Document Title	Specification of LIN Driver
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	072
Document Classification	Standard

Document Version	1.5.1
Document Status	Final
Part of Release	3.2
Revision	3

Document Change History

Date	Version	Changed by	Change Description
28.02.2014	1.5.1	AUTOSAR Release Management	• Editorial changes • Removed chapter(s) on change documentation
29.05.2012	1.5.0	AUTOSAR Administration	• Updated LIN185 and clarification added • Deleted LIN132 and LIN136
07.04.2011	1.4.0	AUTOSAR Administration	• Added detection that MCU power on was caused by LIN communication (add LIN185) • Corrected instance that calls Lin_Init (LIN255) • Legal disclaimer revised
20.09.2010	1.3.0	AUTOSAR Administration	• Add LIN184 • Legal disclaimer revised
23.06.2008	1.2.1	AUTOSAR Administration	Legal disclaimer revised
11.12.2007	1.2.0	AUTOSAR Administration	• Editorial Changes • Tables generated in Chapter 8 and 10 • Document meta information extended • Small layout adaptations made
30.01.2007	1.1.0	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
11.05.2006	1.0.0	AUTOSAR Administration	Initial release

Disclaimer

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

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

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

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

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Advice for users

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

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

Table of Contents

1 Introduction and functional overview

This 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 LIN 2.0 specification [15]. It is assumed that the reader is familiar with this specification. This document will not describe LIN 2.0 functionality again, but it will try to follow the same order as the LIN 2.0 specification.

The LIN driver applies to LIN 2.0 master nodes only. Operating as a slave node is out of scope. The LIN master in AUTOSAR deviates from the LIN 2.0 specification 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 slaves together with the AUTOSAR LIN master (i.e. the LIN driver).

LIN063: 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.

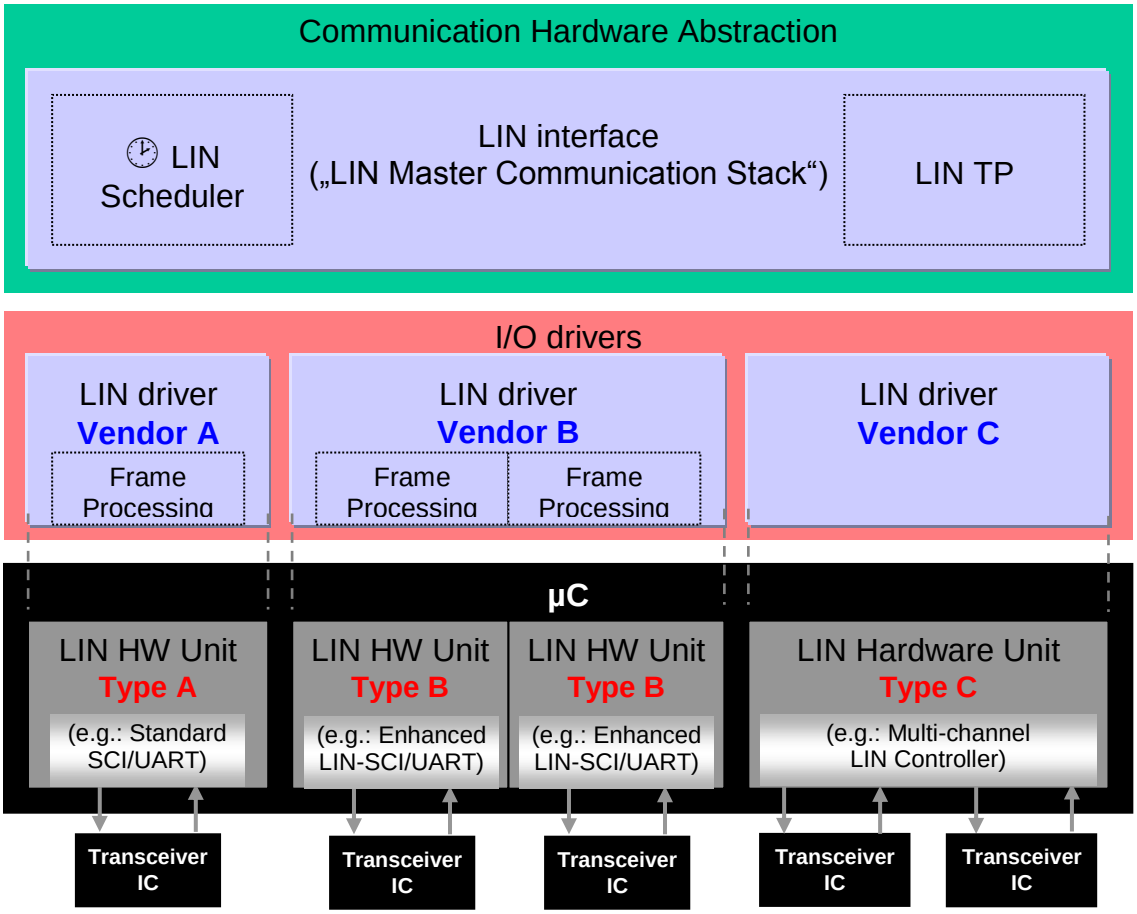

Figure 1-1: Overview LIN Software Architecture Layering

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 AUTOSAR glossary must appear here.

Acronym:	***Description:***
DEM	Diagnostic Event Manager
DET	Development Error Tracer
ISR	Interrupt Service Routine
LIN	Local Interconnect Network (as defined by [15])
MCU	Micro Controller Unit
PDU	Protocol Data Unit. Consists of Identifier, data length and Data (SDU)
PID	Protected ID (as defined by [15])
PLL	Phase-Locked Loop
SCI	Serial Communication Interface
SDU	Service Data Unit. Data that is transported inside the PDU
SFR	Special Function Register
SWS	Software Specification
TP	Transport Layer
UART	Universal Asynchronous Receiver Transmitter

Abbreviation	***Description:***
Id	Identifier

2.2 Glossary

Besides AUTOSAR terminology this document also uses terms defined in the LIN 2.0 specification [15], 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 [15]: “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 [15]: “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 [15]: “A header is the first part of a frame; it is always sent by the master.”
LIN node	As defined by [15]: “Loosely speaking, a node is an ECU. However, a single ECU may be connected to multiple LIN clusters.”
LIN response	As defined by [15]: “A LIN frame consists of a header and a response. Also called a

	Frame response.”

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:

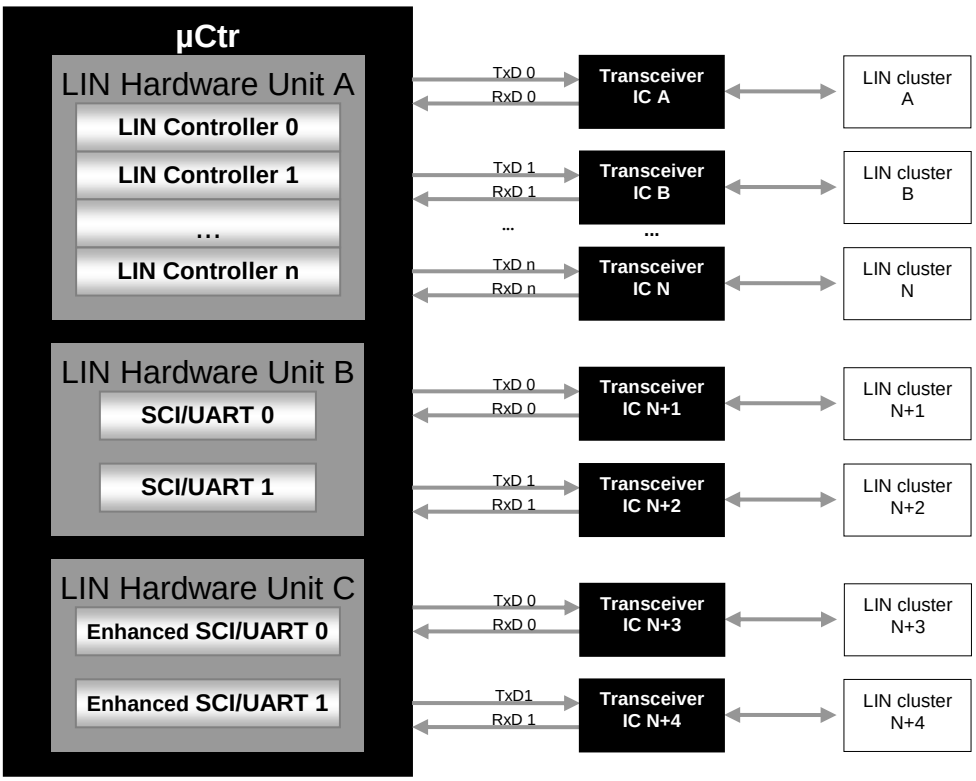

Figure 2-1: LIN hardware unit classification

3 Related documentation

3.1 Input documents

[1] List of Basic Software Modules
AUTOSAR_BasicSoftwareModules.pdf

[2] Layered Software Architecture
AUTOSAR_LayeredSoftwareArchitecture.pdf

[3] General Requirements on Basic Software Modules
AUTOSAR_SRS_General.pdf

[4] Specification of Standard Types
AUTOSAR_SWS_StandardTypes.pdf

[5] Specification of Development Error Tracer
AUTOSAR_SWS_DevelopmentErrorTracer.pdf

[6] General Requirements on SPAL
AUTOSAR_SRS_SPAL_General.pdf

[7] Requirements on LIN
AUTOSAR_SRS_LIN.pdf

[8] Specification of LIN Interface
AUTOSAR_SWS_LIN_Interface.pdf

[9] Specification of ECU Configuration
AUTOSAR_ECU_Configuration.pdf

[10] Specification of MCU driver
AUTOSAR_SWS_MCU_Driver.pdf

[11] Specification of Diagnostics Event Manager
AUTOSAR_SWS_DEM.pdf

[12] Specification of C Implementation Rules
AUTOSAR_SWS_C_ImplementationRules.pdf

[13] Specification of ECU State Manager
AUTOSAR_SWS_ECU_StateManager.pdf

[14] AUTOSAR Basic Software Module Description Template,
AUTOSAR_BSW_Module_Description.pdf

3.2 Related standards and norms

[15] LIN Specification Package Revision 2.0, September 23, 2003
http://www.lin-subbus.org/

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.

Driver scope

LIN045: One LIN driver provides access to one LIN hardware unit type (simple UART or dedicated LIN hardware) that may consist of several LIN channels. 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.

LIN177: 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 BSW00347 with a Vendor Id (in case of several LIN drivers from different vendors) and a vendor specific name (in case of different hardware units are implemented by one Vendor). Any combination of these extensions is possible.

The LIN Interface is responsible for calling the correct function. The necessary information shall be given in an XML file during configuration. See [8] 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 [10]
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. PLL on → PLL off) 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 (Development Error Tracer) [5]
In development mode, the Lin module reports development error through the Det_ReportError function of module DET.

Module DEM (Diagnostic Event Manager) [11]
The Lin module reports production errors to the Diagnostic Event Manager

OS (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 (specified by [8]) is the only user of the LIN driver services.

5.1 File structure

5.1.1 Code file structure

LIN064: The code file structure shall not be defined within this specification completely. At this point it shall be pointed out that the code-file structure shall include the following files named:

- Lin_Lcfg.c – for link time configurable parameters and
- Lin_PBcfg.c – for post build time configurable parameters.

These files shall contain all link time and post-build time configurable parameters.

5.1.2 Header file structure

LIN075: The include file structure shall be as follows:

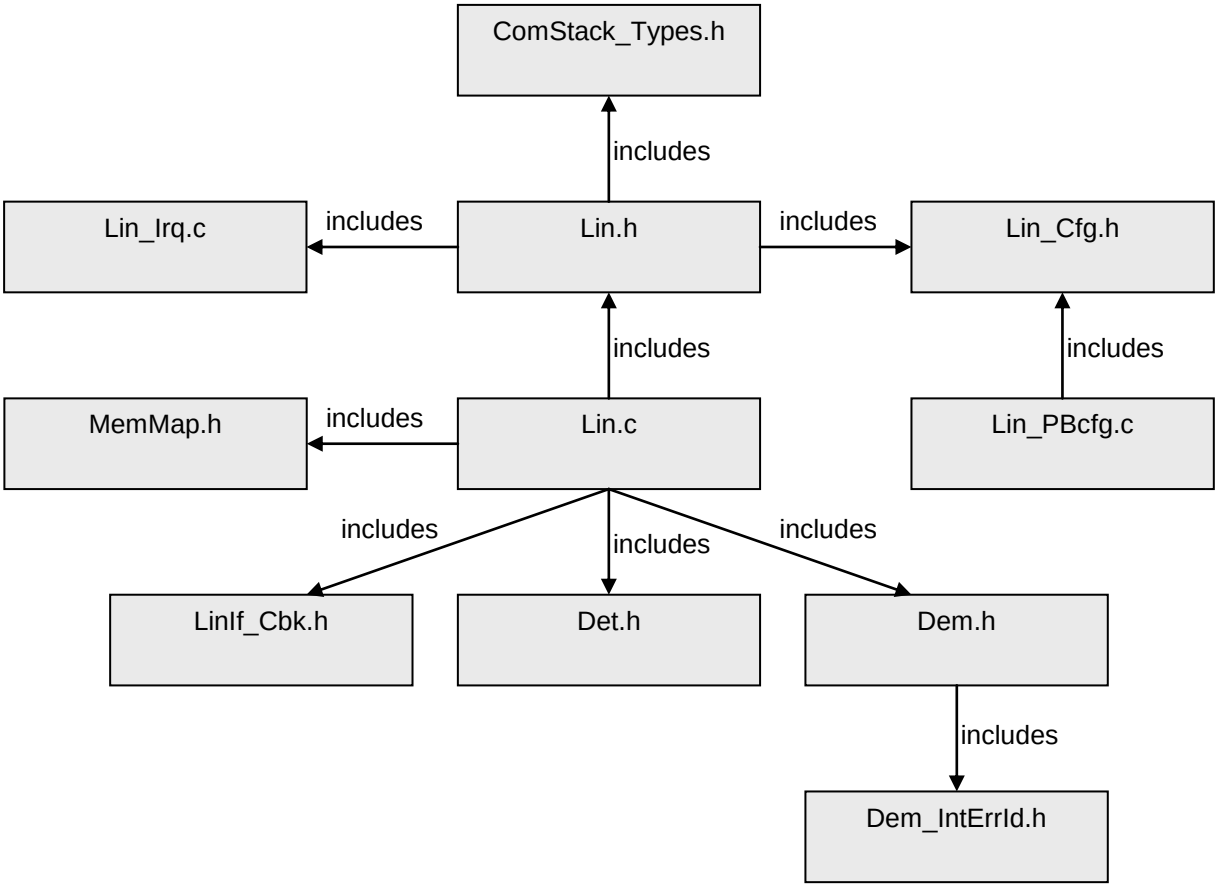

Figure 5-1: Header File structure for the LIN driver

- `Lin.c` shall include `Lin.h`
- `Lin.c` shall include `MemMap.h`
- `Lin.h` shall include `Lin_Cfg.h`
- `Lin.h` shall include `ComStack_Types.h`

LIN023: The module Lin_Irq.c contains the implementation of interrupt frames.
The implementation of the interrupt service routine shall be in Lin.c

LIN042: The header file LinIf_Cbk.h contains the declarations of the callback functions imported by the modules calling the callbacks. The LIN driver itself does not provide callback functions (no Lin_Cbk.h)

LIN054: The file Lin.h only contains external declarations of constants, global data, type definitions and services that are specified in the LIN driver SWS.
Constants, global data types and functions that are only used by LIN driver internally, are declared in Lin.c

LIN065: The module shall include the Dem.h file. By this inclusion the APIs to report errors as well as the required Event Id symbols are included.
This specification defines the name of the Event Id symbols which are provided by XML to the DEM configuration tool. The DEM configuration tool assigns ECU dependent values to the Event Id symbols and publishes the symbols in Dem_IntErrId.h.

6 Requirements traceability

Document: AUTOSAR requirements on Basic Software, general [3]

Requirement	***Satisfied by***
[BSW003] Version identification	Software Documentation Requirements are not covered in the LIN driver SWS
[BSW00300] Module naming convention	Fulfilled by the function name definitions in Chapter 8.3
[BSW00301] Limit imported information	See Chapter 5.1.2
[BSW00302] Limit exported information	LIN054
[BSW00304] AUTOSAR integer data types	LIN047, Chapter 8.2 and Chapter 10.3
[BSW00305] Self-defined data types naming convention	Fulfilled by the function name definitions in Chapter 8.2
[BSW00306] Avoid direct use of compiler and platform specific keywords	LIN055
[BSW00307] Global variables naming convention	Not applicable (requirement on implementation)
[BSW00308] Definition of global data	LIN055
[BSW00309] Global data with read-only constraint	LIN055
[BSW00310] API naming convention	See Chapter 5.1.2
[BSW00312] Shared code shall be reentrant	Not applicable
[BSW00314] Separation of interrupt frames and service routines	LIN023
[BSW00318] Format of module version numbers	LIN002
[BSW00321] Enumeration of module version numbers	LIN002
[BSW00323] API parameter checking	LIN048, LIN049
[BSW00325] Runtime of interrupt service routines	Not applicable (requirement on implementation)
[BSW00326] Transition from ISRs to OS tasks	Not applicable (requirement on implementation)
[BSW00327] Error values naming convention	LIN048
[BSW00328] Avoid duplication of code	Not applicable (requirement on implementation, fulfilled e.g. by defining a LIN driver that controls multiple channels)
[BSW00329] Avoidance of generic interfaces	Not applicable (no generic interfaces specified within this SWS)
[BSW00330] Usage of macros / inline functions instead of functions	Not applicable (requirement on implementation)
[BSW00331] Separation of error and status values	Not applicable
[BSW00333] Documentation of callback function context	Software Documentation Requirements are not covered in the LIN driver SWS
[BSW00334] Provision of XML file	Software Documentation Requirements are not covered in the LIN driver SWS
[BSW00335] Status values naming convention	Fulfilled by the state diagram description in chapter 7.3.3
[BSW00336] Shutdown interface	Not applicable
[BSW00337] Classification of errors	LIN048
[BSW00338] Detection and Reporting of development errors	LIN049, LIN052
[BSW00339] Reporting of production relevant error status	Not applicable
[BSW00341] Microcontroller compatibility documentation	Software Documentation Requirements are not covered in the LIN driver SWS

[BSW00342] Usage of source code and object code	Not applicable (requirement on implementation)
[BSW00343] Specification and configuration of time	Not applicable
[BSW00344] Reference to link-time configuration	LIN013
[BSW00345] Pre-compile-time configuration	See Chapter10
[BSW00346] Basic set of module files	See Chapter 5.1.2
[BSW00347] Naming separation of different instances of BSW drivers	LIN045
[BSW00348] Standard type header	See Chapter 5.1.2
[BSW00350] Development error detection keyword	LIN066
[BSW00353] Platform specific type header	Not applicable (automatically included with standard types)
[BSW00355] Do not redefine AUTOSAR integer data types	no redefined integer types in Chapter 8.2 and Chapter 10.3
[BSW00357] Standard API return type	Not applicable (this type is not used within this SWS)
[BSW00358] Return type of init() functions	fulfilled by 8.3.1.1
[BSW00359] Return type of callback functions	Not applicable (no callback function specified)
[BSW00360] Parameters of callback functions	Not applicable (no callback function specified)
[BSW00361] Compiler specific language extension header	Not applicable (automatically included with standard types)
[BSW00369] Do not return development error codes via API	LIN059
[BSW00370] Separation of callback interface from API	LIN042
[BSW00371] Do not pass function pointers via API	Fulfilled by the function definitions in Chapter 8.3
[BSW00373] Main processing function naming convention	Not applicable (no main processing function specified)
[BSW00374] Module vendor identification	LIN002
[BSW00375] Notification of wake-up reason	LIN041
[BSW00376] Return type and parameters of main processing functions	Not applicable (no main processing function specified)
[BSW00377] Module specific API return types	See 8.2.8
[BSW00378] AUTOSAR boolean type	Not applicable (not used)
[BSW00379] Module identification	LIN002
[BSW00380] Separate C-File for configuration parameters	LIN064
[BSW00381] Separate configuration header file for pre-compile time parameters	See Chapter 5.1.2
[BSW00383] List dependencies of configuration files	Not applicable (implementation specific documentation)
[BSW00384] List dependencies to other modules	See Chapter 5
[BSW00385] List possible error notificatons	LIN048
[BSW00386] Configuration for detecting an error	See Chapter 7.6
[BSW00387] Specify the configuration class of callback function	Chapter 8.6.3
[BSW00388] Introduce containers	See Chapter 10.2
[BSW00389] Containers shall have names	See Chapter 10.2
[BSW00390] Parameter content shall be unique within the module	See Chapter 8
[BSW00391] Parameter shall have unique names	fulfilled by parameter definitions in Chapter 10.2
[BSW00392] Parameters shall have a type	fulfilled by parameter definitions in Chapter 10.2

[BSW00393] Parameters shall have a range	fulfilled by parameter definitions in Chapter 10.2
[BSW00394] Specify the scope of the parameters	fulfilled by parameter definitions in Chapter 10.2
[BSW00395] List the required parameters (per parameter)	Not applicable (parameters are defined in a way that their values are independent from other settings. The dependency is in the code generation (implementation) not in the configuration description -> hardware abstraction)
[BSW00396] Configuration classes	fulfilled by parameter definitions in Chapter 10.2
[BSW00397] Pre-compile-time parameters	Not applicable (this is not a requirement, but a definition of a technical term)
[BSW00398] Link-time parameters	Not applicable (this is not a requirement, but a definition of a technical term)
[BSW00399] Loadable Post-build time parameters	Not applicable (this is not a requirement, but a definition of a technical term)
[BSW004] Version check	LIN062
[BSW00400] Selectable Post-build time parameters	Not applicable (this is not a requirement, but a definition of a technical term)
[BSW00401] Documentation of multiple instances of configuration parameters	Software Documentation Requirements are not covered in the LIN driver SWS
[BSW00402] Published information	LIN002
[BSW00404] Reference to post build time configuration	LIN013
[BSW00405] Reference to multiple configuration sets	LIN011, LIN012, LIN013
[BSW00406] Check module initialization	LIN006
[BSW00407] Function to read out published parameters	LIN001
[BSW00408] Configuration parameter naming convention	fulfilled by Chapter 10.2
[BSW00409] Header files for production code error IDs	LIN065, LIN046
[BSW00410] Compiler switches shall have specified values	fulfilled by Chapter 10.2
[BSW00411] Get version info keyword	LIN066 and 8.3.1.3
[BSW00412] Separate H-File for configuration parameters	See Chapter 5.1.2
[BSW00413] Accessing instances of BSW modules	Not applicable (this requirement has to fulfilled by the LIN Interface
[BSW00414] Parameter of init function	fulfilled by 8.3.1.1
[BSW00415] User dependent include files	Not applicable (only one user for this module)
[BSW00416] Sequence of Initialization	Not applicable (this is a general software integration requirement)
[BSW00417] Reporting of Error Events by Non-Basic Software	Not applicable (LIN driver is a Basic Software Module)
[BSW00419] Separate C-Files for pre-compile time configuration parameters	LIN064
[BSW00420] Production relevant error event rate detection	Not applicable (requirement on the DEM)
[BSW00421] Reporting of production relevant error events	LIN058
[BSW00422] Debouncing of production relevant error status	Not applicable (requirement on the DEM)

[BSW00423] Usage of SW-C template to describe BSW modules with AUTOSAR Interfaces	Not applicable (this module does not provide an AUTOSAR interface)
[BSW00424] BSW main processing function task allocation	Not applicable (requirement on system design, not on a single module)
[BSW00425] Trigger conditions for schedulable objects	Not applicable (trigger conditions are system configuration specific)
[BSW00426] Exclusive areas in BSW modules	Not applicable
[BSW00427] ISR description for BSW modules	Not applicable (no ISR defined for this module, usage of interrupts are implementation specific)
[BSW00428] Execution order dependencies of main processing functions	Not applicable (LIN driver does not contain any main processing functions)
[BSW00429] Restricted BSW OS functionality access	Not applicable (implementation requirement, not for the specification)
[BSW00431] The BSW Scheduler module implements task bodies	Not applicable (applies only to BSW scheduler module)
[BSW00432] Modules should have separate main processing functions for read/receive and write/transmit data path	Not applicable (no main processing function specified)
[BSW00433] Calling of main processing functions	Not applicable (requirement on system design, not on a single module)
[BSW00434] The Schedule Module shall provide an API for exclusive areas	Not applicable (applies only to BSW scheduler module)
[BSW005] No hard coded horizontal interfaces within MCAL	Not applicable (fulfilled by the AUTOSAR architectural concept)
[BSW006] Platform independency	LIN003
[BSW007] HIS MISRA C	Not applicable (requirement on implementation)
[BSW009] Module User Documentation	Software Documentation Requirements are not covered in the LIN driver SWS
[BSW010] Memory resource documentation	Software Documentation Requirements are not covered in the LIN driver SWS
[BSW101] Initialization interface	LIN006
[BSW158] Separation of configuration from implementation	See Chapter 5.1.2
[BSW159] Tool-based configuration	LIN029
[BSW160] Human-readable configuration data	LIN031
[BSW161] Microcontroller abstraction	LIN003
[BSW162] ECU layout abstraction	Not applicable (fulfilled by the AUTOSAR architectural concept)
[BSW164] Implementation of interrupt service routines	LIN155
[BSW167] Static configuration checking	LIN039
[BSW168] Diagnostic Interface of SW components	Not applicable (LIN driver doesn't offer a diagnostic interface)
[BSW170] Data for reconfiguration of AUTOSAR SW-Components	See Chapter10

[BSW171] Configurability of optional functionality	LIN066, LIN067
[BSW172] Compatibility and documentation of scheduling strategy	Software Documentation Requirements are not covered in the LIN driver SWS

Document: AUTOSAR requirements on Basic Software, Cluster: SPAL general [6]

Requirement	***Satisfied by***
[BSW12263] Object code compatible configuration concept	LIN013
[BSW12056] Configuration of notification mechanisms	Not applicable
[BSW12267] Configuration of wake-up sources	Not applicable
[BSW12057] driver module initialization	LIN006
[BSW12125] Initialization of hardware resources	LIN006, LIN007
[BSW12163] driver module deinitialization	LIN009
[BSW12461] Responsibility for register initialization	LIN008
[BSW12462] Provide settings for register initialization	See Chapter 10.3
[BSW12463] Combine and forward settings for register initialization	Not applicable (applies only for configurator)
[BSW12068] MCAL initialization sequence	Not applicable
[BSW12069] Wake-up notification of ECU State Manager	LIN041
[BSW157] Notification mechanisms of drivers and handlers	LIN022, LIN052, LIN053
[BSW12169] Control of operation mode	LIN032
[BSW12063] Raw value mode	LIN016, LIN025
[BSW12075] Use of application buffers	Not applicable (LIN driver does not feature random streaming capability)
[BSW12129] Resetting of interrupt flags	LIN157
[BSW12064] Change of operation mode during running operation	LIN032
[BSW12448] Behavior after development error detection	LIN052, LIN059
[BSW12067] Setting of wake-up conditions	LIN032
[BSW12077] Non-blocking implementation	LIN027, LIN028.
[BSW12078] Runtime and memory efficiency	Not applicable because this is a non-functional requirement
[BSW12092] Access to drivers	Not applicable because this is a non-functional requirement
[BSW12265] Configuration data shall be kept constant	LIN013 (stored in ROM → implicitly constant)
[BSW12264] Specification of configuration items	See Chapter10

Document: AUTOSAR requirements on Basic Software, Cluster: LIN [7]

Requirement	***Satisfied by***
[BSW01501] Usage of LIN 2.0 specification	LIN005, LIN070LIN016
[BSW01504] Usage of AUTOSAR architecture only in LIN master nodes	LIN005LIN070
[BSW01522] Consistent data transfer	LIN025, LIN053, LIN060
[BSW01560] Support for wake-up during transition to sleep-mode	LIN033, LIN034, LIN035
[BSW01567] Compatibility to LIN 2.0 protocol specification	Not applicable for the LIN driver
[BSW01551] Multiple LIN channel support for interface	Not applicable for the LIN driver
[BSW01568] Hardware independence	Not applicable for the LIN driver
[BSW01569] LIN Interface initialization	Not applicable for the LIN driver
[BSW01570] Selection of static configuration sets	Not applicable for the LIN driver
[BSW01564] Schedule Table Manager	Not applicable for the LIN driver

[BSW01546] Schedule Table Handler	Not applicable for the LIN driver
[BSW01561] Main function	Not applicable for the LIN driver
[BSW01549] Timer service for Scheduling	Not applicable for the LIN driver
[BSW01571] Transmission request service	Not applicable for the LIN driver
[BSW01514] Wake-up notification support	Not applicable for the LIN driver
[BSW01515] API to wake-up by upper layer to LIN Interface	Not applicable for the LIN driver
[BSW01502] RX indication and TX confirmation call-backs	Not applicable for the LIN driver
[BSW01558] Check successful communication	Not applicable for the LIN driver
[BSW01527] Notification for missing or erroneous receive LIN-PDU	Not applicable for the LIN driver
[BSW01523] API to send the LIN to sleep-mode	Not applicable for the LIN driver
[BSW01565] Compatibility to LIN 2.0 protocol specification	LIN005, LIN016
[BSW01553] Basic Software SPAL General Requirements	LIN004
[BSW01552] Hardware abstraction LIN	LIN003
[BSW01503] Frame based API for send and received data	LIN024, LIN025
[BSW01555] LIN Interface shall poll the LIN driver for transmit/receive notifications	LIN024
[BSW01547] Support of standard UART and LIN optimized HW	LIN063
[BSW01572] LIN driver initialization	LIN009, LIN011
[BSW01573] Selection of static configuration sets	LIN011, LIN012
[BSW01563] Wake-up Notification	LIN041
[BSW01556] Multiple LIN channel support for driver	LIN007, LIN008, LIN009
[BSW01566] Transition to sleep-mode	LIN033, LIN034, LIN035, LIN073
[BSW01524] Support of reduced power operation mode	LIN032
[BSW01526] Error notification	LIN052, LIN053
[BSW01533] Compatibility to TP of LIN 2.0 specification	Not applicable for the LIN driver
[BSW01540] LIN Transport Layer Initialization	Not applicable for the LIN driver
[BSW01545] LIN Transport Layer Availability	Not applicable for the LIN driver
[BSW01534] Concurrent connection configuration	Not applicable for the LIN driver
[BSW01574] Multiple Transport Layer instances	Not applicable for the LIN driver
[BSW01539] Transport connection properties	Not applicable for the LIN driver
[BSW01544] Error handling	Not applicable for the LIN driver

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 LIN2.0 specification. 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.

LIN004: The Lin module shall fulfill the requirements for Basic Software Modules as specified in [6].

LIN005: The Lin module shall conform to the LIN 2.0 Protocol Specification as specified in [15]. This applies to LIN 2.0 Master nodes only.

Operating as a slave node is out of scope for this AUTOSAR LIN driver specification.

LIN055: The Lin module shall fulfill all design and implementation guidelines as described in [12].

LIN155: The Lin module shall implement the ISRs for all LIN hardware unit interrupts that are needed.

LIN156: The Lin module shall ensure that all unused interrupts are disabled.

LIN157: The Lin module shall reset the interrupt flag at the end of the ISR (if not done automatically by hardware).

LIN158: The Lin module shall not configure the interrupt (i.e. priority) nor set the vector table entry.

7.2 Version Check

7.2.1 Requirements

LIN062: The Lin module shall avoid the integration of incompatible files by the following pre-processor checks:

For included header files:

- <MODULENAME>_AR_MAJOR_VERSION
- <MODULENAME>_AR_MINOR_VERSION

shall be identical.

For the module internal c and h files:

- LIN_SW_MAJOR_VERSION
- LIN_SW_MINOR_VERSION
- LIN_AR_MAJOR_VERSION
- LIN_AR_MINOR_VERSION
- LIN_AR_PATCH_VERSION

shall be identical.

7.3 LIN driver and Channel Initialization

7.3.1 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 (→ 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.

Each channel must also be initialized according to the configuration supplied. This will include (but is not limited to) the baud rate over the bus. For this purpose, the LIN driver provides a LIN channel specific initialization function (→ Lin_InitChannel).

LIN225: There must be at least one statically defined configuration set available for the LIN driver. When the EcuM invokes the initialization functions, it has to provide a specific pointer to the configuration that it wishes to use.

The LIN driver also provides a function to 'disable' each LIN channel separately (→ Lin_DeInitChannel).

7.3.2 Requirements

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

LIN011: The Lin module's configuration shall include a data communication rate set as defined by static configuration data (→Lin_ChannelConfigType).

LIN012: The Lin module shall allow the environment to select between different static configuration data at runtime(→Lin_InitChannel).

LIN013: The Lin module's configuration data, intended for hardware registers, shall be stored as hardware specific data structures in ROM (→Lin_ConfigType, Lin_ChannelConfigType).

LIN014: 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) (→Lin_PduType).

LIN015: Each LIN PID shall be associated with a response data length in bytes (→ Lin_PduType).

7.3.3 State diagrams

The LIN driver has a state machine that is shown in Figure 7-1.

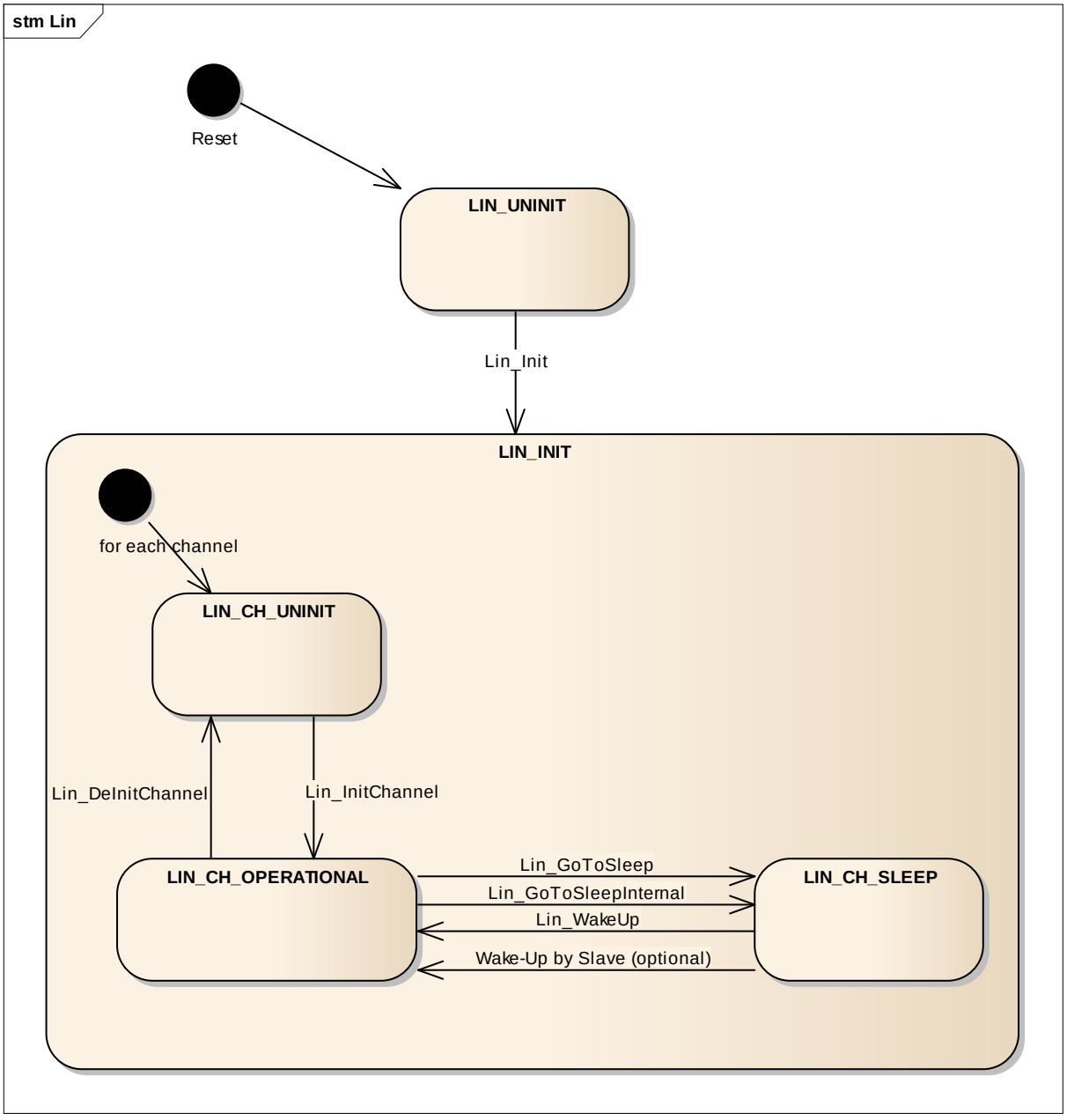

Figure 7-1: LIN driver states

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.

Channel State	Meaning / Activities in the state
LIN_CH_UNINIT	When a channel is in state LIN_CH_UNINIT, the LIN driver is initialized but the LIN channel is not initialized.
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.

LIN145: Reset -> LIN_UNINIT: After reset, the Lin module shall set its state to `LIN_UNINIT`.

LIN146: 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 runtime.

LIN171: On entering the state LIN_INIT, the Lin module shall set each channel into state LIN_CH_UNINIT.

The LIN module's environment must initialize each LIN channel separately by calling the function `Lin_InitChannel`.

LIN147: LIN_CH_UNINIT -> LIN_CH_OPERATIONAL: The function `Lin_InitChannel` shall set the LIN channel state of the referenced channel to `LIN_CH_OPERATIONAL`.

LIN172: LIN_CH_OPERATIONAL -> LIN_CH_SLEEP: If a go to sleep is requested by the LIN interface (`Lin_GoToSleep`), 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` state.

LIN173: LIN_CH_SLEEP -> LIN_CH_OPERATIONAL through Wake-Up by Slave: if a LIN channel is in the state LIN_CH_SLEEP and upon detection of a valid wake-up pulse onto the bus, the Lin module shall put the LIN channel into the state

LIN_CH_OPERATIONAL. The LIN 2.0 specification describes this ‘wake-up’ as a dominant state on the bus lasting between 250µs and 5ms. The activity during `LIN_CH_SLEEP` is to detect a dominant pulse, which shall be handled as valid wake-up request after 150 µs at the last. If such a wake-up was received from the bus, the master node has to begin communication to determine why the wake-up occurred. The form and content of this communication is outside the scope of the LIN driver specification.

A wake-up may also be directly requested from a higher layer in the AUTOSAR architecture (the LIN Interface layer will directly communicate this to the driver).

LIN174: 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. In this case, 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.

LIN184: A mode switch request to the current mode is allowed and shall not lead to an error, even if DET is enabled.

7.4 Frame processing

7.4.1 Background & Rationale

From the point of view of the LIN driver module, transmissions are composed of two actions; the transmission of the LIN header, and the transmission of the response. Only the LIN master node transmits the LIN header, but either the master or one of the slaves may transmit the response [15].

The driver must also be able to access data concerning the checksum model and data length for each LIN PID. LIN2.0 has a different checksum model compared to LIN1.3, but the LIN2.0 master must be able to communicate with both LIN1.3 and LIN2.0 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 [15].

The LIN driver module works with LIN frames as its basic building block. This means that 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_MASTER_RESPONSE) will be given at the same time as the request to send the header. The transmission of the response itself has to be triggered subsequently by another function call.

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.

7.4.2 Requirements

LIN016: The LIN driver shall interpret the supplied identifier as PID. The identifier is then transmitted *as-supplied* within the LIN header (→ Lin_SendHeader).

LIN017: 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 [15] (→ Lin_SendHeader).

LIN018: The LIN driver shall be able to send a LIN header and response.

LIN019: 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.

LIN021: The LIN driver shall abort the current frame transmission if a new frame transmission is requested by the LIN interface (→Lin_SendHeader), also if an ongoing transmission may be still in progress or unsuccessfully completed.

LIN022: The function Lin_GetStatus shall return the status of the current frame transmission request.

LIN024: 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 (→Lin_GetStatus).

LIN025: The LIN driver shall send response data as provided by the LIN interface module (→Lin_SendResponse).

LIN026: 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.

LIN027: 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).

LIN028: The LIN driver shall receive data without blocking.

7.4.3 Data Consistency

Transmit Data Consistency:

LIN053: The LIN driver shall directly copy the data from the upper layer buffers. It is the responsibility of the upper layer to keep the buffer consistent until return of function call.

Receive Data Consistency:

LIN060: The complete LIN frame receive processing (including copying to destination layer) can be implemented in various solutions, for instance with ISR or with the Lin_GetStatus function. Whether with ISR or with Lin_GetStatus function, in any case 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 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.

LIN102: 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.

7.4.4 Data byte mapping

LIN096: 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).

7.5 Sleep and wake-up functionality

7.5.1 Background & Rationale

The 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). 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.

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

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

7.5.2 Requirements

LIN032: When the LIN channel is requested to enter sleep mode it shall perform the transition to low-power mode of the LIN hardware unit (if available) (→ Lin_GoToSleep/Lin_GoToSleepInternal).

LIN033: Each LIN channel shall be able to accept a sleep request independently of the other channel states (→ Lin_GoToSleep/Lin_GoToSleepInternal).

LIN035: The LIN channel shall activate the wake-up detection as soon as possible after completion of the go-to-sleep-command when the LIN bus becomes idle.

LIN037: When a LIN channel is in `LIN_CH_SLEEP` state, the LIN hardware unit shall monitor the bus for a wake-up request on that channel.

LIN040: If a wake-up request was received, the LIN driver shall change state to `LIN_CH_OPERATIONAL` for the channel that received the wake-up pulse.

LIN041: If a wake-up request was received, the LIN driver shall notify via a callback within interrupt context the upper layer (LIN interface) immediately. This notification must identify the channel from where the wake-up was detected.

LIN043: If the LIN driver receives a wake-up request from the LIN interface, the requested channel shall send a wake-up pulse to the bus (→ Lin_WakeUp) and the wake-up detection of bus wake-up events has to be disabled.

The function Lin_GetStatus returns the current state of a given LIN channel.

7.6 Error classification

The error classification depends on the time of error occurrence according to product life cycle:

- Development Errors
  Those errors shall be detected and fixed during development phase. In most cases, those errors are software errors. The detection of errors that shall only occur during development can be switched off for production code (by static configuration namely pre-processor switches).

- Production Errors
  Those errors are hardware errors and software exceptions that cannot be avoided and are also expected to occur in production code.

LIN046: Values for production code Event Ids are assigned externally by the configuration of the DEM. They are published in the file Dem_IntErrId.h and included via Dem.h.

LIN047: Development error values are of type uint8.

LIN048: The following errors and exceptions shall be detectable by the LIN driver depending on its build version (development/production mode)

Type or error	*Relevance*	*Related error code*	*Value [hex]*
API service used without module initialization	Development	LIN_E_UNINIT LIN_E_CHANNEL_UNINIT	0x00 0x01
API service used with an invalid or inactive channel parameter	Development	LIN_E_INVALID_CHANNEL	0x02
API service called with invalid configuration pointer	Development	LIN_E_INVALID_POINTER	0x03
Invalid state transition for the current state	Development	LIN_E_STATE_TRANSITION	0x04
Timeout caused by hardware error	Production	LIN_E_TIMEOUT	Assigned by DEM

7.7 Error detection

LIN049: The detection of development errors is configurable (*ON/OFF*) at pre-compile time. The switch *LinDevErrorDetect* (see chapter 10) shall activate or deactivate the detection of all development errors.

LIN050: If the *LinDevErrorDetect* switch is enabled API parameter checking is enabled.

LIN051: The detection of production code errors cannot be switched off.

LIN097: 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 the 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.

7.8 Error notification

LIN052: Detected development errors shall be reported to the *Det_ReportError* service of the Development Error Tracer (DET) if the pre-processor switch *LinDevErrorDetect* is set (see chapter 10).

LIN058: Production errors shall be reported to Diagnostic Event Manager (DEM) by calling the function Dem_ReportErrorStatus. The only production error that can be reported by the LIN driver is the LIN_E_TIMEOUT error.

8 API specification

8.1 Imported types

In this chapter all types included from other modules are listed:

Module	*Imported Type*
Dem	Dem_EventIdType
EcuM	EcuM_WakeupSourceType
Icu	Icu_ChannelType
Std_Types	Std_ReturnType
	Std_VersionInfoType

8.2 Type definitions

8.2.1 Lin_ConfigType

Name:	`Lin_ConfigType`	
Type:	`Structure`	
Range:	`Hardware and Implementation dependent structure`	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 SFR settings affecting all LIN channels. A pointer to such a structure is provided to the LIN driver initialization routine for configuration of the driver and LIN hardware unit.	

8.2.2 Lin_ChannelConfigType

Name:	`Lin_ChannelConfigType`	
Type:	`Structure`	
Range:	`Hardware and Implementation dependent structure`	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 one LIN Channel. A pointer to such a structure is provided to the LIN channel initialization routine for configuration of the LIN hardware channel.	

8.2.3 Lin_FramePidType

Name:	`Lin_FramePidType`		
Type:	`uint8`		
Range:	`0...0xFE`	--	The LIN identifier (0…0x3F) together with its two parity bits.
Description:	Represents all valid protected Identifier used by Lin_SendHeader().		

8.2.4 Lin_FrameCsModelType

Name:	Lin_FrameCsModelType	
Type:	Enumeration	
Range:	LIN_ENHANCED_CS	Enhanced checksum model
	LIN_CLASSIC_CS	Classic checksum model
Description:	This type is used to specify the Checksum model to be used for the LIN Frame.	

8.2.5 Lin_FrameResponseType

Name:	Lin_FrameResponseType	
Type:	Enumeration	
Range:	LIN_MASTER_RESPONSE	Response is generated from this (master) node
	LIN_SLAVE_RESPONSE	Response is generated from a remote slave node
	LIN_SLAVE_TO_SLAVE	Response is generated from one slave to another slave, for the master the response will be anonymous, it does not have to receive the response.
Description:	This type is used to specify whether the frame processor is required to transmit the response part of the LIN frame.	

8.2.6 Lin_FrameDlType

Name:	Lin_FrameDIType		
Type:	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.		

8.2.7 Lin_PduType

Name:	Lin_PduType		
Type:	Structure		
Element:	Lin_FramePidType	Pid	--
	Lin_FrameCsModelType	Cs	--
	Lin_FrameResponseType	Drc	--
	Lin_FrameDIType	DI	--
	uint8*	SduPtr	--
Description:	This Type is used to provide PID, checksum model, data length and SDU pointer from the LIN Interface to the LIN driver.		

8.2.8 Lin_StatusType

Name:	Lin_StatusType	
Type:	Enumeration	
Range:	LIN_NOT_OK	LIN frame operation return value. Development or production error occurred
	LIN_TX_OK	LIN frame operation return value. Successful transmission.
	LIN_TX_BUSY	LIN frame operation return value. Ongoing transmission (Header or Response).
	LIN_TX_HEADER_ERROR	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	LIN frame operation return value. Erroneous response transmission such as:

		- Mismatch between sent and read back data - Physical bus error
	LIN_RX_OK	LIN frame operation return value. Reception of correct response.
	LIN_RX_BUSY	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	LIN frame operation return value. Erroneous response reception such as: - Framing error - Overrun error - Checksum error or - Short response
	LIN_RX_NO_RESPONSE	LIN frame operation return value. No response byte has been received so far.
	LIN_CH_UNINIT	LIN channel state return value. LIN channel not initialized.
	LIN_CH_OPERATIONAL	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	LIN channel state return value. Sleep mode operation; in this mode 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().	

LIN101: Lin_StatusType: The LIN channel state return value LIN_CH_OPERATIONAL and all LIN frame operation return values can be indicated only, if the LIN channel state-machine is in state LIN_CH_OPERATIONAL.

8.3 Function definitions

This is a list of functions provided for upper layer modules.

8.3.1 Services affecting the complete LIN hardware unit

8.3.1.1 Lin_Init

LIN006:

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.

LIN084: The function Lin_Init shall initialize the Lin module, i.e. static variables, including flags and LIN HW Unit global hardware settings.

Different sets of static configuration may have been configured.

LIN150: The function Lin_Init shall initialize the module according to the configuration set pointed to by the parameter Config.

LIN008: The function Lin_Init shall invoke initializations for relevant hardware register settings common to all channels available on the LIN hardware unit.

LIN106: The Lin module's environment shall not call any function of the Lin module before having called Lin_Init.

LIN099: 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.

LIN105: 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.

8.3.1.2 Lin_WakeupValidation

LIN160:

Service name:	Lin_WakeupValidation

Syntax:	`void Lin_WakeupValidation(`
	
	`)`

Service ID[hex]:	0x0a
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	Identifies LIN channels.

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.

After a wake up caused by LIN bus Transceiver the function Lin_WakeUpValidation will be called by the LIN Interface module to identify the corresponding LIN channel (e.g. in case of multiple transceivers are physically connected to one MCU wake up pin). In this case, LIN Driver only plays a role on validation of this wake up signal.

LIN098: The function Lin_WakeUpValidation shall evaluate each connected LIN channel inside the LIN Driver implementation individually. When a wake-up event on an individual channel (e.g. RxD pin has constant low level) is detected, the function Lin_WakeUpValidation shall notify the ECU State Manager module immediately via the EcuM_SetWakeupEvent call-back function.

LIN107: If development error detection for the LIN module is enabled: if the function Lin_WakeUpValidation is called before the LIN module was initialized, the function Lin_WakeUpValidation shall raise the development error LIN_E_UNINIT.

LIN108: If development error detection for the LIN module is enabled: the function Lin_WakeUpValidation shall raise the development error LIN_E_CHANNEL_UNINIT if no LIN Channel of the LIN driver has been initialized.

LIN109: If development error detection for the LIN module is enabled: the function Lin_WakeupValidation shall raise the development error LIN_E_STATE_TRANSITION if no LIN channel of the driver is in the `LIN_CH_SLEEP` state.

8.3.1.3 Lin_GetVersionInfo

LIN161:

Service name:	Lin_GetVersionInfo

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

Service ID[hex]:	0x01
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	versioninfo Pointer to where is stored the version information of this module.
Return value:	None
Description:	Returns the version information of this module.

LIN001: The function Lin_GetVersionInfo shall return the version information of the LIN module. The version information includes:

- Module Id
- Vendor Id
- Vendor specific version numbers (BSW00407).

LIN110: If source code for caller and callee of Lin_GetVersionInfo is available, the LIN module should realize Lin_GetVersionInfo as a macro, defined in the module's header file.

LIN111: The function Lin_GetVersionInfo shall be pre compile time configurable On/Off by the configuration parameter: LinVersionInfoApi.

8.3.2 Services affecting a single LIN channel

8.3.2.1 Lin_InitChannel

LIN007:

Service name:	Lin_InitChannel	

Syntax:	`void Lin_InitChannel(`
	`  uint8 Channel,`
	`  const Lin_ChannelConfigType* Config`
	`)`	

Service ID[hex]:	0x02	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	Channel	LIN channel to be initialized
	Config	Pointer to LIN channel configuration set
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	(Re-)initializes a LIN channel.	

LIN112: The function Lin_InitChannel shall (re-)initialize a LIN channel. Different sets of static configuration may have been configured. The parameter Config is a pointer to the configuration set of a LIN channel.

LIN113: The function Lin_InitChannel shall initialize only LIN channel specific settings. Hardware register settings that have impact on all LIN channels inside the HW unit shall not be changed.

LIN151: The Lin module's environment shall call the function Lin_InitChannel before calling any other LIN channel related function (e.g. Lin_SendHeader, Lin_SendResponse).

LIN185: During LIN channel initialization it shall be validated if there was a wake-up event on the specific LIN channel (e.g. like RXD pin low). If a wake-up event has been detected, the wake-up shall directly be reported to the EcuM via EcuM_SetWakeupEvent call-back function.

Symbolic names of the available configuration sets are provided by the configuration description of the LIN driver. See chapter 10 about configuration description.

LIN100: If development error detection for the Lin module is enabled: the function Lin_InitChannel shall check the parameter Config for being within the allowed range.

If Config is not in the allowed range, the function Lin_InitChannel shall raise the development error LIN_E_INVALID_POINTER.

LIN114: If development error detection for the LIN module is enabled: if the function Lin_InitChannel is called before the LIN module was initialized, the function Lin_InitChannel shall raise the development error LIN_E_UNINIT.

LIN115: If development error detection for the LIN module is enabled: the function Lin_InitChannel shall raise the development error LIN_E_INVALID_CHANNEL if the channel parameter is invalid.

8.3.2.2 Lin_DeInitChannel

LIN009:

Service name:	Lin_DeInitChannel	

Syntax:	`void Lin_DeInitChannel(`
	`  uint8 Channel`
	`)`	

Service ID[hex]:	0x03	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	Channel	LIN channel to be de-initialized
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	De-Inits a LIN channel.	

LIN086: The function Lin_DeInitChannel shall reset all LIN module global variables and all SFRs that are used by the LIN channel to their default reset value.

LIN152: The function Lin_DeInitChannel shall not change hardware register settings that have impact on other LIN channels.

LIN178: The function Lin_DeInitChannel shall only be executable when the LIN channel state-machine is in state `LIN_CH_OPERATIONAL`.

LIN116: If development error detection for the LIN module is enabled: the function Lin_DeInitChannel shall raise the development error LIN_E_INVALID_CHANNEL if the channel parameter is invalid.

8.3.2.3 Lin_SendHeader

LIN164:

Service name:	Lin_SendHeader

Syntax:	`Std_ReturnType Lin_SendHeader(`
	`  uint8 Channel,`
	`  Lin_PduType* PduInfoPtr`
	`)`

Service ID[hex]:	0x04
Sync/Async:	Synchronous

Reentrancy:	Non Reentrant	
Parameters (in):	Channel	LIN channel to be addressed
	PduInfoPtr	Pointer to PDU containing the PID, Checksum model, Response type, Dl 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.	

LIN087: The function Lin_SendHeader shall send the header part (Break Field, Synch Byte Field and PID Field) of a LIN frame on the addressed LIN channel.

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_SendHeader().

LIN122: The Lin module's environment shall only call Lin_SendHeader on a Channel which is in state `LIN_CH_OPERATIONAL`.

LIN117: If development error detection for the LIN module is enabled: if the function Lin_SendHeader is called before the LIN module was initialized, the function Lin_SendHeader shall raise the development error LIN_E_UNINIT and return with E_NOT_OK.

LIN118: If development error detection for the LIN module is enabled: if the channel Channel is not initialized, the function Lin_SendHeader shall raise the development error LIN_E_CHANNEL_UNINIT and return with E_NOT_OK.

LIN119: If development error detection for the LIN module is enabled: if the channel parameter is invalid, the function Lin_SendHeader shall raise the development error LIN_E_INVALID_CHANNEL and return with E_NOT_OK.

LIN120: If development error detection for the LIN module is enabled: the function Lin_SendHeader shall check the parameter PduInfoPtr for not being a NULL pointer. If PduInfoPtr is a NULL pointer, the function Lin_SendHeader shall raise the development error LIN_E_INVALID_POINTER and return with E_NOT_OK.

LIN121: 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_SendHeader shall raise the development error LIN_E_STATE_TRANSITION and return with E_NOT_OK.

8.3.2.4 Lin_SendResponse

LIN165:

Service name:	Lin_SendResponse
Syntax:	`Std_ReturnType Lin_SendResponse(` `  uint8 Channel,`

	Lin_PduType* PduInfoPtr)	
Service ID[hex]:	0x05	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	Channel	LIN channel to be addressed
	PduInfoPtr	Pointer to PDU containing the PID, Checksum model, Response type, Dl 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 response.	

LIN088: The function Lin_SendResponse shall send a complete LIN response part of a LIN frame on the addressed LIN channel.

LIN128: The function Lin_ SendResponse shall only be executable when the LIN channel state-machine is in state `LIN_CH_OPERATIONAL`.

LIN153: The function Lin_ SendResponse shall only be executable when the prior LIN channel function call for the addressed LIN channel was the Lin_SendHeader function.

LIN123: If development error detection for the LIN module is enabled: if the function Lin_SendResponse is called before the LIN module was initialized, the function Lin_SendResponse shall raise the development error LIN_E_UNINIT and return E_NOT_OK.

LIN124: If development error detection for the LIN module is enabled: if the channel Channel is not initialized, the function Lin_SendResponse shall raise the development error LIN_E_CHANNEL_UNINIT and return E_NOT_OK.

LIN125: If development error detection for the LIN module is enabled: if the channel parameter is invalid, the function Lin_SendResponse shall raise the development error LIN_E_INVALID_CHANNEL and return E_NOT_OK.

LIN126: If development error detection for the LIN module is enabled: the function Lin_SendResponse shall check the parameter PduInfoPtr for not being a NULL pointer. If PduInfoPtr is a NULL pointer, the function Lin_SendResponse shall raise the development error LIN_E_INVALID_POINTER and return E_NOT_OK.

LIN127: 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_SendResponse shall raise the development error LIN_E_STATE_TRANSITION and return E_NOT_OK.

8.3.2.5 Lin_GoToSleep

LIN166:

Service name:	Lin_GoToSleep	

Syntax:	`Std_ReturnType Lin_GoToSleep(`
	`  uint8 Channel`
	`)`	

Service ID[hex]:	0x06	
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: 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.	

LIN089: The function Lin_GoToSleep shall send a go-to-sleep-command on the addressed LIN channel.

LIN073: The function Lin_GoToSleep shall set the channel state to `LIN_CH_SLEEP`, enable the wake-up detection and 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.

LIN034: The LIN channel shall enter `LIN_CH_SLEEP` state upon completion of the go-to-sleep-command, even in case of an erroneous transmission.

LIN074: The function Lin_GoToSleep shall terminate ongoing frame transmission of prior transmission requests, even if the transmission is unsuccessfully completed.

LIN129: 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.

LIN130: If development error detection for the LIN module is enabled: the function Lin_GoToSleep shall raise the development error LIN_E_CHANNEL_UNINIT if the channel Channel is not initialized.

LIN131: 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.

8.3.2.6 Lin_GoToSleepInternal

LIN167:

Service name:	Lin_GoToSleepInternal

Syntax:	`Std_ReturnType Lin_GoToSleepInternal(`
	`  uint8 Channel`
	`)`

Service ID[hex]:	0x09	
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: 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.	

LIN095: The function Lin_GoToSleepInternal shall set the channel state to LIN_CH_SLEEP, enable the wake-up detection and optionally set the LIN hardware unit to reduced power operation mode (if supported by HW).

LIN133: 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.

LIN134: If development error detection for the LIN module is enabled: the function Lin_GoToSleepInternal shall raise the development error LIN_E_CHANNEL_UNINIT if the channel Channel is not initialized.

LIN135: 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.

8.3.2.7 Lin_WakeUp

LIN169:

Service name:	Lin_WakeUp	

Syntax:	`Std_ReturnType Lin_WakeUp(`
	`  uint8 Channel`
	`)`	

Service ID[hex]:	0x07	
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: 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.	

LIN090: The function Lin_WakeUp shall generate a wake up pulse on the addressed LIN channel.

LIN154: The Lin driver's environment shall only call Lin_Wakeup when the LIN channel is in state `LIN_CH_SLEEP`.

LIN137: 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.

LIN138: If development error detection for the LIN module is enabled: the function Lin_WakeUp shall raise the development error LIN_E_CHANNEL_UNINIT if the channel Channel is not initialized.

LIN139: 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.

LIN140: 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`.

8.3.2.8 Lin_GetStatus

LIN168:

Service name:	Lin_GetStatus	

Syntax:	`Lin_StatusType Lin_GetStatus(`
	`  uint8 Channel,`
	`  uint8** Lin_SduPtr`
	`)`	

Service ID[hex]:	0x08	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
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_CH_UNINIT: LIN channel not initialized LIN_CH_OPERATIONAL: 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: Sleep mode operation; in this mode wake-up detection from slave nodes is enabled.
Description:	Gets the status of the LIN driver.	

LIN091: The function Lin_GetStatus shall return the current transmission, reception or operation status of the LIN driver.

LIN092: 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_SendHeader function call.

LIN141: 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 and return LIN_NOT_OK.

LIN142: If development error detection for the LIN module is enabled: if the channel Channel is not initialized, the function Lin_GetStatus shall raise the development error LIN_E_CHANNEL_UNINIT and return LIN_NOT_OK.

LIN143: 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 and return LIN_NOT_OK.

LIN144: 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_INVALID_POINTER and return LIN_NOT_OK.

8.4 Call-back notifications

There are no callback functions within the LIN driver.
The callback notifications are implemented in the LIN interface

8.5 Scheduled functions

There are no scheduled functions within the LIN driver

8.6 Expected Interfaces

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

8.6.1 Mandatory Interfaces

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

API function	*Description*
Dem_ReportErrorStatus	Reports errors to the DEM.
EcuM_SetWakeupEvent	Sets the wakeup event.

8.6.2 Optional Interfaces

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

API function	*Description*
Det_ReportError	Service to report development errors.
EcuM_CheckWakeup	This callout is called by the EcuM to poll a wakeup source. It shall 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	This function disables the notification of a channel.
Icu_EnableNotification	This function enables the notification on the given channel.

LIN176: The Lin module shall invoke the callback function EcuM_CheckWakeup from within the wake-up ISR of the corresponding LIN channel OR from within the function Lin_WakeUpValidation 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.

8.6.3 Configurable interfaces

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

All callback functions that are called by the LIN driver are implemented in the LIN Interface. These callback functions are not configurable.

9 Sequence diagrams

Complete sequence diagrams for transmission, reception and error handling can be found in the LIN Interface Specification [8].

9.1 Receiving a LIN Frame

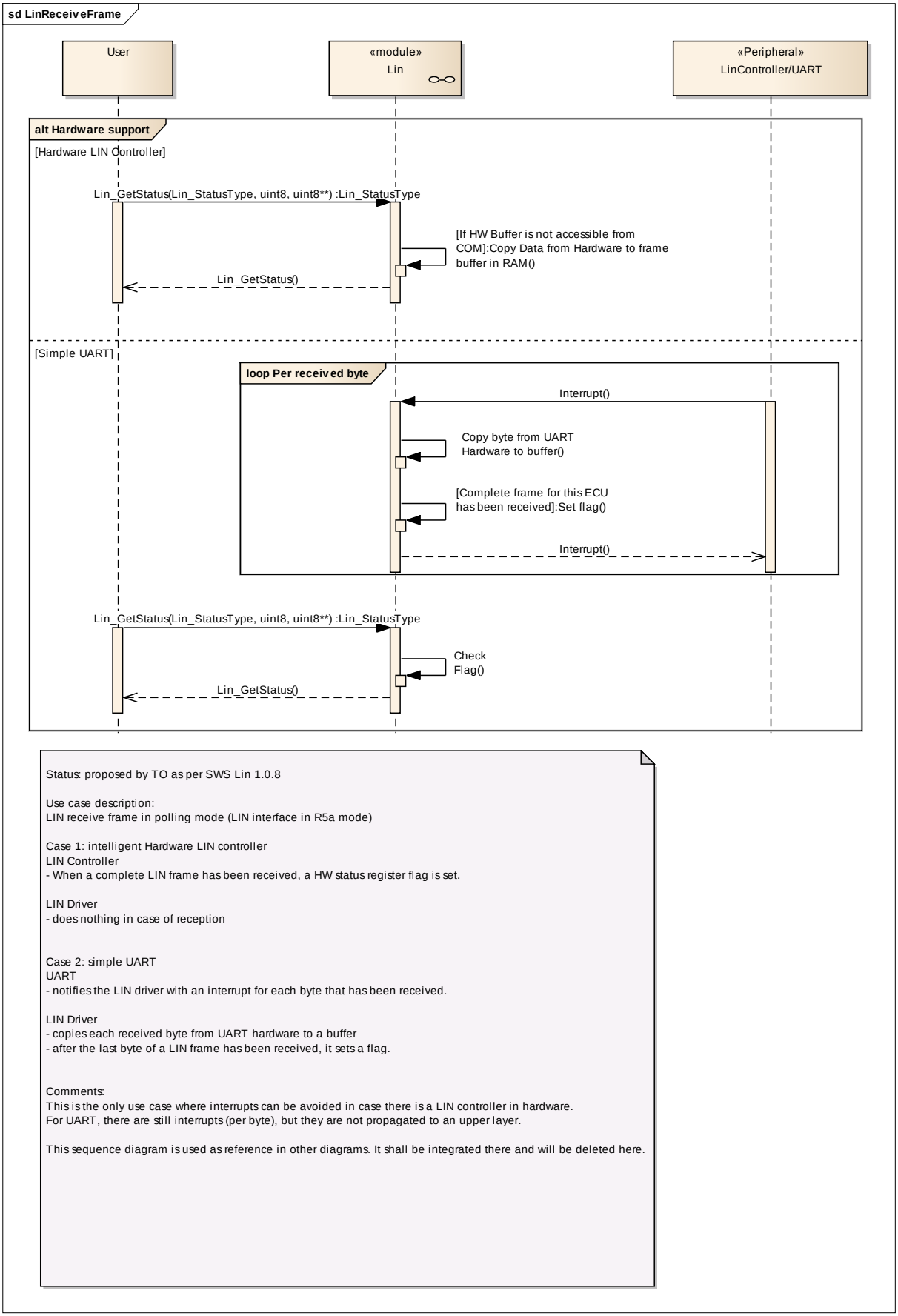

Figure 9-1: LIN Frame Receiving Sequence Chart

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

In addition to this section, it is highly recommended to read the documents:

- AUTOSAR Layered Software Architecture [2]
- AUTOSAR ECU Configuration Specification [9]
  This document describes the AUTOSAR configuration methodology and the AUTOSAR configuration metamodel in detail.

The following is only a short survey of the topic and it will not replace the ECU Configuration Specification document.

10.1.1 Configuration and configuration parameters

Configuration parameters define the variability of the generic part(s) of an implementation of a module. This means that only generic or configurable module implementation can be adapted to the environment (software/hardware) in use during system and/or ECU configuration.

The configuration of parameters can be achieved at different times during the software process: before compile time, before link time or after build time. In the following, the term “configuration class” (of a parameter) shall be used in order to refer to a specific configuration point in time.

In the following tables the configuration class per configuration parameter is specified. In fact, it is important to distinguish between the configuration-classes, because they will result in different implementations and design processes.

Pre-compile time - specifies whether the configuration parameter shall be of configuration class *Pre-compile time* or not

Label	***Description***
X	The configuration parameter shall be of configuration class *Pre-compile time*.
--	The configuration parameter shall never be of configuration class *Pre-compile time*.

Link time - specifies whether the configuration parameter shall be of configuration class *Link time* or not

Label	Description
X	The configuration parameter shall be of configuration class *Link time*.
--	The configuration parameter shall never be of configuration class *Link time*.

Post Build - specifies whether the configuration parameter shall be of configuration class *Post Build* or not

Label	Description
X	The configuration parameter shall be of configuration class *Post Build* and no specific implementation is required.
L	*Loadable* - the configuration parameter shall be of configuration class *Post Build* and only one configuration parameter set resides in the ECU.
M	*Multiple* - the configuration parameter shall be of configuration class *Post Build* and is selected out of a set of multiple parameters by passing a dedicated pointer to the init function of the module.
--	The configuration parameter shall never be of configuration class *Post Build*.

10.1.2 Variants

Variants describe sets of configuration parameters. E.g., Variant 1: only pre-compile time configuration parameters; Variant 2: mix of pre-compile- and post build time-configuration parameters. In one variant a parameter can only be of one configuration class.

10.1.3 Containers

Containers structure the set of configuration parameters. This means:
- *all* configuration parameters are kept in containers.
- (sub-) containers can reference (sub-) containers. It is possible to assign a multiplicity to these references. The multiplicity then defines the possible number of instances of the contained parameters.

Configuration parameters will be clustered into one container whenever
- the configuration parameters logically belong together (e.g., general parameters which are valid for the entire module)
- the configuration parameters need to be instantiated (e.g., parameters of a LIN cluster – those parameters must be instantiated for each LIN channel separately)

10.2 Containers and configuration parameters

The following chapters summarize all configuration parameters.

The described parameters are input for the LIN driver configurator.

LIN029: The code configurator of the LIN driver is LIN hardware Unit specific.

LIN031: The configuration data shall have a symbolic format that is human readable and understandable.

LIN039: Values that can be configured are hardware dependent. Therefore, the rules and constraints cannot be given in the standard. The configuration tool is responsible to do a static configuration checking, also regarding dependencies between modules (e.g. Port driver, MCU driver etc.)

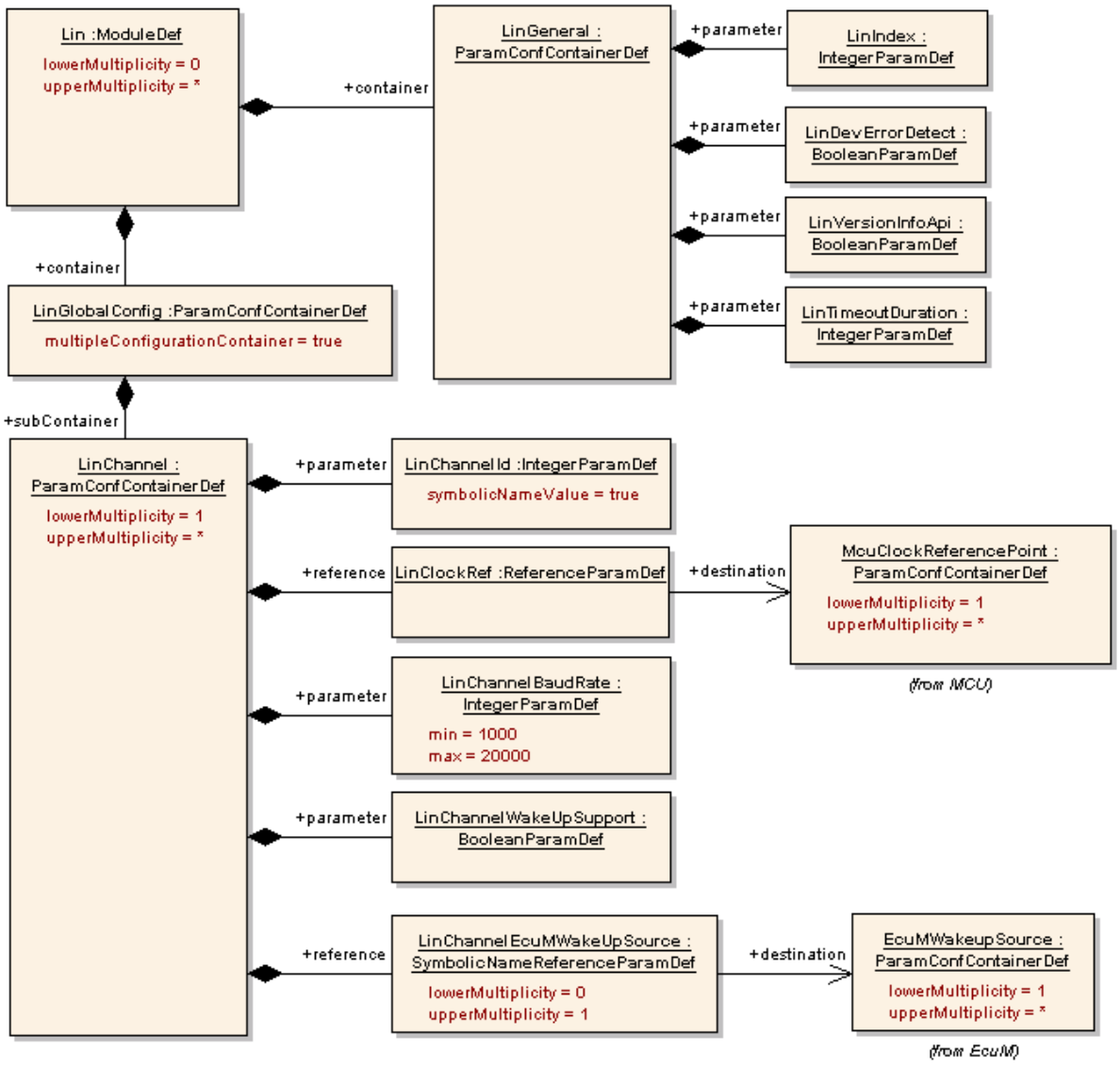

Figure 10-1: Configuration structure for the LIN driver

10.2.1 Variants

Two configuration variants are defined for the LIN driver.

LIN103: Variant 1: Pre-compile Configuration

In the pre-compile configuration all parameters below that are marked as Pre-compile configurable shall be configurable in a pre-compile manner, for example as #defines. The module is most likely delivered as source code.

LIN104: Variant 2: Mix of pre-compile and post build time-configuration for multiple selectable configuration sets

This configuration includes all configuration options of the “Pre-compile Configuration”. Additionally all parameters defined below as post build configurable shall be configurable post build for example by flashing configuration data. The module is most likely delivered as object code.

10.2.2 Lin

Module Name	*Lin*
Module Description	Configuration of the Lin (LIN driver) module.

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
LinGeneral	1	--
LinGlobalConfig	1	This container contains the global configuration parameter of the Lin driver. This container is a MultipleConfigurationContainer, i.e. this container and its sub-containers exit once per configuration set.

10.2.3 LinGeneral

SWS Item	**LIN177 :**
Container Name	LinGeneral
Description	--
Configuration Parameters	

SWS Item	**LIN066 :**		
Name	LinDevErrorDetect {LIN_DEV_ERROR_DETECT}		
Description	Switches the Development Error Detection and Notification ON or OFF.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**:**		
Name	LinIndex {LIN179}		
Description	Specifies the InstanceId of this module instance. If only one instance is present it shall have the Id 0.		
Multiplicity	1		
Type	IntegerParamDef		
Range	..		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**LIN093 :**

Name	LinTimeoutDuration {LIN_TIMEOUT_DURATION}		
Description	Specifies the maximum number of loops for blocking function until a timeout is raised in short term wait loops		
Multiplicity	1		
Type	IntegerParamDef		
Range	..		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**LIN067 :**		
Name	LinVersionInfoApi {LIN_VERSION_INFO_API}		
Description	Switches the Lin_GetVersionInfo function ON or OFF.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

No Included Containers

10.2.4 LinChannel

SWS Item	**LIN069 :**
Container Name	LinChannel
Description	This container contains the configuration (parameters) of the LIN Controller(s).
Configuration Parameters	

SWS Item	**LIN180 :**		
Name	LinChannelBaudRate {LIN_CHANNEL_BAUD_RATE}		
Description	Specifies the baud rate of the LIN channel		
Multiplicity	1		
Type	IntegerParamDef		
Range	1000 .. 20000		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	M	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	**LIN181 :**

Name	LinChannelId		
Description	Identifies the LIN channel. Replaces LIN_CHANNEL_INDEX_NAME from the LIN SWS.		
Multiplicity	1		
Type	IntegerParamDef (Symbolic Name generated for this parameter)		
Range	..		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**LIN182 :**		
Name	LinChannelWakeUpSupport {LIN_CHANNEL_WAKE_UP_SUPPORT}		
Description	Specifies if the LIN hardware channel supports wake up functionality		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**:**		
Name	LinChannelEcuMWakeUpSource		
Description	This parameter contains a reference to the Wakeup Source for this controller as defined in the ECU State Manager. Implementation Type: reference to EcuM_WakeupSourceType		
Multiplicity	0..1		
Type	Reference to [EcuMWakeupSource]		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**LIN094 :**		
Name	LinClockRef {LIN_CLOCK_SRC_REFERENCE}		
Description	Reference to the LIN clock source configuration, which is set in the MCU driver configuration.		
Multiplicity	1		
Type	Reference to [McuClockReferencePoint]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	

	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

No Included Containers

10.2.5 LinGlobalConfig

SWS Item	**LIN178 :**
Container Name	LinGlobalConfig [Multi Config Container]
Description	This container contains the global configuration parameter of the Lin driver. This container is a MultipleConfigurationContainer, i.e. this container and its sub-containers exit once per configuration set.
Configuration Parameters	

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

10.3 Published Information

Published information contains data defined by the implementer of the SW module that does not change when the module is adapted (i.e. configured) to the actual HW/SW environment. It thus contains version and manufacturer information.

The standard common published information like

vendorId (<Module>_VENDOR_ID),
moduleId (<Module>_MODULE_ID),
arMajorVersion (<Module>_AR_MAJOR_VERSION),
arMinorVersion (<Module>_ AR_MINOR_VERSION),
arPatchVersion (<Module>_ AR_PATCH_VERSION),
swMajorVersion (<Module>_SW_MAJOR_VERSION),
swMinorVersion (<Module>_ SW_MINOR_VERSION),
swPatchVersion (<Module>_ SW_PATCH_VERSION),
vendorApiInfix (<Module>_VENDOR_API_INFIX)

is provided in the BSW Module Description Template (see [14] Figure 4.1 and Figure 7.1).

Additional published parameters are listed below if applicable for this module.