 master and slave nodes. The LIN implementation in AUTOSAR deviates from the ISO 17987 specifications as described in this specification of LIN driver, but there will be no change in the behavior on the LIN bus. It is the intention to be able to reuse all existing LIN nodes together with the AUTOSAR LIN implementation (i.e. the LIN driver).

[SWS_Lin_00063] ⌈It is intended to support the complete range of LIN hardware from a simple SCI/UART to a complex LIN hardware controller. Using a SW-UART implementation is out of the scope.⌋

For a closer description of the LIN hardware unit, see chapter 2.3.

1.2 Architectural overview

The LIN driver is part of the microcontroller abstraction layer (MCAL), performs the hardware access and offers a hardware independent API to the upper layer. The only upper layer, which has access to the LIN driver, is the LIN Interface.

A LIN driver can support more than one channel. This means that the LIN driver can handle one or more LIN channels as long as they are belonging to the same LIN hardware unit.

In the example below three different LIN drivers are connected to the LIN interface. However, one LIN driver is the most common configuration.

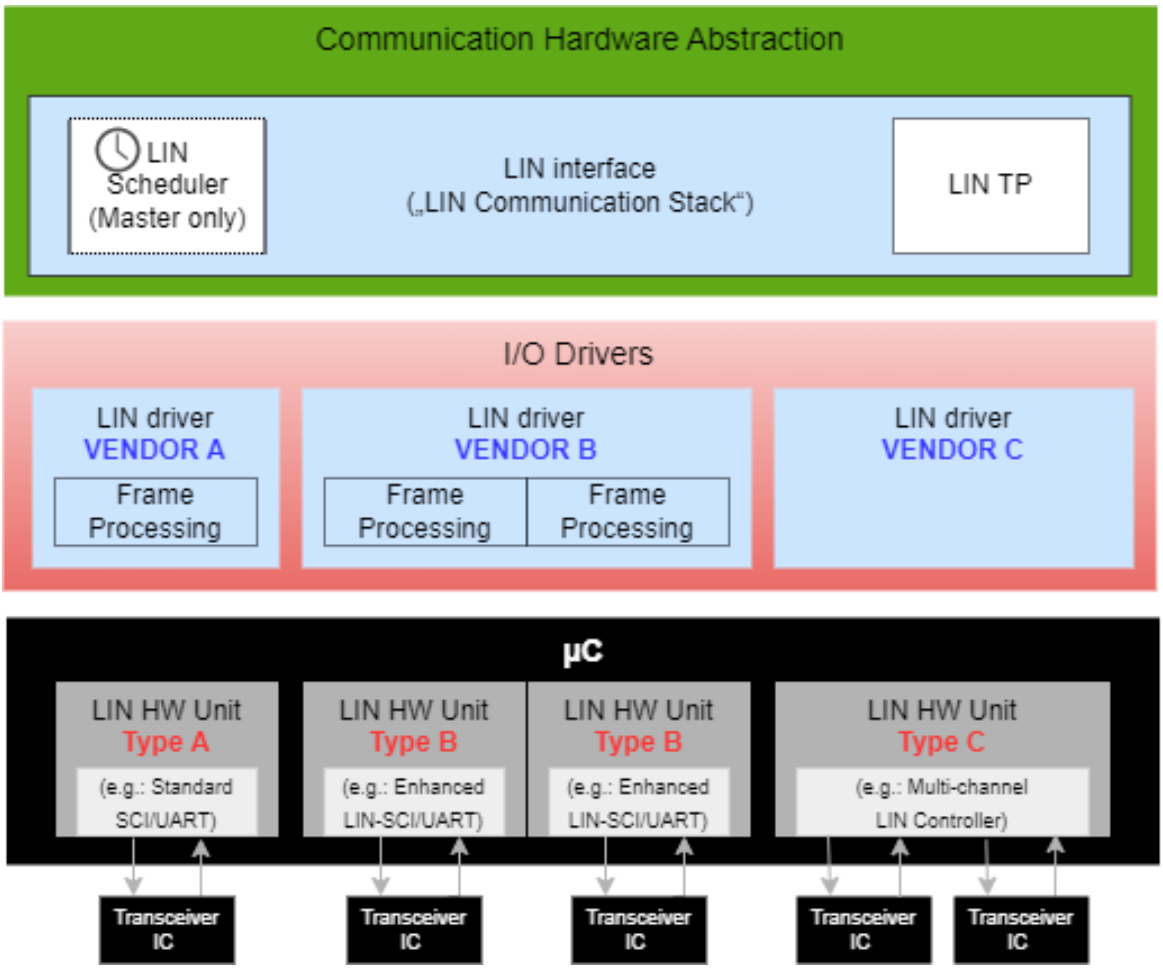

Figure 1.1: Overview LIN Software Architecture Layering

2 Acronyms, abbreviations and glossary

2.1 Acronyms and abbreviations

Acronyms, abbreviations and definitions that have a local scope for the LIN driver and therefore are not contained in the [2, AUTOSAR glossary] must appear here.

Acronym:	Description:
PID	Protected ID (as defined by [1])
PLL	Phase-Locked Loop
RX	Reception
SCI	Serial Communication Interface
SFR	Special Function Register
SPAL	Standard Peripheral Abstraction Layer
SRS	Software Requirement Specification
TX	Transmission
UART	Universal Asynchronous Receiver Transmitter

Table 2.1: Acronyms used in the scope of this Document

Abbreviation:	Description:
Id	Identifier

Table 2.2: Abbreviations used in the scope of this Document

2.2 Glossary

Besides AUTOSAR terminology this document also uses terms defined in the ISO 17987 specifications [1], e.g. LIN frame, header and message.

Glossary:	Description:
enumeration	This can be in "C" programming language an enum or a #define.
LIN channel	The LIN channel entity interlinks the ECUs of a LIN cluster physically: An ECU is part of a LIN cluster if it contains one LIN controller that is connected to one LIN channel of the LIN cluster. An ECU is allowed to connect to a particular LIN cluster through one channel only.
LIN cluster	As defined by [1]: "A cluster is the LIN bus wire plus all the nodes."
LIN controller	A dedicated LIN hardware with a build Frame processing state machine. A hardware which is capable to connect to several LIN clusters is treated as several LIN controllers.
LIN frame	As defined by [1]: "All information is sent packed as frames; a frame consist of the header and a response."
LIN frame processor	Frame processing implies the complete LIN frame handling. Implementation could be achieved as software emulated solution or with a dedicated LIN controller.
LIN hardware unit	A LIN hardware unit may drive one or multiple LIN channels to control one or multiple LIN clusters.
LIN header	As defined by [1]: "A header is the first part of a frame; it is always sent by the master."
LIN node	As defined by [1]: "Loosely speaking, a node is an ECU. However, a single ECU may be connected to multiple LIN clusters."
LIN response	As defined by [1]: "A LIN frame consists of a header and a response. Also called a Frame response."

Table 2.3: Glossary used in the scope of this Document

2.3 LIN hardware unit classification

The on-chip LIN hardware unit combines one or several LIN channels.

The following figure shows a classification of different LIN hardware types connected to multiple LIN physical channels:

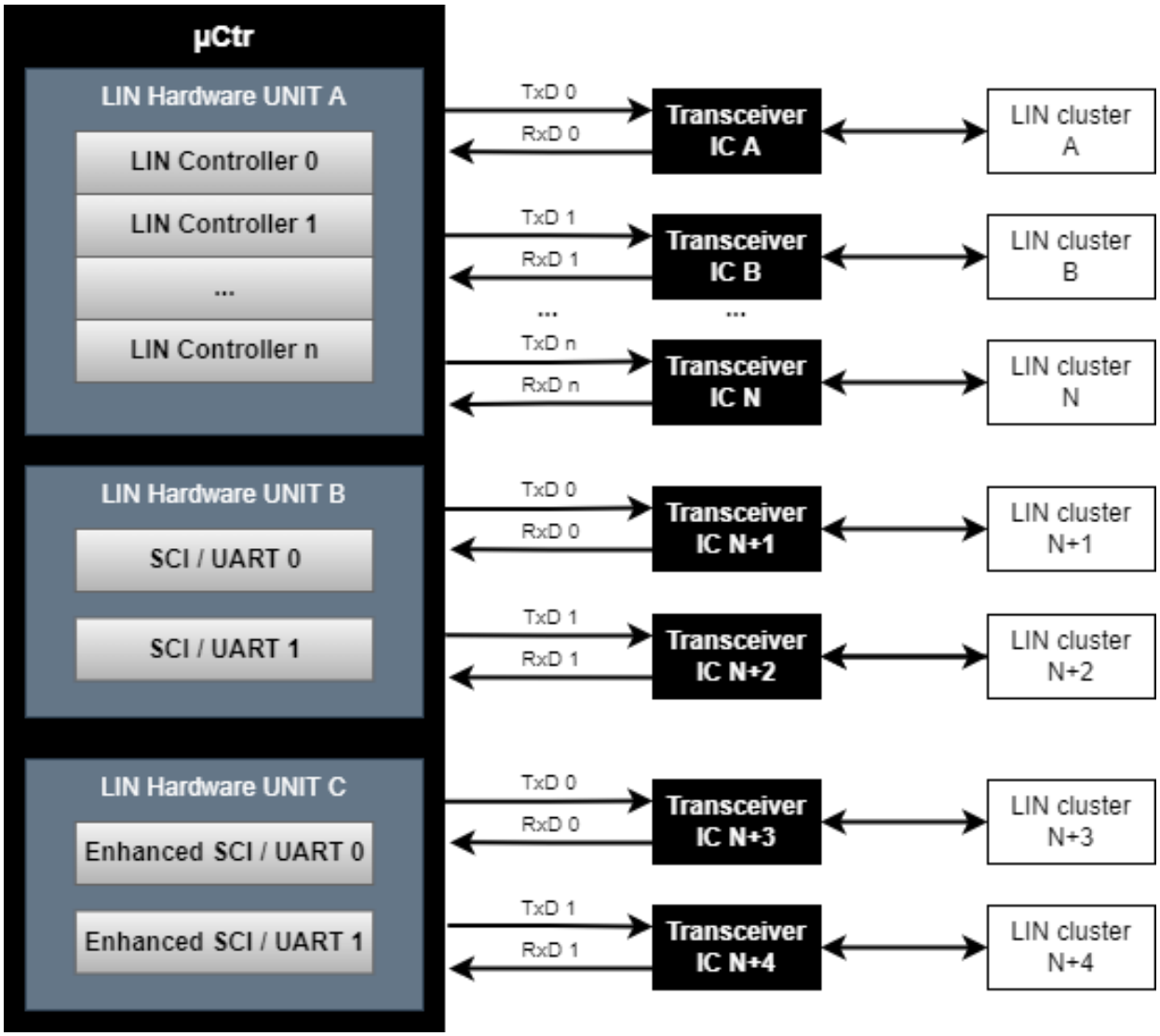

Figure 2.1: LIN hardware unit classification

3 Related documentation

3.1 Input documents & related standards and norms

[1] ISO 17987:2016 (all parts), Road vehicles – Local Interconnect Network (LIN)
https://www.iso.org

[2] Glossary
AUTOSAR_FO_TR_Glossary

[3] General Specification of Basic Software Modules
AUTOSAR_CP_SWS_BSWGeneral

[4] Specification of LIN Interface
AUTOSAR_CP_SWS_LINInterface

[5] Specification of MCU Driver
AUTOSAR_CP_SWS_MCUDriver

[6] Specification of Default Error Tracer
AUTOSAR_CP_SWS_DefaultErrorTracer

[7] Specification of Diagnostic Event Manager
AUTOSAR_CP_SWS_DiagnosticEventManager

[8] Specification of Operating System
AUTOSAR_CP_SWS_OS

[9] General Requirements on Basic Software Modules
AUTOSAR_CP_RS_BSWGeneral

[10] System Template
AUTOSAR_CP_TPS_SystemTemplate

3.2 Related specification

AUTOSAR provides a General Specification on Basic Software modules [3, SWS_BSWGeneral], which is also valid for LIN Driver.

Thus, the specification SWS_BSWGeneral shall be considered as additional and required specification for LIN Driver.

4 Constraints and assumptions

4.1 Limitations

Only one LIN channel of an ECU is allowed to connect to a particular LIN cluster. Unless there are unused (not connected) channels in the ECU, the number of LIN channels is equal to the number of LIN clusters.

4.1.1 Driver scope

[SWS_Lin_00045]

Upstream requirements: SRS_BSW_00347

⌈One LIN driver provides access to one LIN hardware unit type (simple UART or dedicated LIN hardware) that may consist of several LIN channels.⌋

[SWS_Lin_00201] ⌈For different LIN hardware units a separate LIN driver needs to be implemented. It is up to the implementer to adapt the driver to the different instances of similar LIN channels.⌋

[SWS_Lin_00177] ⌈In case several LIN driver instances (of same or different vendor) are implemented in one ECU the file names, API names, and published parameters must be modified such that no two definitions with the same name are generated. The name shall be extended according to [SRS_BSW_00347] with a Vendor Id (needed to distinguish LIN drivers from different vendors) and a Vendor specific name (needed to distinguish different hardware units implemented by one Vendor): <Module abbreviation>_<Vendor Id>_<Vendor specific name>.⌋

The LIN Interface is responsible for calling the correct function. The necessary information shall be given in an XML file during configuration. See [4, Specification of LIN Interface] for description how the LIN Interface handles several LIN drivers.

4.2 Applicability to car domains

This specification is applicable to all car domains, where LIN is used.

5 Dependencies to other modules

Module MCU [5, Specification of MCU Driver]

The hardware of the internal LIN hardware unit depends on the system clock, prescaler(s) and PLL. Hence, the length of the LIN bit timing depends on the clock settings made in module MCU.

The LIN driver module will not take care of setting the registers that configure the clock, prescaler(s) and PLL (e.g. switching on/off the PLL) in its init functions. The MCU module must do this.

Module Port

The Port driver configures the port pins used for the LIN driver as input or output. Hence, the Port driver has to be initialized prior to the use of LIN functions. Otherwise, LIN driver functions will exhibit undefined behavior.

Module DET [6, Specification of Default Error Tracer]

In development mode, the Lin module reports development error through the `Det_ReportError` function of module DET. (see [SWS_Lin_00048])

Module DEM [7, Specification of Diagnostic Event Manager]

The Lin module reports production errors to the Diagnostic Event Manager. (see [SWS_Lin_00058])

OS [8, Specification of Operating System]

The LIN driver uses interrupts and therefore there is a dependency on the OS, which configures the interrupt sources.

LIN driver Users

The LIN Interface [4, Specification of LIN Interface] is the only user of the LIN driver services.

5.1 File structure

5.1.1 Code file structure

[SWS_Lin_00268] ⌈The code file structure shall not be defined within this specification.⌋

5.1.2 Header file structure

[SWS_Lin_00054]

Upstream requirements: SRS_BSW_00302

⌈The file `Lin.h` only contains external declarations of constants, global data, type definitions and services that are specified in LIN driver SWS.⌋

[SWS_Lin_00207] ⌈Constants, global data types and functions that are only used by LIN driver internally, are declared in `Lin.c`.⌋

6 Requirements Tracing

The following tables reference the requirements specified in [9] and links to the fulfillment of these. Please note that if column “Satisfied by” is empty for a specific requirement this means that this requirement is not fulfilled by this document.

Requirement	Description	Satisfied by
[SRS_BSW_00101]	The Basic Software Module shall be able to initialize variables and hardware in a separate initialization function	[SWS_Lin_00006]
[SRS_BSW_00159]	All modules of the AUTOSAR Basic Software shall support a tool based configuration	[SWS_Lin_00029]
[SRS_BSW_00164]	The Implementation of interrupt service routines shall be done by the Operating System, complex drivers or modules	[SWS_Lin_00155]
[SRS_BSW_00167]	All AUTOSAR Basic Software Modules shall provide configuration rules and constraints to enable plausibility checks	[SWS_Lin_00039]
[SRS_BSW_00302]	All AUTOSAR Basic Software Modules shall only export information needed by other modules	[SWS_Lin_00054]
[SRS_BSW_00306]	AUTOSAR Basic Software Modules shall be compiler and platform independent	[SWS_Lin_00055]
[SRS_BSW_00308]	AUTOSAR Basic Software Modules shall not define global data in their header files, but in the C file	[SWS_Lin_00055]
[SRS_BSW_00309]	All AUTOSAR Basic Software Modules shall indicate all global data with read-only purposes by explicitly assigning the const keyword	[SWS_Lin_00055]
[SRS_BSW_00323]	All AUTOSAR Basic Software Modules shall check passed API parameters for validity	[SWS_Lin_00048]
[SRS_BSW_00327]	Error values naming convention	[SWS_Lin_00048]
[SRS_BSW_00337]	Classification of development errors	[SWS_Lin_00048]
[SRS_BSW_00345]	BSW Modules shall support pre-compile configuration	[SWS_Lin_00013]
[SRS_BSW_00347]	A Naming seperation of different instances of BSW drivers shall be in place	[SWS_Lin_00045]
[SRS_BSW_00385]	List possible error notifications	[SWS_Lin_00048]
[SRS_BSW_00404]	BSW Modules shall support post-build configuration	[SWS_Lin_00013]
[SRS_BSW_00405]	BSW Modules shall support multiple configuration sets	[SWS_Lin_00013]
[SRS_BSW_00406]	API handling in uninitialized state	[SWS_Lin_00006]
[SRS_BSW_00458]	Classification of production errors	[SWS_Lin_00290]

Table 6.1: Requirements Tracing

7 Functional specification

The LIN driver module is required to manage the hardware dependent aspects of communication via any LIN cluster attached to the node the driver resides in.

This includes accepting header data for transmission onto the bus, response frame data to transmit, the retrieval of header information and of response frame data intended for the node.

The need for sleep mode management of both the node and of the cluster exists. This implies the ability to detect and generate a 'wake-up' pulse as defined in the ISO 17987 specifications. If the underlying hardware supports a low-power mode then entering and exiting from that state is included.

7.1 General Requirements

The Lin module is a Basic Software Module that has direct access to hardware resources.

[SWS_Lin_00005] ⌈The Lin module shall conform to the ISO 17987 specifications [1]. This applies to ISO 17987 LIN Master and Slave nodes.⌋

[SWS_Lin_00055]

Upstream requirements: SRS_BSW_00306, SRS_BSW_00308, SRS_BSW_00309

⌈The Lin module shall fulfill all design and implementation guidelines as described in [3, SWS_BSWGeneral].⌋

[SWS_Lin_00155]

Upstream requirements: SRS_BSW_00164

⌈The Lin module shall implement the ISRs for all LIN hardware unit interrupts that are needed.⌋

[SWS_Lin_00156] ⌈The Lin module shall ensure that all unused interrupts are disabled.⌋

[SWS_Lin_00157] ⌈The Lin module shall reset the interrupt flag at the end of the ISR (if not done automatically by hardware).⌋

The Lin module shall not configure the interrupt (i.e. priority) nor set the vector table entry.

7.2 LIN driver and Channel Initialization

7.2.1 Background & Rationale

Before communication can be started on a LIN bus, both the LIN driver and the relevant LIN channel must be initialized.

The driver initialization (see `Lin_Init`) handles all aspects of initialization that are of relevance to all channels present in the LIN hardware unit. This may include any static variables or hardware register settings common to all LIN channels that are available. Additionally each channel must also be initialized according to the configuration supplied. This will for example include (but is not limited to) the baud rate over the bus.

[SWS_Lin_00225] ⌈There must be at least one statically defined configuration set available for the LIN driver. When the EcuM invokes the initialization function, it has to provide a specific pointer to the configuration that it wishes to use.⌋

7.2.2 Requirements

The Lin module shall not initialize or configure LIN channels, which are not used.

The Lin module shall allow the environment to select between different static configuration data at runtime.

[SWS_Lin_00011] ⌈The Lin module's configuration shall include a data communication rate set as defined by static configuration data.⌋

[SWS_Lin_00013]

Upstream requirements: SRS_BSW_00345, SRS_BSW_00404, SRS_BSW_00405

⌈The Lin module's configuration data, intended for hardware registers, shall be stored as hardware specific data structures in ROM (see `Lin_ConfigType`).⌋

[SWS_Lin_00014] ⌈Each LIN PID shall be associated with a checksum model (either 'enhanced' where the PID is included in the checksum, or 'classic' where only the response data is check-summed) (see `Lin_PduType`).⌋

[SWS_Lin_00015] ⌈Each LIN PID shall be associated with a response data length in bytes (see `Lin_PduType`).⌋

7.2.3 State diagrams

The LIN driver has a state machine that is shown in Figure 7.1.

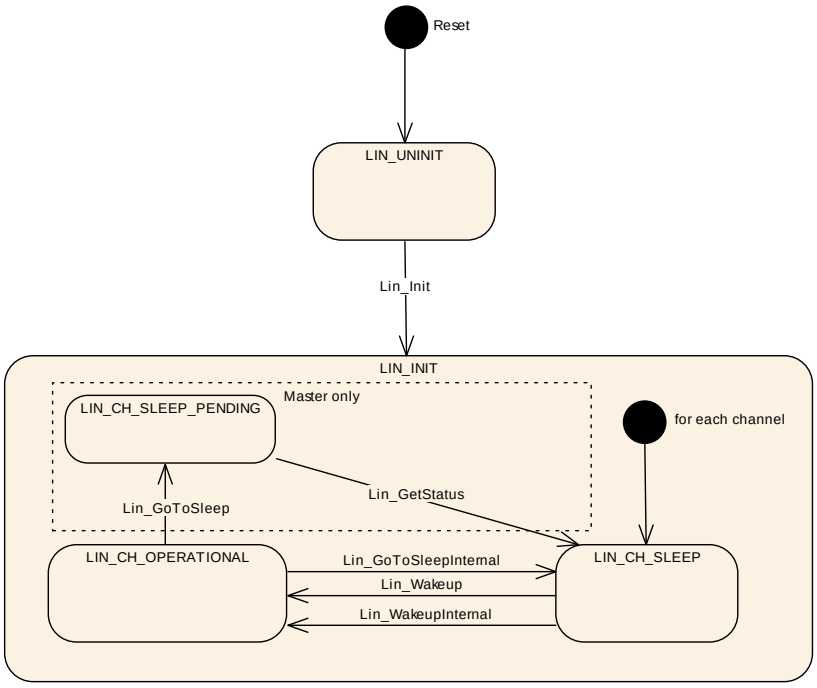

Figure 7.1: Lin driver states

Module State	Meaning / Activities in the state
LIN_UNINIT	The state LIN_UNINIT means that the Lin module has not been initialized yet and cannot be used.
LIN_INIT	The LIN_INIT state indicates that the LIN driver has been initialized, making each available channel ready for service.

Table 7.1: Lin driver states

Channel State	Meaning / Activities in the state
LIN_CH_OPERATIONAL	The individual channel has been initialized (using at least one statically configured data set) and is able to participate in the LIN cluster.
LIN_CH_SLEEP	The detection of a 'wake-up' pulse is enabled. The LIN hardware is into a low power mode if such a mode is provided by the hardware.

Table 7.2: Lin driver channel states

[SWS_Lin_00145] ⌈Reset -> LIN_UNINIT: After reset, the Lin module shall set its state to LIN_UNINIT.⌋

[SWS_Lin_00146] ⌈LIN_UNINIT -> LIN_INIT: The Lin module shall transition from LIN_UNINIT to LIN_INIT when the function `Lin_Init` is called.⌋

The LIN module's environment shall call the function `Lin_Init` only once during run-time.

[SWS_Lin_00171] ⌈On entering the state LIN_INIT, the Lin module shall set each channel into state LIN_CH_SLEEP, enable bus monitoring for a wake-up request on that channel if external wake-up detection is supported by configuration parameter

LinChannelWakeupSupport, and optionally set the LIN hardware unit to reduced power operation mode (if supported by HW).⌋

[SWS_Lin_00263] ⌈LIN_CH_OPERATIONAL -> LIN_CH_SLEEP_PENDING through Lin_GoToSleep: If a go to sleep is requested by the LIN interface, the Lin module shall ensure that the rest of the LIN cluster goes to sleep also. This is achieved by issuing a go-to-sleep-command on the bus before entering the LIN_CH_SLEEP_PENDING state. This requirement is only applicable for LIN master nodes.⌋

[SWS_Lin_00264] ⌈LIN_CH_SLEEP_PENDING -> LIN_CH_SLEEP: When Lin_GetStatus is called, the LIN driver shall directly enter the LIN_CH_SLEEP state, even if the go-to-sleep-command has not yet been sent. This requirement is only applicable for LIN master nodes.⌋

[SWS_Lin_00265] ⌈LIN_CH_OPERATIONAL -> LIN_CH_SLEEP through Lin_GoToSleepInternal: If an internal go to sleep is requested by the LIN interface, the LIN driver shall directly enter the LIN_CH_SLEEP state.⌋

[SWS_Lin_00174] ⌈LIN_CH_SLEEP -> LIN_CH_OPERATIONAL through Lin_Wakeup: If a LIN channel is in the state LIN_CH_SLEEP, the function Lin_Wakeup shall put the LIN channel into the state LIN_CH_OPERATIONAL.⌋

[SWS_Lin_00261] ⌈LIN_CH_SLEEP -> LIN_CH_OPERATIONAL through Lin_WakeupInternal: If a LIN channel is in the state LIN_CH_SLEEP, the function Lin_WakeupInternal shall put the LIN channel into the state LIN_CH_OPERATIONAL.⌋

[SWS_Lin_00209] ⌈Lin_Wakeup: During the state transition from LIN_CH_SLEEP to LIN_CH_OPERATIONAL the LIN Driver shall ensure that the rest of the cluster is awake. This is achieved by issuing a wake-up request, forcing the bus to the dominant state for 250 μs to 5 ms.⌋

[SWS_Lin_00184] ⌈A mode switch request to the current mode is allowed and shall not lead to an error, even if DET is enabled.⌋

7.3 Frame processing

7.3.1 Background & Rationale

A LIN frame is composed of two parts, the LIN header and the LIN response.

The LIN header is always transmitted by the LIN master node and indicates the start of frame, including the LIN PID. The LIN response is transmitted on the bus after the LIN header and can be transmitted by either the master or one of the slaves [1].

The driver must also be able to access data concerning the checksum model and data length for each LIN PID. LIN 2.0 and newer versions have a different checksum model compared to LIN 1.3, but the LIN master must be able to communicate with all slave

node types (LIN 1.3, LIN 2.0, LIN 2.1, LIN 2.2 and ISO 17987). However, a LIN 1.3 master is not able to communicate with LIN 2.0 or newer slaves.

The checksum is a part of the response, and may or may not include the PID depending upon the checksum model for the PID in question. The LIN ID's 60 (0x3c) to 63 (0x3f) must always use the classic (response data only) checksum model [1].

The LIN driver module works with LIN frames as its basic building block. From the LIN driver module point of view, the frame handling significantly differs depending on the implemented node type:

- For a LIN master node, the LIN interface layer requests a particular frame to be sent during one of its scheduler time-slots. Any response from the frame should be available latest before the next frame will be sent.

  In the case that the master is also responsible for sending the frame response, an indication (PduInfoPtr->Drc= LIN_FRAMERESPONSE_TX) will be given at the same time as the request to send the frame.

- A LIN slave node waits for the reception of a LIN header. A received LIN header is indicated to the LIN interface that evaluates the PID and returns information about the response. In the case that the slave is responsible for sending the frame response, the transmission data is also directly provided.

The LIN driver module must be able to retrieve data from the response and make it available to the LIN interface module. It must retrieve all data from the response without blocking.

7.3.2 Requirements

7.3.2.1 LIN Master specific

Following requirements are only appicable for LIN master nodes.

[SWS_Lin_00016] ⌈The LIN driver shall interpret the supplied identifier as PID. The identifier is then transmitted as-supplied within the LIN header (see `Lin_SendFrame`).⌋

[SWS_Lin_00017] ⌈The LIN driver shall be able to send a LIN header. This is composed of the break field, synch byte field, and protected identifier byte field as detailed in [1] (see `Lin_SendFrame`).⌋

[SWS_Lin_00018] ⌈The LIN driver shall be able to send a LIN header and response.⌋

[SWS_Lin_00021] ⌈The LIN driver shall start transmission of the new LIN header when `Lin_SendFrame` is called, even if an ongoing transmission may be still in progress or unsuccessfully completed.⌋

Note: As a result of [SWS_Lin_00021] ongoing frame transmission and receptions are terminated.

[SWS_Lin_00022] ⌈The function `Lin_GetStatus` shall return the status of the current frame transmission request for the channel.⌋

[SWS_Lin_00024] ⌈The LIN driver shall make received data available to the LIN interface module. After successful reception of a whole LIN frame, the received data shall be prepared for function call of the LIN interface (see `Lin_GetStatus`).⌋

[SWS_Lin_00025] ⌈The LIN driver shall send response data as provided by the LIN interface module (see `Lin_SendFrame`).⌋

7.3.2.2 LIN Slave specific

Following requirements are only appicable for LIN Slave nodes.

[SWS_Lin_00272] ⌈The LIN driver shall be able to receive a LIN header at any time in LIN_CH_OPERATIONAL state. The header is composed of the break field, synch byte field, and protected identifier byte field as detailed in [1].⌋

[SWS_Lin_00280] ⌈On LIN header reception, the LIN driver shall call the header indication callback function `LinIf_HeaderIndication` with PduPtr->Pid set to the received PID value and PduPtr->SduPtr set to the (hardware or shadow) buffer of the LIN driver, to which the slave response shall be written by the upper layer.⌋

Note: If the LIN hardware unit provides the ID (frame identifier without parity bits) instead of the PID, the LIN driver is responsible to calculate the PID from the ID to comply with the callback interface.

[SWS_Lin_00271] ⌈If the LIN hardware unit cannot detect invalid PIDs, the LIN driver shall not evaluate the PID value (i.e. it shall not verify parity bits in software). The LIN driver shall provide the PID as-received to the LIN interface module.⌋

[SWS_Lin_00281] ⌈While waiting for a new LIN header, the LIN driver shall call the error indication callback function `LinIf_LinErrorIndication` with error parameter LIN_ERR_HEADER when it detects bus events that do not comply to a valid LIN header (e.g incomplete LIN header).⌋

[SWS_Lin_00273] ⌈The LIN driver shall be able to send, receive or ignore a LIN response.⌋

[SWS_Lin_00282] ⌈After the call of `LinIf_HeaderIndication` when the return value is E_OK, the LIN driver shall evaluate the PduPtr->Drc to determine the type of LIN response.⌋

[SWS_Lin_00284] ⌈If the LIN response is going to be received (LIN_FRAMERESPONSE_RX), the LIN driver shall evaluate the Cs and DI members in parameter PduPtr (after the call of `LinIf_HeaderIndication` with return value E_OK) to configure the LIN response reception.⌋

[SWS_Lin_00274] ⌈After successful reception of a LIN response, the LIN driver shall directly make the received data available to the LIN interface module by calling the Rx indication callback function `LinIf_RxIndication` with the Lin_SduPtr parameter set to the reception data.⌋

[SWS_Lin_00283] ⌈If the LIN response is going to be transmitted (LIN_FRAMERESPONSE_TX), the LIN driver shall evaluate the Cs, DI and SduPtr members in parameter PduPtr (after the call of `LinIf_HeaderIndication` with return value E_OK) to setup and transmit the LIN response.⌋

[SWS_Lin_00286] ⌈If the return value of `LinIf_HeaderIndication` is E_NOT_OK or the returned PduPtr->Drc is LIN_FRAMERESPONSE_IGNORE, the LIN driver shall ignore the response.⌋

[SWS_Lin_00275] ⌈After successful transmission of a LIN response, the transmission shall be directly confirmed to the LIN Interface module by calling the Tx confirmation callback function `LinIf_TxConfirmation`.⌋

[SWS_Lin_00276] ⌈The LIN driver shall not report any events to the LIN interface module if the LIN response is ignored until the reception of a new LIN header.⌋

[SWS_Lin_00277] ⌈The LIN driver shall detect communication errors during response transmission and response reception. Once an error is detected, the current frame handling shall be aborted and the error indication callback function `LinIf_LinErrorIndication` shall be called.⌋

[SWS_Lin_00285] ⌈The handling of each relevant (i.e. not ignored) LIN response must be completed by a call to either `LinIf_RxIndication`, `LinIf_TxConfirmation` or `LinIf_LinErrorIndication`, latest before a new LIN header reception is indicated by a call of `LinIf_HeaderIndication`.⌋

7.3.2.3 Common

[SWS_Lin_00019] ⌈The LIN driver shall be able to calculate either a 'classic' or an 'enhanced' checksum depending upon the checksum model for the current LIN PDU.⌋

[SWS_Lin_00026] ⌈If the LIN hardware unit cannot queue the bytes for transmission or reception (e.g. simple UART implementation), the LIN driver shall provide a temporary communication buffer.⌋

[SWS_Lin_00027] ⌈The LIN driver shall initiate transmission without blocking, including the check of the next byte transmission only upon successful reception of the previous one (receive-back).⌋

[SWS_Lin_00028] ⌈The LIN driver shall receive data without blocking.⌋

7.3.3 Data Consistency

7.3.3.1 Transmit Data Consistency:

[SWS_Lin_00053] ⌈The LIN driver shall directly copy the data from the upper layer buffers.⌋

[SWS_Lin_00210] ⌈For LIN Master nodes, the upper layer of the LIN Driver has to keep the buffer data consistent until return of function call.

For LIN Slave nodes, the upper layer of the LIN Driver has to keep the buffer data consistent until end of response transmission.⌋

7.3.3.2 Receive Data Consistency:

Following is applicable for LIN master and LIN slave nodes:

For the LIN response reception the bytes of the SDU buffer shall be allocated in increasingly consecutive address order. The LIN frame data length information defines the minimum SDU buffer length.

[SWS_Lin_00060] ⌈The complete LIN frame receive processing (including copying to destination layer) can be implemented in an ISR. The received data shall be consistent until either next LIN frame has been received successfully or LIN channel state has changed.⌋

As long as it is guaranteed that neither the ISRs nor `Lin_GetStatus` (Master only) can be interrupted by itself, the LIN hardware (or shadow) buffer is always consistent, because it is written and read in sequence in exactly one function that is never interrupted by itself.

7.3.3.2.1 Receive data consistency (Master only)

[SWS_Lin_00211] ⌈The complete LIN frame receive processing (including copying to destination layer) can be implemented in the `Lin_GetStatus` function. The received data shall be consistent until either next LIN frame has been received successfully or LIN channel state has changed.⌋

7.3.4 Data byte mapping

[SWS_Lin_00096] ⌈Data mapping between memory and the LIN frame is defined in a way that the array element 0 is containing the LSB (the data byte to send/receive first) and the array element (n-1) is containing the MSB (the data byte to send/receive last).⌋

7.4 Sleep and wake-up functionality

There are two different possibilities to wake-up a LIN hardware channel:

1. Internal (Top-Down) wake-up

   The upper layer requests a wake-up through a call to `Lin_Wakeup`.

2. External (Bottom-Up) wake-up

   A bus wake-up event is detected and forwarded to the upper layer through the `Lin_CheckWakeup` API which has to be called by module LinIf. After a successful validation of the wake-up source, `Lin_WakeupInternal` is also called.

The selection of the wake-up modes is controlled by configuration parameter `LinChannelWakeupSupport`:

- Both internal and external wake-up modes are supported by a LIN hardware channel having `LinChannelWakeupSupport` = TRUE
- Only internal wake-up mode is supported by a LIN hardware channel having `LinChannelWakeupSupport` = FALSE.

7.4.1 Background & Rationale

Following is applicable for LIN master nodes only:

The LIN Master node can be awakened either by a wake-up signal generated by one of the slaves, or by a request from the higher layer (LIN interface) (see [SWS_Lin_00209]). The LIN interface controls the message schedule table and so must be able to instruct the LIN driver to put the hardware unit to sleep, or to wake it up (see [SWS_LinIf_00296], [SWS_LinIf_00488]).

For this purpose, the LIN driver provides functions to put the LIN channel into its LIN_CH_SLEEP state (see `Lin_GoToSleep` / `Lin_GoToSleepInternal`).

Following is applicable for LIN slave nodes only:

The LIN slave can be awakened either by a wake-up signal generated by the master node or any other slave node on the same channel, or by a request from the higher layer (LIN interface) (see [SWS_Lin_00209]).

The LIN Interface detects the conditions to enter sleep mode and instructs the LIN driver to put the hardware unit to sleep, or to wake it up (see [SWS_LinIf_00296], [SWS_LinIf_00488]).

For this purpose, the LIN driver provides a function to put the LIN channel into its LIN_CH_SLEEP state (see `Lin_GoToSleep` / `Lin_GoToSleepInternal`). Because the slave node cannot transmit a go-to-sleep-command, the function `Lin_GoToSleep` is not applicable for slave nodes.

Following is applicable for LIN master and LIN slave nodes:

Upon sleep or wake-up the master must communicate the status change with the rest of the network.

7.4.2 Requirements

[SWS_Lin_00032] ⌈When the LIN channel enters sleep mode, it shall perform the transition to low-power mode of the LIN hardware unit (if available) (see `Lin_GoToSleep` / `Lin_GoToSleepInternal`).⌋

[SWS_Lin_00033] ⌈Each LIN channel shall be able to accept a sleep request independently of the other channel states (see `Lin_GoToSleep` / `Lin_GoToSleepInternal`).⌋

[SWS_Lin_00037] ⌈When a LIN channel is in LIN_CH_SLEEP state and wake-up detection is supported by configuration parameter `LinChannelWakeupSupport`, the LIN hardware unit shall monitor the bus for a wake-up request on that channel.⌋

[SWS_Lin_00043] ⌈Lin_Wakeup: If the LIN driver receives a wake-up request from the LIN interface, the requested channel shall send a wake-up pulse to the LIN bus. (see `Lin_Wakeup`)⌋

[SWS_Lin_00262] ⌈Lin_WakeupInternal: If the LIN driver receives an internal wake-up request from the LIN interface, the requested channel shall send no wake-up pulse to the LIN bus. (see `Lin_WakeupInternal`)⌋

For LIN Master nodes, the function `Lin_GetStatus` returns the current state of a given LIN channel.

7.5 Error Classification

Chapter [3, General Specification of Basic Software Modules] 7.2 *"Error Handling"* describes the error handling of the Basic Software in detail. Above all, it constitutes a classification scheme consisting of five error types which may occur in BSW modules.

Based on this foundation, the following section specifies particular errors arranged in the respective subsections below.

7.5.1 Development Errors

[SWS_Lin_00048] Definition of development errors in module Lin

Upstream requirements: SRS_BSW_00323, SRS_BSW_00327, SRS_BSW_00337, SRS_BSW_00385

⌈

Type of error	*Related error code*	*Error value*
API service used without module initialization	LIN_E_UNINIT	0x00
API service used with an invalid or inactive channel parameter	LIN_E_INVALID_CHANNEL	0x02
API service called with invalid configuration pointer	LIN_E_INVALID_POINTER	0x03
Invalid state transition for the current state	LIN_E_STATE_TRANSITION	0x04
API service called with a NULL pointer	LIN_E_PARAM_POINTER	0x05

⌋

7.5.2 Runtime Errors

There are no runtime errors.

7.5.3 Production Errors

7.5.3.1 LIN_E_TIMEOUT

[SWS_Lin_00290] This error is reported when time out caused by hardware error occurs.

Upstream requirements: SRS_BSW_00458

⌈

Diagnostic Event (Error Name)	LIN_E_TIMEOUT
Description	If a change to the LIN hardware control registers results in the need to wait for a status change, this shall be protected by a configurable time out mechanism. If such a time out is detected the LIN_E_TIMEOUT error shall be raised. This situation should only arise in the event of a LIN hardware unit fault and should be communicated to the rest of the system.
Failed condition	A LIN hardware control register has changed and the configured time (see LinTimeout Duration) has elapsed without a status change of the LIN Hardware.
Passed condition	A LIN hardware control register has changed and the status change is done within the configured time (see LinTimeoutDuration).

⌋

[SWS_Lin_00058] ⌈The only production error that can be reported by the LIN driver is the `LIN_E_TIMEOUT` error.⌋

7.6 Security Events

The module does not report security events.

8 API specification

8.1 Imported types

In this chapter all types included from the following files are listed.

[SWS_Lin_00226] Definition of imported datatypes of module Lin ⌈

Module	*Header File*	*Imported Type*
Comtype	ComStack_Types.h	NetworkHandleType
Dem	Rte_Dem_Type.h	Dem_EventIdType
	Rte_Dem_Type.h	Dem_EventStatusType
EcuM	EcuM.h	EcuM_WakeupSourceType
Icu	Icu.h	Icu_ChannelType
Std	Std_Types.h	Std_ReturnType

⌋

8.2 Type definitions

[SWS_Lin_00245] ⌈The content of `Lin_GeneralTypes.h` shall be protected by a `LIN_GENERAL_TYPES` define.⌋

[SWS_Lin_00246] ⌈If different LIN drivers are used, only one instance of this file has to be included in the source tree. For implementation all `Lin_GeneralTypes.h` related types in the documents mentioned before shall be considered.⌋

8.2.1 Lin_ConfigType

[SWS_Lin_00227] Definition of datatype Lin_ConfigType ⌈

Name	Lin_ConfigType	
Kind	Structure	
Elements	Hardware and Implementation dependent structure	
	Type	–
	Comment	The contents of the initialization data structure are LIN hardware specific
Description	This is the type of the external data structure containing the overall initialization data for the LIN driver and the SFR settings affecting the LIN channels. A pointer to such a structure is provided to the LIN driver initialization routine for configuration of the driver, LIN hardware unit and LIN hardware channels.	
Available via	Lin.h	

⌋

8.2.2 Lin_FramePidType

[SWS_Lin_00228] Definition of datatype Lin_FramePidType ⌈

Name	Lin_FramePidType		
Kind	Type		
Derived from	uint8		
Range	0...0xFE	–	The LIN identifier (0...0x3F) together with its two parity bits.
Description	Represents all valid protected identifier used by Lin_SendFrame().		
Available via	Lin_GeneralTypes.h		

⌋

8.2.3 Lin_FrameCsModelType

[SWS_Lin_00229] Definition of datatype Lin_FrameCsModelType ⌈

Name	Lin_FrameCsModelType		
Kind	Enumeration		
Range	LIN_ENHANCED_CS	0x0	Enhanced checksum model
	LIN_CLASSIC_CS	0x1	Classic checksum model
Description	This type is used to specify the Checksum model to be used for the LIN Frame.		
Available via	Lin_GeneralTypes.h		

⌋

8.2.4 Lin_FrameResponseType

[SWS_Lin_00230] Definition of datatype Lin_FrameResponseType ⌈

Name	Lin_FrameResponseType		
Kind	Enumeration		
Range	LIN_FRAMERESPONSE_TX	0x0	Response is generated from this node
	LIN_FRAMERESPONSE_RX	0x1	Response is generated from another node and is relevant for this node.
	LIN_FRAMERESPONSE_IGNORE	0x2	Response is generated from another node and is irrelevant for this node
Description	This type is used to specify whether the frame processor is required to transmit the response part of the LIN frame.		
Available via	Lin_GeneralTypes.h		

⌋

Note: For LIN master nodes, the mapping to the previous definition is as follows: LIN_MASTER_RESPONSE <-> LIN_FRAMERESPONSE_TX LIN_SLAVE_RESPONSE <-> LIN_FRAMERESPONSE_RX LIN_SLAVE_TO_SLAVE <-> LIN_FRAMERESPONSE_IGNORE

8.2.5 Lin_FrameDlType

[SWS_Lin_00231] Definition of datatype Lin_FrameDlType ⌈

Name	Lin_FrameDlType		
Kind	Type		
Derived from	uint8		
Range	1...8	–	Data length of a LIN Frame
Description	This type is used to specify the number of SDU data bytes to copy.		
Available via	Lin_GeneralTypes.h		

⌋

8.2.6 Lin_PduType

[SWS_Lin_00232] Definition of datatype Lin_PduType ⌈

Name	Lin_PduType	
Kind	Structure	
Elements	Pid	
	Type	Lin_FramePidType
	Comment	–
	Cs	
	Type	Lin_FrameCsModelType
	Comment	–
	Drc	
	Type	Lin_FrameResponseType
	Comment	–
	Dl	
	Type	Lin_FrameDlType
	Comment	–
	SduPtr	
	Type	uint8*
	Comment	–
Description	This Type is used to provide PID, checksum model, data length and SDU pointer from the LIN Interface to the LIN driver.	
Available via	Lin_GeneralTypes.h	

⌋

Description for each element of `Lin_PduType` is given in:
- Section 8.2.2 for `Lin_FramePidType`
- Section 8.2.3 for `Lin_FrameCsModelType`
- Section 8.2.4 for `Lin_FrameResponseType`
- Section 8.2.5 for `Lin_FrameDlType`

8.2.7 Lin_StatusType

[SWS_Lin_00233] Definition of datatype Lin_StatusType ⌈

Name	Lin_StatusType		
Kind	Enumeration		
Range	LIN_NOT_OK	0x0	LIN frame operation return value. Development or production error occurred
	LIN_TX_OK	0x1	LIN frame operation return value. Successful transmission.
	LIN_TX_BUSY	0x2	LIN frame operation return value. Ongoing transmission (Header or Response).
	LIN_TX_HEADER_ERROR	0x3	LIN frame operation return value. Erroneous header transmission such as: • Mismatch between sent and read back data • Identifier parity error or • Physical bus error
	LIN_TX_ERROR	0x4	LIN frame operation return value. Erroneous response transmission such as: • Mismatch between sent and read back data • Physical bus error
	LIN_RX_OK	0x5	LIN frame operation return value. Reception of correct response.
	LIN_RX_BUSY	0x6	LIN frame operation return value. Ongoing reception: at least one response byte has been received, but the checksum byte has not been received.
	LIN_RX_ERROR	0x7	LIN frame operation return value. Erroneous response reception such as: • Framing error • Overrun error • Checksum error or • Short response
	LIN_RX_NO_RESPONSE	0x8	LIN frame operation return value. No response byte has been received so far.
	LIN_OPERATIONAL	0x9	LIN channel state return value. Normal operation; the related LIN channel is ready to transmit next header. No data from previous frame available (e.g. after initialization)
	LIN_CH_SLEEP	0xA	LIN channel state return value. Sleep state operation; in this state wake-up detection from slave nodes is enabled.
Description	LIN operation states for a LIN channel or frame, as returned by the API service Lin_GetStatus().		
Available via	Lin_GeneralTypes.h		

⌋

8.2.8 Lin_SlaveErrorType

[SWS_Lin_91140] Definition of datatype Lin_SlaveErrorType ⌈

Name	Lin_SlaveErrorType		
Kind	Enumeration		
Range	LIN_ERR_HEADER	0x0	Error in header
	LIN_ERR_RESP_STOPBIT	0x1	Framing error in response
	LIN_ERR_RESP_CHKSUM	0x2	Checksum error
	LIN_ERR_RESP_DATABIT	0x3	Monitoring error of transmitted data bit in response
	LIN_ERR_NO_RESP	0x4	No response
	LIN_ERR_INC_RESP	0x5	Incomplete response
Description	This type represents the slave error types that are detected during header reception and response transmission / reception.		
Available via	Lin_GeneralTypes.h		

⌋

8.3 Function definitions

This chapter contains all APIs provided by the module.

8.3.1 Services affecting the complete LIN hardware unit

8.3.1.1 Lin_Init

[SWS_Lin_00006] Definition of API function Lin_Init

Upstream requirements: SRS_BSW_00406, SRS_BSW_00101

⌈

Service Name	Lin_Init	
Syntax	`void Lin_Init ( const Lin_ConfigType* Config )`	
Service ID [hex]	0x00	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	Config	Pointer to LIN driver configuration set.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Initializes the LIN module.	
Available via	Lin.h	

⌋

[SWS_Lin_00084] ⌈The function Lin_Init shall initialize the Lin module (i.e. static variables, including flags and LIN HW Unit global hardware settings), as well as the LIN channels.⌋

Different sets of static configuration may have been configured.

[SWS_Lin_00150] ⌈The function Lin_Init shall initialize the module according to the configuration set pointed to by the parameter Config.⌋

[SWS_Lin_00008] ⌈The function Lin_Init shall invoke initializations for relevant hardware register settings common to all channels available on the LIN hardware unit.⌋

[SWS_Lin_00190] ⌈The function Lin_Init shall also invoke initializations for LIN channel specific settings.⌋

[SWS_Lin_00106] ⌈The Lin module's environment shall not call any function of the Lin module before having called Lin_Init.⌋

[SWS_Lin_00099] ⌈If development error detection for the Lin module is enabled: the function Lin_Init shall check the parameter Config for being within the allowed range. If Config is not in the allowed range, the function Lin_Init shall raise the development error LIN_E_INVALID_POINTER.⌋

[SWS_Lin_00105] ⌈If development error detection for the Lin module is enabled: the function Lin_Init shall check the Lin driver for being in the state LIN_UNINIT. If the Lin driver is not in the state LIN_UNINIT, the function Lin_Init shall raise the development error LIN_E_STATE_TRANSITION.⌋

[SWS_Lin_00097] ⌈If a change to the LIN hardware control registers results in the need to wait for a status change, this shall be protected by a configurable time out mechanism (LinTimeoutDuration). If such a time out is detected the LIN_E_TIMEOUT error shall be raised to DEM. This situation should only arise in the event of a LIN hardware unit fault, and should be communicated to the rest of the system.⌋

A LIN_E_TIMEOUT will affect the complete LIN stack in a way that the LIN driver must be re-initialized or the LIN functionality must be switched off.

8.3.1.2 Lin_CheckWakeup

[SWS_Lin_00160] Definition of API function Lin_CheckWakeup ⌈

Service Name	Lin_CheckWakeup
Syntax	`Std_ReturnType Lin_CheckWakeup ( uint8 Channel )`
Service ID [hex]	0x0a

▽

△

Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	Channel	LIN channel to be addressed
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: API call has been accepted `E_NOT_OK`: API call has not been accepted
Description	This function checks if a wakeup has occurred on the addressed LIN channel.	
Available via	Lin.h	

⌋

There are two methods in which wake up detection shall happen, one is from LIN controller hardware [Micro peripheral device] and/or another from LinTranceiver.

[SWS_Lin_00098] ⌈The function Lin_CheckWakeup shall evaluate the wakeup on the addressed LIN channel. When a wake-up event on the addressed LIN channel (e.g. RxD pin has constant low level) is detected, the function Lin_CheckWakeup shall notify the ECU State Manager module immediately via the EcuM_SetWakeupEvent and the Lin Interface module via LinIf_WakeupConfirmation callback function.⌋

[SWS_Lin_00251] ⌈If development error detection for the LIN module is enabled: if the channel parameter is invalid, the function Lin_CheckWakeup shall raise the development error LIN_E_INVALID_CHANNEL otherwise (if DET is disabled) return E_NOT_OK.⌋

[SWS_Lin_00107] ⌈If development error detection for the LIN module is enabled: if the function Lin_CheckWakeup is called before the LIN module was initialized, the function Lin_CheckWakeup shall raise the development error LIN_E_UNINIT.⌋

8.3.2 Services affecting a single LIN channel

8.3.2.1 Lin_SendFrame

Note: This service is only applicable for LIN master node (available only if the ECU has any LIN master channel).

[SWS_Lin_00191] Definition of API function Lin_SendFrame ⌈

Service Name	Lin_SendFrame

Syntax	`Std_ReturnType Lin_SendFrame (`
	`  uint8 Channel,`
	`  const Lin_PduType* PduInfoPtr`
	`)`

Service ID [hex]	0x04
Sync/Async	Asynchronous

▽

△

Reentrancy	Non Reentrant for the same Channel. Reentrant for different Channel.	
Parameters (in)	Channel	LIN channel to be addressed
	PduInfoPtr	Pointer to PDU containing the PID, checksum model, response type, DI and SDU data pointer
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: Send command has been accepted. `E_NOT_OK`: Send command has not been accepted, development or production error occurred.
Description	Sends a LIN header and a LIN response, if necessary. The direction of the frame response (master response, slave response, slave-to-slave communication) is provided by the PduInfoPtr. Only used for LIN master nodes.	
Available via	Lin.h	

⌋

[SWS_Lin_00192] ⌈The function Lin_SendFrame shall send the header part (Break Field, Synch Byte Field and PID Field) and, depending on the direction of the frame response, a complete LIN response part of a LIN frame on the addressed LIN channel.⌋

[SWS_Lin_00193] ⌈In case of receiving data the LIN Interface has to wait for the corresponding response part of the LIN frame by polling with the function Lin_GetStatus() after using the function Lin_SendFrame().⌋

[SWS_Lin_00194] ⌈The Lin module's environment shall only call Lin_SendFrame on a channel which is in state LIN_CH_OPERATIONAL or in one of the sub-states of

LIN_CH_OPERATIONAL.⌋

[SWS_Lin_00239] ⌈In case of errors during header transmission, it is up to the implementer how to handle these errors (stop/continue transmission) and to decide if the corresponding response is valid or not.⌋

[SWS_Lin_00240] ⌈In case of response transmission errors, the ISO 17987 specifications describe within the frame processor state machine how to handle such errors. It is stated that a mismatch between sent and readback data shall be detected not later than after the completion of the byte field containing the mismatch. Furthermore, ISO 17987 specifications specify that the transmission shall be aborted.⌋

[SWS_Lin_00195] ⌈If development error detection for the LIN module is enabled: if the function Lin_SendFrame is called before the LIN module was initialized, the function Lin_SendFrame shall raise the development error LIN_E_UNINIT otherwise (if DET is disabled) return E_NOT_OK.⌋

[SWS_Lin_00197] ⌈If development error detection for the LIN module is enabled: if the channel parameter is invalid, the function Lin_SendFrame shall raise the development error LIN_E_INVALID_CHANNEL otherwise (if DET is disabled) return E_NOT_OK.⌋

[SWS_Lin_00198] ⌈If development error detection for the LIN module is enabled: the function Lin_SendFrame shall check the parameter PduInfoPtr for not being a NULL

pointer. If PduInfoPtr is a NULL pointer, the function Lin_SendFrame shall raise the development error LIN_E_PARAM_POINTER otherwise (if DET is disabled) return E_NOT_OK.⌋

[SWS_Lin_00199] ⌈If development error detection for the LIN module is enabled: if the LIN channel state-machine is in the state LIN_CH_SLEEP, the function Lin_SendFrame shall raise the development error LIN_E_STATE_TRANSITION otherwise (if DET is disabled) return E_NOT_OK.⌋

[SWS_Lin_00287] ⌈The function Lin_SendFrame is only available if the Lin module is configured as LIN master node on at least one channel. In a pure LIN slave configuration, this function is not available. This depends on the configuration parameters LinNodeType.⌋

8.3.2.2 Lin_GoToSleep

Note: This service is only applicable for LIN master node (available only if the ECU has any LIN master channel).

[SWS_Lin_00166] Definition of API function Lin_GoToSleep ⌈

Service Name	Lin_GoToSleep	
Syntax	`Std_ReturnType Lin_GoToSleep ( uint8 Channel )`	
Service ID [hex]	0x06	
Sync/Async	Asynchronous	
Reentrancy	Non Reentrant for the same Channel. Reentrant for different Channel.	
Parameters (in)	Channel	LIN channel to be addressed
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: Sleep command has been accepted `E_NOT_OK`: Sleep command has not been accepted, development or production error occurred
Description	The service instructs the driver to transmit a go-to-sleep-command on the addressed LIN channel. Only used for LIN master nodes.	
Available via	Lin.h	

⌋

[SWS_Lin_00089] ⌈The function Lin_GoToSleep shall send a go-to-sleep-command on the addressed LIN channel as defined in LIN Specification 2.1.⌋

[SWS_Lin_00266] ⌈The function Lin_GoToSleep shall set the channel state to LIN_CH_SLEEP_PENDING, even in case of an erroneous transmission of the go-to-sleep-command.⌋

[SWS_Lin_00220] ⌈If external wake-up detection is supported by configuration parameter LinChannelWakeupSupport, then the function Lin_GoToSleep shall enable bus

monitoring for a wake-up request on that channel, even in case of an erroneous transmission of the go-to-sleep command.⌋

[SWS_Lin_00221] ⌈The function Lin_GoToSleep shall optionally set the LIN hardware unit to reduced power operation mode (if supported by HW), even in case of an erroneous transmission of the go-to-sleep-command.⌋

[SWS_Lin_00255] ⌈The LIN channel shall enter the state LIN_CH_SLEEP the next time Lin_GetStatus is called, independent of the success of the transmission of the goto-sleep-command on the bus.⌋

[SWS_Lin_00074] ⌈The function Lin_GoToSleep shall terminate ongoing frame transmission of prior transmission requests, even if the transmission is unsuccessfully completed.⌋

[SWS_Lin_00129] ⌈If development error detection for the LIN module is enabled: if the function Lin_GoToSleep is called before the LIN module was initialized, the function Lin_GoToSleep shall raise the development error LIN_E_UNINIT.⌋

[SWS_Lin_00131] ⌈If development error detection for the LIN module is enabled: the function Lin_GoToSleep shall raise the development error LIN_E_INVALID_CHANNEL if the channel parameter is invalid.⌋

[SWS_Lin_00288] ⌈The function Lin_GotoSleep is only available if the Lin module is configured as LIN master node on at least one channel. In a pure LIN slave configuration, this function is not available. This depends on the configuration parameters LinNodeType.⌋

8.3.2.3 Lin_GoToSleepInternal

[SWS_Lin_00167] Definition of API function Lin_GoToSleepInternal ⌈

Service Name	Lin_GoToSleepInternal	
Syntax	`Std_ReturnType Lin_GoToSleepInternal ( uint8 Channel )`	
Service ID [hex]	0x09	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant for the same Channel. Reentrant for different Channel.	
Parameters (in)	Channel	LIN channel to be addressed
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: Command has been accepted `E_NOT_OK`: Command has not been accepted, development or production error occurred

▽

△

Description	Sets the channel state to LIN_CH_SLEEP, enables the wake-up detection and optionally sets the LIN hardware unit to reduced power operation mode (if supported by HW).
Available via	Lin.h

⌋

[SWS_Lin_00095] ⌈The function Lin_GoToSleepInternal shall set the channel state to LIN_CH_SLEEP.⌋

[SWS_Lin_00222] ⌈If external wake-up detection is supported by configuration parameter LinChannelWakeupSupport, then the function Lin_GoToSleepInternal shall enable bus monitoring for a wake-up request on that channel.⌋

[SWS_Lin_00223] ⌈The function Lin_GoToSleepInternal shall optionally set the LIN hardware unit to reduced power operation mode (if supported by HW).⌋

[SWS_Lin_00133] ⌈If development error detection for the LIN module is enabled: if the function Lin_GoToSleepInternal is called before the LIN module was initialized, the function Lin_GoToSleepInternal shall raise the development error LIN_E_UNINIT.⌋

[SWS_Lin_00135] ⌈If development error detection for the LIN module is enabled: the function Lin_GoToSleepInternal shall raise the development error LIN_E_INVALID_CHANNEL if the channel parameter is invalid.⌋

8.3.2.4 Lin_Wakeup

[SWS_Lin_00169] Definition of API function Lin_Wakeup ⌈

Service Name	Lin_Wakeup	
Syntax	`Std_ReturnType Lin_Wakeup ( uint8 Channel )`	
Service ID [hex]	0x07	
Sync/Async	Asynchronous	
Reentrancy	Non Reentrant for the same Channel. Reentrant for different Channel.	
Parameters (in)	Channel	LIN channel to be addressed
Parameters (inout)	None	
Parameters (out)	None	
Return value	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
Description	Generates a wake up pulse and sets the channel state to LIN_CH_OPERATIONAL.	
Available via	Lin.h	

⌋

[SWS_Lin_00137] ⌈If development error detection for the LIN module is enabled: if the function Lin_Wakeup is called before the LIN module was initialized, the function Lin_Wakeup shall raise the development error LIN_E_UNINIT.⌋

[SWS_Lin_00139] ⌈If development error detection for the LIN module is enabled: the function Lin_Wakeup shall raise the development error LIN_E_INVALID_CHANNEL if the channel parameter is invalid or the channel is inactive.⌋

[SWS_Lin_00140] ⌈If development error detection for the LIN module is enabled: the function Lin_Wakeup shall raise the development error LIN_E_STATE_TRANSITION if the LIN channel state-machine is not in the state LIN_CH_SLEEP.⌋

Note: The Lin driver's environment shall only call Lin_Wakeup when the LIN channel is in state LIN_CH_SLEEP.

8.3.2.5 Lin_WakeupInternal

[SWS_Lin_00256] Definition of API function Lin_WakeupInternal ⌈

Service Name	Lin_WakeupInternal	
Syntax	`Std_ReturnType Lin_WakeupInternal ( uint8 Channel )`	
Service ID [hex]	0x0b	
Sync/Async	Asynchronous	
Reentrancy	Non Reentrant for the same Channel. Reentrant for different Channel.	
Parameters (in)	Channel	LIN channel to be addressed
Parameters (inout)	None	
Parameters (out)	None	
Return value	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
Description	Sets the channel state to LIN_CH_OPERATIONAL without generating a wake up pulse.	
Available via	Lin.h	

⌋

[SWS_Lin_00257] ⌈The function Lin_WakeupInternal sets the addressed LIN channel to state LIN_CH_OPERATIONAL without generating a wake up pulse.⌋

[SWS_Lin_00258] ⌈If development error detection for the LIN module is enabled: if the function Lin_WakeupInternal is called before the LIN module was initialized, the function Lin_WakeupInternal shall raise the development error LIN_E_UNINIT.⌋

[SWS_Lin_00259] ⌈If development error detection for the LIN module is enabled: the function Lin_WakeupInternal shall raise the development error LIN_E_INVALID_CHANNEL if the channel parameter is invalid or the channel is inactive.⌋

[SWS_Lin_00260] ⌈If development error detection for the LIN module is enabled: the function Lin_WakeupInternal shall raise the development error LIN_E_STATE_TRANSITION if the LIN channel state-machine is not in the state LIN_CH_SLEEP.⌋

Note: The Lin driver's environment shall only call Lin_WakeupInternal when the LIN channel is in state LIN_CH_SLEEP.

8.3.2.6 Lin_GetStatus

Note: This service is only applicable for LIN master node (available only if the ECU has any LIN master channel).

[SWS_Lin_00168] Definition of API function Lin_GetStatus ⌈

Service Name	Lin_GetStatus	
Syntax	`Lin_StatusType Lin_GetStatus ( uint8 Channel, const uint8** Lin_SduPtr )`	
Service ID [hex]	0x08	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant for the same Channel. Reentrant for different Channel.	
Parameters (in)	Channel	LIN channel to be checked
Parameters (inout)	None	
Parameters (out)	Lin_SduPtr	Pointer to pointer to a shadow buffer or memory mapped LIN Hardware receive buffer where the current SDU is stored.
Return value	Lin_StatusType	LIN_NOT_OK: Development or production error occurred LIN_TX_OK: Successful transmission LIN_TX_BUSY: Ongoing transmission (Header or Response) LIN_TX_HEADER_ERROR: Erroneous header transmission such as: - Mismatch between sent and read back data - Identifier parity error or Physical bus error LIN_TX_ERROR: Erroneous response transmission such as: - Mismatch between sent and read back data - Physical bus error LIN_RX_OK: Reception of correct response LIN_RX_BUSY: Ongoing reception: at least one response byte has been received, but the checksum byte has not been received LIN_RX_ERROR: Erroneous response reception such as: - Framing error - Overrun error - Checksum error or Short response LIN_RX_NO_RESPONSE: No response byte has been received so far LIN_OPERATIONAL: Normal operation; the related LIN channel is woken up from the LIN_CH_SLEEP and no data has been sent. LIN_CH_SLEEP: Sleep state operation; in this state wake-up detection from slave nodes is enabled.
Description	Gets the status of the LIN driver. Only used for LIN master nodes.	
Available via	Lin.h	

⌋

[SWS_Lin_00091] ⌈The function Lin_GetStatus shall return the current transmission, reception or operation status of the LIN driver.⌋

[SWS_Lin_00200] ⌈The return states LIN_TX_OK, LIN_TX_BUSY, LIN_TX_HEADER_ERROR, LIN_TX_ERROR, LIN_RX_OK, LIN_RX_BUSY,

LIN_RX_ERROR , LIN_RX_NO_RESPONSE and LIN_OPERATIONAL are substates of the channel state LIN_CH_OPERATIONAL.⌋

[SWS_Lin_00092] ⌈If a SDU has been successfully received, the function Lin_GetStatus shall store the SDU in a shadow buffer or memory mapped LIN Hardware receive buffer referenced by Lin_SduPtr. The buffer will only be valid and must be read until the next `Lin_SendFrame` function call.⌋

[SWS_Lin_00238] ⌈The function Lin_GetStatus shall return LIN_TX_OK, when

- A Master Response Type frame is send and LIN header as well as LIN response of the frame are transmitted successfully or
- A Slave to Slave Response Type frame is send and the LIN header of the frame is transmitted successfully.

⌋

[SWS_Lin_00141] ⌈If development error detection for the LIN module is enabled: if the function Lin_GetStatus is called before the LIN module was initialized, the function Lin_GetStatus shall raise the development error LIN_E_UNINIT otherwise (if DET is disabled) return LIN_NOT_OK.⌋

[SWS_Lin_00143] ⌈If development error detection for the LIN module is enabled: if the channel parameter is invalid or the channel is inactive, the function Lin_GetStatus shall raise the development error LIN_E_INVALID_CHANNEL otherwise (if DET is disabled) return LIN_NOT_OK.⌋

[SWS_Lin_00144] ⌈If development error detection for the LIN module is enabled: the function Lin_GetStatus shall check the parameter Lin_SduPtr for not being a NULL pointer. If Lin_SduPtr is a NULL pointer, the function Lin_GetStatus shall raise the development error LIN_E_PARAM_POINTER otherwise (if DET is disabled) return LIN_NOT_OK.⌋

[SWS_Lin_00289] ⌈The function Lin_GetStatus is only available if the Lin module is configured as LIN master node on at least one channel. In a pure LIN slave configuration, this function is not available. This depends on the configuration parameters LinNodeType.⌋

8.4 Scheduled functions

8.4.1 Lin_MainFunction

[SWS_Lin_91141] Definition of scheduled function Lin_MainFunction ⌈

Service Name	Lin_MainFunction
Syntax	`void Lin_MainFunction ( void )`
Service ID [hex]	0x02
Description	This function polls all channels and handles all possible events. It corresponds to a Lin Interrupt-function for all LIN channels. Note: It is recommended to poll this API in a background loop with a guaranteed minimum rate of 500us (better 250us) in order not to lose any LIN messages.
Available via	Lin.h

⌋

[SWS_Lin_00291] Execution cycle of `Lin_MainFunction` ⌈The frequency of invocations of `Lin_MainFunction` is determined by the configuration parameter `LinMainFunctionPeriod`.⌋

8.5 Expected interfaces

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

8.5.1 Mandatory interfaces

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

[SWS_Lin_00234] Definition of mandatory interfaces required by module Lin ⌈

API Function	*Header File*	*Description*
Dem_SetEventStatus	Dem.h	Called by SW-Cs or BSW modules to report monitor status information to the Dem. BSW modules calling Dem_SetEventStatus can safely ignore the return value. This API will be available only if ({Dem/Dem ConfigSet/DemEventParameter/DemEvent ReportingType} == STANDARD_REPORTING)
EcuM_SetWakeupEvent	EcuM.h	Sets the wakeup event.
LinIf_WakeupConfirmation	LinIf.h	The LIN Driver or LIN Transceiver Driver will call this function to report the wake up source after the successful wakeup detection during CheckWakeup or after power on by bus.

⌋

8.5.2 Optional interfaces

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

[SWS_Lin_00235] Definition of optional interfaces requested by module Lin ⌈

API Function	*Header File*	*Description*
Det_ReportError	Det.h	Service to report development errors.
EcuM_CheckWakeup	EcuM.h	This function can be called to check the given wakeup sources. It will pass the argument to the integrator function EcuM_CheckWakeupHook. It can also be called by the ISR of a wakeup source to set up the PLL and check other wakeup sources that may be connected to the same interrupt.
Icu_DisableNotification	Icu.h	This function disables the notification of a channel.
Icu_EnableNotification	Icu.h	This function enables the notification on the given channel.
LinIf_HeaderIndication	LinIf.h	The LIN Driver will call this function to report a received LIN header. This function is only applicable for LIN slave nodes (available only if the ECU has any LIN slave channel).
LinIf_LinErrorIndication	LinIf.h	The LIN Driver will call this function to report a detected error event during header or response processing. This function is only applicable for LIN slave nodes (available only if the ECU has any LIN slave channel).
LinIf_RxIndication	LinIf.h	The LIN Driver will call this function to report a successfully received response and provides the reception data to the LIN Interface. This function is only applicable for LIN slave nodes (available only if the ECU has any LIN slave channel).
LinIf_TxConfirmation	LinIf.h	The LIN Driver will call this function to report a successfully transmitted response. This function is only applicable for LIN slave nodes (available only if the ECU has any LIN slave channel).

⌋

[SWS_Lin_00176] ⌈The Lin module shall invoke the callback function `EcuM_CheckWakeup` from within the wake-up ISR of the corresponding LIN channel when a valid LIN wake-up pulse has been detected.⌋

Restrictions:

A wake-up ISR can only be raised if supported by the LIN hardware (with or without Icu; up to the LIN driver implementation and the LIN hardware design). Therefore, `EcuM_CheckWakeup` is supported if at least for one channel wake-up is supported (see configuration parameter `LinChannelWakeupSupport`).

8.5.3 Callout interfaces

No callout interfaces.

8.5.4 Configurable interfaces

There is no configurable target for the LIN driver. The LIN driver always reports to LIN interface.

9 Sequence diagrams

Complete sequence diagrams for transmission, reception and error handling can be found in [4, LIN Interface Specification].

9.1 Receiving a LIN Frame

9.1.1 LIN Master

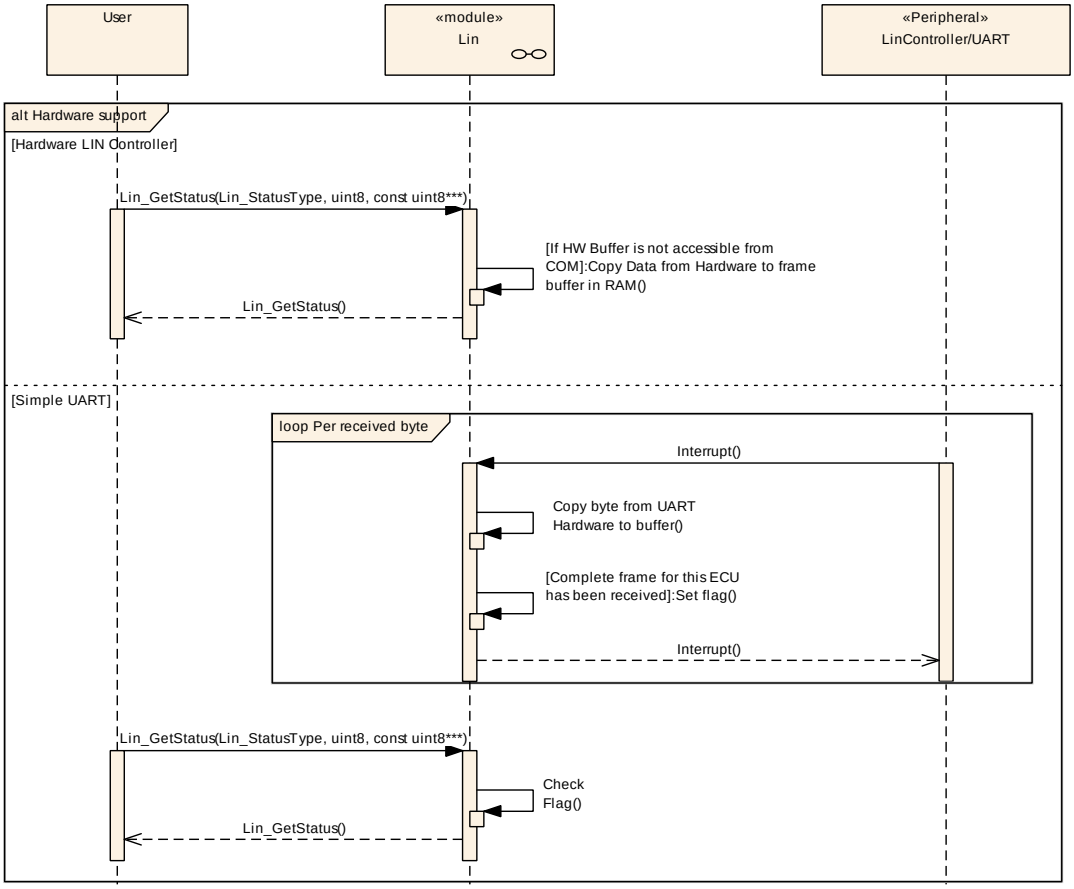

Figure 9.1: LIN Frame Receiving Sequence Chart for LIN Master

9.1.2 LIN Slave

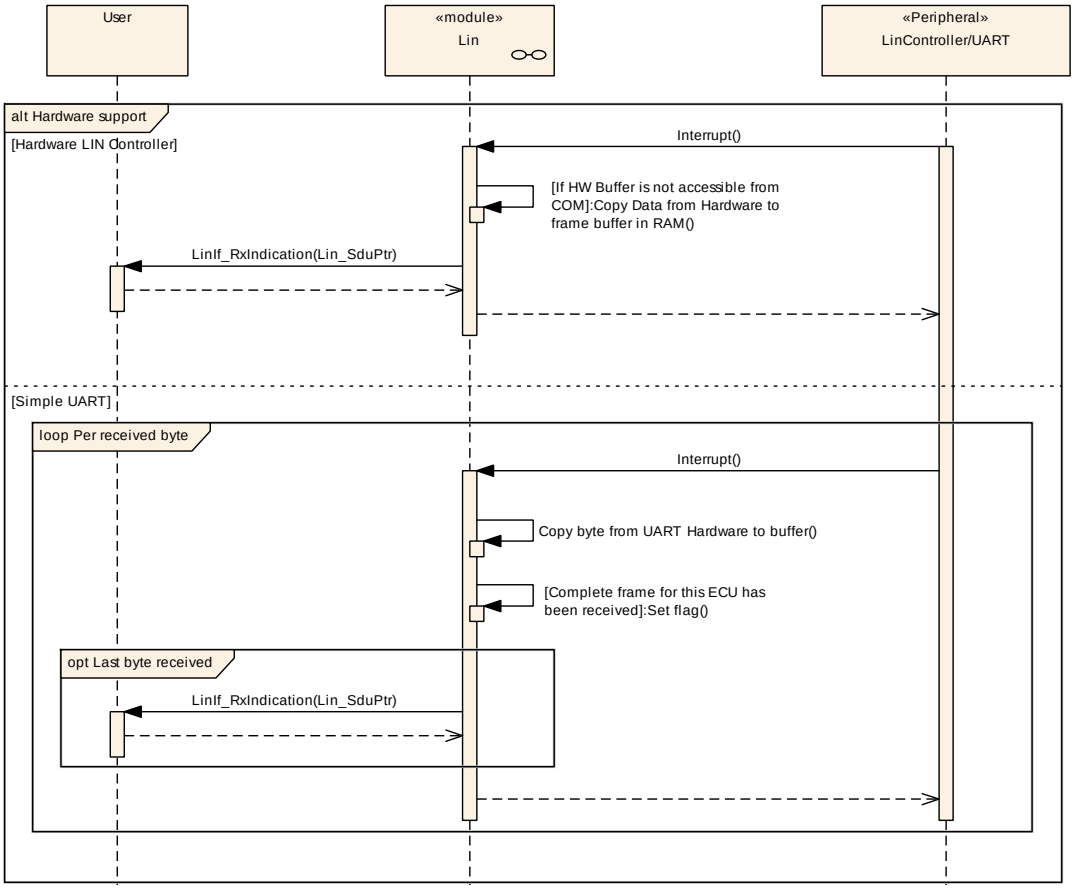

Figure 9.2: LIN Frame Receiving Sequence Chart for LIN Slave

10 Configuration specification

In general, this chapter defines configuration parameters and their clustering into containers. In order to support the specification Chapter 10.1 describes fundamentals.

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

Chapter 10.3 specifies published information of the module LIN Driver.

10.1 How to read this chapter

For details refer to [3] Chapter 10.1 *"Introduction to configuration specification"*.

10.2 Containers and configuration parameters

The following chapters summarize all configuration parameters. The detailed meanings of the parameters describe Chapter 7 and Chapter 8.

The described parameters are inputs for the LIN driver configurator.

[SWS_Lin_00029]

Upstream requirements: SRS_BSW_00159

⌈The code configurator of the LIN driver is LIN hardware Unit specific.⌋

[SWS_Lin_00039]

Upstream requirements: SRS_BSW_00167

⌈Values that can be configured are hardware dependent. Therefore, the rules and constraints cannot be given in the standard.⌋

[SWS_Lin_00224] ⌈The configuration tool is responsible to do a static configuration checking, also regarding dependencies between modules (e.g. Port driver, MCU driver etc.)⌋

[SWS_Lin_00269] ⌈The Lin Driver module shall reject configurations with partition mappings which are not supported by the implementation.⌋

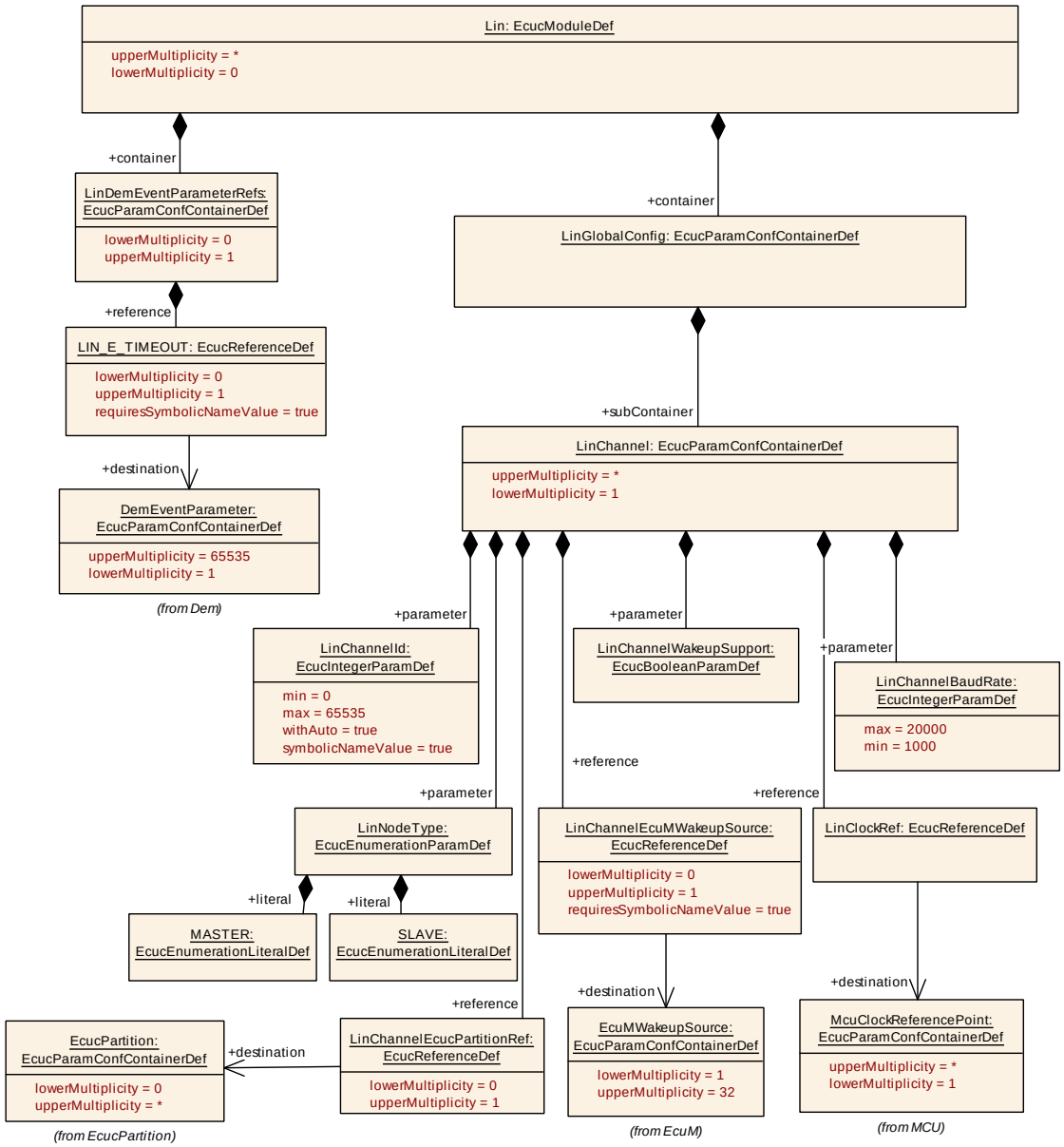

Figure 10.1: Configuration structure for the LIN driver

10.2.1 Lin

[ECUC_Lin_00190] Definition of EcucModuleDef Lin ⌈

Module Name	Lin
Description	Configuration of the Lin (LIN driver) module.
Post-Build Selectable Support	true
Post-Build Loadable Support	true

Included Containers		
Container Name	**Multiplicity**	**Description**
LinDemEventParameterRefs	0..1	Container for the references to DemEventParameter elements which shall be invoked using the API Dem_SetEventStatus in case the corresponding error occurs. The EventId is taken from the referenced DemEventParameter's DemEventId symbolic value. The standardized errors are provided in this container and can be extended by vendor-specific error references.
LinGeneral	1	This container contains the parameters related to each LIN Driver Unit.
LinGlobalConfig	1	This container contains the global configuration parameter of the Lin driver.

⌋

10.2.2 LinGeneral

[ECUC_Lin_00183] Definition of EcucParamConfContainerDef LinGeneral ⌈

Container Name	LinGeneral
Parent Container	Lin
Upstream mapping defined	–
Description	This container contains the parameters related to each LIN Driver Unit.
Multiplicity	1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
LinDevErrorDetect	1	[ECUC_Lin_00066]
LinIndex	1	[ECUC_Lin_00179]
LinMainFunctionPeriod	0..1	[ECUC_Lin_00194]
LinTimeoutDuration	1	[ECUC_Lin_00093]
LinEcucPartitionRef	0..*	[ECUC_Lin_00192]

No Included Containers

⌋

[ECUC_Lin_00066] Definition of EcucBooleanParamDef LinDevErrorDetect ⌈

Parameter Name	LinDevErrorDetect
Parent Container	LinGeneral
Upstream mapping defined	–
Description	Switches the development error detection and notification on or off. • true: detection and notification is enabled. • false: detection and notification is disabled.
Multiplicity	1

▽

△

Type	EcucBooleanParamDef
Default value	false
Pre-Compile only	true

⌋

[ECUC_Lin_00179] Definition of EcucIntegerParamDef LinIndex ⌈

Parameter Name	LinIndex	
Parent Container	LinGeneral	
Upstream mapping defined	–	
Description	Specifies the InstanceId of this module instance. If only one instance is present it shall have the Id 0.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 255	
Default value	–	
Pre-Compile only	true	

⌋

[ECUC_Lin_00194] Definition of EcucFloatParamDef LinMainFunctionPeriod ⌈

Parameter Name	LinMainFunctionPeriod	
Parent Container	LinGeneral	
Upstream mapping defined	–	
Description	Execution cycle of Lin_MainFunction in seconds. Polling mode is assumed if configured. It is recommended to set this value to 500ms or lower.	
Multiplicity	0..1	
Type	EcucFloatParamDef	
Range	[-INF .. INF]	
Default value	2.5E-4	
Pre-Compile only	true	

⌋

[ECUC_Lin_00093] Definition of EcucIntegerParamDef LinTimeoutDuration ⌈

Parameter Name	LinTimeoutDuration	
Parent Container	LinGeneral	
Upstream mapping defined	–	
Description	Specifies the maximum number of loops for blocking function until a timeout is raised in short term wait loops	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 65535	
Default value	–	
Pre-Compile only	true	

⌋

[ECUC_Lin_00192] Definition of EcucReferenceDef LinEcucPartitionRef ⌈

Parameter Name	LinEcucPartitionRef	
Parent Container	LinGeneral	
Upstream mapping defined	–	
Description	Maps the Lin driver to zero or multiple ECUC partitions to make the modules API available in this partition.	
Multiplicity	0..*	
Type	Reference to EcucPartition	
Post-Build Multiplicity	-	Selectable
Post-Build Value	-	Selectable

⌋

10.2.3 LinChannel

[ECUC_Lin_00069] Definition of EcucParamConfContainerDef LinChannel ⌈

Container Name	LinChannel
Parent Container	LinGlobalConfig
Upstream mapping defined	[10, CP_TPS_SystemTemplate]
Description	This container contains the configuration (parameters) of the LIN Controller(s).
Multiplicity	1..*
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
LinChannelBaudRate	1	[ECUC_Lin_00180]
LinChannelId	1	[ECUC_Lin_00181]
LinChannelWakeupSupport	1	[ECUC_Lin_00182]
LinNodeType	1	[ECUC_Lin_00191]
LinChannelEcucPartitionRef	0..1	[ECUC_Lin_00193]
LinChannelEcuMWakeupSource	0..1	[ECUC_Lin_00185]
LinClockRef	1	[ECUC_Lin_00094]

No Included Containers

⌋

[ECUC_Lin_00180] Definition of EcucIntegerParamDef LinChannelBaudRate ⌈

Parameter Name	LinChannelBaudRate
Parent Container	LinChannel
Upstream mapping defined	[10, CP_TPS_SystemTemplate]
Description	Specifies the baud rate of the LIN channel
Multiplicity	1

▽

△

Type	EcucIntegerParamDef	
Range	1000 .. 20000	
Default value	–	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_Lin_00181] Definition of EcucIntegerParamDef LinChannelId ⌈

Parameter Name	LinChannelId	
Parent Container	LinChannel	
Upstream mapping defined	–	
Description	Identifies the LIN channel. Replaces LIN_CHANNEL_INDEX_NAME from the LIN SWS. Dependency: Implicit from each CommunicationConnector on the ECU representing a LIN channel. Increase the LinChannelId for each LIN channel created on the same CommunicationController, for each CommunicationController start indexing at zero.	
Multiplicity	1	
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)	
Range	0 .. 65535	
Default value	–	
With Auto	true	
Pre-Compile only	true	

⌋

[ECUC_Lin_00182] Definition of EcucBooleanParamDef LinChannelWakeupSupport ⌈

Parameter Name	LinChannelWakeupSupport
Parent Container	LinChannel
Upstream mapping defined	[10, CP_TPS_SystemTemplate]
Description	Specifies if the LIN hardware channel supports wake up functionality
Multiplicity	1
Type	EcucBooleanParamDef
Default value	–
Pre-Compile only	true

⌋

[ECUC_Lin_00191] Definition of EcucEnumerationParamDef LinNodeType ⌈

Parameter Name	LinNodeType	
Parent Container	LinChannel	
Upstream mapping defined	[10, CP_TPS_SystemTemplate]	
Description	Specifies the LIN node type of this channel.	
Multiplicity	1	
Type	EcucEnumerationParamDef	
Range	MASTER	Master node
	SLAVE	Slave node

▽

△

Pre-Compile only	true

⌋

[ECUC_Lin_00193] Definition of EcucReferenceDef LinChannelEcucPartitionRef ⌈

Parameter Name	LinChannelEcucPartitionRef	
Parent Container	LinChannel	
Upstream mapping defined	–	
Description	Maps one single Lin channel to zero or one ECUC partitions. The ECUC partition referenced is a subset of the ECUC partitions where the Lin driver is mapped to.	
Multiplicity	0..1	
Type	Reference to EcucPartition	
Post-Build Multiplicity	-	Selectable
Post-Build Value	-	Selectable

⌋

[ECUC_Lin_00185] Definition of EcucReferenceDef LinChannelEcuMWakeup Source ⌈

Parameter Name	LinChannelEcuMWakeupSource
Parent Container	LinChannel
Upstream mapping defined	–
Description	This parameter contains a reference to the Wakeup Source for this channel as defined in the ECU State Manager. This reference is only needed if LinChannelWakeup Support is true. Dependency: LinChannelWakeupSupport
Multiplicity	0..1
Type	Symbolic name reference to EcuMWakeupSource
Pre-Compile only	true

⌋

[ECUC_Lin_00094] Definition of EcucReferenceDef LinClockRef ⌈

Parameter Name	LinClockRef	
Parent Container	LinChannel	
Upstream mapping defined	–	
Description	Reference to the LIN clock source configuration, which is set in the MCU driver configuration. Dependency: LIN clock source configuration in MCU Driver	
Multiplicity	1	
Type	Reference to McuClockReferencePoint	
Post-Build Value	Loadable	Selectable

⌋

The configuration parameter LinChannelWakeupSupport can be ignored during validation of wakeup signal.

[SWS_Lin_CONSTR_00278] ⌈The ECUC partitions referenced by LinChannelEcucPartitionRef shall be a subset of the ECUC partitions referenced by LinEcucPartitionRef.⌋

[SWS_Lin_CONSTR_00291] ⌈If LinEcucPartitionRef references one or more ECUC partitions, LinChannelEcucPartitionRef shall have a multiplicity of one and reference one of these ECUC partitions as well.⌋

[SWS_Lin_CONSTR_00279] ⌈LinChannel and LinTrcvChannel of one communication channel shall all reference the same ECUC partition.⌋

10.2.4 LinGlobalConfig

[ECUC_Lin_00184] Definition of EcucParamConfContainerDef LinGlobalConfig ⌈

Container Name	LinGlobalConfig
Parent Container	Lin
Upstream mapping defined	–
Description	This container contains the global configuration parameter of the Lin driver.
Multiplicity	1
Pre-Compile only	true
Configuration Parameters	

No Included Parameters

Included Containers		
Container Name	**Multiplicity**	**Description**
LinChannel	1..*	This container contains the configuration (parameters) of the LIN Controller(s).

⌋

10.2.5 LinDemEventParameterRefs

[ECUC_Lin_00188] Definition of EcucParamConfContainerDef LinDemEventParameterRefs ⌈

Container Name	LinDemEventParameterRefs
Parent Container	Lin
Upstream mapping defined	–
Description	Container for the references to DemEventParameter elements which shall be invoked using the API Dem_SetEventStatus in case the corresponding error occurs. The Event Id is taken from the referenced DemEventParameter's DemEventId symbolic value. The standardized errors are provided in this container and can be extended by vendor-specific error references.

▽

△

Multiplicity	0..1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
LIN_E_TIMEOUT	0..1	[ECUC_Lin_00189]

No Included Containers

⌋

[ECUC_Lin_00189] Definition of EcucReferenceDef LIN_E_TIMEOUT ⌈

Parameter Name	LIN_E_TIMEOUT
Parent Container	LinDemEventParameterRefs
Upstream mapping defined	–
Description	Reference to the DemEventParameter which shall be issued when the error "Timeout caused by hardware error" has occurred.
Multiplicity	0..1
Type	Symbolic name reference to DemEventParameter
Pre-Compile only	true

⌋

10.3 Published Information

For details refer to [3] Chapter 10.3 *“Published Information”*.

A Not applicable requirements

[SWS_Lin_NA_00999]

Upstream requirements: SRS_BSW_00336, SRS_BSW_00339, SRS_BSW_00383, SRS_BSW_00395, SRS_BSW_00397, SRS_BSW_00398, SRS_BSW_00399, SRS_BSW_00400, SRS_BSW_00416, SRS_BSW_00417, SRS_BSW_00422, SRS_BSW_00423, SRS_BSW_00424, SRS_BSW_00425, SRS_BSW_00426, SRS_BSW_00427, SRS_BSW_00428, SRS_BSW_00429, SRS_BSW_00432, SRS_BSW_00433, SRS_BSW_00168

⌈These requirements are not applicable to this specification.⌋

B Change history of AUTOSAR traceable items

B.1 Traceable item history of this document according to AUTOSAR Release R26-11

B.1.1 Added Constraints in R26-11

none

B.1.2 Changed Constraints in R26-11

none

B.1.3 Deleted Constraints in R26-11

none

B.1.4 Added Specification Items in R26-11

Number	Heading
[ECUC_Lin_00194]	Definition of EcucFloatParamDef LinMainFunctionPeriod
[SWS_Lin_00291]	Execution cycle of `Lin_MainFunction`
[SWS_Lin_91141]	Definition of scheduled function Lin_MainFunction

Table B.1: Added Specification Items in R26-11

B.1.5 Changed Specification Items in R26-11

Number	Heading
[ECUC_Lin_00183]	Definition of EcucParamConfContainerDef LinGeneral
[SWS_Lin_00106]	
[SWS_Lin_00226]	Definition of imported datatypes of module Lin

Table B.2: Changed Specification Items in R26-11

B.1.6 Deleted Specification Items in R26-11

Number	Heading
[ECUC_Lin_00067]	Definition of EcucBooleanParamDef LinVersionInfoApi
[SWS_Lin_00001]	
[SWS_Lin_00161]	Definition of API function Lin_GetVersionInfo
[SWS_Lin_00248]	

Table B.3: Deleted Specification Items in R26-11

B.2 Traceable item history of this document according to AUTOSAR Release R25-11

B.2.1 Added Constraints in R25-11

none

B.2.2 Changed Constraints in R25-11

none

B.2.3 Deleted Constraints in R25-11

none

B.2.4 Added Specification Items in R25-11

none

B.2.5 Changed Specification Items in R25-11

Number	Heading
[SWS_Lin_00021]	
[SWS_Lin_00290]	This error is reported when time out caused by hardware error occurs.

Table B.4: Changed Specification Items in R25-11

B.2.6 Deleted Specification Items in R25-11

none

B.3 Traceable item history of this document according to AUTOSAR Release R24-11

B.3.1 Added Constraints in R24-11

none

B.3.2 Changed Constraints in R24-11

none

B.3.3 Deleted Constraints in R24-11

Number	Heading
[SWS_Lin_CONSTR_00270]	

Table B.5: Deleted Constraints in R24-11

B.3.4 Added Specification Items in R24-11

none

B.3.5 Changed Specification Items in R24-11

Number	Heading
[ECUC_Lin_00192]	Definition of EcucReferenceDef LinEcucPartitionRef
[SWS_Lin_00166]	Definition of API function Lin_GoToSleep
[SWS_Lin_00167]	Definition of API function Lin_GoToSleepInternal
[SWS_Lin_00168]	Definition of API function Lin_GetStatus
[SWS_Lin_00169]	Definition of API function Lin_Wakeup
[SWS_Lin_00191]	Definition of API function Lin_SendFrame
[SWS_Lin_00229]	Definition of datatype Lin_FrameCsModelType

▽

△

Number	Heading
[SWS_Lin_00230]	Definition of datatype Lin_FrameResponseType
[SWS_Lin_00233]	Definition of datatype Lin_StatusType
[SWS_Lin_00256]	Definition of API function Lin_WakeupInternal
[SWS_Lin_91140]	Definition of datatype Lin_SlaveErrorType

Table B.6: Changed Specification Items in R24-11

B.3.6 Deleted Specification Items in R24-11

none

B.4 Traceable item history of this document according to AUTOSAR Release R23-11

B.4.1 Added Constraints in R23-11

Number	Heading
[SWS_Lin_CONSTR_00270]	
[SWS_Lin_CONSTR_00278]	
[SWS_Lin_CONSTR_00279]	
[SWS_Lin_CONSTR_00291]	

Table B.7: Added Constraints in R23-11

B.4.2 Changed Constraints in R23-11

none

B.4.3 Deleted Constraints in R23-11

none

B.4.4 Added Specification Items in R23-11

Number	Heading
[SWS_Lin_00001]	
[SWS_Lin_00005]	
[SWS_Lin_00006]	Definition of API function Lin_Init
[SWS_Lin_00008]	
[SWS_Lin_00011]	
[SWS_Lin_00013]	
[SWS_Lin_00014]	
[SWS_Lin_00015]	
[SWS_Lin_00016]	
[SWS_Lin_00017]	
[SWS_Lin_00018]	
[SWS_Lin_00019]	
[SWS_Lin_00021]	
[SWS_Lin_00022]	
[SWS_Lin_00024]	
[SWS_Lin_00025]	
[SWS_Lin_00026]	
[SWS_Lin_00027]	
[SWS_Lin_00028]	
[SWS_Lin_00029]	
[SWS_Lin_00032]	
[SWS_Lin_00033]	
[SWS_Lin_00037]	
[SWS_Lin_00039]	
[SWS_Lin_00043]	
[SWS_Lin_00045]	
[SWS_Lin_00048]	Definiton of development errors in module Lin
[SWS_Lin_00053]	
[SWS_Lin_00054]	
[SWS_Lin_00055]	
[SWS_Lin_00058]	
[SWS_Lin_00060]	
[SWS_Lin_00063]	
[SWS_Lin_00074]	
[SWS_Lin_00084]	
[SWS_Lin_00089]	

▽

△

Number	Heading
[SWS_Lin_00091]	
[SWS_Lin_00092]	
[SWS_Lin_00095]	
[SWS_Lin_00096]	
[SWS_Lin_00097]	
[SWS_Lin_00098]	
[SWS_Lin_00099]	
[SWS_Lin_00105]	
[SWS_Lin_00106]	
[SWS_Lin_00107]	
[SWS_Lin_00129]	
[SWS_Lin_00131]	
[SWS_Lin_00133]	
[SWS_Lin_00135]	
[SWS_Lin_00137]	
[SWS_Lin_00139]	
[SWS_Lin_00140]	
[SWS_Lin_00141]	
[SWS_Lin_00143]	
[SWS_Lin_00144]	
[SWS_Lin_00145]	
[SWS_Lin_00146]	
[SWS_Lin_00150]	
[SWS_Lin_00155]	
[SWS_Lin_00156]	
[SWS_Lin_00157]	
[SWS_Lin_00160]	Definition of API function Lin_CheckWakeup
[SWS_Lin_00161]	Definition of API function Lin_GetVersionInfo
[SWS_Lin_00166]	Definition of API function Lin_GoToSleep
[SWS_Lin_00167]	Definition of API function Lin_GoToSleepInternal
[SWS_Lin_00168]	Definition of API function Lin_GetStatus
[SWS_Lin_00169]	Definition of API function Lin_Wakeup
[SWS_Lin_00171]	
[SWS_Lin_00174]	
[SWS_Lin_00176]	
[SWS_Lin_00177]	
[SWS_Lin_00184]	
[SWS_Lin_00190]	

▽

△

Number	Heading
[SWS_Lin_00191]	Definition of API function Lin_SendFrame
[SWS_Lin_00192]	
[SWS_Lin_00193]	
[SWS_Lin_00194]	
[SWS_Lin_00195]	
[SWS_Lin_00197]	
[SWS_Lin_00198]	
[SWS_Lin_00199]	
[SWS_Lin_00200]	
[SWS_Lin_00201]	
[SWS_Lin_00207]	
[SWS_Lin_00209]	
[SWS_Lin_00210]	
[SWS_Lin_00211]	
[SWS_Lin_00220]	
[SWS_Lin_00221]	
[SWS_Lin_00222]	
[SWS_Lin_00223]	
[SWS_Lin_00224]	
[SWS_Lin_00225]	
[SWS_Lin_00226]	Definition of imported datatypes of module Lin
[SWS_Lin_00227]	Definition of datatype Lin_ConfigType
[SWS_Lin_00228]	Definition of datatype Lin_FramePidType
[SWS_Lin_00229]	Definition of datatype Lin_FrameCsModelType
[SWS_Lin_00230]	Definition of datatype Lin_FrameResponseType
[SWS_Lin_00231]	Definition of datatype Lin_FrameDlType
[SWS_Lin_00232]	Definition of datatype Lin_PduType
[SWS_Lin_00233]	Definition of datatype Lin_StatusType
[SWS_Lin_00234]	Definition of mandatory interfaces in module Lin
[SWS_Lin_00235]	Definition of optional interfaces in module Lin
[SWS_Lin_00238]	
[SWS_Lin_00239]	
[SWS_Lin_00240]	
[SWS_Lin_00245]	
[SWS_Lin_00246]	
[SWS_Lin_00248]	
[SWS_Lin_00251]	
[SWS_Lin_00255]	

▽

△

Number	Heading
[SWS_Lin_00256]	Definition of API function Lin_WakeupInternal
[SWS_Lin_00257]	
[SWS_Lin_00258]	
[SWS_Lin_00259]	
[SWS_Lin_00260]	
[SWS_Lin_00261]	
[SWS_Lin_00262]	
[SWS_Lin_00263]	
[SWS_Lin_00264]	
[SWS_Lin_00265]	
[SWS_Lin_00266]	
[SWS_Lin_00268]	
[SWS_Lin_00269]	
[SWS_Lin_00271]	
[SWS_Lin_00272]	
[SWS_Lin_00273]	
[SWS_Lin_00274]	
[SWS_Lin_00275]	
[SWS_Lin_00276]	
[SWS_Lin_00277]	
[SWS_Lin_00280]	
[SWS_Lin_00281]	
[SWS_Lin_00282]	
[SWS_Lin_00283]	
[SWS_Lin_00284]	
[SWS_Lin_00285]	
[SWS_Lin_00286]	
[SWS_Lin_00287]	
[SWS_Lin_00288]	
[SWS_Lin_00289]	
[SWS_Lin_00290]	
[SWS_Lin_91140]	Definition of datatype Lin_SlaveErrorType
[SWS_Lin_NA_00999]	

Table B.8: Added Specification Items in R23-11

B.4.5 Changed Specification Items in R23-11

none

B.4.6 Deleted Specification Items in R23-11

none