Document Title	Specification of LIN Interface
Document Owner	AUTOSAR GbR
Document Responsibility	AUTOSAR GbR
Document Version	1.1.0
Document Status	Draft
Part of Release	2.1
Revision	0014

Document Change History

Date	Version	Changed by	Change Description
31.01.2007	1.1.0	AUTOSAR Administration	• 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

Release Notes

Compatibility considerations with respect to current release

See below remarks with respect to Communication Manager and LIN Interface Module.

Errata and known deficiencies

The LIN Interface specification is in draft status and is subject to change.
The known problems include but are not limited to:

- LIN bus specific state management is neither supported by the LIN Interface nor by the Communication Manager
- LIN schedule table switching is not supported
- LIN bus specific error handling is not supported
- LIN Interface frame configuration is ambiguous

The wakeup concept is currently neither harmonized nor consistent throughout all wakeup related specification documents and therefore subject to change.

Known and potential problems resulting from known deficiencies

Currently network communication via the LIN Interface does not work without proprietary implementation extensions to address protocol specific states and frames.
Only one LIN schedule table per physical bus is supported at this time.
LIN error handling is not completely covered.

Due to the fact that the wakeup concept is not harmonized, inconsistent assumptions may lead to:

- the duplication of functionalities across multiple modules
- proprietary implementation extensions
- difficulties during integration

Changes planned for next release

The following changes are planned for the next release:

- AUTOSAR confidential -

The general distribution of functionality in the COM stack is subject to change for the next release. All bus specific state management will be performed by new “bus state manager” functionality, which is located between COM manager and bus interfaces.
The following functionality of the LIN stack will be moved to the LIN bus state manager functionality:

- LIN bus specific error handling
- Control of the actual LIN bus states
- Bidirectional abstraction of communication modes to bus states
- Initialization sequence will be adapted to the revised architecture.
- More changes may be necessary as specification work is not complete yet

Further a rework of the LIN Interface frame configuration will be necessary to have it unambiguous.

The harmonized wakeup concept throughout all wakeup related documents will result in

- adapted specification texts in Chapter 7 of the specification documents
- adapted APIs in Chapter 8 of the specification documents
- adapted wakeup sequences in Chapter 9 of the specification documents

Disclaimer

Any use of these specifications requires membership within the AUTOSAR Development Partnership or an agreement with the AUTOSAR Development Partnership. The AUTOSAR Development Partnership will not be liable for any use of these specifications.

Following the completion of the development of the AUTOSAR specifications commercial exploitation licenses will be made available to end users by way of written License Agreement only.

No part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from the publisher.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Copyright © 2004-2006 AUTOSAR Development Partnership. All rights reserved.

Advice to users of AUTOSAR Specification Documents:

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

Release Notes 2
Compatibility considerations with respect to current release 2
Errata and known deficiencies 2
Known and potential problems resulting from known deficiencies 2
Changes planned for next release 2
1 Introduction and functional overview 9
1.1 Architectural overview 9
1.2 Functional overview 10
2 Acronyms and abbreviations 11
3 Related documentation 12
3.1 Input documents 12
3.2 Related standards and norms 13
4 Constraints and assumptions 14
4.1 Limitations 14
4.2 Applicability to car domains 14
5 Dependencies to other modules 15
5.1 Upper layers 15
5.1.1 PDU Router 15
5.1.2 BSW Scheduler 15
5.1.3 Operating System 15
5.1.4 Module DET (Development Error Tracer) 15
5.1.5 Module DEM (Diagnostic Event Manager) 16
5.1.6 Module ECU State Manager 16
5.1.7 Module Com Manager 16
5.2 Lower layers 16
5.2.1 LIN Driver 16
5.3 File structure 17
5.3.1 Code file structure 17
5.3.2 Header file structure 17
6 Requirements traceability 19
7 Functional specification 24
7.1 Frame Transfer 24
7.1.1 Frame types 24
7.1.1.1 Unconditional frame 25
7.1.1.2 Event triggered frame 25
7.1.1.3 Sporadic frame 25
7.1.1.4 Diagnostic Frames MRF and SRF 26
7.1.1.5 User-defined frames 27
7.1.1.6 Reserved frames 27

7.1.2 Frame reception ... 27
7.1.2.1 Header ... 27
7.1.2.2 Response ... 27
7.1.2.3 Status check ... 27
7.1.3 Frame transmission ... 29
7.1.3.1 Header ... 29
7.1.3.2 Response ... 29
7.1.3.3 Status check ... 30
7.1.4 Slave-to-slave communication ... 30
7.1.4.1 Header ... 30
7.1.4.2 Response ... 30
7.1.4.3 Status check ... 30
7.2 Schedules ... 31
7.2.1 LinIf_MainFunction ... 31
7.2.2 Schedule table manager ... 32
7.3 Node management ... 35
7.3.1 Node Management ... 35
7.3.1.1 LINIF_UNINIT ... 36
7.3.1.2 LINIF_INIT ... 36
7.3.1.3 CHANNEL_UNINIT ... 37
7.3.1.4 CHANNEL_OPERATIONAL ... 37
7.3.1.5 CHANNEL_SLEEP ... 37
7.3.2 Initialization process ... 37
7.3.2.1 LinIf_Init process ... 37
7.3.3 Go to sleep process ... 38
7.3.4 Wake up process ... 38
7.3.4.1 Wakeup during sleep transition ... 39
7.4 Status Management ... 40
7.5 Diagnostics and Node configuration ... 41
7.5.1 Node configuration ... 41
7.5.1.1 Node Model ... 41
7.5.1.2 Node Configuration services ... 42
7.5.1.3 Node Configuration API ... 42
7.5.1.4 Node Configuration in Schedule Table ... 42
7.5.2 Diagnostics – Transport Protocol ... 43
7.5.2.1 LIN TP initialization ... 44
7.5.2.2 LIN TP shut down ... 44
7.5.2.3 State-machine ... 45
7.5.2.4 Buffer handling ... 46
7.5.2.5 TP Transmission ... 47
7.5.2.6 TP transmission error ... 50
7.5.2.7 TP Reception ... 50
7.5.2.8 TP reception error ... 50
7.5.2.9 Unavailability of receive buffer ... 51
7.6 Handling multiple channels and drivers ... 52
7.6.1 Multiple channels ... 52
7.6.2 Multiple LIN drivers ... 52
7.7 Error classification ... 52

7.8 Error detection ... 53
7.9 Error notification ... 54

8 API specification ... 55

8.1 Imported types ... 55
8.1.1 Standard types ... 55
8.2 Type definitions ... 55
8.2.1 LinIf_ChannelIndexType ... 55
8.2.2 LinIf_SchHandleType ... 55
8.2.3 LinTp_ConfigType ... 56
8.3 LIN Interface API ... 56
8.3.1 LinIf_Init ... 56
8.3.2 LinIf_GetVersionInfo ... 57
8.3.3 LinIf_Transmit ... 57
8.3.4 LinIf_ScheduleRequest ... 58
8.3.5 LinIf_GotoSleep ... 59
8.3.6 LinIf_WakeUp ... 60
8.3.7 LinTp_Init ... 60
8.3.8 LinTp_Transmit ... 61
8.3.9 LinTp_ GetVersionInfo ... 62
8.3.10 LinTp_Shutdown ... 63
8.4 Call-back notifications ... 63
8.4.1 LinIf_WakeUpNotification ... 63
8.4.2 LinIf_WakeUpTrcvNotification ... 64
8.5 Scheduled functions ... 64
8.5.1.1 LinIf_MainFunction ... 65
8.6 Expected Interfaces ... 65
8.6.1 Mandatory Interfaces ... 65
8.6.2 Optional interfaces ... 66
8.6.3 Configurable interfaces ... 66

9 Sequence diagrams ... 67

9.1 Frame Transmission ... 67
9.2 Frame Reception ... 69
9.3 Slave to slave communication ... 70
9.4 Reception and transmission at the same time ... 71
9.5 Sporadic frame ... 72
9.6 Event triggered frame ... 73
9.6.1 With no answer ... 73
9.6.2 With answer (No collision) ... 74
9.6.3 With collision ... 75
9.7 Transport Protocol Message transmission ... 77
9.8 Transport Protocol message reception ... 78
9.9 Go-to-sleep process ... 80
9.10 Wake up request ... 81
9.11 Internal wake up ... 82

10 Configuration specification ... 83

- AUTOSAR confidential -

10.1 How to read this chapter ... 83
10.1.1 Configuration and configuration parameters ... 83
10.1.2 Containers ... 83
10.1.3 Specification template for configuration parameters ... 84
10.2 Containers and configuration parameters ... 84
10.2.1 Configuration Tool ... 85
10.2.2 Variants ... 85
10.2.2.1 Variant1 (Pre-compile Configuration) ... 85
10.2.2.2 Variant2 (Link-time Configuration) ... 85
10.2.2.3 Variant3 (Post-build Configuration) ... 85
10.3 LinIf_Configuration ... 86
10.3.1 LinIf_Channel ... 89
10.3.2 LinIf_ScheduleTable ... 91
10.3.3 LinIf_Entry ... 92
10.3.4 LinIf_Frame ... 93
10.4 LIN Transport Layer configuration ... 95
10.4.1 LinTp_xx_NSDU ... 97
10.5 Published Information ... 98
11 Changes to Release 2 ... 99
11.1 Deleted SWS Items ... 99
11.2 Replaced SWS Items ... 99
11.3 Changed SWS Items ... 99
11.4 Added SWS Items ... 99

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 wakeup functionality is covered in the LIN Interface and the LIN Driver.

The base for this document is the LIN 2.0 specification [12]. 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

The Layered Software Architecture [2] positions the LIN INTERFACE 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.

- AUTOSAR confidential -

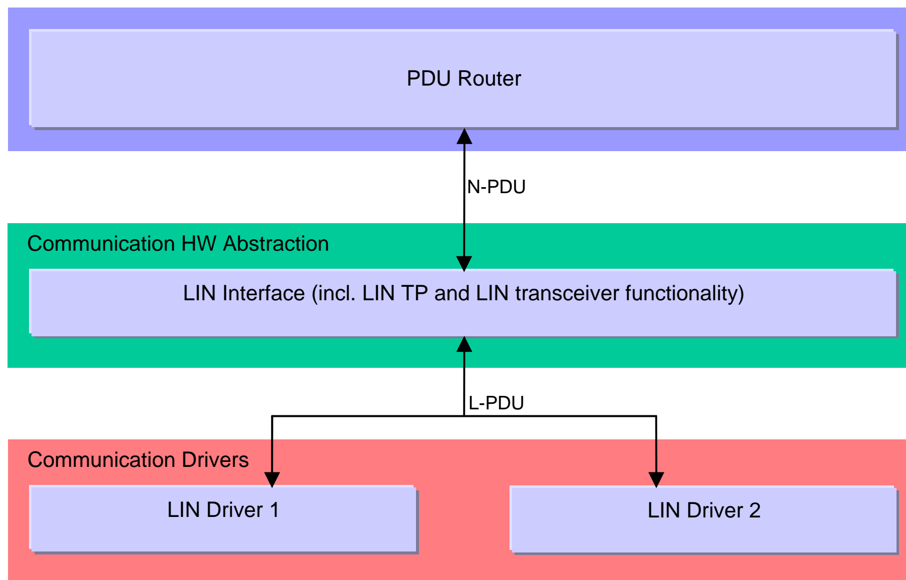

Figure 1 – AUTOSAR BSW software architecture (LIN relevant modules)

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.

- AUTOSAR confidential -

2 Acronyms and abbreviations

In addition to the acronyms and abbreviations found in the LIN 2.0 specification the following are used throughout this document. Some terms already defined in the LIN 2.0 specification is also defined here. The reason that it sometimes needs more clarification and if it used many times in this document.

Abbreviation / Acronym:	***Description:***
API	Application Program Interface
CF	Continuous Frame in TP
DCM	Diagnostic Communication Manager
Delay	The time between to 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 $T_{Frame_Maximum}$ 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
https://svn.autosar.org/repos/10Releases/
AUTOSAR_BasicSoftwareModules.pdf

[2] Layered Software Architecture,
https://svn.autosar.org/repos/10Releases/
AUTOSAR_LayeredSoftwareArchitecture.pdf

[3] General Requirements on Basic Software Modules
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SRS_General.pdf

[4] Specification of Standard Types
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_StandardTypes.pdf

[5] Specification of Development Error Tracer
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_DET.pdf

[6] Requirements on LIN
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SRS_LIN.pdf

[7] Specification of LIN Driver
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_LIN_Driver.pdf

[8] Specification of Diagnostics Event Manager
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_DEM.pdf

[9] Specification of ECU Configuration
https://svn.autosar.org/repos/10Releases/
AUTOSAR_ECU_Configuration.pdf

[10] Specification of ECU State Manager
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_ECU_StateManager.pdf

[11] Specification of the BSW Scheduler
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_BSW_Scheduler.pdf

3.2 Related standards and norms

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

- AUTOSAR confidential -

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.

LINIF377: To be able for the LIN Interface to operate the following modules are interfaced:

- LIN Driver – Lin
- PDU Router – PduR
- ECU State Manager – EcuM
- COM Manager – ComM

5.1 Upper layers

5.1.1 PDU Router

The PDU Router will be interfaced for transmission and reception of frames. It is assumed that the PDU router or a module above will make the copying of the data of the frames for reception and transmission. Additionally the PDU router will handle the TP messages buffers complete or fragmented

5.1.2 BSW Scheduler

The LIN Interface needs the 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 will 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 shared data with above or below modules. The data that is shared will not need a help of the OS 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 implementer to solve these accesses.

5.1.4 Module DET (Development Error Tracer)

In development mode the Det_ReportError – function of module DET [5] will be called.

5.1.5 Module DEM (Diagnostic Event Manager)

Production errors will be reported to the Diagnostic Event Manager [8].

5.1.6 Module ECU State Manager

The ECU state manager will have one purpose for the LIN Interface:
1. When a bus wakeup up is detected by the LIN Interface the ECU state manager is notified.

5.1.7 Module Com Manager

The Com manager will have two purposes for the LIN Interface:
1. The Com manager initializes the LIN Interface.
2. The Com manager start and stop of sending and receiving I-PDU's via AUTOSAR COM.

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 will also tell 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)
- Query of reception of the response part of a frame (Lin_GetStatus). 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)
- Callout when detecting a wake up request (LinIf_WakeUpNotification). When the LIN Driver detects a wake up request, it shall call this function in the LIN Interface.

- Initialization of the LIN Driver (Lin_Init).
- Initialization of the associated hardware (Lin_InitChannel).

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.

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.

LINIF246: The header-file structure shall be used as depicted in Figure 2 - Header file structure.

LINIF457: The EcuM.h file shall be included go get access to the wakeup notification API call from the ECU state manager.

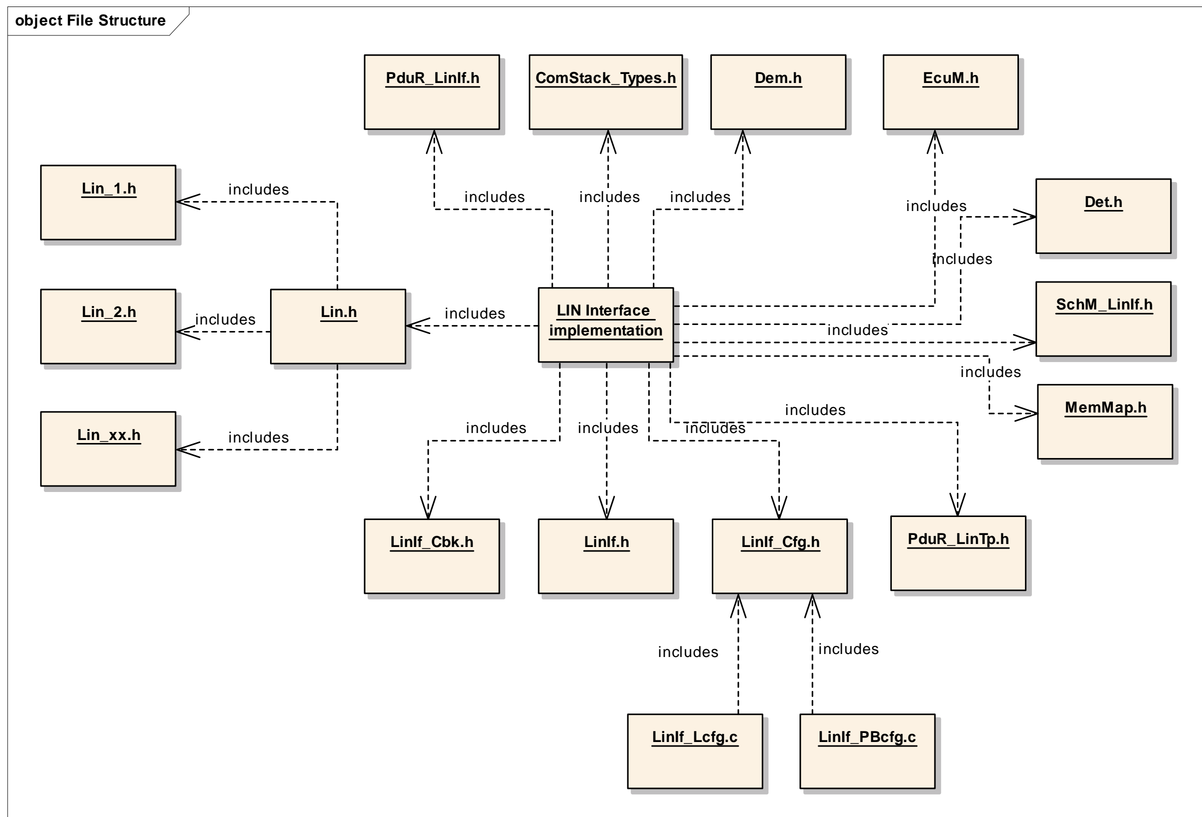

Figure 2 - Header file structure

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 Dem.h file.

LINIF434: The LIN_xx.h file describing the external API and configuration of each LIN Driver shall be included in one common file called Lin.h.

LINIF469: The LIN Interface shall include the SchM_LinIf.h file.

LINIF470: The LIN Interface shall include the ComM.h file

LINIF471: The LIN Interface shall include the MemMap.h file.

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 SRS:s 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	[LINIF377]
[BSW00387] Specify the configuration class of callback function	[LINIF358]
[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	[LINIF358]
[BSW101] Initialization interface	[LINIF198]
[BSW00416] Sequence of Initialization	[LINIF198]
[BSW00406] Check module initialization	[LINIF379] [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 Lin If. [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	[LINIF377]
[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	[LINIF223]
[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

- AUTOSAR confidential -

	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	[LINIF196]
[BSW00347] Naming separation of different instances of BSW drivers	LinIf is not a driver
[BSW00305] Self-defined data types naming convention	Chapter 8.2
[BSW00307] Global variables naming convention	The naming of parameters is made in chapter 10
[BSW00310] API naming convention	Chapter 8.3
[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	[LINIF471]
[BSW158] Separation of configuration from implementation	[LINIF241] [LINIF242] [LINIF243] [LINIF244] [LINIF245]
[BSW00314] Separation of interrupt frames and service routines	No interrupt functions in LinIf
[BSW00370] Separation of callback interface from API	Chapter 8.4
[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	[LINIF246]
[BSW00302] Limit exported information	[LINIF242]
[BSW00328] Avoid duplication of code	Complete LinIf SWS. The LinIf supports multiple channel. 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 8.3
[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 8.3
[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	LINIF469

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 wakeup during transition to sleep-mode	[LINIF295]
[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	[LINIF223]
[BSW01571] Transmission request service	[LINIF201]
[BSW01514] Wake-up notification support	[LINIF358]
[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	[LINIF037] [LINIF257]
[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 optimised	LIN driver requirement
[BSW01572] LIN driver initialization	LIN driver requirement

- AUTOSAR confidential -

[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 10.4
[BSW01544] Error handling	Chapter 7.5.2.6 and chapter 7.5.2.8

- AUTOSAR confidential -

7 Functional specification

This document is referring to the LIN 2.0 specification package [12] in several places, hereafter only LIN 2.0 specification will be used.

This chapter specifies the requirements on the module LIN Interface. Note that the LIN Transport Protocol is included in the LIN Interface. When mentioning LIN Interface both LIN Interface and LIN TP is also referred. See the Basic Software Modules document [1] for an overview of the responsibilities of the LIN Interface.

This chapter is organized so that it tries to follow 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 relevant to reinvent the requirements already specified in the LIN 2.0 specification. However, there are specific details for AUTOSAR and parts that needs to be specified since they are not specified enough or is 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 above requirement basically means that the communication from a LIN 2.0 master and the LIN Interface master will be equal.

LINIF249: It is required that the master behavior that is supported by the LIN Interface is constructed 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/ receiption of the different frames are 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 frame is used to enable sporadic transmission from slaves. The normal usage for this type of frame is to be used in non-time critical functions.

Since more than one slave may respond to an event triggered header, a collision may occur. The transmitting slaves shall detect this and withdraw from communication. If a collision occurs in event triggered frame response then all associated unconditional frames will be polled separately.

LINIF176: The order in which the unconditional frames attached to an event triggered frame will be polled is part of the configuration to decide. The list given from the configuration tool will be regarded as sorted after priority.

7.1.1.3 Sporadic frame

The LIN 2.0 specification defines a sporadic frame. A more precise definition of the sporadic frames are 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 only allowed transmitter of a sporadic frame is the master.

LINIF436: The only frame type allocated to a sporadic slot is unconditional frames.
Upper layers decide the transmission of a sporadic frame. Therefore an API call must be available that will set the sporadic frame pending for transmission.

LINIF435: There shall be an API call defined named LinIf_Transmit.

LINIF011: The API call LinIf_Transmit will be used by the upper layer to request a sporadic frame transmission.

A sporadic frame will be not sent immediately after the API call since the active schedule table must be followed.

LINIF250: The LIN Interface will flag the specific sporadic frame for transfer. The LIN Interface will transmit the frame 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. This is therefore not applicable here.

LINIF014: The priority of sporadic frames allocated to the same schedule slot is based on the list of sporadic frame descriptors provided by the configuration tool (i.e. the first sporadic frame is the most prioritized)

7.1.1.4 Diagnostic Frames MRF and SRF

The Master Request Frame (MRF) and Slave Response Frame (SRF) are frames with 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 will be transported when the corresponding schedule entry is due. The LIN Interface is processing the schedule (Schedule Table Manager) and it knows when a TP transmission is ongoing. Therefore the following requirement can be stated:

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

Note that also the node configuration also makes use of the MRF but above requirement only refers to when the MRF is encountered in the schedule table. The node configuration will 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 SRF header shall always be sent when schedule entry is due except if TP indicates that the upper layer is temporarily unable to provide a receive buffer

7.1.1.5 User-defined frames

The LIN 2.0 specification defines a frame type that is called user-defined frame.

LINIF251: User-defined frames shall not be used in the LIN Interface.

7.1.1.6 Reserved frames

The reserved frames are not allowed to be used in the LIN 2.0 specification and they are not used here.

7.1.2 Frame reception

The LIN Interface always operates as a LIN master. The LIN master controls the schedules and therefore is initiating all frames on the bus.

This chapter applies to all received frame types that are received in 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 Driver primitive Lin_SendHeader shall be called when a new schedule entry is due.

7.1.2.2 Response

Since no response part will be transmitted by this node there is no need to use the Lin_SendResponse call. The LIN Driver will automatically be set to receive after the header is transmitted

7.1.2.3 Status check

LINIF030:The Lin_GetStatus function shall be called earliest after the maximum frame length and latest when next schedule entry is due.

It is up to the implementation to find an efficient way to implement the status check. The normal implementation would be that the status could be checked in each LinIf_MainFunction call after the maximum frame length has passed. In case of that the frame is still going on (busy) the status check may be checked again in the next LinIf_MainFunction (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.

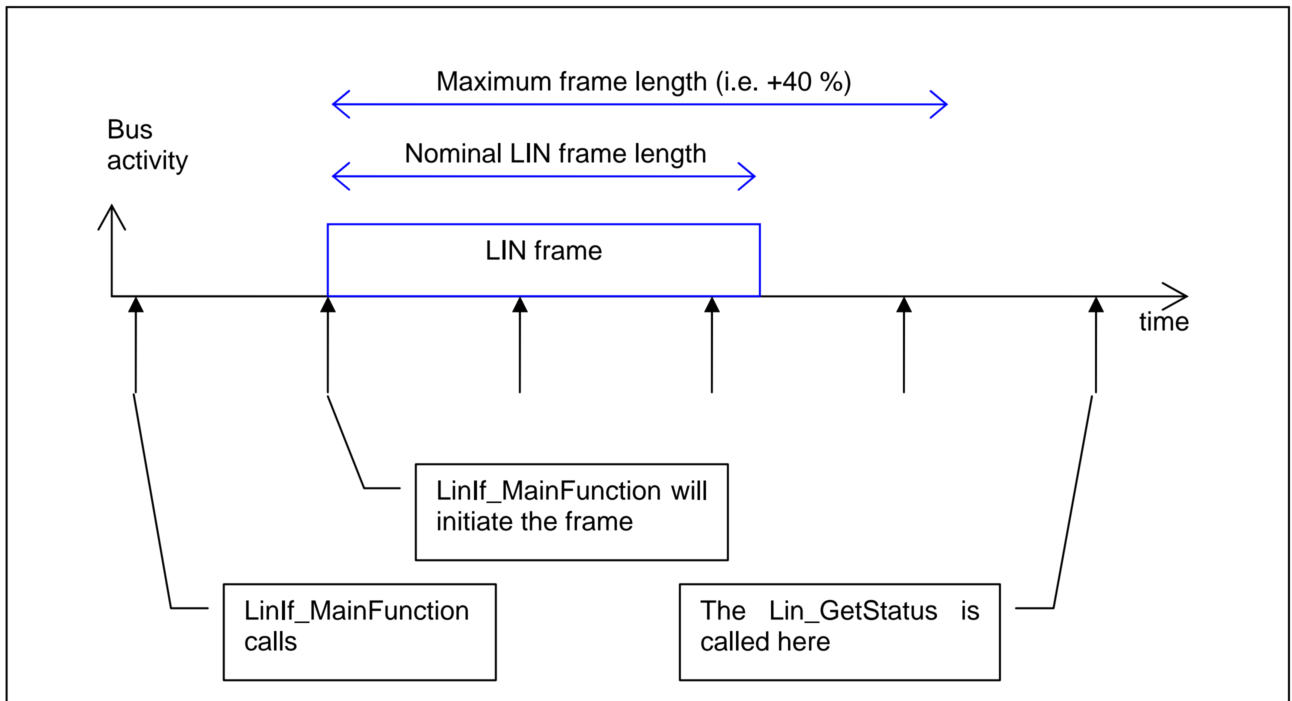

Figure 3 - Lin_GetStatus call example

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

LINIF033: If the status is LIN_RX_OK, then the LIN Interface shall issue a PduR_LinIfRxIndication callback with successful reception of frame.

LINIF259: If receiving an event triggered frame and the status is LIN_RX_ERROR, is not considered as an error.

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

LINIF227: If an event triggered frame header was transmitted and LIN_RX_BUSY or LIN_RX_ERROR occurred all the unconditional frames connected to this event triggered frame will be flagged for polling (see the LIN 2.0 specification how this is made).

LINIF258: If receiving an event triggered frame and the status is LIN_RX_NO_RESPONSE, is not considered as an error. None of the slave wanted to reply on the event triggered frame header.

- AUTOSAR confidential -

LINIF254: If receiving an unconditional frame and the status is LIN_RX_ERROR the unconditional frame will be considered as lost.

LINIF466: If the status is LIN_RX_BUSY or LIN_RX_NO_RESPONSE just before starting the next frame (i.e. in the LinIf_MainFunction that will start the transmission of the header), the frame to be received is considered as lost.
LINIF257: If the LIN frame reception above is considered to be lost the DEM shall be notified with LINIF_E_RESPONSE. This shall be valid for all frames except the event triggered frames but for the associated unconditional frame.

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: If the status reports a header error (LIN_TX_HEADER_ERROR) it shall not be reported to upper layers.

7.1.3 Frame transmission

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

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

7.1.3.1 Header

LINIF224: The LIN Driver function Lin_SendHeader will be called when a new schedule entry is due.

7.1.3.2 Response

LINIF225: After the Lin_SendHeader has returned, the PDU router shall be called by the function PduR_LinIfTriggerTransmit 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: The Lin_SendResponse will be used to provide the LIN Driver with a pointer to the data part.

7.1.3.3 Status check

LINIF127: The Lin_GetStatus function shall be called earliest after the maximum frame length and latest when next schedule entry is due.

LINIF128: If the result is LIN_TX_OK, then the LIN Interface shall issue a PduR_LinIfTxConfirmation callback.
LINIF036:If the status obtained is LIN_TX_ERROR and any LIN frame transmission is attempted the frame is considered as lost.

LINIF465: If the status is LIN_TX_BUSY just before a new frame will be transmitted the old frame is considered as lost.

LINIF037: If the LIN frame transmission above is considered to be lost the DEM shall be notified with LINIF_E_RESPONSE.

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 a sporadic frame is successfully transmitted, the pending flag shall be reset.

7.1.4 Slave-to-slave communication

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

7.1.4.1 Header

LINIF416: The LIN Driver function Lin_SendHeader will be called when a new schedule entry is due.

7.1.4.2 Response

LINIF417: The LIN Interface takes no action for the response since the LIN Interface neither transmits nor receives the response

7.1.4.3 Status check

LINIF418: No action shall be taken to check the transportation of this response.

7.2 Schedules

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

Each channel may have separate sets of schedule tables. The time between to start of frames (delay) is a multiple of the time-base for the specific cluster. Note that the time-base should not be confused with the tick.

LINIF260: There shall be defined one time-base for each LIN channel.
LINIF261: The delay between two frames shall be a multiple of the time-base for the channel.

LINIF262: After the LinIf_Init is finished each channel shall be able to have an active independent schedule table.

LINIF231: A pre-defined schedule table per channel shall exist that is called NULL_SCHEDULE.

LINIF263: The NULL_SCHEDULE shall contain no frames.

7.2.1 LinIf_MainFunction

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

LINIF284: The LinIf_MainFunction will operate all channels. The LinIf_MainFunction will be called periodically with a given period.

LINIF223: The period of the invocation of the LinIf_MainFunction shall be based on the time-bases (the LIN Interface shall be able to handle more then one LIN cluster) that are defined for the different LIN clusters. The Greatest Common Factor (GCF) of the time-bases shall be used.

Further reference to this GCF value is referred to as a 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).

LINIF286: The LinIf_MainFunction shall ask the Schedule Table Manager what frame to be transported.

LINIF287: Transportation (transmission and reception) of frames are only processed within the LinIf_MainFunction.

LINIF289: All calls to the upper layers, except the wake up indication, shall be made from within the LinIf_MainFunction function.

To have all calls within the LinIf_MainFunction implies that these call will be made "periodic" (since the function is called with a specific period).

7.2.2 Schedule table manager

The schedule table manager is not found 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. The priority range is 0 to 255, where 0 is highest priority.

The schedule table manager handles the schedule table and therefore when start frame transmission and reception occurs. The LinIf_MainFuntion will poll the Schedule Table Manager for which frame to transport.

LINIF389: There shall be an API call LinIf_ScheduleRequest that may be used by upper layers to request a schedule table to be executed.

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

LINIF390: There shall be two types of schedule tables: RUN_CONTINUOUS and RUN_ONCE.

The idea to support this two 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-case for RUN_ONCE can be a diagnostic session is started, LIN 2.0 node configuration is made or certain sensors in the slaves needs to be polled.

LINIF391: Each schedule table defined as RUN_ONCE in the configuration shall have a fixed priority.

LINIF396: There shall not be any equal priority on one channel for two or more RUN_ONCE schedule tables.

LINIF392: All RUN_CONTINUOUS schedule table shall have equal lowest priority (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:
Equal priority of RUN_CONTINUOUS schedule table requests is resolved by prioritizing always the latest request.

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

LINIF444: If the NULL_SCHEDULE is requested (or set in case of initialization or sleep), the schedule table manager queue shall be flushed except the NULL_SCHEDULE. The switch to NULL_SCHEDULE shall be made when schedule entry is due (even if the current is RUN_ONE).
It shall not be possible to request a schedule table in CHANNEL_SLEEP.

LINIF467: If a schedule is requested in CHANNEL_SLEEP the request will be rejected and DET will be notified with LINIF_E_SCHEDULE_REQUEST_ERROR.

The LIN INTERFACE allows changing the current schedule table to another one. The LinIf_ScheduleRequest will select the schedule table to be executed. The LinIf_ScheduleRequest function can be called anytime, but the actual switch to the new schedule is made as follows:

LINIF028: The actual switch to the new schedule table takes place at the time of the next schedule entry if current schedule is RUN_CONTINUOUS.

LINIF393: A RUN_ONCE schedule table must be executed from the first entry to the last entry before changing to a new schedule table.

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

LINIF029:At the time of the switch, the pending state of sporadic and event triggered frames shall not be cleared.

LINIF395: The number of requests (i.e. the queue length) that can be handled shall be specified per channel in the configuration.

LINIF394: If the LinIf_ScheduleRequest is called and number of requests are full, the function shall return with error. The error code LINIF_E_SCHEDULE_OVERFLOW shall be reported to DET

LINIF397: If no requests are left to be served, the latest requested RUN_CONTINUOUS schedule table shall be set.

Note that since the init function will set the NULL_SCHEDULE it means that there is always a latest requested schedule table.
Following examples shoes how the schedule table manager works:

Schedule table	***Type***	***Priority***
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 will be executed as shown in Figure 4.

ReadById	CheckSensors	Normal	Normal	…

Figure 4 - Schedule table execution example

Another example of the schedule table request:

Schedule table	***Type***	***Priority***
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 will be executed as shown in
Figure 5.

ReadById	ReadById	CheckSensors	Null-schedule	…

Figure 5 - Schedule table execution example

7.3 Node management

The network management described here is the LIN 2.0 specification network management and not to confuse 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 precise than the LIN 2.0 specification defines how a node operates.

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

LINIF237: The power management in the LIN 2.0 specification shall not be used.

7.3.1 Node Management

The LIN Interface will operate in a state-machine. Each channel connected will operate in a sub-state-machine.

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

The state-machine is depicted in Figure 6.

LINIF290: Each LIN channel has a separate sub-state-machine.

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

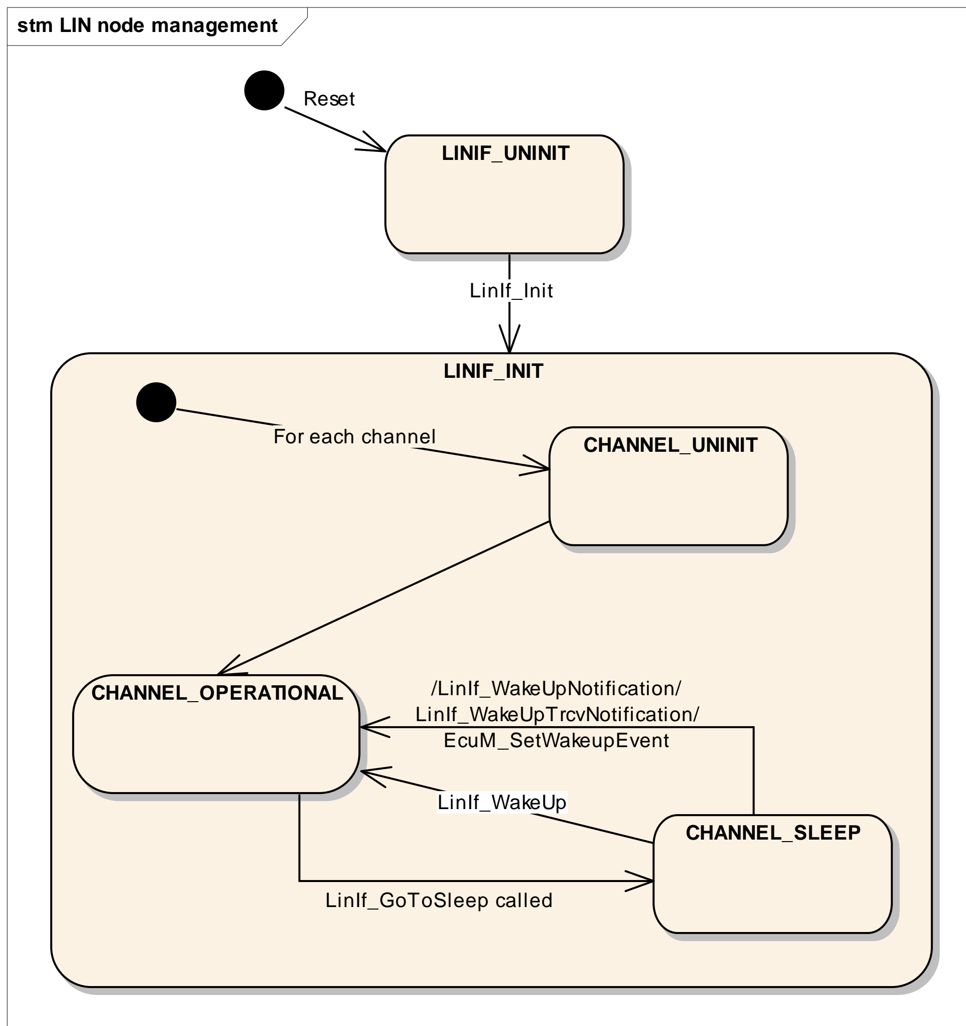

Figure 6 - LIN Interface state-machine and LIN channel sub-state-machine

Following section elaborates the state-machine for the LIN Interface

7.3.1.1 LINIF_UNINIT

LINIF438: There shall be a state called LINIF_UNINIT

LINIF380: The state LINIF_UNINIT is active. If any API call, except LinIf_Init and LinIf_ScheduleRequest, is called then LINIF_E_UNINIT shall be reported to DET.

7.3.1.2 LINIF_INIT

LINIF439: There shall be a state called LINIF_INIT

LINIF381: The LIN Interface state-machine shall transit from LINIF_UNINIT to LINIF_INIT when LinIf_Init is called.

Following sections sets further requirements on the state-machine and the sub-state-machine.

7.3.1.3 CHANNEL_UNINIT

The LIN Interface is initialized but the LIN channel is not initialized.

LINIF440: There shall be a state for each channel called CHANNEL_UNINIT.

7.3.1.4 CHANNEL_OPERATIONAL

The channel is initialized and operates normally.

LINIF441: There shall be a state for each channel called CHANNEL_OPERATIONAL

LINIF189: LIN frame headers shall be transmitted and responses shall be received/transmitted in this state only.

LINIF053: The currently selected schedule table shall be processed in the LinIf_MainFunction calls.

7.3.1.5 CHANNEL_SLEEP

The channel is in sleep. This means that the LIN Interface has successfully transmitted the go-to-sleep-command and no wake up requests have been detected.

LINIF442: There shall be a state for each channel called CHANNEL_SLEEP.

LINIF114: Upon entering or exiting the CHANNEL_SLEEP state the LIN Interface will not set the hardware interface or µ-controller into a new power mode. This is not in the scope of the LIN Interface

LINIF043: In CHANNEL_SLEEP the LinIf_MainFunction shall not initiate any traffic on the bus.

7.3.2 Initialization process

7.3.2.1 LinIf_Init process

LINIF235:The underlying LIN Driver function Lin_Init shall be called.
The configuration that is used will be set at this time.

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

LINIF233: The NULL_SCHEDULE shall be set for each channel.

LINIF294: The underlying LIN Driver channel function Lin_InitChannel shall be called.

7.3.3 Go to sleep process

The LinIf_GotoSleep function 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. This is called the go-to-sleep-command in the LIN 2.0 specification.

LINIF443: There shall be a in API call named LinIf_GotoSleep

LINIF453: Latest when the next schedule entry is due, the Lin_GoToSleep command shall be called instead of the scheduled frame.

This means that the Lin_GoToSleep can be called in the interval from the previous frame is finished until the next schedule entry is due. This is up to the implementer to decide.

LINIF455: If the Lin_GetStatus reports LIN_CH_SLEEP then the go-to-sleep-command was correct transmitted. The channel state shall be set to CHANNEL_SLEEP

LINIF454: If the Lin_GetStatus reports not LIN_CH_SLEEP then the go-to-sleep-command was not successful. The LIN Interface shall switch to CHANNEL_SLEEP and switch to NULL_SCHEDULE.

The reason to do this way is that upper layers require no confirmation that the go-to-sleep command has been sent. The slaves will enter sleep after 4 seconds (according to the LIN 2.0 specification). To be able to send the wakeup request the state must be set to CHANNEL_SLEEP.

LINIF293: Upon entering the CHANNEL_SLEEP state the current used schedule table shall switch to the NULL_SCHEDULE.

7.3.4 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, the LIN Driver detects a wake up request on the bus and calls the LinIf_WakeUpNotification function or the IoHwA

detects a wake up request by a LIN Transceiver which generates an ICU interrupt and calls the LinIf_WakeUpTrcvNotification.

LINIF445: There shall be a callout defined that is named LinIf_WakeUpNotification.

LINIF446: There shall be an API call named LinIf_WakeUp.

LINIF296: If the LinIf_WakeUp is called and the state is CHANNEL_SLEEP then the LIN Interface issues a call Lin_WakeUp to the LIN Driver to transmit a wake up request on the selected channel.

LINIF306: If the LinIf_WakeUp is called, the state is CHANNEL_OPERATIONAL and the go-to-sleep command is not pending then the LinIf_WakeUp returns with no action.

LINIF307: If a wake up request is detected by the LIN Driver and the LinIf_WakeUpNotification is called the EcuM_SetWakeupEvent shall be called with the parameter LINIF_WAKEUP_SOURCE for the woken up channel.

LINIF298: If a wake up request is detected by the LIN Driver and the LinIf_WakeUpNotification is called the state for the specific channel shall be changed to CHANNEL_OPERATIONAL.

LINIF468: There shall be a callout defined that is named LinIf_WakeUpTrcvNotification.

7.3.4.1 Wakeup during sleep transition

Since the wakeup process is a sporadic event it may happen that someone tries to wakeup the cluster while the LIN Interface is processing the go-to-sleep command.

Two cases can occur:

The first is when the LIN Interface has transmitted the go-to-sleep command but not checked the status of the transmission (see Figure 7). If this is occurs following applies:

LINIF186: If the go-to-sleep status check is pending (from the point that it has been transmitted until the status check) and a wakeup is detected on the bus, the CHANNEL_OPERATIONAL shall be maintained and the upper layer shall be notified of the wakeup-event.

The second case is 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) the upper layer is requesting a wakeup. In this case following shall apply:

LINIF459: If the go-to-sleep command is requested and the upper layer requests a wakeup before the go-to-sleep command is checked, no wakeup shall be sent on the bus and the state CHANNEL_OPERATIONAL shall be maintained.

LINIF460: When the go-to-sleep command is checked (Lin_GetStatus) and LIN_CH_SLEEP is returned then Lin_WakeUp shall be called to wakeup the channel again.

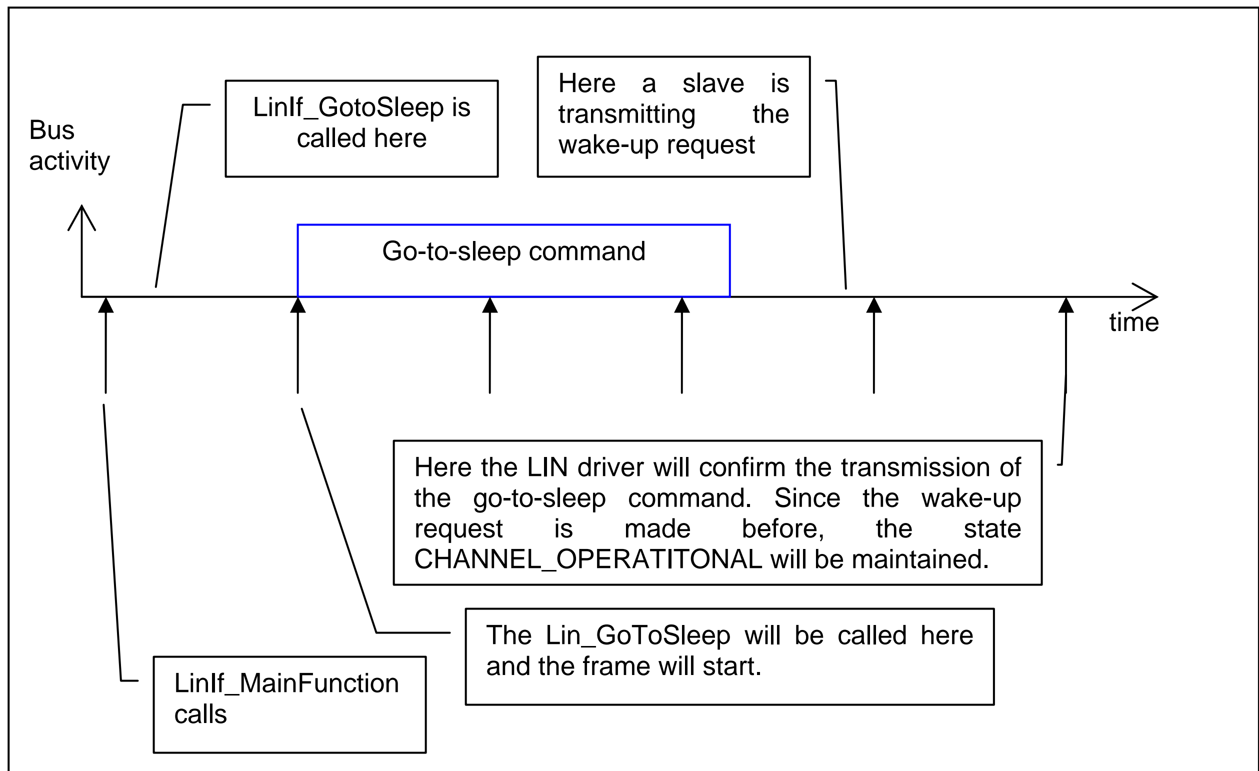

Figure 7 – Wake up requested before confirmation of go-to-sleep-command

7.4 Status Management

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

According to the LIN 2.0 specification each slave node will publish a Response_Error bit in a transmitted frame. Since this is a signal the LIN Interface will not monitor the Response_Error bit. It will be forwarded to upper layers that know of signals.

LINIF300: It is not in the scope of the LIN Interface to monitor the Response_Error bit signal. Since this is a signal it is handled by upper layers.

There is an internal reporting within 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 Succesful_transfer bits. The strategy here is only to report errors and not monitor successful transfers.

The conditions for the Error_in_response will be set in the LIN Interface 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 not anything to do with the AUTOSAR configuration.

The Diagnostic transport layer and the Node Configuration in LIN 2.0 specification share the MRF and SRF. Here 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).

In LIN 2.0 Specification there are two ways for the LIN master to configure slaves:

- By using the 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 can only do Node Configuration using schedule tables

7.5.1.1 Node Model

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

LINIF308: The Node Model shall not be used. It is meant only for slaves and not for masters.

7.5.1.2 Node Configuration services

The LIN Interface provides node configuration services 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 Node Configuration requests Assign Frame ID and Unassign Frame ID are mandatory to implement.

LINIF409: The FreeFormat shall be supported. The response is not defined, therefore a response from a slave cannot be processed.

The Node Configuration request Assign NAD, Conditional change NAD and Data Dump that are optional in the LIN 2.0 specification are also optional to implement in the LIN Interface.

LINIF310: There shall be a configuration item in the LIN Interface configuration that states if the Assign NAD and Conditional Change NAD are supported.

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

LINIF408: The Data Dump request shall not be supported.

7.5.1.3 Node Configuration API

The are number of configuration API calls defined in the LIN 2.0 specification. No API calls will be used for the node configuration in LIN Interface. All configuration will be made using schedule tables.

LINIF402: No Node Configuration requests shall be available as API calls.

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 Read by Identifier service shall not be implemented in the LIN Interface. It is the responsibility of the diagnostic layer to support this function.

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 and therefore it is possible to store this functionality in the configuration.

A number of fixed MRF:s are defined in the LIN 2.0 specification.

LINIF025: The Fixed MRF:s shall be transmitted when encountered in the schedule table:

- AssignFrameId
- UnassignFrameId
- AssignNAD
- FreeFormat
- ConditionalChangeNAD

LINIF229: The LinIF shall process these fixed MRF entries without interaction with an upper layer. In case of failure in the transmission of the fixed MRF, the slave shall not be requested for an answer.

It is possible to put in a SRF after a node configuration command in the schedule table. A slave may answer to a node configuration command as defined in the LIN 2.0. If this is the case the following will happen:

LINIF404: If the answer is positive no action will be taken

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: If no answer is received the LIN Interface shall report to the DEM with LINIF_E_NC_NO_RESPONSE.

Note that there is no negative answer for node configuration requests in LIN 2.0. Only the Read by Identifier supports a negative answer. Since it is not used in the LIN Interface there are no negative responses to process for the LIN Interface.

LINIF407: The LIN Interface shall remember the request until a new node configuration request is made or a TP message is transmitted.

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 LIN 2.0 the Transport Protocol (TP) is optional to implement. There are three types of TP 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 is not defined here. And the User Defined diagnostics will not be used since all Diagnostic communication will use the Diagnostic Transport Layer.

LINIF313: The Diagnostic Transport Layer in the LIN 2.0 specification (without the contained Diagnostic API) shall be used and shall be mandatory. The implementation of this API shall be configurable to make the LIN Interface smaller. This is equal to LIN TP.

LINIF387: There shall be a configuration parameter that states if the TP is used or not.

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

LINIF314: It shall be possible to start a 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 TP transfer stalls.

The TP will be used to transport Diagnostic services and responses. No Diagnostic sessions will be made in parallel and always in service and then subsequent request.

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

7.5.2.1 LIN TP initialization

LINIF447: There shall be an API named LinTp_Init.

LINIF098:TP services shall be available only after LIN Interface has been initialized, the channel is initialized and the TP is initialized (i.e. LinTp_init has been called).

7.5.2.2 LIN TP shut down

LINIF448: There shall be an API named LinTp_Shutdown.

LINIF449: When the LinTp_Shutdown is called it shall not be possible to start any new TP messages. Ongoing TP messages shall be interrupted.

7.5.2.3 State-machine

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

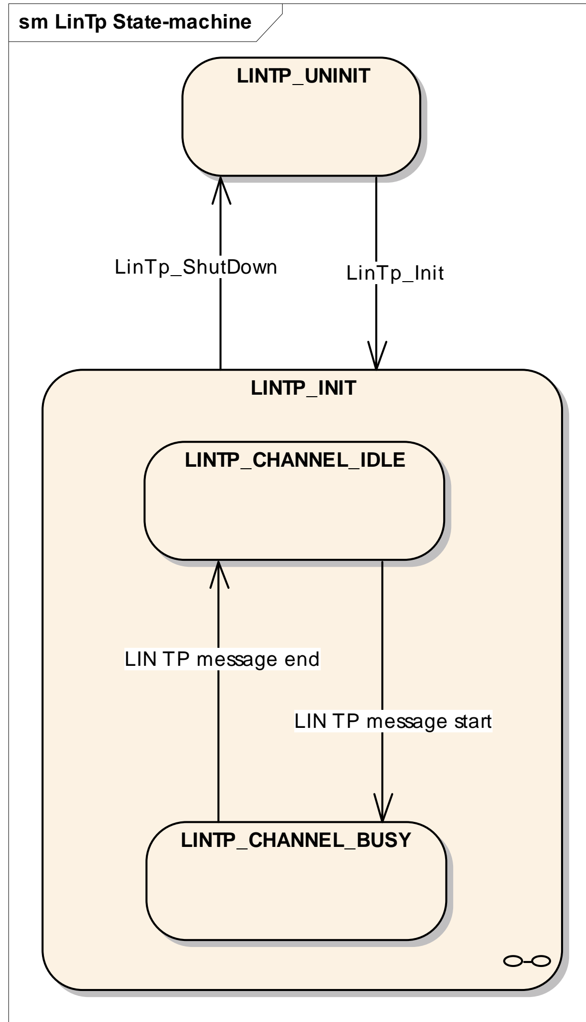

Figure 8 - LIN Transport Protocol state-machine

LINIF315: Each channel shall have one instance of the LinTp sub-state-machine

LINIF316: A state LINTP_UNINIT shall exist

LINIF317: The LINTP_UNINIT shall be set after reset or call of LinTp_ShutDown

LINIF318: It shall not be possible to start any TP message in state LINTP_UNINIT

LINIF319: A state LINTP_INIT shall exist

LINIF320: The state LINTP_INIT shall be set if the LinTp_Init is called.

LINIF412: Each channel shall have a sub-state-machine in the state LINTP_INIT.

LINIF450: There shall be a state called LINTP_CHANNEL_IDLE.

LINIF321: A transmission of a TP message can only be started in the state LINTP_CHANNEL_IDLE.

LINIF414: The LINTP_CHANNEL_IDLE shall be entered when a TP message is successfully sent or an unrecoverable error occurred.

LINIF322: A state LINTP_CHANNEL_BUSY shall exist.

LINIF323: The state LINTP_CHANNEL_BUSY shall be set if a FF or a SF is received and it is detected to be a TP message (i.e. not conflicting with a configuration response from a LIN slave node).

LINIF413: The LINTP_CHANNEL_BUSY shall be entered when the LinTp_Transmit is called.

7.5.2.4 Buffer handling

The buffer for the transmission or reception of a TP message is provided by the PDU router. It is assumed that the buffer that is given by the PDU Router will be owned by LinIf until a new buffer is requested

LINIF324: The request for new buffer shall be made after the frame transportation is requested (i.e. the send_header and send_response (in tx case)). The reason is that the request for new buffer may introduce jitter on the bus.

LINIF325: The request for new buffer shall made in each LinIf_MainFunction if the current buffer length is not enough to fill next MRF (FF, SF and CF). Note that FF and SF is possible since the request for buffer in LinTp_Transmit may return NOT_OK.

This means that the actual setup of each frame for the TP will be made in the LIN Interface. There is no need for extra protection of this buffer copying that is made since the buffer copied from (provided by the PDU router) and copied to (temporary buffer in LIN Interface) are not touched by anyone else except the LIN Interface.

LINIF430: There shall be a buffer that can contain a complete frame. This buffer shall be available for both transmission and reception of an TP message.

7.5.2.5 TP Transmission

A TP message will not be transmitted directly on LIN. Since all frames must follow the schedule table also TP message must do this. All TP messages are using the MRF and SRF for transportation.

LINIF451: There shall be an API named LinTp_Transmit.

LINIF326: The API LinTp_Transmit call shall prepare a TP message for transmission.

LINIF422: The LinTp_Transmit function will be given an N-SDU id. The N-SDU id shall be converted to a specific channel and the destination NAD for the slave.

LINIF388: If LinTp_Transmit is called and a TP message is already ongoing on the selected channel the function shall return without interfering with the ongoing TP message

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: When the upper layer initiates a TP transmission the LIN Interface shall request for a buffer from the upper layer.

The Figure 9 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.

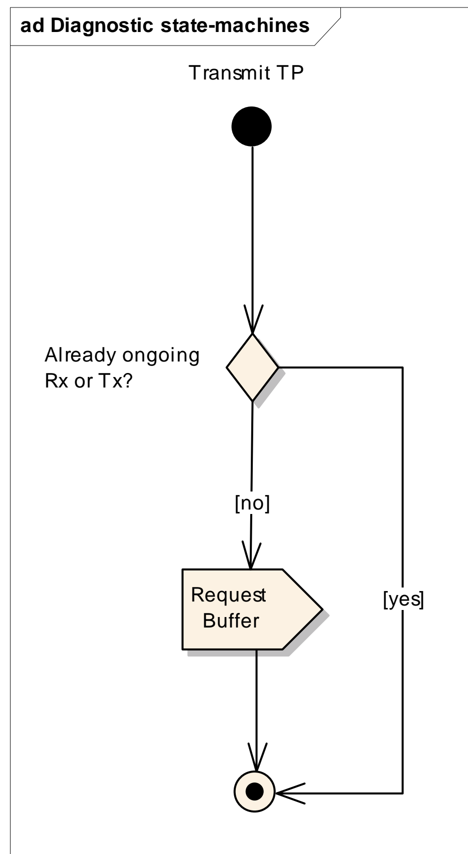

Figure 9 - TP transmission initiation

LINIF328: The LIN TP will request data, by calling PduR_LinTpProvideTxBuffer, from the upper layer as soon as there is insufficient data to be sent.

It can happen that the PduR_LinTpProvideTxBuffer will provide with message bytes that at least covers the next frame.

Example: If there is no bytes left in the current buffer the next TP frame is a CF. This means that 6 bytes is required for the next frame. If the PduR_LinTpProvideTxBuffer just provides with one byte it has to be called 6 times to fill the next frame.

LINIF329: If there is not sufficient bytes provided from PduR_LinTpProvideTxBuffer for the next MRF then the next MRF will not be sent (i.e. the frame slot will be empty). This includes the BUSY return value from the PduR_LinTpProvideTxBuffer.

The Figure 10 - TP message transmission shows the process after the TP message is initiated.

LINIF330: All the handling of the TP message transmission after the message is initiated shall be kept in the LinIf_MainFunction.

It may occur that the requested buffer from the upper layer does not fit exactly into a frame or a number of frames. The left over data must then be copied before new buffer can be requested. This is called Save Buffer in the Figure 10 - TP message transmission.

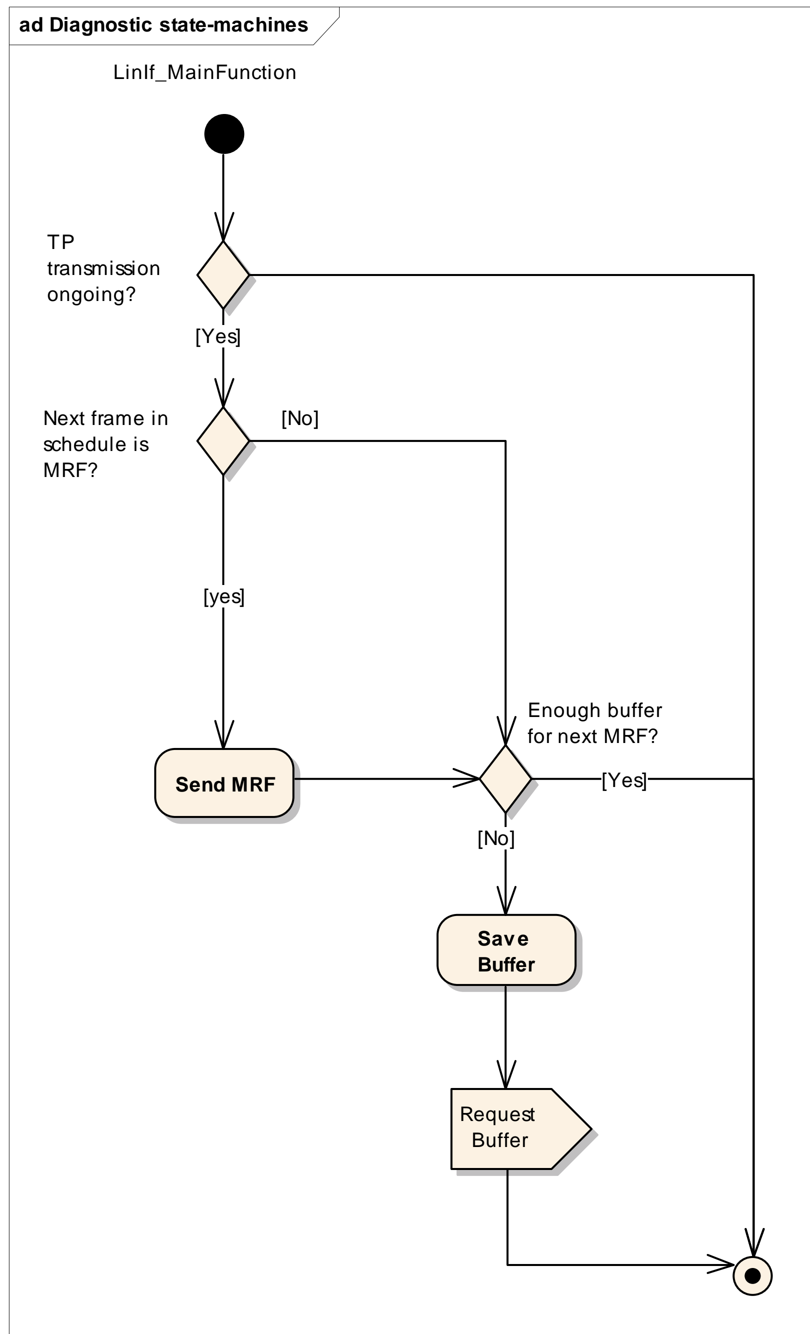

Figure 10 - TP message transmission

LINIF068: When the last frame (SF or CF) is correct transmitted the PduR_LinTpTxConfirmation is called to indicate that transmission ended with success.

7.5.2.6 TP transmission error

LINIF069: In case there is a LIN error on the MRF carrying the TP message shall be aborted. The PduR_LinTpTxConfirmation is called to notify of the error and abortion.

LINIF073: If the callback PduR_LinTpProvideTxBuffer indicates permanent failure then attempting to recover is worthless. In this case the TP transmit sequence is aborted and failure is indicated in the PduR_LinTpTxConfirmation callback.

7.5.2.7 TP Reception

The LIN Interface shall be prepared that a reception of a TP message can start at 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 TP message is started, it must have the possibility to store part of the TP message.

LINIF075 The reception of a FF or SF in a SRF indicates a towards the upper layers by calling the PduR_LinTpProvideRxBuffer callback function.

LINIF076: The LIN TP shall be able to convert the NAD from the transmitting LIN slave to a N-SDU id that the upper layer understands.

LINIF221: Subsequently received SRF frames shall be processed and stored in the buffer provided by PduR_LinTpProvideRxBuffer.

LINIF077: If the buffer provided by PduR_LinTpProvideRxBuffer is exhausted before the entire TP message is stored, the LIN INTERFACE shall request a new buffer by calling PduR_LinTpProvideRxBuffer.

LINIF078: If the last SRF in the TP message (CF) is successfully received and stored, the LIN Interface shall indicate success by calling PduR_LinTpRxIndication.

7.5.2.8 TP reception error

If the TP message reception is ongoing and an LIN error occurs on one SRF, the LIN TP will not be notified since the LIN Interface will reject the content. The way to detect errors is to check the contents of the successful SRF:s received.

LINIF079: The following errors will stop the current TP message reception immediately.

- Incorrect sequence number.
- Unexpected PCI is received (e.g. a SF is received after a CF).
- Incorrect NAD.

It is possible that a LIN slave will start a new TP message in while another TP message reception is ongoing.

LINIF080: If the FF or a SF is received when a TP reception is ongoing, a new TP reception shall start.

LINIF081: If a TP message is stopped because of an error the PduR_LinTpRxIndication shall be called with an appropriate status to indicate failure.

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 will continue 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 will continue to transmit responses to the SRF headers. This means that no error-recovery is supported.

LINIF332: If the PduR_LinTpRxIndication indicates permanent failure or the Lin Interface encountered an error, the rest of the CF frames from the slave in the TP message shall be discarded.

7.5.2.9 Unavailability of receive buffer

The callback PduR_LinTpProvideRxBuffer requesting the buffer may indicate that it is not available. This can be:

- a wait condition
- a permanent failure

The LIN Interface handles these cases differently.

LINIF085: If the LIN Interface has to wait for the buffer, then the LIN TP shall suspend sending LIN headers for the SRF until one of the following conditions are met:
the upper layer does provide a new buffer
the upper layer indicates a permanent failure in a later callback
a new TP message is started

It is necessary to check if the TP message can be resumed.

LINIF086: Requesting a receive buffer is retried periodically in the LinIf_MainFunction.

LINIF087: If the callback indicates permanent failure then the LIN TP will stop the ongoing TP message reception. A failure is indicated in the PduR_LinTpRxIndication callback.

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 contains different LIN hardware. In such case multiple LIN drivers will be used.

7.6.1 Multiple channels

LINIF461: Each channel shall have a unique channel index (LinIf_ChannelIndex). The index shall be unique even when the LIN channels are located on different LIN drivers.

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: For each channel, it must be stated in the configuration which LIN driver will be use.

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

7.7 Error classification

This chapter lists and classifies all that errors that can be detected within this software module. Each error is classified according to relevance (development / production) and related error code. For development errors, a value is defined.

LINIF266: Values for production code Event Ids are assigned externally by the configuration of the Dem. They are included via Dem.h.
The Dem.h includes the API:s to report errors as well as the required Event Id symbols. The table below defines the name of the Event Id symbols that are provided by XML to the DEM configuration tool. The DEM configuration tool assigns ECU dependent values to the Event Id symbols and publishes the symbols in Dem_IntErrId.h.

LINIF267: Development error values are of type uint8.

The following
Table 3 shows the available error codes for the LIN Interface and the LIN TP:

LINIF376:

Type or error	*Relevance*	*Related error code*	*Value [hex]*
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	Production	LINIF_E_RESPONSE	Assigned by DEM
If a slave did not answer on a node configuration request	Production	LINIF_E_NC_NO_RESPONSE	Assigned by DEM

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 *LINIF_DEV_ERROR_DETECT* (see chapter 10) shall activate or deactivate the detection of all development errors.

LINIF269: If the *LINIF_DEV_ERROR_DETECT* 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 cannot be switched off.

7.9 Error notification

LINIF271: Detected development errors will be reported to the error hook of the Development Error Tracer (DET) if the pre-processor switch *LINIF_DEV_ERROR_DETECT* 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. The standard AUTOSAR types are defined in the AUTOSAR Specification of Standard Types document [4].

The following standard AUTOSAR header files are included:

- ComStack_Types.h

The following standard AUTOSAR types are used:

- Std_ReturnType
- Std_VersionInfoType

The following type is used from the ECU state manager:

- EcuM_WakeupSourceType

8.2 Type definitions

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

8.2.1 LinIf_ChannelIndexType

LINIF196:

Type:	uint8	
Range:	0..255	index value
Description:	Index of the LIN channel within the ECU (0 is the first LIN channel) The number of LIN channels (and LINcontrollers) is limited to 256 per ECU.	

8.2.2 LinIf_SchHandleType

LINIF197:

Type:	uint8	
Range:	0..255	index value
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.

8.2.3 LinTp_ConfigType

LINIF426:

Type:	Struct	
Range:	N/A	--
Description:	This is the base type for the configuration of the LIN Transport Protocol A pointer to an instance of this struct will be used in the initialization of the LIN Transport Protocol. The outline of the struct is defined in chapter 10 Configuration Specification	

8.3 LIN Interface API

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

8.3.1 LinIf_Init

LINIF198:

Service name:	LinIf_Init	
Syntax:	void LinIf_Init (const void *ConfigPtr uint8 LinIf_ConfigIdx);	
Service ID:	0x01	
Sync/Async:	Synchronous	
Reentrancy:	Non reentrant	
Parameters (in):	ConfigPtr	Pointer to the LIN Interface configuration LINIF371: This parameter is only relevant for Variant 3. It shall be ignored for Variant 1 and Variant 2. The parameter will still be there for compatible reason.
	ConfigIdx	LINIF461: Index of one of possibly multiple configuration sets contained in the configuration structure of the LIN Interface. LINIF271: If development error detection is enabled (i.e. LINIF_DEV_ERROR_DETECT equals ON), it shall be reported to the DET module (using LINIF:E_NONEXISTING_CHANNEL) if LinIf_ConfigIdx has an invalid value.

Parameters (out):	None
Return value:	--
Description:	This function initializes the LinIf. See section 7.3.2 for a detailed description LINIF420: The Lin_Init shall be called from this function
Caveats:	--
Configuration:	Dependent on Variant (see parameter)

8.3.2 LinIf_GetVersionInfo

LINIF340:

Service name:	LinIf_ GetVersionInfo	

Syntax:	void LinIf_GetVersionInfo {
	Std_VersionInfoType *versioninfo
	}	

Service ID [hex]:	0x03	
Sync/Async:	Synchronous	
Reentrancy:	Non reentrant	
Parameters (in):	None	
Parameters (out):	versioninfo	Pointer to where to store the version information of this module.
Return value:	None	--
Description:	LINIF278: 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. Hint: If source code this function may be realized as a macro. The macro will then be defined in the modules header file.	
Caveats:	--	
Configuration:	LINIF279: This function shall be pre compile time configurable On/Off by the configuration parameter: LINIF_VERSION_INFO_API	

8.3.3 LinIf_Transmit

LINIF201:

Service name:	LinIf_Transmit
Syntax:	Std_ReturnType LinIf_Transmit (

- AUTOSAR confidential -

	PduIdType LinTxPduId, const PduInfoType *PduInfoPtr);	
Service ID:	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 (out):	None	
Return value:	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:	LINIF105: This call indicates a request from an upper layer to transmit the frame specified by the LinTxPduId. This call only marks the sporadic frame as pending for transmission. LINIF341: Invocation for non-sporadic frame is refused (meaningless). LINIF106: Repeated invocations while the sporadic frame is still pending are tolerated, but the PDU will only be set to pending once.	
Caveats:	None	
Configuration:	LINIF452: The configuration shall keep a lookup table to convert the Upper layer identification to a frame pointer that the LIN Interface recognizes.	

8.3.4 LinIf_ScheduleRequest

LINIF202:

Service name:	LinIf_ScheduleRequest
Syntax:	Std_ReturnType LinIf_ScheduleRequest (LinIf_ChannelIndexType Channel, LinIf_SchHandleType Schedule);
Service ID:	0x05
Sync/Async:	Asynchronous
Reentrancy:	Reentrant

Parameters (in):	Channel	Channel index
	Schedule	Identification of the new schedule to be set
Parameters (out):	None	
Return value:	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:	This function can be used simultaneously by different upper layers to request a schedule table to be executed. See chapter 7.2.2 Schedule table manager for further details.	
Caveats:	This function can be called from different modules separately.	
Configuration:	Schedule tables are part of the LIN INTERFACE configuration.	

8.3.5 LinIf_GotoSleep

LINIF204:

Service name:	LinIf_GotoSleep	
Syntax:	Std_ReturnType LinIf_GotoSleep (LinIf_ChannelIndexType Channel);	
Service ID:	0x06	
Sync/Async:	Asynchronous	
Reentrancy:	Non reentrant	
Parameters (in):	Channel	Identification of the LIN channel
Return value:	E_OK	Request to go to sleep has been accepted or sleep transition is already in progress.
	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) - controller is already in sleep state
Description:	This call initiates a transition into sleep mode on the selected channel. The transition is carried out by transmitting a LIN diagnostic master request frame with its first data byte equal to 0x00. See chapter 7.3.3 for more information on sleep transitions. LINIF113: This call has no effect if the LIN channel is already serving the go-to-sleep-.	
Caveats:	This function 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.
Configuration:	--

8.3.6 LinIf_WakeUp

LINIF205:

Service name:	LinIf_WakeUp	
Syntax:	Std_ReturnType LinIf_WakeUp (LinIf_ChannelIndexType Channel);	
Service ID:	0x07	
Sync/Async:	Asynchronous	
Reentrancy:	Non reentrant	
Parameters (in):	Channel	Identification of the LIN channel
Parameters (out):	None	
Return value:	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:	This call initiates the wake up process on the selected controller. See chapter for 7.3.4 more information on wake up transitions. LINIF432: If the controller is not in sleep state, this call does nothing and returns E_OK. The channels/controllers other than the one selected by Channel are not affected.	
Caveats:	--	
Configuration:	--	

8.3.7 LinTp_Init

LINIF350:

Service name:	LinTp_Init
Syntax:	void LinTp_Init (const LinTp_ConfigType *ConfigPtr);
Service ID:	0x40
Sync/Async:	Synchronous
Reentrancy:	Non reentrant

Parameters (in):	ConfigPtr	Pointer to the LIN Transport Protocol configuration LINIF427: This parameter is only relevant for Variant 3. It shall be ignored for Variant 1 and Variant 2. The parameter will still be there for compatible reason.
Parameters (out):	--	
Return value:	--	--
Description:	Initializes the LIN Transport Layer. LINIF410: This function must be called before any other LIN TP API calls can be made.	
Caveats:	--	
Configuration:	--	

8.3.8 LinTp_Transmit

LINIF351:

Service name:	LinTp_Transmit	
Syntax:	Std_ReturnType LinTp_Transmit (PduIdType LinTpTxSduId, const PduInfoType* LinTpTxInfoPtr)	
Service ID [hex]:	0x41	
Sync/Async:	Asynchronous	
Reentrancy:	Non reentrant	
Parameters (in):	LinTpTxSduId	This parameter contains the unique identifier of the N-SDU (TP message) to be transmitted. LINIF456: The LinTpTxSduId shall be used to derive the correct NAD and channel from the configuration.
	LinTpTxInfoPtr	A pointer to a structure with N-SDU related data containing: • pointer to a N-SDU buffer • length of this buffer.
Parameters (out):	None	
Return value:	E_OK	The request can be started successfully
	E_NOT_OK	The request can not be started (e.g. there is already an ongoing TP message on the selected channel)
Description:	This service is used to request the transfer of segmented data over the LIN bus. LINIF415: The channel used must be initialized (LinIf_Init) before this call. See section 7.5 for detailed description of the usage.	
Caveats:	--	
Configuration:	--	

8.3.9 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 (out):	versioninfo	Pointer to where to store the version information of this module.
Return value:	none	
Description:	LINIF353: 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.	
Caveats:	--	
Configuration:	LINIF354: This function shall be pre compile time configurable On/Off by the configuration parameter: LINTP_VERSION_INFO_API	

8.3.10 LinTp_Shutdown

LINIF355:

Service name:	LinTp_Shutdown
Syntax:	void LinTp_Shutdown(void)
Service ID [hex]:	0x43
Sync/Async:	Synchronous
Reentrancy:	Non reentrant
Parameters (in):	None
Parameters (out):	None
Return value:	None
Description:	LINIF356: This service closes all pending transport protocol connection, frees all resources and sets the corresponding LinTp module into the LINTP_UNINIT state. LINIF433: The function shall affect all channels. LINIF357: All pending transmissions/receptions is considered to be lost and shall not be reported.
Caveats:	None
Configuration:	--

8.4 Call-back notifications

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

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

8.4.1 LinIf_WakeUpNotification

LINIF378:

Service name:	LinIf_WakeUpNotification	
Syntax:	void LinIf_WakeUpNotification (LinIf_ChannelIndexType Channel);	
Service ID [hex]:	0x60	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Channel	Identification of the LIN channel
Parameters (out):	--	--
Return value:	None	

Description:	This function is called when the LIN Driver detects a wake up request on the bus. The LIN Interface will recognize the caller by the parameter of the function.
Caveats:	This function may be called in an interrupt context. It is possible that two LIN slaves on two different clusters will send a wake up request at the same time, therefore the function must be reentrant.
Configuration:	--

8.4.2 LinIf_WakeUpTrcvNotification

LINIF468:

Service name:	LinIf_WakeUpTrcvNotification	
Syntax:	void LinIf_WakeUpTrcvNotification (LinIf_ChannelIndexType Channel);	
Service ID:	0x61	
Sync/Async:	Synchronous	
Reentrancy:	reentrant	
Parameters (in):	Channel	Identification of the LIN channel
Parameters (out):	None	
Return value:	None	
Description:	This API is called by underlying SPAL IoHwA driver in case a wake up interrupt is detected.	
Caveats:	Wake up by bus is always asynchronous to the transition to sleep and standby. In worst case wake up occurs during transition to sleep. In such a case the driver shall create a wake up by bus notification immediately after the API calls to enter standby or sleep has finished. The EcuM must be able to handle the wake up event immediately after requesting the standby or sleep mode.	
Configuration:	If no LIN Transceiver is used or wake up by bus is used or wake up by bus is polled this function need not be present in compiled code.	
Call Context	The call context of this API is expected to be within the ISR handler of the underlying driver. This has to be documented (BSW00333).	

8.5 Scheduled functions

These functions are directly called by Basic Software Scheduler.

The functions shall have no return value and no parameter. All functions shall be non reentrant.

8.5.1.1 LinIf_MainFunction

LINIF384:

Service name:	LinIf_MainFunction
Service ID [hex]:	0x80
Description:	This is the main processing function of the LIN interface. The function is described in chapter 7.2.
Timing:	This function will be called each tick (fixed cyclic) Fixed cyclic means that one cycle time is defined at configuration and shall not be changed because functionality is requiring that fixed timing.
Pre condition:	Design hint: This function may be interrupted by other LIN Interface API calls. Critical areas that are also modified by other API calls shall be protected. Other LIN Interface API calls that may touch the same resources are the LinIf_GotoSleep and LinIf_Transmit.
Configuration:	The period of the invocation is defined in chapter 7.2.

8.6 Expected Interfaces

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

8.6.1 Mandatory Interfaces

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

LINIF359:

API function	*Module*	*Description*
Lin_Init	Lin	Initializes the LIN driver
Lin_InitChannel	Lin	Initializes a LIN channel in the LIN driver
Lin_GoToSleep	Lin	Transmits the go-to-sleep-command
Lin_SendHeader	Lin	Send LIN frame header
Lin_SendResponse	Lin	Transmits response part of LIN frame
Lin_WakeUpValidation	Lin	Validation of a wake up request on the LIN Driver
Lin_WakeUp	Lin	Transmits wake up request on the specific channel
Lin_GetStatus	Lin	Returns the status of the ongoing or latest finished frame
Lin_GotoSleepInternal	Lin	Monitors the bus sleep state without sending the GotoSleep command in advance.
Dem_ReportErrorStatus	DEM	Called to indicate error status to DEM.
PduR_LinTpProvideTxBuffer	PduR	Request a buffer to upper layers where the LinTp shall read the data to transmit.
PduR_LinTpProvideRxBuffer	PduR	Request a buffer to upper layers where the LinTp shall write the receive data.
PduR_LinTpTxConfirmation	PduR	Confirmation status of a transmit N-SDU

		to upper layers.
PduR_LinTpRxIndication	PduR	Indication status of a receive N-SDU to upper layers.
PduR_LinIfRxIndication	PduR	Indication of reception of a frame
PduR_LinIfTxConfirmation	PduR	Confirmation of the transmitted frame
PduR_LinIfTriggerTransmit	PduR	Callout to get the SDU for the transmitted frame
EcuM_SetWakeupEvent	EcuM	This function will be called when the LIN Interface detects a bus-wakeup.

8.6.2 Optional interfaces

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

LINIF360:

API function	***Module***	***Description***	***Configuration parameter (description see chapter 10)***
Det_ReportError	DET	Development error notification	LINIF_DEV_ERROR_DETECT

8.6.3 Configurable interfaces

No configurable interfaces.

9 Sequence diagrams

This chapter will show 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 left out 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 will request 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 Manager. The LinIf_MainFunction will get 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 11 does not show any timing information. The timing information is depicted in Figure 12 following the diagram.

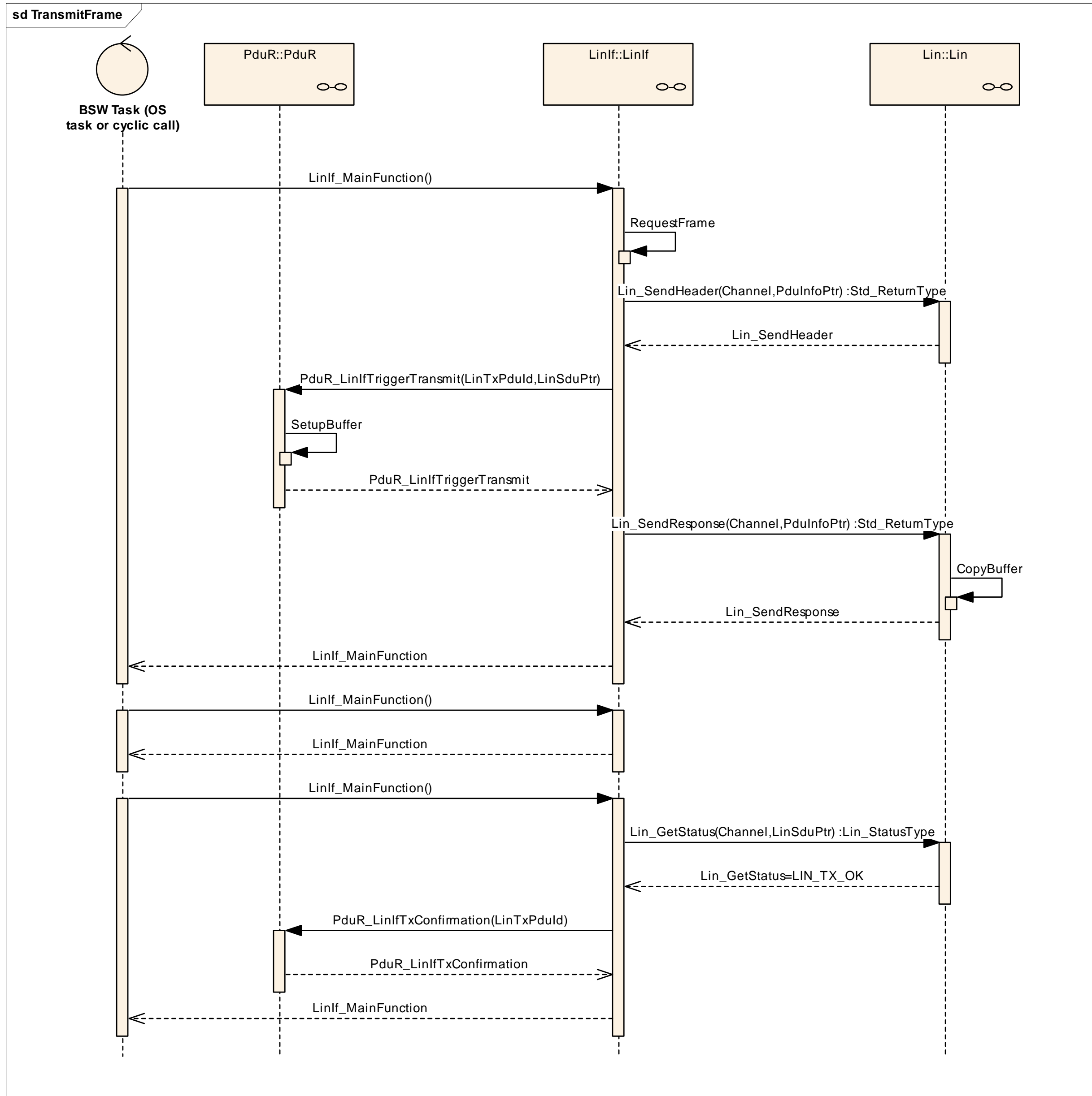

Figure 11- Frame transmission

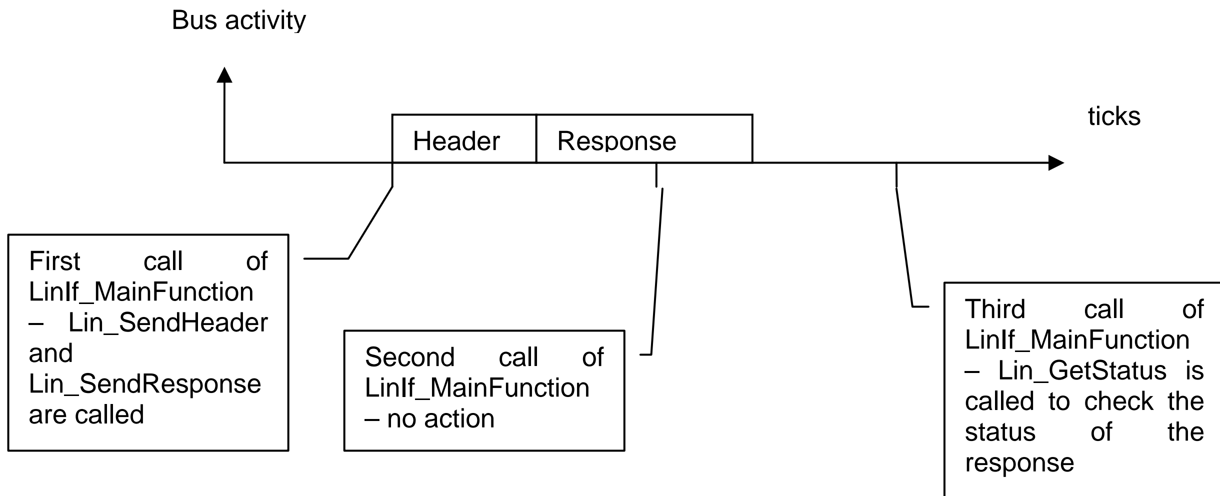

Figure 12 - Timing information for transmitted frame

9.2 Frame Reception

The following use-case shows the reception of a LIN frame. The first call of the LinIf_MainFunction will request 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 Manager. The LinIf_MainFunction will get 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.

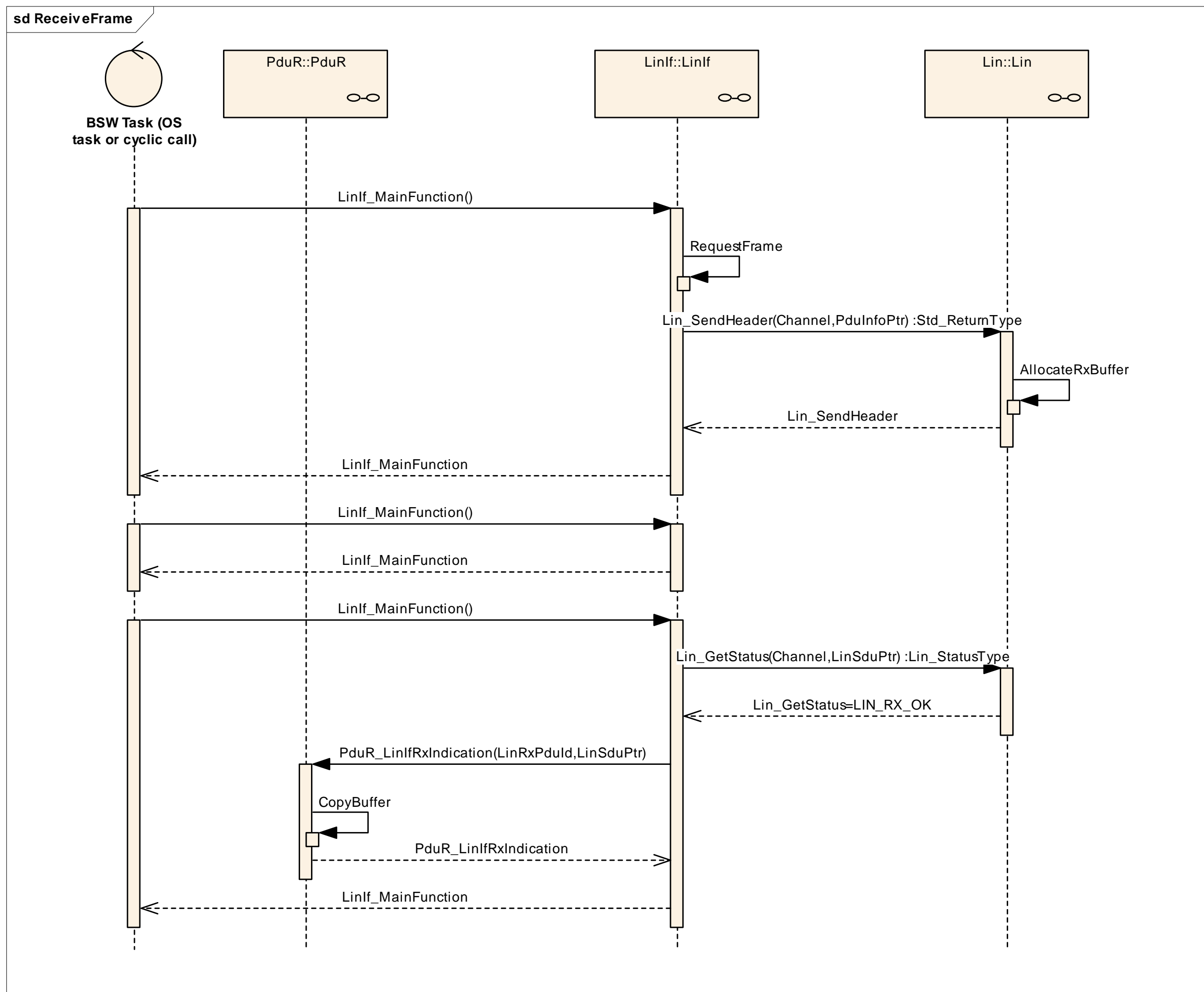

Figure 13 - Frame reception

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) will transmit 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.

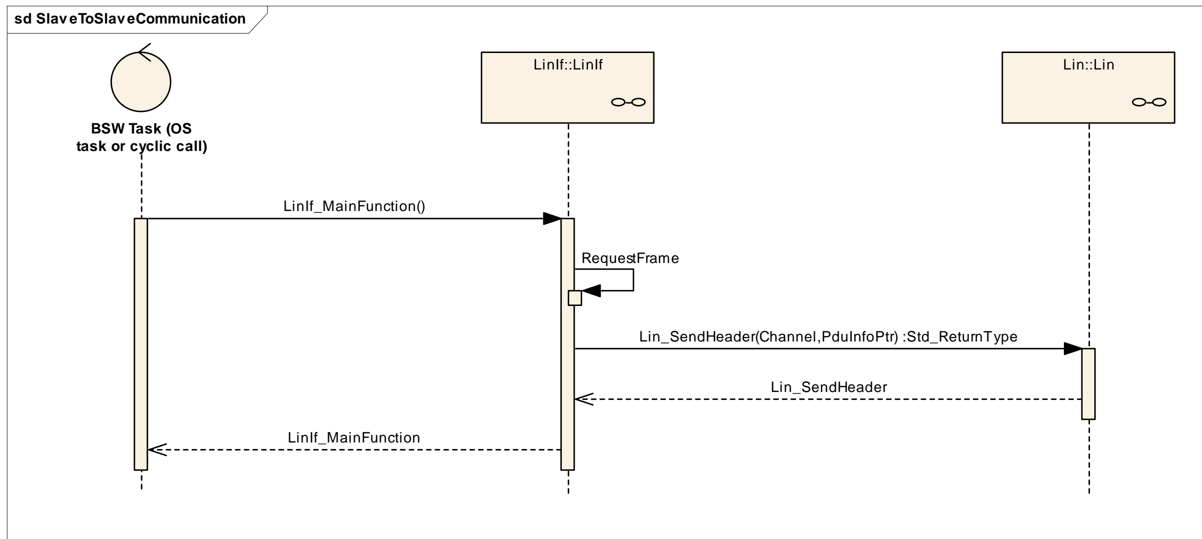

Figure 14 - Slave to slave communication

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 Manager. The LinIf_MainFunction will get 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.

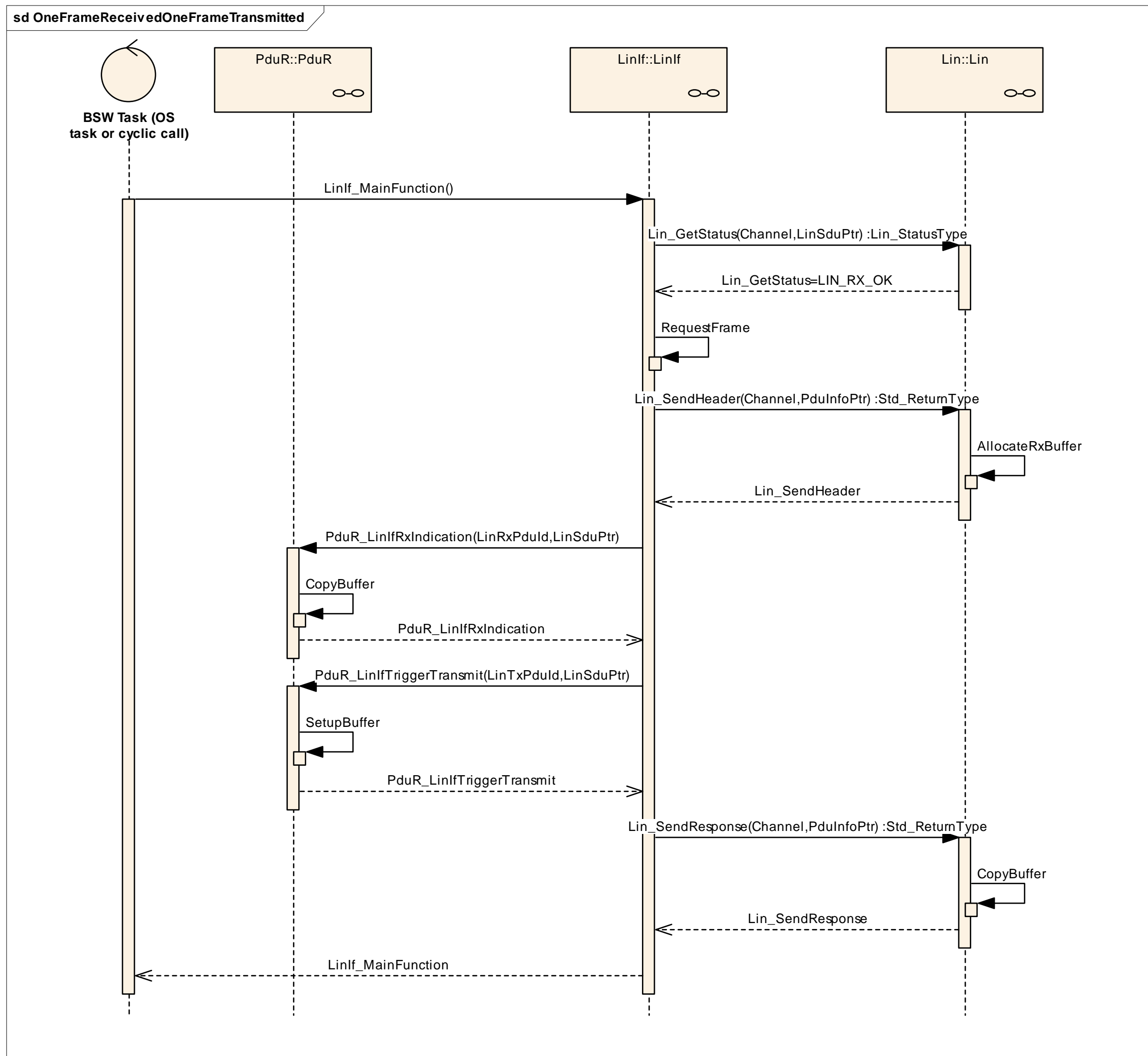

Figure 15 - One frame received and one frame transmitted

9.5 Sporadic frame

The following use-case shows a upper layer requesting transmission of a sporadic frame. Actually, this call will not initiate the transmission of the frame since the schedule table must be followed. It will just mark 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 will transmit 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 will be one flag for each sporadic frame.

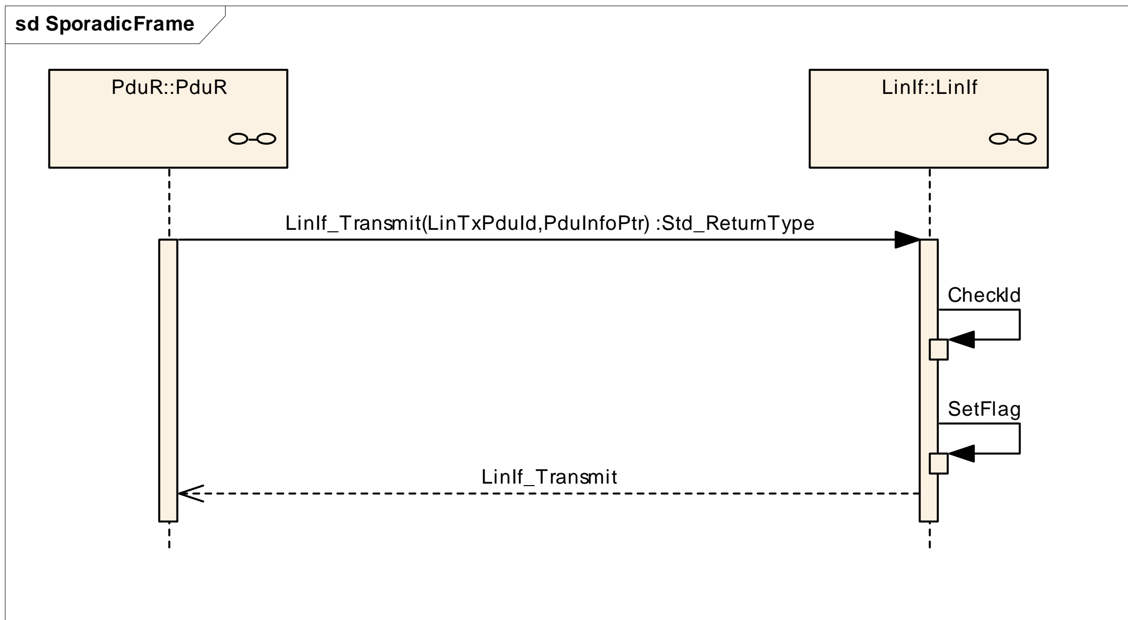

Figure 16 - Sporadic frame

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 will request 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 Manager. The LinIf_MainFunction will get 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 will respond to the event triggered frame header. The LinIf_MainFunction will recognize this and take no action since this is not considered to be a communication error.

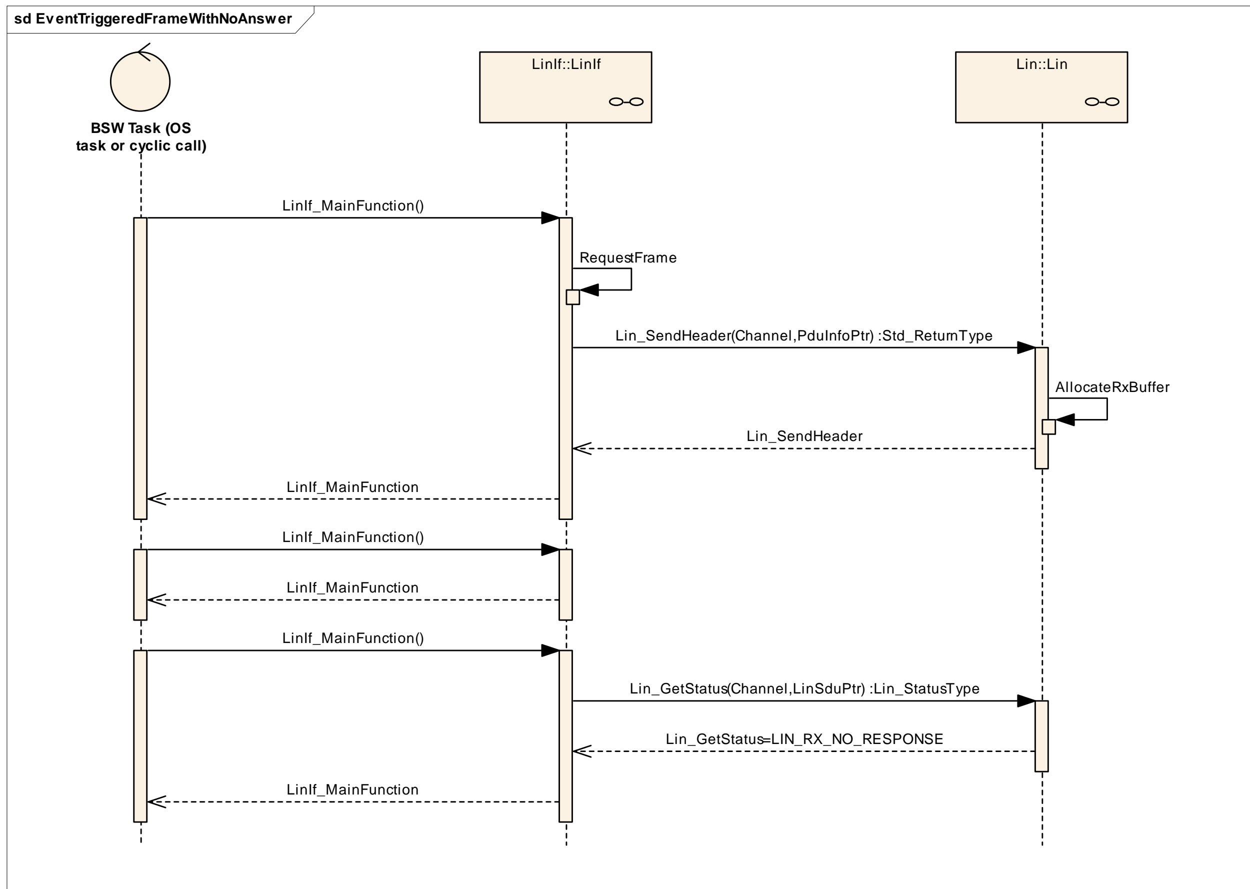

Figure 17 – Event triggered frame with no answer

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 will request 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 Manager. The LinIf_MainFunction will get 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.

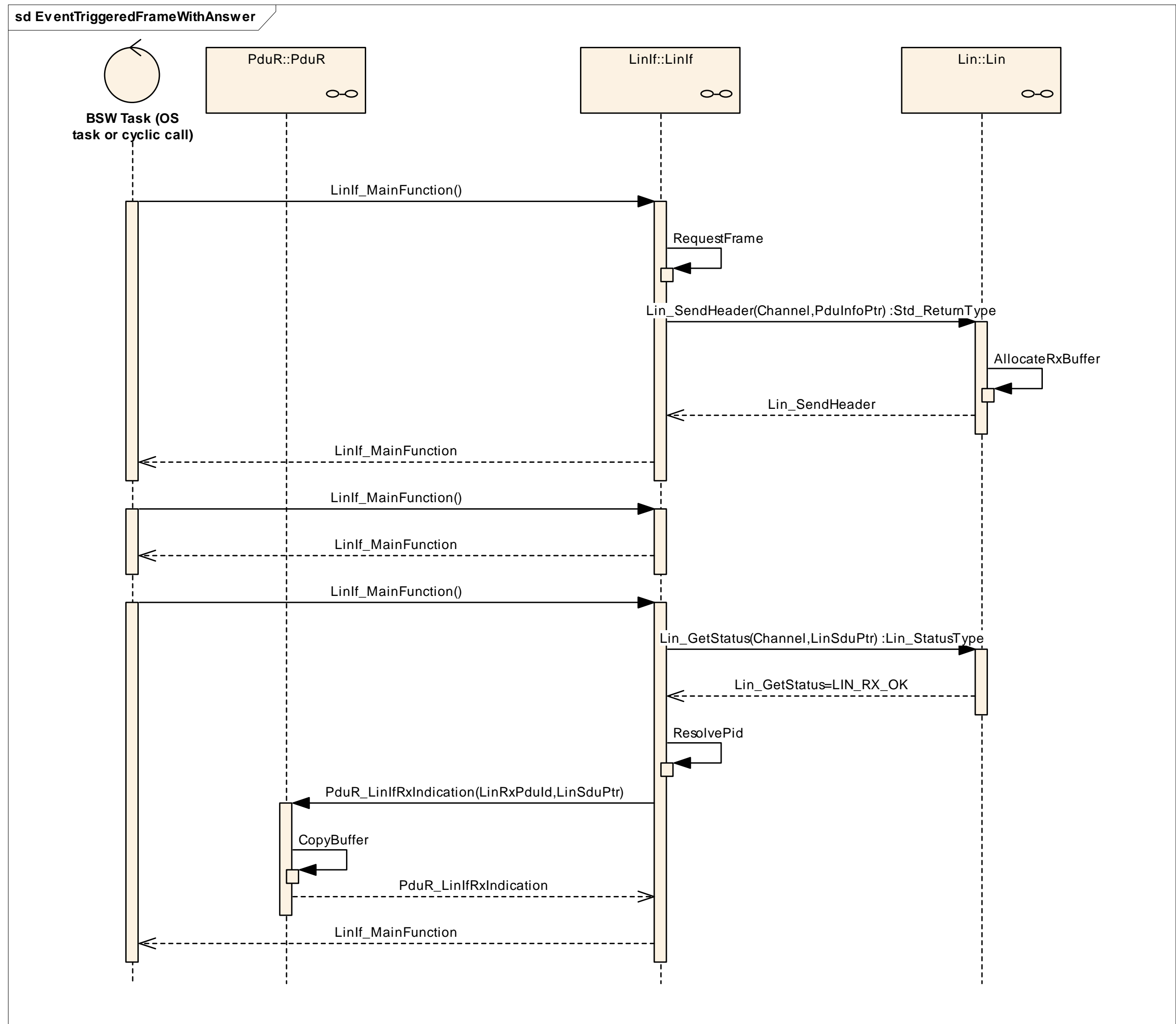

Figure 18 – Event triggered frame with answer (no collision)

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 slaves. This means that there will be a collision in the response field.

The first call of the LinIf_MainFunction will request 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 Manager. The LinIf_MainFunction will get 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 SetAllFramesPending will set 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)

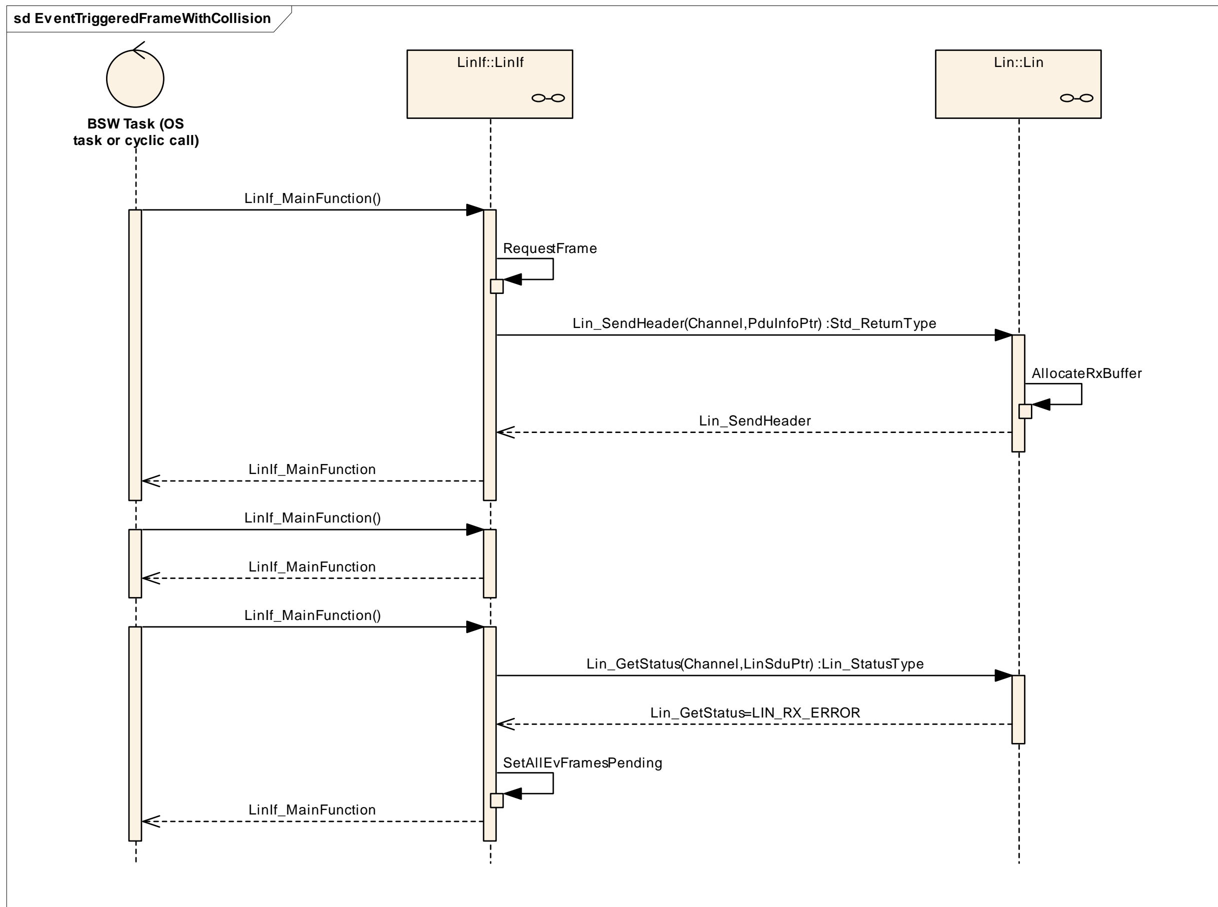

Figure 19 – Event triggered frame with collision

9.7 Transport Protocol Message transmission

The following diagram Figure 20 shows the transmission of a TP message. Both the initiation of the message and the continue buffer request is shown. The actual transmission of the MRF is not shown in the diagram, it has the same behavior as frame transmission 9.1.

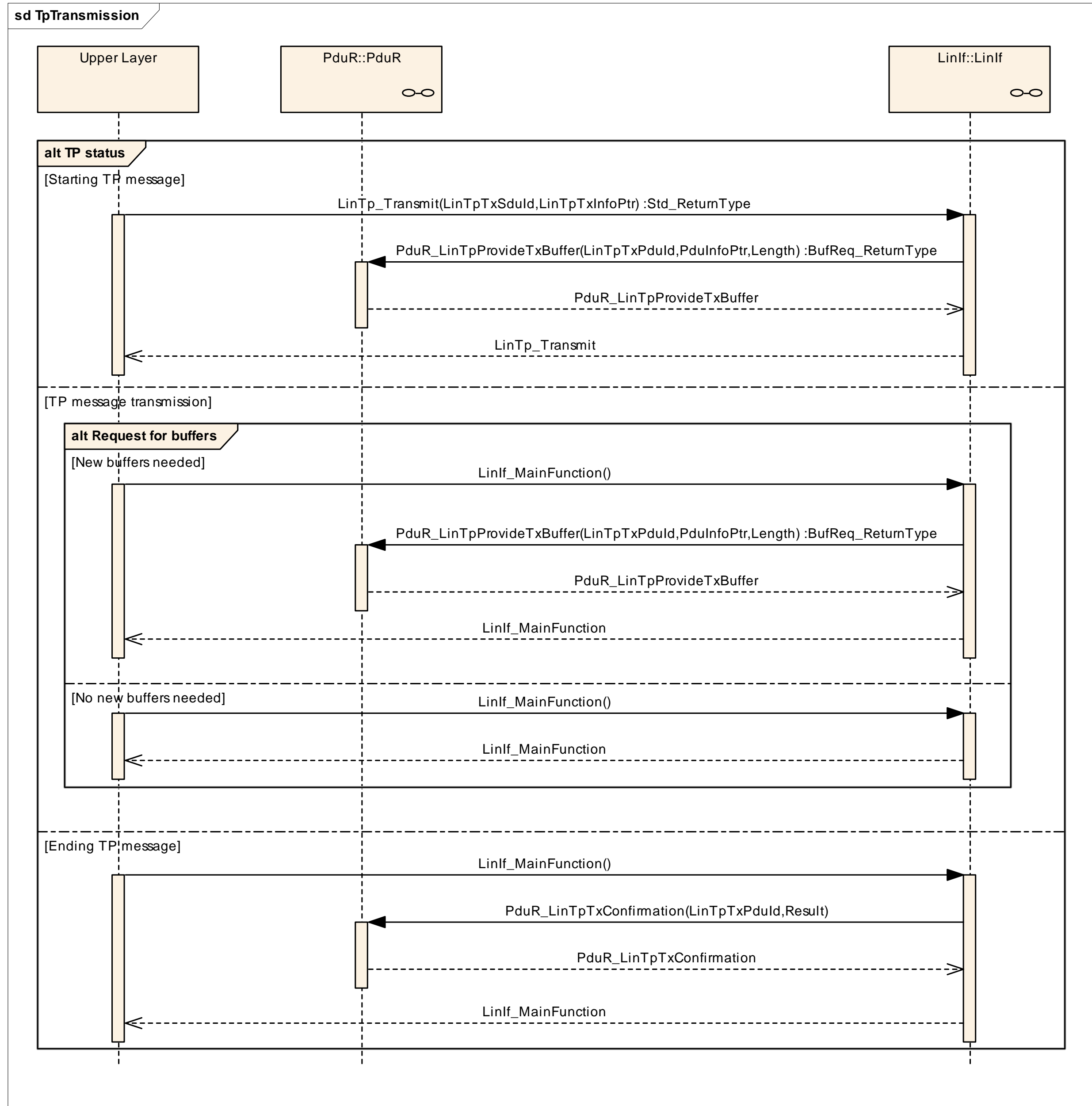

Figure 20 -Transport Protocol Transmission

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 continue buffer request and the finish of the message are shown. The actual reception of the SRFs is not shown in the diagram, 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 (CF or FF) 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.

- AUTOSAR confidential -

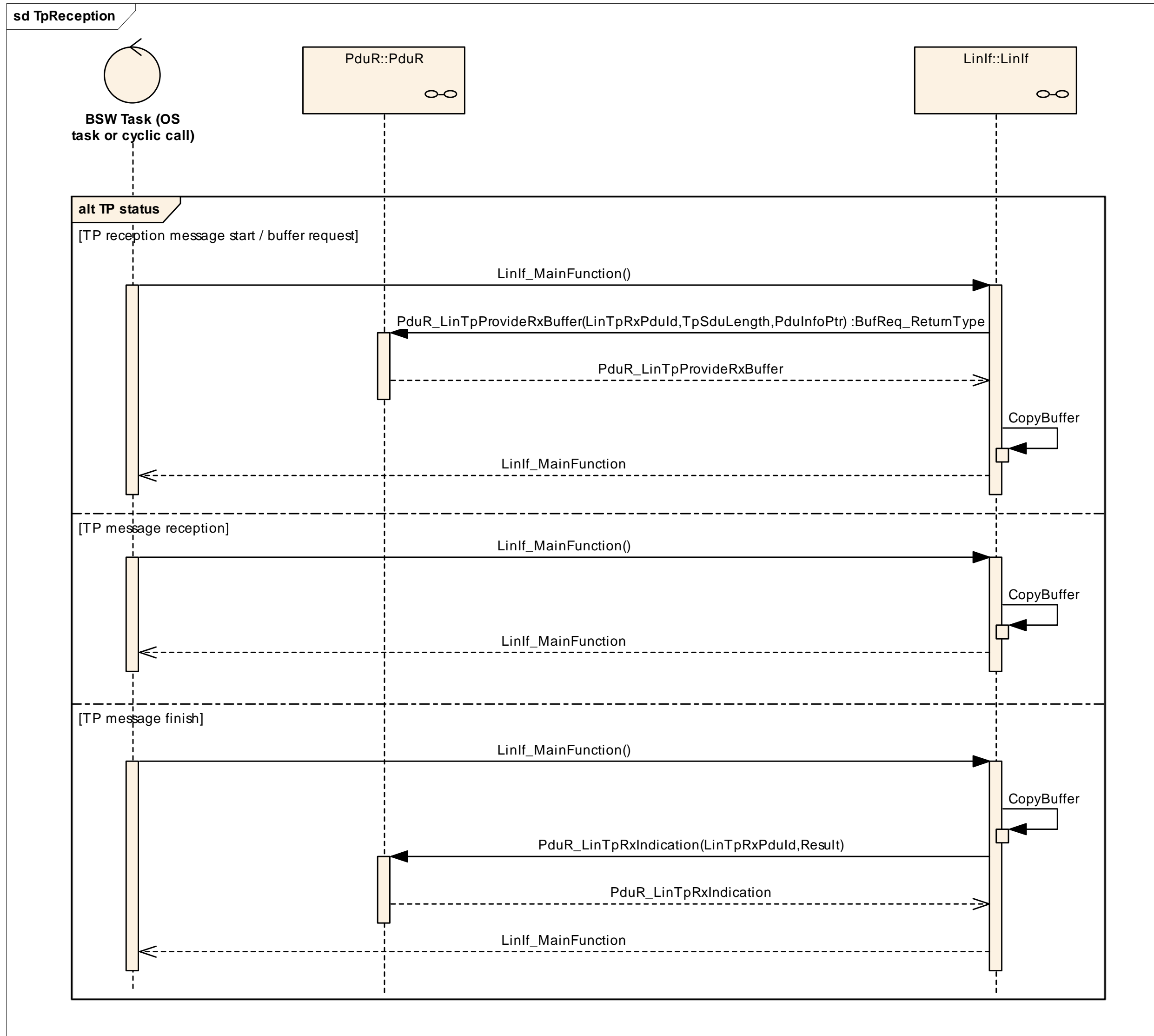

Figure 21 - Transport protocol message reception

9.9 Go-to-sleep process

This use-case in Figure 22 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 will be transmitted when the next schedule entry is due.

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

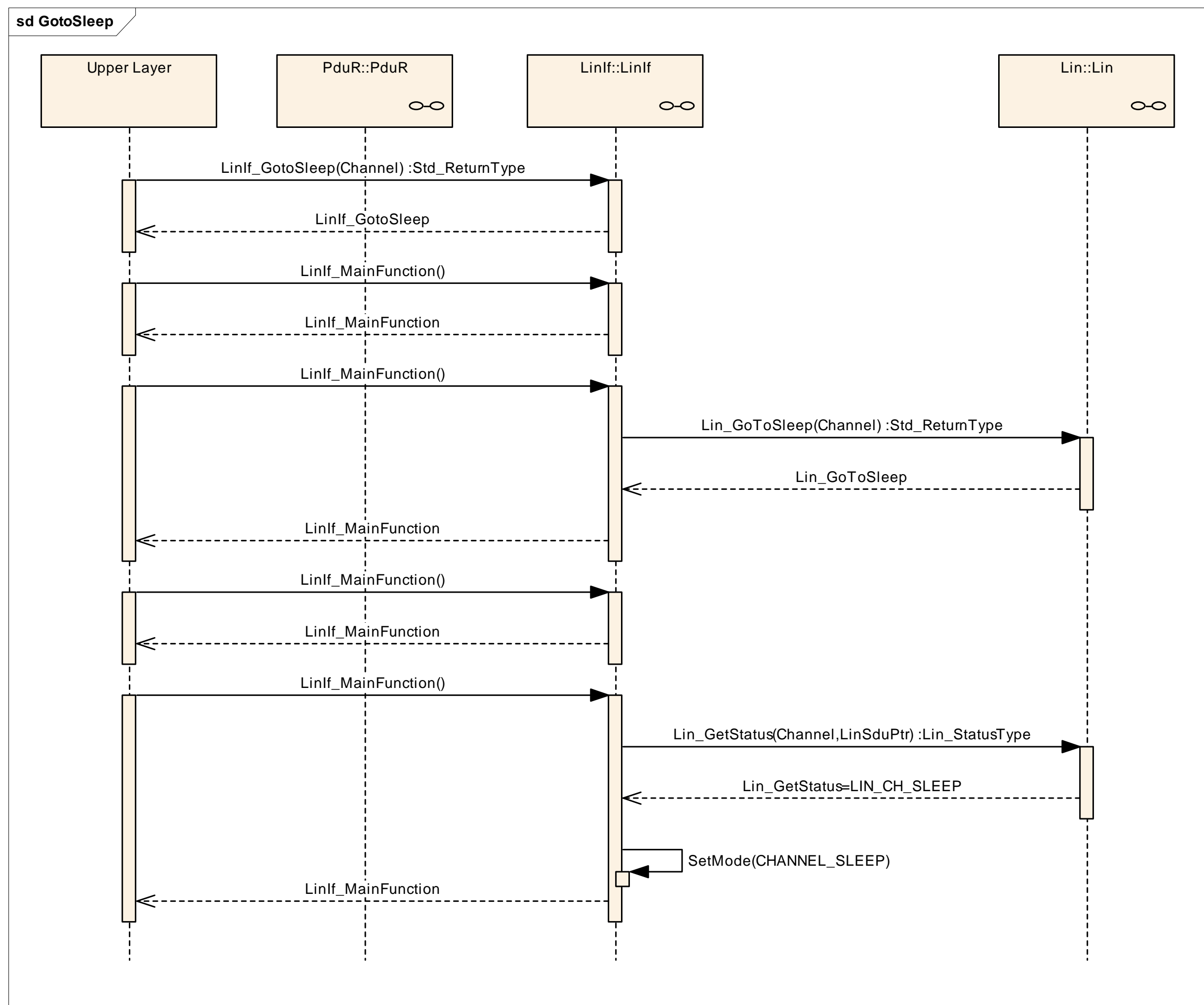

Figure 22 - Go-to-sleep–command process

9.10 Wake up request

There are two different use cases in Figure 23:

1. The first shows when a wake up by a Transceiver occurs.
2. The second shows when a wake up by a LIN Driver occurs.

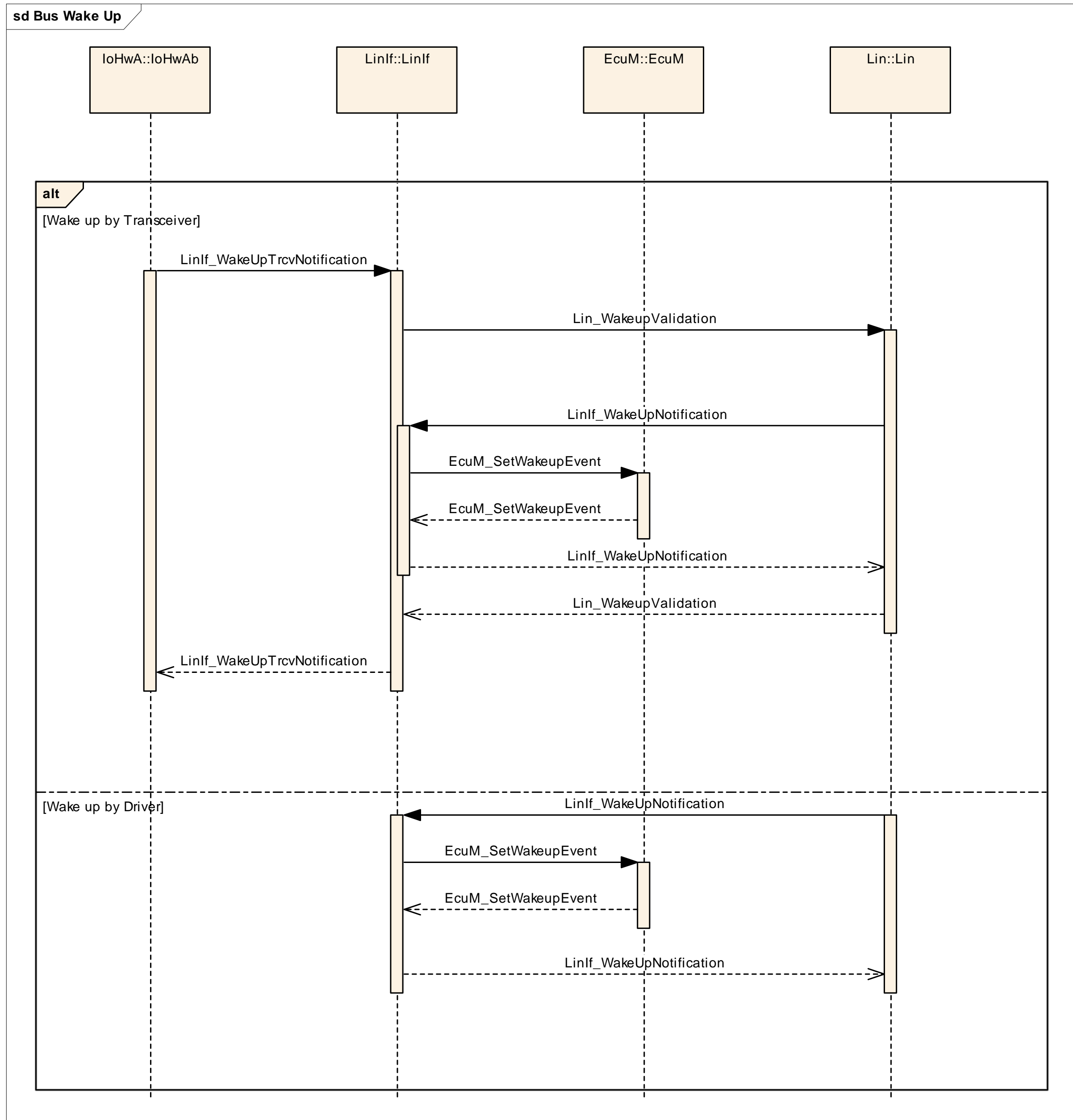

Figure 23 - Bus wake up

9.11 Internal wake up

There are two different use-cases in Figure 24:

1. The first shows when the Upper Layer request wake up of the LIN cluster AND the cluster is in sleep.
2. The first shows when the Upper Layer request wake up of the LIN cluster AND the cluster is awake.

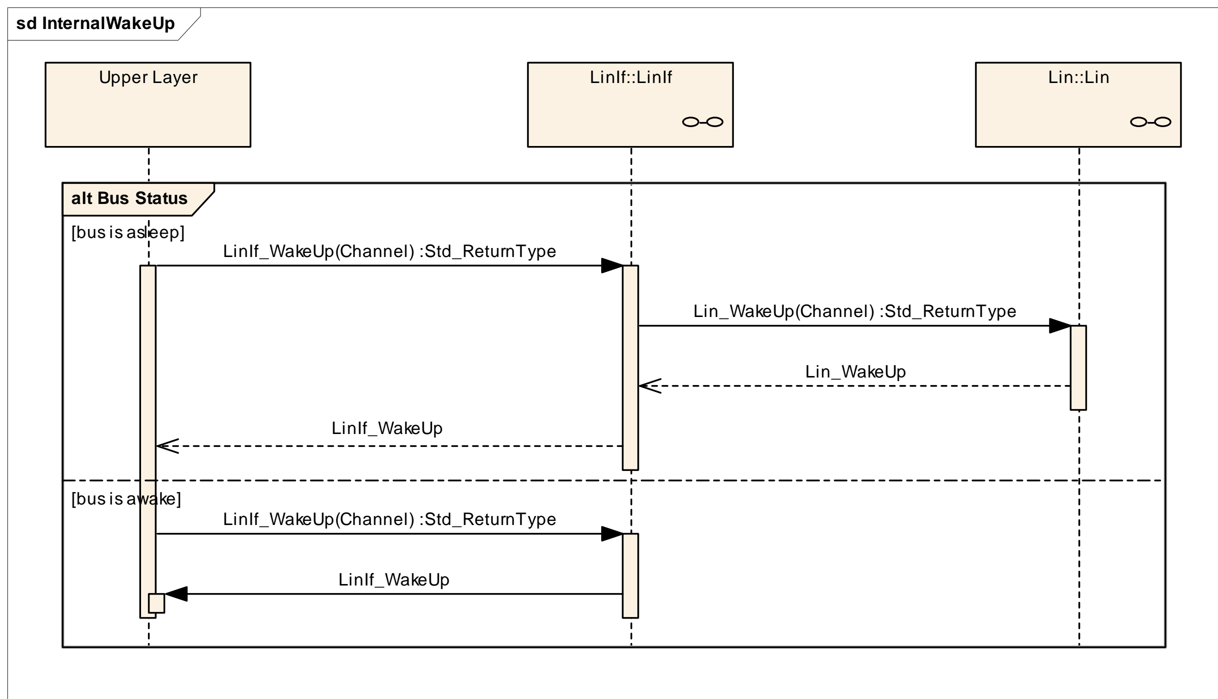

Figure 24 - Internal wake up

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. Chapter 10.4 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

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

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

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

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

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

10.2 Containers and configuration parameters

The following chapters summarize all configuration parameters. The detailed meanings of the parameters describe Chapters 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)

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)

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)

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 25 depicts the LIN Interface configuration.

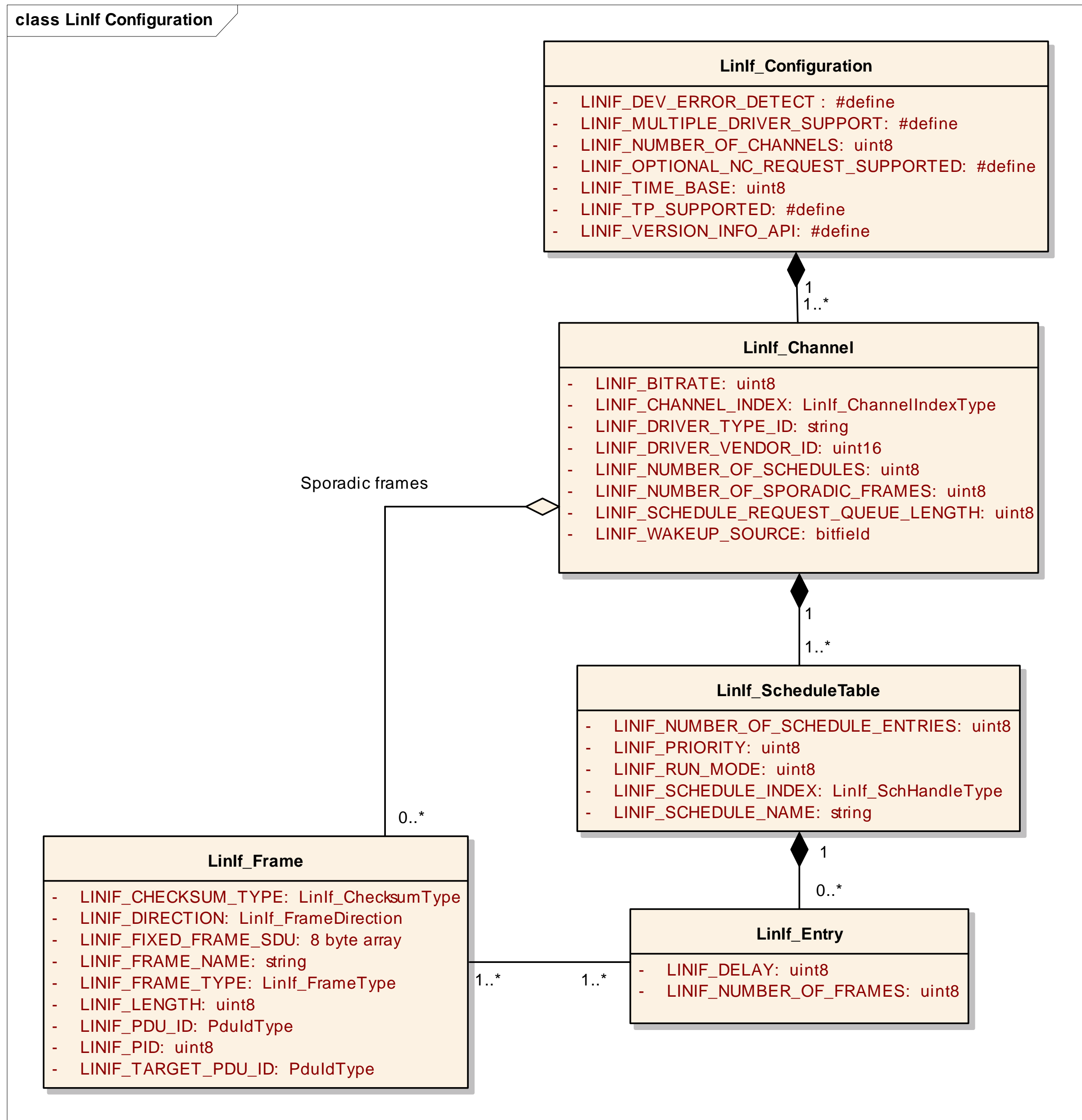

Figure 25 - LIN Interface configuration

SWS Item	LINIF370:
Container Name	LinIf_Configuration
Description	Singleton descriptor for the LIN Interface
Configuration Parameters	

Name	LINIF_DEV_ERROR_DETECT		
Description	Switches the Development Error Detection and Notification ON or OFF		
Type or Unit	#define		
Range	ON	Enabled	
	OFF	Disabled	
Configuration Class	**Pre-compile**	X	Variant 1, Variant 2 and Variant 3
	Link time	--	
	Post Build	--	
Scope	Module		
Dependency	--		

Name	LINIF_VERSION_INFO_API		
Description	Switches the LinIf_GetVersionInfo function ON or OFF		
Type or Unit	#define		
Range	ON	Enabled	
	OFF	Disabled	
Configuration Class	**Pre-compile**	X	Variant 1, Variant 2 and Variant 3
	Link time	--	
	Post Build	--	
Scope	Module		
Dependency	--		

Name	LINIF_NUMBER_OF_CHANNELS		
Description	Number of channels the LIN interface is connected to. Note that each LIN channel must be connected to one unique cluster (in CAN it is possible to have more than one channel to the same cluster)		
Type or Unit	uint8		
Range	1..255	Channel	
Configuration Class	**Pre-compile**	X	Variant 1
	Link time	X	Variant 2 and Variant 3
	Post Build	--	
Scope	Module		
Dependency	--		

Name	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.		
Type or Unit	#define		
Range	0	TP Not supported	
	1	TP supported	
Configuration Class	**Pre-compile**	X	Variant 1, Variant 2 and Variant 3
	Link time	--	
	Post Build	--	
Scope	Module		
Dependency	--		

Name	LINIF_DRIVER_CHANNEL_REF_CONFIG		
Description	This parameter is the symbolic reference to the LIN Driver Channel configuration		
Type or Unit	#define		
Range			
Configuration Class	**Pre-compile**	X	Variant 1, Variant 2 and Variant 3
	Link time	--	

	Post Build	--	
Scope	Module		
Dependency	--		

Name	LINIF_NC_OPTIONAL_REQUEST_SUPPORTED		
Description	States if the node configuration commands Assign NAD and Conditional Change NAD are supported		
Type or Unit	#define		
Range	ON	Used	
	OFF	Not used	
Configuration Class	***Pre-compile***	X	Variant 1, Variant 2 and Variant 3
	Link time	--	
	Post Build	--	
Scope	Module		
Dependency	--		

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

Name	LINIF_TIME_BASE		
Description	The time-base for this channel		
Type or Unit	uint8		
Range	0..255	In ms (Normally 2, 5 or 10 ms)	
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	x	Variant 2
	Post Build	x	Variant 3
Scope	Module		
Dependency	--		

Name	LINIF_MULTIPLE_DRIVER_SUPPORT		
Description	States if multiple drivers are included in the LIN Interface or not. 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 multiple drivers are not used.		
Type or Unit	#define		
Range	0	Multiple LIN drivers Not supported	
	1	Multiple LIN supported	
Configuration Class	***Pre-compile***	X	Variant 1, Variant 2 and Variant 3
	Link time	--	
	Post Build	--	
Scope	Module		
Dependency	--		

10.3.1 LinIf_Channel

SWS Item	LINIF364:
Container Name	LinIf_Channel
Description	Describes each LIN channel the LIN Interface is connected to
Configuration Parameters	

Name	LINIF_CHANNEL_INDEX		
Description	Index of the controller. To be able to interface to the correct hw.		
Type or Unit	LinIf_ChannelIndexType		
Range	--	See 8.2.1 LinIf_ChannelIndexType	
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	X	Variant 2, Variant 3
	Post Build	--	--
Scope	Module		
Dependency	--		

Name	LINIF_DRIVER_VENDOR_ID		
Description	Vendor ID of the LIN driver of this channel according to the AUTOSAR vendor list		
Type or Unit	uint16		
Range	--	--	
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	x	Variant 2, Variant 3
	Post Build	--	--
Scope	Module		
Dependency	--		

Name	LINIF_DRIVER_TYPE_ID		
Description	Identifier of the LIN driver. It is used to identify the correct driver together with the Vendor ID.		
Type or Unit	String		
Range	--		
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	x	Variant 2, Variant 3
	Post Build	--	--
Scope	Module		
Dependency	--		

Name	LINIF_BITRATE		
Description	Bitrate of the channel. This parameter is needed to calculate the delays in the schedule tables		
Type or Unit	Uint8		
Range	1000..20000	bps	
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	x	Variant 2
	Post Build	x	Variant 3
Scope	Module		
Dependency	--		

Name	LINIF_NUMBER_OF_SCHEDULES
Description	Number of schedules that are connected to this channel.

Type or Unit	uint8		
Range	1..255	Schedules (notes that each channel have always one predefined NULL_SCHEDULE)	
Configuration Class	*Pre-compile*	x	Variant 1
	Link time	x	Variant 2
	Post Build	x	Variant 3
Scope	Module		
Dependency	--		

Name	LINIF_SCHEDULE_REQUEST_QUEUE_LENGTH		
Description	Number of schedule requests the schedule table manager can handle for this channel		
Type or Unit	uint8		
Range	1..255	0 requests is not allowed	
Configuration Class	*Pre-compile*	x	Variant 1
	Link time	x	Variant 2
	Post Build	x	Variant 3
Scope	Module		
Dependency	--		

Name	LINIF_NUMBER_OF_SPORADIC_FRAMES		
Description	Number of sporadic frames listed to this channel		
Type or Unit	uint8		
Range	0..255	--	
Configuration Class	*Pre-compile*	x	Variant 1
	Link time	x	Variant 2
	Post Build	x	Variant 3
Scope	Module		
Dependency	--		

Name	LINIF_WAKEUP_SOURCE		
Description	Bit-field with one unique bit set for each channel. This is used when indicating a wakeup to the ECU state manager		
Type or Unit	EcuM_WakeupSourceType		
Range	bitfield	--	
Configuration Class	*Pre-compile*	x	Variant 1
	Link time	x	Variant 2
	Post Build	x	Variant 3
Scope	Module		
Dependency	--		

Included Containers		
Container Name	*Multiplicity*	*Scope / Dependency*
LinIf_ScheduleTable	1..*	Each schedule table can only be connected to one channel.

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
LinIf_Frame	0..*	All sporadic frames must be listed to be able to get the upper layer ID (N-SDU id) and PID conversion.

10.3.2 LinIf_ScheduleTable

SWS Item	LINIF365:
Container Name	LinIf_ScheduleTable
Description	Describes a schedule table
Configuration Parameters	

Name	LINIF_SCHEDULE_NAME		
Description	Optional schedule name used to cross-reference with a LDF		
Type or Unit	string		
Range	--	According to the LIN 2.0 specification	
Configuration Class	**Pre-compile**	x	Variant 1
	Link time	x	Variant 2
	Post Build	x	Variant 3
Scope	Module		
Dependency	--		

Name	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.		
Type or Unit	LinIf_SchHandleType		
Range	--	--	
Configuration Class	**Pre-compile**	x	Variant 1
	Link time	x	Variant 2
	Post Build	x	Variant 3
Scope	Module		
Dependency	--		

Name	LINIF_NUMBER_OF_SCHEDULE_ENTRIES		
Description	Number of schedules entries that the schedule contains, e.g. the NULL_SCHEDULE contains no schedule entries		
Type or Unit	uint8		
Range	0..255	Schedule entries	
Configuration Class	**Pre-compile**	x	Variant 1
	Link time	x	Variant 2
	Post Build	x	Variant 3
Scope	Module		
Dependency	--		

Name	LINIF_RUN_MODE	
Description	The schedule table can be executed in two different modes	
Type or Unit	uint8	
Range	0	RUN_CONTINUOUS
	1	RUN_ONCE
	2.255	reserved

Configuration Class	Pre-compile	x	Variant 1
	Link time	x	Variant 2
	Post Build	x	Variant 3
Scope	Module		
Dependency	--		

Name	LINIF_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 Priority 0 is reserved for the NULL__SCHEDULE		
Type or Unit	uint8		
Range	1..254	Only for RUN_ONCE	
	255	Only RUN_CONTINUOUS	
Configuration Class	Pre-compile	x	Variant 1
	Link time	x	Variant 2
	Post Build	x	Variant 3
Scope	Module		
Dependency	--		

Included Containers		
Container Name	Multiplicity	Scope / Dependency
LinIf_Entry	0..*	--

10.3.3 LinIf_Entry

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

Name	LINIF_DELAY		
Description	Delay to next frame in schedule table		
Type or Unit	uint16		
Range	0..65535	ms	
Configuration Class	Pre-compile	x	Variant 1
	Link time	x	Variant 2
	Post Build	x	Variant 3
Scope	Module		
Dependency	--		

Name	LINIF_NUMBER_OF_FRAMES	
Description	The number of frames this schedule entry contains A schedule entry may contain more than one frame if it is a sporadic frame or a event triggered frame	
Type or Unit	uint8	
Range	1..255	0 is irrelevant

Configuration Class	**Pre-compile**	x	Variant 1
	Link time	x	Variant 2
	Post Build	x	Variant 3
Scope	Module		
Dependency	--		

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
LinIf_Frame	1..*	Note that this is a association to a list of frames and not an aggregation. A frame may belong to more than one LinIf_Entry. An entry can contain more than one frame in case of event triggered frame or sporadic slot.

10.3.4 LinIf_Frame

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

Name	LINIF_FRAME_NAME		
Description	Optional frame name used to cross-reference with a LDF		
Type or Unit	string		
Range	--	According to the LIN 2.0 specification	
Configuration Class	**Pre-compile**	x	Variant 1
	Link time	x	Variant 2
	Post Build	x	Variant 3
Scope	Module		
Dependency	--		

Name	LINIF_PID		
Description	Protected ID of the LIN frame. There is no reason to calculate the Parity in run-time.		
Type or Unit	uint8		
Range	0..255	According to the LIN 2.0 specification	
Configuration Class	**Pre-compile**	x	Variant 1
	Link time	x	Variant 2
	Post Build	x	Variant 3
Scope	Module		
Dependency	--		

Name	LINIF_LENGTH		
Description	Length of the LIN SDU.		
Type or Unit	uint8		
Range	1..8	According to the length of a LIN 2.0 frame	
Configuration Class	**Pre-compile**	x	Variant 1
	Link time	x	Variant 2
	Post Build	x	Variant 3
Scope	Module		

- AUTOSAR confidential -

Dependency	--

Name	LINIF_FRAME_TYPE		
Description	Type of frame that is described (e.g. sporadic frame). Note that types 7-11 are the fixed MRF types. The sporadic slot is not found among the frame types. A sporadic slot is a set of sporadic frames.		
Type or Unit	uint8		
Range	0	Unconditional Frame	
	1	MRF	
	2	SRF	
	4	Event triggered frame	
	5	Sporadic frame	
	6	AssignFrameId	
	7	UnassignFrameId	
	8	AssignNAD	
	9	FreeFormat	
	10	Conditional Change NAD	
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	x	Variant 2
	Post Build	x	Variant 3
Scope	Module		
Dependency	--		

Name	LINIF_CHECKSUM_TYPE		
Description	Type of checksum that the frame is using.		
Type or Unit	uint8		
Range	0	Classic	
	1	Enhanced	
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	x	Variant 2
	Post Build	x	Variant 3
Scope	Module		
Dependency	--		

Name	LINIF_DIRECTION		
Description	Direction of the frame		
Type or Unit	uint8		
Range	0	TX	
	1	RX	
	2	SLAVE_TO_SLAVE	
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	x	Variant 2
	Post Build	x	Variant 3
Scope	Module		
Dependency	--		

Name	LINIF_PDU_ID		
Description	Identifier of the frame for the LIN Interface		
Type or Unit	PduIdType		
Range	--	According to the PduIdType	
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	x	Variant 2
	Post Build	x	Variant 3

- AUTOSAR confidential -

Scope	Module		
Dependency	--		

Name	LINIF_TARGET_PDU_ID		
Description	Identifier of the frame for the upper layer		
Type or Unit	PduIdType		
Range	--	According to the PduIdType	
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	x	Variant 2
	Post Build	x	Variant 3
Scope	Module		
Dependency	--		

Name	LINIF_FIXED_FRAME_SDU		
Description	In case this is a fixed frame this is the SDU (response). The Byte order shall be MSB first.		
Type or Unit	8 byte array		
Range	--	--	
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	x	Variant 2
	Post Build	x	Variant 3
Scope	Module		
Dependency	--		

10.4 LIN Transport Layer configuration

The Figure 26 shows the outline of the LIN Transport Protocol configuration.

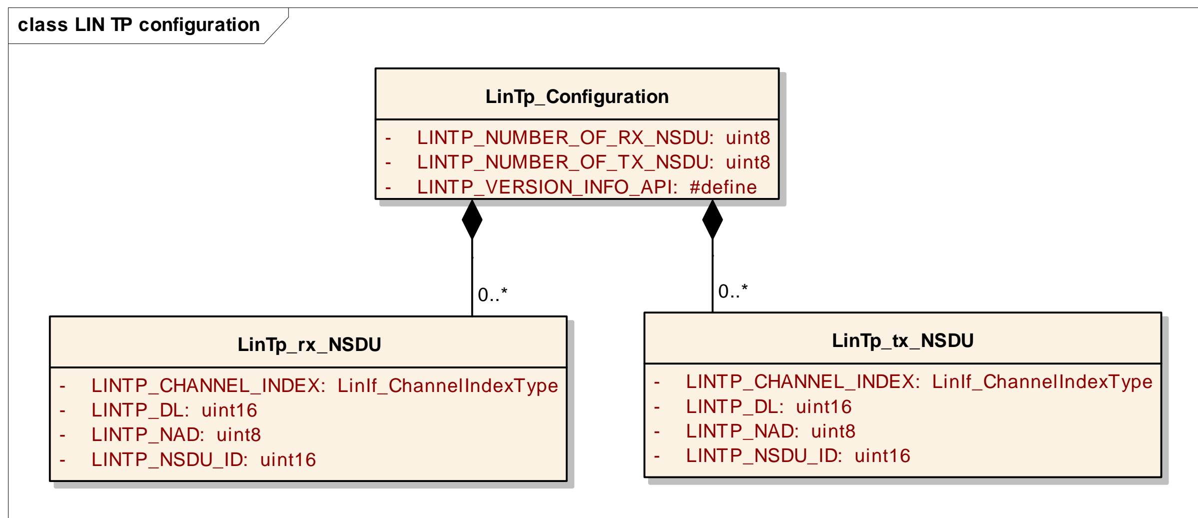

Figure 26 - LIN Transport Protocol configuration

SWS Item	LINIF425:
Container Name	LinTp_Configuration
Description	Singleton descriptor for the LIN Transport Protocol
Configuration Parameters	

Name	LINTP_VERSION_INFO_API		
Description	Switches the LinTp_GetVersionInfo function ON or OFF		
Type or Unit	#define		
Range	ON	Enabled	
	OFF	Disabled	
Configuration Class	Pre-compile	X	Variant 1, Variant 2 and Variant 3
	Link time	--	
	Post Build	--	
Scope	Module		
Dependency	--		

Name	LINTP_NUMBER_OF_RX_NSDU		
Description	Number of transport protocol messages that can be received for all channels this node is connected to		
Type or Unit	uint8		
Range	0..255	Number of RX N-SDU:s	
Configuration Class	Pre-compile	X	Variant 1
	Link time	X	Variant 2
	Post Build	X	Variant 3
Scope	Module		
Dependency	--		

Name	LINTP_NUMBER_OF_TX_NSDU		
Description	Number of transport protocol messages that can be transmitted for all channels this node is connected to		
Type or Unit	uint8		
Range	0..255	Number of TX N-SDU:s	
Configuration Class	Pre-compile	X	Variant 1
	Link time	X	Variant 2
	Post Build	X	Variant 3
Scope	Module		
Dependency	--		

Included Containers

Container Name	Multiplicity	Scope / Dependency
LinTp_rx_NSDU	0..*	The list of the received NSDU:s for this node. Note that the layout of the container is identical for rx and tx
LinTp_tx_NSDU	0..*	The list of the received NSDU:s for this node. Note that the layout of the container is identical for rx and tx

10.4.1 LinTp_xx_NSDU

This container is identical for tx and rx, hence the name xx.

SWS Item	LINIF428:
Container Name	LinTp_xx_NSDU
Description	For each received (xx=rx) and transmitted (xx=tx) N-SDU on any channel the node is connected to
Configuration Parameters	

Name	LINTP_NSDU_ID		
Description	The identifier of the Transport Protocol message. This id will be the one that is communicated with upper layers.		
Type or Unit	PduIdType		
Range	--	--	
Configuration Class	***Pre-compile***	X	Variant 1
	Link time	X	Variant 2
	Post Build	X	Variant 3
Scope	Module		
Dependency	--		

Name	LINTP_CHANNEL_INDEX		
Description	Index of the channel this N-SDU belongs to		
Type or Unit	LinIf_ChannelIndexType		
Range	--	--	
Configuration Class	***Pre-compile***	X	Variant 1
	Link time	X	Variant 2
	Post Build	X	Variant 3
Scope	Module		
Dependency	--		

Name	LINTP_NAD		
Description	A N-SDU transported on LIN is identified using the NAD for the specific slave		
Type or Unit	uint8		
Range	--	--	
Configuration Class	***Pre-compile***	X	Variant 1
	Link time	X	Variant 2
	Post Build	X	Variant 3
Scope	Module		
Dependency	--		

Name	LINTP_DL		
Description	Data Length Code of this RxNsdu. In case of variable length message, this value indicates the minimum data length. Depends on SF or FF N-SDU the value shall be limited to, respectively, 1 (since only extended addressing is used) and 4095. Note that this is not relevant for Tx. The reason for this is to have identical structures for Tx and Rx.		
Type or Unit	uint16		
Range	--	--	
	Pre-compile	X	Variant 1
	Link time	X	Variant 2

	Post Build	X	Variant 3
Scope	Module		
Dependency	--		

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.

SWS Item	LINIF280	
Information elements		
Information element name	**Type / Range**	**Information element description**
LINIF_VENDOR_ID	#define/uint16	Vendor ID of the dedicated implementation of this module according to the AUTOSAR vendor list
LINIF_MODULE_ID	#define/uint8	Module ID of this module from Module List
LINIF_AR_MAJOR_VERSION	#define/uint8	Major version number of AUTOSAR specification on which the appropriate implementation is based on.
LINIF_AR_MINOR_VERSION	#define/uint8	Minor version number of AUTOSAR specification on which the appropriate implementation is based on.
LINIF_AR_PATCH_VERSION	#define/uint8	Patch level version number of AUTOSAR specification on which the appropriate implementation is based on.
LINIF_SW_MAJOR_VERSION	#define/uint8	Major version number of the vendor specific implementation of the module. The numbering is vendor specific.
LINIF_SW_MINOR_VERSION	#define/uint8	Minor version number of the vendor specific implementation of the module. The numbering is vendor specific.
LINIF_SW_PATCH_VERSION	#define/uint8	Patch level version number of the vendor specific implementation of the module. The numbering is vendor specific.

11 Changes to Release 2

11.1 Deleted SWS Items

SWS Item	*Rationale*
LINIF199	Bug 13850
LINIF468	Bug 14617
LINIF403	Bug 14618
LINIF337	Bug 15483

11.2 Replaced SWS Items

SWS Item of Release 1	*replaced by SWS Item*	*Rationale*
--	--	--

11.3 Changed SWS Items

SWS Item	*Rationale*
LINIF434	Bug 13280
LINIF367	Bug 13816
LINIF428	Bug 12394
LINIF364	Bug 12420, Bug 14538
LINIF243	Bug 13083
LINIF198	Bug 13083
LINIF439	Bug 13850
LINIF370	Bug 14536
LINIF367	Bug 14537

11.4 Added SWS Items

SWS Item	*Rationale*
LINIF468	Bug 14803
LINIF469	Bug 15483
LINIF470	Bug 16041