

Document Title	Specification of LIN Interface
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	073
Document Classification	Standard

Document Version	2.4.0
Document Status	Final
Part of Release	3.2
Revision	3

Document Change History			
Date	Version	Changed by	Change Description
28.02.2014	2.4.0	AUTOSAR Release Management	<ul style="list-style-type: none"> • Added CDD callback functions • Removed wakeup transition due to LinIf_Cbk_CheckWakeup • Editorial changes • Removed chapter(s) on change documentation
17.05.2012	2.3.0	AUTOSAR Administration	<ul style="list-style-type: none"> • Added the requirement of checking the parameter error • Added the requirement of calling the function Lin_GoToSleepInternal • Clarified the buffer handling requirement for LinTp • Changed the handling of unexpected PDU for LinTp (stop reception -> ignore message)
07.04.2011	2.2.0	AUTOSAR Administration	<ul style="list-style-type: none"> • Added the schedule table change due to BswM • Added the P2 Timeout observation for LIN TP • Deleted the handling of Response_Error • Clarified the LIN TP reception error and the development error code • Legal disclaimer revised
20.09.2010	2.1.0	AUTOSAR Administration	<ul style="list-style-type: none"> • Updated LINIF226 • Use PduInfoType for RxIndication, TriggerTransmit and Transmit APIs • Clarification of time parameters specified as float • Legal disclaimer revised

Document Change History			
Date	Version	Changed by	Change Description
23.06.2008	2.0.1	AUTOSAR Administration	Legal disclaimer revised
15.11.2007	2.0.0	AUTOSAR Administration	<ul style="list-style-type: none"> • Interaction with LIN State Manager added • LIN Interface configuration reworked • Detection of LIN Response Error added • Wake-up concept reworked • Document meta information extended • Small layout adaptations made
31.01.2007	1.1.0	AUTOSAR Administration	<ul style="list-style-type: none"> • Start-up and Wake-up reworked for Transceiver needs. • File structure and requirements traceability adapted to new template. • Reworked configuration after integrator input. • Removed API's: LinIf_InitChannel() LinIf_DeInitChannel() • 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	8
1.1	Architectural overview	8
1.2	Functional overview	9
2	Acronyms and abbreviations	10
3	Related documentation.....	11
3.1	Input documents.....	11
3.2	Related standards and norms	12
4	Constraints and assumptions	13
4.1	Limitations	13
4.2	Applicability to car domains	13
5	Dependencies to other modules.....	14
5.1	Upper layers	14
5.1.1	PDU Router.....	14
5.1.2	BSW Scheduler.....	14
5.1.3	Operating System	14
5.1.4	Module DET (Development Error Tracer)	15
5.1.5	Module DEM (Diagnostic Event Manager)	15
5.1.6	Module ECU State Manager	15
5.1.7	Module LIN State Manager	15
5.1.8	Module BSW Mode Manager.....	15
5.2	Lower layers	15
5.2.1	LIN Driver.....	15
5.3	File structure.....	16
5.3.1	Code file structure.....	16
5.3.2	Header file structure.....	16
6	Requirements traceability	21
7	Functional specification	26
7.1	Frame Transfer.....	26
7.1.1	Frame types.....	26
7.1.1.1	Unconditional frame	27
7.1.1.2	Event-triggered frame.....	27
7.1.1.3	Sporadic frame	27
7.1.1.4	Diagnostic Frames MRF and SRF	28
7.1.1.5	User-defined frames.....	28
7.1.1.6	Reserved frames	29
7.1.2	Frame reception.....	29
7.1.2.1	Header.....	29
7.1.2.2	Response	29
7.1.2.3	Status check.....	29
7.1.3	Frame transmission	31

7.1.3.1	Header.....	31
7.1.3.2	Response	31
7.1.3.3	Status check.....	32
7.1.4	Slave-to-slave communication	32
7.1.4.1	Header.....	32
7.1.4.2	Response	32
7.1.4.3	Status check.....	32
7.2	Schedules.....	33
7.2.1	LinIf_MainFunction.....	33
7.2.2	Schedule table manager	34
7.3	Network management	36
7.3.1	Node Management	37
7.3.1.1	LIN Interface state-machine	37
7.3.1.2	LIN channel sub-state-machine.....	37
7.3.2	Go to sleep process	39
7.3.3	Wake up process	40
7.3.3.1	Wakeup during sleep transition	40
7.4	Status Management	41
7.5	Diagnostics and Node configuration	41
7.5.1	Node configuration.....	42
7.5.1.1	Node Model	42
7.5.1.2	Node Configuration services	42
7.5.1.3	Node Configuration API.....	43
7.5.1.4	Node Configuration in Schedule Table	43
7.5.2	Diagnostics – Transport Protocol.....	44
7.5.2.1	LIN TP initialization.....	45
7.5.2.2	State-machine	46
7.5.2.3	LIN TP transmission	47
7.5.2.4	LIN TP transmission error.....	49
7.5.2.5	LIN TP reception	50
7.5.2.6	LIN TP reception error.....	50
7.5.2.7	Unavailability of receive buffer.....	53
7.6	Handling multiple channels and drivers.....	53
7.6.1	Multiple channels	53
7.6.2	Multiple LIN drivers	53
7.7	Error classification	54
7.8	Error detection	54
7.9	Error notification	55
8	API specification.....	56
8.1	Imported types.....	56
8.1.1	Standard types.....	56
8.1.2	Type definitions.....	56
8.1.2.1	LinIf_SchHandleType	56
8.1.2.2	LinIf_ConfigType	56
8.1.2.3	LinTp_ConfigType	57
8.1.2.4	LinTp_Mode	57
8.2	LIN Interface API	57

8.2.1	LinIf_Init	57
8.2.2	LinIf_GetVersionInfo	58
8.2.3	LinIf_Transmit	59
8.2.4	LinIf_ScheduleRequest	60
8.2.5	LinIf_GotoSleep	61
8.2.6	LinIf_WakeUp	61
8.2.7	LinTp_Init	62
8.2.8	LinTp_Transmit	63
8.2.9	LinTp_GetVersionInfo	64
8.2.10	LinTp_Shutdown	64
8.2.11	LinTp_CancelTransmit	65
8.2.12	LinTp_ChangeParameter	65
8.3	Call-back notifications	66
8.3.1	LinIf_Cbk_CheckWakeUp	66
8.4	Scheduled functions	67
8.4.1	LinIf_MainFunction	67
8.5	Expected Interfaces	67
8.5.1	Mandatory Interfaces	67
8.5.2	Optional interfaces	68
8.5.3	Configurable interfaces	68
8.5.3.1	<User>_ScheduleRequest_Confirmation	68
8.5.3.2	<User>_GotoSleep_Confirmation	69
8.5.3.3	<User>_WakeUp_Confirmation	69
8.5.3.4	<User_TriggerTransmit>	70
8.5.3.5	<User_TxConfirmation>	70
8.5.3.6	<User_RxIndication>	71
9	Sequence diagrams	72
9.1	Frame Transmission	72
9.2	Frame Reception	74
9.3	Slave to slave communication	75
9.4	Reception and transmission at the same time	76
9.5	Sporadic frame	77
9.6	Event-triggered frame	78
9.6.1	With no answer	79
9.6.2	With answer (No collision)	80
9.6.3	With collision	81
9.7	Transport Protocol message transmission	82
9.8	Transport Protocol message reception	83
9.9	Go to sleep process	84
9.10	Wake up request	87
9.11	Internal wake-up	87
10	Configuration specification	88
10.1	How to read this chapter	88
10.1.1	Configuration and configuration parameters	88
10.1.2	Containers	88
10.1.3	Specification template for configuration parameters	88
10.2	Containers and configuration parameters	89

10.2.1	Configuration Tool	89
10.2.2	Variants	90
10.2.2.1	Variant1 (Pre-compile Configuration).....	90
10.2.2.2	Variant2 (Link-time Configuration)	90
10.2.2.3	Variant3 (Post-build Configuration).....	90
10.3	LinIf_Configuration	90
10.3.1	LinIf	91
10.3.2	LinIfGeneral	92
10.3.3	LinIfGlobalConfig	93
10.3.4	LinIfChannel	94
10.3.5	LinIfFrame	96
10.3.6	LinIfFixedFrameSdu	98
10.3.7	LinIfFixedFrameSduByte	98
10.3.8	LinIfPduDirection	99
10.3.9	LinIfSubstitutionFrames	99
10.3.10	LinIfRxPdu	100
10.3.11	LinIfTxPdu	101
10.3.12	LinIfScheduleTable	103
10.3.13	LinIfEntry	104
10.3.14	LinIfMaster	105
10.3.15	LinIfSlave	106
10.3.16	LinIfNodeComposition	107
10.3.17	LinIfSlaveToSlavePdu	107
10.3.18	LinIfInternalPdu	108
10.4	LIN Transport Layer configuration	108
10.4.1	LinTp	109
10.4.2	LinTpGeneral	110
10.4.3	LinTpGlobalConfig	110
10.4.4	LinTpChannelConfig	112
10.4.5	LinTpRxNSdu	113
10.4.6	LinTpTxNSdu	114
10.5	Published Information	115

1 Introduction and functional overview

This specification specifies the functionality, API and the configuration of the AUTOSAR Basic Software module LIN Interface (LinIf) and the LIN Transport Protocol (LinTp). The LIN TP is a part of the LIN Interface.

The wake-up functionality is covered within the LIN Interface and the LIN Driver.

The base for this document is the LIN 2.0 specification [17]. 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 Interface module 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 document 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 Interface).

The LIN Interface is designed to be hardware independent. The interfaces to above (PDU Router) and below module (LIN Driver) are well defined.

The LIN Interface may handle more than one LIN Driver. A LIN Driver can support more than one channel. This means that the LIN Driver can handle one or more LIN channels.

1.1 Architectural overview

According to the Layered Software Architecture [2], the LIN Interface is located within the BSW architecture as shown below. In this example, the LIN Interface is connected to two LIN Drivers. However, one LIN Driver is the most common configuration.

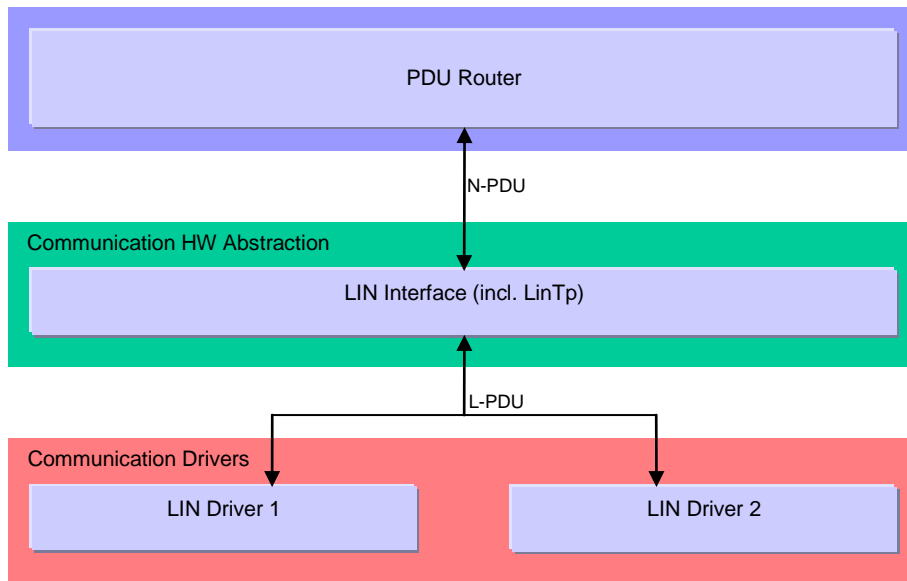


Figure 1 – AUTOSAR BSW software architecture (LIN relevant modules)

1.2 Functional overview

The LIN Interface is responsible for providing LIN 2.0 master functionality towards the upper layers. This means:

- Executing the currently selected schedule for each LIN bus the ECU is connected to (transmitting headers and transmitting/receiving responses).
- Switching schedule tables when requested by the upper layer(s).
- Accepting frame transmit requests from the upper layers and transmit the data part as response within the appropriate LIN frame.
- Providing frame receive notification for the upper layer when the corresponding response is received within the appropriate frame.
- Go to sleep and wake-up services.
- Error handling.
- Diagnostic Transport Layer services.

2 Acronyms and abbreviations

In addition to the acronyms and abbreviations found in the LIN 2.0 specification, the following acronyms and abbreviations are used throughout this document. Some terms already defined in the LIN 2.0 specification have also been defined here in order to provide more clarification, especially for terms used very often in this document.

Abbreviation / Acronym:	Description:
API	Application Program Interface
CF	Continuous Frame in TP
DCM	Diagnostic Communication Manager
Delay	The time between start of frames in a schedule table. The unit is in number of time-bases for the specific cluster.
DEM	Diagnostic Event Manager
DET	Development Error Tracer
EcuM	ECU State Manager
FF	First Frame in TP
LinIf	LIN Interface (the subject of this document)
LinTp	LIN Transport Protocol
Maximum frame length	The maximum frame length is the $T_{\text{Frame_Maximum}}$ as defined in the LIN 2.0 specification (i.e. The nominal frame length plus 40 %).
MRF	Master Request Frame
NAD	Node Address. Each slave in LIN must have a unique NAD.
NC	Node Configuration
N-SDU	Network Layer – Service Data Unit
PDU	Protocol Data Unit
PDUR	PDU Router module
Schedule entry is due	This means that the LIN Interface has arrived to a new entry in the schedule table and a frame (received or transmitted) will be initiated.
SDU	Service Data Unit
SF	Single Frame in TP
Slave-to-slave	There exist 3 different directions of frames on the LIN bus: Response transmitted by the master, Response received by the slave and Response transmitted by one slave and received by another slave. The slave-to-slave is describing the last one. This is not described explicitly in the LIN 2.0 specification.
Sporadic frame	This is one of the unconditional frames that are attached to a sporadic slot.
Sporadic slot	This is a placeholder for the sporadic frames. The reason to name it slot is that it has no LIN frame ID.
SRF	Slave Response Frame
SWS	Software specification
Tick	Predefined period that the LinIf_MainFunction function shall be called to handle the communication on all channels.
TP	Transport Protocol

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_DET.pdf

- [6] Requirements on LIN
AUTOSAR_SRS_LIN.pdf

- [7] Specification of LIN Driver
AUTOSAR_SWS_LIN_Driver.pdf

- [8] Specification of Diagnostics Event Manager
AUTOSAR_SWS_DEM.pdf

- [9] Specification of ECU Configuration
AUTOSAR_ECU_Configuration.pdf

- [10] Specification of ECU State Manager
AUTOSAR_SWS_ECU_StateManager.pdf

- [11] Specification of BSW Scheduler
AUTOSAR_SWS_BSW_Scheduler.pdf

- [12] Specification of LIN State Manager

AUTOSAR_SWS_LIN_StateManager.pdf

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

[14] Specification of PDU Router
AUTOSAR_SWS_PDU_Router.pdf

[15] Specification of Communication Stack Types
AUTOSAR_SWS_ComStackTypes.pdf

[16] Specification of Basic Software Mode Manager
AUTOSAR_SWS_BSWModeManager.pdf

3.2 Related standards and norms

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

4 Constraints and assumptions

4.1 Limitations

The LIN Interface module can only be used as a LIN master in a LIN cluster. There is only one instance of the LIN Interface in each ECU. If the underlying LIN Driver supports multiple channels, the LIN Interface may be master on more than one cluster.

4.2 Applicability to car domains

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

5 Dependencies to other modules

This section describes the relations to other modules within the basic software. It describes the services that are used from these modules.

To be able for the LIN Interface to operate, the following modules are interfaced:

- LIN Driver – Lin
- PDU Router – PduR
- Development Error Tracer – DET
- Diagnostics Event Manager – DEM
- ECU State Manager – EcuM
- LIN State Manager – LinSM
- BSW Mode Manager – BswM

5.1 Upper layers

5.1.1 PDU Router

The LIN Interface connects to the PDU Router and/or alternative modules above (e.g. Complex Driver) for transmission and reception of frames. It is assumed that these modules are responsible for the copying of the data of the frames for reception and transmission. In case of TP, the PDU Router is the only module above and handles the TP messages buffers either as complete or fragmented messages. The possibility of having a choice of the module above is expressed e.g. in callback functions by prefixing names with “<User_>”.

5.1.2 BSW Scheduler

The LIN Interface needs the cyclic invocation of its main scheduling function with a predefined period (i.e. the tick) and a jitter. For the LIN Interface, the tick is used as the smallest time entity in the scheduling of communication. The LIN Interface does not consider the jitter. It should be part of the consistency check of the configuration (e.g. the delay of each schedule table entry).

5.1.3 Operating System

The LIN Interface does contain access of data shared with neighboring modules. Sharing this data does not rely on OS functionality to protect the data for consistency. However, there may be reentrant functions that access the same data in the LIN Interface. It is up to the LIN Interface’s implementer to solve these accesses.

5.1.4 Module DET (Development Error Tracer)

In development mode, the LIN Interface calls the function `Det_ReportError` of the module DET [5] when it detects a development error.

5.1.5 Module DEM (Diagnostic Event Manager)

The LIN Interface reports production errors to the Diagnostic Event Manager [8].

5.1.6 Module ECU State Manager

The purpose of the ECU state manager with respect to the LIN Interface is as follows:

1. The ECU state manager initializes the LIN Interface.

5.1.7 Module LIN State Manager

The LIN state manager is responsible for the control flow of the whole LIN stack. Therefore, it has the following purposes regarding the LIN Interface:

1. The state manager forwards a schedule table request to the LIN Interface.
2. The state manager requests the transmission of wake-up and sleep command.

5.1.8 Module BSW Mode Manager

LIN TP that is a part of LIN Interface connects to BSW Mode Manager for requesting the schedule table change when upper layer requests the LIN TP operation.

5.2 Lower layers

5.2.1 LIN Driver

The LIN Interface requires the services of the underlying LIN Driver specified by [7].

The LIN Interface assumes the following primitives to be provided by the LIN Driver:

- Transmission of the header part of a frame (`Lin_SendHeader`). It is assumed that this primitive also tells the direction of the frame response (transmit, receive or slave-to-slave communication).
- Transmission of the response part of a frame (`Lin_SendResponse`).
- Transmission of the go-to-sleep command (`Lin_GoToSleep`).
- Setting a LIN channel to state `LIN_CH_SLEEP` without transmitting a go-to-sleep command (`Lin_GoToSleepInternal`).
- Transmission of the wake-up command (`Lin_Wakeup`).

- Query of reception of the response part of a frame (Lin_GetStatus). The following cases are assumed to be distinguished:
 - Successful reception/transmission.
 - No reception.
 - Erroneous reception/transmission (framing error, bit error, checksum error).
 - Ongoing reception – at least one response byte has been received, but the checksum byte has not been received.
 - Ongoing transmission.
 - Channel In sleep (the go-to-sleep command has been successfully transmitted).

LINIF129: The LIN Interface shall not use or access the LIN hardware or assume information about it any way other than what the LIN Driver provides through the function calls to the LIN Driver listed above.

5.3 File structure

5.3.1 Code file structure

This chapter describes the c-files that implement the LIN Interface Configuration.

LINIF241: 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:

- LinIf_Lcfg.c – for link time configurable parameters
- LinIf_PBcfg.c – for post build time configurable parameters
- LinIf_Cfg.c – for pre-compile time configuration parameters

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

5.3.2 Header file structure

This chapter describes the header files that will be included by the LIN Interface and possible other modules.

LINIF242: A header file LinIf.h shall exist that contains all data exported from the LIN Interface – API declarations (except callbacks), extern types, and global data.

LINIF243: A header file LinIf_Cbk.h shall exist that contains function declarations for the callback functions in the LIN Interface.

LINIF244: A header file LinIf_Cfg.h shall exist that contains the pre-compile time parameters.

LINIF245: The header file Linif_Cfg.h shall contain declarations of the link time and the post-build time configurable parameters.

The header file structure shall be used as depicted in

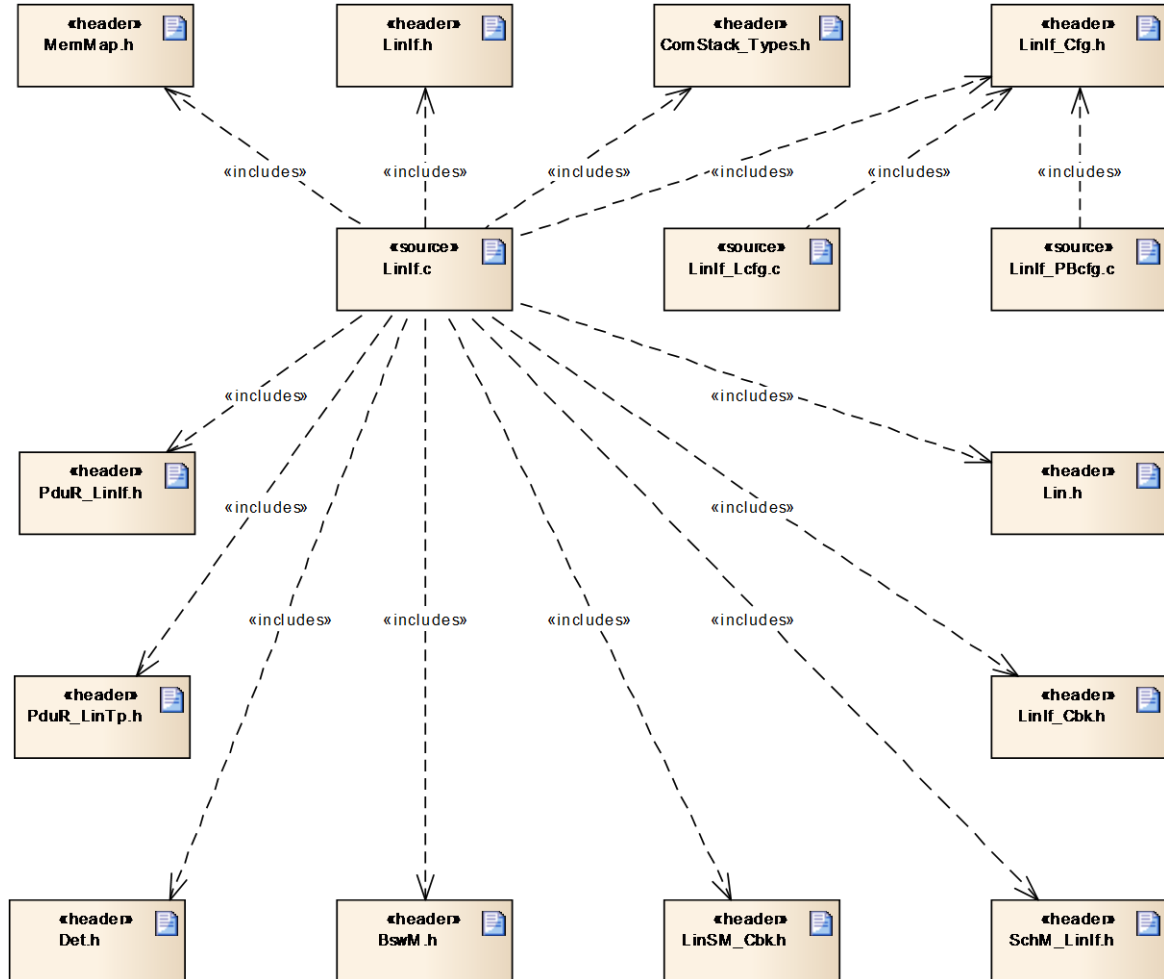


Figure 2 – Header file structure.

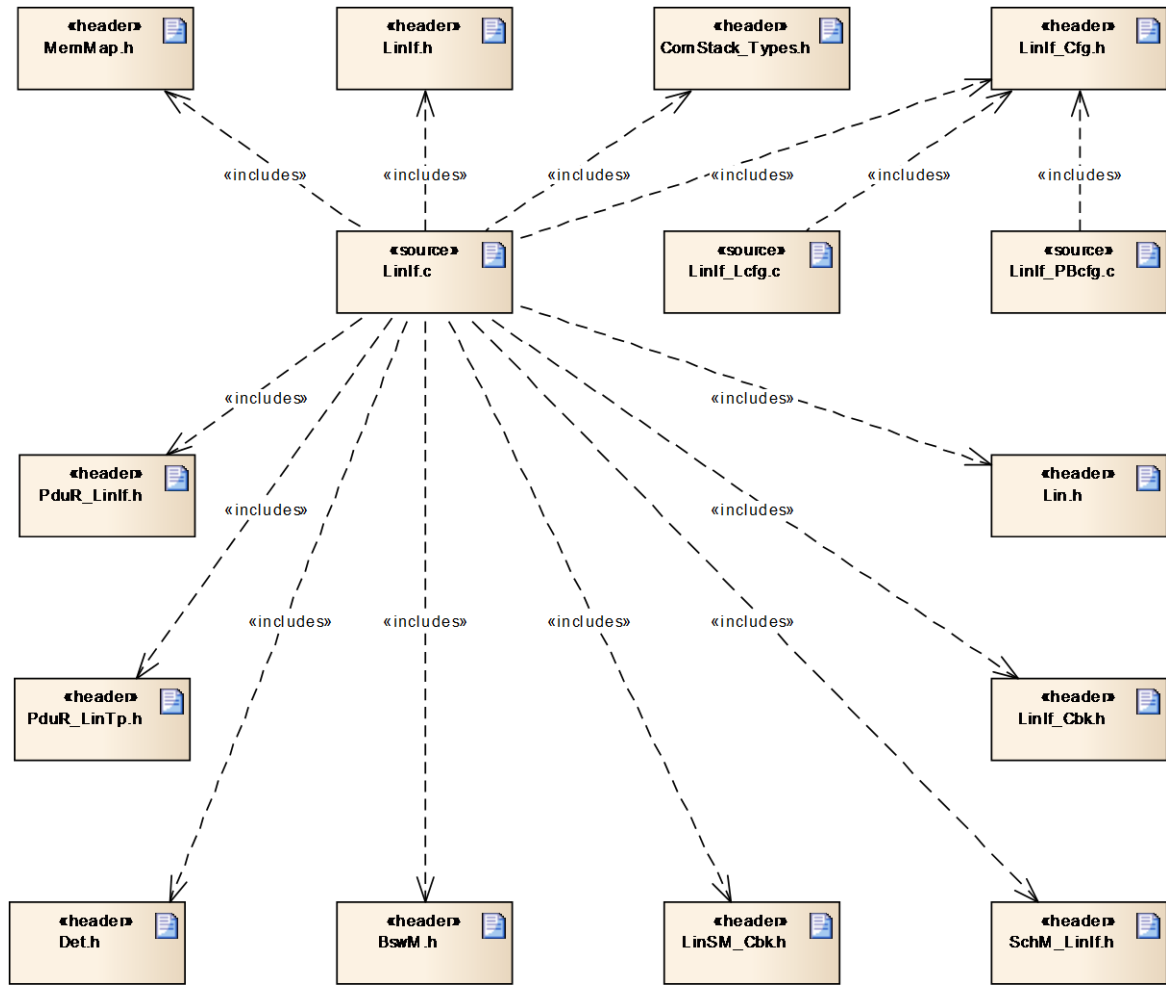


Figure 2 – Header file structure

The LIN Interface implementation object in

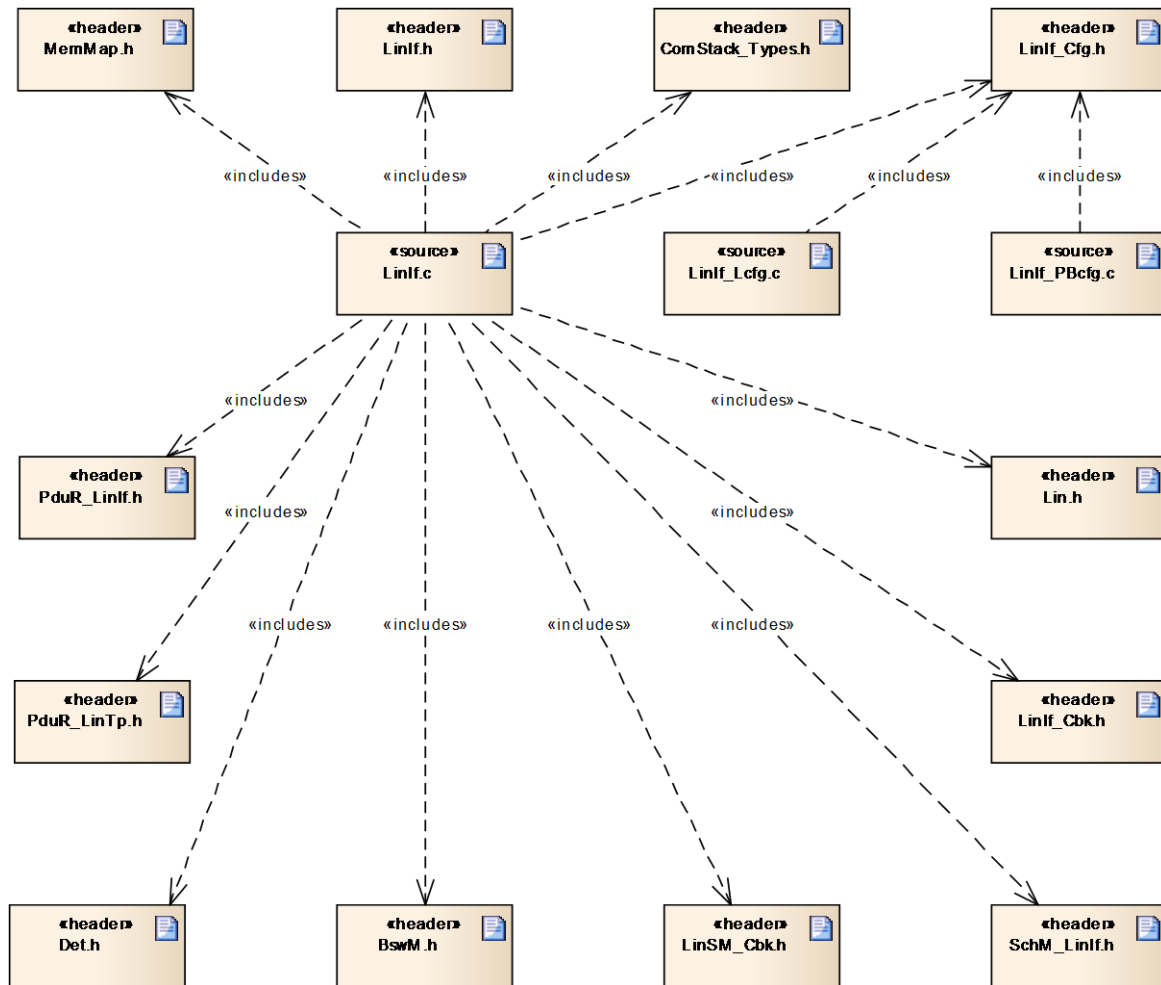


Figure 2 represents one or more c-files. It is not required to make the complete implementation in one file.

LINIF382: The LIN Interface header files (Linif_Cbk.h, Linif.h and Linif_Cfg.h) shall contain the version number: LINIF_SW_MAJOR_VERSION.

LINIF383: The LIN Interface shall check (pre compile-time) that the correct versions of the header files are used.

LINIF247: The LIN Interface shall include the file Dem.h.

LINIF434: The LIN Interface shall include the file Lin.h.

LINIF589: The LIN Interface shall include the file SchM_Linif.h.

LINIF590: The LIN Interface shall include the file MemMap.h.

LINIF497: The LIN Interface shall include the file PduR_Linif.h.

LINIF498: The LIN Interface shall include the file Det.h if the configuration parameter LinIfDevErrorDetect is enabled.

LINIF499: The LIN Interface shall include the file ComStack_Types.h.

LINIF494: The LIN Interface shall include the file PduR_LinTp.h.

LINIF556: The LIN Interface shall include the file LinSM_Cbk.h.

LINIF650: The LIN Interface shall include the file BswM.h.

6 Requirements traceability

This chapter contains a matrix that shows the link between the SWS requirements defined for the LIN Interface and the input requirement documents (SRS).

The following two SRSs acts as input requirements to the LIN Interface:

1. AUTOSAR – General Requirements on Basic Software Modules [3]
2. AUTOSAR –AUTOSAR Requirements on Basic Software Modules, Cluster: LIN [6]

Document: AUTOSAR requirements on Basic Software, general

Requirement	Satisfied by
[BSW00344] Reference to link-time configuration	[LINIF373] [LINIF371]
[BSW00404] Reference to post build time configuration	[LINIF373] [LINIF371]
[BSW00405] Reference to multiple configuration sets	[LINIF373] [LINIF371]
[BSW00345] Pre-compile-time configuration	[LINIF244]
[BSW159] Tool-based configuration	Chapter 10.2
[BSW167] Static configuration checking	[LINIF375]
[BSW171] Configurability of optional functionality	[LINIF310] [LINIF387]
[BSW170] Data for reconfiguration of AUTOSAR SW-Components	[LINIF373]
[BSW00380] Separate C-Files for configuration parameters	[LINIF241]
[BSW00419] Separate C-Files for pre-compile time configuration parameters	[LINIF241]
[BSW00381] Separate configuration header file for pre-compile time parameters	[LINIF244]
[BSW00412] Separate H-File for configuration parameters	[LINIF245]
[BSW00383] List dependencies of configuration files	No configuration from other modules are used
[BSW00384] List dependencies to other modules	Chapter 5
[BSW00387] Specify the configuration class of callback function	Chapter 8.3
[BSW00388] Introduce containers	Chapter 10.2
[BSW00389] Containers shall have names	Chapter 10.2
[BSW00390] Parameter content shall be unique within the module	Chapter 10.2
[BSW00391] Parameter shall have unique names	Chapter 10.2
[BSW00392] Parameters shall have a type	Chapter 10.2
[BSW00393] Parameters shall have a range	Chapter 10.2
[BSW00394] Specify the scope of the parameters	Chapter 10.2
[BSW00395] List the required parameters (per parameter	Chapter 10.2
[BSW00396] Configuration classes	Chapter 10.2
[BSW00397] Pre-compile-time parameters	Chapter 10.2
[BSW00398] Link-time parameters	Chapter 10.2
[BSW00399] Loadable Post-build time parameters	Chapter 10.2
[BSW00400] Selectable Post-build time parameters	Chapter 10.2
[BSW00402] Published information	[LINIF280]
[BSW00375] Notification of wake-up	[LINIF378]
[BSW101] Initialization interface	[LINIF198]
[BSW00416] Sequence of initialization	[LINIF198]
[BSW00406] Check module initialization	[LINIF380] [LINIF376]

[BSW168] Diagnostic Interface of SW components	LIN Interface does not offer a diagnostic interface
[BSW00407] Function to read out published parameters	LIN If [LINIF340] and LIN Tp [LINIF352]
[BSW00423] Usage of SW-C template to describe BSW modules with AUTOSAR Interfaces	LinIf has no AUTOSAR interfaces
[BSW00424] BSW main processing function task allocation	LinIf does not provide any task handling
[BSW00425] Trigger conditions for schedulable objects	Scheduling frames are a property of the LIN 2.0 specification that is inherited in LinIf. [LINIF248]
[BSW00426] Exclusive areas in BSW modules	LinIf does not require any exclusive areas. It may however be used in the implementation.
[BSW00427] ISR description for BSW modules	LinIf has no Interrupt functions defined
[BSW00428] Execution order dependencies of main processing functions	No dependency to other modules regarding the call of the main function
[BSW00429] Restricted BSW OS functionality access	No access of OS operations required
[BSW00431] The BSW Scheduler module implements task bodies	NOT APPLICABLE for LinIf, it is a requirement on BSW schedule
[BSW00432] Modules should have separate main processing functions for read/receive and write/transmit data path	NOT APPLICABLE for LinIf SWS since the scheduling is made in LIN If
[BSW00433] Calling of main processing functions	NOT APPLICABLE for LinIf, it is a requirement on BSW schedule
[BSW00434] The Schedule Module shall provide an API for exclusive areas	NOT APPLICABLE for LinIf, it is a requirement on BSW schedule
[BSW00336] Shutdown interface	[LINIF355]
[BSW00337] Classification of errors	[LINIF376]
[BSW00338] Detection and Reporting of development errors	[LINIF376]
[BSW00369] Do not return development error codes via API	Chapter 8
[BSW00339] Reporting of production relevant error status	[LINIF376]
[BSW00417] Reporting of Error Events by Non-Basic Software	NOT APPLICABLE for LinIf SWS
[BSW00323] API parameter checking	[LINIF376]
[BSW004] Version check	[LINIF383]
[BSW00409] Header files for production code error IDs	[LINIF266]
[BSW00385] List possible error notifications	[LINIF376]
[BSW00386] Configuration for detecting an error	[LINIF268]
[BSW161] Microcontroller abstraction	[LINIF129]
[BSW162] ECU layout abstraction	[LINIF129]
[BSW005] No hard coded horizontal interfaces within MCAL	Chapter 5
[BSW00415] User dependent include files	[LINIF243] [LINIF244] [LINIF245]
[BSW164] Implementation of interrupt service routines	No required interrupt functions
[BSW00325] Runtime of interrupt service routines	No required interrupt functions
[BSW00326] Transition from ISRs to OS tasks	No required interrupt functions
[BSW00342] Usage of source code and object code	The LinIf configuration enables both the source code and the binary way. See chapter 10
[BSW00343] Specification and configuration of time	[LINIF474]
[BSW160] Human-readable configuration data	the generation of the configuration data is not in the scope of LinIf
[BSW007] HIS MISRA C	This is mostly a requirement on

	the construction and not the design (i.e. SWS). The API chapter 8 is following MISRA C
[BSW00300] Module naming convention	Chapter 5.3
[BSW00413] Accessing instances of BSW modules	[LINIF197] [LINIF469]
[BSW00347] Naming separation of different instances of BSW drivers	Linlf is not a driver
[BSW00305] Self-defined data types naming convention	Chapter 8.1.2
[BSW00307] Global variables naming convention	The naming of parameters is made in chapter 10
[BSW00310] API naming convention	Chapter 8.2
[BSW00373] Main processing function naming convention	[LINIF384]
[BSW00327] Error values naming convention	[LINIF376]
[BSW00335] Status values naming convention	[LINIF039] [LINIF290] [LINIF316] [LINIF319]
[BSW00350] Development error detection keyword	[LINIF268]
[BSW00408] Configuration parameter naming convention	Chapter 10.2
[BSW00410] Compiler switches shall have defined values	Construction requirement and not a design requirement
[BSW00411] Get version info keyword	[LINIF279]
[BSW00436] Module Header File Structure for the Basic Software Memory Mapping	[LINIF590]
[BSW158] Separation of configuration from implementation	[LINIF241] [LINIF242] [LINIF243] [LINIF244] [LINIF245]
[BSW00314] Separation of interrupt frames and service routines	No interrupt functions in Linlf
[BSW00370] Separation of callback interface from API	Chapter 8.3
[BSW00348] Standard type header	Chapter 5.3
[BSW00353] Platform specific type header	Chapter 5.3
[BSW00361] Compiler specific language extension header	Chapter 5.3
[BSW00301] Limit imported information	Chapter 5.3
[BSW00302] Limit exported information	[LINIF242]
[BSW00328] Avoid duplication of code	Complete Linlf SWS. The Linlf supports multiple channels. Same code is executed for each channel (with different parameters)
[BSW00312] Shared code shall be reentrant	Some API calls needs to be reentrant. See chapter 8.
[BSW006] Platform independency	[LINIF129]
[BSW00357] Standard API return type	Chapter 8.2
[BSW00377] Module specific API return types	No module specific return types needed. See chapter 8.
[BSW00304] AUTOSAR integer data types	Only uint8 and uint16 are used
[BSW00355] Do not redefine AUTOSAR integer data types	No redefine made
[BSW00378] AUTOSAR boolean type	No Boolean return types used
[BSW00306] Avoid direct use of compiler and platform specific keywords	No platform specific keywords are used.
[BSW00308] Definition of global data	No global data is required.
[BSW00309] Global data with read-only constraint	No global data is required.
[BSW00371] Do not pass function pointers via API	Function pointers not used, Chapter 8.2
[BSW00358] Return type of init functions	[LINIF198] [LINIF350]
[BSW00414] Parameter of init function	[LINIF371]
[BSW00376] Return type and parameters of main processing functions	[LINIF384]
[BSW00359] Return type of callback functions	[LINIF378]
[BSW00360] Parameters of callback functions	[LINIF378]

[BSW00329] Avoidance of generic interfaces	Generic interfaces are not used
[BSW00330] Usage of macros / inline functions instead of functions	No restriction
[BSW00331] Separation of error and status values	NOT APPLICABLE for LinIf SWS
[BSW009] Module User Documentation	SWS template 1.19 is used
[BSW00401] Documentation of multiple instances of configuration parameters	Chapter 10.2
[BSW172] Compatibility and documentation of scheduling strategy	Chapter 8
[BSW010] Memory resource documentation	NOT APPLICABLE for LinIf SWS
[BSW00333] Documentation of callback function context	NOT APPLICABLE for LinIf SWS
[BSW00374] Module vendor identification	[LINIF280]
[BSW00379] Module identification	[LINIF280]
[BSW003] Version identification	NOT APPLICABLE for LinIf SWS
[BSW00318] Format of module version numbers	[LINIF280]
[BSW00321] Enumeration of module version numbers	NOT APPLICABLE for LinIf SWS
[BSW00341] Microcontroller compatibility documentation	NOT APPLICABLE for LinIf SWS
[BSW00334] Provision of XML	NOT APPLICABLE for LinIf SWS
[BSW00435] Header File Structure for the Basic Software Scheduler	[LINIF589]

Document: AUTOSAR requirements on Basic Software, cluster LIN

Requirement	Satisfied by
[BSW01501] Usage of LIN 2.0 specification	[LINIF248]
[BSW01504] Usage of AUTOSAR architecture only in LIN master	[LINIF248]
[BSW01522] Consistent data transfer	The LinIf will not make any copying of data from unprotected buffers
[BSW01560] Support for wake-up during transition to sleep-mode	Chapter 7.3.3.1
[BSW01567] Compatibility to LIN 2.0 protocol specification	[LINIF248]
[BSW01551] Multiple LIN channel support for interface	[LINIF386]
[BSW01568] Hardware independence	[LINIF129]
[BSW01569] LIN Interface initialization	[LINIF198]
[BSW01570] Selection of static configuration sets	[LINIF371]
[BSW01564] Schedule Table Manager	[LINIF202]
[BSW01546] Schedule Table Handler	[LINIF384]
[BSW01561] Main function	[LINIF384]
[BSW01549] Timer service for Scheduling	[LINIF474]
[BSW01571] Transmission request service	[LINIF201]
[BSW01514] Wake-up notification support	[LINIF378]
[BSW01515] API to wake-up by upper layer to LIN Interface	[LINIF205]
[BSW01502] RX indication and TX confirmation call-backs	[LINIF128]
[BSW01558] Check successful communication	[LINIF033]
[BSW01527] Notification for missing or erroneous receive LIN-PDU	[LINIF465] [LINIF466]
[BSW01523] API to send the LIN to sleep-mode	[LINIF204]
[BSW01565] Compatibility to LIN 2.0 protocol specification	LIN driver requirement
[BSW01553] Basic Software SPAL General requirements	LIN driver requirement
[BSW01552] Hardware abstraction LIN	LIN driver requirement
[BSW01503] Frame based API for send and received data	LIN driver requirement
[BSW01555] LIN Interface shall poll the LIN Driver for transmit/receive notifications	LIN driver requirement
[BSW01547] Support of standard UART and LIN optimized	LIN driver requirement
[BSW01572] LIN driver initialization	LIN driver requirement

[BSW01573] Selection of static configuration sets	LIN driver requirement
[BSW01563] Wake-up Notification	LIN driver requirement
[BSW01556] Multiple LIN channel support for driver	LIN driver requirement
[BSW01566] Transition to sleep-mode mode	LIN driver requirement
[BSW01524] Support of reduced power operation mode	LIN driver requirement
[BSW01526] Error notification	LIN driver requirement
[BSW01533] Usage of LIN 2.0 specification	[LINIF313]
[BSW01540] LIN Transport Layer Initialization	[LINIF350]
[BSW01545] LIN Transport Layer Availability	[LINIF098]
[BSW01534] Concurrent connection configuration	[LINIF062]
[BSW01574] Multiple Transport Layer instances	[LINIF314]
[BSW01539] Transport connection properties	Chapter 0
[BSW01544] Error handling	Chapter 7.5.2.4 and chapter 7.5.2.6

7 Functional specification

This chapter is organized in a way following the same order as the LIN 2.0 specification. This is not always the case since the LIN 2.0 specification sometimes put requirements in different parts of its document. The intention is to enable reading both documents in parallel. It is not required to reinvent the requirements already specified in the LIN 2.0 specification. However, there are specific details for AUTOSAR and parts that need to be specified since they are not specified enough or are missing. Specification of these parts will be made here.

The LIN Interface shall support the behavior of a master in the LIN 2.0 specification. The following requirements are the base requirements and the rest of the requirements in this chapter are refinements of this base requirement.

LINIF248: The LIN Interface shall support the behavior of the master in the LIN 2.0 specification.

The requirement above basically means that the communication from a LIN 2.0 master and the LIN Interface master will be equal.

LINIF249: The LIN Interface shall realize the master behavior so that existing slaves can be reused.

LINIF386: The LIN Interface shall be able to handle one or more LIN channels.

7.1 Frame Transfer

All the functionality of the Protocol Specification in the LIN 2.0 specification is used. Some parts of the specification need some clarification and additional requirements to suite the LIN Interface.

7.1.1 Frame types

The following requirements apply to the different frame types that are specified in the LIN 2.0 specification. The existing frame types are:

- Unconditional frame
- Event-triggered frame
- Sporadic frame
- Diagnostic frames MRF and SRF
- User-defined frame

The actual transmission/reception of the different frames is detailed in the chapters 7.1.2 Frame reception and 7.1.3 Frame transmission.

7.1.1.1 Unconditional frame

This is the normal frame type that is used in LIN clusters. Its transportation on the bus strictly follows the schedule table.

7.1.1.2 Event-triggered frame

Event-triggered frames are used to enable sporadic transmission from slaves. The normal usage for this type of frame is in non-time-critical functions.

Since more than one slave may respond to an event-triggered frame header, a collision may occur. The transmitting slaves shall detect this and withdraw from communication.

LINIF227: If a collision occurs in an event-triggered frame response, then the LIN Interface shall flag all associated unconditional frames attached to an event-triggered frame for polling (see the LIN 2.0 specification how this is realized).

LINIF176: The order in which the unconditional frames attached to an event-triggered frame are polled is given by the LIN Interface configuration (configuration parameter LinIfFramePriority).

7.1.1.3 Sporadic frame

The LIN 2.0 specification defines a sporadic frame. A more precise definition of the sporadic frames is needed here:

- Sporadic slot – This is a placeholder for the sporadic frames. The reason to name it “slot” is that it has no LIN frame ID.
- Sporadic frame – This is one of the unconditional frames that are attached to a sporadic slot.

The LIN 2.0 specification does not specify how slaves may transmit sporadic frames.

LINIF012: The master shall be the only allowed transmitter of a sporadic frame (defined in the LIN 2.0 specification).

LINIF436: Only an unconditional frame shall allocate a sporadic slot (defined in the LIN 2.0 specification).

Upper layers decide the transmission of a sporadic frame. Therefore, an API call must be available to set the sporadic frame pending for transmission.

LINIF470: The LIN Interface shall flag the specific sporadic frame (defined in the LIN 2.0 specification) for transfer.

LINIF471: The LIN Interface shall transmit the specific sporadic frame (defined in the LIN 2.0 specification) in the associated sporadic slot according to the priority of the sporadic frames.

The priority of the sporadic frames is the order in which the sporadic frames are listed in the LDF. The priority mechanism of the LDF is not applicable here.

LINIF014: The priority of sporadic frames (defined in the LIN 2.0 specification) allocated to the same schedule slot is defined by the list of sporadic frame descriptors provided by the configuration tool (i.e. the first sporadic frame in the list is the most prioritized one).

7.1.1.4 Diagnostic Frames MRF and SRF

The Master Request Frame (MRF) and Slave Response Frame (SRF) are frames with a fixed ID that are used for transportation of LIN 2.0 node configuration services and TP messages.

The LIN 2.0 specification is vague in specifying when MRF and SRF are to be transported and when the corresponding schedule entry is due. The LIN Interface processes the schedule (Schedule Table Manager) and therefore knows when a TP transmission is ongoing. Therefore, the following requirement can be stated:

LINIF066: The LIN Interface shall send a MRF if there is an ongoing TP transmission and there is data to be sent.

Note that also the node configuration mechanism uses the MRF but above requirement does only apply when the MRF is encountered in the schedule table. The node configuration shall have special schedule entries as seen below.

For the slave response frame, the master node sends only the header. Generally, it is always sent because the master cannot know whether the slave has anything to send in the response part of the frame. An exception to that is the case when the master node wishes to prevent reception of such a frame during a TP frame sequence because there is no buffer to store them.

LINIF023: The LIN Interface shall always send a SRF header when schedule entry is due except if the TP indicates that the upper layer is temporarily unable to provide a receive buffer.

7.1.1.5 User-defined frames

LINIF251: The LIN Interface shall not use user-defined frames (defined in the LIN 2.0 specification).

7.1.1.6 Reserved frames

The LIN 2.0 specification does not allow reserved frames.

LINIF472: The LIN Interface shall not use reserved frames (defined in the LIN 2.0 specification).

7.1.2 Frame reception

The LIN master controls the schedules and therefore initiates all frames on the bus.

The requirements in this chapter are applicable to all received frame types that are received by the master if scheduled and pending for transportation (e.g. a schedule entry with a SRF can be silent or pending for transportation).

7.1.2.1 Header

LINIF419: The LIN Interface shall call the function `Lin_SendHeader` of the LIN Driver module when a new schedule entry for a frame reception is due.

LINIF693: If the LIN Interface has transmitted the associated unconditional frame header attached to an event-triggered frame successfully, it shall reset the pending flag.

7.1.2.2 Response

Since no response part is transmitted by the master, there is no need to use the `Lin_SendResponse` call. The LIN Driver will automatically be set to reception state after the header is transmitted.

7.1.2.3 Status check

LINIF030: The LIN Interface shall determine the status of the LIN Driver module by calling the function `Lin_GetStatus` earliest after the maximum frame length and latest when the next schedule entry is due.

It is up to the LIN Interface module's implementer to find an efficient way to determine the status check of the LIN Driver. The normal implementation would be that the status is checked within each `LinIf_MainFunction` function call after the maximum frame length has passed. In this case, the frame transmission is still going on (busy) the status determination shall be checked again within the next `LinIf_MainFunction` function call (if the current `LinIf_MainFunction` does not start a new frame of course).

The **Figure 3** shows an example of how the frame transmission is initiated and confirmed on the bus.

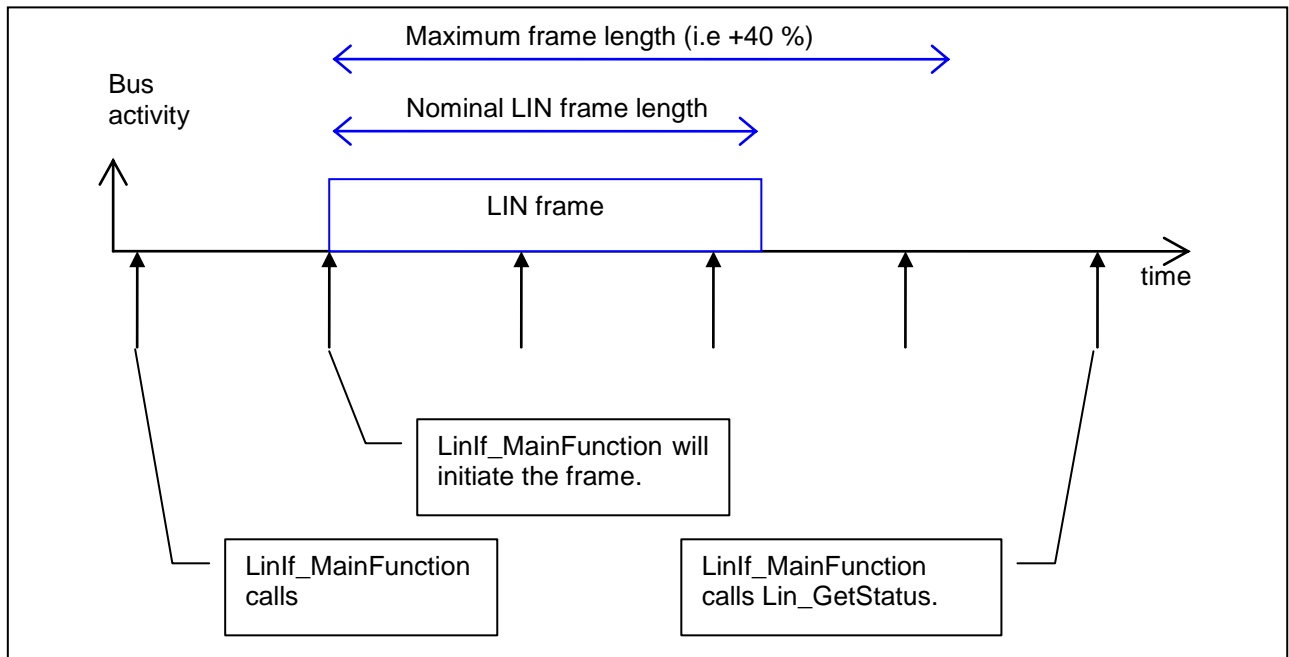


Figure 3 – Lin_GetStatus call example

When the status from the function `Lin_GetStatus` is returned and a frame is received, the following interpretation for different types of frames takes place:

LINIF033: When the LIN Interface determines the LIN Driver module’s status to be `LIN_RX_OK`, the LIN Interface shall issue a `<User_RxIndication>` callback with successful reception of frame.

LINIF259: When the LIN Interface is receiving an event-triggered frame and the LIN Driver module’s status is `LIN_RX_BUSY` or `LIN_RX_ERROR`, the LIN Interface shall not consider the status as an error.

This is considered to be a bus collision. More than one slave tried to respond to the event-triggered frame header. Instead, the following shall apply:

LINIF258: When the LIN Interface has received an event-triggered frame and determined the LIN Driver module’s status to be `LIN_RX_NO_RESPONSE`, the LIN Interface shall not consider this status as an error.

None of the slave wanted to reply on the event-triggered frame header.

LINIF254: When the LIN Interface has determined the LIN Driver module’s status as `LIN_RX_BUSY` or `LIN_RX_ERROR`, the LIN Interface shall consider the received frame as lost. Therefore, the LIN Interface shall raise the development error code `LINIF_E_RESPONSE`, if this frame is an unconditional frame and the development error detection is enabled.

LINIF466: When the LIN Interface has determined the LIN Driver module's status as `LIN_RX_NO_RESPONSE`, the LIN Interface shall consider the expected frame as lost. Therefore, the LIN Interface shall raise the development error code `LINIF_E_RESPONSE`, if this frame is an unconditional frame and the development error detection is enabled.

If there is disturbance on the bus, the LIN Interface may have problems sending out the header. The philosophy of the LIN 2.0 specification in this case is not reporting the error to upper layers. The same behavior applies also for transmitted and slave-to-slave frames.

LINIF458: The LIN Interface shall not report a header error to the upper layers when the return code of the LIN Driver module's function `Lin_GetStatus` is `LIN_TX_HEADER_ERROR`.

7.1.3 Frame transmission

A LIN frame is transmitted in the `LinIf_MainFunction` when a new schedule entry is due.

The requirements in this chapter are applicable to all transmitted frame types that are transmitted by the master if scheduled and pending for transportation (e.g. an unconditional frame that is scheduled is always pending for transportation, a sporadic frame slot may be pending for transportation or silent).

7.1.3.1 Header

LINIF224: The LIN Interface shall call the LIN Driver module's function `Lin_SendHeader` when a new schedule entry for a frame transmission is due.

7.1.3.2 Response

LINIF225: After the function `Lin_SendHeader` has returned, the LIN Interface shall call the PDU Router's function `<User_TriggerTransmit>` to get the data part of the frame (data in the LIN frame response).

The reason for requesting the data part after the header-request is to minimize the jitter on the bus.

LINIF226: If the return code of function `<User_TriggerTransmit>` is `E_OK`, the LIN Interface shall call the LIN Driver module's function `Lin_SendResponse` to provide the LIN Driver a pointer to the data part. If the return code is `E_NOT_OK`, the LIN Interface shall not transmit the data part of the frame.

7.1.3.3 Status check

LINIF128: If the return code of the function `Lin_GetStatus` is `LIN_TX_OK`, the LIN Interface shall issue a `<User_TxConfirmation>` callback.

LINIF036: If the return code of the function `Lin_GetStatus` is `LIN_TX_ERROR` and any LIN frame transmission is attempted, the LIN Interface shall consider the transmitted frame as lost.

LINIF465: If, just before a new frame is transmitted, the return code of the function `Lin_GetStatus` is `LIN_TX_BUSY`, the LIN Interface shall consider the old frame as lost and raise the development error code `LINIF_E_RESPONSE`, if the development error detection is enabled.

Note that since the master is also transmitting the response part of the LIN frame, the no response error is meaningless for transmitted frames.

LINIF463: If the LIN Interface has transmitted a sporadic frame successfully, it shall reset the pending flag.

7.1.4 Slave-to-slave communication

The third direction of a frame is the slave-to-slave communication. This is a supported but not recommended way to use the LIN bus. It creates dependencies between the slaves that are not desirable.

7.1.4.1 Header

LINIF416: The LIN Interface shall call the LIN Driver module's function `Lin_SendHeader` when a new schedule entry for a slave-to-slave communication is due.

7.1.4.2 Response

LINIF417: The LIN Interface shall not be involved in the slave-to-slave communication, in either transmission or reception of the response.

7.1.4.3 Status check

LINIF418: The LIN Interface shall not check the LIN Driver module's status after the transportation of the slave-to-slave communication response.

7.2 Schedules

The schedule table is the basis of all communication in an operational LIN cluster. Because the LIN Interface always operates as a LIN master, it has to process the schedule table.

Each channel may have separate sets of schedule tables. The time between starts of frames (delay) is a multiple of the time-base for the specific cluster. The time-base shall not be mixed up with the tick.

LINIF260: The LIN Interface's integrator shall define one time-base (configuration parameter `LinIfTimeBase`) for one LIN cluster.

The LIN Interface will use the time-base to invoke the function call `LinIf_MainFunction`.

LINIF261: The delay between processing two frames shall be a multiple of the LIN Interface time-base (configuration parameter `LinIfTimeBase`).

LINIF231: The LIN Interface shall provide a predefined schedule table per channel (named `NULL_SCHEDULE`).

LINIF263: The schedule table `NULL_SCHEDULE` shall contain no frames.

7.2.1 `LinIf_MainFunction`

The `LinIf_MainFunction` is the central processing function in the LIN Interface. It has to be called periodically. The task of the function `LinIf_MainFunction` is to poll the Schedule Table Manager, initiate frame transmission, resolve transmissions and interact with upper and lower layers.

LINIF474: The environment of the LIN Interface shall call the function `LinIf_MainFunction` periodically with a given period which is given by the configuration parameter `LinIfTimeBase`.

Further reference to this GCF value is referred to as "tick".

Example: The LIN Interface is connected to three buses using the time bases 6, 9 and 12 ms. The prime-factors are $6=3*2$, $9=3*3$ and $12=3*2*2$. The Greatest Common Factor here is 3, therefore the `LinIf_MainFunction` shall be called with a period of 3 ms. The 3 ms also defines the tick.

It is up to the designer of the LIN clusters to set the time-base for each cluster. Normally, these time bases are chosen to be equal (e.g. 5 ms).

7.2.2 Schedule table manager

The schedule table manager is not defined in the LIN 2.0 specification. This enables concurrent requests of a schedule table to be executed. Different upper layers may not have the possibility to ask other modules what schedule table they want to be executed. Therefore, a priority scheme is needed.

LINIF379: The priority range of the schedule table manager shall be 0 to 255 where 0 represents the highest priority.

The schedule table manager handles the schedule table and therefore indicates when the start of frame transmission and reception occurs. The LinIf_MainFunction polls the Schedule Table Manager for which frame to transport.

LINIF390: The schedule table manager of the LIN Interface shall have two types of schedule tables: RUN_CONTINUOUS and RUN_ONCE.

The idea to support two types of schedule tables is that there is a set of “normal” schedule tables defined as RUN_CONTINUOUS that are executed in normal communication. The RUN_ONCE schedule table is used for making specific requests from the LIN cluster. The use cases for RUN_ONCE schedule tables are:

- starting a diagnostic session
- make a LIN 2.0 node configuration
- poll event-triggered or sporadic frames

LINIF391: Each schedule table of the type RUN_ONCE shall have a fixed priority.

LINIF396: Each schedule table of the type RUN_ONCE shall have a unique priority for one channel.

LINIF392: All schedule tables of the type RUN_CONTINUOUS shall have the same and the lowest priority (i.e. 255).

The reason for having equal lowest priority for the RUN_CONTINUOUS schedule tables is that it should always be possible to change to a new RUN_CONTINUOUS schedule table. This means that a deadlock can never occur.

LINIF400: If the request is for a schedule table of the type RUN_CONTINUOUS and if the request has the same priority like other requests in the table, the schedule table manager of the LIN Interface shall always prioritize the latest schedule table request.

Special treatment is needed for the NULL_SCHEDULE. Since, it should be possible to set this schedule at any time.

LINIF444: If the LIN Interface’s environment is requesting a NULL_SCHEDULE (or set in case of initialization or sleep) and the requested queue is not of the type NULL_SCHEDULE, the schedule table manager of the LIN Interface shall flush the schedule table manager queue. The schedule table manager of the LIN Interface

shall change the requested queue to NULL_SCHEDULE when the schedule entry is due (even if the current is RUN_ONCE).

LINIF467: The LIN Interface shall reject a request for a schedule if the corresponding channel is in the state CHANNEL_SLEEP and raise the development error code LINIF_E_SCHEDULE_REQUEST_ERROR if the development error detection is enabled.

The LIN Interface allows changing of the current schedule table to another one. The function Linf_ScheduleRequest will select the schedule table to be executed. The actual switch to the new schedule is made as follows:

LINIF028: The LIN Interface shall select a new schedule table for execution during the next schedule entry if the current schedule is RUN_CONTINUOUS.

LINIF393: The LIN Interface shall execute a schedule table of the type RUN_ONCE from the first entry to the last entry before changing to a new schedule table.

LINIF495: If the switch from one schedule table to another schedule table has been performed, the schedule table manager shall call the function <User>_ScheduleRequest_Confirmation.

For the sporadic and event-triggered frames, a schedule table switch means that the states of these frames are not affected. For example, if the master is solving a collision of event-triggered frames, the solving continues even if the schedule table is changed (if the new schedule table contains the event-triggered frame of course).

LINIF029: The state of sporadic and event-triggered frames shall not be cleared when the schedule table is changed.

LINIF395: The number of schedule requests (i.e. the queue length) to handle per channel shall be configurable by the configuration parameter LinfScheduleRequestQueueLength.

LINIF397: The LIN Interface shall perform the latest requested schedule table of the type RUN_CONTINUOUS if no further schedule requests are left to be served after a RUN_ONCE schedule table.

LINIF485: The definition where the execution of a schedule table shall be proceeded in case it has been interrupted by a table of the type RUN_ONCE or MRF/SRF shall be configurable by the configuration parameter LinfResumePosition.

Note that since the function Linf_Init will set the NULL_SCHEDULE it means that there is always a latest requested schedule table. The following examples show how the schedule table manager works:

<i>Schedule table</i>	<i>Type</i>	<i>Priority</i>
Normal	RUN_CONTINUOUS	255
ReadById	RUN_ONCE	1

CheckSensors	RUN_ONCE	2
Empty		

Table 1 – Schedule table requests example

The schedule table requests in Table 1 are executed as shown in Figure 4.

ReadById	CheckSensors	Normal	Normal	...
----------	--------------	--------	--------	-----

Figure 4 – Schedule table execution example

Another example of the schedule table request:

<i>Schedule table</i>	<i>Type</i>	<i>Priority</i>
CheckSensors	RUN_ONCE	2
ReadById	RUN_ONCE	1
ReadById	RUN_ONCE	1
Empty		

Table 2 – Schedule table requests example

The schedule table requests in Table 2 are executed as shown in Figure 5.

ReadById	ReadById	CheckSensors	Null-schedule	...
----------	----------	--------------	---------------	-----

Figure 5 – Schedule table execution example

7.3 Network management

The network management described in this chapter is based on the LIN 2.0 specification network management and shall be not mixed up with the AUTOSAR network management.

In addition to the wake-up request and the go-to-sleep command, the network management is extended with node management. The node management describes more precisely than the LIN 2.0 specification how a node operates.

The LIN 2.0 specification defines a power management state-machine that all LIN nodes shall incorporate. However, this is not within the scope of the LIN Interface SWS.

LINIF237: The power management of the LIN 2.0 specification shall not be applicable to the LIN Interface.

7.3.1 Node Management

The LIN Interface shall operate as a state-machine. Each physical channel which is connected to the LIN Interface operates in a sub-state-machine.

7.3.1.1 LIN Interface state-machine

LINIF039: The LIN Interface shall have one state-machine.

The state-machine is depicted in **Figure 6**.

LINIF438: The LIN Interface state-machine shall have the state LINIF_UNINIT.

LINIF439: The LIN Interface state-machine shall have the state LINIF_INIT.

LINIF380: If development errors are enabled and the state of the LIN Interface is LINIF_UNINIT and a function is called except LinIf_Init, LinIf_GetVersionInfo and LinTp_GetVersionInfo, the corresponding function shall raise the development error code LINIF_E_UNINIT.

LINIF381: When the LIN Interface's environment has called the function LinIf_Init, the LIN Interface state-machine shall transit from LINIF_UNINIT to LINIF_INIT.

7.3.1.2 LIN channel sub-state-machine

The sub-state-machine of the state LINIF_INIT is depicted in **Figure 6**.

LINIF290: Each LIN channel shall have a separate channel state-machine.

LINIF441: The LIN channel sub-state-machine shall have the state CHANNEL_OPERATIONAL.

LINIF476: In the LIN channel sub-state CHANNEL_OPERATIONAL the corresponding LIN channel shall be initialized and operate normally.

LINIF189: The LIN Interface shall transmit LIN frame headers and receive/transmit responses only when the corresponding LIN channel is in the state CHANNEL_OPERATIONAL.

LINIF053: In the state CHANNEL_OPERATIONAL, the LIN Interface shall process the currently selected schedule table within the function LinIf_MainFunction.

LINIF507: The LIN Interface shall transit from LINIF_UNINIT to CHANNEL_OPERATIONAL when the function LinIf_Init is called.

LINIF442: The LIN channel sub-state-machine shall have the state CHANNEL_SLEEP.

LINIF478: The LIN Interface shall transit from the channel state CHANNEL_SLEEP to CHANNEL_OPERATIONAL when it has detected a wake-up request for the corresponding channel.

LINIF114: When entering or exiting the LIN channel state CHANNEL_SLEEP, the LIN Interface shall not set the hardware interface or the μ -controller into a new power mode.

LINIF043: When a channel is in the LIN channel state CHANNEL_SLEEP, the function Linf_MainFunction shall not initiate any traffic on the bus for the corresponding LIN channel.

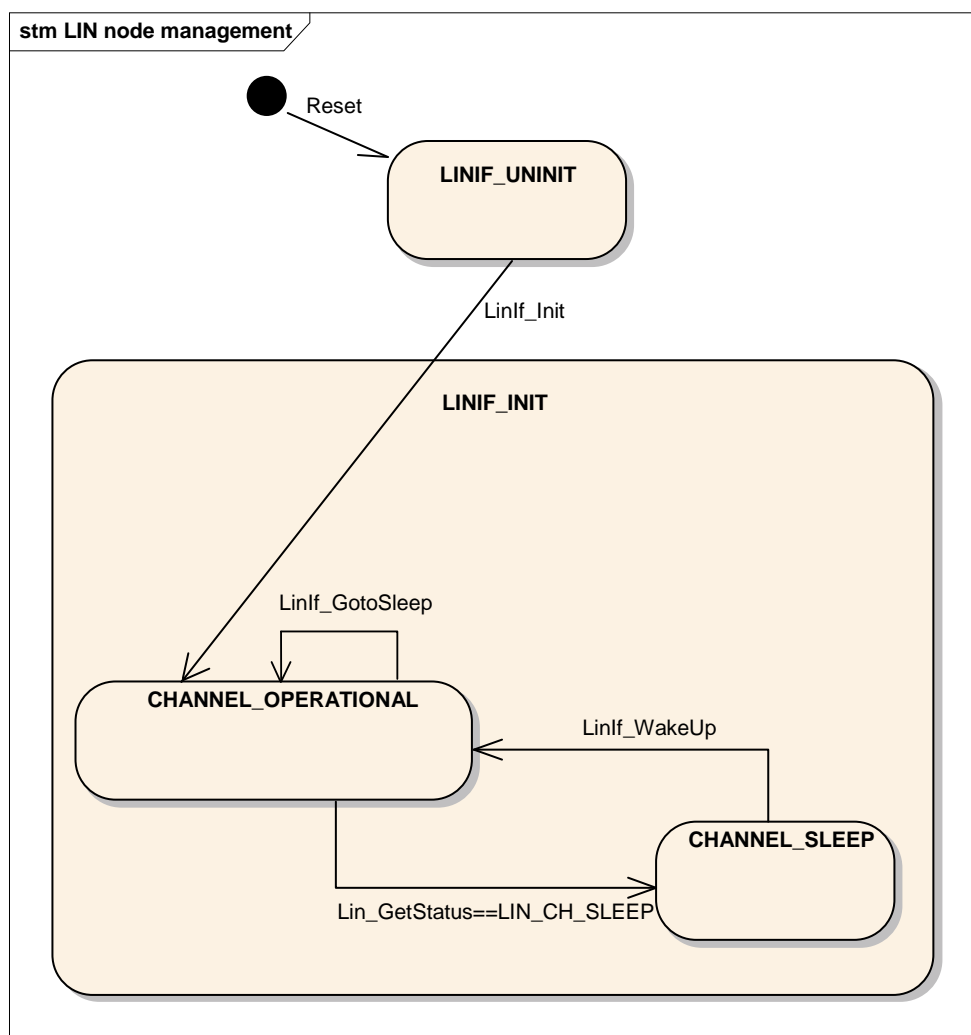


Figure 6 – LIN Interface state-machine and LIN Interface channel sub-state-machine

7.3.2 Go to sleep process

The function `LinIf_GotoSleep` initiates a transition into sleep mode on the selected channel/controller. The transition is carried out by transmitting a LIN diagnostic master request frame with its first data byte equal to 0 (zero). This is called the go-to-sleep command in the LIN 2.0 specification.

LINIF453: When processing the go-to-sleep command and the channel is not in the state `CHANNEL_SLEEP`, the function `LinIf_MainFunction` shall call the function `Lin_GoToSleep` instead of the scheduled frame latest when the next schedule entry is due.

LINIF597: When processing the go-to-sleep command and the channel is in the state `CHANNEL_SLEEP`, the function `LinIf_MainFunction` shall call the function `Lin_GoToSleepInternal` instead of the scheduled frame latest when the next schedule entry is due.

This means that the function `LinIf_MainFunction` can call the function `Lin_GoToSleep` in the interval starting when the previous frame is finished until the next schedule entry is due. This is up to the implementer to decide.

LINIF455: When processing the go-to-sleep command, the function `LinIf_MainFunction` shall call the function `Lin_GetStatus` of the LIN Driver module. When the return code of the function `Lin_GetStatus` is `LIN_CH_SLEEP`, the function `LinIf_MainFunction` shall set the channel state of the affected channel to `CHANNEL_SLEEP`. In this case, the go-to-sleep command transmission has successfully been performed.

LINIF454: When processing the go-to-sleep command, the function `LinIf_MainFunction` shall call the function `Lin_GetStatus` of the LIN Driver module. When the return code of the function `Lin_GetStatus` is not `LIN_CH_SLEEP`, the go-to-sleep command transmission has failed.

LINIF557: When the go-to-sleep command was sent successful or the function `Lin_GoToSleepInternal` was called, the LIN Interface shall invoke the function `<User>_GotoSleep_Confirmation` with the parameter `TRUE`.

LINIF558: When the go-to-sleep command was not sent successful, the LIN Interface shall invoke the function `<User>_GotoSleep_Confirmation` with the parameter `FALSE`.

LINIF293: When entering the `CHANNEL_SLEEP` state during the go-to-sleep command process, the function `LinIf_MainFunction` shall switch the current used schedule table switch to the `NULL_SCHEDULE`.

7.3.3 Wake up process

There are different possibilities to wake-up a LIN channel. Either the upper layer requests a wake-up through the `LinIf_WakeUp` call or a bus wake-up is detected. If a bus wake-up is detected, `LinIf_WakeUp` is also called when the upper layer enters the `FULL_COM` mode after a successful validation through the function `LinIf_Cbk_CheckWakeUp`.

LINIF496: When the return code of the function `LinIf_WakeUp` is `E_OK`, the LIN Interface shall issue the function `<User>_WakeUp_Confirmation` with the parameter `TRUE`.

LINIF670: When the return code of the function `LinIf_WakeUp` is `E_NOT_OK`, the LIN Interface shall not issue the function `<User>_WakeUp_Confirmation`.

7.3.3.1 Wakeup during sleep transition

It may happen that the upper layer requests a wake-up, when the upper layer has requested the go-to-sleep command to be transmitted and while it is pending (from the go-to-sleep request until the status check of the frame). In this case, the following shall apply:

LINIF459: If the go-to-sleep command is requested and the upper layer requests a wake-up before the go-to-sleep command is checked, the LIN Interface shall not send a wake-up on the bus and shall maintain the LIN channel state `CHANNEL_OPERATIONAL`.

LINIF460: When the LIN Interface has checked the go-to-sleep command during the transition to sleep, using the function `Lin_GetStatus` of the LIN Driver module and the return code of this function is `LIN_CH_SLEEP`, the LIN Interface shall call the function `Lin_WakeUp` to wake-up the channel again.

LINIF699: In case of **LINIF460**, LIN Interface shall not invoke the function `<User>_GotoSleep_Confirmation`, regardless of the return code of the function `Lin_GetStatus`.

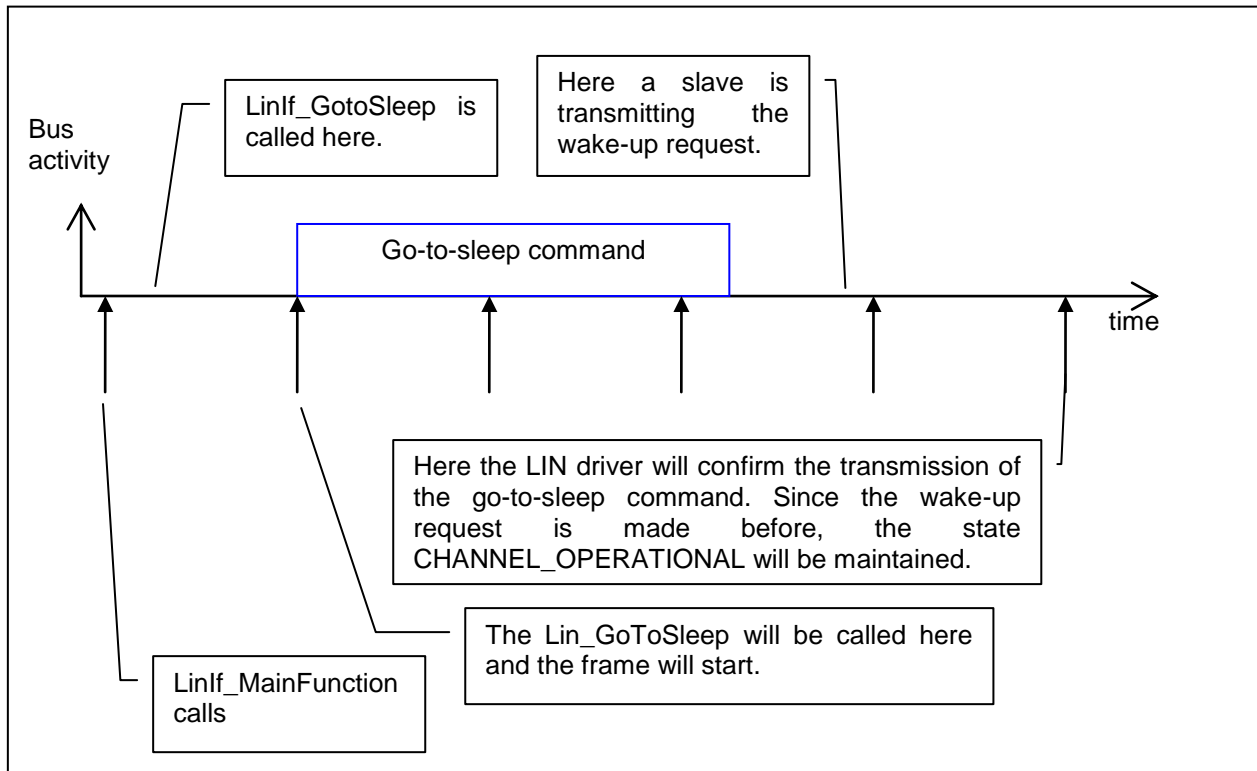


Figure 7 – Wake up requested before confirmation of go-to-sleep command

7.4 Status Management

The LIN Interface has to be able to report communication errors on the bus in the same manner as the LIN 2.0 specification describes. However, the reporting is different.

There is an internal reporting within the own node (by using the API call `l_ifc_read_status` defined in the LIN 2.0 specification) that sets the `Error_in_response` (not to be confused with the slave signal `Response_Error`) and the `Successful_transfer` bits. The strategy here is only to report errors and not to monitor successful transfers.

The conditions for the `Error_in_response` will be set in the LIN Interface in the same way as described in the LIN 2.0 specification but not reported in the same way. How the `Error_in_reponse` is handled is described in chapters 7.1.2.3 and 7.1.3.3.

7.5 Diagnostics and Node configuration

Note that node configuration here means the configuration described in the LIN 2.0 specification and has nothing to do with the AUTOSAR configuration.

The Diagnostic Transport Layer and the Node Configuration in LIN 2.0 specification share the MRF and SRF. This will not be a conflict since the Node Configuration is using the fixed frame types.

7.5.1 Node configuration

The Node Configuration in the LIN 2.0 specification is about configuring a slave to be able to operate in a LIN cluster and make the LIN cluster collision free (in terms of NAD and frame ID's).

The LIN 2.0 specification specifies two ways for the LIN master to configure slaves:

- By using the LIN 2.0 API and by using the services directly in the Schedule Table.
- By using the defined Node Configuration API.

The idea here is to store the Node Configuration services in the configuration. Therefore, only the Schedule Table approach is used.

LINIF401: The LIN Interface shall only do the Node Configuration (defined in the LIN 2.0 specification) by using services directly in the Schedule Table.

7.5.1.1 Node Model

The LIN 2.0 specification defines a Node Model that describes where the configuration is stored.

LINIF308: The LIN Interface shall not use a Node Model (defined in the LIN 2.0 specification).

The Node Model is meant only for slaves and not for masters.

7.5.1.2 Node Configuration services

The LIN Interface provides node configuration services as specified in the LIN 2.0 specification. The node configuration mechanism uses the same LIN frame structure as the LIN TP. The Node Configuration will only use Single Frames (SF) for transportation.

LINIF309: The LIN Interface shall support the Node Configuration requests "Assign Frame ID" (defined in the LIN 2.0 specification) and "Unassign Frame ID" (defined in the LIN 2.0 specification).

LINIF409: The LIN Interface shall support the FreeFormat (defined in the LIN 2.0 specification).

The response of the FreeFormat is not defined within the LIN 2.0 specification. Therefore, a response from a slave cannot be processed.

The Node Configuration requests “Assign NAD”, “Conditional change NAD” and “Data Dump” are optional in the LIN 2.0 specification. Thus, they shall be optional in the LIN Interface.

LINIF310: The support for the Node Configuration request “Assign NAD” (defined in the LIN 2.0 specification) and “Conditional change NAD” (defined in the LIN 2.0 specification) shall be pre-compile time configurable On/Off by the configuration parameter `LinIfNcOptionalRequestSupported`.

The LIN 2.0 specification states that the Data Dump request shall not be used in operational clusters.

LINIF408: The LIN Interface shall not support the Node Configuration request Data Dump (defined in the LIN 2.0 specification).

7.5.1.3 Node Configuration API

The Read-by-Identifier service is not considered as node configuration. It is more considered as a diagnostic service. Therefore, it is senseless to support the Read-by-Identifier service as a schedule table command.

LINIF090: The LIN Interface shall not support the function Read-by-Identifier (defined in the LIN 2.0 specification).

It is the responsibility of the diagnostic layer to support the function Read-by-Identifier.

7.5.1.4 Node Configuration in Schedule Table

The LIN 2.0 specification allows Node Configuration in schedule tables. This decouples the application from this functionality. Therefore, it is possible to store this functionality in the configuration.

A number of fixed MRFs are defined in the LIN 2.0 specification.

LINIF479: The LIN Interface shall process the fixed MRF entries without the interaction with an upper layer.

LINIF480: The LIN Interface shall not request a slave for an answer when the process of a fixed MRF fails.

It is possible to put a SRF in the schedule table after a node configuration command. A slave may answer to a node configuration command as defined in the LIN 2.0 specification.

LINIF404: The LIN Interface shall take no action if it has put a SRF in the schedule table after a node configuration command and if the answer of the slave is positive.

The response from the slave is optional for the node configuration requests according to the LIN 2.0 specification. However, if the SRF header is scheduled after a node configuration request, it is considered that a response is expected. Therefore, the following shall apply:

LINIF405: The LIN Interface shall raise the development error code LINIF_E_NC_NO_RESPONSE, if it has put a SRF in the schedule table after a node configuration command and it has not received an answer from the slave and the development error detection is enabled. The error shall always be reported, even if the previous configuration command was not transmitted successfully.

Note that there is no negative answer for node configuration requests defined in the LIN 2.0 specification. Only the function Read-by-Identifier supports a negative answer. As this function is not supported within the LIN Interface, there are no negative responses to process for the LIN Interface.

Note that the case when TP message is suddenly received from a slave without a TP request from the master does not exist on LIN.

7.5.2 Diagnostics – Transport Protocol

In the LIN 2.0 specification, the Transport Protocol (TP) is optional to implement. There are three types of diagnostics defined:

- Signal Based diagnostics
- User Defined diagnostics
- Diagnostic Transport Layer

It is only relevant to support the Diagnostic Transport Layer in the LIN Interface (and this is what is called the LIN TP). The Signal Based diagnostics has no meaning since signals are not defined here. The User Defined diagnostics shall not be used since all Diagnostic communication shall use the Diagnostic Transport Layer.

LINIF313: The LIN Interface shall support the Diagnostic Transport Layer (defined in the LIN 2.0 specification) without the contained Diagnostic API which represents the LIN TP.

The support of the LIN TP shall be configurable on/off to make the LIN Interface smaller.

LINIF387: The support for the LIN TP shall be pre-compile time configurable by the configuration parameter LinIfTpSupported.

It is possible that the LIN Interface has more than one channel (connected to more than one LIN cluster).

LINIF314: The LIN Interface shall support the start of a LIN TP message on each separate channel and they shall be independent of each other.

The designer of the schedule tables has to include master request and slave response frames. Otherwise, LIN TP transfer stalls.

The LIN TP is used to transport Diagnostic services and responses. No Diagnostic sessions are made in parallel. And service requests are always proceeded in sequence.

LINIF062: There shall be only one active LIN TP message at one time (i.e. only half-duplex) on one channel.

LINIF615: If the transmission for a further functional request is triggered (i.e. LinTp_Transmit is called) while the LIN Interface waits for a diagnostic response of a LIN slave or receives a diagnostic response the current TP handling (reception or P2 timeout supervision) shall be terminated.

LINIF641: If the transmission of LIN TP message is requested by the function LinTp_Transmit, the LIN Interface shall request a schedule table change to the diagnostic request schedule by calling the function BswM_LinTp_RequestMode with the parameter LINTP_DIAG_REQUEST.

LINIF642: If the transmission of LIN TP message requested by the function LinTp_Transmit is completed, the LIN Interface shall request a schedule table change to the diagnostic response schedule by calling the function BswM_LinTp_RequestMode with the parameter LINTP_DIAG_RESPONSE.

LINIF643: If the transmission of LIN TP response is completed, the LIN Interface shall request a schedule table change to the applicative schedule by calling the function BswM_LinTp_RequestMode with the parameter LINTP_APPLICATIVE_SCHEDULE.

LINIF646: If the configuration parameter LinTpScheduleChangeDiag is TRUE, a schedule table change to the diagnostic or applicative schedule by calling the function BswM_LinTp_RequestMode is done.

LINIF647: The LIN Interface shall provide an additional configuration parameter LinTpScheduleChangeDiag.

7.5.2.1 LIN TP initialization

LINIF098: The LIN TP functions except LinTp_Init and LinTp_GetVersionInfo shall only be available when the LIN TP and the corresponding channel has been initialized.

7.5.2.2 State-machine

The following Figure 8 shows the state-machine of the LIN TP.

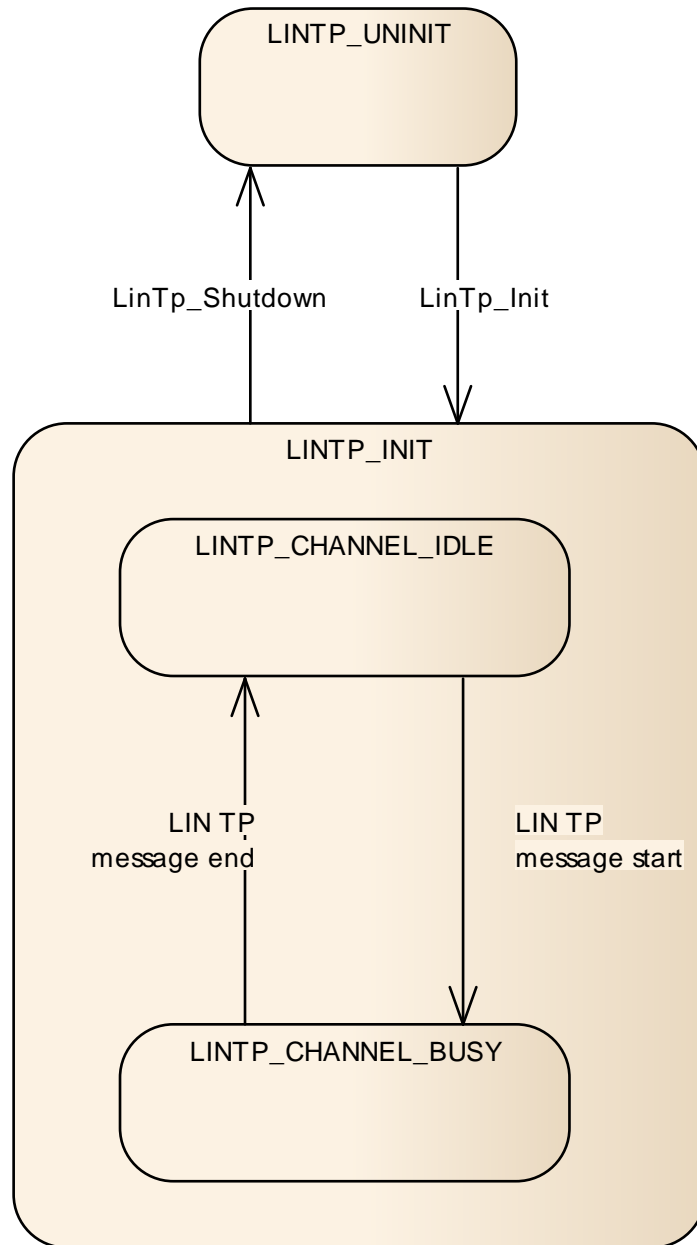


Figure 8 – LIN Transport Protocol state-machine

LINIF315: Each channel of the LIN Interface shall have one instance of the LIN TP state-machine which is called LIN TP channel state-machine.

LINIF316: The LIN TP state-machine shall have the state LINTP_UNINIT.

LINIF483: The LIN Interface shall set the LIN TP state to LINTP_UNINIT for all corresponding channels after a reset.

LINIF694: If development error detection is enabled and the state of the LIN Interface is LINTP_UNINIT and a LIN TP API function is called except LinTp_Init and LinTp_GetVersionInfo, the corresponding function shall raise the development error code LINIF_E_UNINIT.

LINIF319: The LIN TP state-machine shall have the state LINTP_INIT.

LINIF412: Each channel of the LIN TP state LINTP_INIT shall have a sub-state-machine.

LINIF450: The sub-state-machine of the LIN TP state LINTP_INIT shall have the state LINTP_CHANNEL_IDLE.

LINIF321: The LIN Interface shall start only a transmission of a TP message if the channel is in the sub-state LINTP_CHANNEL_IDLE.

LINIF414: The LIN Interface shall set the sub-state of a channel to LINTP_CHANNEL_IDLE when it has successfully terminated the transmission or reception of a LIN TP message.

LINIF688: The LIN Interface shall set the sub-state of a channel to LINTP_CHANNEL_IDLE when it has detected an unrecoverable error on this channel.

LINIF322: The sub-state-machine of the LIN TP state LINTP_INIT shall have the state LINTP_CHANNEL_BUSY.

LINIF323: The LIN Interface shall set the sub-state of a channel to LINTP_CHANNEL_BUSY when it has received a FF or a SF on the channel and it has detected it as a TP message (i.e. not conflicting with a configuration response from a LIN slave node).

7.5.2.3 LIN TP transmission

Since all frames must follow the schedule table, also LIN TP message must do this. All LIN TP messages are using the MRF and SRF for transportation.

The Figure 9 (left side) shows the initiation of the transmission of a TP message. Note that error handling is not shown, it will be described in a subsequent section.

LINIF328: After a transmission request from the upper layer, the LIN Interface shall call the function PduR_LinTpProvideTxBuffer to get the available transmission data for a SF, FF and CF.

Note that the first PduR_LinTpProvideTxBuffer might return just the FF data.

LINIF329: If the function `PduR_LinTpProvideTxBuffer` returns `BUFREQ_E_BUSY`, the LIN Interface shall not send the next MRF.

LINIF330: If the function `PduR_LinTpProvideTxBuffer` returns `BUFREQ_E_BUSY`, the LIN Interface shall later (implementation specific) retry to get the transmission data.

LINIF672: When the transmission data is provided, the LIN Interface shall resume the transmission of the MRF.

The Figure 9 (right side) shows the process after the LIN TP message has been initiated.

It may occur that the provided buffer by the PDU Router does not fit exactly into a frame or a number of frames.

LINIF695: After sending a MRF, the LIN Interface shall check if the remaining number of bytes in the transmit buffer (provided by the PDU Router) is sufficient to fill the next MRF. If it is not the case, the LIN Interface shall store the remaining bytes internally for sending in the next MRF and request a new transmission data to fill the next MRF completely.

This situation is indicated as “Save Buffer” in Figure 9 (right side).

LINIF068: When the LIN Interface has transmitted the last MRF (SF or CF) successfully, it shall notify the PDU Router by calling the function `PduR_LinTpTxConfirmation` with the result `NTFRSLT_OK`.

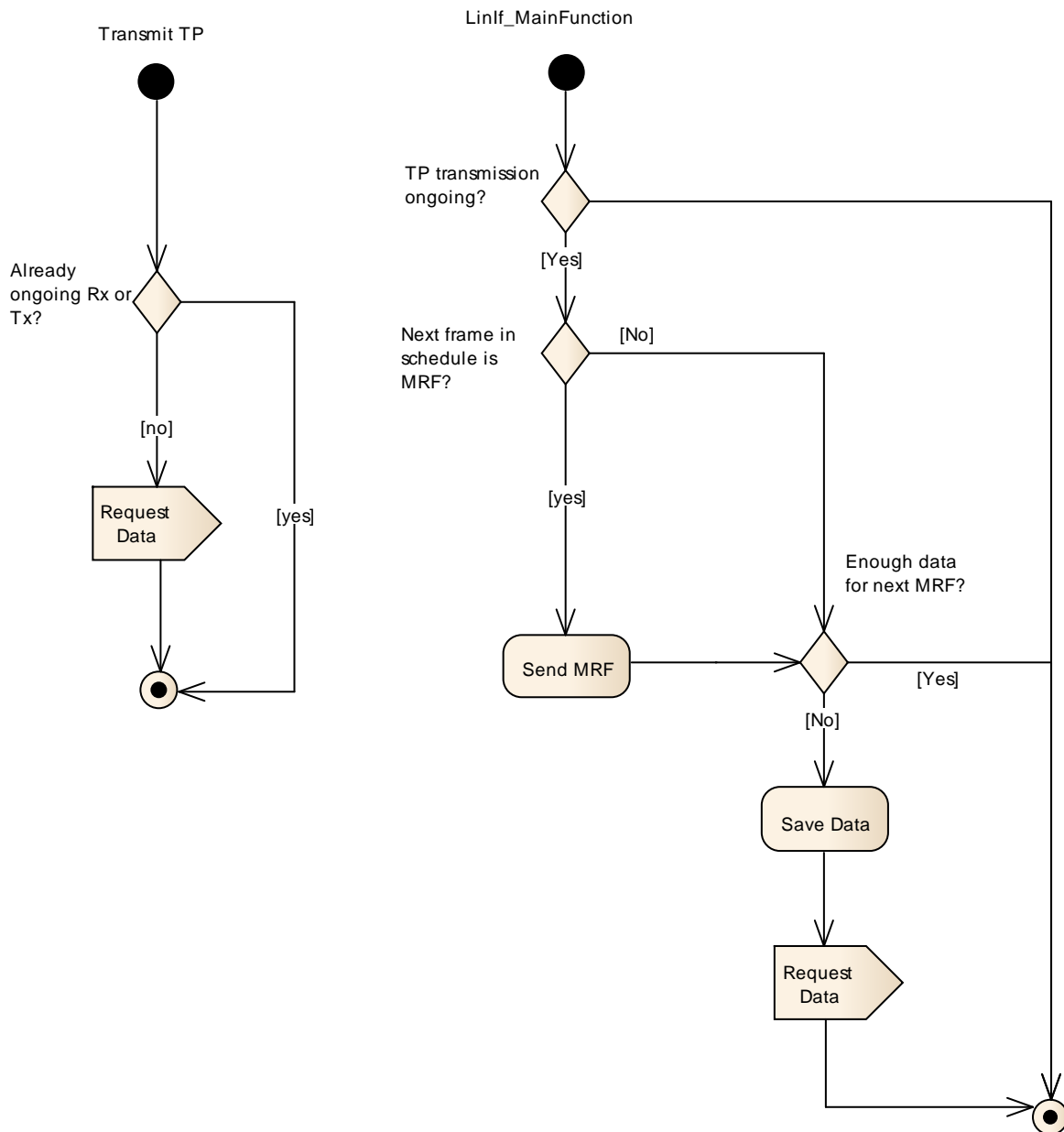


Figure 9 – TP transmission initiation (left side) and TP message transmission (right side)

7.5.2.4 LIN TP transmission error

LINIF069: If a LIN error on the MRF occurs, the LIN Interface shall abort the transmission and notify the PDU Router by calling the function PduR_LinTpTxConfirmation with the result NTFRSLT_E_NOT_OK.

LINIF073: If the function PduR_LinTpProvideTxBuffer returns BUFREQ_E_NOT_OK, the LIN Interface shall abort the transmission and notify the

PDU Router by calling the function `PduR_LinTpTxConfirmation` with the result `NTFRSLT_E_NOT_OK`.

LINIF673: When the LIN Interface has aborted the transmission, it shall request a schedule table change to the applicative schedule by calling the function `BswM_LinTp_RequestMode` with parameter `LINTP_APPLICATIVE_SCHEDULE` (see **LINIF646**).

7.5.2.5 LIN TP reception

The LIN Interface shall be prepared to receive a TP message anytime. The LIN slave will transport the TP message in a SRF to the LIN master (LIN Interface). The first SRF in the TP message will always be a FF or a SF.

Since the LIN Interface does not know when a slave is starting the response to a TP message, it must have the possibility to store part of the TP message.

LINIF075: The LIN Interface shall call the function `PduR_LinTpProvideRxBuffer` when the start of a TP message is indicated by the reception of a FF or a SF in a SRF.

LINIF076: The LIN Interface shall be able to convert the NAD from the transmitting LIN slave to an N-SDU Id that the upper layer understands.

LINIF221: The LIN Interface shall process and store subsequently received SRF frames in a buffer provided by the PDU Router (requesting the buffer with the function `PduR_LinTpProvideRxBuffer`).

LINIF077: When receiving a LIN TP message, the LIN Interface shall request a new buffer by calling the function `PduR_LinTpProvideRxBuffer` again if the buffer (provided by the PDU Router) is exhausted before the entire LIN TP message has been stored.

LINIF078: When the LIN Interface has received the last SRF (SF or CF) successfully, it shall notify the RDU Router by calling the function `PduR_LinTpRxIndication` with the result `NTFRSLT_OK`.

7.5.2.6 LIN TP reception error

If the LIN TP message reception is ongoing and a LIN error occurs on one SRF, the LIN Interface is not notified since it rejects the content. The way to detect errors is to check the contents of the successfully received SRF's.

LINIF079: In case the incorrect sequence number is received, the LIN Interface shall stop the current LIN TP message reception.

LINIF651: In case the unexpected PCI is received (a FF or a SF is received after a CF), the LIN Interface shall stop the current LIN TP message reception.

LINIF696: In case the unexpected PCI is received (a CF is received instead of a FF or a SF), the LIN Interface shall ignore the LIN TP message.

LINIF697: In case the unexpected PCI is received (unknown PCI type is received), the LIN Interface shall ignore the LIN TP message.

LINIF652: In case the invalid data length is received (a SF with a length of 0 (zero) or greater than 6, a FF with a length of less than 7), the LIN Interface shall ignore the LIN TP message.

LINIF612: The LIN Interface shall detect if the NAD (node address of addressed LIN slave) of a diagnostic response differs from the NAD of the request.

LINIF613: In case the incorrect NAD is received and the configuration parameter `LinTpDropNotRequestedNad` is TRUE, the LIN Interface shall stop the current LIN TP message reception.

LINIF648: In case the incorrect NAD is received and the configuration parameter `LinTpDropNotRequestedNad` is FALSE, the LIN Interface shall continue the current LIN TP message reception.

LINIF614: The LIN Interface shall request a schedule table change to the applicative schedule table by calling the function `BswM_LinTp_RequestMode` with the parameter `LINTP_APPLICATIVE_SCHEDULE` when it detects one of the errors that specified in **LINIF079**, **LINIF651** and **LINIF613** (see **LINIF646**).

LINIF081: In case the incorrect sequence number is received, the LIN Interface shall report this failure to PDU Router by calling the function `PduR_LinTpRxIndication` with the result `NTFRSLT_E_WRONG_SN`.

LINIF653: In case the unexpected PCI is received (a FF or a SF is received after a CF), the LIN Interface shall report this failure to PDU Router by calling the function `PduR_LinTpRxIndication` with the result `NTFRSLT_E_UNEXP_PDU`.

LINIF655: In case the incorrect NAD is received and the configuration parameter `LinTpDropNotRequestedNad` is TRUE, the LIN Interface shall report this failure to PDU Router by calling the function `PduR_LinTpRxIndication` with the result `NTFRSLT_E_UNEXP_PDU`.

LINIF080: The LIN Interface shall start a new LIN TP reception if it is receiving a FF or a SF when another LIN TP reception is ongoing. The old message shall be considered as lost.

In the situation where the LIN Interface (master) has encountered a permanent error (either by upper layer signaling permanent error or the bus indicated an erroneous

frame) the slave continues to transmit the rest of the frames when the master transmits a SRF header. The slave cannot know when the master has encountered a problem. The slave continues to transmit responses to the SRF headers. This means that no error-recovery is supported.

LINIF617: The LIN Interface shall provide a P2 timeout (configuration parameter `LinTpP2Timing`) observation in order to trigger a schedule table change by calling the function `BswM_LinTp_RequestMode` (from diagnostic response schedule to applicative schedule) in case the addressed LIN node does not respond.

LINIF618: The P2 timeout monitoring timer shall be started when a last MRF frame has been transmitted by master, which states the P2 timeout monitoring should be started only in LIN TP diagnostic mode.

LINIF698: The P2 timeout monitoring timer shall be stopped when a valid response has been received.

LINIF619: If the P2 timer value has expired before a valid response has been received, the LIN Interface shall terminate the ongoing transmission and trigger a schedule table switch to the applicative schedule by calling the function `BswM_LinTp_RequestMode` with the parameter `LINTP_APPLICATIVE_SCHEDULE` (see **LINIF646**).

LINIF620: The LIN Interface shall provide an additional configuration parameter `LinTpP2Timing`.

LINIF621: The LIN Interface shall provide UDS Response Pending handling. Therefore:

1. TP response PDUs containing an UDS response pending service are received and forwarded to the PDU Router as any other response PDUs.
2. After reception of a response pending frame the P2 timeout counter is reloaded with the timeout time `P2max`.

LINIF622: The LIN Interface shall provide an additional configuration parameter `LinTpP2Max` as `P2max` timeout time.

LINIF623: If more UDS response pending frames have been received than allowed (configuration parameter `LinTpMaxNumberOfRespPendingFrames`), the LIN Interface shall ignore all further diagnostic responses and shall switch to the applicative schedule by calling the function `BswM_LinTp_RequestMode` with the parameter `LINTP_APPLICATIVE_SCHEDULE` (see **LINIF646**).

LINIF624: The LIN Interface shall provide a configuration parameter `LinTpMaxNumberOfRespPendingFrames`.

7.5.2.7 Unavailability of receive buffer

The function `PduR_LinTpProvideRxBuffer` of the PDU Router may indicate that the requested buffer is not available.

The LIN Interface handles this case differently.

LINIF676: If the function `PduR_LinTpProvideRxBuffer` returns `BUFREQ_E_NOT_OK` or `BUFREQ_E_OVFL`, the LIN Interface shall abort the reception.

LINIF701: If the function `PduR_LinTpProvideRxBuffer` returns `BUFREQ_OK` with a smaller available buffer size than needed for the data received in the first SRF (SF or FF), the LIN Interface shall abort the reception.

LINIF087: When the LIN Interface has aborted the reception, it shall notify the PDU Router by calling the function `PduR_LinTpRxIndication` with the result `NTFRSLT E_NOT_OK`. Note that it shall not notify the PDU Router, if the first call of function `PduR_LinTpProvideRxBuffer` returns `BUFREQ_E_NOT_OK` or `BUFREQ_E_OVFL`.

LINIF678: When the LIN Interface has aborted the reception, it shall request a schedule table change to the applicative schedule by calling the function `BswM_LinTp_RequestMode` with the parameter `LINTP_APPLICATIVE_SCHEDULE` (see **LINIF646**).

7.6 Handling multiple channels and drivers

Normally, only one LIN driver (supporting multiple channels) is needed for the LIN Interface. However, rarely, some hardware configurations the ECU contain different LIN hardware. In such cases, multiple LIN drivers are used.

7.6.1 Multiple channels

LINIF461: Each channel of the LIN Interface shall have a unique channel index even when the LIN channels are located on different LIN Drivers and shall be configurable by the configuration parameter `LinIfChannelRef`.

7.6.2 Multiple LIN drivers

To be able to distinguish the LIN drivers, it is assumed that the LIN driver API names are extended with the `Vendor_Id` and a `Type_Id`.

LINIF462: The allocation of each channel to a LIN Driver shall be pre-compile time configurable by the configuration parameter `LinIfMultipleDriversSupported`.

The LIN driver shall also have name extensions for all published parameters, variables, types and files.

7.7 Error classification

LINIF266: Values for production code Event Ids are assigned externally by the configuration of the DEM. They are included via Dem.h.

LINIF267: Development error values are of type uint8.

LINIF376: The following Table 3 shows the available error codes, which shall be detected by the LIN Interface and the LIN TP:

<i>Type or error</i>	<i>Relevance</i>	<i>Related error code</i>	<i>Value [hex]</i>
API called without initialization of LIN Interface	Development	LINIF_E_UNINIT	0x00
Initialization API is used when already initialized	Development	LINIF_E_ALREADY_INITIALIZED	0x10
Referenced channel does not exist (identification is out of range)	Development	LINIF_E_NONEXISTENT_CHANNEL	0x20
API service called with wrong parameter	Development	LINIF_E_PARAMETER	0x30
API service called with invalid pointer	Development	LINIF_E_PARAMETER_POINTER	0x40
Schedule request queue overflow	Development	LINIF_E_SCHEDULE_OVERFLOW	0x50
Schedule request made in channel sleep	Development	LINIF_E_SCHEDULE_REQUEST_ERROR	0x51
LIN frame error detected	Development	LINIF_E_RESPONSE	0x60
Slave did not answer on a node configuration request	Development	LINIF_E_NC_NO_RESPONSE	0x61

Table 3 – Error codes for DET and DEM

7.8 Error detection

LINIF268: The detection of development errors is configurable (ON/OFF) at pre-compile time. The switch LinIfDevErrorDetect (see chapter 10) shall activate or deactivate the detection of all development errors.

LINIF269: If the LinIfDevErrorDetect switch is enabled API parameter checking is enabled. The detailed description of the detected errors can be found in chapter 7.7 and chapter 8.

LINIF270: The detection of production code errors can not be switched off.

7.9 Error notification

LINIF271: Detected development errors shall be reported to the Det_ReportError service of the Development Error Tracer (DET) if the preprocessor switch LinIfDevErrorDetect is set (see chapter 10).

LINIF272: Production errors shall be reported to Diagnostic Event Manager (DEM).

8 API specification

8.1 Imported types

8.1.1 Standard types

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

LINIF469:

<i>Module</i>	<i>Imported Type</i>
ComStack_Types	BufReq_ReturnType
	NetworkHandleType
	NotifResultType
	PduIdType
	PduInfoType
	PduLengthType
	TPParameterType
Lin	Lin_PduType
	Lin_StatusType
	Lin_ChannelConfigType
Std_Types	Std_ReturnType
	Std_VersionInfoType

8.1.2 Type definitions

This chapter shows the definitions of the types used in the LIN Interface.

8.1.2.1 LinIf_SchHandleType

LINIF197:

Name:	LinIf_SchHandleType
Type:	uint8
Description:	Index of the schedule table that is selectable and followed by LIN Interface. Value is unique per LIN channel/controller, but not per ECU. The number of schedule tables is limited to 255

8.1.2.2 LinIf_ConfigType

LINIF668:

Name:	LinIf_ConfigType
Type:	Structure
Range:	Implementation specific.
Description:	A pointer to an instance of this structure will be used in the initialization of the LIN

	Interface. The outline of the structure is defined in chapter 10 Configuration Specification.
--	--

8.1.2.3 LinTp_ConfigType

LINIF426:

Name:	LinTp_ConfigType
Type:	Structure
Range:	Implementation specific.
Description:	This is the base type for the configuration of the LIN Transport Protocol. A pointer to an instance of this structure will be used in the initialization of the LIN Transport Protocol. The outline of the structure is defined in chapter 10 Configuration Specification.

8.1.2.4 LinTp_Mode

LINIF629:

Name:	LinTp_Mode						
Type:	Enumeration						
Range:	<table border="1" style="width: 100%;"> <tr> <td>LINTP_APPLICATIVE_SCHEDULE</td> <td>Applicative schedule is selected</td> </tr> <tr> <td>LINTP_DIAG_REQUEST</td> <td>Master request schedule table is selected</td> </tr> <tr> <td>LINTP_DIAG_RESPONSE</td> <td>Slave response schedule table is selected</td> </tr> </table>	LINTP_APPLICATIVE_SCHEDULE	Applicative schedule is selected	LINTP_DIAG_REQUEST	Master request schedule table is selected	LINTP_DIAG_RESPONSE	Slave response schedule table is selected
LINTP_APPLICATIVE_SCHEDULE	Applicative schedule is selected						
LINTP_DIAG_REQUEST	Master request schedule table is selected						
LINTP_DIAG_RESPONSE	Slave response schedule table is selected						
Description:	This type denotes which Schedule table can be requested by LIN TP during diagnostic session						

8.2 LIN Interface API

This is a list of API calls provided for upper layer modules.

8.2.1 LinIf_Init

LINIF198:

Service name:	LinIf_Init
Syntax:	<pre>void LinIf_Init(const LinIf_ConfigType* ConfigPtr)</pre>
Service ID[hex]:	0x01
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	ConfigPtr Pointer to the LIN Interface configuration
Parameters (inout):	None

Parameters (out):	None
Return value:	None
Description:	Initializes the LIN Interface.

LINIF262: The function `LinIf_Init` shall generate an active independent schedule table for each configured channel.

LINIF371: The parameter `ConfigPtr` of the function `LinIf_Init` shall be only relevant for the configuration variant Variant 3. The parameter `ConfigPtr` shall be ignored for the configuration variant Variant 1 and the configuration variant Variant 2 but will be still there for compatible reasons.

LINIF486: If development error detection is enabled and the parameter `ConfigPtr` has an invalid value, the function `LinIf_Init` shall raise the development error code `LINIF_E_PARAMETER_POINTER`.

LINIF373: The function `LinIf_Init` shall accept a parameter that references to a LIN Interface configuration descriptor.

LINIF233: The function `LinIf_Init` shall set the schedule type `NULL_SCHEDULE` for each configured channel.

LINIF294: The function `LinIf_Init` shall call the function `Lin_InitChannel` for each channel, which is configured in the LIN Interface.

LINIF562: If development error detection is enabled and the LIN Interface is already initialized the function `LinIf_Init` shall raise the development error code `LINIF_E_ALREADY_INITIALIZED`.

8.2.2 `LinIf_GetVersionInfo`

LINIF340:

Service name:	<code>LinIf_GetVersionInfo</code>
Syntax:	<pre>void LinIf_GetVersionInfo(Std_VersionInfoType* versioninfo)</pre>
Service ID[hex]:	0x03
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	<code>versioninfo</code> Pointer to where to store the version information of this module.
Return value:	None
Description:	Returns the version information of this module.

LINIF278: The function `LinIf_GetVersionInfo` shall return the version information of this module. The version information includes:

- Two bytes for the vendor ID

- One byte for the module ID
- Three bytes version number. The numbering shall be vendor specific; it consists of the major, the minor and the patch version number of the module.
- The AUTOSAR specification version number shall not be included.

LINIF487: If source code for caller and callee of `LinIf_GetVersionInfo` is available, the LIN Interface should realize `LinIf_GetVersionInfo` as a macro, defined in the module's header file.

LINIF279: The function `LinIf_GetVersionInfo` shall be pre-compile time configurable On/Off by the configuration parameter `LinIfVersionInfoApi`.

8.2.3 LinIf_Transmit

LINIF201:

Service name:	LinIf_Transmit	
Syntax:	<pre>Std_ReturnType LinIf_Transmit(PduIdType LinTxPduId, const PduInfoType* PduInfoPtr)</pre>	
Service ID[hex]:	0x04	
Sync/Async:	Asynchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	LinTxPduId	Upper layer identification of the LIN frame to be transmitted (not the LIN protected ID). This parameter is used to determine the corresponding LIN protected ID (PID) and implicitly the LIN Driver instance as well as the corresponding LIN Controller device.
	PduInfoPtr	Pointer to a structure with frame related data: DLC and pointer to frame data buffer. This parameter is not used by this call.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Transmit request has been accepted. E_NOT_OK: Transmit request has not been accepted due to one or more of the following reasons: - LIN Interface has not been initialized - referenced PDU does not exist (identification is out of range) - referenced PDU is not a sporadic frame
Description:	Indicates a request.	

LINIF105: The function `LinIf_Transmit` shall indicate a request from an upper layer to transmit a sporadic frame specified by the parameter `LinTxPduId`.

LINIF341: The function `LinIf_Transmit` shall only mark a sporadic frame as pending for transmission and shall refuse non-sporadic frames.

LINIF106: The function `LinIf_Transmit` shall tolerate repeated invocations while the sporadic frame is still pending but set the pending PDU only once.

LINIF452: The configuration of the function `LinIf_Transmit` shall keep a lookup table to convert the upper layer identification to a frame pointer (configuration parameter `LinIfFrame`) that the LIN Interface recognizes.

8.2.4 LinIf_ScheduleRequest

LINIF202:

Service name:	LinIf_ScheduleRequest	
Syntax:	<pre>Std_ReturnType LinIf_ScheduleRequest(NetworkHandleType Channel, LinIf_SchHandleType Schedule)</pre>	
Service ID[hex]:	0x05	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant	
Parameters (in):	Channel	Channel index.
	Schedule	Identification of the new schedule to be set.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Schedule table request has been accepted. E_NOT_OK: Schedule table switch request has not been accepted due to one of the following reasons: - LIN Interface has not been initialized - referenced channel does not exist (identification is out of range) - referenced schedule table does not exist (identification is out of range) - No more requests are allowed because the request queue is full - State is sleep
Description:	Requests a schedule table to be executed.	

LINIF504: Schedule tables are configured by the `LinIfScheduleTable` container in the LIN Interface configuration.

LINIF389: The function `LinIf_ScheduleRequest` shall request the schedule table manager to be executed.

It is possible that each channel has multiple schedule tables. Each channel has a set of schedule tables that are selectable at run-time.

LINIF394: If development error detection is enabled and no more requests are allowed because the request queue is full, the function `LinIf_ScheduleRequest` shall raise the development error code `LINIF_E_SCHEDULE_OVERFLOW` and return `E_NOT_OK`.

LINIF563: If development error detection is enabled and an invalid channel is given, the function `LinIf_ScheduleRequest` shall raise the development error code `LINIF_E_NONEXISTENT_CHANNEL`.

8.2.5 LinIf_GotoSleep

LINIF204:

Service name:	LinIf_GotoSleep	
Syntax:	Std_ReturnType LinIf_GotoSleep(NetworkHandleType Channel)	
Service ID[hex]:	0x06	
Sync/Async:	Asynchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	Channel	Identification of the LIN channel.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Request to go to sleep has been accepted or sleep transition is already in progress or controller is already in sleep state E_NOT_OK: Request to go to sleep has not been accepted due to one or more of the following reasons: - LIN Interface has not been initialized - referenced channel does not exist (identification is out of range)
Description:	Initiates a transition into the Sleep Mode on the selected channel.	

LINIF488: The function LinIf_GotoSleep shall initiate a transition into sleep mode on the selected channel (see LINIF453 and LINIF597).

LINIF564: If development error detection is enabled and an invalid channel is given, the function LinIf_GotoSleep shall raise the development error code LINIF_E_NONEXISTENT_CHANNEL.

LINIF113: The function LinIf_GotoSleep shall have no effect on the channel referenced by the parameter Channel if the channel is already in the sleep state.

The function LinIf_GotoSleep will start the process of putting the cluster into sleep and not do it immediately.

The channels/controllers other than the one selected by Channel are not affected.

8.2.6 LinIf_WakeUp

LINIF205:

Service name:	LinIf_WakeUp	
Syntax:	Std_ReturnType LinIf_WakeUp(NetworkHandleType Channel)	
Service ID[hex]:	0x07	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant	
Parameters (in):	Channel	Identification of the LIN channel.
Parameters (inout):	None	

Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Request to wake up has been accepted or the controller is not in sleep state. E_NOT_OK: Request to wake up has not been accepted due to one or more of the following reasons: - LIN Interface has not been initialized - referenced channel does not exist (identification is out of range)
Description:	Initiates the wake up process.	

LINIF432: The function `LinIf_WakeUp` shall do nothing and return `E_OK` when the referenced channel is not in the sleep state.

The channels/controllers other than the one selected by `Channel` are not affected.

LINIF296: The function `LinIf_WakeUp` shall call the function `Lin_WakeUp` of the LIN Driver module to transmit a wake-up request on the selected channel if the channel is in the channel state `CHANNEL_SLEEP`.

LINIF565: If development error detection is enabled and an invalid channel is given, the function `LinIf_WakeUp` shall raise the development error code `LINIF_E_NONEXISTENT_CHANNEL`.

8.2.7 LinTp_Init

LINIF350:

Service name:	<code>LinTp_Init</code>	
Syntax:	<pre>void LinTp_Init(const LinTp_ConfigType* ConfigPtr)</pre>	
Service ID[hex]:	0x40	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	<code>ConfigPtr</code>	Pointer to the LIN Transport Protocol configuration.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Initializes the LIN Transport Layer.	

LINIF427: The parameter `ConfigPtr` of the function `LinTp_Init` shall be only relevant for the configuration variant Variant 3. The parameter `ConfigPtr` shall be ignored for the configuration variant Variant 1 and the configuration variant Variant 2 but will be still there for compatible reasons.

LINIF410: The LIN Interface's environment shall call the function `LinTp_Init` before using any other LIN TP function.

LINIF320: The function `LinTp_Init` shall set the sub-state `LINTP_CHANNEL_IDLE` for each configured channel of the LIN TP channel state-machine.

LINIF593: If development error detection is enabled and the LIN Interface is already initialized the function `LinTp_Init` shall raise the development error code `LINIF_E_ALREADY_INITIALIZED`.

8.2.8 LinTp_Transmit

LINIF351:

Service name:	LinTp_Transmit	
Syntax:	<pre>Std_ReturnType LinTp_Transmit(PduIdType LinTpTxSduId, const PduInfoType* LinTpTxInfoPtr)</pre>	
Service ID[hex]:	0x41	
Sync/Async:	Asynchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	LinTpTxSdulId	This parameter contains the unique identifier of the N-SDU (TP message) to be transmitted.
	LinTpTxInfoPtr	A pointer to a structure with N-SDU related data containing: <ul style="list-style-type: none"> - pointer to a N-SDU buffer - length of this buffer.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: The request can be started successfully E_NOT_OK: The request can not be started (e.g. Buffer has not been provided)
Description:	Requests the transfer of segmented data.	

LINIF326: The function `LinTp_Transmit` shall prepare a LIN TP message for transmission.

LINIF415: The LIN Interface's environment shall call the function `LinIf_Init` for initializing the referenced channel before using the function `LinTp_Transmit`.

LINIF413: The function `LinTp_Transmit` shall set the sub-state of the referenced channel to `LINTP_CHANNEL_BUSY`.

LINIF422: The function `LinTp_Transmit` shall convert the N-SDU Id (given by the parameter `LinTpTxSdulId`) to a specific channel and a destination NAD for the slave.

LINIF616: If the transmission for a further physical request is triggered while transmission of a previously triggered physical request is ongoing, the LIN Interface shall accept the new physical request and drop the old physical request.

The PDU Router may not make the message available directly. If necessary, it will split the message into parts and pass them to the LIN TP one at a time.

LINIF327: The function `LinTp_Transmit` shall request the PDU Router for a new buffer using the function `PduR_LinTpProvideTxBuffer`.

8.2.9 LinTp_GetVersionInfo

LINIF352:

Service name:	LinTp_GetVersionInfo
Syntax:	void LinTp_GetVersionInfo(Std_VersionInfoType* versioninfo)
Service ID[hex]:	0x42
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	versioninfo Pointer to where to store the version information of this module.
Return value:	None
Description:	Returns the version information of this module.

LINIF353: The function LinTp_GetVersionInfo shall return the version information of the LIN TP. The version information includes:

The version number consists of three parts:

- Two bytes for the vendor ID
- One byte for the module ID
- Three bytes version number. The numbering shall be vendor specific; it consists of the major, the minor and the patch version number of the module.
- The AUTOSAR specification version number shall not be included.

LINIF354: The function LinTp_GetVersionInfo shall be pre-compile time configurable On/Off by the configuration parameter LinTpVersionInfoApi.

8.2.10 LinTp_Shutdown

LINIF355:

Service name:	LinTp_Shutdown
Syntax:	void LinTp_Shutdown()
Service ID[hex]:	0x43
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	Shut downs the LIN TP.

LINIF356: The function LinTp_Shutdown shall close all pending transport protocol connection of the LIN TP, free all resources of the LIN TP and set the corresponding LIN TP module to the state LINTP_UNINIT.

LINIF433: The function `LinTp_Shutdown` shall affect all configured channels.

LINIF357: The function `LinTp_Shutdown` shall reject all pending transmissions and receptions and shall not report it.

LINIF482: The function `LinTp_Shutdown` shall interrupt ongoing LIN TP messages.

LINIF484: The function `LinTp_Shutdown` shall set the LIN TP state of the corresponding channels to `LINTP_UNINIT`.

8.2.11 `LinTp_CancelTransmit`

LINIF500:

Service name:	<code>LinTp_CancelTransmit</code>	
Syntax:	<pre>Std_ReturnType LinTp_CancelTransmit(PduIdType LinTpTxSduId)</pre>	
Service ID[hex]:	0x46	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	<code>LinTpTxSduId</code>	This parameter contains the Lin TP instance unique identifier of the Lin N-SDU which transfer has to be cancelled.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	<code>Std_ReturnType</code>	<code>E_NOT_OK</code> : Cancellation request of the transfer of the specified Lin N-SDU is rejected
Description:	This is a dummy method introduced for interface compatibility.	

LINIF490: The cancellation request shall always be rejected by returning `E_NOT_OK`.

Note: The `TxCConfirmation` is not invoked in the call of the `LinTp_CancelTransmit`.

8.2.12 `LinTp_ChangeParameter`

LINIF501:

Service name:	<code>LinTp_ChangeParameter</code>	
Syntax:	<pre>Std_ReturnType LinTp_ChangeParameter(PduIdType id, TPParameterType parameter, uint16 value)</pre>	
Service ID[hex]:	0x44	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	<code>id</code>	Identifier of the received N-SDU on which the reception parameter has to be changed.
	<code>parameter</code>	The selected parameter that the request shall change (STmin).
	<code>value</code>	The new value of the parameter.

Parameters (inout):	None
Parameters (out):	None
Return value:	Std_ReturnType E_NOT_OK: request is not accepted.
Description:	This service is used to request the change of reception parameter STmin for a specified N-SDU.

LINIF592: The change parameter request shall always be rejected by returning E_NOT_OK.

LINIF578: If development error detection is enabled and the parameter id has an invalid value, the function LinTp_ChangeParameter shall raise the development error code LINIF_E_PARAMETER.

8.3 Call-back notifications

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

The function prototypes of the callback functions are provided in the file LinIf_Cbk.h.

8.3.1 LinIf_Cbk_CheckWakeup

LINIF378:

Service name:	LinIf_Cbk_CheckWakeup
Syntax:	void LinIf_Cbk_CheckWakeup(NetworkHandleType Channel)
Service ID[hex]:	0x60
Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	Channel Identification of the LIN channel
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	Will be called when the EcuM has been notified about a wakeup on a specific LIN channel.

The LIN Interface will recognize the caller by the parameter of the function LinIf_Cbk_CheckWakeup.

LINIF510: The function LinIf_Cbk_CheckWakeup shall issue the call of the underlying function Lin_WakeupValidation.

LINIF566: If development error detection is enabled and an invalid channel is given, the function LinIf_Cbk_CheckWakeup shall raise the development error code LINIF_E_NONEXISTENT_CHANNEL.

The function LinIf_Cbk_CheckWakeup may be called in an interrupt or polling mode.

8.4 Scheduled functions

These functions are directly called by Basic Software Scheduler. The following functions shall have no return value and no parameter. All functions shall be non-reentrant.

8.4.1 LinIf_MainFunction

LINIF384:

Service name:	LinIf_MainFunction
Syntax:	void LinIf_MainFunction()
Service ID[hex]:	0x80
Timing:	FIXED_CYCLIC
Description:	The main processing function of the LIN Interface.

Design hint: The function LinIf_MainFunction may be interrupted by other LIN Interface functions. Critical areas that are also modified by other functions shall be protected. Other LIN Interface API calls that may touch the same resources are the LinIf_GotoSleep and LinIf_Transmit.

LINIF473: The function LinIf_MainFunction shall operate all channels.

LINIF286: The function LinIf_MainFunction shall poll the Schedule Table Manager which frame shall be transported.

LINIF287: Only the function LinIf_MainFunction shall process the transportation (transmission and reception) of frames.

8.5 Expected Interfaces

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

8.5.1 Mandatory Interfaces

This chapter defines all interfaces that are required to fulfill the core functionality.

LINIF359:

API function	Description
BswM_LinTp_RequestMode	Function called by LinTP to request a mode for the corresponding LIN channel. The LinTp_Mode mainly correlates to the LIN schedule table that should be used.
Lin_GetStatus	Gets the status of the LIN driver.
Lin_GoToSleep	The service instructs the driver to transmit a go-to-sleep-command on the addressed LIN channel.
Lin_GoToSleepInternal	Sets the channel state to LIN_CH_SLEEP, enables the wake-up detection and optionally sets the LIN hardware unit.

Lin_InitChannel	(Re-)initializes a LIN channel.
Lin_SendHeader	Sends a LIN header.
Lin_SendResponse	Sends a LIN response.
Lin_WakeUp	Generates a wake up pulse.
Lin_WakeupValidation	Identifies LIN channels.
PduR_LinTpProvideRxBuffer	Provides Rx Buffer for the LIN TP
PduR_LinTpProvideTxBuffer	Provide Tx data for the LIN TP
PduR_LinTpRxIndication	Rx indicator for the LIN TP
PduR_LinTpTxConfirmation	Tx confirmation for the LIN TP

8.5.2 Optional interfaces

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

LINIF360:

<i>API function</i>	<i>Description</i>
Det_ReportError	Service to report development errors.

8.5.3 Configurable interfaces

In this chapter, all interfaces are listed, where the target function of any upper layer to be called has to be set up by configuration. These call-out services are specified and implemented in the upper communication modules, which use the LIN Interface according to the AUTOSAR BSW architecture. The specific call-out notification is specified in the corresponding SWS document (see chapter [3 Related documentation]).

As far the interface name is not specified to be mandatory, no call-out is performed, if no API name is configured. This chapter describes only the content of notification of the call-out, the call context inside the LIN Interface and exact time by the call event.

<User>_NotificationName – This condition is applied for such interface services that will be implemented in the upper layer ('user') and called by the LIN Interface. This condition displays the symbolic name of the functional group in a call-out service in the corresponding upper layer. Each upper layer can define no, one or several call-out services for the same functionality (i.e. transmit confirmation).

8.5.3.1 <User>_ScheduleRequest_Confirmation

LINIF520:

Service name:	< User >_ScheduleRequest_Confirmation
Syntax:	void < User >_ScheduleRequest_Confirmation(NetworkHandleType channel)

Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	channel Identification of the LIN channel
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	The LinIf will call this function when the schedule table change request has been performed.

LINIF600: Configuration of <User>_ScheduleRequest_Confirmation: The name of the API <User>_ScheduleRequest_Confirmation which will be called by the LIN Interface module shall be configured for the LIN Interface module by parameter LinIfScheduleRequestConfirmationUL.

8.5.3.2 <User>_GotoSleep_Confirmation

LINIF521:

Service name:	< User >_GotoSleep_Confirmation
Syntax:	void < User >_GotoSleep_Confirmation(NetworkHandleType channel, boolean success)
Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	channel Identification of the LIN channel success True if goto sleep was successfully sent, false otherwise
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	The LinIf will call this function when the go to sleep command is sent unsuccessfully/successfully on the bus.

LINIF601: Configuration of <User>_GotoSleep_Confirmation: The name of the API <User>_GotoSleep_Confirmation which will be called by the LIN Interface module shall be configured for the LIN Interface module by parameter LinIfGotoSleepConfirmationUL.

8.5.3.3 <User>_WakeUp_Confirmation

LINIF522:

Service name:	< User >_WakeUp_Confirmation
Syntax:	void < User >_WakeUp_Confirmation(NetworkHandleType channel, boolean success)
Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	channel Identification of the LIN channel

	success	True if wakeup was successfully sent, false otherwise
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	The LinIf will call this function when the wake up signal command is sent unsuccessfully/successfully on the bus.	

LINIF602: Configuration of <User>_WakeUp_Confirmation: The name of the API <User>_WakeUp_Confirmation which will be called by the LIN Interface module shall be configured for the LIN Interface module by parameter LinIfWakeupConfirmationUL.

8.5.3.4 <User_TriggerTransmit>

LINIF528:

Service name:	<User_TriggerTransmit>	
Syntax:	Std_ReturnType <User_TriggerTransmit>(PduIdType TxPduId, PduInfoType* PduInfoPtr)	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant for different PduIds. Non reentrant for the same PduId.	
Parameters (in):	TxPduId	ID of the SDU that is requested to be transmitted.
Parameters (inout):	PduInfoPtr	Contains a pointer to a buffer (SduDataPtr) to where the SDU shall be copied to. On return, the service will indicate the length of the copied SDU data in SduLength.
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: SDU has been copied and SduLength indicates the number of copied bytes. E_NOT_OK: No SDU has been copied. PduInfoPtr must not be used since it may contain a NULL pointer or point to invalid data.
Description:	The lower layer communication module requests the buffer of the SDU for transmission from the upper layer module.	

LINIF608: Configuration of <User_TriggerTransmit>: The name of the API <User_TriggerTransmit> which will be called by the LIN Interface module shall be configured for the LIN Interface module by parameter LinIfTxTriggerTransmitUL.

8.5.3.5 <User_TxConfirmation>

LINIF529:

Service name:	<User_TxConfirmation>	
Syntax:	void <User_TxConfirmation>(PduIdType TxPduId)	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant for different PduIds. Non reentrant for the same PduId.	
Parameters (in):	TxPduId	ID of the I-PDU that has been transmitted.
Parameters	None	

(inout):	
Parameters (out):	None
Return value:	None
Description:	The lower layer communication module confirms the transmission of an I-PDU.

LINIF609: Configuration of <User_TxConfirmation>: The name of the API <User_TxConfirmation> which will be called by the LIN Interface module shall be configured for the LIN Interface module by parameter LinIfTxConfirmationUL.

8.5.3.6 <User_RxIndication>

LINIF530:

Service name:	<User_RxIndication>	
Syntax:	<pre>void <User_RxIndication>(PduIdType RxPduId, const PduInfoType* PduInfoPtr)</pre>	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant for different PduIds. Non reentrant for the same PduId.	
Parameters (in):	RxPduId	ID of the received I-PDU.
	PduInfoPtr	Contains the length (SduLength) of the received I-PDU and a pointer to a buffer (SduDataPtr) containing the I-PDU.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Indication of a received I-PDU from a lower layer communication module.	

LINIF610: Configuration of <User_RxIndication>: The name of the API <User_RxIndication> which will be called by the LIN Interface module shall be configured for the LIN Interface module by parameter LinIfRxIndicationUL.

9 Sequence diagrams

This chapter shows use cases for LIN communication and API usage. As the communication is in real-time, it is not easy to show the real-time behavior in the UML dynamic diagrams. It is advisable to read the corresponding descriptive text to each UML diagram.

To show the behavior of the modules in the different use cases, there are local function calls made to show what is done and when to get information. It is not mandatory to use these local functions. They are here just to make the use cases more understandable.

Note that all parameters and return types are omitted to make the diagrams easier to read and understand. If needed for clarification the parameter value or return value are shown.

9.1 Frame Transmission

The following use case shows the transmission of a LIN frame. The first call of the `LinIf_MainFunction` requests transmission of the header and the response. During the second call, the frame is under transmission. In the third call of the `LinIf_MainFunction`, the frame is finished.

The `RequestFrame` call in the diagram is the interface call to the Schedule Table Manager. The `LinIf_MainFunction` gets the frame to send and the delay to the next frame.

The `CopyBuffer` call is to show that the copying of the SDU is made in the LIN Driver and not in the LIN Interface.

The dynamic diagram in Figure 10 does not show any timing information. The timing information is depicted in **Figure 11** following the diagram.

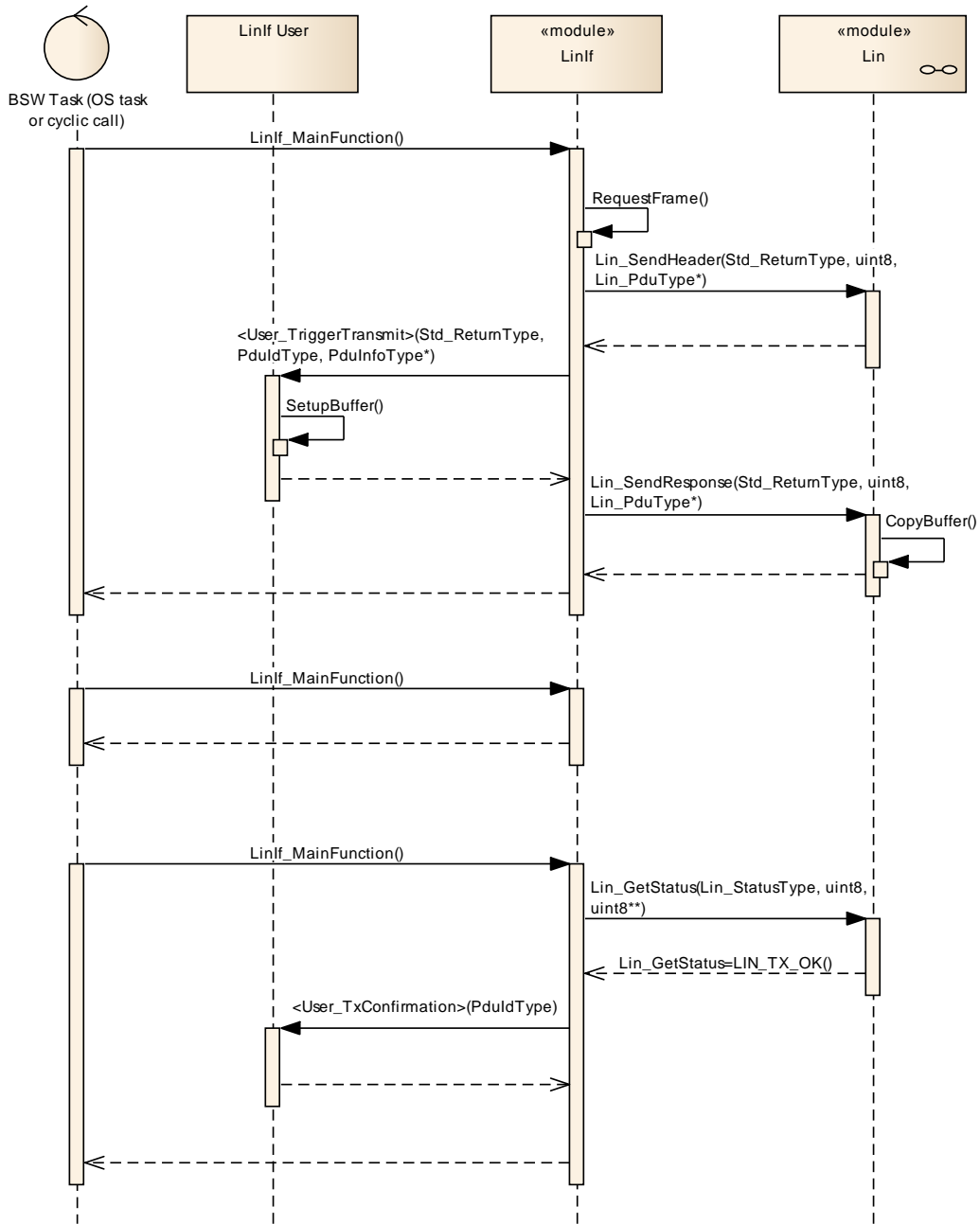


Figure 10 – Frame transmission

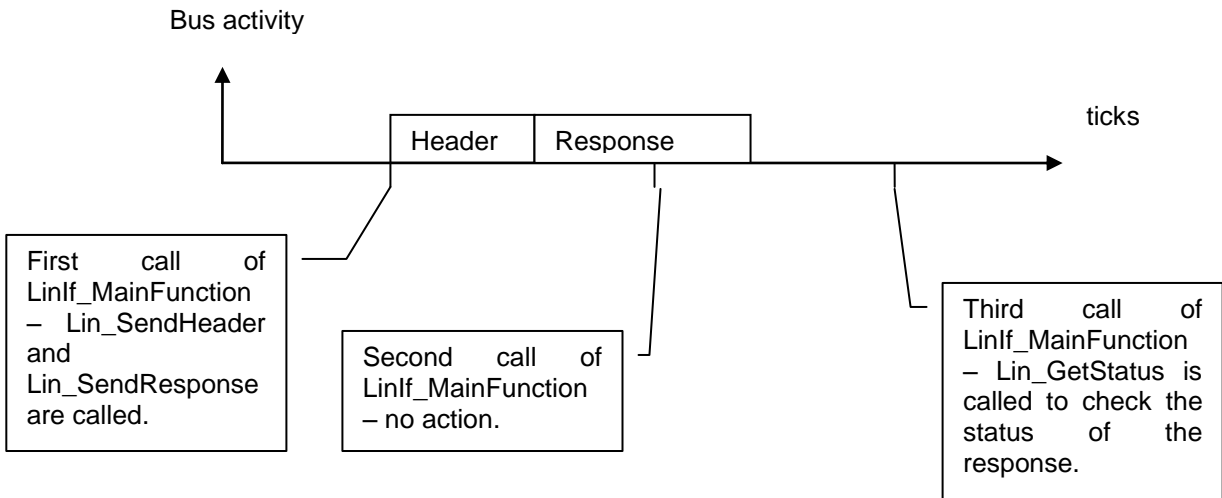


Figure 11 – Timing information for transmitted frame

9.2 Frame Reception

The following use case shows the reception of a LIN frame. The first call of the `LinIf_MainFunction` requests transmission of the header. During the second call, the frame is under transmission. In the third call, the frame is finished (this call is called after the maximum frame length).

The `RequestFrame` call in the diagram is the interface call to the Schedule Table Manager. The `LinIf_MainFunction` gets the frame to send and the delay to the next frame.

The `AllocateRxBuffer` call is to show that the storage of the received frame is made in the LIN Driver and not in the LIN Interface.

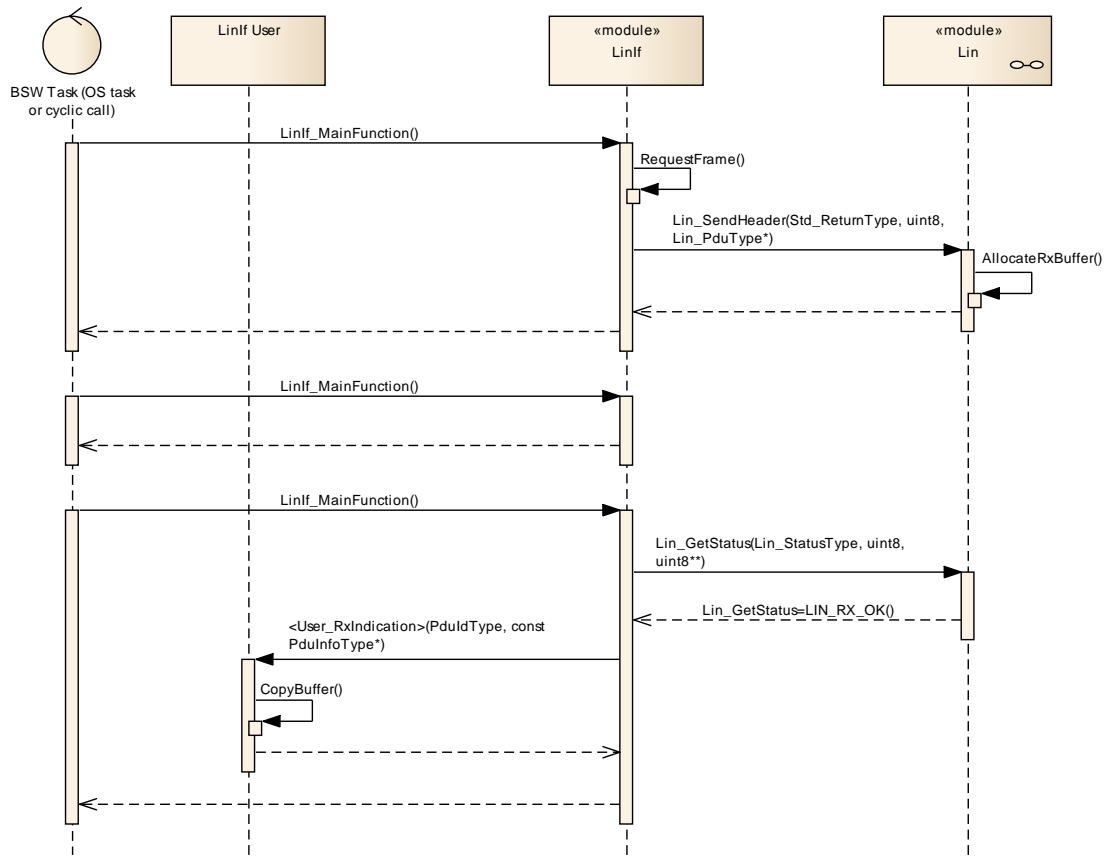
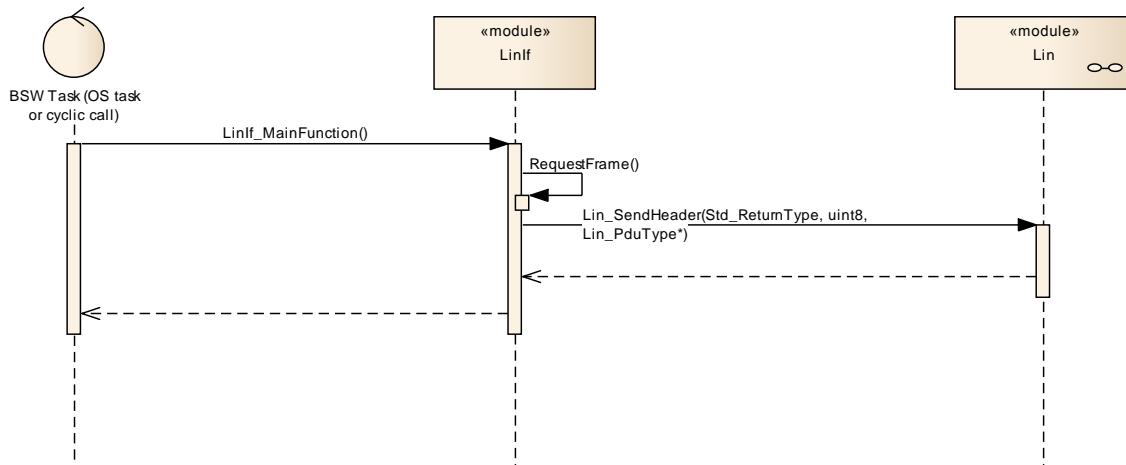


Figure 12 – Frame reception

9.3 Slave to slave communication

The third direction for a LIN frame is that two slaves communicate with each other. In this case, the master (LIN Interface) transmits the header and one slave transmits the response. The difference between the transmit direction is that the master does not monitor the response of the frame. Therefore, the frame header is transmitted and no further action is made.



Status:Proposed by WP11-2.1.1 as per LinIf SWS 1.1.2
Description:
Comments:

Figure 13 – Slave to slave communication

9.4 Reception and transmission at the same time

This use case shows the situation where a LIN frame is received and a new LIN frame is initiated in the same LinIf_MainFunction. The purpose is to propose a way to handle this situation without introducing too much jitter. It shows only the LinIf_MainFunction that processes the both frames.

The RequestFrame call in the diagram is the interface call to the Schedule Table Manager. The LinIf_MainFunction gets the frame to send and the delay to the next frame.

The CopyBuffer call is to show that the copying of the SDU is made in the LIN Driver and not in the LIN Interface.

The AllocateRxBuffer call is to show that the storage of the received SDU is made in the LIN Driver and not in the LIN Interface.

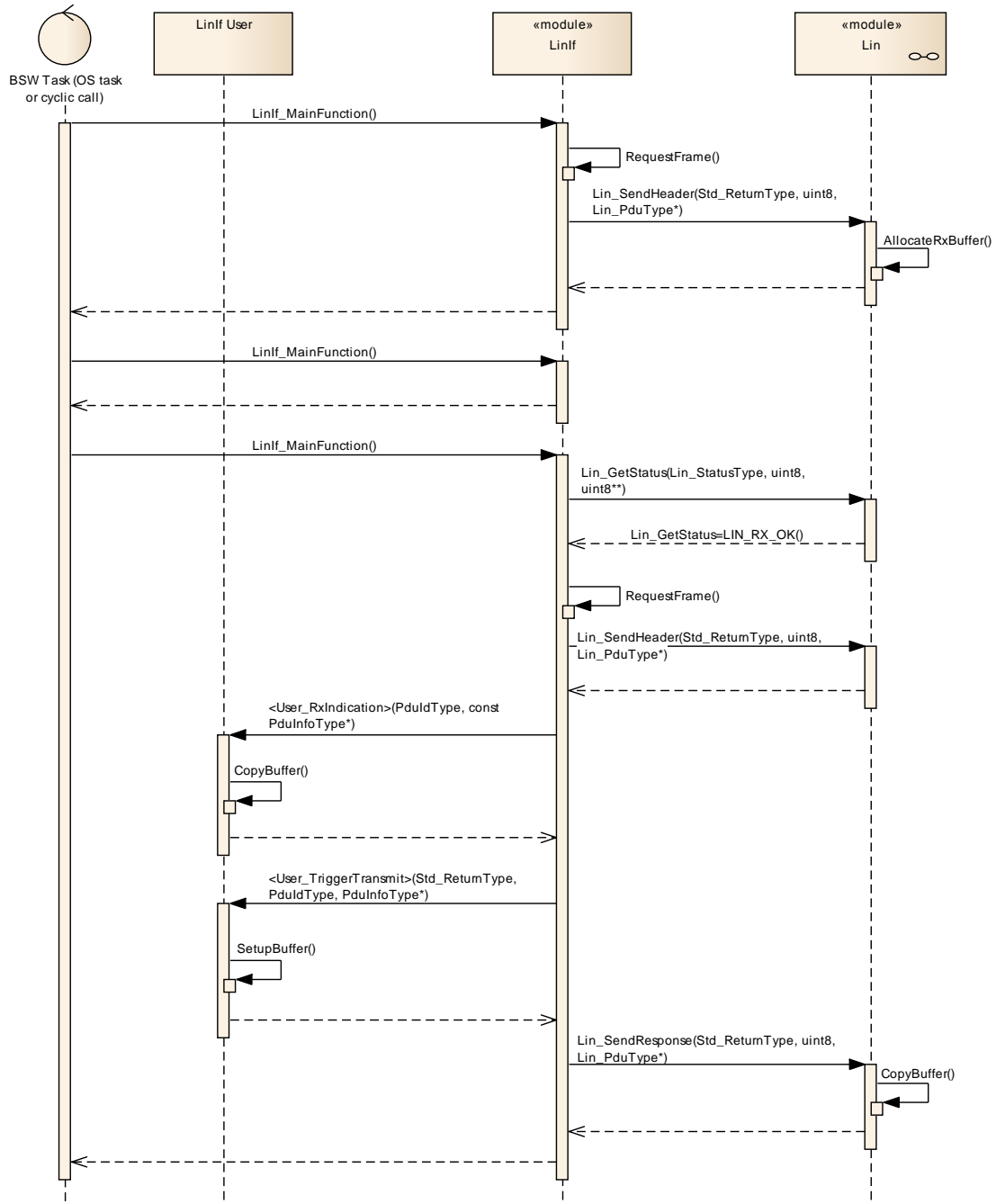


Figure 14 – One frame received and one frame transmitted

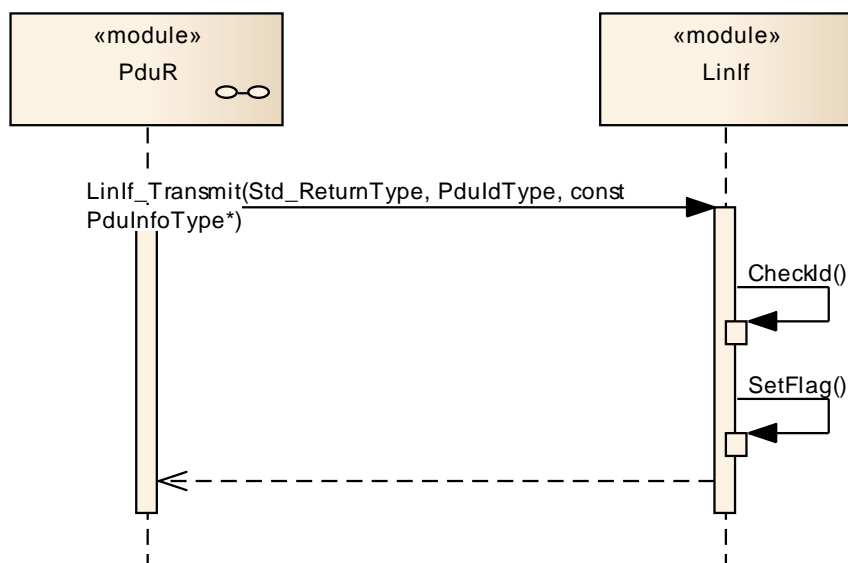
9.5 Sporadic frame

The following use case shows an upper layer requesting transmission of a sporadic frame. Actually, this call does not initiate the transmission of the frame since the

schedule table must be followed. It just marks the frame for transmission. When the sporadic slot (note that the schedule entry for a sporadic frame is a slot and not a frame) is due in the schedule table, the LinIf_MainFunction transmits the sporadic frame as a normal transmitted frame and according to the priority rules for sporadic frames.

The CheckId function is to show that the LIN Interface must check what frame is passed (convert the ID from the upper layer to the correct PID) from the upper layer.

The SetFlag function is a local function to flag the sporadic frame for transmission in the LIN Interface. There is one flag for each sporadic frame.



Status: Proposed by WP11-2.1.1 as per LinIf SWS 1.1.2
 Description:
 Comments:

Figure 15 – Sporadic frame

9.6 Event-triggered frame

There are three results for an event-triggered frame:

1. No answer
2. One slave node answers
3. Two or more slaves answers so that there is a collision on the bus

All three use cases are shown below.

9.6.1 With no answer

The following use case shows the transmission of an event-triggered frame header and no response.

The first call of the `LinIf_MainFunction` requests transmission of the header. During the second call, the frame is under transmission. In the third call, the frame is finished (this call is called after the maximum frame length).

The `RequestFrame` call in the diagram is the interface call to the Schedule Table Manager. The `LinIf_MainFunction` gets the frame to send and the delay to the next frame.

The `AllocateRxBuffer` call is to show that the storage of the received SDU is made in the LIN Driver and not in the LIN Interface.

No slave responds to the event-triggered frame header. The `LinIf_MainFunction` recognizes this situation and takes no action since this is not considered to be a communication error.

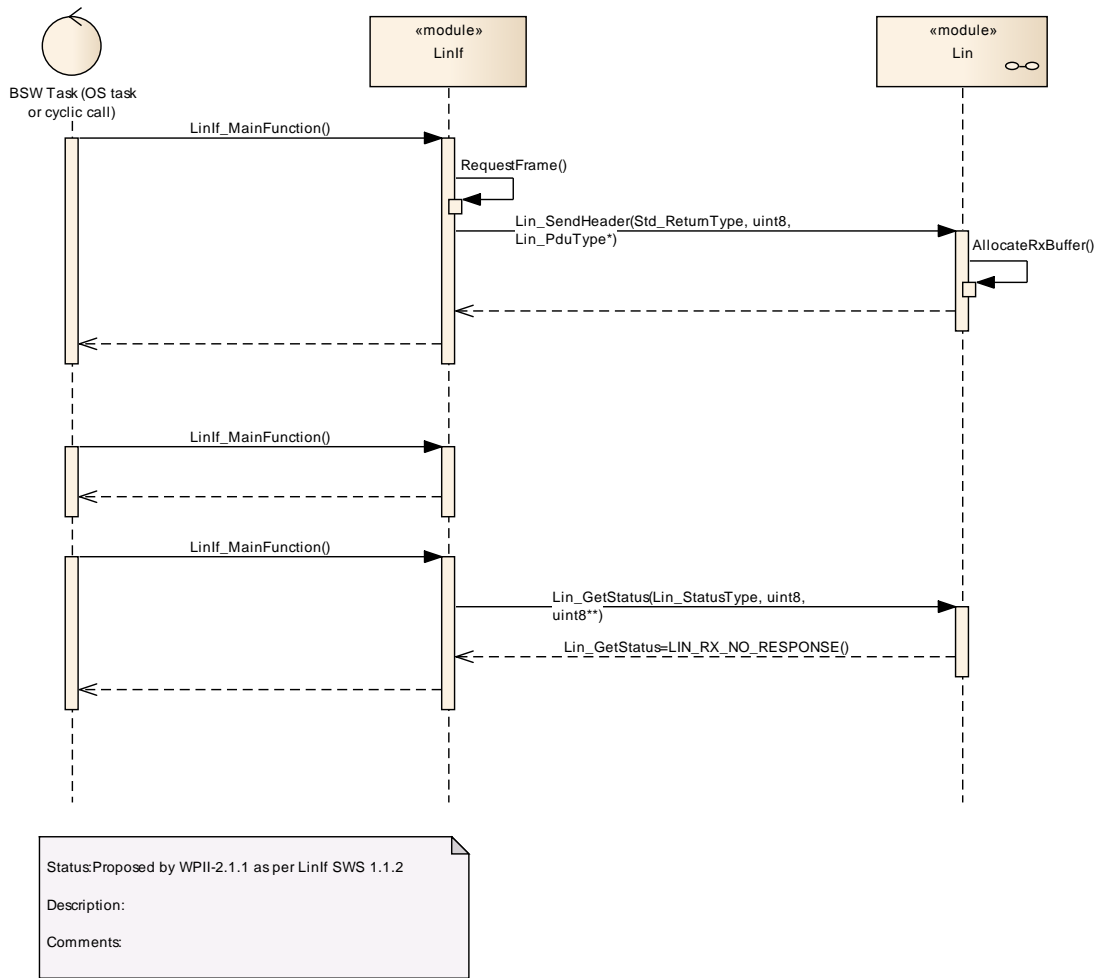


Figure 16 – Event-triggered frame with no answer

9.6.2 With answer (No collision)

The following use case shows the transmission of an event-triggered frame header with a response from one slave.

The first call of the LinIf_MainFunction requests transmission of the header. During the second call, the frame is under transmission. In the third call, the frame is finished (this call is called after the maximum frame length).

The RequestFrame call in the diagram is the interface call to the Schedule Table Manager. The LinIf_MainFunction gets the frame to send and the delay to the next frame.

The AllocateRxBuffer call is to show that the storage of the received SDU is made in the LIN Driver and not in the LIN Interface.

The ResolvePid call is to show that the received PID in the first data field is converted to the PdulId that upper layer understands.

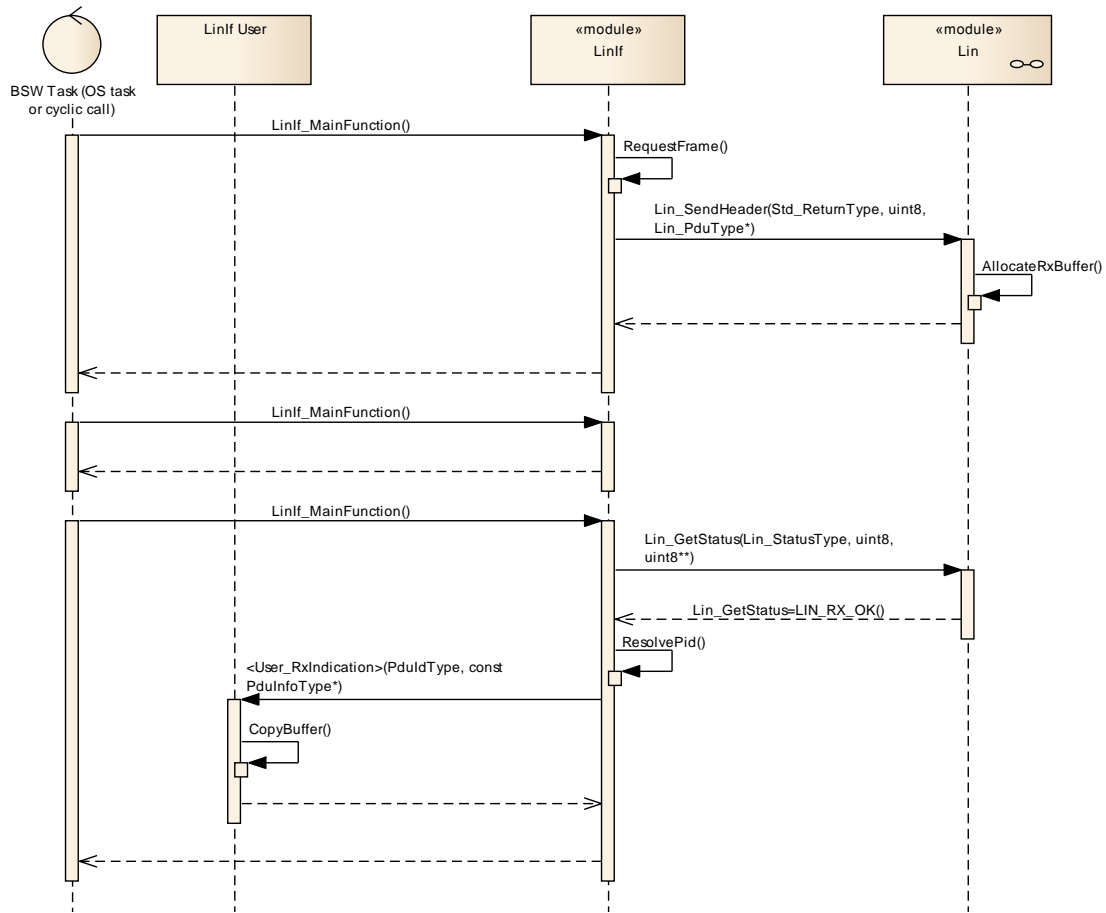


Figure 17 – Event-triggered frame with answer (no collision)

9.6.3 With collision

The following use case shows the transmission of an event-triggered frame header with a response from more than one slave. This means that there is a collision in the response field.

The first call of the LinIf_MainFunction requests transmission of the header. During the second call, the frame is under transmission. In the third call, the frame is finished (this call is called after the maximum frame length).

The RequestFrame call in the diagram is the interface call to the Schedule Table Manager. The LinIf_MainFunction gets the frame to send and the delay to the next frame.

The AllocateRxBuffer call is to show that the storage of the received SDU is made in the LIN Driver and not in the LIN Interface.

The local function SetAllEvFramesPending sets the Unconditional frames that are connected to this event-triggered frame (note that the event-triggered frame is a frame and not a slot – compare with the sporadic frame).

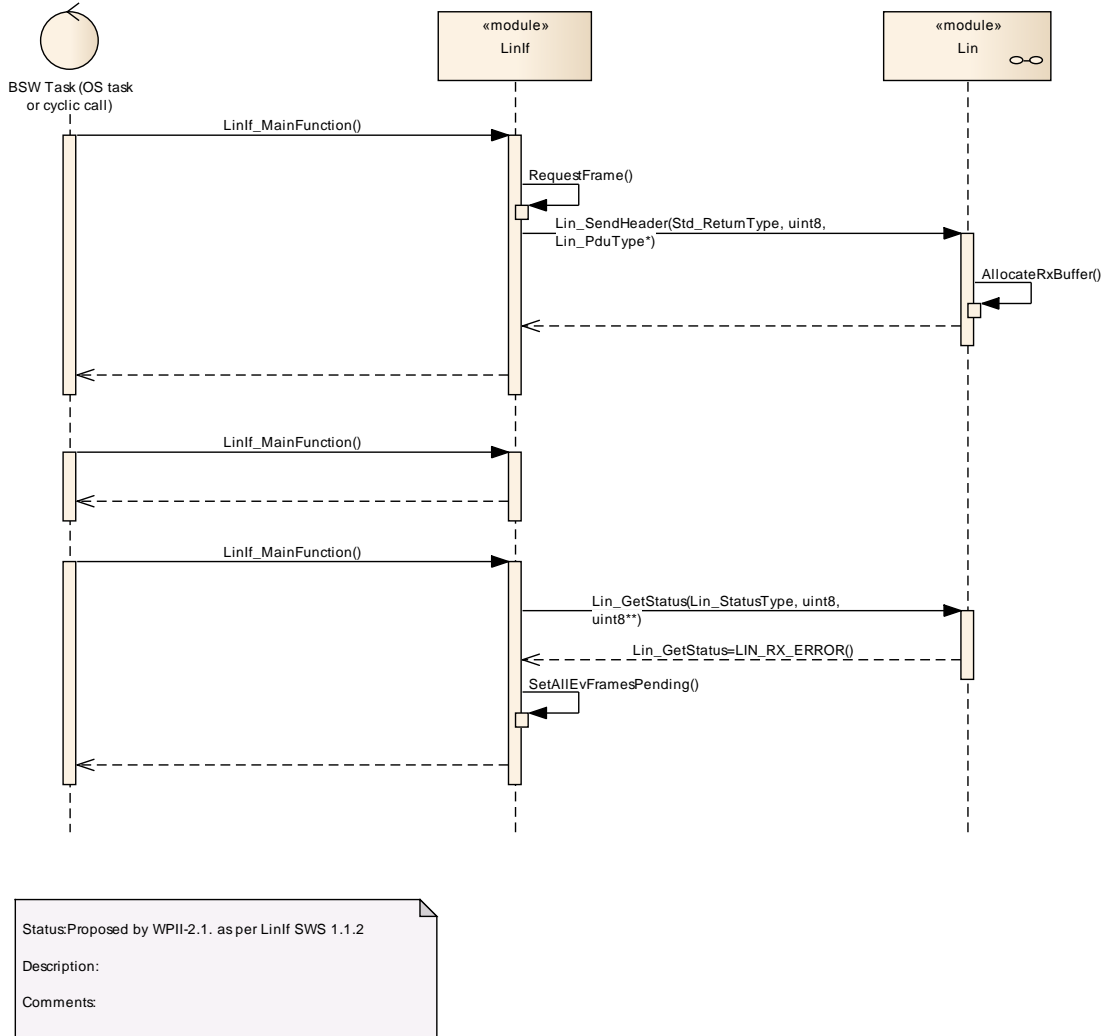


Figure 18 – Event-triggered frame with collision

9.7 Transport Protocol message transmission

The following diagram Figure 19 shows the transmission of a TP message. Both the initiation of the message and the continuous buffer request are shown. The actual transmission of the MRF is not shown in the diagram and it has the same behavior as frame transmission.

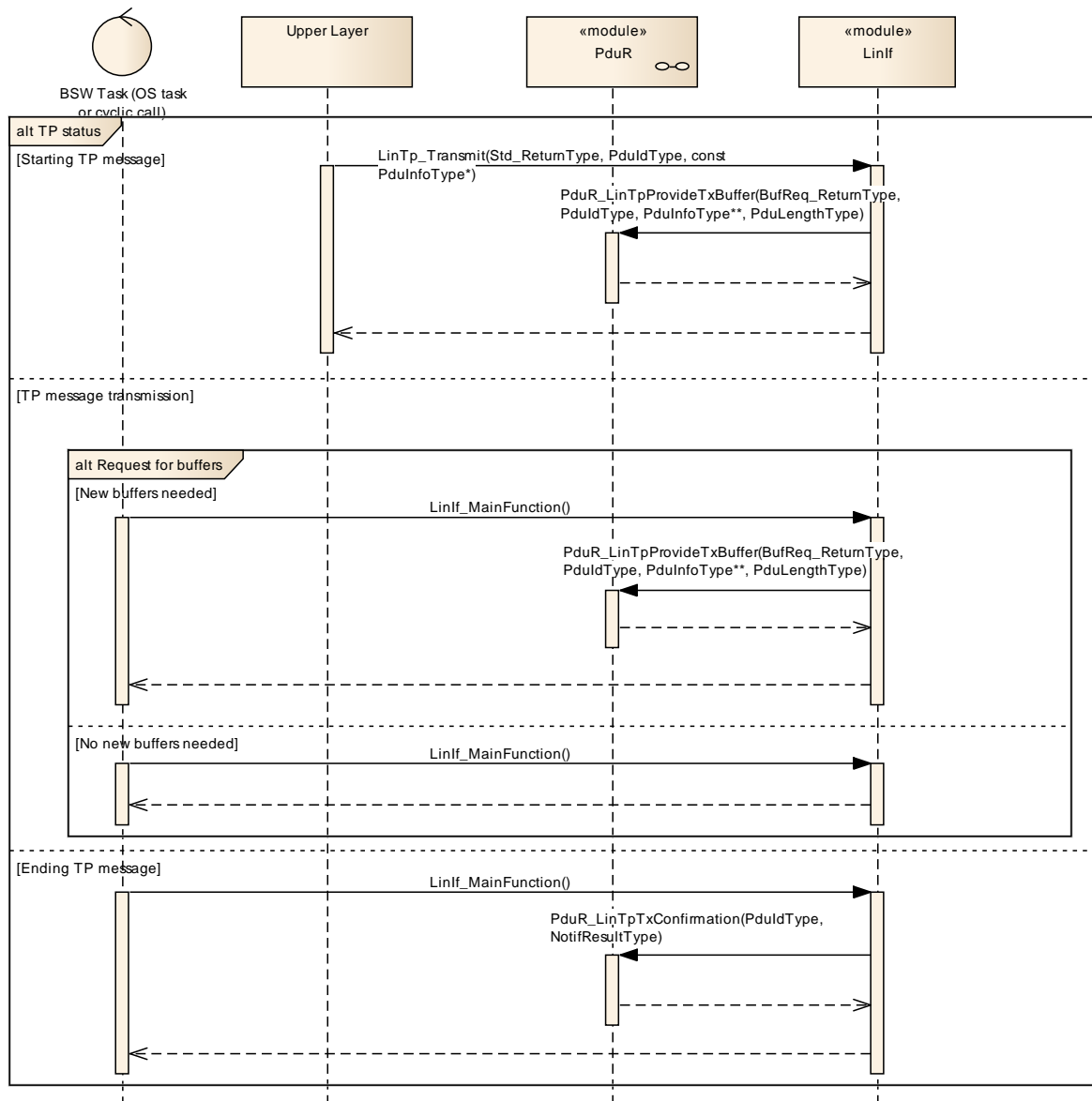


Figure 19 – Transport Protocol message transmission

9.8 Transport Protocol message reception

The following diagram Figure 20 shows the reception of a TP message. Both the initiation of the message, the continuous buffer request and the finish of the message are shown. The actual reception of the SRF is not shown in the diagram and it has the same behavior as frame reception.

The TP message start is always initiated by receiving a SF or FF from the LIN Driver. In addition, if a SF or FF is received when there is an ongoing reception, a new TP message reception is initiated.

The continuous reception of the message is made by either just copying the N-SDU from the SRF to the provided buffer (TP message reception in the diagram) or if the buffer is not big enough a new buffer must be provided before the copying (TP reception message start/buffer request in the diagram).

The TP message is finished after the last N-PDU (SF or CF) is received. The PDU Router will be notified of the reception of the complete message. Note that the trivial case where more buffer space must be provided in the last part of the TP message is not shown. A new buffer must then be requested before finishing the TP message.

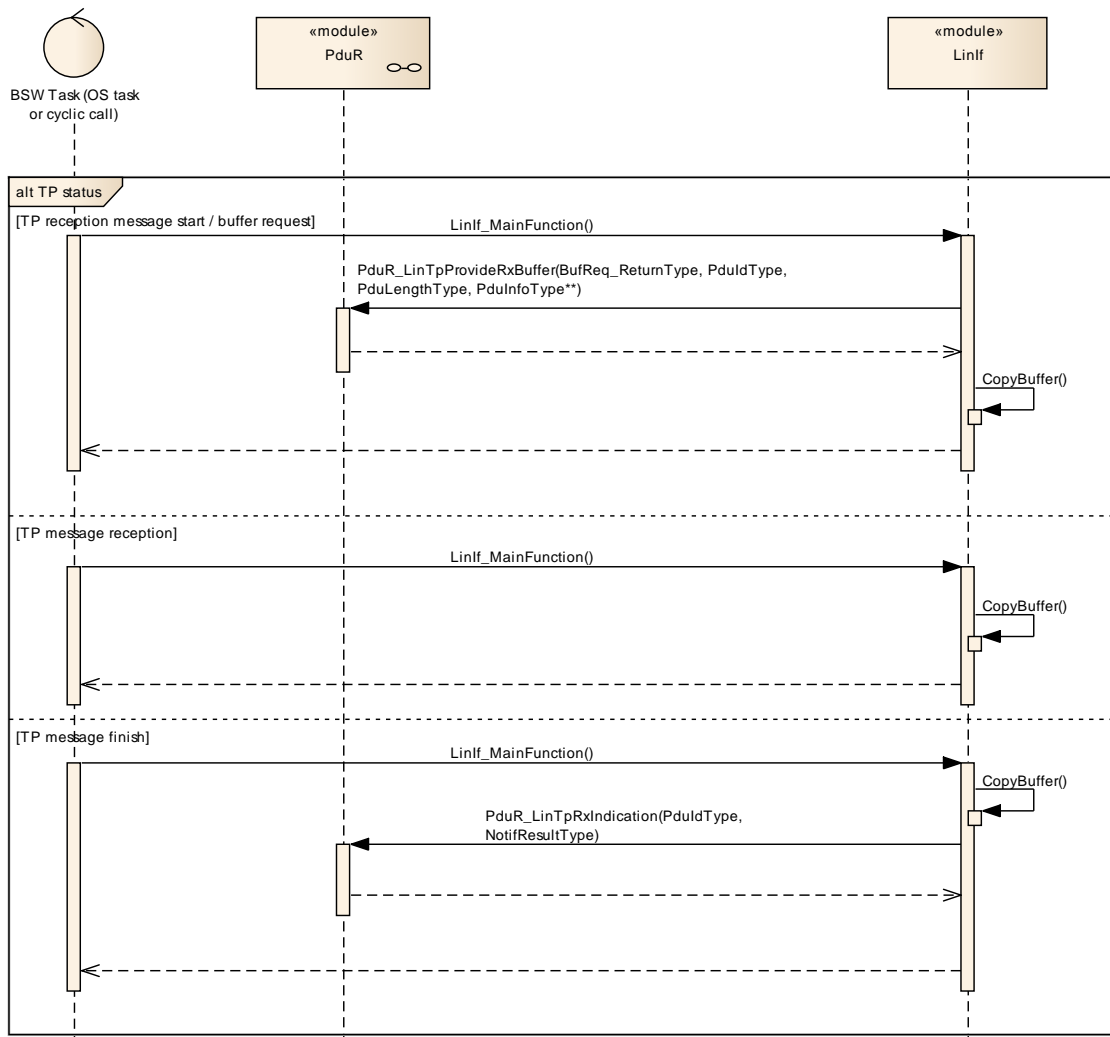


Figure 20 – Transport Protocol message reception

9.9 Go to sleep process

This use case in Figure 21 shows the execution of the `LinIf_GotoSleep` command.

The LinIf_MainFunction that is executed subsequent to the LinIf_GotoSleep call is to show that the go-to-sleep command is not executed immediately. The go-to-sleep command is transmitted when the next schedule entry is due.

Note that the LIN Interface sets the state to sleep even if the status is failure.

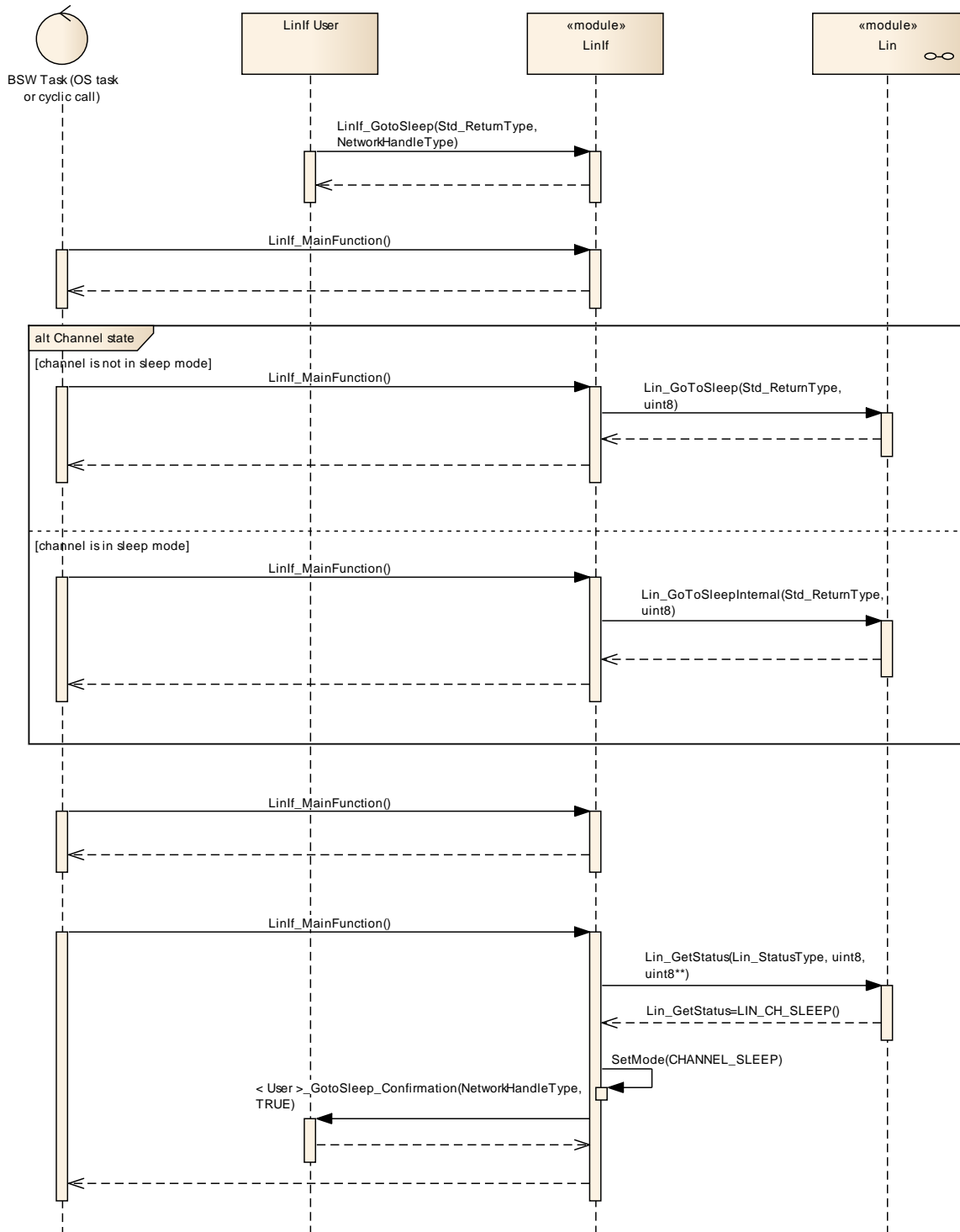


Figure 21 – Go-to-sleep command process

9.10 Wake up request

The wake-up use cases are described in chapter 9 of the AUTOSAR Specification of the ECU State Manager [10].

9.11 Internal wake-up

There are two different use cases in Figure 22:

1. The first shows when the upper layer request wake-up of the LIN cluster AND the cluster is in sleep.
2. The second shows when the upper layer request wake-up of the LIN cluster AND the cluster is awake.

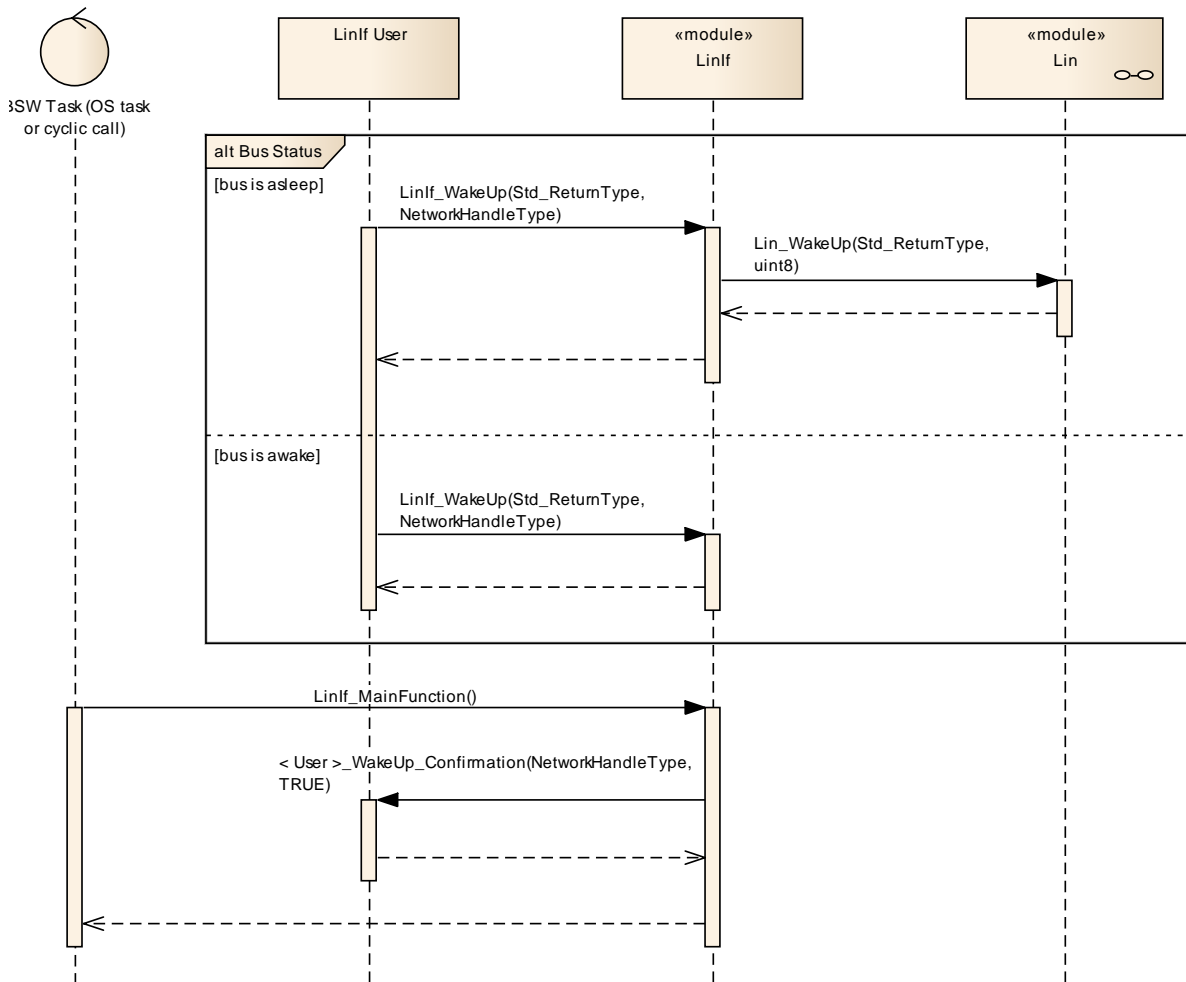


Figure 22 – Internal wake-up

10 Configuration specification

In general, this chapter defines configuration parameters and their clustering into containers.

The chapter 10.3 specifies the structure (containers) and the parameters of the module LIN Interface. The chapter 0 specifies published information of the module LIN TP.

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 meta-model 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: pre 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.

10.1.2 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.

10.1.3 Specification template for configuration parameters

The following tables consist of three sections:

- the general section
- the configuration parameter section

- the section of included/referenced containers

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

<i>Label</i>	<i>Description</i>
x	The configuration parameter shall be of configuration class <i>Pre-compile time</i> .
--	The configuration parameter shall never be of configuration class <i>Pre-compile time</i> .

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

<i>Label</i>	<i>Description</i>
x	The configuration parameter shall be of configuration class <i>Link time</i> .
--	The configuration parameter shall never be of configuration class <i>Link time</i> .

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

<i>Label</i>	<i>Description</i>
x	The configuration parameter shall be of configuration class <i>Post Build</i> and no specific implementation is required.
L	<i>Loadable</i> – the configuration parameter shall be of configuration class <i>Post Build</i> and only one configuration parameter set resides in the ECU.
M	<i>Multiple</i> – the configuration parameter shall be of configuration class <i>Post Build</i> 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 <i>Post Build</i> .

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.

LINIF374: For post-build time support, the LIN Interface configuration structure `LinIf_Configuration` shall be constructed so that it may be exchangeable in memory.

Example: The `LinIf_Configuration` is placed in a specific Flash sector. This flash sector may be reflashed after the ECU is placed in the vehicle.

10.2.1 Configuration Tool

A configuration tool will create a configuration structure that is understood by the LIN Interface.

The philosophy of the LIN 2.0 specification is that a LIN cluster is static. Therefore, many relations and behavior may be checked before the configuration is given to the LIN Interface. To avoid time consuming checking in the LIN Interface it is possible to do lots of checking offline.

LINIF375: The LIN Interface shall not make any consistency check of the configuration in run-time in production software. It may be done if the development error detection is enabled.

10.2.2 Variants

Three configuration variants are defined for LIN Interface.

10.2.2.1 Variant1 (Pre-compile Configuration)

LINIF491: 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.

10.2.2.2 Variant2 (Link-time Configuration)

LINIF492: This configuration includes all configuration options of the “Pre-compile Configuration”. Additionally all parameters defined below, as link-time configurable shall be configurable at link time for example by linking a special configured parameter object file.

The module is most likely delivered as object code.

10.2.2.3 Variant3 (Post-build Configuration)

LINIF493: This configuration includes all configuration options of the “Link-time 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.

10.3 LinIf Configuration

The Figure 23 depicts the LIN Interface configuration.

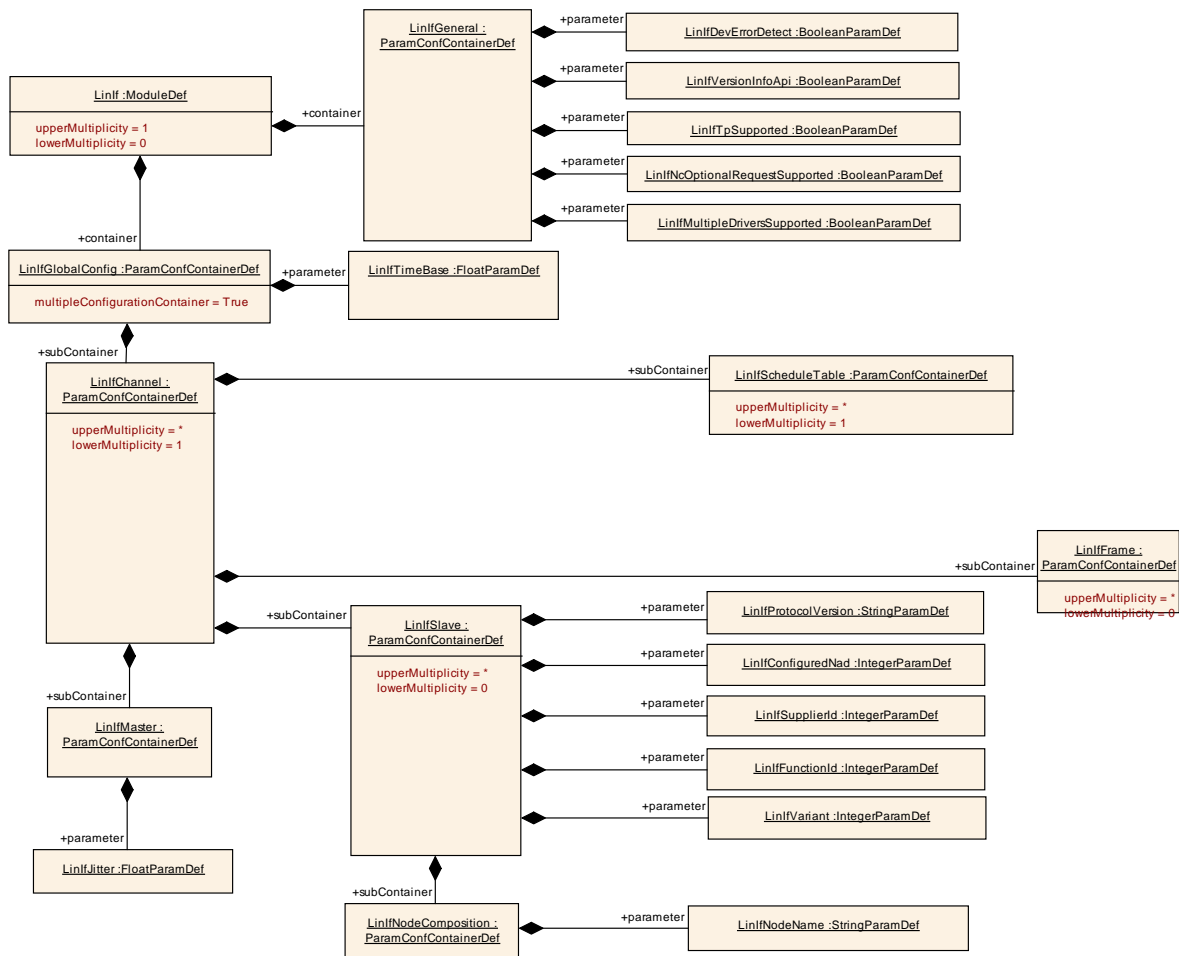


Figure 23 – LIN Interface configuration

10.3.1 LinIf

SWS Item	LINIF370 :
Module Name	<i>LinIf</i>
Module Description	Configuration of the LinIf (LIN Interface) module.

Included Containers		
Container Name	Multiplicity	Scope / Dependency
LinIfGeneral	1	--
LinIfGlobalConfig	1	This container contains the global configuration parameter of the LinIf. It is a MultipleConfigurationContainer, i.e. this container and its sub-containers exist once per configuration set.

10.3.2 LinIfGeneral

SWS Item	ECUC_LinIf_00019 :		
Container Name	LinIfGeneral		
Description	--		
Configuration Parameters			

SWS Item	ECUC_LinIf_00010 :		
Name	LinIfDevErrorDetect {LINIF_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	ECUC_LinIf_00024 :		
Name	LinIfMultipleDriversSupported {LINIF_MULTIPLE_DRIVER_SUPPORT}		
Description	States if multiple drivers are included in the LIN Interface or not. The reason for this parameter is to reduce the size of LIN Interface if multiple drivers are not used.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	ECUC_LinIf_00026 :		
Name	LinIfNcOptionalRequestSupported {LINIF_OPTIONAL_REQUEST_SUPPORTED}		
Description	States if the node configuration commands Assign NAD and Conditional Change NAD are supported.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	ECUC_LinIf_00045 :		
Name	LinIfTpSupported {LINIF_TP_SUPPORTED}		
Description	States if the TP is included in the LIN Interface or not. The reason for this parameter is to reduce the size of LIN Interface if the TP is not used.		

Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	ECUC_Linlf_00053 :		
Name	LinlfVersionInfoApi {LINIF_VERSION_INFO_API}		
Description	Switches the Linlf_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

10.3.3 LinlfGlobalConfig

SWS Item	ECUC_Linlf_00020 :		
Container Name	LinlfGlobalConfig [Multi Config Container]		
Description	This container contains the global configuration parameter of the Linlf. It is a MultipleConfigurationContainer, i.e. this container and its sub-containers exit once per configuration set.		
Configuration Parameters			

SWS Item	ECUC_Linlf_00044 :		
Name	LinlfTimeBase {LINIF_TIME_BASE}		
Description	Defines the interval of calls to main functions in seconds.		
Multiplicity	1		
Type	FloatParamDef		
Range	-INF .. INF		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

Included Containers		
Container Name	Multiplicity	Scope / Dependency
LinlfChannel	1..*	--

10.3.4 LinIfChannel

SWS Item	LINIF364 :
Container Name	LinIfChannel{LinIf_Channel}
Description	--
Configuration Parameters	

SWS Item	ECUC_LinIf_00002 :		
Name	LinIfChannelId		
Description	Internal ID for the channel on LIN Interface level. This parameter shall map the NetworkHandleType to the physical LIN channel. Implementation Type: NetworkHandleType		
Multiplicity	1		
Type	IntegerParamDef (Symbolic Name generated for this parameter)		
Range	..		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency			

SWS Item	ECUC_LinIf_00601 :		
Name	LinIfGotoSleepConfirmationUL		
Description	This parameter defines the upper layer (UL) module to which the confirmation of the goto-sleep command shall be sent.		
Multiplicity	1		
Type	EnumerationParamDef		
Range	CDD	Complex Device Driver	
	LIN_SM	LIN State Manager	
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	ECUC_LinIf_00600 :		
Name	LinIfScheduleRequestConfirmationUL		
Description	This parameter defines the upper layer (UL) module to which the confirmation of the successfully performed schedule table change shall be sent.		
Multiplicity	1		
Type	EnumerationParamDef		
Range	CDD	Complex Device Driver	
	LIN_SM	LIN State Manager	
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	ECUC_LinIf_00643 :		
Name	LinIfScheduleRequestQueueLength {LINIF_SCHEDULE_REQUEST_QUEUE_LENGTH}		
Description	Number of schedule requests the schedule table manager can handle for this channel.		
Multiplicity	1		

Type	IntegerParamDef		
Range	1 .. 255		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	ECUC_LinIf_00602 :		
Name	LinIfWakeupConfirmationUL		
Description	This parameter defines the upper layer (UL) module to which the confirmation of the wake-up shall be sent.		
Multiplicity	1		
Type	EnumerationParamDef		
Range	CDD	Complex Device Driver	
	LIN_SM	LIN State Manager	
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	ECUC_LinIf_00637 :		
Name	LinIfCddRef		
Description	Reference to the CDD module description. This parameter is only required when LinIfWakeupConfirmationUL, LinIfScheduleRequestConfirmationUL, and/or LinIfGotoSleepConfirmationUL is set to CDD.		
Multiplicity	0..1		
Type	Foreign reference to [MODULE-CONFIGURATION]		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	ECUC_LinIf_00003 :		
Name	LinIfChannelRef {LINIF_CHANNEL_INDEX}		
Description	Reference to the used channel in Lin. Replaces LINIF_CHANNEL_INDEX		
Multiplicity	1		
Type	Reference to [LinChannel]		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency			

Included Containers		
Container Name	Multiplicity	Scope / Dependency
LinIfFrame	0..*	Generic container for all types of LIN frames.
LinIfMaster	1	Each Master can only be connected to one physical channel. This could be compared to the Node parameter in a LDF file.
LinIfScheduleTable	1..*	Describes a schedule table. Each LinIfChannel may have

		several schedule tables. Each schedule table can only be connected to one channel.
LinIfSlave	0..*	The Node attributes of the Slaves are provided with these parameter.

10.3.5 LinIfFrame

SWS Item	LINIF367 :		
Container Name	LinIfFrame{LinIf_Frame}		
Description	Generic container for all types of LIN frames.		
Configuration Parameters			

SWS Item	ECUC_LinIf_00005 :		
Name	LinIfChecksumType {LINIF_CHECKSUM_TYPE}		
Description	Type of checksum that the frame is using.		
Multiplicity	1		
Type	EnumerationParamDef		
Range	CLASSIC	Classic	
	ENHANCED	Enhanced	
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	ECUC_LinIf_00639 :		
Name	LinIfFrameName {LINIF_FRAME_NAME}		
Description	Optional frame name used to cross-reference with a LDF		
Multiplicity	0..1		
Type	StringParamDef		
Default value	--		
regularExpression	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	LINIF467 :		
Name	LinIfFramePriority {LINIF_FRAME_PRIORITY}		
Description	Priority of an unconditional frame if used as a sporadic frame or in case of collision resolving of event triggered frames		
Multiplicity	1		
Type	IntegerParamDef		
Range	0 .. 255		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	ECUC_LinIf_00017 :		
Name	LinIfFrameType {LINIF_FRAME_TYPE}		
Description	Type of frame that is described (e.g. sporadic frame). The sporadic slot is not found among the frame types. A sporadic slot is a set of sporadic frames.		
Multiplicity	1		
Type	EnumerationParamDef		
Range	ASSIGN	AssignFrameId	
	ASSIGN_NAD	AssignNAD	
	CONDITIONAL	Conditional Change NAD	
	EVENT_TRIGGERED	Event triggered frame	
	FREE	FreeFormat	
	MRF	MRF	
	SPORADIC	Sporadic frame	
	SRF	SRF	
	UNASSIGN	UnassignFrameId	
	UNCONDITIONAL	Unconditional Frame	
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	ECUC_LinIf_00023 :		
Name	LinIfLength {LINIF_LENGTH}		
Description	Length of the LIN SDU in bytes.		
Multiplicity	1		
Type	IntegerParamDef		
Range	1 .. 8		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	ECUC_LinIf_00028 :		
Name	LinIfPid {LINIF_PID}		
Description	Protected ID of the LIN frame. There is no reason to calculate the Parity in run-time.		
Multiplicity	1		
Type	IntegerParamDef		
Range	0 ..		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	ECUC_LinIf_00644 :		
Name	LinIfTxTargetPduld {LINIF_TARGET_PDU_ID}		
Description	Identifier of the frame for the upper layer		
Multiplicity	1		
Type	IntegerParamDef		
Range	..		
Default value	--		

ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

Included Containers		
Container Name	Multiplicity	Scope / Dependency
LinIfFixedFrameSdu	0..1	In case this is a fixed frame this is the SDU (response). This value should represent an eight byte array. The Byte order shall be MSB first.
LinIfPduDirection	1	Direction of the frame
LinIfSubstitutionFrames	0..*	List of unconditional Frames that can be sent in an event-triggered Frame or a sporadic Frame slot.

10.3.6 LinIfFixedFrameSdu

SWS Item	ECUC_LinIf_00012 :
Container Name	LinIfFixedFrameSdu{LINIF_FIXED_FRAME_SDU}
Description	In case this is a fixed frame this is the SDU (response). This value should represent an eight byte array. The Byte order shall be MSB first.
Configuration Parameters	

Included Containers		
Container Name	Multiplicity	Scope / Dependency
LinIfFixedFrameSduByte	8	This container represents a byte within the 8 byte array.

10.3.7 LinIfFixedFrameSduByte

SWS Item	ECUC_LinIf_00013 :
Container Name	LinIfFixedFrameSduByte{LINIF_FIXED_FRAME_SDU_BYTE}
Description	This container represents a byte within the 8 byte array.
Configuration Parameters	

SWS Item	ECUC_LinIf_00014 :		
Name	LinIfFixedFrameSduBytePos		
Description	Index of the Byte in the SDU (response) 8 byte array.		
Multiplicity	1		
Type	IntegerParamDef		
Range	0 .. 7		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	ECUC_LinIf_00015 :
Name	LinIfFixedFrameSduByteVal

Description	Byte value in the SDU (response) 8-byte array.		
Multiplicity	1		
Type	IntegerParamDef		
Range	0 .. 255		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

No Included Containers

10.3.8 LinIfPduDirection

SWS Item	ECUC_LinIf_00027 :
Choice container Name	LinIfPduDirection{LINIF_DIRECTION}
Description	Direction of the frame

Container Choices		
Container Name	Multiplicity	Scope / Dependency
LinIfInternalPdu	0..1	Represents a Diagnostic or Configuration frame : no Message ID (no Pduld).
LinIfRxPdu	0..1	represents a received PDU/frame
LinIfSlaveToSlavePdu	0..1	represents a slave-to-slave PDU/frame. Master does only send the header but doesn't receive the response.
LinIfTxPdu	0..1	represents a transmitted PDU/frame

10.3.9 LinIfSubstitutionFrames

SWS Item	ECUC_LinIf_00042 :
Container Name	LinIfSubstitutionFrames
Description	List of unconditional Frames that can be sent in an event-triggered Frame or a sporadic Frame slot.
Configuration Parameters	

SWS Item	ECUC_LinIf_00041 :		
Name	LinIfSubstitutionFrameRef		
Description	Reference to an unconditional Frame that can be sent in an event-triggered Frame or a sporadic Frame slot.		
Multiplicity	1		
Type	Reference to [LinIfFrame]		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

No Included Containers

10.3.10 LinIfRxPdu

SWS Item	ECUC_LinIf_00035 :
Container Name	LinIfRxPdu
Description	represents a received PDU/frame
Configuration Parameters	

SWS Item	ECUC_LinIf_00055 :		
Name	LinIfRxIndicationUL		
Description	This parameter defines the name of the <User_RxIndication>. This parameter depends on the parameter LinIfUserRxIndicationUL. If LinIfUserRxIndicationUL equals PDUR, the name of the <User_RxIndication> is fixed. If LinIfUserRxIndicationUL equals CDD, the name of the <User_RxIndication> is selectable.		
Multiplicity	0..1		
Type	FunctionNameDef		
Default value	--		
regularExpression	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	ECUC_LinIf_00641 :		
Name	LinIfRxPduId {LINIF_PDU_ID}		
Description	Identifier of the frame for the LIN Interface		
Multiplicity	1		
Type	IntegerParamDef (Symbolic Name generated for this parameter)		
Range	..		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	ECUC_LinIf_00610 :		
Name	LinIfUserRxIndicationUL		
Description	This parameter defines the upper layer (UL) module to which the indication of the successfully received LINRXPDUID has to be routed via <User_RxIndication>. This <User_RxIndication> has to be invoked when the indication of the configured LINRXPDUID will be received by a Rx indication event from the LIN Driver module.		
Multiplicity	1		
Type	EnumerationParamDef		
Range	CDD	Complex Device Driver	
	PDUR	Pdu Router	
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE

	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	ECUC_Linlf_00036 :		
Name	LinlfRxPduRef		
Description	Reference to the PDU that is received in this frame.		
Multiplicity	1		
Type	Reference to [Pdu]		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

No Included Containers

10.3.11 LinlfTxPdu

SWS Item	ECUC_Linlf_00049 :		
Container Name	LinlfTxPdu		
Description	represents a transmitted PDU/frame		
Configuration Parameters			

SWS Item	ECUC_Linlf_00054 :		
Name	LinlfTxConfirmationUL		
Description	This parameter defines the name of the <User_TxConfirmation>. This parameter depends on the parameter LinlfUserTxUL. If LinlfUserTxUL equals PDUR, the name of the <User_TxConfirmation> is fixed. If LinlfUserTxUL equals CDD, the name of the <User_TxConfirmation> is selectable.		
Multiplicity	0..1		
Type	FunctionNameDef		
Default value	--		
regularExpression	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	ECUC_Linlf_00050 :		
Name	LinlfTxPduId {LINIF_PDU_ID}		
Description	Identifier of the frame for the upper layer. This id is only relevant for sporadic frames.		
Multiplicity	1		
Type	IntegerParamDef (Symbolic Name generated for this parameter)		
Range	..		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time		

	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item ECUC_Linlf_00628 :			
Name LinlfTxTriggerTransmitUL			
Description This parameter defines the name of the <User_TriggerTransmit>. This parameter depends on the parameter LinlfUserTxUL. If LinlfUserTxUL equals PDUR, the name of the <User_TriggerTransmit> is fixed. If LinlfUserTxUL equals CDD, the name of the <User_TriggerTransmit> is selectable.			
Multiplicity 0..1			
Type FunctionNameDef			
Default value --			
regularExpression --			
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency scope: ECU			

SWS Item ECUC_Linlf_00609 :			
Name LinlfUserTxUL			
Description This parameter defines the upper layer (UL) module to which the trigger of the transmitted LinTxPdu (via the <User_TriggerTransmit>) or the confirmation of the successfully transmitted LinTxPdu has to be routed (via the <User_TxConfirmation>).			
Multiplicity 1			
Type EnumerationParamDef			
Range	CDD	Complex Device Driver	
	PDUR	Pdu Router	
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency scope: ECU			

SWS Item ECUC_Linlf_00051 :			
Name LinlfTxPduRef			
Description Reference to the PDU that is transmitted in this frame.			
Multiplicity 1			
Type Reference to [Pdu]			
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

No Included Containers

10.3.12 LinIfScheduleTable

SWS Item	LINIF365 :
Container Name	LinIfScheduleTable{LinIf_ScheduleTable}
Description	Describes a schedule table. Each LinIfChannel may have several schedule tables. Each schedule table can only be connected to one channel.
Configuration Parameters	

SWS Item	ECUC_LinIf_00033 :		
Name	LinIfResumePosition		
Description	Defines, where a schedule table shall be proceeded in case it has been interrupted by a RUN_ONCE table or MRF/SRF.		
Multiplicity	1		
Type	EnumerationParamDef		
Range	CONTINUE_AT_IT_POINT	Continue schedule table where it was interrupted	
	START_FROM_BEGINNING	Start schedule table from beginning	
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	ECUC_LinIf_00034 :		
Name	LinIfRunMode {LINIF_RUN_MODE}		
Description	The schedule table can be executed in two different modes.		
Multiplicity	1		
Type	EnumerationParamDef		
Range	RUN_CONTINUOUS		
	RUN_ONCE		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	ECUC_LinIf_00642 :		
Name	LinIfSchedulePriority {LINIF_SCHEDULE_PRIORITY}		
Description	Priority of the schedule table. The priority is used in the schedule table manager. The RUN_ONCE run mode schedules shall not have equal priority. 0 Reserved for NULL_SCHEDULE 1..254 Only for RUN_ONCE 255 Only RUN_CONTINUOUS		
Multiplicity	1		
Type	IntegerParamDef		
Range	0 .. 255		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	ECUC_LinIf_00037 :		
Name	LinIfScheduleTableIndex {LINIF_SCHEDULE_INDEX}		
Description	This is the unique index used by upper layers to identify a schedule. Note that the NULL_SCHEDULE for each channel has index 0.		

Multiplicity	1		
Type	IntegerParamDef (Symbolic Name generated for this parameter)		
Range	..		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	ECUC_LinIf_00038 :		
Name	LinIfScheduleTableName {LINIF_SCHEDULE_NAME}		
Description	Optional schedule name used to cross-reference with a LDF. This parameter shall always be accompanied by LINIF_SCHEDULE_INDEX		
Multiplicity	0..1		
Type	StringParamDef		
Default value	--		
regularExpression	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

Included Containers		
Container Name	Multiplicity	Scope / Dependency
LinIfEntry	0..*	Describes an entry in the schedule table (also known as Frame Slot).

10.3.13 LinIfEntry

SWS Item	LINIF366 :	
Container Name	LinIfEntry{LinIf_Entry}	
Description	Describes an entry in the schedule table (also known as Frame Slot).	
Configuration Parameters		

SWS Item	ECUC_LinIf_00009 :		
Name	LinIfDelay {LINIF_DELAY}		
Description	Delay to next frame in schedule table in [s]		
Multiplicity	1		
Type	FloatParamDef		
Range	-INF .. INF		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	ECUC_LinIf_00011 :		
Name	LinIfEntryIndex		

Description	Position of the Frame Entry in the Schedule Table.		
Multiplicity	1		
Type	IntegerParamDef		
Range	..		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	ECUC_LinIf_00016 :		
Name	LinIfFrameRef		
Description	Reference to the frames that belong to this schedule table entry.		
Multiplicity	1..*		
Type	Reference to [LinIfFrame]		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

No Included Containers

10.3.14 LinIfMaster

SWS Item	LINIF504 :
Container Name	LinIfMaster
Description	Each Master can only be connected to one physical channel. This could be compared to the Node parameter in a LDF file.
Configuration Parameters	

SWS Item	ECUC_LinIf_00629 :		
Name	LinIfJitter		
Description	The jitter specifies the differences between the maximum and minimum delay from time base tick to the header sending start point in seconds.		
Multiplicity	1		
Type	FloatParamDef		
Range	-INF .. INF		
Default value	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

No Included Containers

10.3.15 LinIfSlave

SWS Item	ECUC_LinIf_00039 :
Container Name	LinIfSlave
Description	The Node attributes of the Slaves are provided with these parameter.
Configuration Parameters	

SWS Item	ECUC_LinIf_00008 :		
Name	LinIfConfiguredNad {LINIF_CONFIGURED_NAD}		
Description	Definition of the initial node address		
Multiplicity	1		
Type	IntegerParamDef		
Range	..		
Default value	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	ECUC_LinIf_00018 :		
Name	LinIfFunctionId {LINIF_FUNCTION_ID}		
Description	LIN function ID		
Multiplicity	1		
Type	IntegerParamDef		
Range	..		
Default value	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	ECUC_LinIf_00029 :		
Name	LinIfProtocolVersion {LINIF_PROTOCOL_VERSION}		
Description	Defines the LIN Protocol version which is used by the slave.		
Multiplicity	1		
Type	StringParamDef		
Default value	--		
regularExpression	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	ECUC_LinIf_00043 :		
Name	LinIfSupplierId {LINIF_SUPPLIER_ID}		
Description	LIN Supplier ID		
Multiplicity	1		
Type	IntegerParamDef		
Range	..		
Default value	--		
ConfigurationClass	Pre-compile time	X	All Variants

	<i>Link time</i>	--	
	<i>Post-build time</i>	--	
Scope / Dependency			

SWS Item	ECUC_LinIf_00052 :		
Name	LinIfVariant {LINIF_VARIANT}		
Description	Specifies the Variant ID		
Multiplicity	1		
Type	IntegerParamDef		
Range	..		
Default value	--		
ConfigurationClass	<i>Pre-compile time</i>	X	All Variants
	<i>Link time</i>	--	
	<i>Post-build time</i>	--	
Scope / Dependency			

Included Containers		
Container Name	Multiplicity	Scope / Dependency
LinIfNodeComposition	1	Generic container that describes the node composition

10.3.16 LinIfNodeComposition

SWS Item	ECUC_LinIf_00638 :		
Container Name	LinIfNodeComposition{LinIf_NodeComposition}		
Description	Generic container that describes the node composition		
Configuration Parameters			

SWS Item	ECUC_LinIf_00640 :		
Name	LinIfNodeName {LINIF_NODE_NAME}		
Description	--		
Multiplicity	1		
Type	StringParamDef		
Default value	--		
regularExpression	--		
ConfigurationClass	<i>Pre-compile time</i>	X	All Variants
	<i>Link time</i>	--	
	<i>Post-build time</i>	--	
Scope / Dependency			

No Included Containers		
-------------------------------	--	--

10.3.17 LinIfSlaveToSlavePdu

SWS Item	ECUC_LinIf_00040 :		
Container Name	LinIfSlaveToSlavePdu		

Description	represents a slave-to-slave PDU/frame. Master does only send the header but doesn't receive the response.
Configuration Parameters	

No Included Containers

10.3.18 LinIfInternalPdu

SWS Item	ECUC_LinIf_00021 :
Container Name	LinIfInternalPdu
Description	Represents a Diagnostic or Configuration frame : no Message ID (no PdulId).
Configuration Parameters	

No Included Containers

10.4 LIN Transport Layer configuration

The Figure 24 shows the outline of the LIN Transport Protocol configuration.

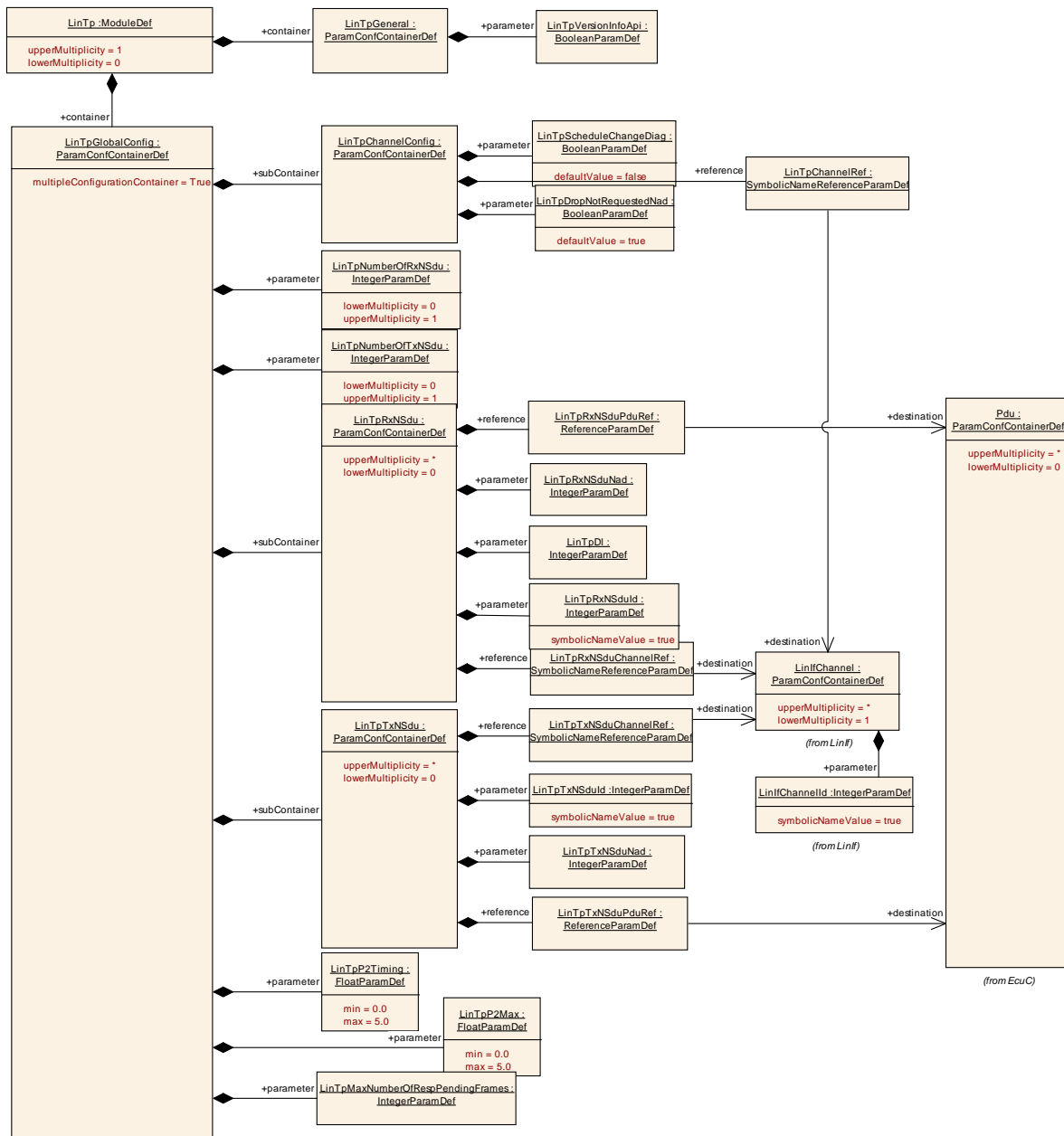


Figure 24 – LIN Transport Protocol configuration

10.4.1 LinTp

SWS Item	LINIF425 :
Module Name	<i>LinTp</i>
Module Description	Singleton descriptor for the LIN Transport Protocol.

Included Containers		
Container Name	Multiplicity	Scope / Dependency
LinTpGeneral	1	Container that holds all LIN transport protocol general parameters.

LinTpGlobalConfig	1	This container contains the global configuration parameter of the LinTp. It is a MultipleConfigurationContainer, i.e. this container and its sub-containers exit once per configuration set.
-------------------	---	--

10.4.2 LinTpGeneral

SWS Item	ECUC_LinTp_00617 :		
Container Name	LinTpGeneral		
Description	Container that holds all LIN transport protocol general parameters.		
Configuration Parameters			

SWS Item	ECUC_LinTp_00068 :		
Name	LinTpVersionInfoApi {LINTP_VERSION_INFO_API}		
Description	Switches the LinTp_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

10.4.3 LinTpGlobalConfig

SWS Item	ECUC_LinTp_00056 :		
Container Name	LinTpGlobalConfig [Multi Config Container]		
Description	This container contains the global configuration parameter of the LinTp. It is a MultipleConfigurationContainer, i.e. this container and its sub-containers exit once per configuration set.		
Configuration Parameters			

SWS Item	LINIF624 :		
Name	LinTpMaxNumberOfRespPendingFrames {LINTP_MAX_NUMBER_OF_RESP_PENDING_FRAMES}		
Description	Configures the maximum number of allowed response pending frames.		
Multiplicity	1		
Type	IntegerParamDef		
Range	..		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope	/scope: Module		

Dependency	
-------------------	--

SWS Item	ECUC_LinTp_00057 :		
Name	LinTpNumberOfRxNSdu {LINTP_NUMBER_OF_RX_NSDU}		
Description	Number of transport protocol messages that can be received for all channels this node is connected to. Please note that this parameter is deprecated and will be removed in the next major version. The value of this parameter can be calculated from the multiplicity of the container LinTpRxNSdu.		
Multiplicity	0..1		
Type	IntegerParamDef		
Range	..		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	ECUC_LinTp_00058 :		
Name	LinTpNumberOfTxNSdu {LINTP_NUMBER_OF_TX_NSDU}		
Description	Number of transport protocol messages that can be transmitted for all channels this node is connected to. Please note that this parameter is deprecated and will be removed in the next major version. The value of this parameter can be calculated from the multiplicity of the container LinTpTxNSdu.		
Multiplicity	0..1		
Type	IntegerParamDef		
Range	..		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	LINIF622 :		
Name	LinTpP2Max {LINTP_P2_MAX}		
Description	P2 Timeout when a response pending frame is expected in seconds.		
Multiplicity	1		
Type	FloatParamDef		
Range	0 .. 5		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	LINIF625 :		
Name	LinTpP2Timing {LINTP_P2_TIMING}		
Description	Definition of the P2 timeout observation parameter in seconds.		
Multiplicity	1		
Type	FloatParamDef		
Range	0 .. 5		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD

Scope / Dependency	scope: Module
---------------------------	---------------

Included Containers		
Container Name	Multiplicity	Scope / Dependency
LinTpChannelConfig	1	This container contains the channel specific configuration parameter of LinTp.
LinTpRxNSdu	0..*	For each received N-SDU on any channel the node is connected to.
LinTpTxNSdu	0..*	For each transmitted N-SDU on any channel the node is connected to.

10.4.4 LinTpChannelConfig

SWS Item	LINIF071 :
Container Name	LinTpChannelConfig{LINTP_CHANNEL_CONFIG}
Description	This container contains the channel specific configuration parameter of LinTp.
Configuration Parameters	

SWS Item	LINIF072 :		
Name	LinTpDropNotRequestedNad {LINTP_DROP_NOT_REQUESTED_NAD}		
Description	Configures if TP Frames of not requested LIN-Slaves are dropped or not. TRUE: Drop TP frames of not requested LIN-Slaves. FALSE: Keep TP frames of not requested LIN-Slaves.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	true		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	LINIF070 :		
Name	LinTpScheduleChangeDiag {LINTP_SCHEDULE_CHANGE_DIAG}		
Description	Enables or disables the call of BswM_LinTp_RequestMode() to diagnostic request/response schedule.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	false		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	LINIF073 Conf :		
Name	LinTpChannelRef		
Description	Index of the channel this LinTp channel belongs to.		

Multiplicity	1		
Type	Reference to [LinIfChannel]		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

No Included Containers

10.4.5 LinTpRxNSdu

SWS Item	LINIF428 :		
Container Name	LinTpRxNSdu{LinTp_rx_NSdu}		
Description	For each received N-SDU on any channel the node is connected to.		
Configuration Parameters			

SWS Item	ECUC_LinTp_00616 :		
Name	LinTpDI {LINTP_DL}		
Description	Data Length Code of this RxNSdu. In case of variable length message, this value indicates the minimum data length. Range of minimum length is 1 to 4095. Note that this is not relevant for Tx. The reason for this is to have identical structures for Tx and Rx.		
Multiplicity	1		
Type	IntegerParamDef		
Range	..		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	ECUC_LinTp_00061 :		
Name	LinTpRxNSduId {LINTP_NSdu_ID}		
Description	The identifier of the Transport Protocol message. This ID will be the one that is communicated with upper layers.		
Multiplicity	1		
Type	IntegerParamDef (Symbolic Name generated for this parameter)		
Range	..		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	ECUC_LinTp_00062 :		
Name	LinTpRxNSduNad {LINTP_NAD}		
Description	A N-SDU transported on LIN is identified using the NAD for the specific slave.		
Multiplicity	1		
Type	IntegerParamDef		

Range	..		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	ECUC_LinTp_00060 :		
Name	LinTpRxNSduChannelRef {LINTP_CHANNEL_INDEX}		
Description	Index of the channel this N-SDU belongs to.		
Multiplicity	1		
Type	Reference to [LinIfChannel]		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	ECUC_LinTp_00063 :		
Name	LinTpRxNSduPduRef		
Description	Reference to the global PDU		
Multiplicity	1		
Type	Reference to [Pdu]		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

No Included Containers

10.4.6 LinTpTxNSdu

SWS Item	ECUC_LinTp_00511 :		
Container Name	LinTpTxNSdu{LinTp_tx_NSdu}		
Description	For each transmitted N-SDU on any channel the node is connected to.		
Configuration Parameters			

SWS Item	ECUC_LinTp_00065 :		
Name	LinTpTxNSduId {LINTP_NSdu_ID}		
Description	The identifier of the Transport Protocol message. This ID will be the one that is communicated with upper layers.		
Multiplicity	1		
Type	IntegerParamDef (Symbolic Name generated for this parameter)		
Range	..		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD

Scope / Dependency	
---------------------------	--

SWS Item	ECUC_LinTp_00066 :		
Name	LinTpTxNSduNad {LINTP_NAD}		
Description	A N-SDU transported on LIN is identified using the NAD for the specific slave.		
Multiplicity	1		
Type	IntegerParamDef		
Range	..		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	ECUC_LinTp_00064 :		
Name	LinTpTxNSduChannelRef {LINTP_CHANNEL_INDEX}		
Description	Index of the channel this N-SDU belongs to.		
Multiplicity	1		
Type	Reference to [LinIfChannel]		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	ECUC_LinTp_00067 :		
Name	LinTpTxNSduPduRef		
Description	Reference to the global PDU		
Multiplicity	1		
Type	Reference to [Pdu]		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

No Included Containers

10.5 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 [13] Figure 4.1 and Figure 7.1).

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