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

Document Version	2.1.0
Document Status	Final
Part of Release	3.1
Revision	5

Document Change History

Date	Version	Changed by	Change Description
20.09.2010	2.1.0	AUTOSAR Administration	• Updated LINIF226 • Use PduInfoType for RxIndication, TriggerTransmit and Transmit APIs • Clarification of time parameters specified as float • Legal disclaimer revised
23.06.2008	2.0.1	AUTOSAR Administration	Legal disclaimer revised
15.11.2007	2.0.0	AUTOSAR Administration	• Interaction with LIN State Manager added • LIN Interface configuration reworked • Detection of LIN Response Error added • Wake-up concept reworked • Document meta information extended • Small layout adaptations made
31.01.2007	1.1.0	AUTOSAR Administration	• Start-up and Wake-up reworked for Transceiver needs. • File structure and requirements traceability adapted to new template. • Reworked configuration after integrator input. • Removed API's: LinIf_InitChannel() LinIf_DeInitChannel() • Legal disclaimer revised • Release Notes added • “Advice for users” revised • “Revision Information” added
11.05.2006	1.0.0	AUTOSAR Administration	Initial release

Disclaimer

Advice for users

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

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

Table of Contents

1 Introduction and functional overview

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

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

The base for this document is the LIN 2.0 specification [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

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

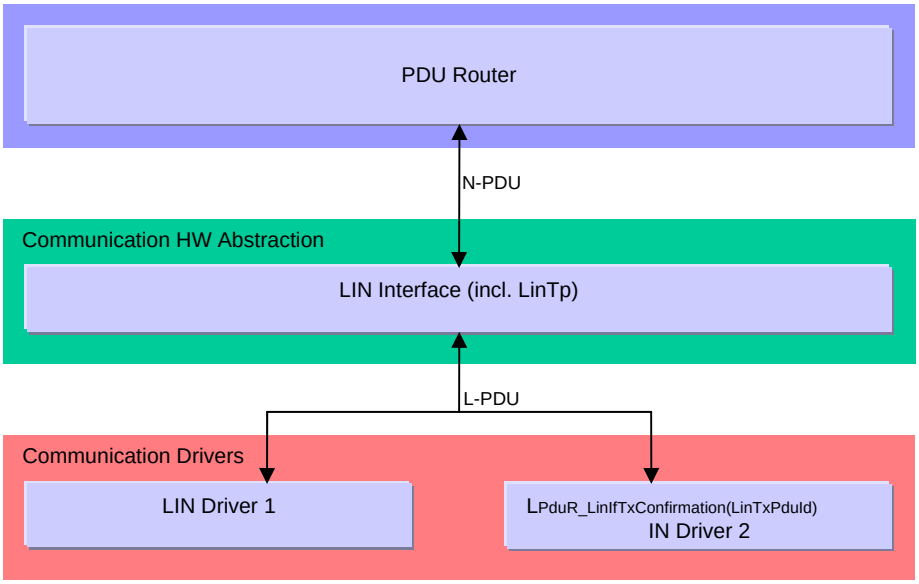

Figure 1 – AUTOSAR BSW software architecture (LIN relevant modules)

1.2 Functional overview

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

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

2 Acronyms and abbreviations

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

Abbreviation / Acronym:	***Description:***
API	Application Program Interface
CF	Continuous Frame in TP
DCM	Diagnostic Communication Manager
Delay	The time between 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
AUTOSAR_BasicSoftwareModules.pdf

[2] Layered Software Architecture,
AUTOSAR_LayeredSoftwareArchitecture.pdf

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

[4] Specification of Standard Types
AUTOSAR_SWS_StandardTypes.pdf

[5] Specification of Development Error Tracer
AUTOSAR_SWS_DET.pdf

[6] Requirements on LIN
AUTOSAR_SRS_LIN.pdf

[7] Specification of LIN Driver
AUTOSAR_SWS_LIN_Driver.pdf

[8] Specification of Diagnostics Event Manager
AUTOSAR_SWS_DEM.pdf

[9] Specification of ECU Configuration
AUTOSAR_ECU_Configuration.pdf

[10] Specification of ECU State Manager
AUTOSAR_SWS_ECU_StateManager.pdf

[11] Specification of the BSW Scheduler
AUTOSAR_SWS_BSW_Scheduler.pdf

[12] Specification of LIN State Manager

AUTOSAR_SWS_LIN_StateManager.pdf

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

3.2 Related standards and norms

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

4 Constraints and assumptions

4.1 Limitations

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

4.2 Applicability to car domains

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

5 Dependencies to other modules

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

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

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

5.1 Upper layers

5.1.1 PDU Router

The LIN Interface connects to the PDU Router for transmission and reception of frames. It is assumed that the PDU router or a module above it is responsible for the copying of the data of the frames for reception and transmission. Additionally, the PDU router handles the TP messages buffers either as complete or fragmented messages.

5.1.2 BSW Scheduler

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

5.1.3 Operating System

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

5.1.4 Module DET (Development Error Tracer)

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

5.1.5 Module DEM (Diagnostic Event Manager)

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

5.1.6 Module ECU State Manager

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

1. When a bus wake-up is detected by the LIN Interface, the ECU state manager is notified.
2. The ECU state manager initializes the LIN Interface.

5.1.7 Module LIN State Manager

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

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

5.2 Lower layers

5.2.1 LIN Driver

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

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

- Transmission of the header part of a frame (Lin_SendHeader). It is assumed that this primitive also tells the direction of the frame response (transmit, receive or slave-to-slave communication)
- Transmission of the response part of a frame (Lin_SendResponse).
- Transmission of the go-to-sleep-command (Lin_GoToSleep)
- Query of reception of the response part of a frame (Lin_GetStatus). The following cases are assumed to be distinguished:
    - Successful reception/transmission.
    - No reception.
    - Erroneous reception/transmission (framing error, bit error, checksum error).
    - Ongoing reception - at least one response byte has been received, but the checksum byte has not been received.
    - Ongoing Transmission.
    - Channel In sleep (the go-to-sleep command has been successfully transmitted)

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

5.3 File structure

5.3.1 Code file structure

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

LINIF241: The code file structure shall not be defined within this specification completely. At this point, it shall be pointed out that the code-file structure shall include the following files named:

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

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

5.3.2 Header file structure

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

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

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

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

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

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

LINIF457: The LIN Interface shall include the EcuM.h file to get access to the wake-up notification API call from the ECU state manager.

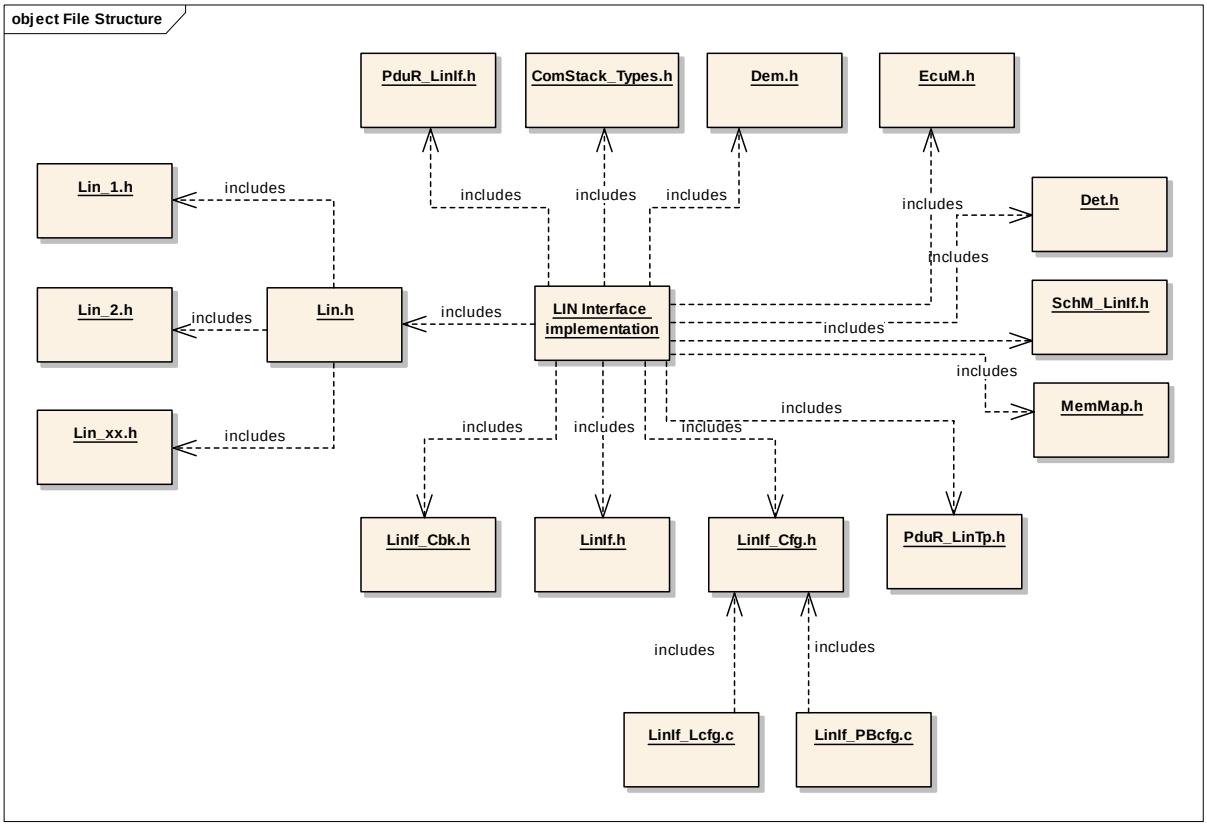

Figure 2 - Header file structure

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 file Dem.h.

LINIF434: The file Lin.h shall include the LIN_xx.h files, which describe the external API and configuration of each LIN Driver

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

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

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

LINIF453: The LIN Interface shall include the file PduR_LinIf.h.

LINIF455: The LIN Interface shall include the file Det.h if the configuration parameter LinIfDevErrorDetect is enabled.

LINIF458: The LIN Interface shall include the file ComStackTypes.h.

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

6 Requirements traceability

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

The following two 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	Chapter 5
[BSW00387] Specify the configuration class of callback function	Chapter 8.3
[BSW00388] Introduce containers	Chapter 10.2
[BSW00389] Containers shall have names	Chapter 10.2
[BSW00390] Parameter content shall be unique within the module	Chapter 10.2
[BSW00391] Parameter shall have unique names	Chapter 10.2
[BSW00392] Parameters shall have a type	Chapter 10.2
[BSW00393] Parameters shall have a range	Chapter 10.2
[BSW00394] Specify the scope of the parameters	Chapter 10.2
[BSW00395] List the required parameters (per parameter	Chapter 10.2
[BSW00396] Configuration classes	Chapter 10.2
[BSW00397] Pre-compile-time parameters	Chapter 10.2
[BSW00398] Link-time parameters	Chapter 10.2
[BSW00399] Loadable Post-build time parameters	Chapter 10.2
[BSW00400] Selectable Post-build time parameters	Chapter 10.2
[BSW00402] Published information	[LINIF280]
[BSW00375] Notification of wake-up	LINIF378:
[BSW101] Initialization interface	[LINIF198]
[BSW00416] Sequence of Initialization	[LINIF198]
[BSW00406] Check module initialization	[LINIF380]

	[LINIF376]
[BSW168] Diagnostic Interface of SW components	LIN Interface does not offer a diagnostic interface
[BSW00407] Function to read out published parameters	LIN If [LINIF340] and LIN Tp [LINIF352]
[BSW00423] Usage of SW-C template to describe BSW modules with AUTOSAR Interfaces	LinIf has no AUTOSAR interfaces
[BSW00424] BSW main processing function task allocation	LinIf does not provide any task handling
[BSW00425] Trigger conditions for schedulable objects	Scheduling frames are a property of the LIN 2.0 specification that is inherited in 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	Chapter 5
[BSW00415] User dependent include files	[LINIF243] [LINIF244] [LINIF245]
[BSW164] Implementation of interrupt service routines	No required interrupt functions
[BSW00325] Runtime of interrupt service routines	No required interrupt functions
[BSW00326] Transition from ISRs to OS tasks	No required interrupt functions
[BSW00342] Usage of source code and object code	The LinIf configuration enables both the source code and the binary way. See chapter 10
[BSW00343] Specification and configuration of time	[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 the construction and not the design (i.e. SWS). The API chapter 8 is following MISRA C
[BSW00300] Module naming convention	chapter 5.3
[BSW00413] Accessing instances of BSW modules	[LINIF197:] LINIF469:
[BSW00347] Naming separation of different instances of BSW drivers	LinIf is not a driver
[BSW00305] Self-defined data types naming convention	Chapter 0
[BSW00307] Global variables naming convention	The naming of parameters is made in chapter 10
[BSW00310] API naming convention	Chapter 8.2
[BSW00373] Main processing function naming convention	[LINIF384]
[BSW00327] Error values naming convention	[LINIF376]
[BSW00335] Status values naming convention	[LINIF039] [LINIF290] [LINIF316] [LINIF319]
[BSW00350] Development error detection keyword	[LINIF268]
[BSW00408] Configuration parameter naming convention	Chapter 10.2
[BSW00410] Compiler switches shall have defined values	Construction requirement and not a design requirement
[BSW00411] Get version info keyword	[LINIF279]
[BSW00436] Module Header File Structure for the Basic Software Memory Mapping	[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 8.3
[BSW00348] Standard type header	Chapter 5.3
[BSW00353] Platform specific type header	Chapter 5.3
[BSW00361] Compiler specific language extension header	Chapter 5.3
[BSW00301] Limit imported information	Chapter 5.3
[BSW00302] Limit exported information	[LINIF242]
[BSW00328] Avoid duplication of code	Complete 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 8.2
[BSW00377] Module specific API return types	No module specific return types needed. See chapter 8.
[BSW00304] AUTOSAR integer data types	Only uint8 and uint16 are used
[BSW00355] Do not redefine AUTOSAR integer data types	No redefine made
[BSW00378] AUTOSAR boolean type	No Boolean return types used
[BSW00306] Avoid direct use of compiler and platform specific keywords	No platform specific keywords are used.
[BSW00308] Definition of global data	No global data is required.
[BSW00309] Global data with read-only constraint	No global data is required.
[BSW00371] Do not pass function pointers via API	Function pointers not used, Chapter 8.2
[BSW00358] Return type of init() functions	[LINIF198] [LINIF350]
[BSW00414] Parameter of init function	[LINIF371]
[BSW00376] Return type and parameters of main processing functions	[LINIF384]
[BSW00359] Return type of callback functions	[LINIF378]

[BSW00360] Parameters of callback functions	[LINIF378]
[BSW00329] Avoidance of generic interfaces	Generic interfaces are not used
[BSW00330] Usage of macros / inline functions instead of functions	No restriction
[BSW00331] Separation of error and status values	NOT APPLICABLE for LinIf SWS
[BSW009] Module User Documentation	SWS template 1.19 is used
[BSW00401] Documentation of multiple instances of configuration parameters	Chapter 10.2
[BSW172] Compatibility and documentation of scheduling strategy	Chapter 8
[BSW010] Memory resource documentation	NOT APPLICABLE for LinIf SWS
[BSW00333] Documentation of callback function context	NOT APPLICABLE for LinIf SWS
[BSW00374] Module vendor identification	[LINIF280]
[BSW00379] Module identification	[LINIF280]
[BSW003] Version identification	NOT APPLICABLE for LinIf SWS
[BSW00318] Format of module version numbers	[LINIF280]
[BSW00321] Enumeration of module version numbers	NOT APPLICABLE for LinIf SWS
[BSW00341] Microcontroller compatibility documentation	NOT APPLICABLE for LinIf SWS
[BSW00334] Provision of XML	NOT APPLICABLE for LinIf SWS
[BSW00435] Header File Structure for the Basic Software Scheduler	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 wake-up during transition to sleep-mode	Chapter 7.3.4.1
[BSW01567] Compatibility to LIN 2.0 protocol specification	[LINIF248]
[BSW01551] Multiple LIN channel support for interface	[LINIF386]
[BSW01568] Hardware independence	[LINIF129]
[BSW01569] LIN Interface initialization	LINIF198
[BSW01570] Selection of static configuration sets	[LINIF371]
[BSW01564] Schedule Table Manager	[LINIF202]
[BSW01546] Schedule Table Handler	[LINIF384]
[BSW01561] Main function	[LINIF384]
[BSW01549] Timer service for Scheduling	[LINIF223]
[BSW01571] Transmission request service	[LINIF201]
[BSW01514] Wake-up notification support	LINIF378:
[BSW01515] API to wake-up by upper layer to LIN Interface	[LINIF205]
[BSW01502] RX indication and TX confirmation call-backs	[LINIF128]
[BSW01558] Check successful communication	[LINIF033]
[BSW01527] Notification for missing or erroneous receive LIN-PDU	LINIF465 LINIF466
[BSW01523] API to send the LIN to sleep-mode	[LINIF204]
[BSW01565] Compatibility to LIN 2.0 protocol specification	LIN driver requirement
[BSW01553] Basic Software SPAL General requirements	LIN driver requirement
[BSW01552] Hardware abstraction LIN	LIN driver requirement
[BSW01503] Frame based API for send and received data	LIN driver requirement
[BSW01555] LIN Interface shall poll the LIN Driver for transmit/receive notifications	LIN driver requirement
[BSW01547] Support of standard UART and LIN optimised	LIN driver requirement

[BSW01572] LIN driver initialization	LIN driver requirement
[BSW01573] Selection of static configuration sets	LIN driver requirement
[BSW01563] Wake-up Notification	LIN driver requirement
[BSW01556] Multiple LIN channel support for driver	LIN driver requirement
[BSW01566] Transition to sleep-mode mode	LIN driver requirement
[BSW01524] Support of reduced power operation mode	LIN driver requirement
[BSW01526] Error notification	LIN driver requirement
[BSW01533] Usage of LIN 2.0 specification	[LINIF313]
[BSW01540] LIN Transport Layer Initialization	[LINIF350]
[BSW01545] LIN Transport Layer Availability	[LINIF098]
[BSW01534] Concurrent connection configuration	[LINIF062]
[BSW01574] Multiple Transport Layer instances	[LINIF314]
[BSW01539] Transport connection properties	Chapter 10.4
[BSW01544] Error handling	Chapter 7.5.2.5 and chapter 7.5.2.7

7 Functional specification

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

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

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

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

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

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

7.1 Frame Transfer

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

7.1.1 Frame types

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

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

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

7.1.1.1 Unconditional frame

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

7.1.1.2 Event triggered frame

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

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

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

7.1.1.3 Sporadic frame

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

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

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

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

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

Upper layers decide the transmission of a sporadic frame. Therefore an API call must be available that sets the sporadic frame pending for transmission.

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

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

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

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

7.1.1.4 Diagnostic Frames MRF and SRF

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

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

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

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

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

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

7.1.1.5 User-defined frames

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

7.1.1.6 Reserved frames

The LIN 2.0 specification does not allow reserved frames.

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

7.1.2 Frame reception

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

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

7.1.2.1 Header

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

7.1.2.2 Response

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

7.1.2.3 Status check

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

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

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

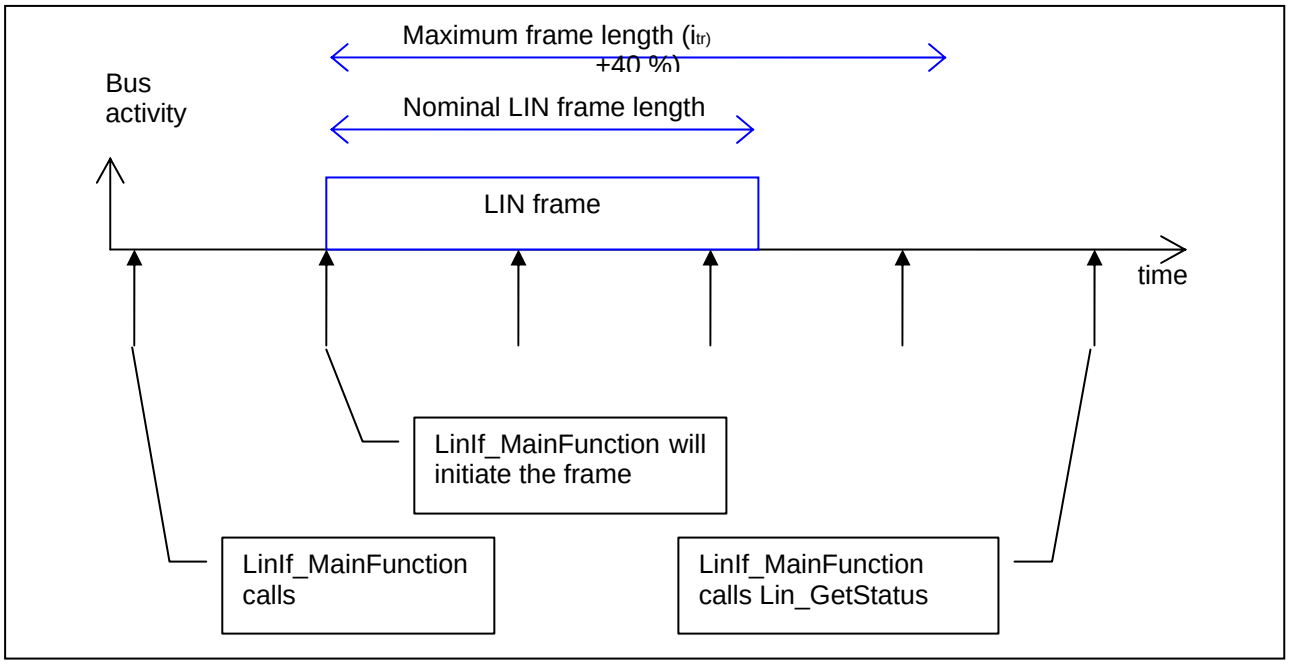

Figure 3 - Lin_GetStatus call example

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

LINIF033: When the LIN Interface determines the LIN Driver module's status to be LIN_RX_OK, the LIN Interface shall issue a PduR_LinIfRxIndication callback with successful reception of frame.

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

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

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

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

LINIF254: When the LIN Interface has received an unconditional frame and determined the LIN Driver module's status to be LIN_RX_ERROR, the LIN Interface shall consider the received frame as lost.

LINIF466: If the LIN Interface has determined the LIN Driver module's status as 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), it shall

consider the next frame which will be received by the LIN Interface, as lost. Therefore, the LIN Interface shall raise the production error code LINIF_E_RESPONSE if it is not an event-triggered 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: The LIN Interface shall not report a header error to the upper layers when the return code of the LIN Driver module's function Lin_GetStatus is LIN_TX_HEADER_ERROR.

7.1.3 Frame transmission

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

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

7.1.3.1 Header

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

LINIF227: When the LIN Interface has transmitted an event triggered frame header and determined the LIN Driver module's status to be LIN_RX_BUSY or LIN_RX_ERROR, the LIN Interface shall flag all unconditional frames connected to this frame for polling (see the LIN 2.0 specification how this is realized).

7.1.3.2 Response

LINIF225: After the function Lin_SendHeader has returned, the LIN Interface shall call the PDU router's 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: If the return code of function PduR_LinIfTriggerTransmit is E_OK, the LIN Interface shall call the LIN Driver module's function Lin_SendResponse to provide the LIN Driver a pointer to the data part. If the return code is E_NOT_OK, the LIN Interface shall not transmit the data part of the frame.

7.1.3.3 Status check

LINIF128: If the return code of the Lin_GetStatus function is LIN_TX_OK, the LIN Interface shall issue a PduR_LinIfTxConfirmation callback.

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

LINIF465: If, just before a new frame is transmitted, the return code of the Lin_GetStatus function is is LIN_TX_BUSY, the LIN Interface shall consider the old frame as lost and raise the production error 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 the LIN Interface has transmitted a sporadic frame successfully, it shall reset the pending flag.

7.1.4 Slave-to-slave communication

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

7.1.4.1 Header

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

7.1.4.2 Response

LINIF417: The LIN Interface shall not be involved in the slave-to-slave communication, neither in transmission or reception of the response.

7.1.4.3 Status check

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

7.2 Schedules

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

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

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

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

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

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

LINIF263: The schedule table NULL_SCHEDULE shall contain no frames.

7.2.1 LinIf_MainFunction

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

LINIF474: The environment of the LIN Interface shall call the function LinIf_MainFunction periodically with a given period.

LINIF223: The period of the invocation of the function LinIf_MainFunction shall be based on the time-bases (configuration parameter `LinIfTimeBase`) which are defined for different LIN clusters. The LIN Interface shall use the Greatest Common Factor (GCF) of the time-bases.

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

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

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

7.2.2 Schedule table manager

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

LINIF463: The priority range of the schedule table manager shall be 0 to 255 where 0 represents the highest priority.

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

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

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

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

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

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

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

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

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

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

LINIF444: If the LIN Interface's environment is requesting a NULL_SCHEDULE (or set in case of initialization or sleep) and the requested queue is not of the type NULL_SCHEDULE, the schedule table manager of the LIN Interface shall flush the schedule table manager queue. The schedule table manager of the LIN Interface shall change the requested queue to NULL_SCHEDULE when the schedule entry is due (even if the current is RUN_ONCE).

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

The LIN Interface allows changing of the current schedule table to another one. The 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 LIN Interface shall select a new schedule table for execution during the next schedule entry if the current schedule is RUN_CONTINUOUS.

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

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

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 continues even if the schedule table is changed (if the new schedule table contains the event-triggered frame of course)

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

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

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

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

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

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 are 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 are executed as shown in Figure 5.

ReadById	ReadById	CheckSensors	Null-schedule	…

Figure 5 - Schedule table execution example

7.3 Network management

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

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

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

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

7.3.1 Node Management

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

7.3.1.1 LIN Interface state-machine

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

The state-machine is depicted in Figure 6.

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

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

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

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

7.3.1.2 LIN channel sub-state-machine

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

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

LINIF440: The LIN channel sub-state-machine shall have the state CHANNEL_UNINIT.

LINIF475: In the LIN channel sub-state CHANNEL_UNINIT the LIN Interface shall be initialized but the corresponding LIN channel shall not be initialized.

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

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

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

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

LINIF507: The LIN Interface shall transit from CHANNEL_UNINIT to the CHANNEL_OPPERATIONAL when the LinIfChannelInit function is called.

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

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

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

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

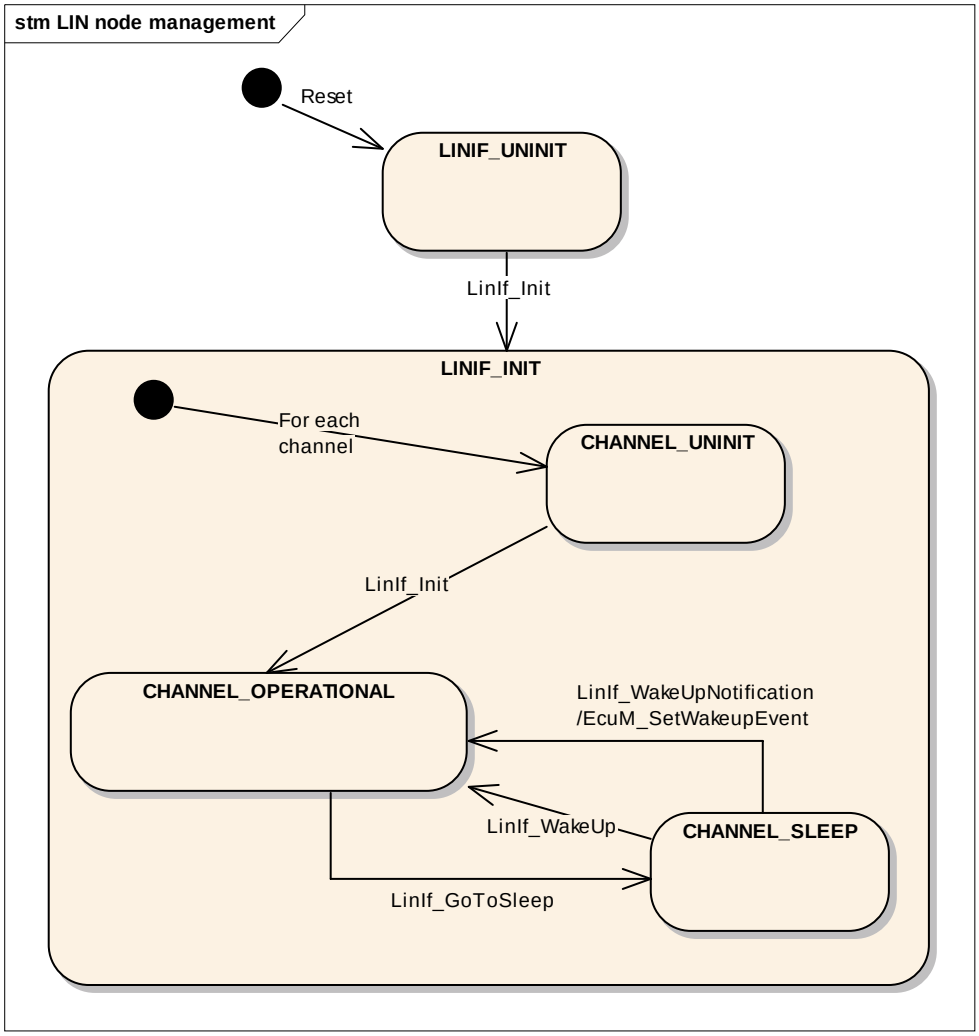

Figure 6 - LIN Interface state-machine and LIN Interface channel sub-state-machine

7.3.2 Initialization process

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.

LINIF453: When processing the Go-to-sleep-command, the function LinIf_MainFunction shall call the function Lin_GoToSleep instead of the scheduled frame latest when the next schedule entry is due.

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

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

LINIF454: When processing the Go-to-sleep-command, the function LinIf_MainFunction shall call the function Lin_GetStatus of the LIN Driver module. When the return code of the function Lin_GetStatus is not LIN_CH_SLEEP, the function LinIf_MainFunction shall set the channel state of the affected channel to CHANNEL_SLEEP and switch the current used schedule table to the NULL_SCHEDULE. In this case, The go-to-sleep-command transmission has failed.

The reason to do it 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 wake-up request, the state must be set to CHANNEL_SLEEP.

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

7.3.4 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 or a bus wake-up is detected. If a

wake-up on the channel is detected, the LinIf_Cbk_CheckWakeup function will be called by the IoHwA.
LINIF496: When the wake-up command was sent, the LinIf shall issue the LinSm_WakeUp_Confirmation callback.

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

7.3.4.1 Wakeup during sleep transition

Since the wake-up process is a sporadic event it may happen that someone tries to wake-up 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 occurs, the following shall apply:

LINIF186: If the go-to-sleep status check is pending (from the point that it has been transmitted until the status check) and if the LIN Interface has detected a wake-up on the bus, the LIN Interface shall maintain the LIN channel state CHANNEL_OPERATIONAL and notify the upper layer of the wake-up-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 wake-up. In this case, the following shall apply:

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

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

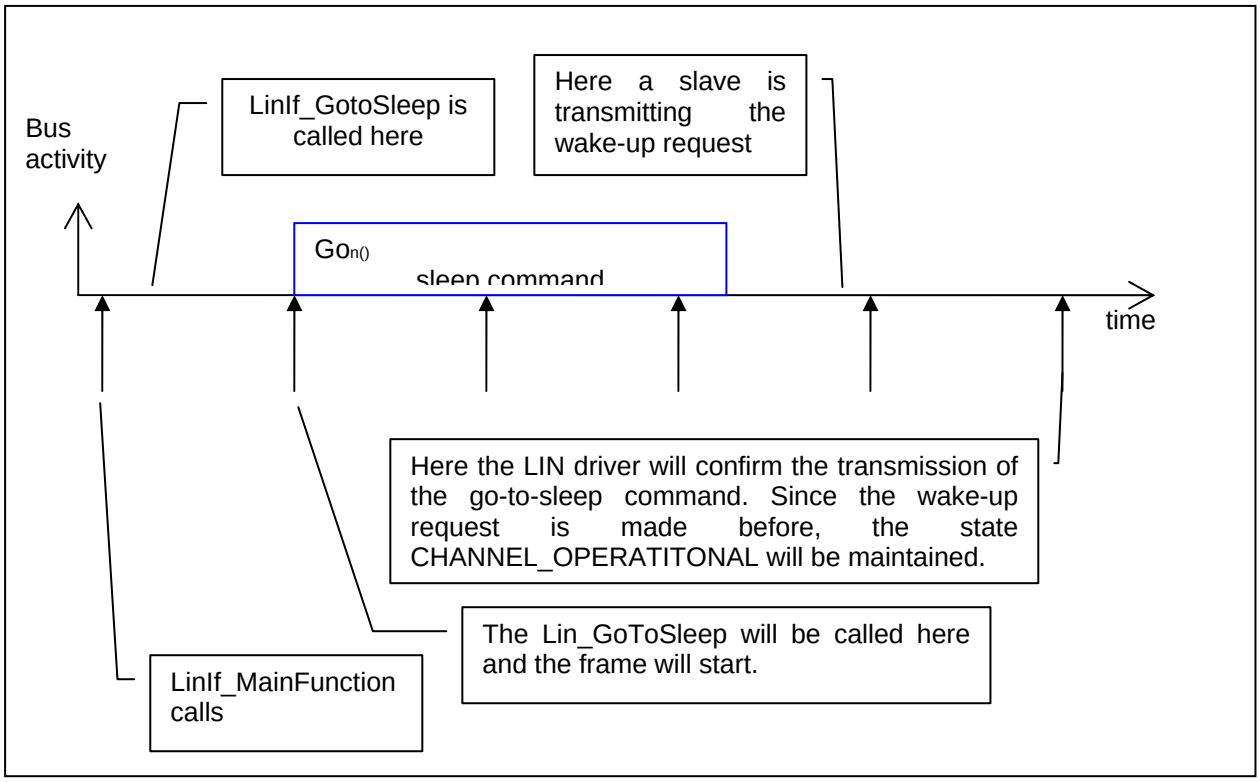

Figure 7 – Wake up requested before confirmation of go-to-sleep-command

7.4 Status Management

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

According to the LIN 2.0 specification, each slave node publishes a Response_Error bit in a transmitted frame.

LINIF464: The schedule table manager shall monitor the Response_Error bit (defined in the LIN 2.0 specification) if it is configured for the received frame (Configuration parameter: `LinIfResponseErrorFrameRef`) and raise the production error LINIF_E_CHANNEL_X_SLAVE_Y when the bit is set.

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

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

7.5 Diagnostics and Node configuration

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

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

7.5.1 Node configuration

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

The LIN 2.0 Specification specifices two ways for the LIN master to configure slaves:

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

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

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

7.5.1.1 Node Model

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

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

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

7.5.1.2 Node Configuration services

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

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

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

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

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

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

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

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

7.5.1.3 Node Configuration API

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

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

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

7.5.1.4 Node Configuration in Schedule Table

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

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

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

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

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

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

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

LINIF405: The LIN Interface shall raise the production error code LINIF_E_NC_NO_RESPONSE if it has put a SRF in the schedule table after a node configuration command and if it has not received an answer from the slave.

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

LINIF407: The LIN Interface shall store 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 the LIN 2.0 specification, the Transport Protocol (TP) is optional to implement. There are three types of diagnostics defined:

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

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

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

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

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

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

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

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

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

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

7.5.2.1 LIN TP initialization

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

7.5.2.2 State-machine

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

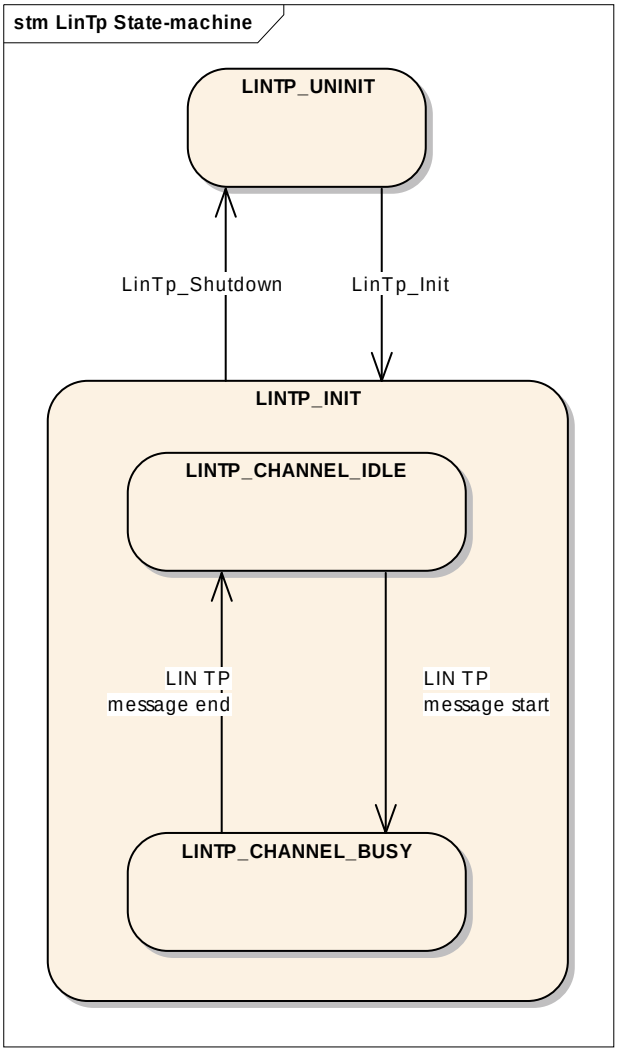

Figure 8 - LIN Transport Protocol state-machine

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

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

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

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

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

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

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

LINIF414: The LIN Interface shall set the sub-state of a channel to LINTP_CHANNEL_IDLE when it has successfully sent a LIN TP message or detected an unrecoverable error on this channel.

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

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

7.5.2.3 Buffer handling

The PDU router shall provide the buffer for the transmission or reception of a TP message. It is assumed that this buffer will be used by LinIf until a new buffer is requested.

The reason is that the request for a new buffer may introduce jitter on the bus.

LINIF325: The function LinIf_MainFunction shall request the PDU Router for a new buffer (for the transmission or reception of a TP message) if the current buffer length is not big enough to fill the next MRF (FF, SF and CF).

Note that FF and SF is possible since the request for buffer in LinTp_Transmit may indicate that temporarily no buffer is available.

This means that the actual setup of each frame for the LIN TP is 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: The LIN Interfache shall provide a buffer that can contain a complete frame. This buffer shall be available for both transmission and reception of a LIN TP message.

7.5.2.4 LIN TP Transmission

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

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

LINIF328: The LIN TP will request a buffer 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 provides a buffer which is too small for the next frame.

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

LINIF329: The function LinIf_MainFunction shall not send the next MRF if a buffer request by PduR_LinTpProvideTxBuffer was not successful. This includes the BUSY return value from the PduR_LinTpProvideTxBuffer.

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

LINIF330: The function LinIf_MainFunction shall keep the handling of a LIN TP message after the LIN Interface has initiated the LIN TP message.

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

LINIF068: The function LinIf_MainFunction shall call the function PduR_LinTpTxConfirmation of the PDU Router when it has transmitted the last frame (SF or CF) correctly.

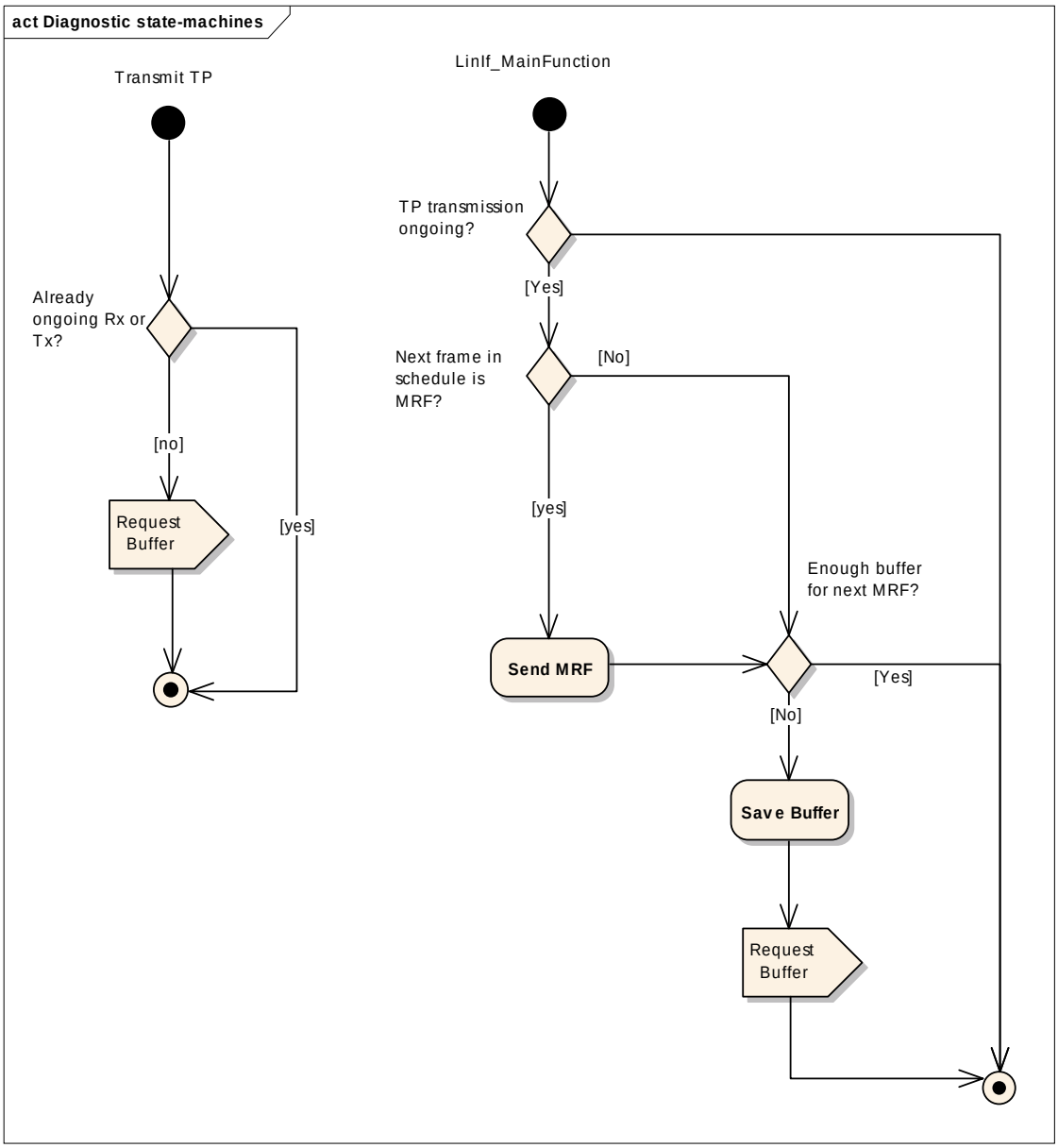

Figure 9 - TP transmission initiation (left side) and TP message transmission (right side)

7.5.2.5 LIN TP transmission error

LINIF069: The function LinIf_MainFunction shall abort the transmission of a LIN TP Message and call the function PduR_LinTpTxConfirmation of the PDU Router to notify the error and the abortion if a LIN error on the MRF, which carries the LIN TP Message, occurs.

LINIF073: If the callback PduR_LinTpProvideTxBuffer indicates permanent failure by its return code of the callback, the function LinIf_MainFunction shall abort the sequence of the LIN TP transmit and call the function PduR_LinTpTxConfirmation of the PDU Router.

7.5.2.6 TP Reception

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

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

LINIF075 The PdR_LinTpProvideRxBuffer callback function shall be called when the start of a SRF is indicated by the reception of a FF or SF.

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

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

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

LINIF078: When the LIN Interface has successfully received and stored the last SRF in a LIN TP message (CF), it shall call the function PduR_LinTpRxIndication of the PDU Router.

7.5.2.7 LIN TP reception error

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

LINIF079: The LIN Interface shall stop the current LIN TP message reception when it detects one of the following errors:

- 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 starts a new LIN TP message during an ongoing LIN TP message reception.

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

LINIF081: If a TP message is stopped by the LinIf because of an error, the PduR_LinTpRxIndication shall be called by the Lin TP with an appropriate status to indicate the 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 continues to transmit the rest of the frames when the master transmits a SRF header. The slave cannot know when the master has encountered a problem. The slave continues to transmit responses to the SRF headers. This means that no error-recovery is supported.

LINIF332: The function PduR_LinTpRxIndication will be used by the LIN Interface to determine an error. The rest of the CF frames from the slave in the TP message shall be discarded.

7.5.2.8 Unavailability of receive buffer

The function PduR_LinTpProvideRxBuffer of the PDU Router may indicate that the requested buffer is not available. The reason for that can be the following:

- a wait condition
- a permanent failure

The LIN Interface handles these cases differently.

LINIF085: The LIN TP has to wait for a requested buffer using the function PduR_LinTpProvideRxBuffer, the LIN TP shall suspend sending LIN headers for the SRF until one of the following conditions are met:

- a new buffer is provided by PduR_LinTpProvideRxBuffer
- the upper layer indicates a permanent failure by the callback PduR_LinTpRxIndication
- a new reception of a LIN TP message is started

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

LINIF086: The function LinIf_MainFunction shall retry requesting a receive buffer periodically if a receive buffer is unavailable.

LINIF087: The LIN TP shall stop the ongoing LIN TP message reception if the function PduR_LinTpRxIndication of the PDU Router indicates a permanent failure by its parameter `Result`.

7.6 Handling multiple channels and drivers

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

7.6.1 Multiple channels

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

7.6.2 Multiple LIN drivers

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

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

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

7.7 Error classification

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

LINIF267: Development error values are of type uint8.

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

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
LIN response error detected	Production	LINIF_E_CHANNEL_X_SLAVE_Y With X being the index of the channel and Y the name of the slave.	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 `LinIfDevErrorDetect` (see Chapter 10) shall activate or deactivate the detection of all development errors.

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

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

7.9 Error notification

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

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

8 API specification

8.1 Imported types

8.1.1 Standard types

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

LINIF469:

Header file	*Imported Type*
Dem_Types.h	Dem_EventIdType
BufReq_Types.h	BufReq_ReturnType
Lin_Types.h	Lin_ChannelConfigType
	Lin_StatusType
	Lin_PduType
	Lin_ConfigType
EcuM_Types.h	EcuM_WakeupSourceType
PrimitiveTypes.h	PduInfoType
ComStack_Types.h	PduIdType
	NetworkHandleType
	PduLengthType
FrTp_Types.h	NotifResultType
LinIf_Types.h	LinTpConfigType
	PduInfoType
LinTp_Types.h	LinTp_ParameterValueType
	LinTp_CancelReasonType
Std_Types.h	Std_ReturnType
	Std_VersionInfoType

Type definitions

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

8.1.2 LinIf_SchHandleType

LINIF197:

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

8.1.3 LinTp_ConfigType

LINIF426:

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

8.1.4 LinTp_ParameterValueType

LINIF508:

Name:	`LinTp_ParameterValueType`
Type:	`uint8`
Description:	Range of LinTP_STMIN

8.1.5 LinTp_CancelReasonType

LINIF509:

Name:	LinTp_CancelReasonType
Type:	uint8
Description:	Reason for the cancelation of the TP transmission.

8.2 LIN Interface API

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

8.2.1 LinIf_Init

LINIF198:

Service name:	LinIf_Init
Syntax:	`void LinIf_Init(`

	`const void* ConfigPtr`)
Service ID[hex]:	0x01
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	ConfigPtr — Pointer to the LIN Interface configuration
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	Initializes the LIN Interface.

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

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

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

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

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

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

8.2.2 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 (inout):	None
Parameters (out):	versioninfo — Pointer to where to store the version information of this module.
Return value:	None
Description:	Returns the version information of this module.

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

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

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

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

8.2.3 LinIf_Transmit

LINIF201:

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

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

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

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

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

8.2.4 LinIf_ScheduleRequest

LINIF202:

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

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

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

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

LINIF394: If development error detection is enabled and no more requests are allowed because the request queue is full, the function LinIf_ScheduleRequest shall

raise the development error LINIF_E_SCHEDULE_OVERFLOW and return E_NOT_OK.

8.2.5 LinIf_GotoSleep

LINIF204:

Service name:	LinIf_GotoSleep	

Syntax:	`Std_ReturnType LinIf_GotoSleep(`
	`  NetworkHandleType Channel`
	`)`	

Service ID[hex]:	0x06	
Sync/Async:	Asynchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	Channel	Identification of the LIN channel.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Request to go to sleep has been accepted or sleep transition is already in progress. 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:	Initiates a transition into the Sleep Mode on the selected channel.	

LINIF488: The function LinIf_GotoSleep shall initiate a transition into sleep mode on the selected channel by transmitting a LIN diagnostic master request frame with its first data byte equal to 0x00.

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

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

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

8.2.6 LinIf_WakeUp

LINIF205:

Service name:	LinIf_WakeUp

Syntax:	`Std_ReturnType LinIf_WakeUp(`
	`  NetworkHandleType Channel`
	`)`

Service ID[hex]:	0x07	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant	
Parameters (in):	Channel	Identification of the LIN channel.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Request to wake up has been accepted or the controller is not in sleep state. E_NOT_OK: Request to wake up has not been accepted due to one or more of the following reasons: - LIN Interface has not been initialized - referenced channel does not exist (identification is out of range)
Description:	Initiates the wake up process.	

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

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

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

LINIF306:If the go-to-sleep command is not pending and the channel state is CHANNEL_OPERATIONAL, the LinIf_WakeUp shall return with E_OK and perform no action.

8.2.7 LinTp_Init

LINIF350:

Service name:	LinTp_Init	

Syntax:	`void LinTp_Init(`
	`  const LinTpConfigType* ConfigPtr`
	`)`	

Service ID[hex]:	0x40	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	ConfigPtr	Pointer to the LIN Transport Protocol configuration.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Initializes the LIN Transport Layer.	

LINIF427: The parameter ConfigPtr of the function LinTp_Init shall be only relevant for the configuration variant Variant 3. The parameter ConfigPtr shall be ignored for

the configuration variant Variant 1 and the configuration variant Variant 2 but will be still there for compatible reasons.

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

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

8.2.8 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.
	LinTpTxInfoPtr	A pointer to a structure with N-SDU related data containing: - pointer to a N-SDU buffer - length of this buffer.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: The request can be started successfully E_NOT_OK: The request can not be started (e.g. there is already an ongoing TP message on the selected channel)
Description:	Requests the transfer of segmented data.	

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

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

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

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

LINIF388: If the LIN Interface's environment is calling the function LinTp_Transmit and a LIN 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: The function LinTp_Transmit shall request the PDU Router for a new buffer using the function PduR_LinTpProvideTxBuffer().

8.2.9 LinTp_GetVersionInfo

LINIF352:

Service name:	LinTp_GetVersionInfo

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

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

LINIF353: The function LinTp_GetVersionInfo shall return the version information of the LIN TP. The version information includes:
The version number consists of three parts:

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

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

8.2.10 LinTp_Shutdown

LINIF355:

Service name:	LinTp_Shutdown

Syntax:	`void LinTp_Shutdown(`
	
	`)`

Service ID[hex]:	0x43
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters	None

(inout):	
Parameters (out):	None
Return value:	None
Description:	Shutdowns the LIN TP.

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

LINIF433: The function LinTp_Shutdown shall affect all configured channels.

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

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

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

8.2.11 LinTp_CancelTransmitRequest

LINIF500:

Service name:	LinTp_CancelTransmitRequest	
Syntax:	`Std_ReturnType LinTp_CancelTransmitRequest( PduIdType LinTpTxSduId, LinTp_CancelReasonType LinTpCancelReason )`	
Service ID[hex]:	0x46	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant	
Parameters (in):	LinTpTxSduId	This parameter contains the Lin TP instance unique identifier of the Lin N-SDU which transfer has to be cancelled.
	LinTpCancelReason	The reason for cancellation
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_NOT_OK: Cancellation request of the transfer of the specified Lin N-SDU is rejected
Description:	Cancels the request to transmit a LIN frame.	

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

8.2.12 LinTp_ChangeParameterRequest:

LINIF501:

Service name:	LinTp_ChangeParameterRequest	

Syntax:	`void LinTp_ChangeParameterRequest(`
	`  PduIdType LinTpTxSduId,`
	`  LinTp_ParameterValueType LinTpParameterValue`
	`)`	

Service ID[hex]:	0x44	
Sync/Async:	Asynchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	LinTpTxSduId	Gives the ID of the connection (message) for whose channel the change shall be done.
	LinTpParameterValue	This parameter contains the new value of LinTP_STMIN
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Changes parameter request.	

LINIF506: This function shall do nothing.

8.3 Call-back notifications

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

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

8.3.1 LinIf_Cbk_CheckWakeup

LINIF378:

Service name:	LinIf_Cbk_CheckWakeup	

Syntax:	`void LinIf_Cbk_CheckWakeup(`
	`  NetworkHandleType Channel`
	`)`	

Service ID[hex]:	0x60	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Channel	Identification of the LIN channel
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Will be called when the IoHwA detects a wake-up for the specific LIN channel.	

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

The function LinIf_Cbk_CheckWakeup may be called in an interrupt context.

8.4 Scheduled functions

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

8.4.1.1 LinIf_MainFunction

LINIF384:

Service name:	LinIf_MainFunction

Syntax:	`void LinIf_MainFunction(`
	
	`)`

Service ID[hex]:	0x80
Timing:	FIXED_CYCLIC
Description:	The main processing function of the LIN Interface.

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

LINIF473: The function LinIf_MainFunction shall operate all channels.

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

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

LINIF289: The function LinIf_MainFunction shall make all calls to the upper layers except for the wake-up indication.

To have all calls within the LinIf_MainFunction implies that these calls are made “periodically” (since the function is called with a specific period).

8.5 Expected Interfaces

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

8.5.1 Mandatory Interfaces

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

LINIF359:

API function	*Description*
PduR_LinTpTxConfirmation	Tx confirmation for the LIN TP
Lin_GetStatus	Gets the status of the LIN driver.
PduR_LinTpRxIndication	Rx indicator for the LIN TP
Lin_WakeUpValidation	--
EcuM_SetWakeupEvent	Sets the wakeup event.
Lin_SendHeader	Sends a LIN header.
Lin_SendResponse	Sends a LIN response.
Lin_GoToSleepInternal	Sets the channel state to LIN_CH_SLEEP, enables the wake-up detection and optionally sets the LIN hardware unit.
PduR_LinTpProvideRxBuffer	Provides Rx Buffer for the LIN TP
PduR_LinTpProvideTxBuffer	Provides Tx Buffer for the LIN TP
PduR_LinIfTriggerTransmit	Triggers the transmission of a LIN frame
PduR_LinIfTxConfirmation	Tx confirmation for the LIN Interface
Lin_GetVersionInfo	Returns the version information of this module.
Lin_Init	Initializes the LIN module.
Lin_DeInitChannel	De-Inits a LIN channel.
Lin_WakeupValidation	Identifies LIN channels.
Lin_InitChannel	(Re-)initializes a LIN channel.
PduR_LinIfRxIndication	Rx indicator for the LIN Interface
Lin_WakeUp	Generates a wake up pulse.
Dem_ReportErrorStatus	Reports errors to the DEM.

Further functions are required by the LIN Interface:

PduR_LinTpChangeParameterConfirmation	

8.5.2 Optional interfaces

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

LINIF360:

API function	*Description*
Det_ReportError	Service to report development errors.

8.5.3 Configurable interfaces

No configurable interfaces.

9 Sequence diagrams

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

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

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

9.1 Frame Transmission

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

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

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

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

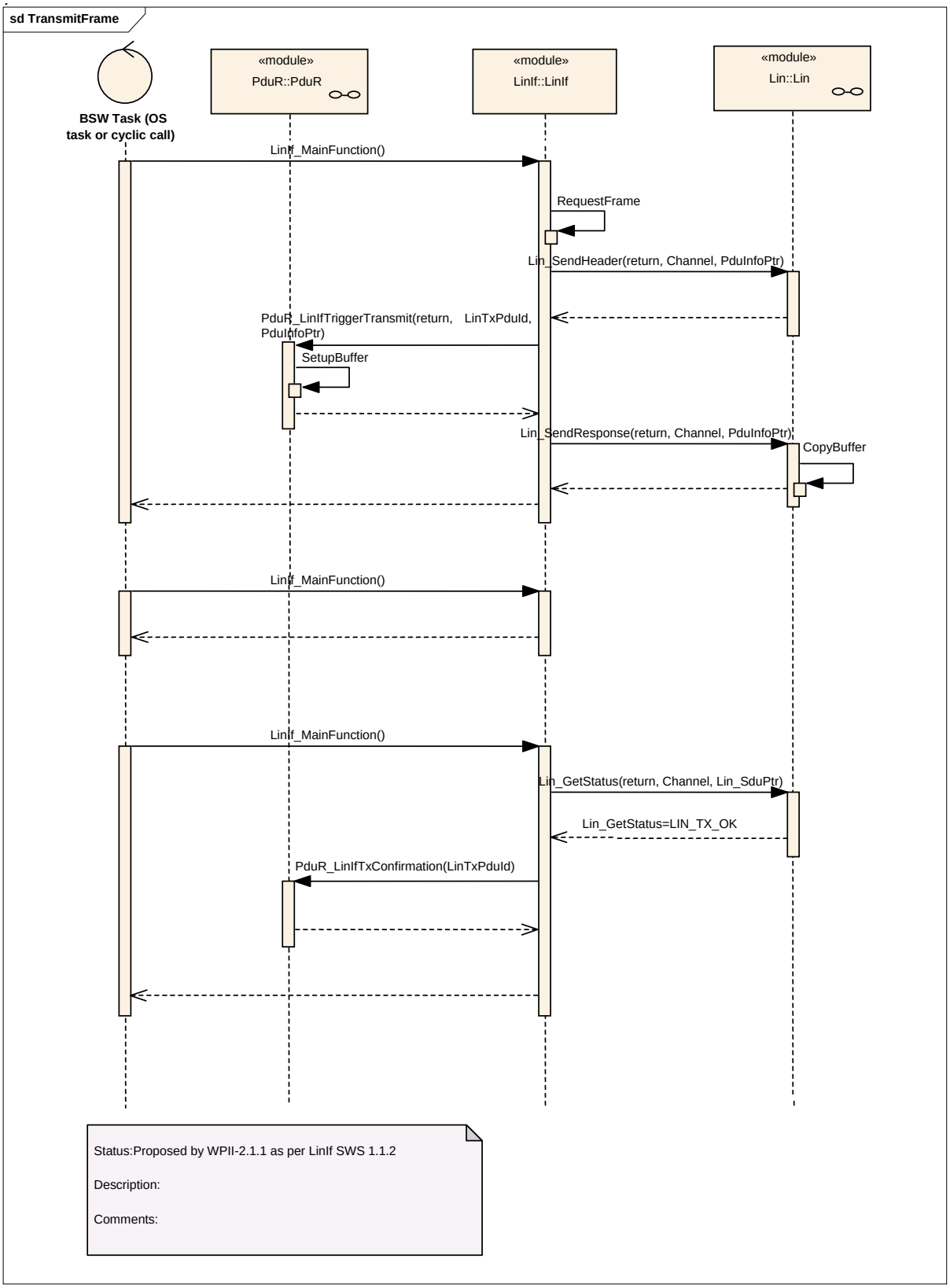

Figure 10- Frame transmission

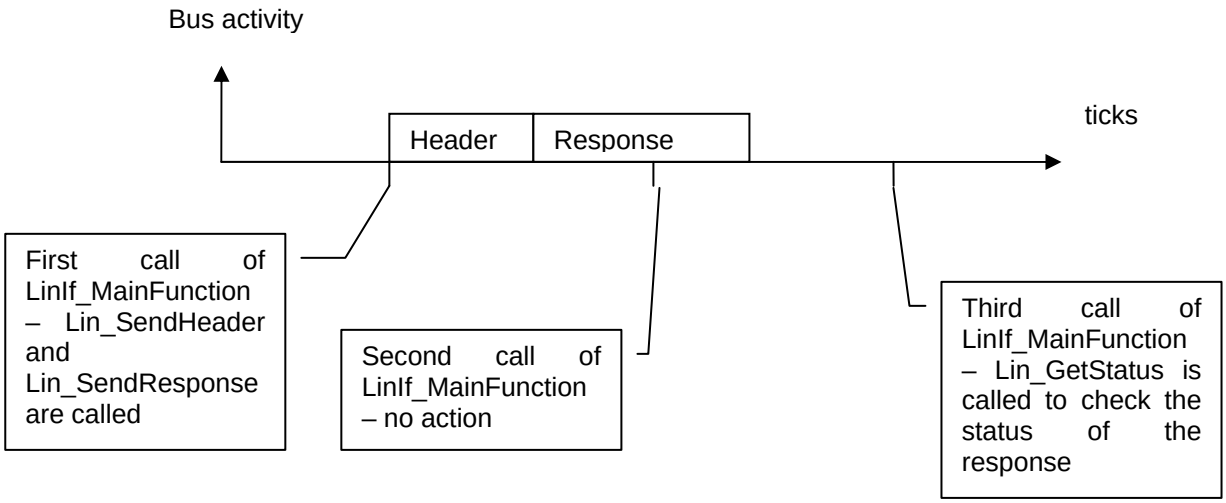

Figure 11 - Timing information for transmitted frame

9.2 Frame Reception

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

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

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

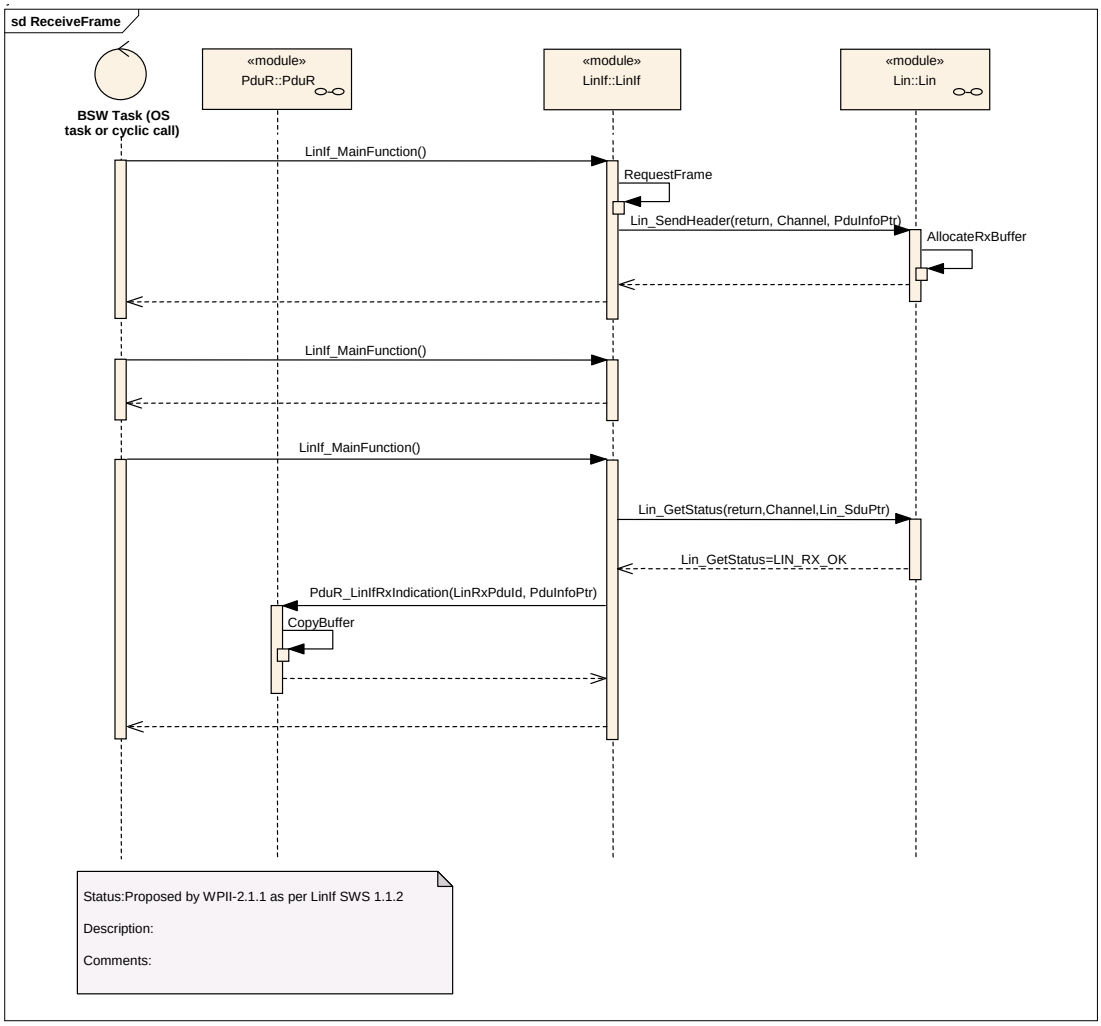

Figure 12 - Frame reception

9.3 Slave to slave communication

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

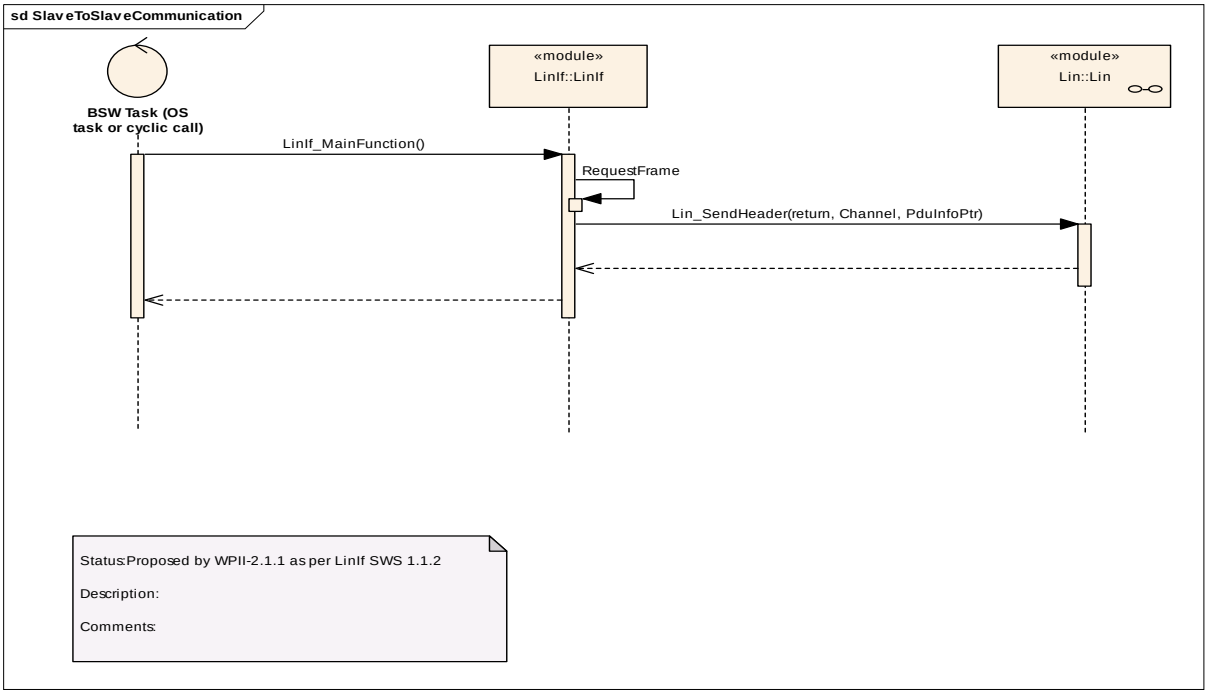

Figure 13 - Slave to slave communication

9.4 Reception and transmission at the same time

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

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

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

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

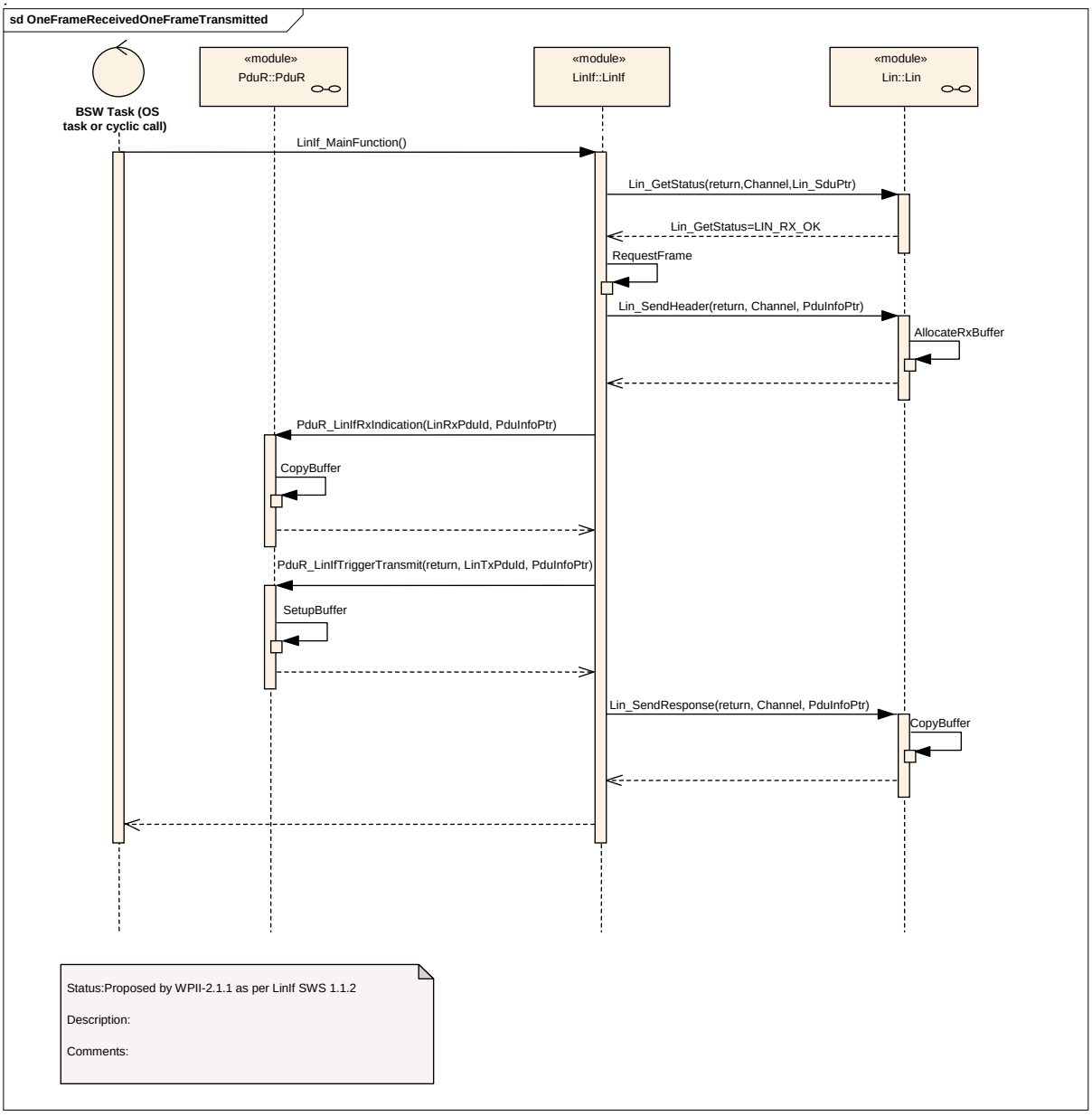

Figure 14 - One frame received and one frame transmitted

9.5 Sporadic frame

The following use case shows an upper layer requesting transmission of a sporadic frame. Actually, this call does not initiate the transmission of the frame since the schedule table must be followed. It just marks the frame for transmission. When the sporadic slot (note that the schedule entry for a sporadic frame is a slot and not a frame) is due in the schedule table, the LinIf_MainFunction transmits the sporadic frame as a normal transmitted frame and according to the priority rules for sporadic frames.

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

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

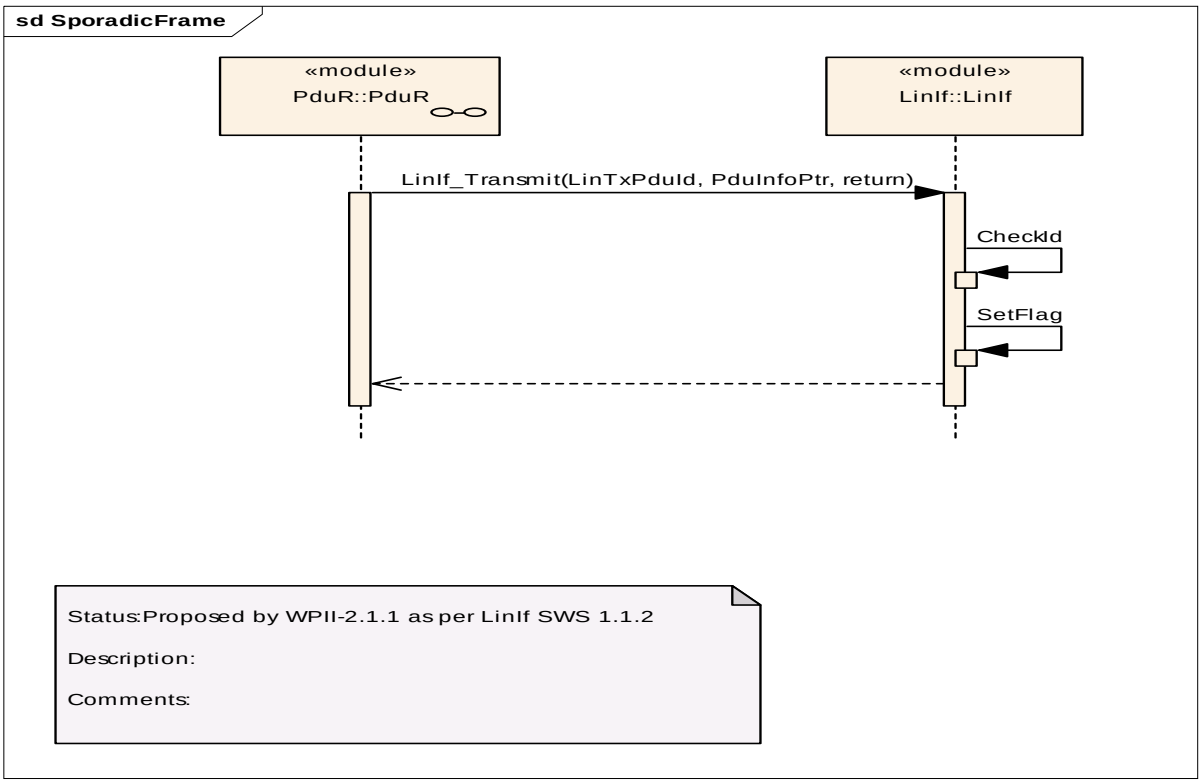

Figure 15 - Sporadic frame

9.6 Event triggered frame

There are three results for an event triggered frame:

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

All three use cases are shown below.

9.6.1 With no answer

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

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

The RequestFrame call in the diagram is the interface call to the Schedule Manager. The LinIf_MainFunction gets the frame to send and the delay to the next frame.
The AllocateRxBuffer call is to show that the storage of the received SDU is made in the LIN Driver and not in the LIN Interface.

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

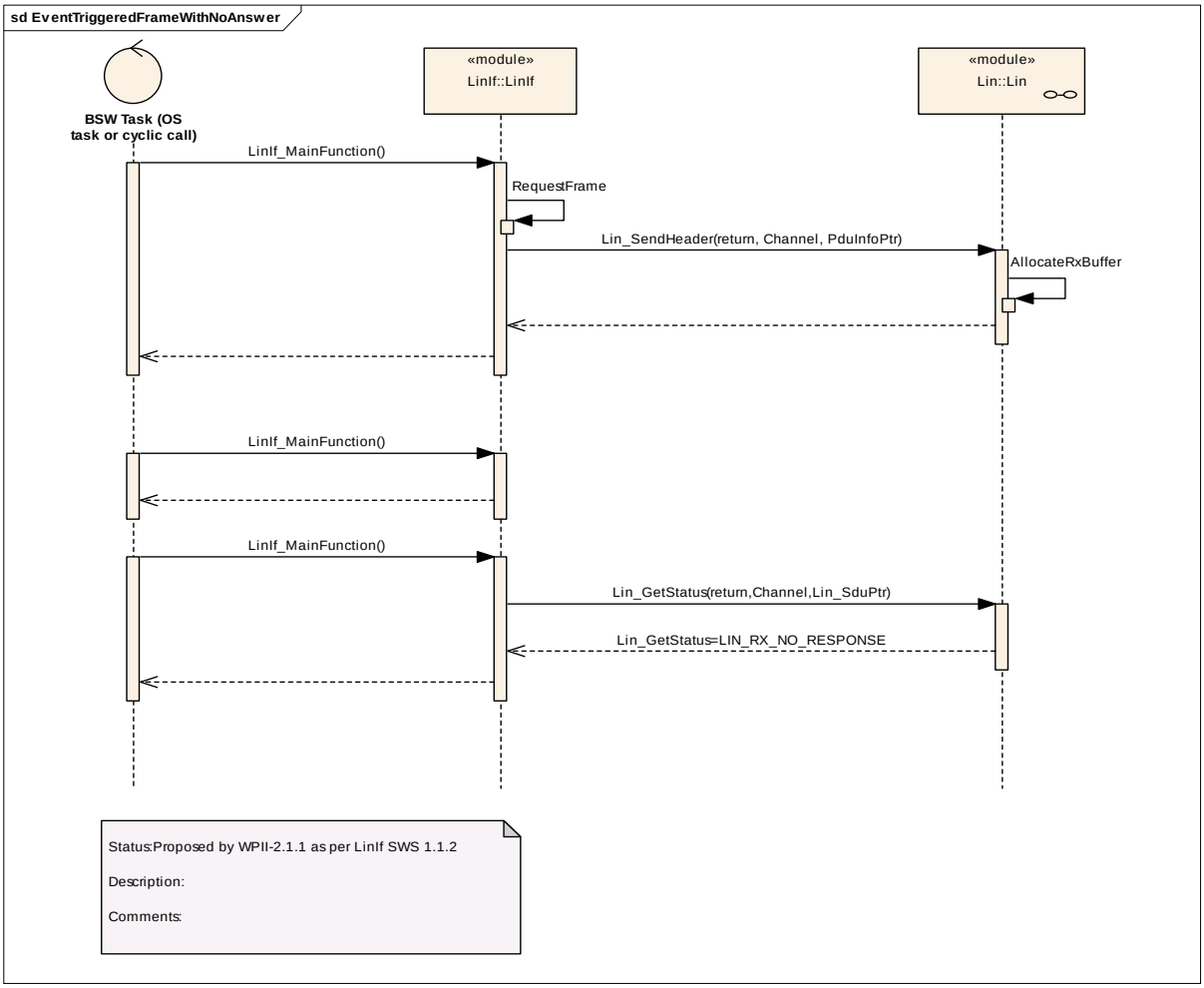

Figure 16 – Event triggered frame with no answer

9.6.2 With answer (No collision)

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

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

The RequestFrame call in the diagram is the interface call to the Schedule Manager. The LinIf_MainFunction gets the frame to send and the delay to the next frame.
The AllocateRxBuffer call is to show that the storage of the received SDU is made in the LIN Driver and not in the LIN Interface.

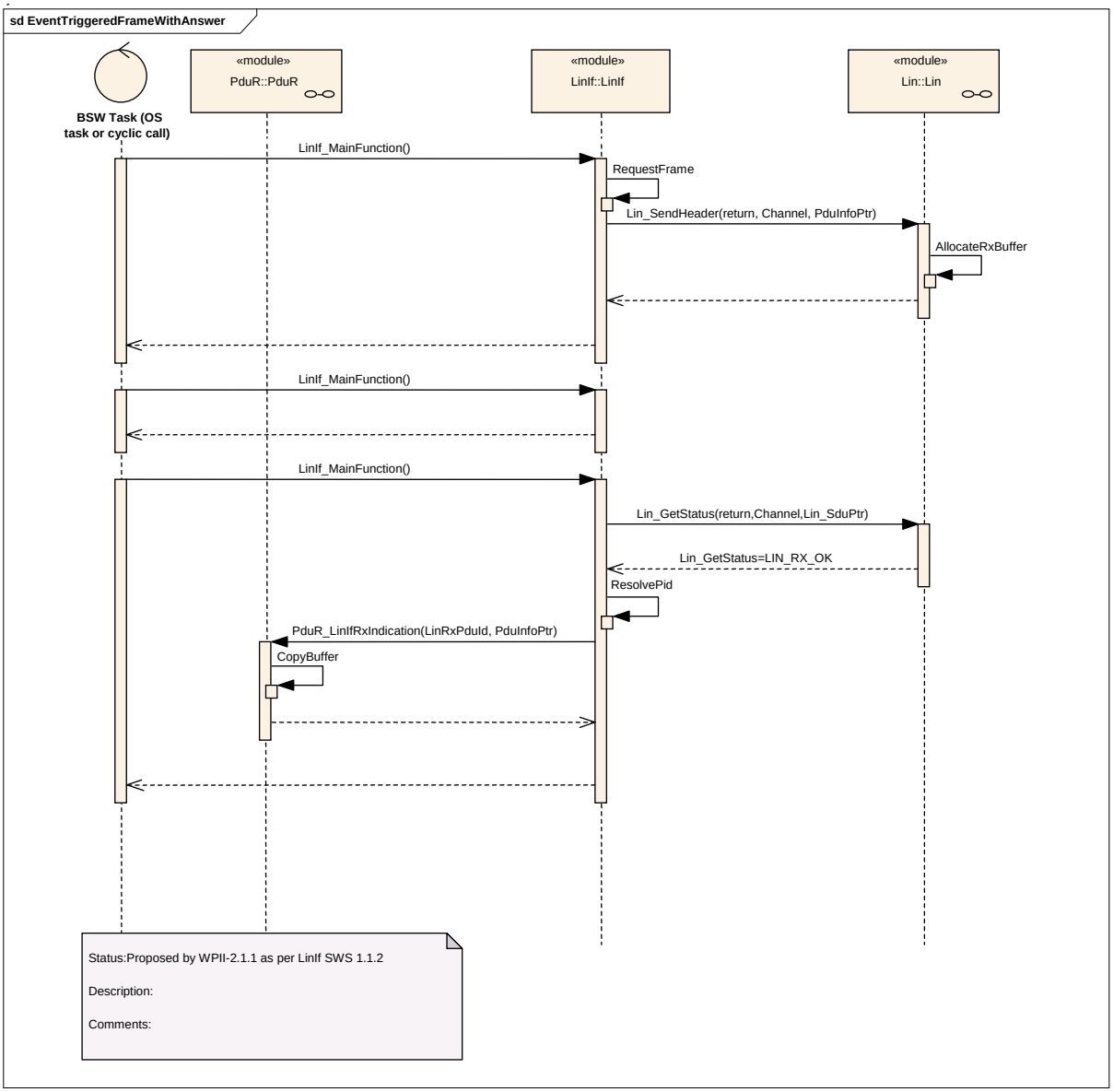

Figure 17 – Event triggered frame with answer (no collision)

9.6.3 With collision

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

The first call of the LinIf_MainFunction requests transmission of the header. During the second call, the frame is under transmission. In the third call, the frame is finished (this call is called after the maximum frame length).
The RequestFrame call in the diagram is the interface call to the Schedule Manager. The LinIf_MainFunction gets the frame to send and the delay to the next frame.

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

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

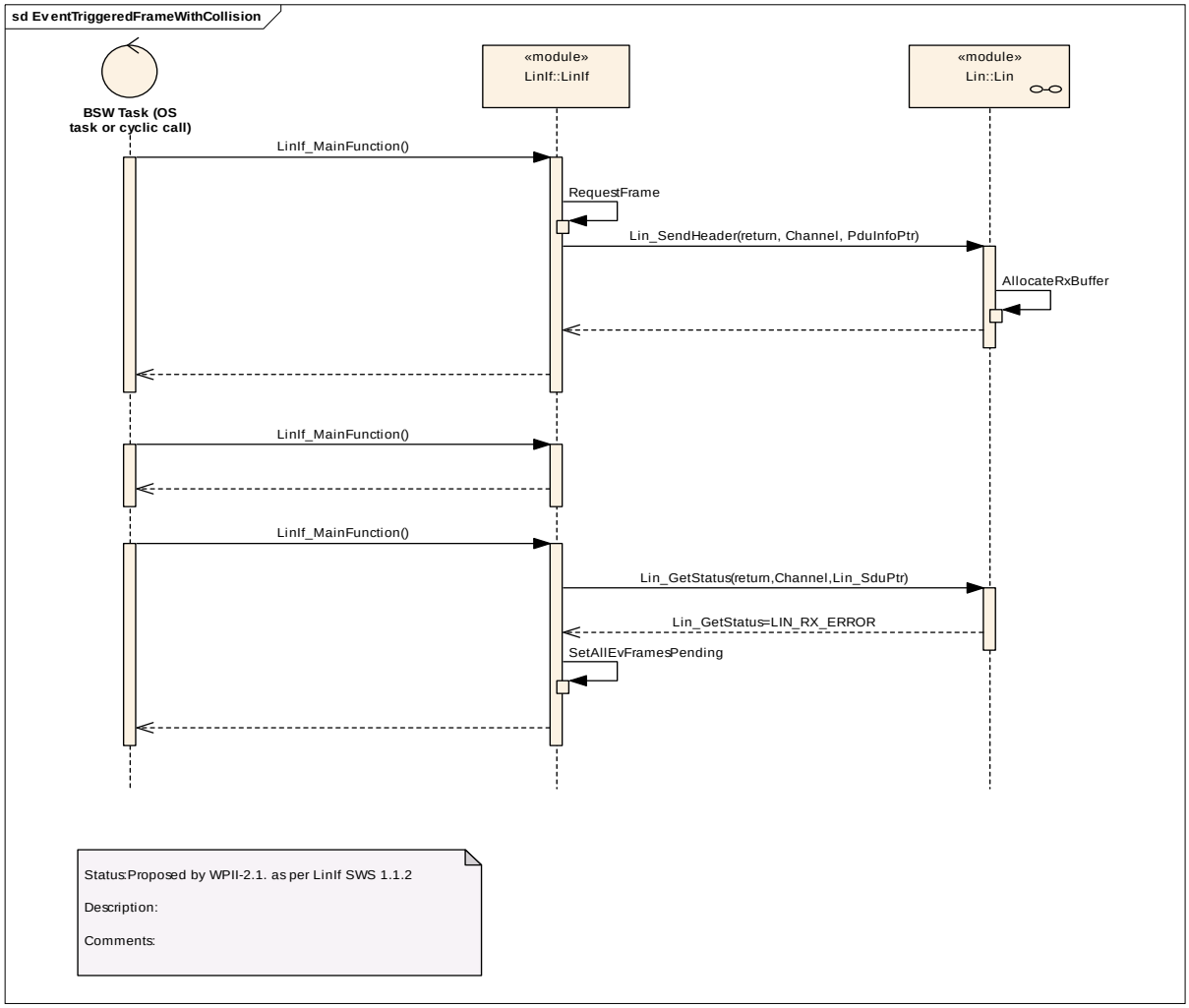

Figure 18 – Event triggered frame with collision

9.7 Transport Protocol Message transmission

The following diagram Figure 19 shows the transmission of a TP message. Both the initiation of the message and the 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.

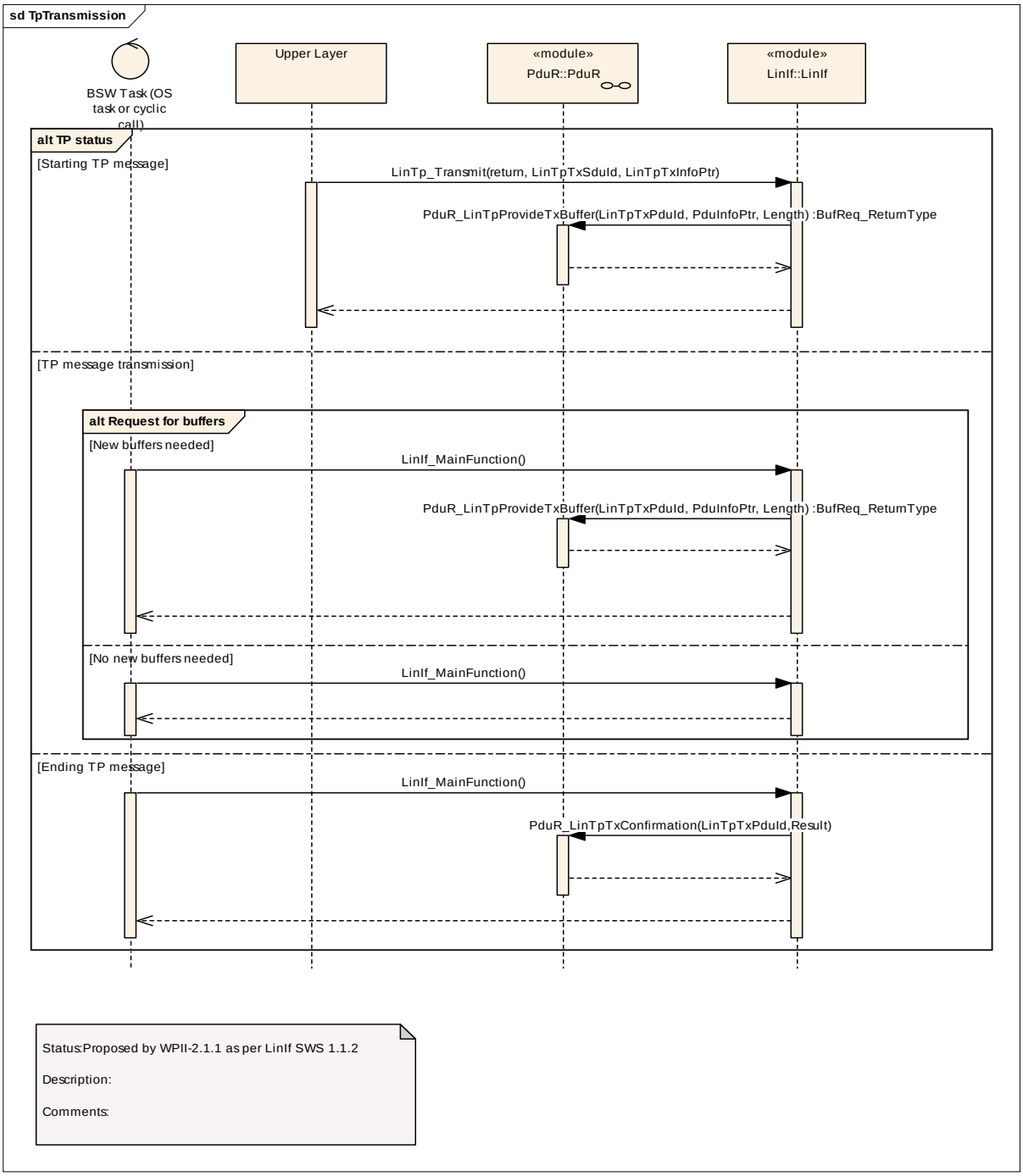

Figure 19 -Transport Protocol Transmission

9.8 Transport Protocol message reception

The following diagram Figure 19 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.

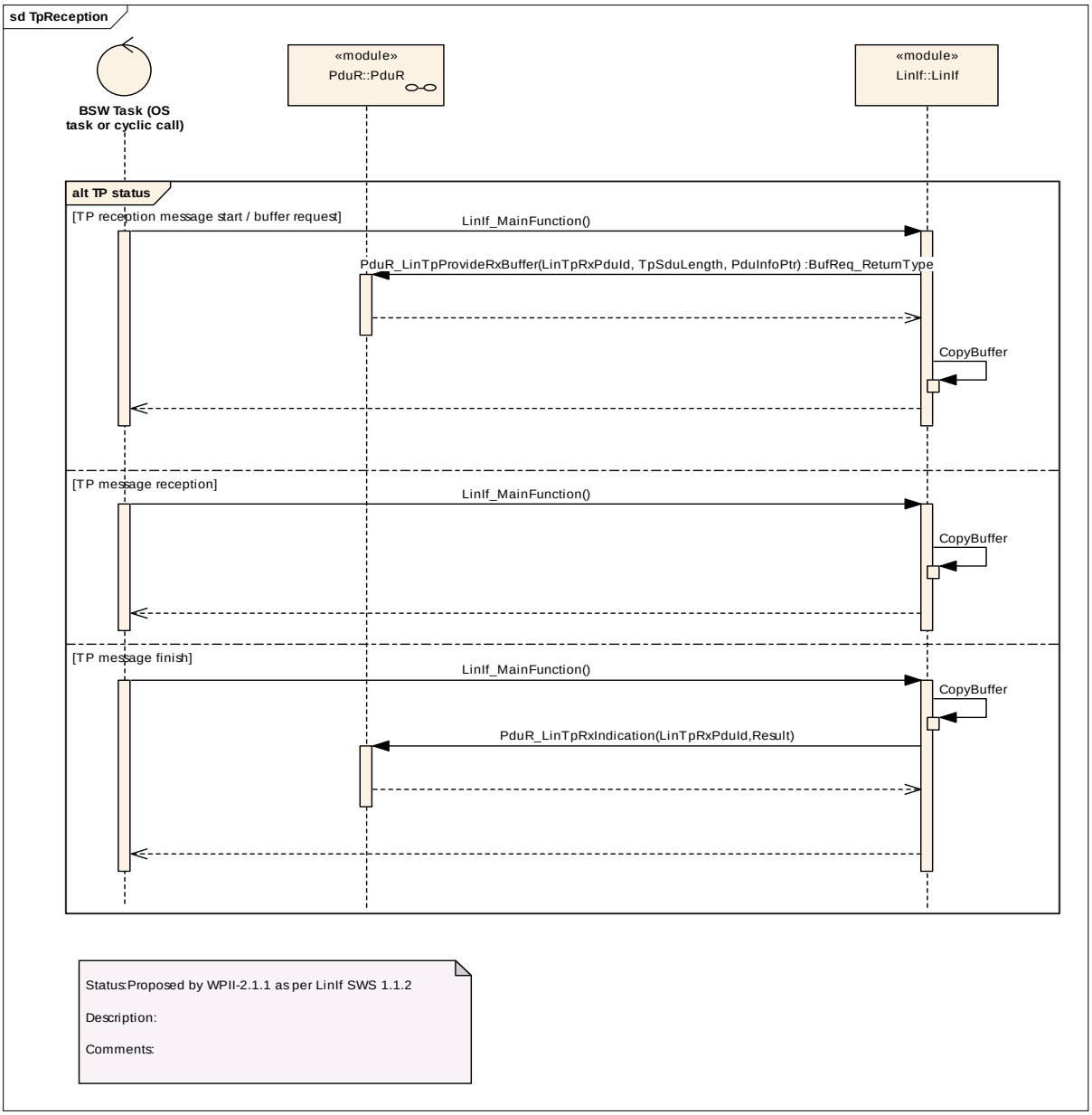

Figure 20 - Transport protocol message reception

9.9 Go-to-sleep process

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

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

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

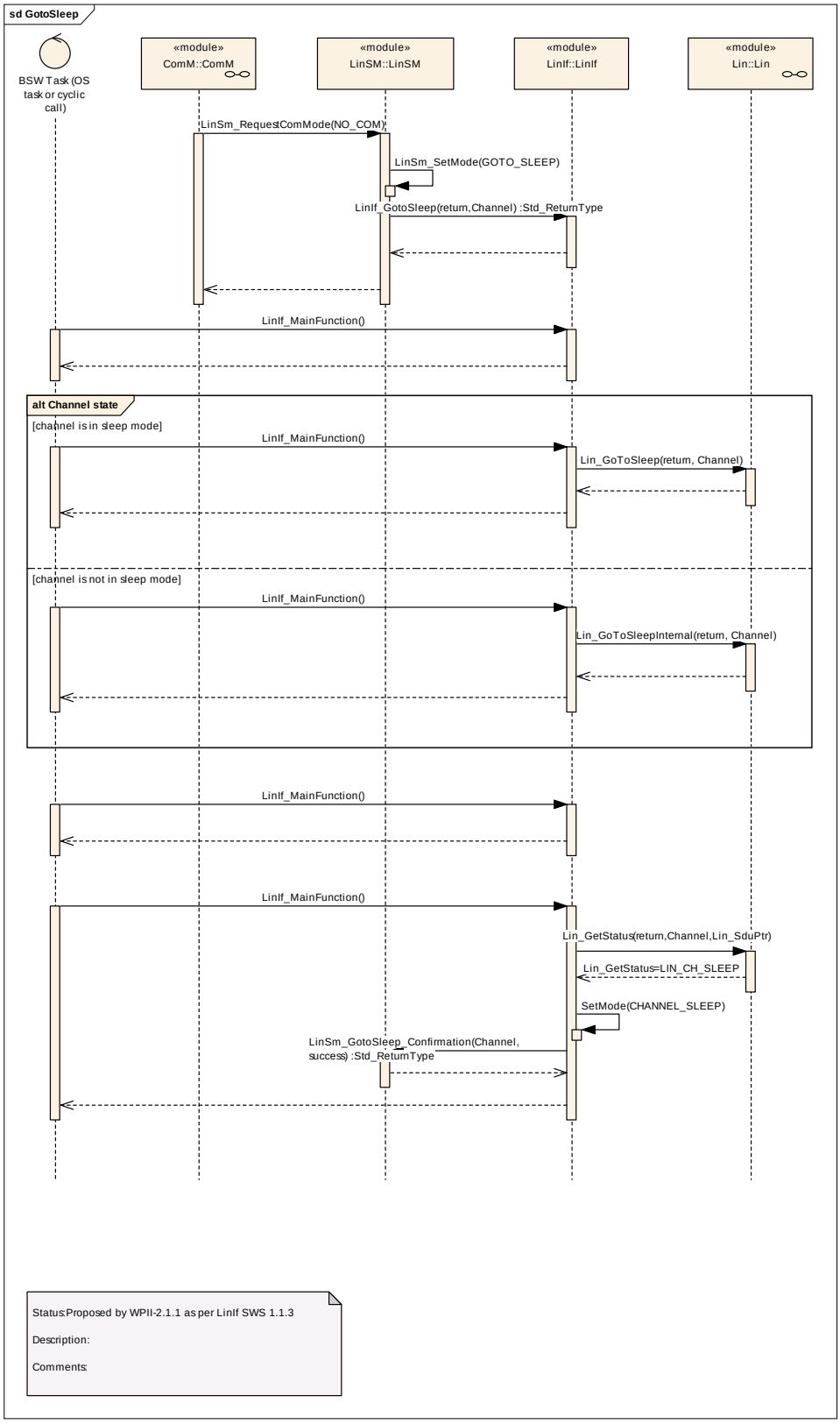

Figure 21 - Go-to-sleep–command process

9.10 Wake up request

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

9.11 Internal wake-up

There are two different use cases in Figure 22:

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

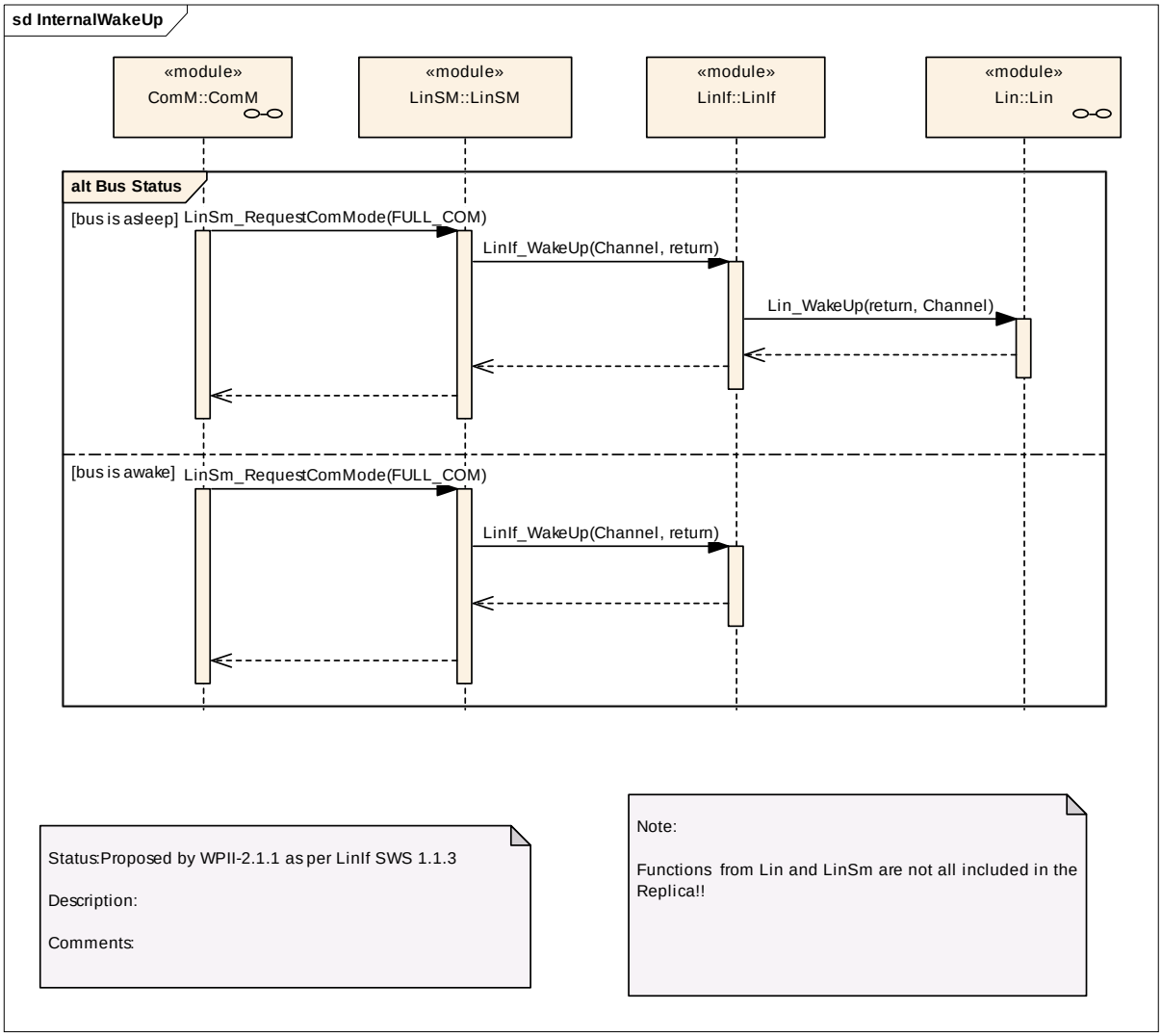

Figure 22 - Internal wake-up

10 Configuration specification

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

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

LINIF491: In the pre-compile configuration all parameters below that are marked as Pre-compile configurable shall be configurable in a pre-compile manner, for example as #defines.

The module is most likely delivered as source code.

10.2.2.2 Variant2 (Link-time Configuration)

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

The module is most likely delivered as object code.

10.2.2.3 Variant3 (Post-build Configuration)

LINIF493: This configuration includes all configuration options of the “Link-time configuration”. Additionally all parameters defined below, as post build configurable shall be configurable post build for example by flashing configuration data.

The module is most likely delivered as object code.

10.3 LinIf_Configuration

The Figure 25 depicts the LIN Interface configuration.

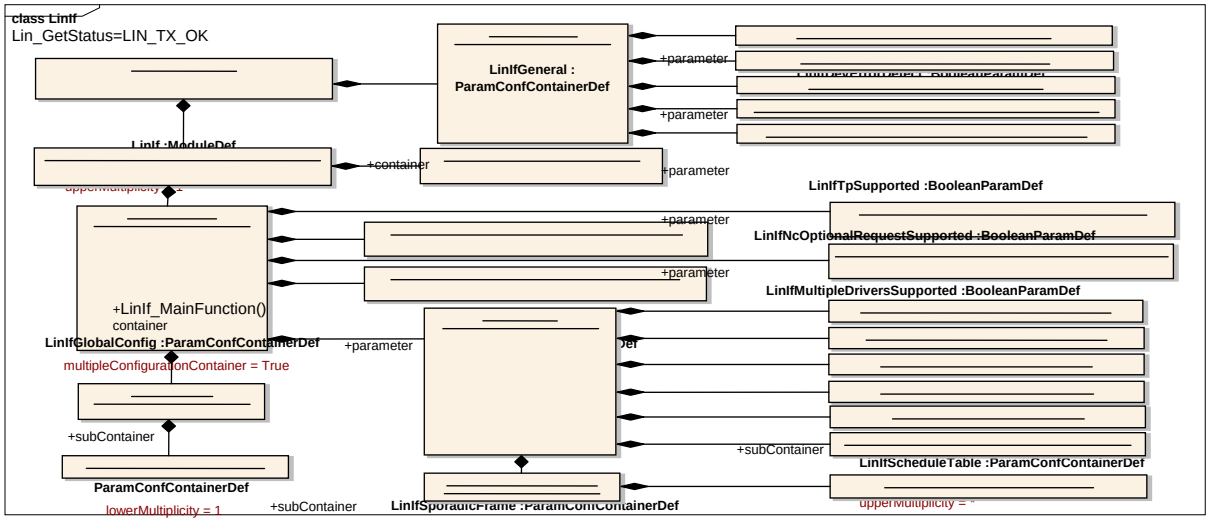
class LinIf
Lin_GetStatus=LIN_TX_OK
LinIfGeneral : ParamConfContainerDef
+parameter
+parameter
+container
+parameter
LinIfTpSupported :BooleanParamDef
+parameter
LinIfNcOptionalRequestSupported :BooleanParamDef
+parameter
LinIfMultipleDriversSupported :BooleanParamDef
+LinIf_MainFunction()
container
LinIfGlobalConfig :ParamConfContainerDef
multipleConfigurationContainer = True
+parameter
+subContainer
ParamConfContainerDef
lowerMultiplicity = 1
upperMultiplicity = *
+subContainer
LinIfScheduleTable :ParamConfContainerDef
+subContainer
LinIfSporadicFrame :ParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *
+subContainer
LinIfEventTriggeredFrame :ParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *
+subContainer
LinIfDynamicFrame :ParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *
+parameter
LinIfProtocolVersion :IntegerParamDef
LinIfSlave : ParamConfContainerDef
+subContainer
lowerMultiplicity = 0
upperMultiplicity = *
+parameter
LinIfConfiguredNad :IntegerParamDef
+parameter
LinIfSupplierId :IntegerParamDef
+subContainer
LinIfMaster : ParamConfContainerDef
+parameter
LinIfFunctionId :IntegerParamDef
+parameter
LinIfVariant :IntegerParamDef
+parameter
LinIfResponseErrorBitPos :IntegerParamDef
+parameter
LinIfJitter :FloatParamDef
+subContainer
LinIfNodeComposition : ParamConfContainerDef
+parameter
LinIfNodeName :StringParamDef

Figure 23 - LIN Interface configuration

10.3.1 LinIf

Module Name	**LinIf**
Module Description	Configuration of the LinIf (LIN Interface) module.

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

10.3.2 LinIfGeneral

SWS Item	**--**
Container Name	LinIfGeneral
Description	--
Configuration Parameters	

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

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

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

SWS Item	**--**		
Name	LinIfTpSupported {LINIF_TP_SUPPORTED}		
Description	States if the TP is included in the LIN Interface or not. The reason for this parameter is to reduce the size of LIN Interface if the TP is not used.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

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

No Included Containers

10.3.3 LinIfGlobalConfig

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

SWS Item	**--**
Name	LinIfTimeBase {LINIF_TIME_BASE}
Description	The delay between processing two frames is a multiple of the LIN Interface

	time-base in seconds.		
Multiplicity	1		
Type	FloatParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

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

10.3.4 LinIfChannel

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

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

SWS Item	**--**		
Name	LinIfScheduleRequestQueueLength {LINIF_SCHEDULE_REQUEST_QUEUE_LENGTH}		
Description	Number of schedule requests the schedule table manager can handle for this channel.		
Multiplicity	1		
Type	IntegerParamDef		
Range	1 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	**--**
Name	LinIfChannelRef {LINIF_CHANNEL_INDEX}

Description	Reference to the used channel in Lin. Replaces LINIF_CHANNEL_INDEX		
Multiplicity	1		
Type	Reference to LinChannel		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency			

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
LinIfFrame	0..*	Generic container for all types of LIN frames.
LinIfMaster	1	Each Master can only be connected to one physical channel. This could be compared to the Node parameter in a LDF file.
LinIfScheduleTable	1..*	Describes a schedule table. Each LinIfChannel may have several schedule tables. Each schedule table can only be connected to one channel.
LinIfSlave	0..*	The Node attributes of the Slaves are provided with these parameter.
LinIfWakeUpSource	0..1	This container contains the configuration (parameters) needed to configure a wakeup capable channel

10.3.5 LinIfFrame

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

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

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

Scope / Dependency	

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

SWS Item	**--**		
Name	LinIfFrameType {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.		
Multiplicity	1		
Type	EnumerationParamDef		
Range	ASSIGN	AssignFrameId	
	ASSIGN_NAD	AssignNAD	
	CONDITIONAL	Conditional Change NAD	
	EVENT_TRIGGERED	Event triggered frame	
	FREE	FreeFormat	
	MRF	MRF	
	SPORADIC	Sporadic frame	
	SRF	SRF	
	UNASSIGN	UnassignFrameId	
	UNCONDITIONAL	Unconditional Frame	
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

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

SWS Item	**--**
Name	LinIfPid {LINIF_PID}
Description	Protected ID of the LIN frame. There is no reason to calculate the Parity in

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

SWS Item	**--**		
Name	LinIfTxTargetPduId {LINIF_TARGET_PDU_ID}		
Description	Identifier of the frame for the upper layer		
Multiplicity	1		
Type	IntegerParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

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

10.3.6 LinIfFixedFrameSdu

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

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

SWS Item	**--**

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

No Included Containers

10.3.7 LinIfPduDirection

SWS Item	**--**
Choice Container Name	LinIfPduDirection{LINIF_DIRECTION}
Description	Direction of the frame

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

10.3.8 LinIfRxPdu

SWS Item	**--**
Container Name	LinIfRxPdu
Description	represents a received PDU/frame
Configuration Parameters	

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

SWS Item	**--**		
Name	LinIfRxPduRef		
Description	Reference to the PDU that is received in this frame.		
Multiplicity	1		
Type	Reference to Pdu		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency			

No Included Containers

10.3.9 LinIfTxPdu

SWS Item	**--**
Container Name	LinIfTxPdu
Description	represents a transmitted PDU/frame
Configuration Parameters	

SWS Item	**--**		
Name	LinIfTxPduId {LINIF_PDU_ID}		
Description	Identifier of the frame for the upper layer. This id is only relevant for sporadic frames.		
Multiplicity	1		
Type	IntegerParamDef (Symbolic Name generated for this parameter)		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	**--**		
Name	LinIfTxPduRef		
Description	Reference to the PDU that is transmitted in this frame.		
Multiplicity	1		
Type	Reference to Pdu		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency			

No Included Containers

10.3.10 LinIfScheduleTable

SWS Item	**LINIF365 :**
Container Name	LinIfScheduleTable{LinIf_ScheduleTable}
Description	Describes a schedule table. Each LinIfChannel may have several schedule

	tables. Each schedule table can only be connected to one channel.
Configuration Parameters	

SWS Item	**--**		
Name	LinIfResumePosition		
Description	Defines, where a schedule table shall be proceeded in case if it has been interrupted by a run-once table or MRF/SRF.		
Multiplicity	1		
Type	EnumerationParamDef		
Range	CONTINUE_AT_IT_POINT	=0 Start from the beginning	
	START_FROM_BEGINNING	=0 Start from the beginning	
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

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

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

SWS Item	**--**		
Name	LinIfScheduleTableIndex {LINIF_SCHEDULE_INDEX}		
Description	This is the unique index used by upper layers to identify a schedule. Note that the NULL_SCHEDULE for each channel has index 0.		
Multiplicity	1		
Type	IntegerParamDef (Symbolic Name generated for this parameter)		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE

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

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

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

10.3.11 LinIfEntry

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

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

SWS Item	**--**		
Name	LinIfEntryIndex		
Description	Position of the Frame Entry in the Schedule Table.		
Multiplicity	1		
Type	IntegerParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**--**		
Name	LinIfCollisionResolvingRef		
Description	Reference to the schedule table, which resolves the collision.		
Multiplicity	0..1		
Type	Reference to LinIfScheduleTable		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency			

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

No Included Containers

10.3.12 LinIfMaster

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

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

No Included Containers

10.3.13 LinIfSlave

SWS Item	**--**
Container Name	LinIfSlave

Description	The Node attributes of the Slaves are provided with these parameter.
Configuration Parameters	

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

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

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

SWS Item	**--**		
Name	LinIfResponseErrorBitPos {LINIF_RESPONSE_ERROR_BIT_POS}		
Description	Specifies the frame and the position in the frame		
Multiplicity	1		
Type	IntegerParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**--**
Name	LinIfSupplierId {LINIF_SUPPLIER_ID}
Description	LIN Supplier ID
Multiplicity	1
Type	IntegerParamDef

Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

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

SWS Item	**--**		
Name	LinIfResponseErrorEventRef		
Description	Reference to DEM Event		
Multiplicity	1		
Type	Reference to DemEventParameter		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**--**		
Name	LinIfResponseErrorFrameRef		
Description	Reference to the frame which contains the response error bit.		
Multiplicity	1		
Type	Reference to LinIfFrame		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency			

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

10.3.14 LinIfNodeComposition

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

SWS Item	**--**

Name	LinIfNodeName {LINIF_NODE_NAME}		
Description	--		
Multiplicity	1		
Type	StringParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

No Included Containers

10.3.15 LinIfSlaveToSlavePdu

SWS Item	**--**
Container Name	LinIfSlaveToSlavePdu
Description	represents a slave-to-slave PDU/frame. Master does only send the header but doesn't receive the response. Added for completeness
Configuration Parameters	

No Included Containers

10.3.16 LinIfInternalPdu

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

No Included Containers

10.3.17 LinIfWakeUpSource

SWS Item	**--**
Container Name	LinIfWakeUpSource{LINIF_WAKEUP_SOURCE}
Description	This container contains the configuration (parameters) needed to configure a wakeup capable channel
Configuration Parameters	

SWS Item	**--**
Name	LinIfChannelWakeupInfo
Description	If the wakeup-capability is true the wakeup source referenced is transmitted to the ECU State Manager (EcuM) . Implementation Type: reference to EcuM_WakeupSourceType
Multiplicity	1

Type	Reference to EcuMWakeupSource		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency			

No Included Containers

10.4 LIN Transport Layer configuration

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

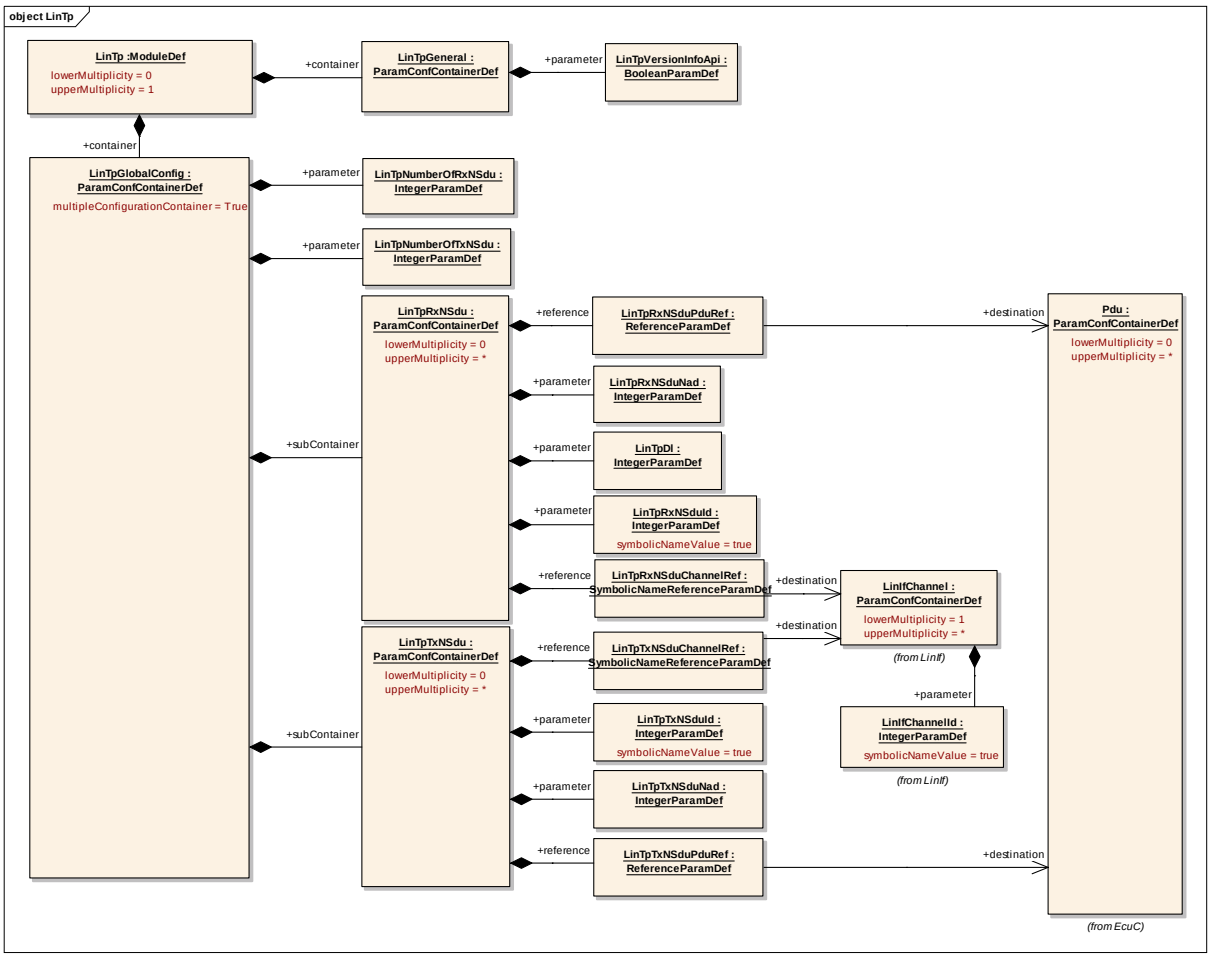

Figure 24 – LIN Transport Protocol configuration

10.4.1 LinTp

Module Name	**LinTp**
Module Description	Singleton descriptor for the LIN Transport Protocol.

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
LinTpGeneral	1	Container that holds all LIN transport protocol general parameters.
LinTpGlobalConfig	1	This container contains the global configuration parameter of the LinTp. It is a MultipleConfigurationContainer, i.e. this container and its sub-containers exit once per configuration set.

10.4.2 LinTpGeneral

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

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

No Included Containers

10.4.3 LinTpRxNSdu

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

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

SWS Item	**--**
Name	LinTpRxNSduId {LINTP_NSDU_ID}
Description	The identifier of the Transport Protocol message. This ID will be the one that is communicated with upper layers.

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

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

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

SWS Item	**--**		
Name	LinTpRxNSduPduRef		
Description	Reference to the global PDU		
Multiplicity	1		
Type	Reference to Pdu		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency			

No Included Containers

10.4.4 LinTpTxNSdu

SWS Item	**LINIF428 :**
Container Name	LinTpTxNSdu{LinTp_tx_NSDU}
Description	For each transmitted N-SDU on any channel the node is connected to.
Configuration Parameters	

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

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

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

SWS Item	**--**		
Name	LinTpTxNSduPduRef		
Description	Reference to the global PDU		
Multiplicity	1		
Type	Reference to Pdu		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency			

No Included Containers

10.5 Published Information

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

The standard common published information like

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

is provided in the BSW Module Description Template (see [13] Figure 4.1 and Figure 7.1).

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

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
LINIF504	Bug 44994

11.4 Added SWS Items

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

12 Changes during SWS Improvements by Technical Office

12.1 Deleted SWS Items

SWS Item	*Rationale*
LINIF377	No requirement on the LIN Interface
LINIF011	No requirement on the LIN Interface
LINIF435	Redundant to LINIF201
LINIF257	Merged it with LINIF466
LINIF037	Merged with LINIF465
LINIF443	Redundant to LINIF204
LINIF445	Redundant to LINIF468
LINIF446	Redundant to LINIF205
LINIF468	Redundant to LINIF378
LINIF300	Merged it with LINIF464
LINIF447	Redundant to LINIF198
LINIF448	Redundant to LINIF355
LINIF481	Redundant to LINIF318
LINIF430	Requirement on the PDU Router
LINIF451	Redundant to LINIF351
LINIF456	Redundant to LINIF422

12.2 Replaced SWS Items

SWS Item of Release 1	*replaced by SWS Item*	*Rationale*
LINIF250	LINIF470, LINIF471	Made requirement atomic.
LINIF284	LINIF473, LINIF474	Made requirement atomic.
LINIF229	LINIF479, LINIF480	Made requirement atomic.
LINIF449	LINIF481, LINIF482	Made requirement atomic.
LINIF317	LINIF483, LINIF484	Made requirement atomic.

12.3 Changed SWS Items

Many requirements have been changed to improve understandability without changing the technical contents.

12.4 Added SWS Items

SWS Item	*Rationale*
LINIF469	UML model linking of imported types
LINIF472	Requirement on handling reserved frames.
LINIF475	Requirement on LIN channel state-machine
LINIF476	Requirement on LIN channel state-machine
LINIF478	Requirement on LIN channel state-machine
LINIF486	Requirement on LinIf_Init (previous ID was LINIF271 which did exist 2x)
LINIF487	Requirement on LinIf_GetVersionInfo
LINIF488	Requirement on LinIf_GotoSleep
LINIF489	Requirement on LinTp_CancelTransmitRequest

LINIF490	Requirement on LinTp_CancelTransmitRequest
LINIF491	Definition of configuration variant needs an ID
LINIF492	Definition of configuration variant needs an ID
LINIF493	Definition of configuration variant needs an ID