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

Document Version	4.0.0
Document Status	Final
Part of Release	4.0
Revision	3

Document Change History

Date	Version	Changed by	Change Description
28.10.2011	4.0.0	AUTOSAR Administration	• Added the As/Cs/Cr timeout observation for LIN TP. • Clarified the buffer handling requirement for LIN TP. • Deleted CDD for LIN TP. • Added the specification of transceiver wakeup.
15.11.2010	3.1.0	AUTOSAR Administration	• Added 5.3.3 Version Check. • Changed from the parameter name “NetworkHandleType Transceiver” to “NetworkHandleType Channel” • Changed the type definitions and deleted from LIN Interface: LinIf_TrcvModeType → LinTrcv_TrcvModeType, LinTp_ParameterValueType → TpParameterType • Changed the function name with “WakeUp” to “Wakeup” • Changed the configuration parameter for time to “in second”
30.11.2009	3.0.0	AUTOSAR Administration	• Support of LIN 2.1 Specification • Added support for LIN Transceiver Driver • LIN schedule table manager removed due to Basic Software Modemanager which controls this now • Interaction with Complex Device Driver extended • Legal disclaimer revised
23.06.2008	2.0.1	AUTOSAR Administration	• Legal disclaimer revised

Document Change History			
Date	**Version**	**Changed by**	**Change Description**
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, LIN Driver and LIN Transceiver Driver.

The base for this document is the LIN 2.1 specification [18]. It is assumed that the reader is familiar with this specification. This document will not describe LIN 2.1 functionality again but it will try to follow the same order as the LIN 2.1 specification.

The LIN Interface module applies to LIN 2.1 master nodes only. Operating as a slave node is out of scope. The LIN master in AUTOSAR deviates from the LIN 2.1 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.

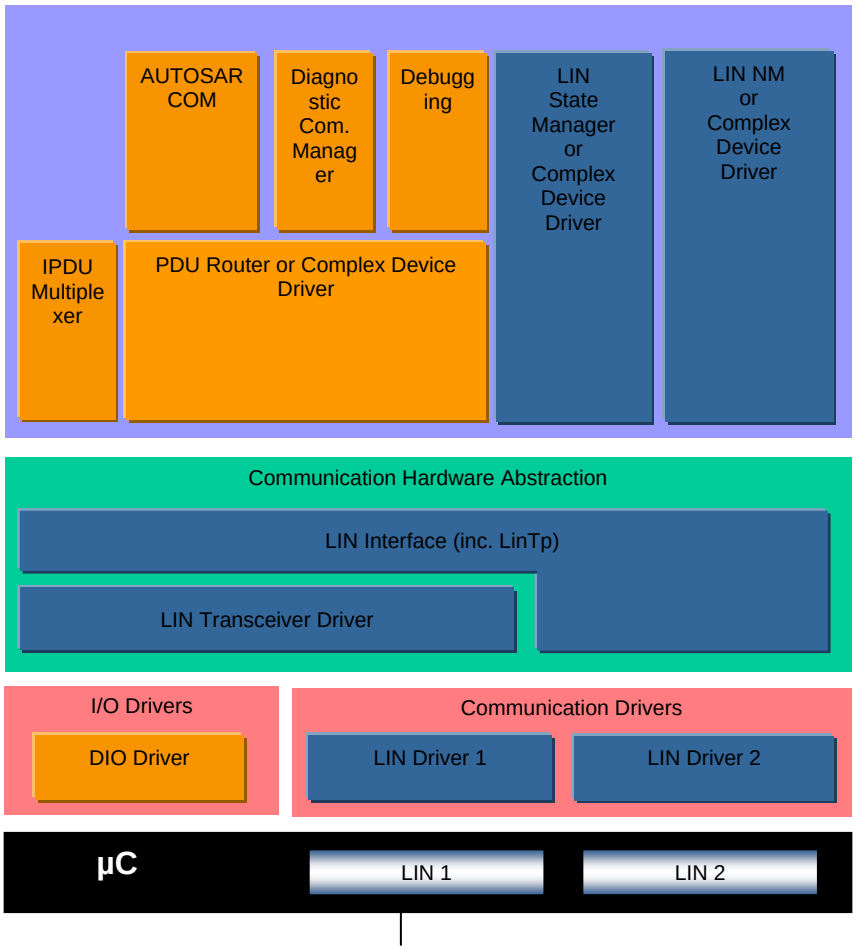

Figure 1 – AUTOSAR BSW software architecture (LIN relevant modules)

1.2 Functional overview

The LIN Interface is responsible for providing LIN 2.1 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 2.1 specification, the following acronyms and abbreviations are used throughout this document. Some terms already defined in the LIN 2.1 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
Delay	The time between start of frames in a schedule table. The unit is in number of time-bases for the specific cluster.
DEM	Diagnostic Event Manager
DET	Development Error Tracer
EcuM	ECU State Manager
FF	First Frame in TP
LDF	LIN Description File
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 2.1 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
N_As	Time for transmission of the LIN frame (any N-PDU) on the sender side (see ISO 15765-2).
N_Cr	Time until reception of the next consecutive frame N-PDU (see ISO 15765-2).
N_Cs	Time until transmission of the next consecutive frame N-PDU (see ISO 15765-2).
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 2.1 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_TR_BSWModuleList.pdf

[2] Layered Software Architecture
AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf

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

[4] Specification of Standard Types
AUTOSAR_SWS_StandardTypes.pdf

[5] Specification of Development Error Tracer
AUTOSAR_SWS_DevelopmentErrorTracer.pdf

[6] Requirements on LIN
AUTOSAR_SRS_LIN.pdf

[7] Specification of LIN Driver
AUTOSAR_SWS_LINDriver.pdf

[8] Specification of Diagnostic Event Manager
AUTOSAR_SWS_DiagnosticEventManager.pdf

[9] Specification of ECU Configuration
AUTOSAR_TPS_ECUConfiguration.pdf

[10] Specification of ECU State Manager
AUTOSAR_SWS_ECUStateManager.pdf

[11] Specification of BSW Scheduler
AUTOSAR_SWS_BSW_Scheduler.pdf

[12] Specification of LIN State Manager
AUTOSAR_SWS_LINStateManager.pdf

[13] Basic Software Module Description Template
AUTOSAR_TPS_BSWModuleDescriptionTemplate.pdf

[14] Specification of LIN Transceiver Driver
AUTOSAR_SWS_LINTransceiverDriver.pdf

[15] Specification of PDU Router
AUTOSAR_SWS_PDURouter.pdf

[16] Specification of Communication Stack Types
AUTOSAR_SWS_CommunicationStackTypes.pdf

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

3.2 Related standards and norms

[18] LIN Specification Package Revision 2.1, November 24, 2006
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
- LIN Transceiver Driver – LinTrcv
- PDU Router – PduR
- Development Error Tracer – DET
- Diagnostic Event Manager – DEM
- ECU State Manager – EcuM
- LIN State Manager – LinSM
- BSW Mode Manager – BswM

5.1 Upper layers

5.1.1 PDU Router

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

5.1.2 BSW Scheduler

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

5.1.3 Operating System

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

5.1.4 Module DET (Development Error Tracer)

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

5.1.5 Module DEM (Diagnostic Event Manager)

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

5.1.6 Module ECU State Manager

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

1. The ECU state manager initializes the LIN Interface.

5.1.7 Module LIN State Manager

The LIN Interface connects to the LIN state manager and/or alternative modules above (e.g. Complex Device Driver) which 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.

The possibility of having a choice of the module above is expressed e.g. in callback functions by prefixing names with “<User>_”.

5.1.8 Module BSW Mode Manager

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

5.2 Lower layers

5.2.1 LIN Driver

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

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

- Transmission of the header part of a frame (Lin_SendFrame). 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_SendFrame).
- Transmission of the go-to-sleep command (Lin_GoToSleep).
- Setting a LIN channel to state LIN_CH_SLEEP without transmitting a go-to-sleep command (Lin_GoToSleepInternal).

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

[LINIF129] ⌈The LIN Interface shall not use or access the LIN hardware or assume information about it any way other than what the LIN Driver provides through the function calls to the LIN Driver listed above.⌋ (BSW161, BSW162, BSW006, BSW01568)

5.2.2 LIN Transceiver Driver

Optionally, the LIN Interface requires the services of the underlying LIN Transceiver Driver specified by [14].

The LIN Interface maps the following services for all underlying LIN Transceiver Drivers to one unique interface.

- Unique LIN Transceiver Driver mode request and read services to manage the operation modes of each underlying LIN transceiver device.
- Read service for LIN transceiver wake up reason support.
- Mode request service to enable/disable/clear wake up event state of each used LIN transceiver.

[LINIF534] ⌈The LIN Interface shall not use or access the LIN hardware or assume information about it any way other than what the LIN Transceiver Driver provides through the function calls to the LIN Transceiver 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. ⌋ (BSW00380, BSW00419, BSW158)

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. ⌋ (BSW158, BSW00302)

[LINIF245] ⌈The header file LinIf_Cfg.h shall contain declarations of the pre-compile time configurable parameters.

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

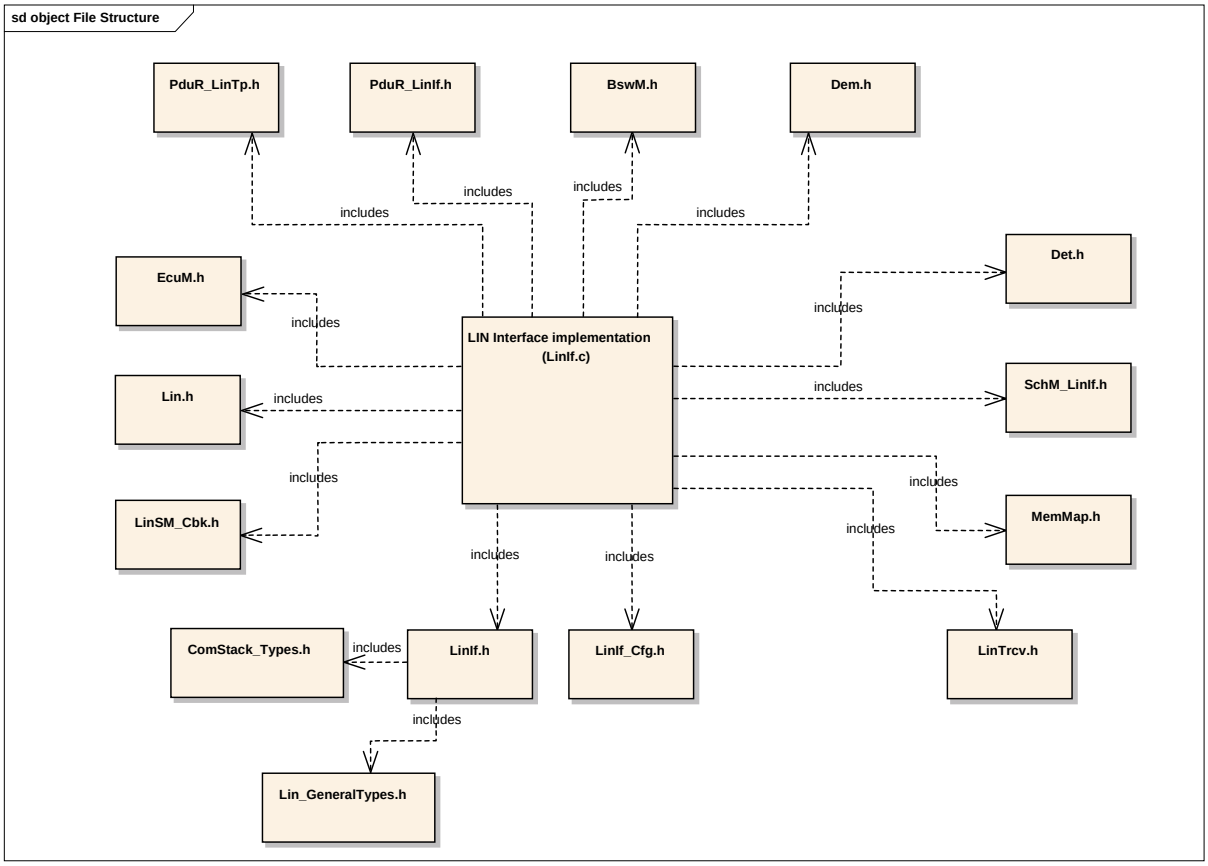

Figure 2 – Header file structure

The LIN Interface implementation object in Figure 2 represents one or more c-files. It is not required to make the complete implementation in one file. ⌋ (BSW00345, BSW00381, BSW00412, BSW00415, BSW158)

[LINIF247] ⌈The LIN Interface shall include the file Dem.h. ⌋()

[LINIF434] ⌈The LIN Interface shall include the file Lin.h. ⌋()

[LINIF589] ⌈The LIN Interface shall include the file SchM_LinIf.h. ⌋(BSW00435)

[LINIF590] ⌈The LIN Interface shall include the file MemMap.h. ⌋(BSW00436)

[LINIF497] ⌈The LIN Interface shall include the defined include files of all upper layer BSW modules it is connected to, e.g. in case of connection to the PDU Router the file PduR_LinIf.h. ⌋()

[LINIF498] ⌈The LIN Interface shall include the file Det.h if the configuration parameter LinIfDevErrorDetect is enabled. ⌋()

[LINIF499] ⌈The LIN Interface shall include the file ComStack_Types.h. ⌋()

[LINIF638] ⌈The LIN Interface shall include the file Lin_GeneralTypes.h. ⌋()

[LINIF561] ⌈The LIN Interface shall include the file PduR_LinTp.h, if the LinTp is enabled (configuration parameter LinIfTpSupported). ⌋()

[LINIF555] ⌈The LIN Interface shall include the file LinTrcv.h, if and only if the configuration parameter LinIfTrcvDriverSupported is set to TRUE. ⌋()

[LINIF556] ⌈The LIN Interface shall include the file LinSM_Cbk.h. ⌋()

[LINIF650] ⌈The LIN Interface shall include the file BswM.h. ⌋()

[LINIF690] ⌈ The LIN Interface shall include the file EcuM.h. ⌋()

[LINIF669] ⌈The LIN Interface shall include the file <CDD_Cbk.h> for callback declaration of CDD. <CDD_Cbk.h> is configurable via configuration parameter LinIfPublicCddHeaderFile. ⌋()

5.3.3 Version check

[LINIF383] ⌈The LIN Interface module shall perform Inter Module Checks to avoid integration of incompatible files. The imported include files shall be checked by preprocessing directives.

The following version numbers shall be verified:

- <MODULENAME>_AR_RELEASE_MAJOR_VERSION
- <MODULENAME>_AR_RELEASE_MINOR_VERSION

Where <MODULENAME> is the module short name of the other (external) modules which provide header files included by the LIN Interface module.

If the values are not identical to the expected values, an error shall be reported. ⌋ (BSW004)

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).

Requirement	Satisfied by
-	LINIF581
-	LINIF341
-	LINIF426
-	LINIF450
-	LINIF536
-	LINIF479
-	LINIF618
-	LINIF615
-	LINIF565
-	LINIF105
-	LINIF389
-	LINIF259
-	LINIF689
-	LINIF254
-	LINIF677
-	LINIF263
-	LINIF247
-	LINIF519
-	LINIF412
-	LINIF444
-	LINIF576
-	LINIF357
-	LINIF564
-	LINIF231
-	LINIF417
-	LINIF529
-	LINIF463
-	LINIF081
-	LINIF488
-	LINIF635
-	LINIF690
-	LINIF669
-	LINIF454
-	LINIF538
-	LINIF557
-	LINIF501

-	LINIF596
-	LINIF569
-	LINIF087
-	LINIF308
-	LINIF597
-	LINIF546
-	LINIF455
-	LINIF271
-	LINIF562
-	LINIF422
-	LINIF427
-	LINIF043
-	LINIF621
-	LINIF507
-	LINIF039
-	LINIF433
-	LINIF309
-	LINIF410
-	LINIF322
-	LINIF330
-	LINIF622
-	LINIF416
-	LINIF415
-	LINIF356
-	LINIF467
-	LINIF053
-	LINIF534
-	LINIF260
-	LINIF286
-	LINIF528
-	LINIF658
-	LINIF432
-	LINIF405
-	LINIF224
-	LINIF671
-	LINIF360
-	LINIF616
-	LINIF652
-	LINIF029
-	LINIF623
-	LINIF460
-	LINIF548

-	LINIF189
-	LINIF258
-	LINIF595
-	LINIF568
-	LINIF482
-	LINIF321
-	LINIF638
-	LINIF408
-	LINIF586
-	LINIF522
-	LINIF646
-	LINIF500
-	LINIF237
-	LINIF452
-	LINIF637
-	LINIF486
-	LINIF610
-	LINIF359
-	LINIF577
-	LINIF483
-	LINIF571
-	LINIF462
-	LINIF261
-	LINIF663
-	LINIF670
-	LINIF642
-	LINIF287
-	LINIF069
-	LINIF608
-	LINIF472
-	LINIF667
-	LINIF176
-	LINIF620
-	LINIF329
-	LINIF640
-	LINIF293
-	LINIF602
-	LINIF434
-	LINIF685
-	LINIF593
-	LINIF573
-	LINIF563

-	LINIF673
-	LINIF068
-	LINIF076
-	LINIF580
-	LINIF327
-	LINIF012
-	LINIF540
-	LINIF552
-	LINIF575
-	LINIF627
-	LINIF655
-	LINIF461
-	LINIF537
-	LINIF521
-	LINIF414
-	LINIF351
-	LINIF496
-	LINIF647
-	LINIF036
-	LINIF688
-	LINIF014
-	LINIF644
-	LINIF579
-	LINIF668
-	LINIF090
-	LINIF679
-	LINIF498
-	LINIF556
-	LINIF233
-	LINIF636
-	LINIF684
-	LINIF326
-	LINIF353
-	LINIF539
-	LINIF617
-	LINIF073
-	LINIF674
-	LINIF249
-	LINIF506
-	LINIF643
-	LINIF485
-	LINIF683

-	LINIF315
-	LINIF320
-	LINIF665
-	LINIF520
-	LINIF661
-	LINIF409
-	LINIF270
-	LINIF458
-	LINIF560
-	LINIF657
-	LINIF676
-	LINIF075
-	LINIF634
-	LINIF566
-	LINIF323
-	LINIF629
-	LINIF601
-	LINIF558
-	LINIF404
-	LINIF619
-	LINIF659
-	LINIF066
-	LINIF085
-	LINIF473
-	LINIF530
-	LINIF289
-	LINIF656
-	LINIF086
-	LINIF471
-	LINIF470
-	LINIF555
-	LINIF413
-	LINIF654
-	LINIF645
-	LINIF106
-	LINIF549
-	LINIF503
-	LINIF666
-	LINIF678
-	LINIF499
-	LINIF080
-	LINIF567

-	LINIF551
-	LINIF584
-	LINIF664
-	LINIF296
-	LINIF397
-	LINIF561
-	LINIF686
-	LINIF225
-	LINIF592
-	LINIF478
-	LINIF290
-	LINIF675
-	LINIF490
-	LINIF497
-	LINIF625
-	LINIF600
-	LINIF495
-	LINIF626
-	LINIF624
-	LINIF588
-	LINIF267
-	LINIF682
-	LINIF649
-	LINIF681
-	LINIF381
-	LINIF541
-	LINIF436
-	LINIF487
-	LINIF613
-	LINIF453
-	LINIF641
-	LINIF650
-	LINIF578
-	LINIF662
-	LINIF660
-	LINIF078
-	LINIF113
-	LINIF419
-	LINIF574
-	LINIF484
-	LINIF609
-	LINIF278

-	LINIF030
-	LINIF023
-	LINIF648
-	LINIF572
-	LINIF612
-	LINIF653
-	LINIF680
-	LINIF672
-	LINIF570
-	LINIF226
-	LINIF594
-	LINIF272
-	LINIF614
-	LINIF639
-	LINIF418
BSW003	LINIF692
BSW00302	LINIF242
BSW00321	LINIF692
BSW00323	LINIF269
BSW00327	LINIF376
BSW00328	LINIF386
BSW00331	LINIF692
BSW00333	LINIF692
BSW00334	LINIF692
BSW00335	LINIF442, LINIF441, LINIF439, LINIF438, LINIF319, LINIF316
BSW00336	LINIF355
BSW00337	LINIF376
BSW00338	LINIF376
BSW00339	LINIF376
BSW00341	LINIF692
BSW00343	LINIF474
BSW00344	LINIF371, LINIF373
BSW00345	LINIF245
BSW00350	LINIF268
BSW00358	LINIF350, LINIF198
BSW00359	LINIF692
BSW00360	LINIF692
BSW00373	LINIF384
BSW00375	LINIF378
BSW00376	LINIF384
BSW00380	LINIF241
BSW00381	LINIF245

BSW00385	LINIF376
BSW00386	LINIF268
BSW004	LINIF383
BSW00404	LINIF371, LINIF373
BSW00405	LINIF371, LINIF373
BSW00406	LINIF376, LINIF535, LINIF687
BSW00407	LINIF352, LINIF340
BSW00409	LINIF266
BSW00411	LINIF279, LINIF354
BSW00412	LINIF245
BSW00413	LINIF469, LINIF197
BSW00414	LINIF350, LINIF198
BSW00415	LINIF245
BSW00416	LINIF350, LINIF198
BSW00417	LINIF692
BSW00419	LINIF241
BSW00422	LINIF692
BSW00425	LINIF248
BSW00431	LINIF692
BSW00432	LINIF692
BSW00433	LINIF692
BSW00434	LINIF692
BSW00435	LINIF589
BSW00436	LINIF590
BSW00437	LINIF692
BSW00439	LINIF692
BSW00440	LINIF692
BSW006	LINIF129
BSW010	LINIF692
BSW01502	LINIF033, LINIF128
BSW01504	LINIF248
BSW01514	LINIF378
BSW01515	LINIF205
BSW01523	LINIF204
BSW01527	LINIF465, LINIF466
BSW01534	LINIF062
BSW01540	LINIF350
BSW01544	LINIF651, LINIF079
BSW01545	LINIF098
BSW01546	LINIF028, LINIF393, LINIF384
BSW01549	LINIF474
BSW01551	LINIF386

BSW01555	LINIF384
BSW01558	LINIF033, LINIF128
BSW01560	LINIF186, LINIF459
BSW01561	LINIF384
BSW01564	LINIF202
BSW01568	LINIF129
BSW01569	LINIF198
BSW01570	LINIF371
BSW01571	LINIF201
BSW01574	LINIF314
BSW01576	LINIF248
BSW01577	LINIF248
BSW01579	LINIF313
BSW01584	LINIF544
BSW01585	LINIF544
BSW01586	LINIF544
BSW01587	LINIF545
BSW01588	LINIF547
BSW01589	LINIF550
BSW01590	LINIF401
BSW101	LINIF350, LINIF198
BSW158	LINIF242, LINIF241, LINIF245
BSW161	LINIF129
BSW162	LINIF129
BSW170	LINIF373
BSW171	LINIF387, LINIF310
BSW33200002	LINIF516
BSW33200022	LINIF518, LINIF517, LINIF515

7 Functional specification

This chapter is organized in a way following the same order as the LIN 2.1 specification. This is not always the case since the LIN 2.1 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 2.1 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 2.1 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 2.1 specification.⌋(BSW00425, BSW01576, BSW01504, BSW01577)

The requirement above basically means that the communication from a LIN 2.1 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.⌋(BSW00328, BSW01551)

7.1 Frame Transfer

All the functionality of the Protocol Specification in the LIN 2.1 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 2.1 specification. The existing frame types are:

- Unconditional frame
- Event-triggered frame
- Sporadic frame
- Diagnostic frames MRF and SRF
- Reserved frames

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

7.1.1.1 Unconditional frame

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

7.1.1.2 Event-triggered frame

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

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

[LINIF588] ⌈If a collision occurs in an event-triggered frame response, then the LIN Interface shall switch to the corresponding collision resolving schedule table. ⌋()

[LINIF176] ⌈The LIN Interface shall switch to the given collision resolving schedule table at the end of the current frame slot after a collision has been detected. ⌋()

[LINIF519] ⌈The collision resolving schedule table is given by the LIN Interface configuration (configuration parameter LinIfCollisionResolvingRef). ⌋()

7.1.1.3 Sporadic frame

The LIN 2.1 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 2.1 specification does not specify how slaves may transmit sporadic frames.

[LINIF012] ⌈The master shall be the only allowed transmitter of a sporadic frame (defined in the LIN 2.1 specification). ⌋()

[LINIF436] ⌈Only a sporadic frame shall allocate a sporadic slot (defined in the LIN 2.1 specification).

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

[LINIF470] ⌈The LIN Interface shall flag the specific sporadic frame (defined in the LIN 2.1 specification) for transfer.⌋()

[LINIF471] ⌈The LIN Interface shall transmit the specific sporadic frame (defined in the LIN 2.1 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 2.1 specification) allocated to the same schedule slot is defined by the configuration parameter LinIfFramePriority.⌋()

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 2.1 node configuration services and TP messages.

The LIN 2.1 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 Reserved frames

The LIN 2.1 specification does not allow reserved frames.

[LINIF472] ⌈The LIN Interface shall not use reserved frames (defined in the LIN 2.1 specification). ⌋ ()

7.1.2 Frame reception

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

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

7.1.2.1 Header

[LINIF419] ⌈The LIN Interface shall call the function Lin_SendFrame of the LIN Driver module when a new schedule entry for a frame reception is due. ⌋ ()

7.1.2.2 Response

The LIN Driver will automatically be set to reception state after the header is transmitted.

7.1.2.3 Status check

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

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

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

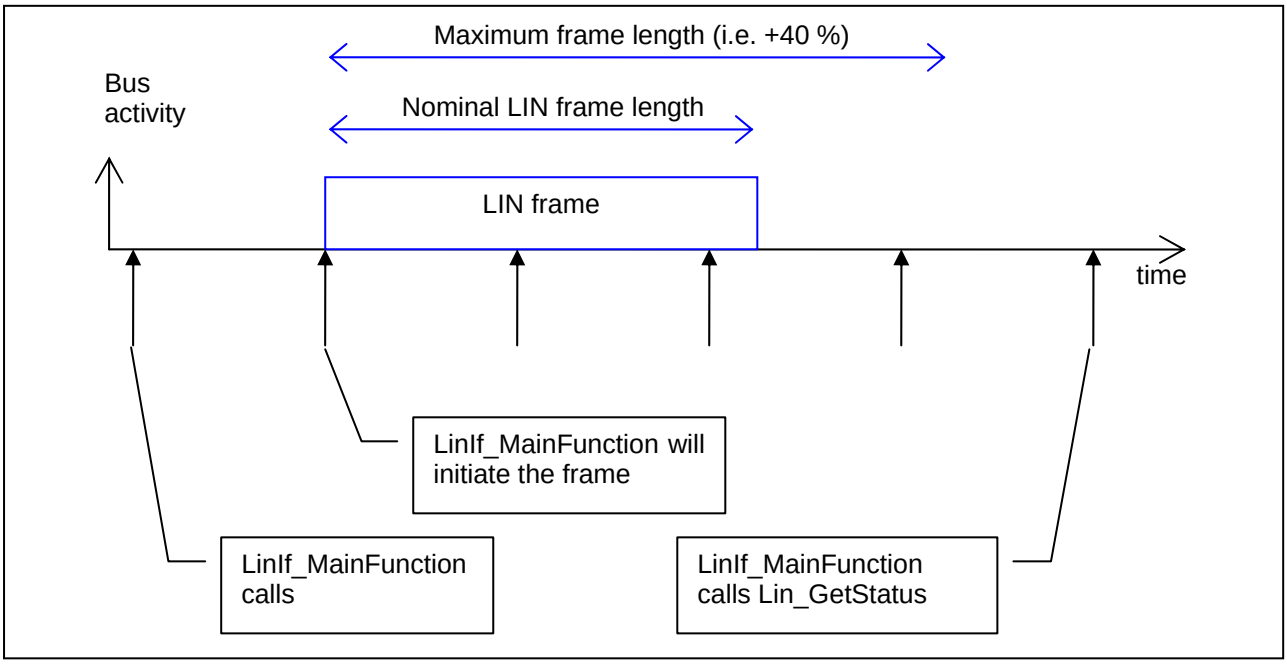

Figure 3 – Lin_GetStatus call example

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

[LINIF033] ⌈The LIN Interface shall invoke <User_RxIndication> with the received data, only when LIN Interface determines the LIN Driver module's status is LIN_RX_OK. ⌋(BSW01502, BSW01558)

[LINIF259] ⌈When the LIN Interface is receiving an event-triggered frame and the LIN Driver module's status is LIN_RX_BUSY or LIN_RX_ERROR, the LIN Interface shall not consider the status as an error. ⌋()

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

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

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

[LINIF254] ⌈When the LIN Interface has 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. Therefore, the LIN Interface shall raise the development error code LINIF_E_RESPONSE, if the development error detection is enabled. ⌋()

[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 development error code LINIF_E_RESPONSE, if it is not an event-triggered frame and the development error detection is enabled.

If there is disturbance on the bus, the LIN Interface may have problems sending out the header. The philosophy of the LIN 2.1 specification in this case is not reporting the error to upper layers. The same behavior applies also for transmitted and slave-to-slave frames.⌋(BSW01527)

[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_SendFrame when a new schedule entry for a frame transmission is due.⌋()

7.1.3.2 Response

[LINIF225] ⌈Before invoking the function Lin_SendFrame, the LIN Interface shall call the function <User_TriggerTransmit> to get the data part of the frame (data in the LIN frame response).⌋()

[LINIF226] ⌈After getting the data part of the frame, the LIN Interface shall call the LIN Driver module's function Lin_SendFrame to provide the LIN Driver a pointer to the data part.⌋()

7.1.3.3 Status check

[LINIF128] ⌈If the return code of the function Lin_GetStatus is LIN_TX_OK, the LIN Interface shall issue a <User_TxConfirmation> callback.⌋(BSW01502, BSW01558)

[LINIF036] ⌈If the return code of the function Lin_GetStatus is LIN_TX_ERROR and any LIN frame transmission is attempted, the LIN Interface shall consider the transmitted frame as lost.⌋()

[LINIF465] ⌈If, just before a new frame is transmitted, the return code of the function Lin_GetStatus is LIN_TX_BUSY, the LIN Interface shall consider the old frame as lost and raise the development error code LINIF_E_RESPONSE, if the development error detection is enabled.⌋(BSW01527)

[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_SendFrame when a new schedule entry for a slave-to-slave communication is due.⌋()

7.1.4.2 Response

[LINIF417] ⌈The LIN Interface shall not be involved in the slave-to-slave communication, in either transmission or reception of the response.⌋()

7.1.4.3 Status check

[LINIF418] ⌈The LIN Interface shall not check the LIN Driver module's status after the transportation of the slave-to-slave communication response.⌋()

7.2 Schedules

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

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

[LINIF260] ⌈The LIN Interface's integrator shall define a time-base (configuration parameter LinIfClusterTimeBase) for each LIN cluster. ⌋()

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

[LINIF231] ⌈The LIN Interface shall provide a predefined schedule table per channel (named NULL_SCHEDULE). ⌋()

[LINIF263] ⌈The schedule table NULL_SCHEDULE shall contain no frames. ⌋()

7.2.1 LinIf_MainFunction

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

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

The LIN Interface uses the Greatest Common Factor (GCF) of the time-base.

Further reference to this GCF value is referred to as "tick". ⌋(BSW00343, BSW01549)

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.

7.2.2 Schedule table manager

The schedule table manager is not defined in the LIN 2.1 specification.

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.

The schedule table manager of the LIN Interface supports 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 2.1 node configuration
- poll event-triggered or sporadic frames

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) the schedule table manager of the LIN Interface shall change to NULL_SCHEDULE when the next schedule entry is due (even if the current is RUN_ONCE). ⌋()

[LINIF467] ⌈The LIN Interface shall reject a request for a schedule if the corresponding channel is in the state CHANNEL_SLEEP and raise the development error code LINIF_E_SCHEDULE_REQUEST_ERROR if the development error detection is enabled. ⌋()

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

[LINIF028] ⌈The LIN Interface shall select a new schedule table for execution during the next schedule entry if the current schedule is RUN_CONTINUOUS. ⌋(BSW01546)

[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. ⌋(BSW01546)

[LINIF495] ⌈If the switch from one schedule table to another schedule table has been performed, the schedule table manager shall call the function <User>_ScheduleRequestConfirmation.

For the sporadic frames, a schedule table switch means that the states of these frames are not affected. ⌋()

[LINIF029] ⌈The state of sporadic frames shall not be cleared when the schedule table is changed.⌋()

[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 shall be configurable by the configuration parameter LinIfResumePosition.⌋()

Note: Since the function LinIf_Init will set the NULL_SCHEDULE it means that there is always a latest requested schedule table.

7.3 Network management

The network management described in this chapter is based on the LIN 2.1 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 2.1 specification how a node operates.

The LIN 2.1 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 2.1 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 4.⌋()

[LINIF438] ⌈The LIN Interface state-machine shall have the state LINIF_UNINIT.⌋(BSW00335)

[LINIF439] ⌈The LIN Interface state-machine shall have the state LINIF_INIT. ⌋ (BSW00335)

[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 4.

[LINIF290] ⌈Each LIN channel shall have a separate channel state-machine. ⌋()

[LINIF441] ⌈The LIN channel sub-state-machine shall have the state CHANNEL_OPERATIONAL. ⌋(BSW00335)

Note: In the LIN channel state CHANNEL_OPERATIONAL the corresponding LIN channel shall be initialized and operate normally.

[LINIF189] ⌈The LIN Interface shall transmit LIN frame headers and receive/transmit responses only when the corresponding LIN channel is in the state CHANNEL_OPERATIONAL. ⌋()

[LINIF053] ⌈In the state CHANNEL_OPERATIONAL, the LIN Interface shall process the currently selected schedule table within the function LinIf_MainFunction. ⌋()

[LINIF507] ⌈The LIN Interface shall transit from LINIF_UNINIT to

- CHANNEL_OPERATIONAL when the function LinIf_Init is called and the configuration parameter LinIfStartupState is set to NORMAL.
- CHANNEL_SLEEP when the function LinIf_Init is called and the configuration parameter LinIfStartupState is set to SLEEP. ⌋()

[LINIF442] ⌈The LIN channel sub-state-machine shall have the state CHANNEL_SLEEP. ⌋(BSW00335)

[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. ⌋()

Note: 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.⌋ ()

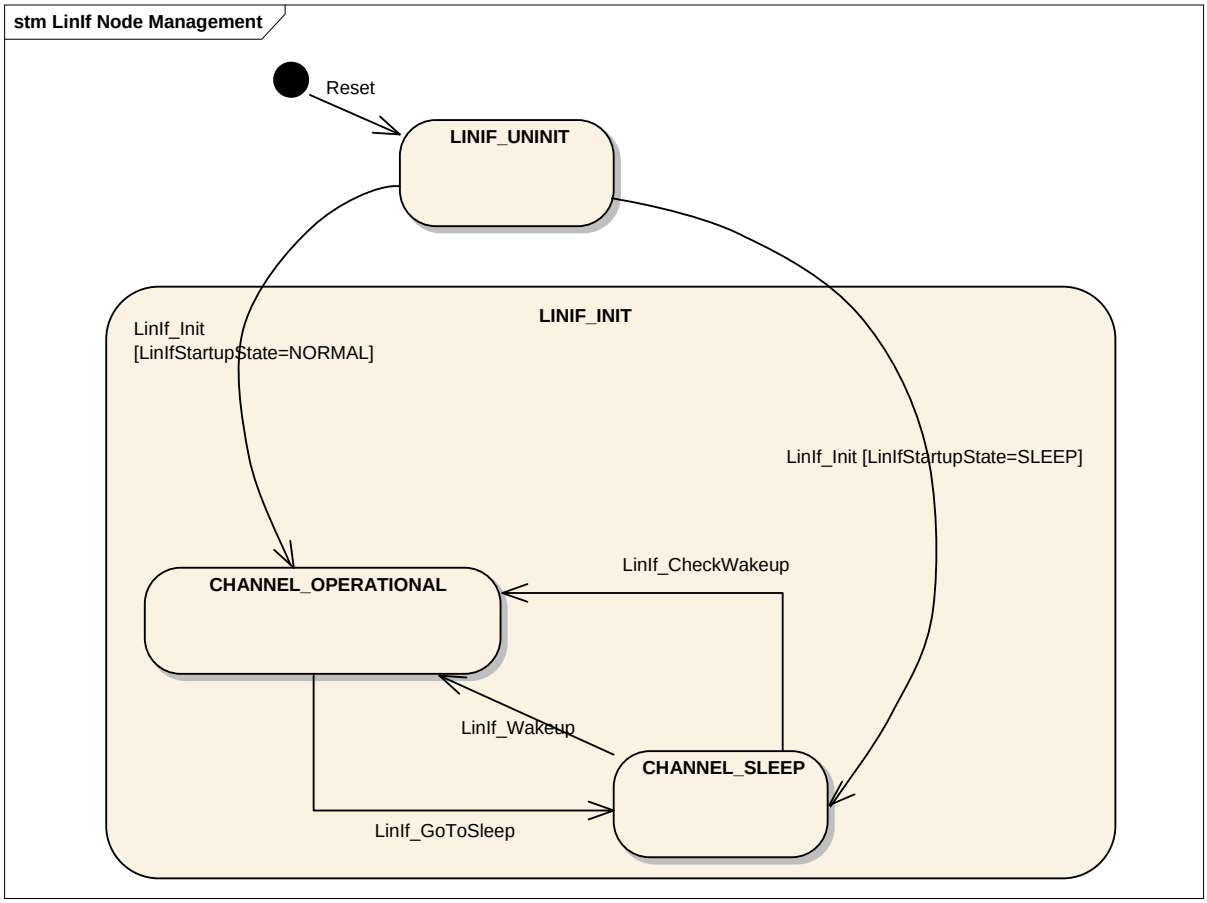

Figure 4 – LIN Interface state-machine and LIN Interface channel sub-state-machine

7.3.2 Go to sleep process

The function LinIf_GotoSleep initiates a transition into sleep mode on the selected channel/controller. The transition is carried out by transmitting a LIN diagnostic master request frame with its first data byte equal to 0 (zero). This is called the go-to-sleep command in the LIN 2.1 specification.

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

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

Rational: This will prevent a wake-up of the attached LIN slaves due to the transmission of the go-to-sleep command.

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, after the delay of the sleep mode frame has passed. 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, after the delay of the sleep mode frame has passed. When the return code of the function Lin_GetStatus is not LIN_CH_SLEEP, the go-to-sleep command transmission has failed. ⌋()

[LINIF557] ⌈When the go-to-sleep command was sent successful, the LIN Interface shall invoke the function <User>_GotoSleepConfirmation with the parameter TRUE. ⌋()

[LINIF558] ⌈When the go-to-sleep command was not sent successful, the LIN Interface shall invoke the function <User>_GotoSleepConfirmation with the parameter FALSE. ⌋()

[LINIF293] ⌈When entering the CHANNEL_SLEEP state during the go-to-sleep command process, the function LinIf_MainFunction shall switch the current used schedule table switch to the NULL_SCHEDULE. ⌋()

7.3.3 Wake up process

There are different possibilities to wake-up a LIN channel. Either the upper layer requests a wake-up through the LinIf_Wakeup call or a bus wake-up is detected. If a wake-up on the channel is detected, the function LinIf_CheckWakeup will be called by the ECU State Manager.

[LINIF496] ⌈If the referenced channel was in the sleep state and the wake-up command was sent, the LIN Interface shall issue the <User>_WakeupConfirmation with the parameter TRUE. ⌋()

[LINIF670] ⌈If the referenced channel was not in the sleep state and the wake-up command was not sent, the LIN Interface shall issue the <User>_WakeupConfirmation with the parameter TRUE. ⌋()

[LINIF503] ⌈The function LinIf_CheckWakeup shall issue the call of function Lin_CheckWakeup or LinTrcv_CheckWakeup depending on the given parameter WakeupSource. ⌋()

7.3.3.1 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 5). 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 LinIf_CheckWakeup is invoked, the LIN Interface shall maintain the LIN channel state CHANNEL_OPERATIONAL. ⌋(BSW01560)

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. ⌋(BSW01560)

[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. ⌋()

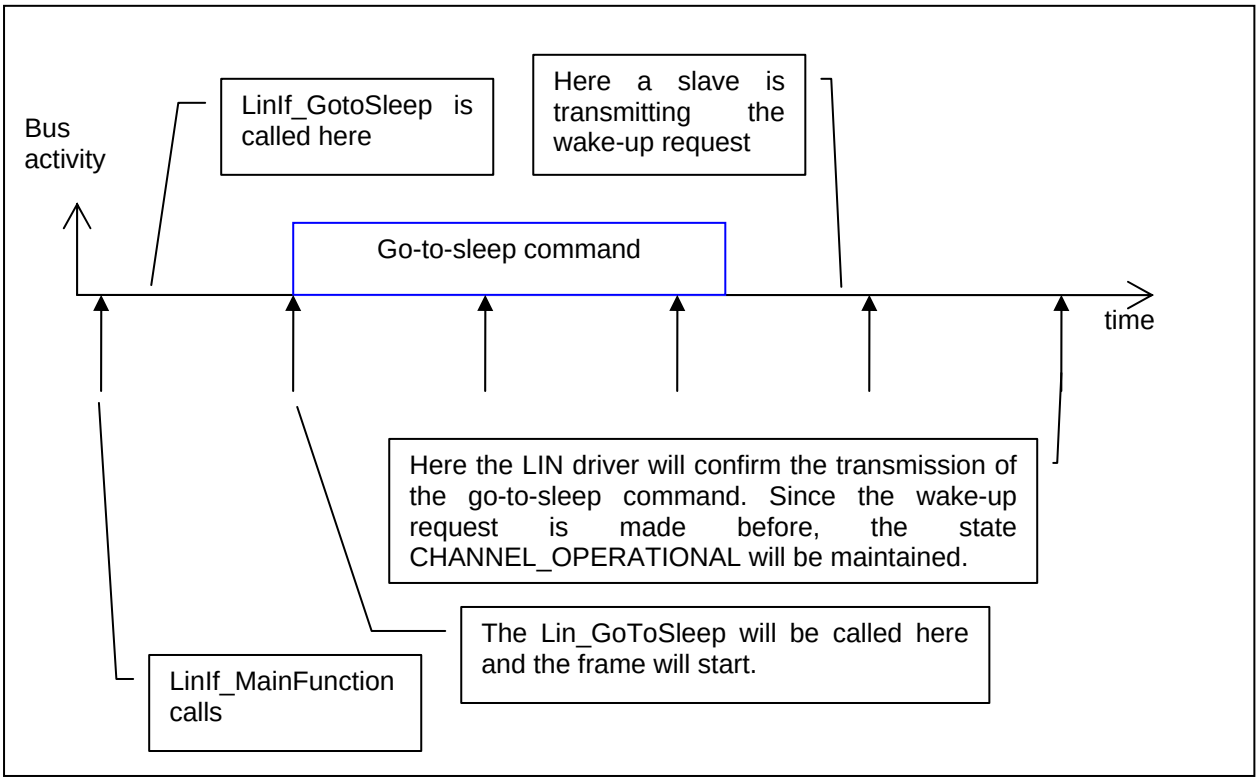

Figure 5 – Wake up requested before confirmation of go-to-sleep command

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 2.1 specification describes. However, the reporting is different.

There is an internal reporting within the own node (by using the API call l_ifc_read_status defined in the LIN 2.1 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 2.1 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 2.1 specification and has nothing to do with the AUTOSAR configuration.

The Diagnostic Transport Layer and the Node Configuration in LIN 2.1 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 2.1 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 2.1 specification specifies two ways for the LIN master to configure slaves:

- By using the LIN 2.1 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 2.1 specification) by using services directly in the Schedule Table.⌋(BSW01590)

7.5.1.1 Node Model

The LIN 2.1 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 2.1 specification).

The Node Model, which is specified in the LIN specification, 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 2.1 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), "Assign Frame ID range" (defined in the LIN 2.1 specification), "Unassign Frame ID" (defined in the LIN 2.0 specification) and "Save Configuration" (defined in the LIN 2.1 specification).⌋()

[LINIF409] ⌈The LIN Interface shall support the FreeFormat (defined in the LIN 2.1 specification).

The response of the FreeFormat is not defined within the LIN 2.1 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 2.1 specification. ⌋()

[LINIF310] ⌈The support for the Node Configuration request “Assign NAD” (defined in the LIN 2.1 specification) and “Conditional change NAD” (defined in the LIN 2.1 specification) shall be pre-compile time configurable On/Off by the configuration parameter LinIfNcOptionalRequestSupported.

The LIN 2.1 specification states that the Data Dump request shall not be used in operational clusters. ⌋(BSW171)

[LINIF408] ⌈The LIN Interface shall not support the Node Configuration request Data Dump (defined in the LIN 2.1 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 identification 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 2.1 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 2.1 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 2.1 specification.

[LINIF479] ⌈The LIN Interface shall process the fixed MRF entries without the interaction with an upper layer. ⌋()

Note: 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 2.1 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 not optional for the node configuration requests according to the LIN 2.1 specification. However, if the SRF header is scheduled after a node configuration request, it is considered that a response is expected. Therefore, the following shall apply:

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

Note that there is no negative answer for node configuration requests defined in the LIN 2.1 specification. Only the function Read-by-Identifier supports a negative answer. As this function is not supported within the LIN Interface, there are no negative responses to process for the LIN Interface.

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

7.5.2 Diagnostics – Transport Protocol

In the LIN 2.1 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 2.1 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. ⌋ (BSW01579)

[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). ⌋(BSW171)

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

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

The LIN TP is used to transport Diagnostic services and responses. No Diagnostic sessions are made in parallel. And service requests are always proceeded in sequence. ⌋(BSW01574)

[LINIF062] ⌈There shall be only one active LIN TP message at one time (i.e. only half-duplex) on one channel. ⌋(BSW01534)

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

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

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

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

[LINIF644] ⌈If the function LinTp_CancelReceive is called and LIN TP is in state LINTP_CHANNEL_BUSY, the LIN Interface shall request a schedule table change to the applicative schedule by calling the function BswM_LinTp_RequestMode with the parameter LINTP_APPLICATIVE_SCHEDULE. ⌋()

[LINIF645] ⌈If the function LinTp_CancelTransmit is called and LIN TP is in state LINTP_CHANNEL_BUSY, the LIN Interface shall request a schedule table change to

the applicative schedule by calling the function BswM_LinTp_RequestMode with the parameter LINTP_APPLICATIVE_SCHEDULE.」()

[LINIF646]「If the configuration parameter LinTpScheduleChangeDiag is TRUE, a schedule table change to the diagnostic or applicative schedule by calling the function BswM_LinTp_RequestMode is done.」()

[LINIF647]「The LIN Interface shall provide an additional configuration parameter LinTpScheduleChangeDiag.」()

7.5.2.1 LIN TP initialization

[LINIF098]「The LIN TP functions except LinTp_Init and LinTp_GetVersionInfo shall only be available when the LIN TP and the corresponding channel has been initialized.」(BSW01545)

7.5.2.2 State-machine

The following Figure 6 shows the state-machine of the LIN TP.

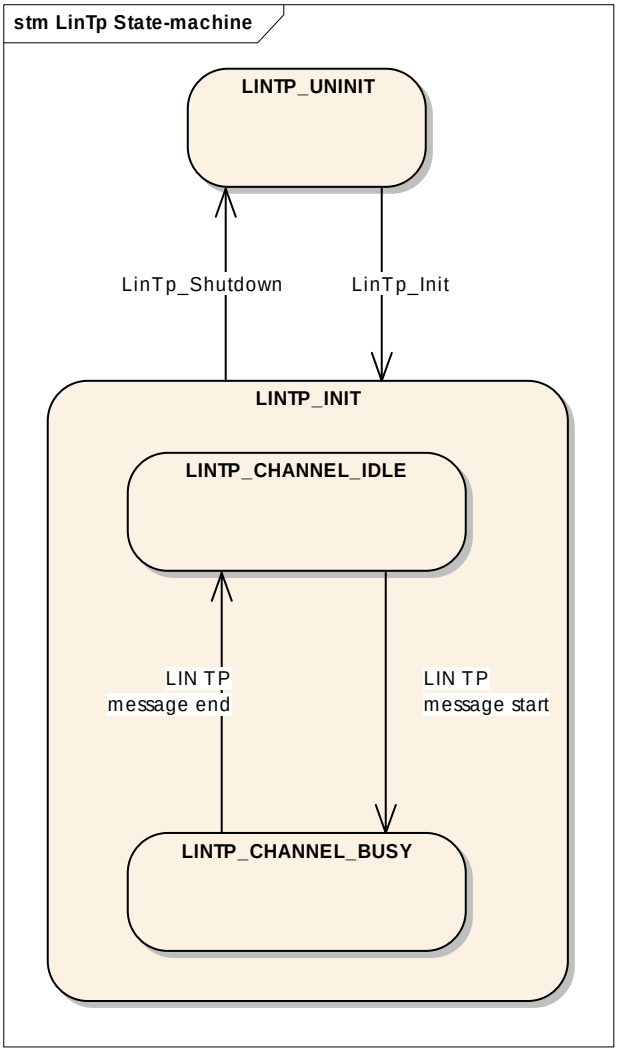

Figure 6 – LIN Transport Protocol state-machine

[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. ⌋ (BSW00335)

[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. ⌋ (BSW00335)

[LINIF412] ⌈Each channel of the LIN TP state LINTP_INIT shall have a sub-state-machine. ⌋ ()

[LINIF450] ⌈The sub-state-machine of the LIN TP state LINTP_INIT shall have the state LINTP_CHANNEL_IDLE.⌋()

[LINIF321] ⌈The LIN Interface shall start only a transmission of a TP message if the channel is in the sub-state LINTP_CHANNEL_IDLE.⌋()

[LINIF414] ⌈The LIN Interface shall set the sub-state of a channel to LINTP_CHANNEL_IDLE when it has successfully terminated the transmission or reception of a LIN TP message.⌋()

[LINIF688] ⌈The LIN Interface shall set the sub-state of a channel to LINTP_CHANNEL_IDLE when it has detected an unrecoverable error on this channel.⌋()

[LINIF322] ⌈The sub-state-machine of the LIN TP state LINTP_INIT shall have the state LINTP_CHANNEL_BUSY.⌋()

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

7.5.2.3 Buffer handling

The LIN Interface requests the data buffer to the upper layer (PDU Router) for the transmission or reception of a TP message.

7.5.2.4 LIN TP transmission

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

[LINIF671] ⌈After a transmission request from the upper layer, the LIN Interface shall call the function PduR_LinTpCopyTxData with the PduInfo pointer containing data buffer (SduDataPtr) and data length (SduLength) for each segment that is sent, e.g. the data length is 5 bytes (including SID) for FF, up to 6 bytes for SF and 6 bytes for CF (or less in case of the last CF). The upper layer copy the transmit data to the PduInfo.⌋()

[LINIF329] ⌈If the function PduR_LinTpCopyTxData returns BUFREQ_E_BUSY, the LIN Interface shall not send the next MRF.⌋()

[LINIF330] ⌈If the function PduR_LinTpCopyTxData returns BUFREQ_E_BUSY, the LIN Interface shall later (implementation specific) retry to request a transmit data. ⌋()

[LINIF672] ⌈When the transmit data is provided, the LIN Interface shall resume the transmission of the MRF. ⌋()

[LINIF068] ⌈When the LIN Interface has transmitted the last MRF (SF or CF) successfully, it shall notify the upper layer by calling the function PduR_LinTpTxConfirmation with the result NTFRSLT_OK. ⌋()

7.5.2.5 LIN TP transmission error

[LINIF069] ⌈If a LIN error on the MRF occurs (the return code of the function Lin_GetStatus is LIN_TX_HEADER_ERROR or LIN_TX_ERROR), the LIN Interface shall abort the transmission and notify the upper layer by calling the function PduR_LinTpTxConfirmation with the result NTFRSLT_E_NOT_OK. ⌋()

[LINIF073] ⌈If the function PduR_LinTpCopyTxData reports BUFREQ_E_NOT_OK, the LIN Interface shall abort the transmission and notify the upper layer by calling the function PduR_LinTpTxConfirmation with the result NTFRSLT_E_NOT_OK. ⌋()

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

[LINIF656] ⌈The LIN Interface shall provide the N_As timeout observation in order to switch a schedule table from diagnostic schedule to applicative schedule in case that transmission for MRF is not successful. ⌋()

[LINIF657] ⌈The LIN Interface shall start the N_As timer after invocation of the function Lin_SendFrame for MRF (FF or CF) and stop after receiving LIN driver status as LIN_TX_OK for MRF by calling the function Lin_GetStatus. ⌋()

[LINIF658] ⌈In case of N_As timeout occurrence the LIN Interface shall abort the transmission and notify the upper layer by calling the function PduR_LinTpTxConfirmation with the result NTFRSLT_E_TIMEOUT_A and request a schedule table change to the applicative schedule by calling the function BswM_LinTp_RequestMode with the parameter LINTP_APPLICATIVE_SCHEDULE (see LINIF646). ⌋()

[LINIF659] ⌈The LIN Interface shall provide an additional configuration parameter LinTpNas as N_As timeout time. ⌋()

[LINIF660] ⌈The LIN Interface shall provide the N_Cs timeout observation (LIN 2.1 specification defines the following requirement: (N_Cs+N_As) < 0.9*N_Cr timeout) in order to switch a schedule table from diagnostic schedule to applicative schedule in case that transmission for MRF is not successful.⌋()

[LINIF661] ⌈The LIN Interface shall start the N_Cs timer after receiving LIN driver status as LIN_TX_OK for MRF (FF or CF except last CF) by calling the function Lin_GetStatus and stop after invocation of the function Lin_SendFrame for MRF (next CF).⌋()

[LINIF662] ⌈In case of N_Cs timeout occurrence the LIN Interface shall abort the transmission and notify the upper layer by calling the function PduR_LinTpTxConfirmation with the result NTFRSLT_E_NOT_OK and request a schedule table change to the applicative schedule by calling the function BswM_LinTp_RequestMode with the parameter LINTP_APPLICATIVE_SCHEDULE (see LINIF646).⌋()

[LINIF663] ⌈The LIN Interface shall provide an additional configuration parameter LinTpNcs as N_Cs timeout time.⌋()

7.5.2.6 LIN TP reception

The LIN Interface shall prepare so 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 LIN Interface shall call the function PduR_LinTpStartOfReception when the start of a SRF is indicated by the reception of a FF or a SF. The output pointer parameter provides the LIN Interface with currently available receive buffer size.⌋()

[LINIF076] ⌈The LIN Interface shall be able to convert the NAD from the transmitting LIN slave to an N-SDU Id that the upper layer understands.⌋()

[LINIF674] ⌈After reception of each SRF (SF, FF and CF), the LIN Interface shall call the function PduR_LinTpCopyRxData with a PduInfo pointer containing data buffer (SduDataPtr) and data length (SduLength), e.g. the data length is 5 bytes (including SID) for FF, up to 6 bytes for SF and 6 bytes for CF (or less in case of the last CF). The output pointer parameter provides the LIN Interface with available receive buffer size after data have been copied.⌋()

[LINIF675] ⌈In case that remaining buffer is too small for the next CF reception, the LIN Interface might call the function PduR_LinTpCopyRxData with a data length 0 (zero) until a buffer of sufficient size is provided. These calls might be performed until the start of the next CF slot.⌋ ()

[LINIF078] ⌈When the LIN Interface has received the last SRF (SF or CF) successfully, it shall notify the upper layer by calling the function PduR_LinTpRxIndication with the result NTFRSLT_OK.⌋ ()

7.5.2.7 LIN TP reception error

If a LIN error occurs while receiving SRF, the LIN Interface checks the timeout of SRF and does not notify a LIN error. If the reception of SRF is successful, the LIN Interface checks the contents of received SRF's.

[LINIF079] ⌈In case that incorrect sequence number is received, the LIN Interface shall stop the current LIN TP message reception.⌋ (BSW01544)

[LINIF651] ⌈In case that unexpected PCI is received (e.g. a SF is received after a CF), the LIN Interface shall stop the current LIN TP message reception.⌋ (BSW01544)

[LINIF652] ⌈In case that invalid data length is received (a SF with a length of 0 (zero) or greater than 6, a FF with a length of less than 7), the LIN Interface shall stop the current LIN TP message reception.⌋ ()

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

[LINIF613] ⌈In case that incorrect NAD is received and the configuration parameter LinTpDropNotRequestedNad is FALSE, the LIN Interface shall stop the current LIN TP message reception.⌋ ()

[LINIF648] ⌈In case that incorrect NAD is received and the configuration parameter LinTpDropNotRequestedNad is TRUE, the LIN Interface shall continue the current LIN TP message reception.⌋ ()

[LINIF614] ⌈The LIN Interface shall request a schedule table change to the applicative schedule by calling the function BswM_LinTp_RequestMode with the parameter LINTP_APPLICATIVE_SCHEDULE when it detects one of the errors that specified in LINIF079, LINIF651, LINIF652 and LINIF613 (see LINIF646).⌋ ()

[LINIF081] ⌈In case that incorrect sequence number is received, the LIN Interface shall report this failure to PDU Router by calling the function PduR_LinTpRxIndication with the result NTFRSLT_E_WRONG_SN. ⌋()

[LINIF653] ⌈In case that unexpected PCI is received, the LIN Interface shall report this failure to PDU Router by calling the function PduR_LinTpRxIndication with the result NTFRSLT_E_UNEXP_PDU. ⌋()

[LINIF654] ⌈In case that invalid data length is received, the LIN Interface shall report this failure to PDU Router by calling the function PduR_LinTpRxIndication with the result NTFRSLT_E_UNEXP_PDU. ⌋()

[LINIF655] ⌈In case that incorrect NAD is received and the configuration parameter LinTpDropNotRequestedNad is FALSE, the LIN Interface shall report this failure to PDU Router by calling the function PduR_LinTpRxIndication with the result NTFRSLT_E_UNEXP_PDU. ⌋()

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

In the situation where the LIN Interface (master) has encountered a permanent error (either by upper layer signaling permanent error or the bus indicated an erroneous frame) the slave continues to transmit the rest of the frames when the master transmits a SRF header. The slave cannot know when the master has encountered a problem. The slave continues to transmit responses to the SRF headers. This means that no error-recovery is supported. ⌋()

[LINIF664] ⌈The LIN Interface shall provide the N_Cr timeout observation in order to switch a schedule table from diagnostic schedule to applicative schedule in case that reception for SRF is not successful. ⌋()

[LINIF665] ⌈The LIN Interface shall start the N_Cr timer after receiving LIN driver status as LIN_RX_OK for SRF (FF or CF except last CF) by calling the function Lin_GetStatus and stop after receiving LIN driver status as LIN_RX_OK for SRF (next CF) by calling the function Lin_GetStatus. ⌋()

[LINIF666] ⌈In case of N_Cr timeout occurrence the LIN Interface shall abort the reception and notify the upper layer by calling the function PduR_LinTpRxIndication with the result NTFRSLT_E_TIMEOUT_CR and request a schedule table change to the applicative schedule by calling the function BswM_LinTp_RequestMode with the parameter LINTP_APPLICATIVE_SCHEDULE (see LINIF646). ⌋()

[LINIF667] ⌈The LIN Interface shall provide an additional configuration parameter LinTpNcr as N_Cr timeout time.⌋()

[LINIF617] ⌈The LIN Interface shall provide the P2 timeout observation in order to switch a schedule table from diagnostic schedule to applicative schedule in case that reception for SRF is not successful.⌋()

[LINIF618] ⌈The LIN Interface shall start the P2 timer after invocation of the function Lin_SendFrame for last MRF (SF or CF) and stop after receiving LIN driver status as LIN_RX_OK for SRF (SF or FF) by calling the function Lin_GetStatus. Note that the P2 timeout monitoring shall be started only in LIN TP diagnostic mode.⌋()

[LINIF619] ⌈In case of P2 timeout occurrence the LIN Interface shall abort the reception and notify the upper layer by calling the function PduR_LinTpRxIndication with the result NTFRSLT_E_NOT_OK and request a schedule table change to the applicative schedule by calling the function BswM_LinTp_RequestMode with the parameter LINTP_APPLICATIVE_SCHEDULE (see LINIF646).⌋()

[LINIF620] ⌈The LIN Interface shall provide an additional configuration parameter LinTpP2Timing as P2 timeout time.⌋()

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

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

[LINIF622] ⌈The LIN Interface shall provide an additional configuration parameter LinTpP2Max as P2max timeout time.⌋()

[LINIF623] ⌈If more UDS response pending frames have been received than allowed (configuration parameter LinTpMaxNumberOfRespPendingFrames), the LIN Interface shall abort the reception and notify the upper layer by calling the function PduR_LinTpRxIndication with the result NTFRSLT_E_NOT_OK and request a schedule table change to the applicative schedule by calling the function BswM_LinTp_RequestMode with the parameter LINTP_APPLICATIVE_SCHEDULE (see LINIF646).⌋()

[LINIF624] ⌈The LIN Interface shall provide a configuration parameter LinTpMaxNumberOfRespPendingFrames.⌋()

7.5.2.8 Unavailability of receive buffer

The function PduR_LinTpStartOfReception and PduR_LinTpCopyRxData 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.

[LINIF676] ⌈If the function PduR_LinTpStartOfReception returns BUFREQ_E_NOT_OK or BUFREQ_E_OVFL, the LIN Interface shall abort the reception.⌋()

[LINIF677] ⌈If the function PduR_LinTpCopyRxData returns BUFREQ_E_NOT_OK, the LIN Interface shall abort the reception.⌋()

[LINIF087] ⌈When the LIN Interface has aborted the reception, it shall notify the upper layer by calling the function PduR_LinTpRxIndication with the result NTFRSLT E_NOT_OK.⌋()

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

[LINIF679] ⌈If the function PduR_LinTpStartOfReception returns BUFREQ_E_BUSY, the LIN Interface shall suspend sending LIN headers for next SRF.⌋()

[LINIF085] ⌈If the function PduR_LinTpCopyRxData returns BUFREQ_E_BUSY, the LIN Interface shall suspend sending LIN headers for next SRF.⌋()

[LINIF086] ⌈In case of BUFREQ_E_BUSY, the LIN Interface shall call the function PduR_LinTpCopyRxData to try to copy the received data again during the next processing of the MainFunction.⌋()

[LINIF680] ⌈When the receive buffer is provided, the LIN Interface shall resume the transmission of LIN headers for SRF.⌋()

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 and link 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. ⌋()

7.6.3 Multiple LIN transceiver drivers

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

[LINIF560] ⌈The allocation of each channel to a LIN transceiver driver shall be pre-compile time configurable by the configuration parameter LinIfMultipleTrcvDriverSupported.

The LIN transceiver 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. ⌋(BSW00409)

[LINIF267] ⌈Development error values are of type uint8. ⌋()

[LINIF376] ⌈The following Table 1 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 made in channel sleep	Development	LINIF_E_SCHEDULE_REQUEST_ERROR	0x51
Referenced transceiver channel does not exist (identification is out of range)	Development	LINIF_E_TRCV_INV_CHANNEL	0x52
API service called with invalid parameter for LIN transceiver operation mode	Development	LINIF_E_TRCV_INV_MODE	0x53
Referenced transceiver state is not normal	Development	LINIF_E_TRCV_NOT_NORMAL	0x54
API service called with invalid parameter for WakeupSource	Development	LINIF_E_PARAM_WAKEUPSOURCE	0x55
LIN frame error detected	Development	LINIF_E_RESPONSE	0x60
Slave did not answer on a node configuration request	Development	LINIF_E_NC_NO_RESPONSE	0x61

Table 1 – Error codes for DET and DEM

⌋(BSW00406, BSW00337, BSW00338, BSW00339, BSW00385, BSW00327)

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.⌋ (BSW00386, BSW00350)

[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.⌋ (BSW00323)

[LINIF535] ⌈All LIN Interface API services other than LinIf_Init, LinIf_GetVersionInfo and LinTp_GetVersionInfo shall:

- not execute their normal operation,
- report to the development error tracer (using LINIF_E_UNINIT), if the LinIfDevErrorDetect switch is enabled,
- and return E_NOT_OK, if the API has a return value.

Unless the LIN Interface has been initialized with a preceding call of LinIf_Init.⌋ (BSW00406)

[LINIF687] ⌈All LIN TP API services other than LinTp_Init and LinTp_GetVersionInfo shall:

- not execute their normal operation,
- report to the development error tracer (using LINIF_E_UNINIT), if the LinIfDevErrorDetect switch is enabled,
- and return E_NOT_OK, if the API has a return value.

Unless the LIN Interface has been initialized with a preceding call of LinTp_Init.⌋ (BSW00406)

[LINIF270] ⌈The detection of production code errors can not be switched off.⌋ ()

7.9 Error notification

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

[LINIF272] ⌈Production errors shall be reported to Diagnostic Event Manager (DEM).⌋ ()

[LINIF579] ⌈Additional errors that are detected because of specific implementation shall be added in the LIN Interface module implementation specification. The classification and enumeration shall be compatible to the errors listed above LINIF376.⌋ ()

7.10 Debugging

[LINIF515] ⌈Each variable that shall be debugged or traced by AUTOSAR shall be defined as global variable. ⌋(BSW33200022)

[LINIF516] ⌈All type definitions of variables which shall be debugged shall be accessible by the header file LinIf.h. ⌋(BSW33200002)

[LINIF517] ⌈It shall be possible to calculate the size of elements by c-“sizeof” operation. ⌋(BSW33200022)

[LINIF518] ⌈Optionally it shall be possible to decode the structure elements. ⌋(BSW33200022)

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]⌈

Module	*Imported Type*
ComStack_Types	BufReq_ReturnType
	NetworkHandleType
	NotifResultType
	PduIdType
	PduInfoType
	PduLengthType
	RetryInfoType
	TPParameterType
Dem	Dem_EventIdType
	Dem_EventStatusType
EcuM	EcuM_WakeupSourceType
Lin	Lin_PduType
	Lin_StatusType
LinTrcv	LinTrcv_TrcvModeType
Lin_GeneralTypes	LinTrcv_TrcvWakeupModeType
	LinTrcv_TrcvWakeupReasonType
Std_Types	Std_ReturnType
	Std_VersionInfoType

⌋(BSW00413)

8.1.2 Type definitions

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

8.1.2.1 LinIf_SchHandleType

[LINIF197]⌈

Name:	`LinIf_SchHandleType`		
Type:	`uint8`		
Range:	`range`	0..255	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. The NULL_SCHEDULE for each channel has index 0.
	`NULL_SCHEDULE`	0x00	The NULL_SCHEDULE.
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

⌋ (BSW00413)

8.1.2.2 LinIf_ConfigType

[LINIF668] ⌈

Name:	`LinIf_ConfigType`	
Type:	`Structure`	
Range:	`implementation specific`	--
Description:	A pointer to an instance of this structure will be used in the initialization of the LIN Interface. The outline of the structure is defined in chapter 10 Configuration Specification.	

⌋ ()

8.1.2.3 LinTp_ConfigType

[LINIF426] ⌈

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

⌋ ()

8.1.2.4 LinTp_Mode

[LINIF629] ⌈

Name:	`LinTp_Mode`	
Type:	`Enumeration`	
Range:	`LINTP_APPLICATIVE_SCHEDULE`	Applicative schedule is selected
	`LINTP_DIAG_REQUEST`	Master request schedule table is selected
	`LINTP_DIAG_RESPONSE`	Slave response schedule table is selected
Description:	This type denotes which Schedule table can be requested by LIN TP during diagnostic session	

⌋ ()

8.2 LIN Interface API

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

8.2.1 LinIf_Init

[LINIF198] ⌈

Service name:	LinIf_Init	

Syntax:	`void LinIf_Init(`
	`  const LinIf_ConfigType* 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.	

⌋(BSW101, BSW00416, BSW00358, BSW01569, BSW00414)

[LINIF371] ⌈The parameter ConfigPtr of the function LinIf_Init shall be only relevant for the configuration variant VARIANT-POST-BUILD. The parameter ConfigPtr shall be ignored for the configuration variant VARIANT-PRE-COMPILE and the configuration variant VARIANT-LINK-TIME but will be still there for compatible reasons.⌋(BSW00344, BSW00404, BSW00405, BSW01570)

[LINIF486] ⌈If development error detection is enabled and the parameter ConfigPtr has an invalid value, the function LinIf_Init shall raise the development error code LINIF_E_PARAMETER_POINTER.⌋()

[LINIF373] ⌈The function LinIf_Init shall accept a parameter that references to a LIN Interface configuration descriptor.⌋(BSW00344, BSW00404, BSW00405, BSW170)

[LINIF233] ⌈The function LinIf_Init shall set the schedule type NULL_SCHEDULE for each configured channel.⌋()

[LINIF562] ⌈If development error detection is enabled and the LIN Interface is already initialized the function LinIf_Init shall raise the development error code LINIF_E_ALREADY_INITIALIZED.⌋()

8.2.2 LinIf_GetVersionInfo

[LINIF340] ⌈

Service name:	LinIf_GetVersionInfo

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

Service ID[hex]:	0x03
Sync/Async:	Synchronous
Reentrancy:	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.

⌋(BSW00407)

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

- Two bytes for the vendor ID
- Two bytes 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. ⌋()

[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. ⌋()

[LINIF640] ⌈If development error detection is enabled and the parameter versioninfo has an invalid value, the function LinIf_GetVersionInfo shall raise the development error code LINIF_E_PARAMETER_POINTER. ⌋()

[LINIF279] ⌈The function LinIf_GetVersionInfo shall be pre-compile time configurable On/Off by the configuration parameter LinIfVersionInfoApi. ⌋(BSW00411)

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)
Description:	Indicates a request.	

⌋(BSW01571)

[LINIF105] ⌈The function LinIf_Transmit shall indicate a request from an upper layer to transmit a 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 container LinIfFrame) that the LIN Interface recognizes.⌋()

[LINIF570] ⌈If development error detection is enabled and the parameter PduInfoPtr has an invalid value, the function LinIf_Transmit shall raise the development error code LINIF_E_PARAMETER_POINTER.⌋()

[LINIF575] ⌈If development error detection is enabled and the parameter LinTxPduId has an invalid value, the function LinIf_Transmit shall raise the development error code LINIF_E_PARAMETER.⌋()

Caveats: The LIN Interface has to be initialized with a call of LinIf_Init before this API service may be called, see LINIF535.

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) - State is sleep
Description:	Requests a schedule table to be executed.	

」(BSW01564)

[LINIF506] 「Schedule tables are configured by the LinIfScheduleTable container in the LIN Interface configuration. 」()

[LINIF389] 「The function LinIf_ScheduleRequest shall request the schedule table manager to be executed.

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

[LINIF563] 「If development error detection is enabled and an invalid channel is given, the function LinIf_ScheduleRequest shall raise the development error code LINIF_E_NONEXISTENT_CHANNEL. 」()

[LINIF567] 「If development error detection is enabled and an invalid schedule table is given or the corresponding channel is in the state CHANNEL_SLEEP, the function LinIf_ScheduleRequest shall raise the development error code LINIF_E_SCHEDULE_REQUEST_ERROR. 」()

Caveats: The LIN Interface has to be initialized with a call of LinIf_Init before this API service may be called, see LINIF535.

8.2.5 LinIf_GotoSleep

[LINIF204] 「

Service name:	LinIf_GotoSleep
Syntax:	Std_ReturnType LinIf_GotoSleep(

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

」(BSW01523)

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

[LINIF564] 「If development error detection is enabled and an invalid channel is given, the function LinIf_GotoSleep shall raise the development error code LINIF_E_NONEXISTENT_CHANNEL. 」()

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

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

Caveats: The LIN Interface has to be initialized with a call of LinIf_Init before this API service may be called, see LINIF535.

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.	

」(BSW01515)

[LINIF432] ⌈When the referenced channel is not in the sleep state, the function LinIf_Wakeup will not forward the call to the LIN driver. In this case, it will simulate a successful wakeup by returning E_OK. ⌋()

[LINIF296] ⌈The function LinIf_Wakeup shall call the function Lin_Wakeup of the LIN Driver module to transmit a wake-up request on the selected channel if the channel is in the channel state CHANNEL_SLEEP. ⌋()

[LINIF565] ⌈If development error detection is enabled and an invalid channel is given, the function LinIf_Wakeup shall raise the development error code LINIF_E_NONEXISTENT_CHANNEL. ⌋()

Caveats: The LIN Interface has to be initialized with a call of LinIf_Init before this API service may be called, see LINIF535.

8.2.7 LinIf_SetTrcvMode

[LINIF544] ⌈

Service name:	LinIf_SetTrcvMode	
Syntax:	`Std_ReturnType LinIf_SetTrcvMode( NetworkHandleType Channel, LinTrcv_TrcvModeType TransceiverMode )`	
Service ID[hex]:	0x08	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Channel	Identification of the LIN channel
	TransceiverMode	Requested mode transition
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Will be returned, if the transceiver state has been changed to the requested mode. E_NOT_OK: Will be returned, if the transceiver state change has failed or the parameter is out of the allowed range. The previous state has not been changed.
Description:	Set the given LIN transceiver to the given mode.	

」(BSW01584, BSW01585, BSW01586)

[LINIF536] ⌈This service shall call the underlying function LinTrcv_SetOpMode(LinNetwork, OpMode) for the corresponding requested LIN transceiver.⌋()

[LINIF537] ⌈This API shall be applicable to all LIN transceivers with all values independent if the transceiver hardware supports these modes or not.⌋()

[LINIF538] ⌈The API LinIf_SetTrcvMode returns the value that is returned by LinTrcv_SetOpMode.⌋()

[LINIF539] ⌈If development error detection is enabled and an invalid value for Channel is given, the function LinIf_SetTrcvMode shall report LINIF_E_TRCV_INV_CHANNEL to the development error tracer.⌋()

[LINIF540] ⌈If development error detection is enabled and an invalid mode is requested for TransceiverMode, the function LinIf_SetTrcvMode shall report LINIF_E_TRCV_INV_MODE to the development error tracer.⌋()

[LINIF568] ⌈If development error detection is enabled and the requested mode is LINTRCV_TRCV_MODE_SLEEP and the current mode is not LINTRCV_TRCV_MODE_NORMAL, the function LinIf_SetTrcvMode shall report LINIF_E_TRCV_NOT_NORMAL to the development error tracer.⌋()

[LINIF634] ⌈The function LinIf_SetTrcvMode is required only if at least one LIN channel uses the LIN transceiver driver. This function shall be pre-compile time configurable On/Off by the configuration parameter LinIfTrcvDriverSupported.⌋()

Caveats: The LIN Interface has to be initialized with a call of LinIf_Init before this API service may be called, see LINIF535.

8.2.8 LinIf_GetTrcvMode

[LINIF545] ⌈

Service name:	LinIf_GetTrcvMode	
Syntax:	`Std_ReturnType LinIf_GetTrcvMode( NetworkHandleType Channel, LinTrcv_TrcvModeType* TransceiverModePtr )`	
Service ID[hex]:	0x09	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Channel	Identification of the LIN channel
Parameters (inout):	None	
Parameters (out):	TransceiverModePtr	Pointer to a memory location where output value will be

		stored.
Return value:	Std_ReturnType	E_OK: The call of the LINTransceiver Driver's API service has returned E_OK E_NOT_OK: The call of the LIN Transceiver Driver's API service has returned E_NOT_OK
Description:	Returns the actual state of a LIN Transceiver Driver.	

」(BSW01587)

[LINIF541] 「This service shall invoke the underlying function LinTrcv_GetOpMode(LinNetwork, OpMode) for the corresponding requested LIN transceiver. 」()

[LINIF546] 「If development error detection is enabled and an invalid value for Channel is given, the function LinIf_GetTrcvMode shall report LINIF_E_TRCV_INV_CHANNEL to the development error tracer. 」()

[LINIF571] 「If development error detection is enabled and the parameter TransceiverModePtr has an invalid value, the function LinIf_GetTrcvMode shall raise the development error code LINIF_E_PARAMETER_POINTER. 」()

[LINIF635] 「The function LinIf_GetTrcvMode is required only if at least one LIN channel uses the LIN transceiver driver. This function shall be pre-compile time configurable On/Off by the configuration parameter LinIfTrcvDriverSupported. 」()

Caveats: The LIN Interface has to be initialized with a call of LinIf_Init before this API service may be called, see LINIF535.

8.2.9 LinIf_GetTrcvWakeupReason

[LINIF547] 「

Service name:	LinIf_GetTrcvWakeupReason	
Syntax:	`Std_ReturnType LinIf_GetTrcvWakeupReason( NetworkHandleType Channel, LinTrcv_TrcvWakeupReasonType* TrcvWuReasonPtr )`	
Service ID[hex]:	0x0a	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Channel	Identification of the LIN channel
Parameters (inout):	None	
Parameters (out):	TrcvWuReasonPtr	Pointer to a memory location where output vale will be stored.
Return value:	Std_ReturnType	E_OK: The call of the LIN Transceiver Driver's API service has returned E_OK. E_NOT_OK: The call of the LIN Transceiver Driver's API service has returned E_NOT_OK.
Description:	Returns the reason for the wake up that has been detected by the LIN Transceiver Driver.	

⌋(BSW01588)

[LINIF548] ⌈This API shall return the reason for the wake up that the LIN transceiver has detected by invoking the underlying function LinTrcv_GetBusWuReason(LinNetwork, Reason) for the corresponding requested LIN transceiver. ⌋()

[LINIF549] ⌈If development error detection is enabled and an invalid value for Channel is given, the function LinIf_GetTrcvWakeupReason shall report LINIF_E_TRCV_INV_CHANNEL to the development error tracer. ⌋()

[LINIF573] ⌈If development error detection is enabled and the parameter TrcvWuReasonPtr has an invalid value, the function LinIf_GetTrcvWakeupReason shall raise the development error code LINIF_E_PARAMETER_POINTER. ⌋()

[LINIF572] ⌈If development error detection is enabled and the current mode is not LINTRCV_TRCV_MODE_NORMAL, the function LinIf_GetTrcvWakeupReason shall report LINIF_E_TRCV_NOT_NORMAL to the development error tracer. ⌋()

[LINIF636] ⌈The function LinIf_GetTrcvWakeupReason is required only if at least one LIN channel uses the LIN transceiver driver. This function shall be pre-compile time configurable On/Off by the configuration parameter LinIfTrcvDriverSupported. ⌋()

Caveats:

- The LIN Interface has to be initialized with a call of LinIf_Init before this API service may be called, see LINIF535.
- Please be aware, that if more than one network is available, each network may report a different wake up reason. This API has a “per network” view and does not vote the more important reason or sequence internally. The same may be true if e.g. one transceiver controls the power supply and the other is just powered or un-powered. Then one may be able to return LINTRCV_TRCV_WU_POWER_ON, whereas the other may state e.g. LINTRCV_TRCV_WU_RESET.
  It is up to the EcuM to decide how to handle that wake up information.

8.2.10 LinIf_SetTrcvWakeupMode

[LINIF550] ⌈

Service name:	LinIf_SetTrcvWakeupMode

Syntax:	`Std_ReturnType LinIf_SetTrcvWakeupMode(`
	`  NetworkHandleType Channel,`
	`  LinTrcv_TrcvWakeupModeType LinTrcvWakeupMode`
	`)`

Service ID[hex]:	0x0b

Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Channel	Identification of the LIN channel
	LinTrcvWakeupMode	Requested transceiver wake up reason.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: The call of the LIN Transceiver Driver's API service has returned E_OK. E_NOT_OK: The call of the LIN Transceiver Driver's API service has returned E_NOT_OK.
Description:	This API enables, disables and clears the notification for wakeup events on the addressed network	

⌋(BSW01589)

[LINIF551] ⌈This API shall enable, disable or clear the notification for wake up events on the addressed network by calling the underlying function LinTrcv_SetWakeupMode(LinNetwork, TrcvWakeupMode). ⌋()

[LINIF552] ⌈

Enabled: If the LIN Transceiver Driver has stored a wake up event pending for the addressed network, the notification shall be executed within the API call or immediately after (depending on the implementation).

Disabled: If a wake up event is pending on the addressed network, the notification must not be executed unless the wake up notification is enabled again.

Clear: Clearing of wake up events shall be used, when the wake up notification is disabled to clear all stored wake up events under control of the higher layer. ⌋()

[LINIF595] ⌈If development error detection is enabled and an invalid value for Channel is given, the function LinIf_SetTrcvWakeupMode shall report LINIF_E_TRCV_INV_CHANNEL to the development error tracer. ⌋()

[LINIF596] ⌈If development error detection is enabled and an invalid value for LinTrcvWakeupMode is given, the function LinIf_SetTrcvWakeupMode shall report LINIF_E_PARAMETER to the development error tracer. ⌋()

[LINIF637] ⌈The function LinIf_SetTrcvWakeupMode is required only if at least one LIN channel uses the LIN transceiver driver. This function shall be pre-compile time configurable On/Off by the configuration parameter LinIfTrcvDriverSupported. ⌋()

Caveats:

- The LIN Interface has to be initialized with a call of LinIf_Init before this API service may be called, see LINIF535.

- The implementation may be e.g. disabling the interrupt source for the wake up. If the interrupt is level triggered a pending interrupt is automatically stored and raised after enabling the notification again. It is very important not to lose wake up events during the disabled period.

8.2.11 LinIf_CancelTransmit

[LINIF580] ⌈

Service name:	LinIf_CancelTransmit	
Syntax:	`Std_ReturnType LinIf_CancelTransmit( PduIdType LinTxPduId )`	
Service ID[hex]:	0x0c	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	LinTxPduId	Upper layer identification of the LIN frame for which a cancellation should be done.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: CancelTransmit request has been accepted.
Description:	This is a dummy method introduced for interface compatibility.	

⌋ ()

[LINIF649] ⌈The cancellation request shall always be rejected by returning E_OK. ⌋ ()

[LINIF581] ⌈The function LinIf_CancelTransmit shall be pre-compile time configurable On/Off by the configuration parameter LinIfCancelTransmitSupported. ⌋ ()

[LINIF594] ⌈If development error detection is enabled and an invalid value for LinTxPduId is given, the function LinIf_CancelTransmit shall report LINIF_E_PARAMETER to the development error tracer. ⌋ ()

8.2.12 LinTp_Init

[LINIF350] ⌈

Service name:	LinTp_Init	
Syntax:	`void LinTp_Init( const LinTp_ConfigType* 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.

⌋(BSW101, BSW00414, BSW00416, BSW00358, BSW01540)

[LINIF427] ⌈The parameter ConfigPtr of the function LinTp_Init shall be only relevant for the configuration variant VARIANT-POST-BUILD. The parameter ConfigPtr shall be ignored for the configuration variant VARIANT-PRE-COMPILE and the configuration variant VARIANT-LINK-TIME but will be still there for compatible reasons.⌋()

[LINIF410] ⌈The LIN Interface's environment shall call the function LinTp_Init before using any other LIN TP function.⌋()

[LINIF320] ⌈The function LinTp_Init shall set the sub-state LINTP_CHANNEL_IDLE for each configured channel of the LIN TP channel state-machine.⌋()

[LINIF569] ⌈If development error detection is enabled and the parameter ConfigPtr has an invalid value, the function LinTp_Init shall raise the development error code [LINIF_E_PARAMETER_POINTER.⌋()

[LINIF593] ⌈If development error detection is enabled and the LIN Interface is already initialized the function LinTp_Init shall raise the development error code LINIF_E_ALREADY_INITIALIZED.⌋()

[LINIF681] ⌈The function LinTp_Init shall be pre-compile time configurable On/Off by the configuration parameter LinIfTpSupported.⌋()

Caveats: The LIN Interface has to be initialized with a call of LinIf_Init before this API service may be called, see LINIF535.

8.2.13 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. N-SDU data has not been copied)
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.⌋()

[LINIF584] ⌈The function LinTp_Transmit shall accept a functional transmission request also when a TP message is currently ongoing on the selected channel.⌋()

[LINIF616] ⌈If the transmission for a further physical request is triggered while transmission of a previously triggered physical request is ongoing, the LIN Interface shall accept the new physical request and drop the old physical request.⌋()

[LINIF586] ⌈According to the LIN 2.1 specification, the NAD 0x7E shall be used for a functional transmission request.⌋()

[LINIF327] ⌈The function LinTp_Transmit shall request the PDU Router for a new buffer using the function PduR_LinTpCopyTxData.⌋()

[LINIF574] ⌈If development error detection is enabled and the parameter LinTpTxInfoPtr has an invalid value, the function LinTp_Transmit shall raise the development error code LINIF_E_PARAMETER_POINTER.⌋()

[LINIF576] ⌈If development error detection is enabled and the parameter LinTpTxSduId has an invalid value, the function LinIf_Transmit shall raise the development error code LINIF_E_PARAMETER. ⌋()

[LINIF682] ⌈The function LinTp_Transmit shall be pre-compile time configurable On/Off by the configuration parameter LinIfTpSupported. ⌋()

Caveats: The LIN Interface has to be initialized with a call of LinIf_Init and LinTp_Init before this API service may be called, see LINIF535 and LINIF687.

8.2.14 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.

⌋(BSW00407)

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

- Two bytes for the vendor ID
- Two bytes 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. ⌋()

[LINIF639] ⌈If development error detection is enabled and the parameter versioninfo has an invalid value, the function LinTp_GetVersionInfo shall raise the development error LINIF_E_PARAMETER_POINTER. ⌋()

[LINIF354] ⌈The function LinTp_GetVersionInfo shall be pre-compile time configurable On/Off by the configuration parameter LinTpVersionInfoApi. ⌋(BSW00411)

8.2.15 LinTp_Shutdown

[LINIF355]⌈

Service name:	LinTp_Shutdown

Syntax:	`void LinTp_Shutdown(`
	`  void`
	`)`

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

⌋(BSW00336)

[LINIF356]⌈The function LinTp_Shutdown shall close all pending transport protocol connection of the LIN TP and free all resources of the LIN TP.⌋()

[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.⌋()

[LINIF683]⌈The function LinTp_Shutdown shall be pre-compile time configurable On/Off by the configuration parameter LinIfTpSupported.⌋()

Caveats: The LIN Interface has to be initialized with a call of LinIf_Init and LinTp_Init before this API service may be called, see LINIF535 and LINIF687.

8.2.16 LinTp_CancelTransmit

[LINIF500]⌈

Service name:	LinTp_CancelTransmit

Syntax:	`Std_ReturnType LinTp_CancelTransmit(`
	`  PduIdType LinTpTxSduId`
	`)`

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

」()

[LINIF490] 「The cancellation request shall always be rejected by returning E_NOT_OK.」()

[LINIF577] 「If development error detection is enabled and the parameter LinTpTxSduId has an invalid value, the function LinTp_CancelTransmit shall raise the development error LINIF_E_PARAMETER.」()

[LINIF684] 「The function LinTp_CancelTransmit shall be pre-compile time configurable On/Off by the configuration parameter LinIfTpSupported.」()

Caveats: The LIN Interface has to be initialized with a call of LinIf_Init and LinTp_Init before this API service may be called, see LINIF535 and LINIF687.

8.2.17 LinTp_ChangeParameter

[LINIF501] 「

Service name:	LinTp_ChangeParameter	
Syntax:	`Std_ReturnType LinTp_ChangeParameter( PduIdType id, TPParameterType parameter, uint16 value )`	
Service ID[hex]:	0x44	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	id	Identifier of the received N-SDU on which the reception parameter has to be changed.
	parameter	The selected parameter that the request shall change (STmin).
	value	The new value of the parameter.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_NOT_OK: request is not accepted.
Description:	This service is used to request the change of reception parameter STmin for a specified N-SDU.	

⌋()

[LINIF592] ⌈The change parameter request shall always be rejected by returning E_NOT_OK.⌋()

[LINIF578] ⌈If development error detection is enabled and the parameter id has an invalid value, the function LinTp_ChangeParameter shall raise the development error code LINIF_E_PARAMETER.⌋()

[LINIF685] ⌈The function LinTp_ChangeParameter shall be pre-compile time configurable On/Off by the configuration parameter LinIfTpSupported.⌋()

Caveats: The LIN Interface has to be initialized with a call of LinIf_Init and LinTp_Init before this API service may be called, see LINIF535 and LINIF687.

8.2.18 LinIf_CheckWakeup

[LINIF378] ⌈

Service name:	LinIf_CheckWakeup	
Syntax:	`Std_ReturnType LinIf_CheckWakeup( EcuM_WakeupSourceType WakeupSource )`	
Service ID[hex]:	0x60	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	WakeupSource	Source device, which initiated the wakeup event: LIN controller or LIN transceiver
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: No error has occurred during execution of the API E_NOT_OK: An error has occurred during execution of the API
Description:	Will be called when the EcuM has been notified about a wakeup on a specific LIN channel.	

The LIN Interface will recognize the caller by the parameter of the function LinIf_CheckWakeup.⌋(BSW01514, BSW00375)

[LINIF566] ⌈If development error detection is enabled and the parameter WakeupSource has an invalid value, the function LinIf_CheckWakeup shall raise the development error code LINIF_E_PARAM_WAKEUPSOURCE and return E_NOT_OK.⌋()

[LINIF689] ⌈The function LinIf_CheckWakeup is only available if wake-up is supported by the LIN transceiver driver or by at least one LIN driver channel. This depends on the configuration parameters LinChannelWakeupSupport and

LinTrcvWakeUpSupport, which depends on the used LIN controller / transceiver type and the used wake up strategy.

The function LinIf_CheckWakeup may be called in an interrupt or polling mode. ⌋()

8.2.19 LinTp_CancelReceive

[LINIF625] ⌈

Service name:	LinTp_CancelReceive	
Syntax:	`Std_ReturnType LinTp_CancelReceive( PduIdType LinTpRxSduId )`	
Service ID[hex]:	0x47	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	LinTpRxSduId	This parameter contains the Lin TP instance unique identifier of the Lin N-SDU reception of which has to be cancelled.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_NOT_OK: Cancellation request of the reception of the specified Lin N-SDU is rejected
Description:	This is a dummy method introduced for interface compatibility.	

⌋()

[LINIF626] ⌈The cancellation request shall always be rejected by returning E_NOT_OK. ⌋()

[LINIF627] ⌈If development error detection is enabled and the parameter LinTpRxSduId has an invalid value, the function LinTp_CancelReceive shall raise the development error code LINIF_E_PARAMETER. ⌋()

[LINIF686] ⌈The function LinTp_CancelReceive shall be pre-compile time configurable On/Off by the configuration parameter LinIfTpSupported. ⌋()

Caveats: The LIN Interface has to be initialized with a call of LinIf_Init and LinTp_Init before this API service may be called, see LINIF535 and LINIF687.

8.3 Call-back notifications

The LIN Interface does not contain any callback notifications.

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(`
	`  void`
	`)`

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. ⌋ (BSW00373, BSW00376, BSW01546, BSW01561, BSW01555)

[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*
BswM_LinTp_RequestMode	Function called by LinTP to request a mode for the corresponding LIN channel. The LinTp_Mode mainly correlates to the LIN schedule table that should be used.
Dem_ReportErrorStatus	Queues the reported events from the BSW modules (API is only used by BSW modules). The interface has an asynchronous behavior, because the processing of the event is done within the Dem main function.
Lin_GetStatus	Gets the status of the LIN driver.
Lin_GoToSleep	The service instructs the driver to transmit a go-to-sleep-command on the addressed LIN channel.
Lin_GoToSleepInternal	Sets the channel state to LIN_CH_SLEEP, enables the wake-up detection and optionally sets the LIN hardware unit.
Lin_SendFrame	Sends a LIN header and a LIN response, if necessary. The direction of the frame response (master response, slave response, slave-to-slave communication) is provided by the PduInfoPtr.
Lin_Wakeup	Generates a wake up pulse.

⌋ ()

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.
LinTrcv_CheckWakeup	Notifies the calling function if a wakeup is detected.
LinTrcv_GetBusWuReason	This API provides the reason for the wakeup that the LIN transceiver has detected in the parameter "Reason". The ability to detect and differentiate the possible wakeup reasons depends strongly on the LIN transceiver hardware.
LinTrcv_GetOpMode	API detects the actual software state of LIN transceiver driver.
LinTrcv_SetOpMode	The internal state of the LIN transceiver driver is switched to mode given in the parameter OpMode.
LinTrcv_SetWakeupMode	This API enables, disables and clears the notification for wakeup events on the addressed network.
Lin_CheckWakeup	This function checks if a wakeup has occurred on the addressed LIN channel.
PduR_LinTpCopyRxData	This function is called when transport protocol module have data to copy to the receiving module. Several calls may be made during one transportation of an I-PDU.
PduR_LinTpCopyTxData	This function is called by the transport protocol module to query the transmit data of an I-PDU segment. Each call to this function copies the next part of the transmit data until TpDataState indicates TP_DATARETRY. In this case the API restarts to copy the data beginning at the location indicated by TpTxDataCnt.
PduR_LinTpRxIndication	Called by the transport protocol module after an I-PDU has been received successfully or when an error occurred. It is also used to confirm cancellation of an I-PDU.
PduR_LinTpStartOfReception	This function will be called by the transport protocol module at the start

	of receiving an I-PDU. The I-PDU might be fragmented into multiple N-PDUs (FF with one or more following CFs) or might consist of a single N-PDU (SF). The service shall provide the currently available maximum buffer size when invoked with TpSdulength equal to 0.
PduR_LinTpTxConfirmation	This function is called by a transport protocol module after the I-PDU has been transmitted on its network, the result will reveal if the transmission was successful or not.

⌋ ()

8.5.3 Configurable interfaces

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

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

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

8.5.3.1 <User>_ScheduleRequestConfirmation

[LINIF520] ⌈

Service name:	<User>_ScheduleRequestConfirmation	

Syntax:	`void <User>_ScheduleRequestConfirmation(`
	`  NetworkHandleType channel,`
	`  LinIf_SchHandleType schedule`
	`)`	

Service ID[hex]:	0x00	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	channel	Identification of the LIN channel
	schedule	Index to newly active Schedule table
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	The LinIf will call this function when the schedule table change request has been performed.	

⌋ ()

[LINIF600] ⌈Configuration of <User>_ScheduleRequestConfirmation: The name of the API <User>_ScheduleRequestConfirmation which will be called by the LIN Interface module shall be configured for the LIN Interface module by parameter LinIfScheduleRequestConfirmationUL. ⌋()

8.5.3.2 <User>_GotoSleepConfirmation

[LINIF521] ⌈

Service name:	<User>_GotoSleepConfirmation	

Syntax:	`void <User>_GotoSleepConfirmation(`
	`  NetworkHandleType channel,`
	`  boolean success`
	`)`	

Service ID[hex]:	0x00	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	channel	Identification of the LIN channel
	success	True if goto sleep was successfully sent, false otherwise
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	The LinIf will call this function when the go to sleep command is sent not successfully/successfully on the bus.	

⌋()

[LINIF601] ⌈Configuration of <User>_GotoSleepConfirmation: The name of the API <User>_GotoSleepConfirmation which will be called by the LIN Interface module shall be configured for the LIN Interface module by parameter LinIfGotoSleepConfirmationUL. ⌋()

8.5.3.3 <User>_WakeupConfirmation

[LINIF522] ⌈

Service name:	<User>_WakeupConfirmation	

Syntax:	`void <User>_WakeupConfirmation(`
	`  NetworkHandleType channel,`
	`  boolean success`
	`)`	

Service ID[hex]:	0x00	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	channel	Identification of the LIN channel
	success	True if wakeup was successfully sent, false otherwise
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	

Description:	The LinIf will call this function when the wake up signal command is sent not successfully/successfully on the bus.

⌋()

[LINIF602] ⌈Configuration of <User>_WakeupConfirmation: The name of the API <User>_WakeupConfirmation which will be called by the LIN Interface module shall be configured for the LIN Interface module by parameter LinIfWakeupConfirmationUL. ⌋()

8.5.3.4 <User_TriggerTransmit>

[LINIF528] ⌈

Service name:	<User_TriggerTransmit>	

Syntax:	`Std_ReturnType <User_TriggerTransmit>(`
	`  PduIdType TxPduId,`
	`  PduInfoType* PduInfoPtr`
	`)`	

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

⌋()

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

8.5.3.5 <User_TxConfirmation>

[LINIF529] ⌈

Service name:	<User_TxConfirmation>	

Syntax:	`void <User_TxConfirmation>(`
	`  PduIdType TxPduId`
	`)`	

Sync/Async:	Synchronous	
Reentrancy:	Reentrant for different PduIds. Non reentrant for the same PduId.	
Parameters (in):	TxPduId	ID of the I-PDU that has been transmitted.

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

⌋()

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

8.5.3.6 <User_RxIndication>

[LINIF530] ⌈

Service name:	<User_RxIndication>	

Syntax:	`void <User_RxIndication>(`
	`  PduIdType RxPduId,`
	`  PduInfoType* PduInfoPtr`
	`)`	

Sync/Async:	Synchronous	
Reentrancy:	Reentrant for different PduIds. Non reentrant for the same PduId.	
Parameters (in):	RxPduId	ID of the received I-PDU.
	PduInfoPtr	Contains the length (SduLength) of the received I-PDU and a pointer to a buffer (SduDataPtr) containing the I-PDU.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Indication of a received I-PDU from a lower layer communication module.	

⌋()

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

9 Sequence diagrams

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

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

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

9.1 Frame Transmission

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

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

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

The dynamic diagram in Figure 7 does not show any timing information. The timing information is depicted in Figure 8 following the diagram.

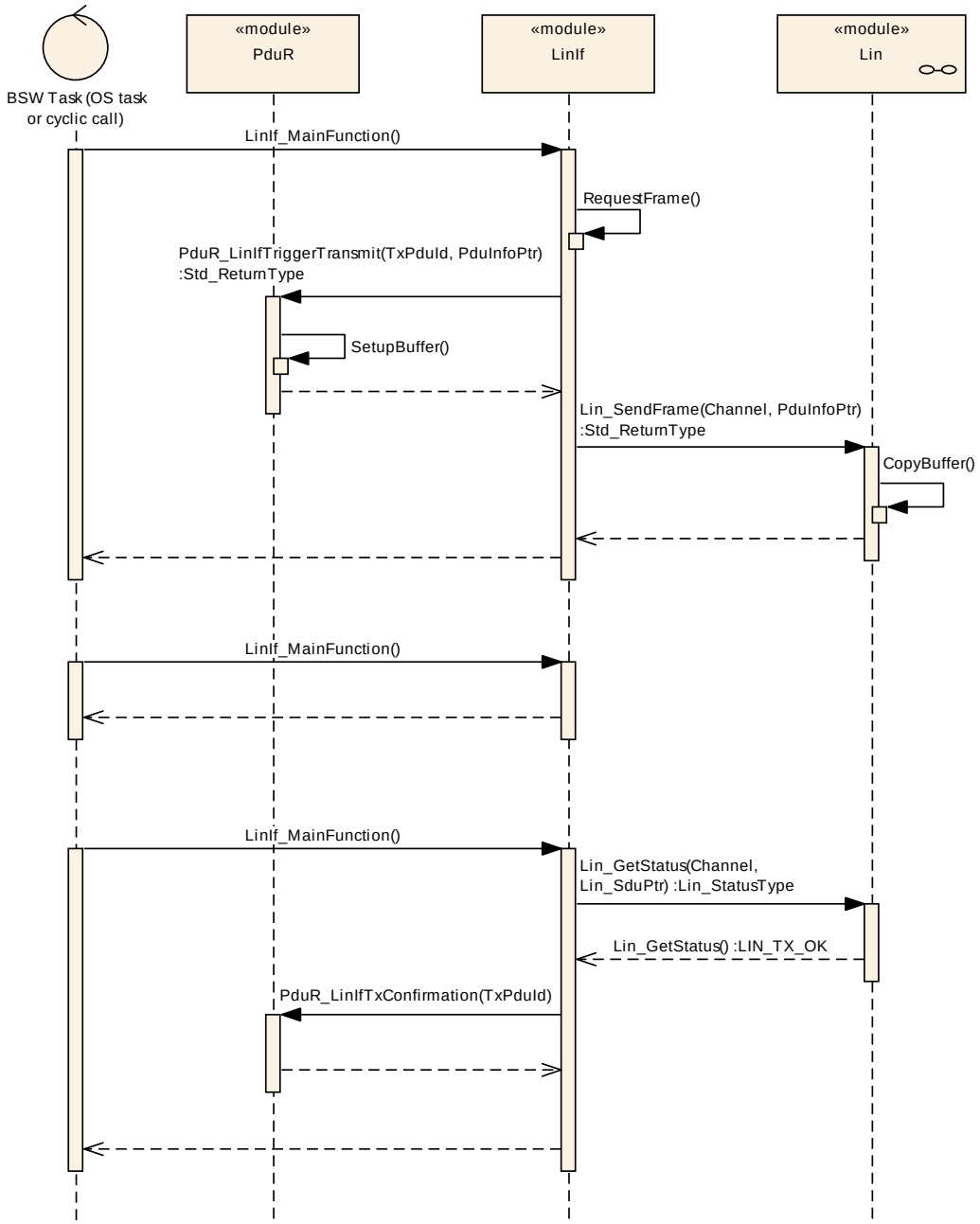

Figure 7 – Frame transmission

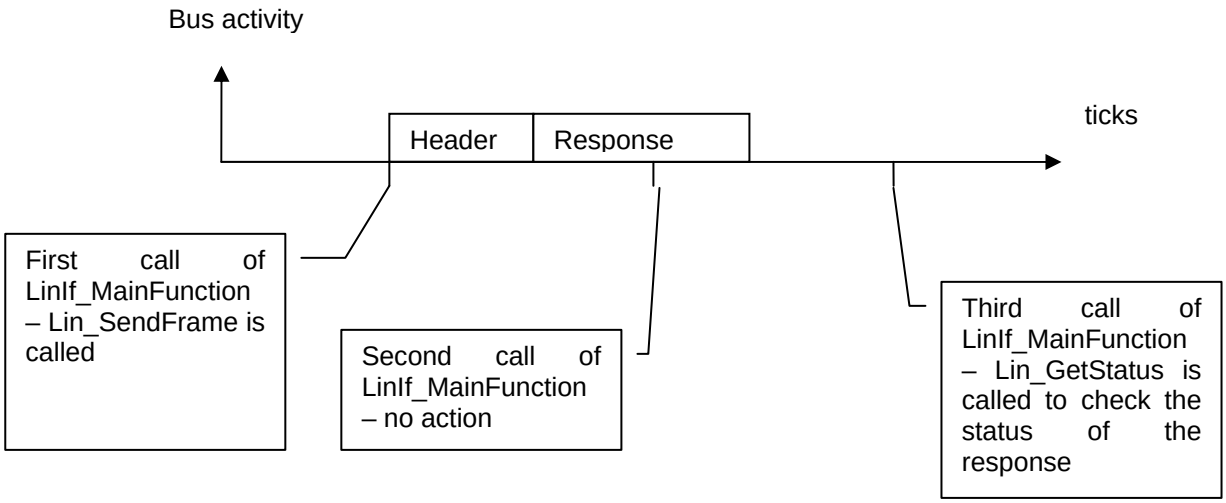

Figure 8 – Timing information for transmitted frame

9.2 Frame Reception

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

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

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

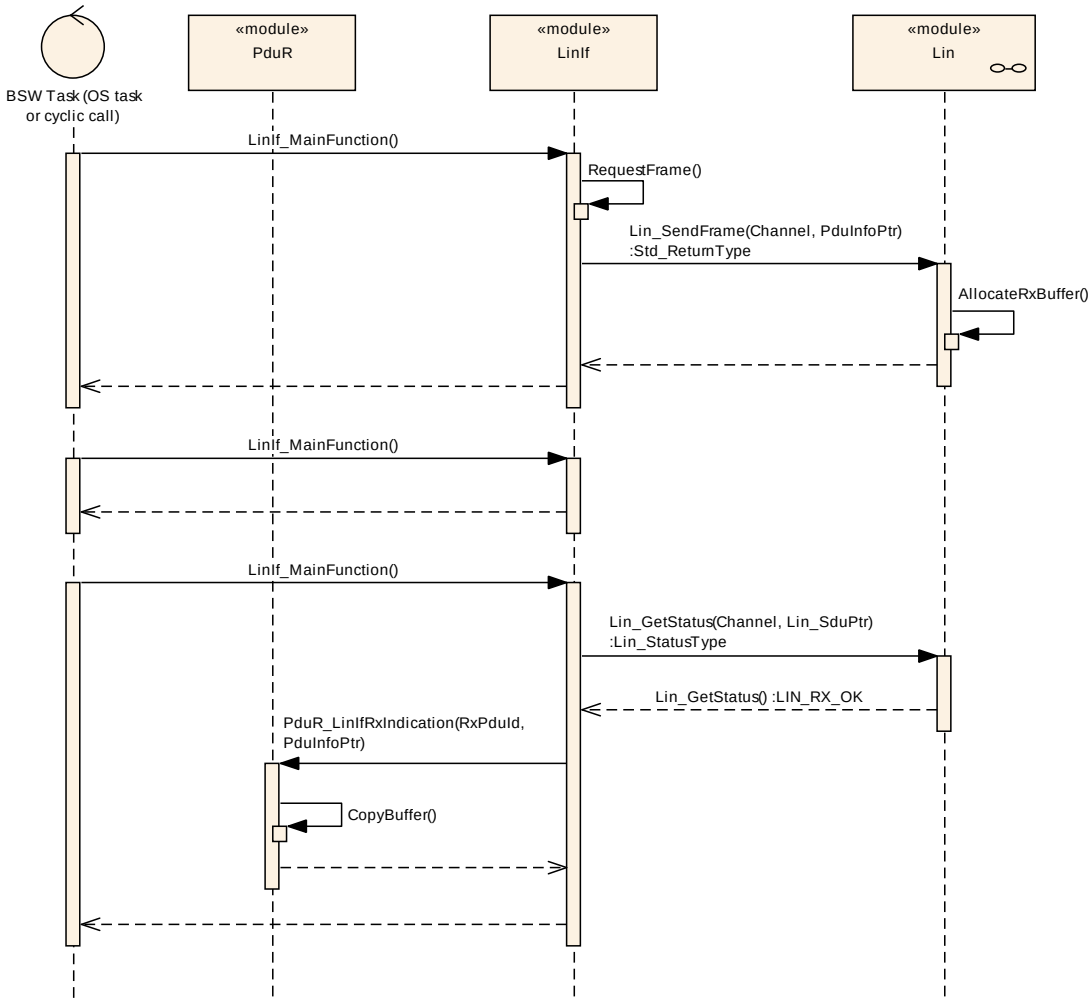

Figure 9 – Frame reception

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.

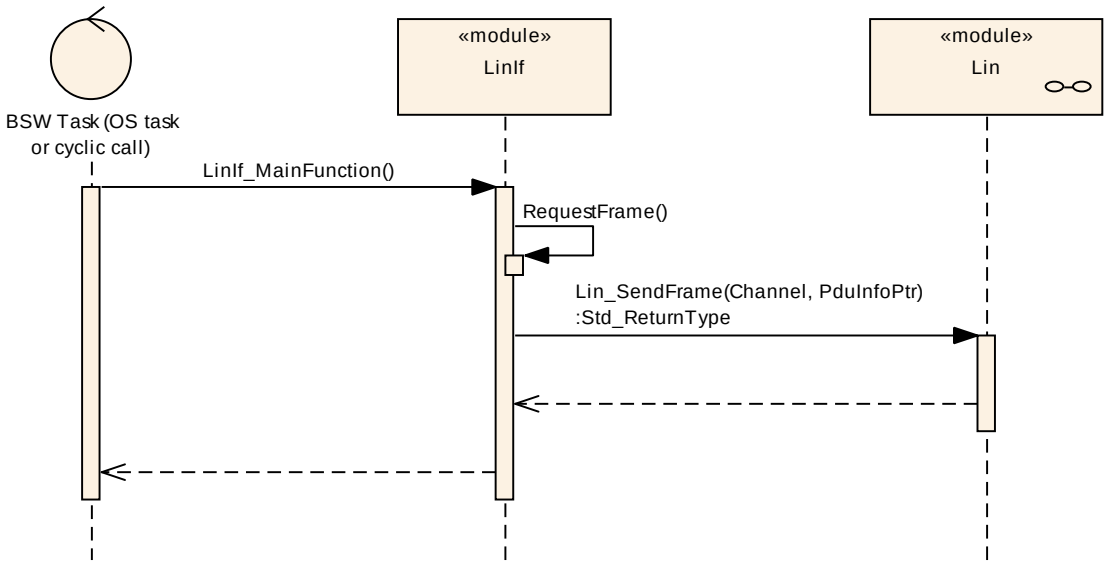

Figure 10 – Slave to slave communication

9.4 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.

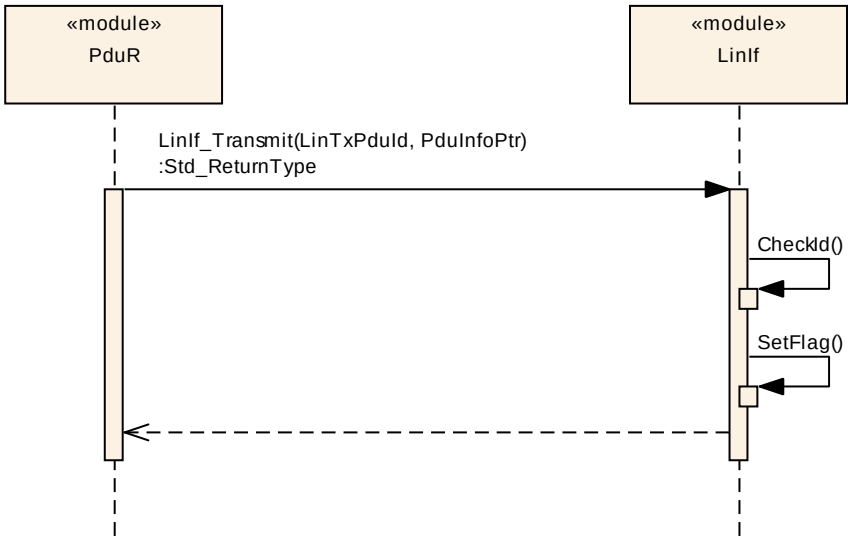

Figure 11 – Sporadic frame

9.5 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.

9.5.1 With no answer

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

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

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

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

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

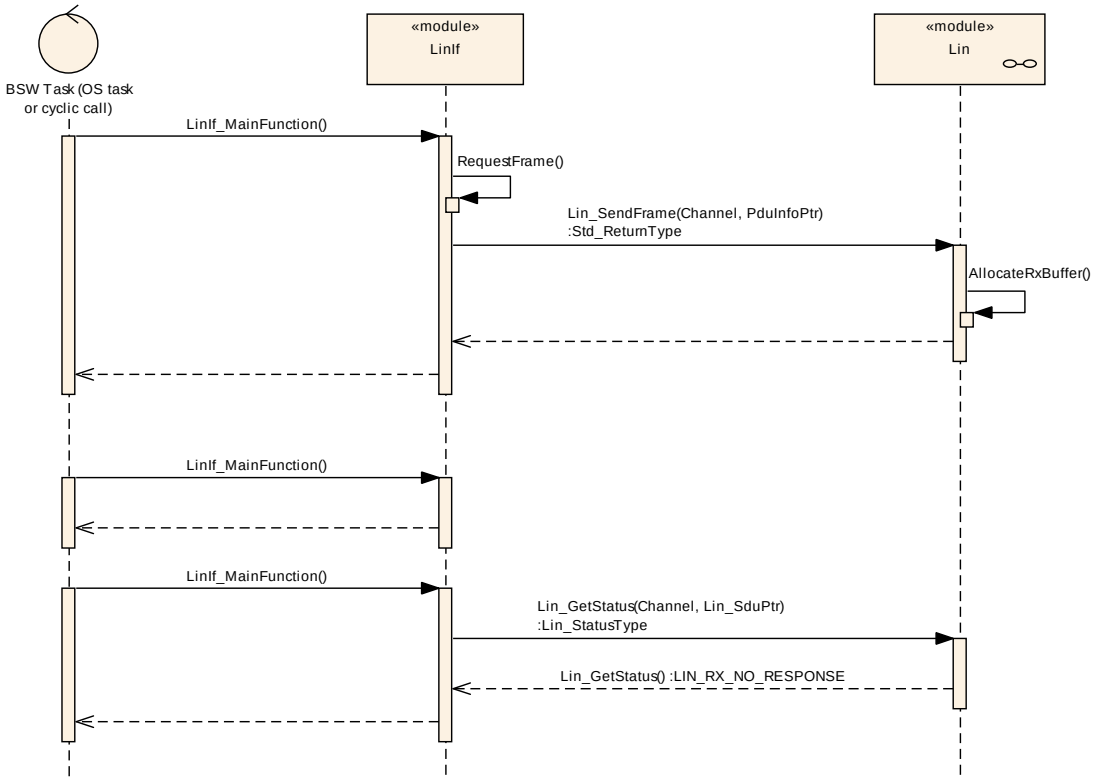

Figure 12 – Event-triggered frame with no answer

9.5.2 With answer (No collision)

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

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

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

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

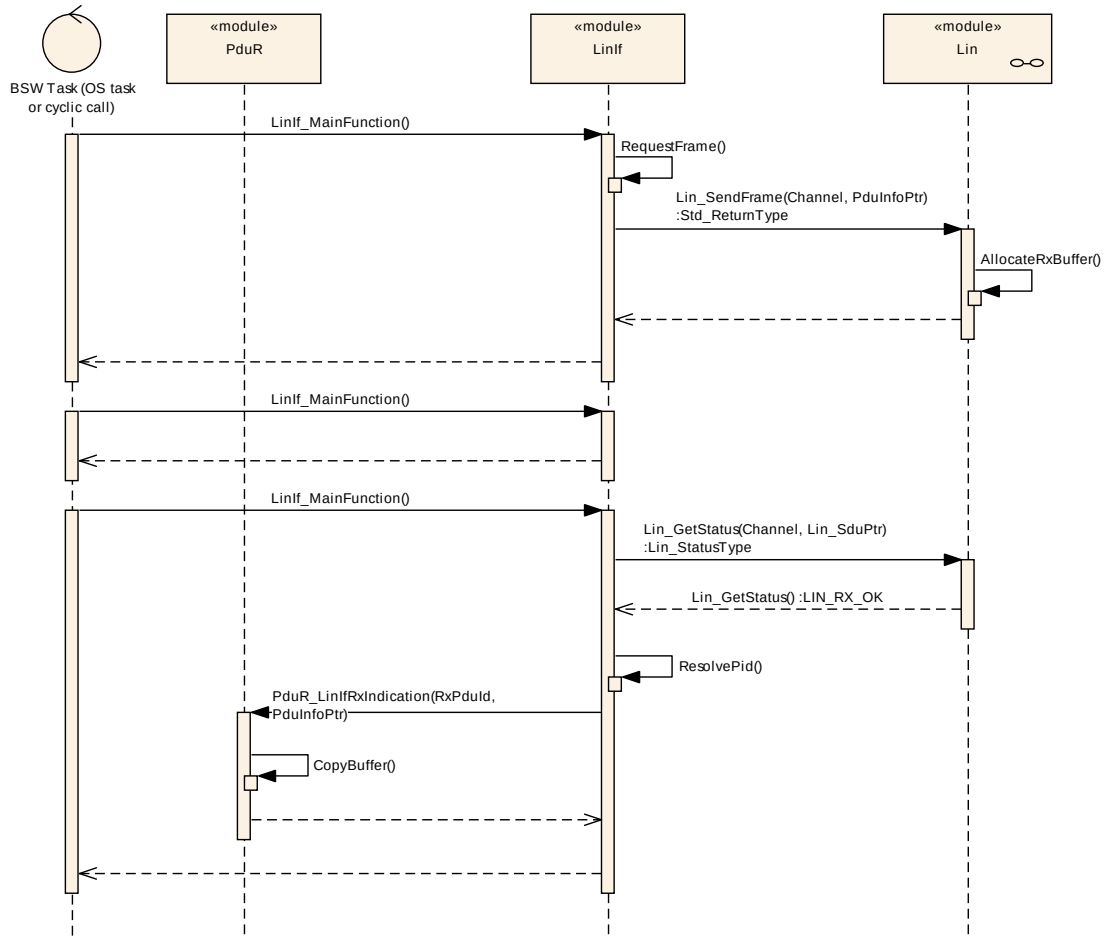

Figure 13 – Event-triggered frame with answer (no collision)

9.5.3 With collision

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

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

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

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

The local function ChangeToCollisionResolvingSchedule switches to the corresponding collision resolving schedule table to enable sporadic transmission from slave.

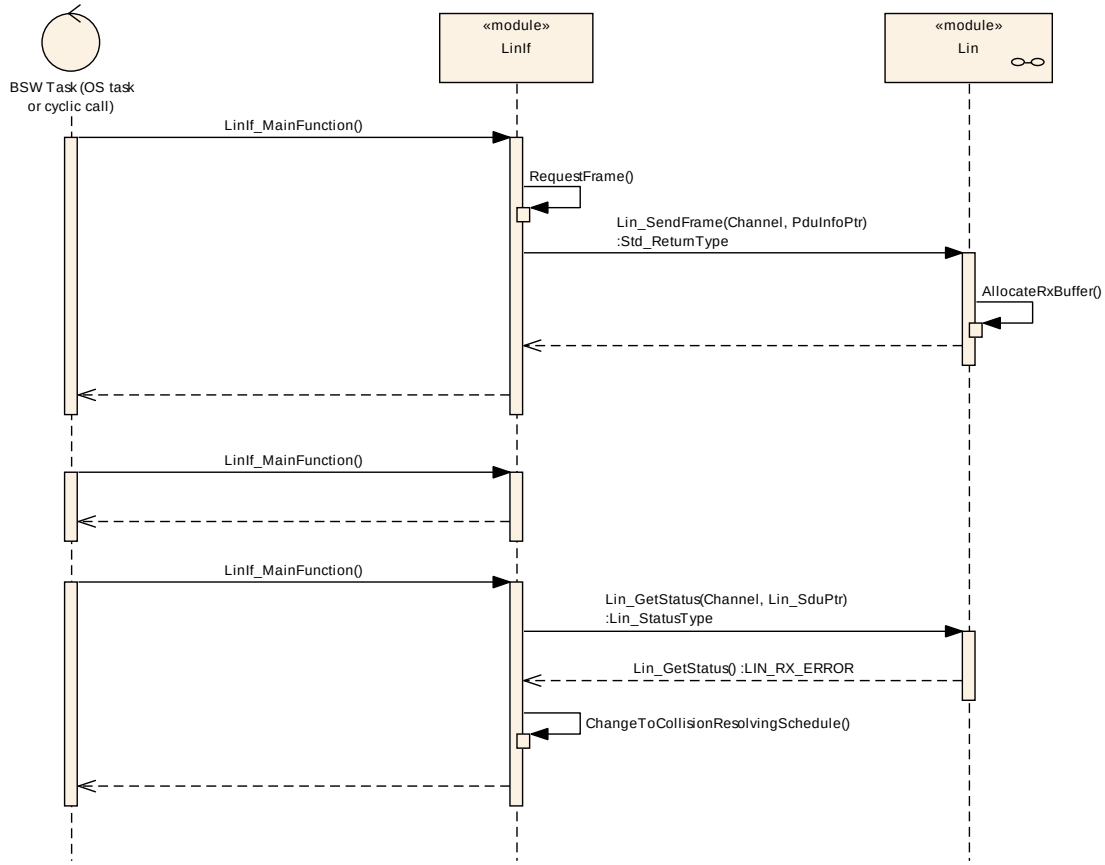

Figure 14 – Event-triggered frame with collision

9.6 Transport Protocol message transmission

The following diagram Figure 15 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 and it has the same behavior as frame transmission.

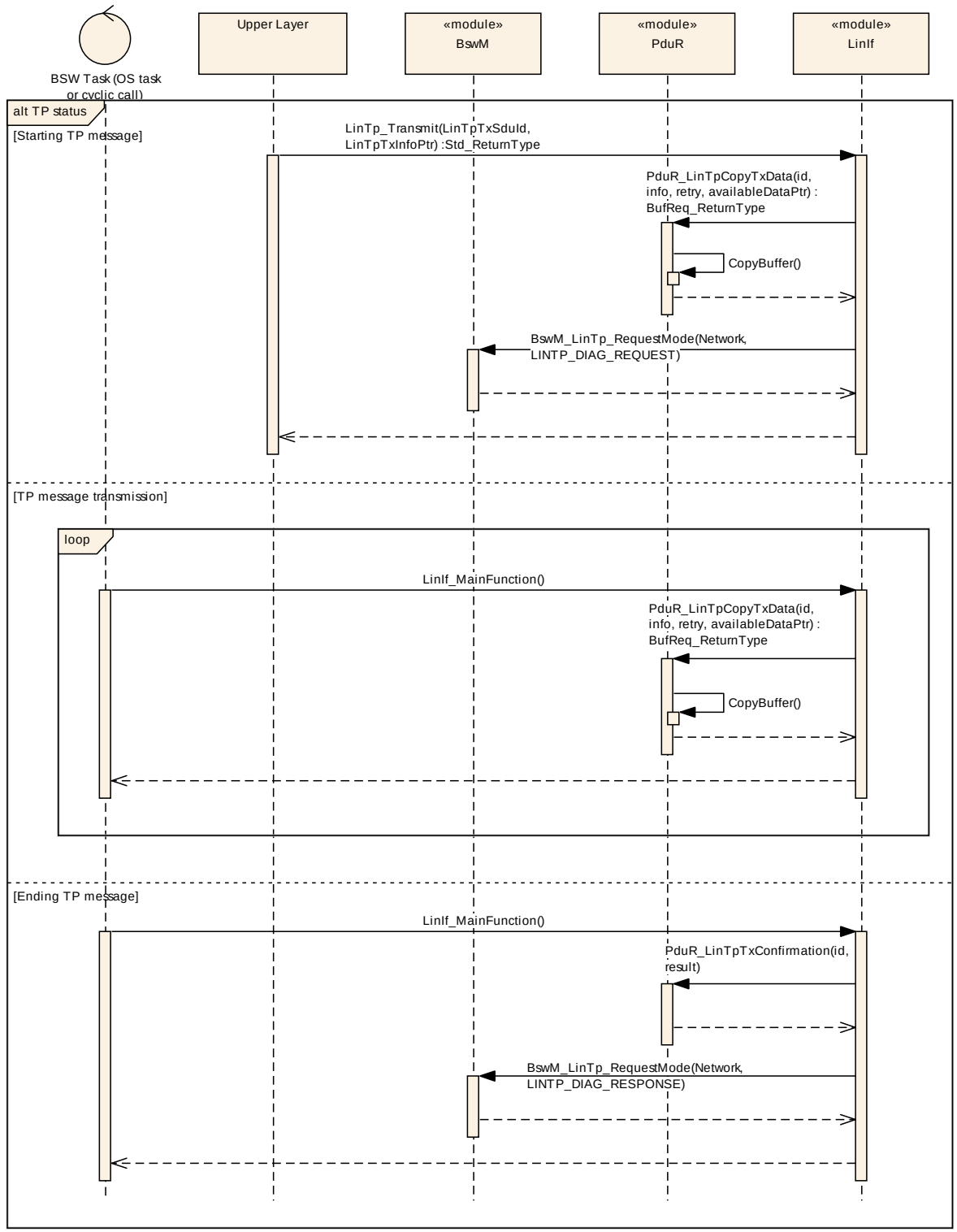

Figure 15 –Transport Protocol message transmission

9.7 Transport Protocol message reception

The following diagram Figure 16 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 SRF is not shown in the diagram and it has the same behavior as frame reception.

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

The continuous reception of the message is made by copying the N-SDU from the SRF to the provided buffer.

The TP message is finished after the last N-PDU (SF or CF) is received. The PDU Router will be notified of the reception of the complete message.

Note that the trivial case where more buffer space must be provided is not shown. A new buffer must then be requested before copying the TP message.

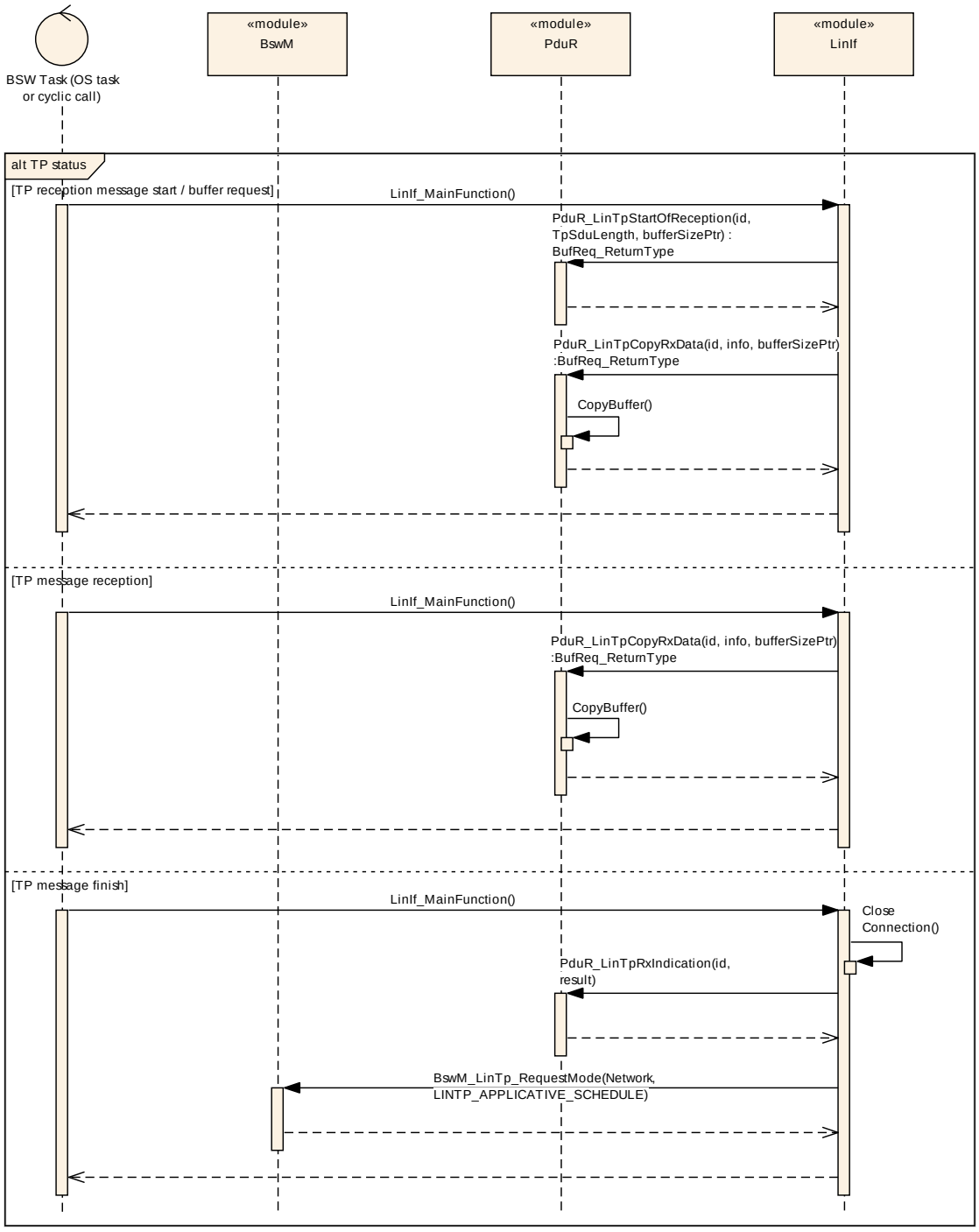

Figure 16 – Transport Protocol message reception

9.8 Go to sleep process

This use case in Figure 17 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.

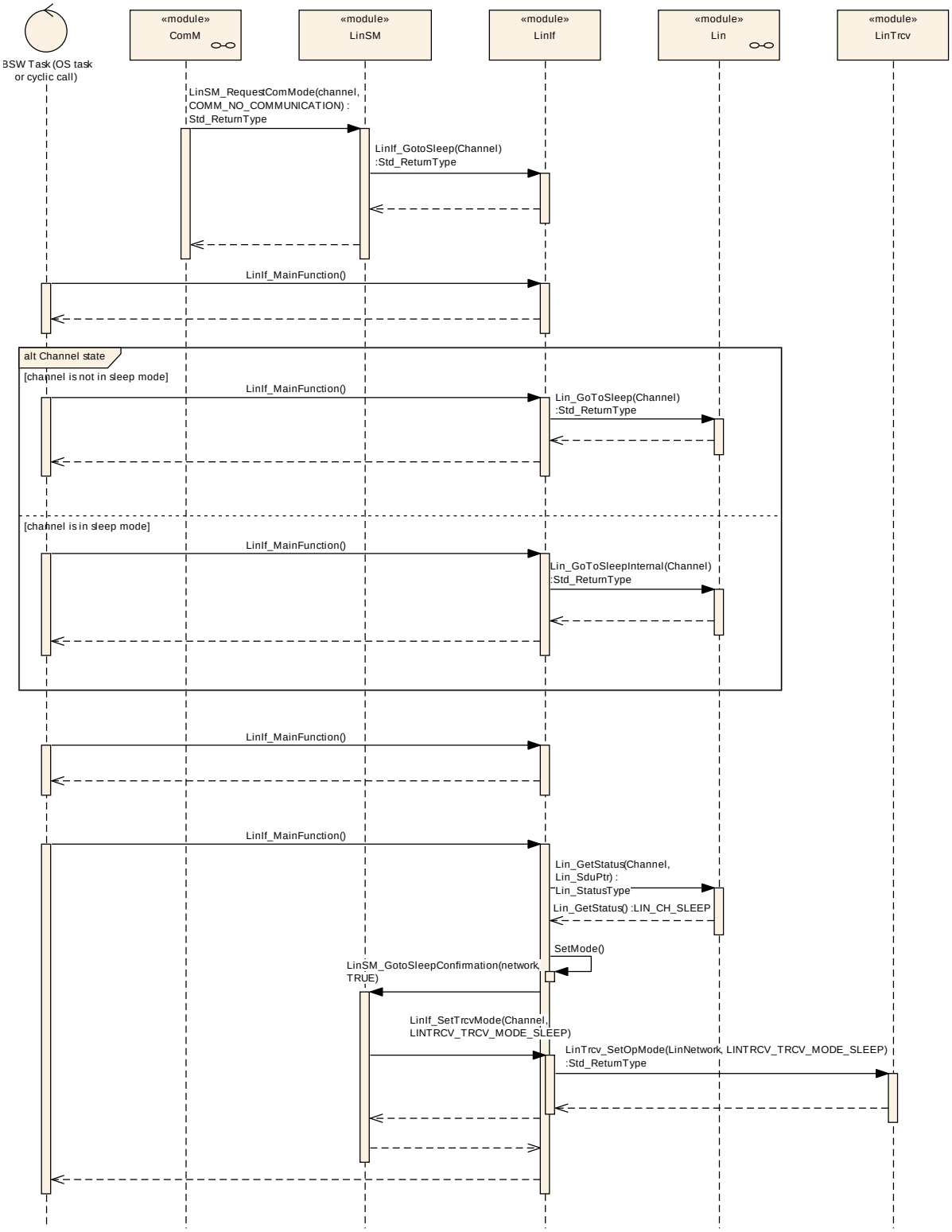

Figure 17 – Go-to-sleep command process

9.9 Wake up request

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

9.10 Internal wake-up

There are two different use cases in Figure 18:

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

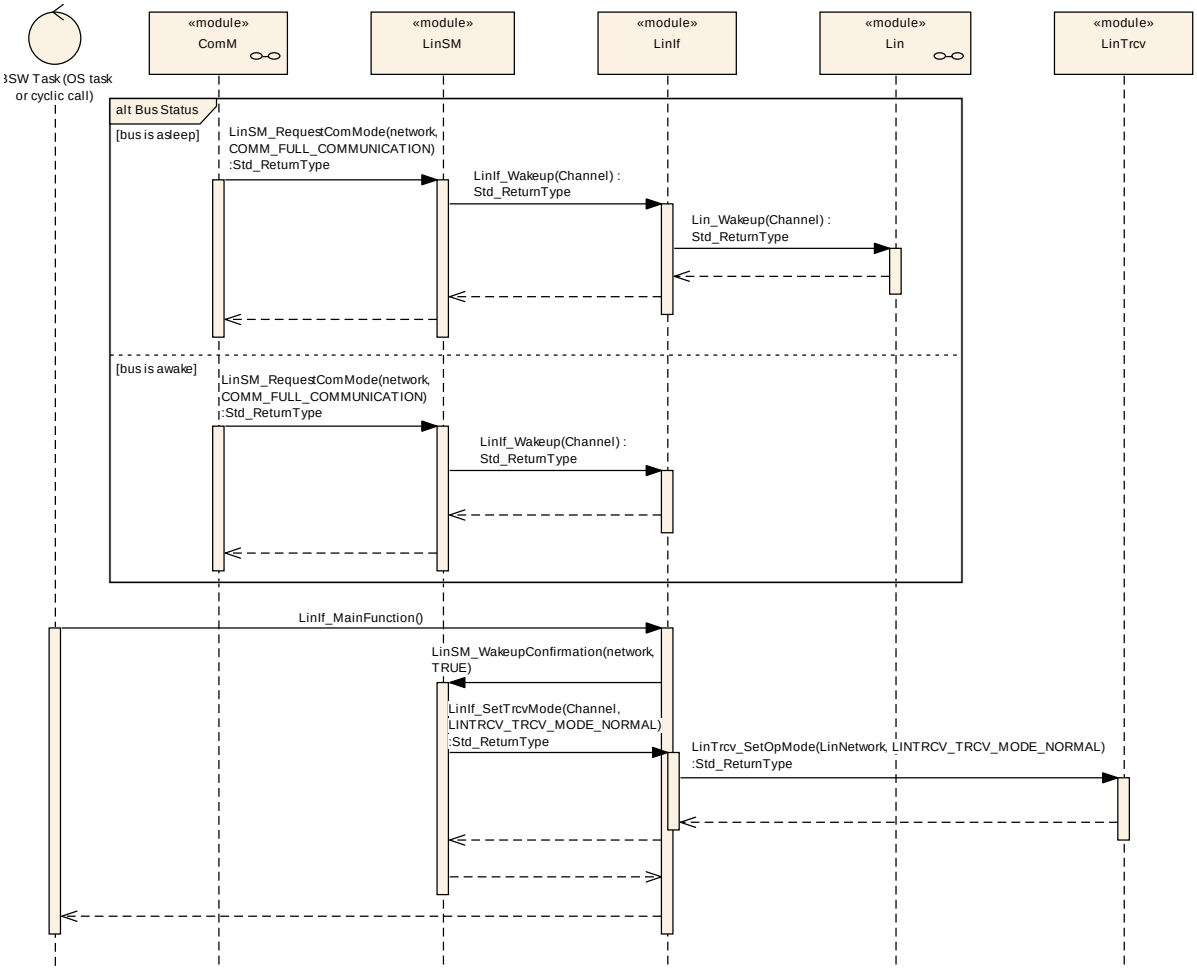

Figure 18 – Internal wake-up

10 Configuration specification

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

The chapter 10.3 specifies the structure (containers) and the parameters of the module LIN Interface. The chapter 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 chapter 7 and chapter 8.

[LINIF374] ⌈For post-build time support, the LIN Interface configuration structure LinIf_Configuration shall be constructed so that it may be exchangeable in memory.⌋ ()

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

10.2.1 Configuration Tool

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

The philosophy of the LIN 2.1 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. ⌋(BSW167)

10.2.2 Variants

Three configuration variants are defined for LIN Interface.

10.2.2.1 VARIANT-PRE-COMPILE

[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 VARIANT-LINK-TIME

[LINIF492] ⌈The variant VARIANT-LINK-TIME shall include all configuration options of the variant VARIANT-PRE-COMPILE. Additionally, all parameters that are marked 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 VARIANT-POST-BUILD

[LINIF493] ⌈This configuration includes all configuration options of the “VARIANT-LINK-TIME”. 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 19 depicts the LIN Interface configuration.

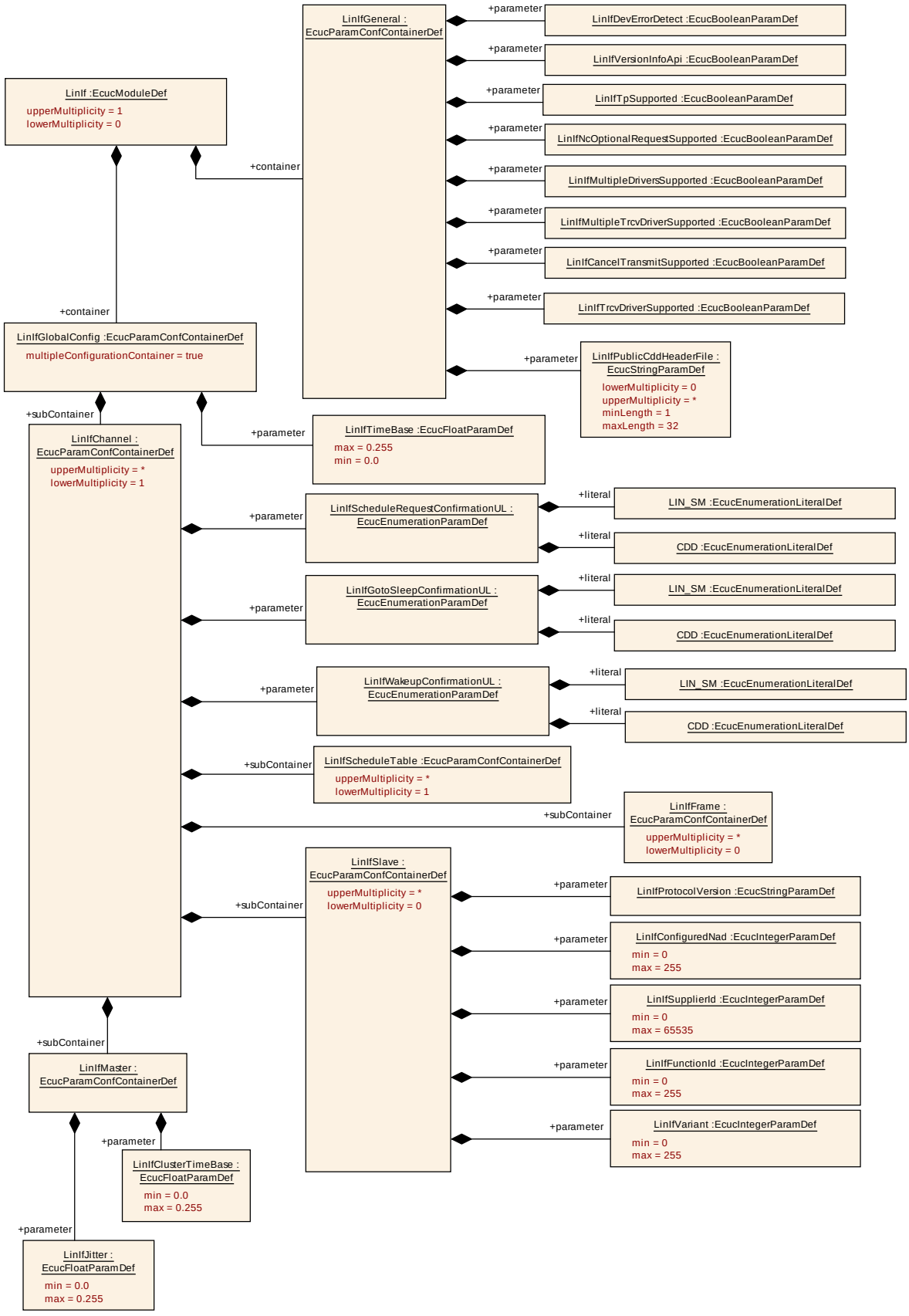

Figure 19 – LIN Interface configuration

10.3.1 LinIf

SWS Item	**LINIF370_Conf :**
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 LinIfGlobalConfig

SWS Item	**LINIF020_Conf :**
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	**LINIF044_Conf :**		
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	EcucFloatParamDef		
Range	0 .. 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	scope: Module		

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

10.3.3 LinIfGeneral

SWS Item	**LINIF019_Conf :**
Container Name	LinIfGeneral
Description	--
Configuration Parameters	

SWS Item	**LINIF001_Conf :**
Name	LinIfCancelTransmitSupported {LINIF_CANCEL_TRANSMIT_SUPPORTED}

Description	Global Pre-Compile Switch to reliably prevent the generation of the dummy LinIf_CancelTransmit API.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**LINIF010_Conf :**		
Name	LinIfDevErrorDetect {LINIF_DEV_ERROR_DETECT}		
Description	Switches the Development Error Detection and Notification ON or OFF.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**LINIF024_Conf :**		
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	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**LINIF025_Conf :**		
Name	LinIfMultipleTrcvDriverSupported {LINIF_MULTIPLE_TRCV_DRIVER_SUPPORTED}		
Description	States if multiple transceiver drivers are included in the LIN Interface or not. The reason for this parameter is to reduce the size of LIN Interface if multiple transceiver drivers are not used.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**LINIF026_Conf :**
Name	LinIfNcOptionalRequestSupported {LINIF_OPTIONAL_REQUEST_SUPPORTED}
Description	States if the node configuration commands Assign NAD and Conditional Change NAD are supported.
Multiplicity	1
Type	EcucBooleanParamDef
Default value	--

ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**LINIF631_Conf :**		
Name	LinIfPublicCddHeaderFile {LINIF_PUBLIC_CDD_HEADERFILE}		
Description	Defines header files for callback functions which shall be included in case of CDDs. Range of characters is 1.. 32.		
Multiplicity	0..*		
Type	EcucStringParamDef		
Default value	--		
maxLength	32		
minLength	1		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**LINIF045_Conf :**		
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	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**LINIF635_Conf :**		
Name	LinIfTrcvDriverSupported {LINIF_TRCV_DRIVER_SUPPORTED}		
Description	States if transceiver drivers are included in the LIN Interface or not. The reason for this parameter is to reduce the size of LIN Interface if transceiver drivers are not used.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**LINIF053_Conf :**		
Name	LinIfVersionInfoApi {LINIF_VERSION_INFO_API}		
Description	Switches the LinIf_GetVersionInfo function ON or OFF.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

No Included Containers

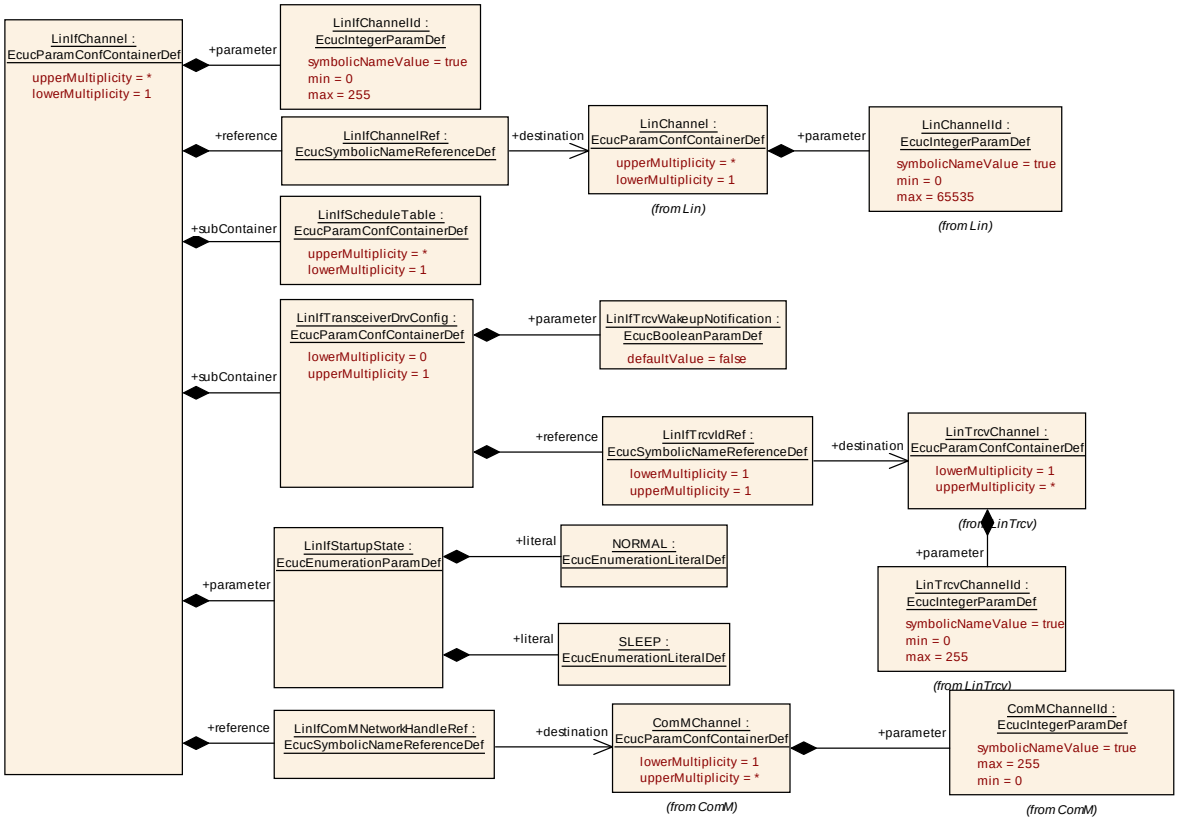

Figure 20 – LIN Interface Channel configuration

10.3.4 LinIfChannel

SWS Item	**LINIF364_Conf :**
Container Name	LinIfChannel{LinIf_Channel}
Description	--
Configuration Parameters	

SWS Item	**LINIF002_Conf :**		
Name	LinIfChannelId		
Description	This parameter holds the unique channel index value. The value shall be the same as the ComMChannelId of the ComMChannel referenced by LinIfComMNetworkHandleRef. Implementation Type: NetworkHandleType		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**LINIF601_Conf :**
Name	LinIfGotoSleepConfirmationUL

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

SWS Item	**LINIF600_Conf :**		
Name	LinIfScheduleRequestConfirmationUL		
Description	This parameter defines the upper layer (UL) module to which the confirmation of the successfully performed schedule table change.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	CDD	Complex Device Driver	
	LIN_SM	LIN State Manager	
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	**LINIF069_Conf :**		
Name	LinIfStartupState		
Description	Defines the state of each LIN channel after startup		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	NORMAL	The state of the channel shall be CHANNEL_OPERATIONAL after startup	
	SLEEP	The state of the channel shall be CHANNEL_SLEEP after startup	
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**LINIF602_Conf :**		
Name	LinIfWakeupConfirmationUL		
Description	This parameter defines the upper layer (UL) module to which the confirmation of the wake-up shall be sent.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	CDD	Complex Device Driver	
	LIN_SM	LIN State Manager	
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	**LINIF003_Conf :**
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	scope: Lin Channel		

SWS Item	**LINIF626_Conf :**		
Name	LinIfComMNetworkHandleRef		
Description	Unique handle to identify one certain LIN network. Reference to one of the network handles configured for the ComM.		
Multiplicity	1		
Type	Reference to [ComMChannel]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: Module		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
LinIfFrame	0..*	Generic container for all types of LIN frames. The shortName of this container is used as LinIfFrameName.
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. The ShortName of this container is used as LinIfNodeName.
LinIfTransceiverDrvConfig	0..1	This container contains the configuration (parameters) of all addressed LIN transceivers by each underlying LIN Transceiver Driver.

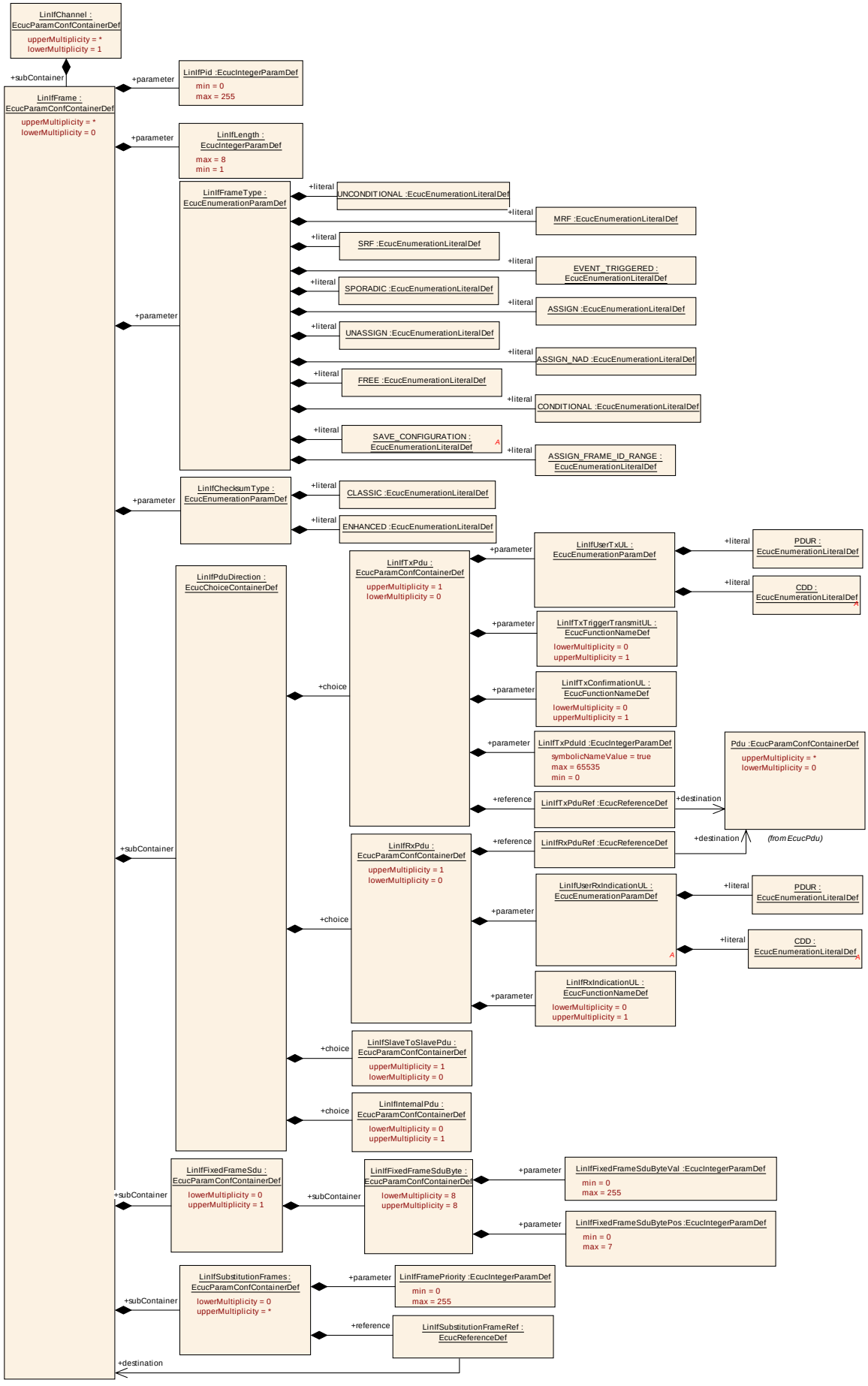
LinIfChannel : EcucParamConfContainerDef
upperMultiplicity = *
lowerMultiplicity = 1
+subContainer
LinIfFrame : EcucParamConfContainerDef
upperMultiplicity = *
lowerMultiplicity = 0
+parameter
LinIfPid :EcucIntegerParamDef
min = 0
max = 255
LinIfLength : EcucIntegerParamDef
max = 8
min = 1
LinIfFrameType : EcucEnumerationParamDef
+literal
UNCONDITIONAL :EcucEnumerationLiteralDef
MRF :EcucEnumerationLiteralDef
SRF :EcucEnumerationLiteralDef
EVENT_TRIGGERED : EcucEnumerationLiteralDef
SPORADIC :EcucEnumerationLiteralDef
ASSIGN :EcucEnumerationLiteralDef
UNASSIGN :EcucEnumerationLiteralDef
ASSIGN_NAD :EcucEnumerationLiteralDef
FREE :EcucEnumerationLiteralDef
CONDITIONAL :EcucEnumerationLiteralDef
SAVE_CONFIGURATION : EcucEnumerationLiteralDef
ASSIGN_FRAME_ID_RANGE : EcucEnumerationLiteralDef
LinIfChecksumType : EcucEnumerationParamDef
CLASSIC :EcucEnumerationLiteralDef
ENHANCED :EcucEnumerationLiteralDef
LinIfPduDirection : EcucChoiceContainerDef
+choice
LinIfTxPdu : EcucParamConfContainerDef
upperMultiplicity = 1
lowerMultiplicity = 0
LinIfUserTxUL : EcucEnumerationParamDef
PDUR : EcucEnumerationLiteralDef
CDD : EcucEnumerationLiteralDef
LinIfTxTriggerTransmitUL : EcucFunctionNameDef
lowerMultiplicity = 0
upperMultiplicity = 1
LinIfTxConfirmationUL : EcucFunctionNameDef
lowerMultiplicity = 0
upperMultiplicity = 1
LinIfTxPduId :EcucIntegerParamDef
symbolicNameValue = true
max = 65535
min = 0
LinIfTxPduRef :EcucReferenceDef
+reference
+destination
Pdu :EcucParamConfContainerDef
upperMultiplicity = *
lowerMultiplicity = 0
(from EcucPdu)
+subContainer
LinIfRxPdu : EcucParamConfContainerDef
upperMultiplicity = 1
lowerMultiplicity = 0
LinIfRxPduRef :EcucReferenceDef
LinIfUserRxIndicationUL : EcucEnumerationParamDef
LinIfRxIndicationUL : EcucFunctionNameDef
lowerMultiplicity = 0
upperMultiplicity = 1
LinIfSlaveToSlavePdu : EcucParamConfContainerDef
upperMultiplicity = 1
lowerMultiplicity = 0
LinIfInternalPdu : EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 1
LinIfFixedFrameSdu : EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 1
LinIfFixedFrameSduByte : EcucParamConfContainerDef
lowerMultiplicity = 8
upperMultiplicity = 8
LinIfFixedFrameSduByteVal :EcucIntegerParamDef
min = 0
max = 255
LinIfFixedFrameSduBytePos :EcucIntegerParamDef
min = 0
max = 7
LinIfSubstitutionFrames : EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *
LinIfFramePriority :EcucIntegerParamDef
min = 0
max = 255
LinIfSubstitutionFrameRef : EcucReferenceDef
+destination

Figure 21 – LIN Interface Frame configuration

10.3.5 LinIfFrame

SWS Item	**LINIF367_Conf :**		
Container Name	LinIfFrame{LinIf_Frame}		
Description	Generic container for all types of LIN frames. The shortName of this container is used as LinIfFrameName.		
Configuration Parameters			

SWS Item	**LINIF005_Conf :**		
Name	LinIfChecksumType {LINIF_CHECKSUM_TYPE}		
Description	Type of checksum that the frame is using.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
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	scope: Module		

SWS Item	**LINIF017_Conf :**		
Name	LinIfFrameType {LINIF_FRAME_TYPE}		
Description	Type of frame that is described (e.g. sporadic frame). The sporadic slot is not found among the frame types. A sporadic slot is a set of sporadic frames.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	ASSIGN	AssignFrameId	
	ASSIGN_FRAME_ID_RANGE	AssignFrameIdRange	
	ASSIGN_NAD	AssignNAD	
	CONDITIONAL	Conditional Change NAD	
	EVENT_TRIGGERED	Event triggered frame	
	FREE	FreeFormat	
	MRF	MRF	
	SAVE_CONFIGURATION	SaveConfiguration	
	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	scope: Module		

SWS Item	**LINIF023_Conf :**		
Name	LinIfLength {LINIF_LENGTH}		
Description	Length of the LIN SDU in bytes.		
Multiplicity	1		
Type	EcucIntegerParamDef		
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	scope: Module		

SWS Item	**LINIF028_Conf :**		
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	EcucIntegerParamDef		
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	scope: Module		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
LinIfFixedFrameSdu	0..1	In case this is a fixed frame this is the SDU (response). This container 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 a sporadic Frame slot.

10.3.6 LinIfFixedFrameSdu

SWS Item	**LINIF012_Conf :**
Container Name	LinIfFixedFrameSdu{LINIF_FIXED_FRAME_SDU}
Description	In case this is a fixed frame this is the SDU (response). This container represent an eight byte array. The Byte order shall be MSB first.
Configuration Parameters	

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

10.3.7 LinIfFixedFrameSduByte

SWS Item	**LINIF013_Conf :**
Container Name	LinIfFixedFrameSduByte
Description	This container represents a byte within the 8 byte array.
Configuration Parameters	

SWS Item	**LINIF014_Conf :**	
Name	LinIfFixedFrameSduBytePos	
Description	Index of the Byte in the SDU (response) 8 byte array.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 7	

Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**LINIF015_Conf :**		
Name	LinIfFixedFrameSduByteVal		
Description	Byte value in the SDU (response) 8-byte array.		
Multiplicity	1		
Type	EcucIntegerParamDef		
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	scope: Module		

No Included Containers

10.3.8 LinIfPduDirection

SWS Item	**LINIF027_Conf :**
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

10.3.9 LinIfSubstitutionFrames

SWS Item	**LINIF042_Conf :**
Container Name	LinIfSubstitutionFrames
Description	List of unconditional Frames that can be sent in a sporadic Frame slot.
Configuration Parameters	

SWS Item	**LINIF513_Conf :**	
Name	LinIfFramePriority {LINIF_FRAME_PRIORITY}	
Description	Priority of an unconditional frame if used as a sporadic frame.	
Multiplicity	1	
Type	EcucIntegerParamDef	
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	scope: Module		

SWS Item	**LINIF041_Conf :**		
Name	LinIfSubstitutionFrameRef		
Description	Reference to an unconditional Frame that can be sent in a sporadic Frame slot.		
Multiplicity	1		
Type	Reference to [LinIfFrame]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

No Included Containers

10.3.10 LinIfRxPdu

SWS Item	**LINIF035_Conf :**
Container Name	LinIfRxPdu
Description	represents a received PDU/frame
Configuration Parameters	

SWS Item	**LINIF055_Conf :**		
Name	LinIfRxIndicationUL		
Description	This parameter defines the name of the <User_RxIndication>. This parameter depends on the parameter LinIfUserRxIndicationUL. If LinIfUserRxIndicationUL equals PDUR, the name of the <User_RxIndication> is fixed. If LinIfUserRxIndicationUL equals CDD, the name of the <User_RxIndication> is selectable.		
Multiplicity	0..1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	**LINIF610_Conf :**	
Name	LinIfUserRxIndicationUL	
Description	This parameter defines the upper layer (UL) module to which the indication of the successfully received LINRXPDUID has to be routed via <User_RxIndication>. This <User_RxIndication> has to be invoked when the indication of the configured LINRXPDUID will be received by a Rx indication event from the LIN Driver module.	
Multiplicity	1	
Type	EcucEnumerationParamDef	
Range	CDD	Complex Device Driver

	PDUR	Pdu Router	
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	**LINIF036_Conf :**		
Name	LinIfRxPduRef		
Description	Reference to the PDU that is received in this frame.		
Multiplicity	1		
Type	Reference to [Pdu]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: PduId		

No Included Containers

10.3.11 LinIfTxPdu

SWS Item	**LINIF049_Conf :**
Container Name	LinIfTxPdu
Description	represents a transmitted PDU/frame
Configuration Parameters	

SWS Item	**LINIF054_Conf :**		
Name	LinIfTxConfirmationUL		
Description	This parameter defines the name of the <User_TxConfirmation>. This parameter depends on the parameter LinIfUserTxUL. If LinIfUserTxUL equals PDUR, the name of the <User_TxConfirmation> is fixed. If LinIfUserTxUL equals CDD, the name of the <User_TxConfirmation> is selectable.		
Multiplicity	0..1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	**LINIF050_Conf :**		
Name	LinIfTxPduId {LINIF_PDU_ID}		
Description	Identifier of the Pdu for the upper layer.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD

Scope / Dependency	scope: Module

SWS Item	**LINIF628_Conf :**		
Name	LinIfTxTriggerTransmitUL		
Description	This parameter defines the name of the <User_TriggerTransmit>. This parameter depends on the parameter LinIfUserTxUL. If LinIfUserTxUL equals PDUR, the name of the <User_TriggerTransmit> is fixed. If LinIfUserTxUL equals CDD, the name of the <User_TriggerTransmit> is selectable.		
Multiplicity	0..1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	**LINIF609_Conf :**		
Name	LinIfUserTxUL		
Description	This parameter defines the upper layer (UL) module to which the trigger of the transmitted LinTxPdu (via the <User_TriggerTransmit>) or the confirmation of the successfully transmitted LinTxPdu has to be routed (via the <User_TxConfirmation>).		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	CDD	Complex Device Driver	
	PDUR	Pdu Router	
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	**LINIF051_Conf :**		
Name	LinIfTxPduRef		
Description	Reference to the PDU that is transmitted in this frame.		
Multiplicity	1		
Type	Reference to [Pdu]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: PduId		

No Included Containers

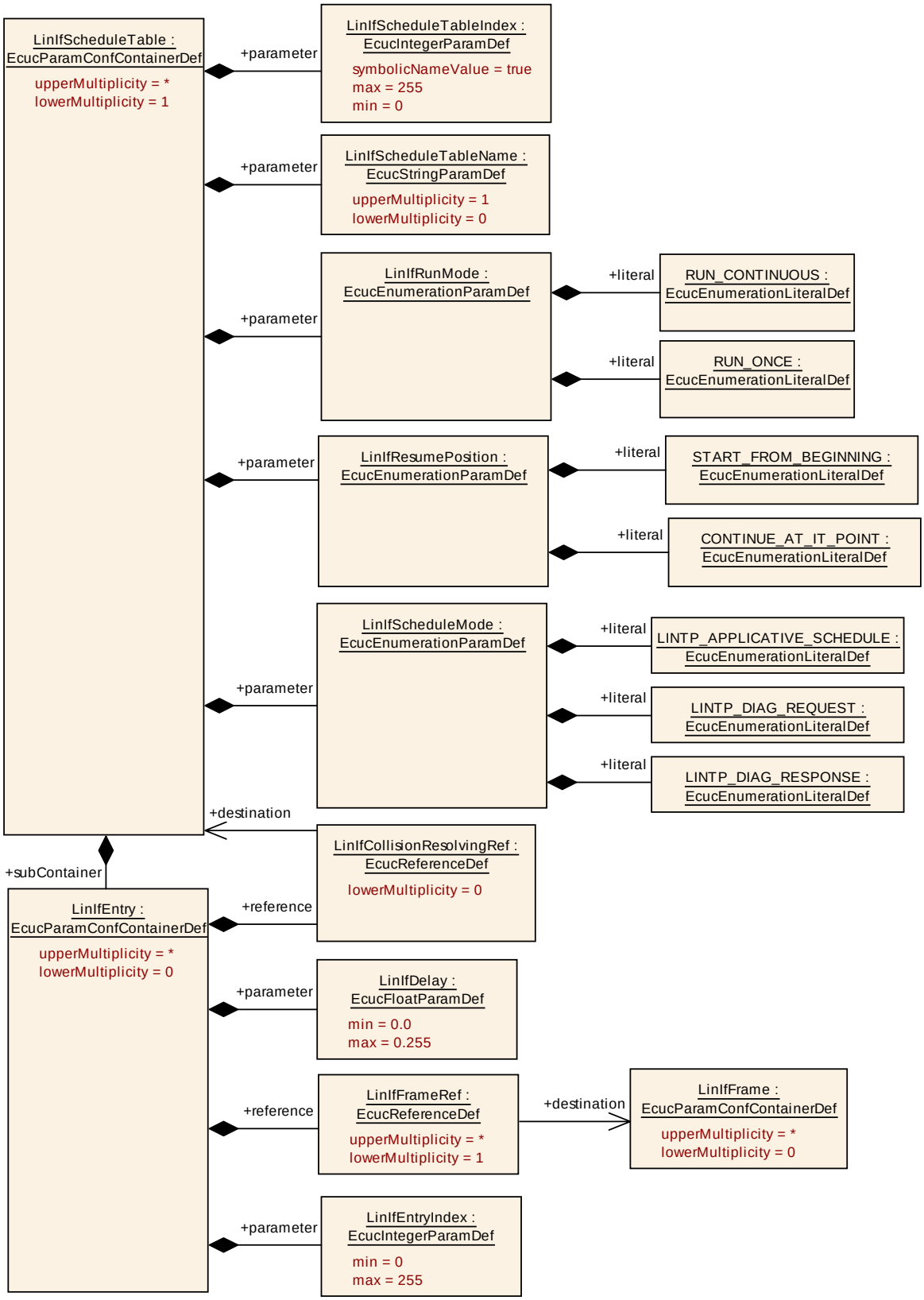

Figure 22 – LIN Interface Schedule Table configuration

10.3.12 LinIfScheduleTable

SWS Item	**LINIF365_Conf :**		
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	**LINIF033_Conf :**		
Name	LinIfResumePosition		
Description	Defines, where a schedule table shall be proceeded in case if it has been interrupted by a RUN-ONCE table.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	CONTINUE_AT_IT_POINT	= 1 Continue at IT Point	
	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	scope: Module		

SWS Item	**LINIF034_Conf :**		
Name	LinIfRunMode {LINIF_RUN_MODE}		
Description	The schedule table can be executed in two different modes.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
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	scope: Module		

SWS Item	**LINIF630_Conf :**		
Name	LinIfScheduleMode		
Description	The schedule table can be executed in three different modes.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	LINTP_APPLICATIVE_SCHEDULE	--	
	LINTP_DIAG_REQUEST	--	
	LINTP_DIAG_RESPONSE	--	
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**LINIF037_Conf :**		
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	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
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	scope: Module		

SWS Item	**LINIF038_Conf :**		
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	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

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

10.3.13 LinIfEntry

SWS Item	**LINIF366_Conf :**
Container Name	LinIfEntry{LinIf_Entry}
Description	Describes an entry in the schedule table (also known as Frame Slot).
Configuration Parameters	

SWS Item	**LINIF009_Conf :**		
Name	LinIfDelay {LINIF_DELAY}		
Description	Delay to next entry in schedule table in seconds.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	0 .. 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	scope: Module		

SWS Item	**LINIF011_Conf :**		
Name	LinIfEntryIndex		
Description	Position of the Frame Entry in the Schedule Table. The first entry index in the schedule table is 0.		
Multiplicity	1		
Type	EcucIntegerParamDef		
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	scope: Module		

SWS Item	**LINIF007_Conf :**		
Name	LinIfCollisionResolvingRef		
Description	Reference to the schedule table, which resolves the collision.		
Multiplicity	0..1		
Type	Reference to [LinIfScheduleTable]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**LINIF016_Conf :**		
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	scope: Module		

No Included Containers

10.3.14 LinIfMaster

SWS Item	**LINIF512_Conf :**
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	**LINIF006_Conf :**		
Name	LinIfClusterTimeBase {LINIF_TIME_BASE}		
Description	Defines a time-base for one LIN cluster in seconds (normally 0.002, 0.005 or 0.010s).		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	0 .. 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	scope: Module		

SWS Item	**LINIF629_Conf :**		
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	EcucFloatParamDef		
Range	0 .. 0.255		

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

No Included Containers

10.3.15 LinIfSlave

SWS Item	**LINIF039_Conf :**
Container Name	LinIfSlave
Description	The Node attributes of the Slaves are provided with these parameter. The ShortName of this container is used as LinIfNodeName.
Configuration Parameters	

SWS Item	**LINIF008_Conf :**		
Name	LinIfConfiguredNad {LINIF_CONFIGURED_NAD}		
Description	Definition of the initial node address		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**LINIF018_Conf :**		
Name	LinIfFunctionId {LINIF_FUNCTION_ID}		
Description	LIN function ID		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**LINIF029_Conf :**		
Name	LinIfProtocolVersion {LINIF_PROTOCOL_VERSION}		
Description	Defines the LIN Protocol version which is used by the slave.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-

			BUILD
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**LINIF043_Conf :**		
Name	LinIfSupplierId {LINIF_SUPPLIER_ID}		
Description	LIN Supplier ID		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**LINIF052_Conf :**		
Name	LinIfVariant {LINIF_VARIANT}		
Description	Specifies the Variant ID		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: Module		

No Included Containers

10.3.16 LinIfSlaveToSlavePdu

SWS Item	**LINIF040_Conf :**
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

10.3.17 LinIfInternalPdu

SWS Item	**LINIF021_Conf :**
Container Name	LinIfInternalPdu
Description	Represents a Diagnostic or Configuration frame : no Message ID (no PduId).
Configuration Parameters	

No Included Containers

10.3.18 LinIfTransceiverDrvConfig

SWS Item	**LINIF046_Conf :**
Container Name	LinIfTransceiverDrvConfig
Description	This container contains the configuration (parameters) of all addressed LIN transceivers by each underlying LIN Transceiver Driver.
Configuration Parameters	

SWS Item	**LINIF048_Conf :**		
Name	LinIfTrcvWakeupNotification		
Description	Selects whether wakeup indication notification is supported. True: Enabled False: Disabled		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**LINIF047_Conf :**		
Name	LinIfTrcvIdRef		
Description	Logical handle of the underlying LIN transceiver to be served by the LIN Interface.		
Multiplicity	1		
Type	Reference to [LinTrcvChannel]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: ECU		

No Included Containers

10.4 LIN Transport Layer configuration

The Figure 23 shows the outline of the LIN Transport Protocol configuration.

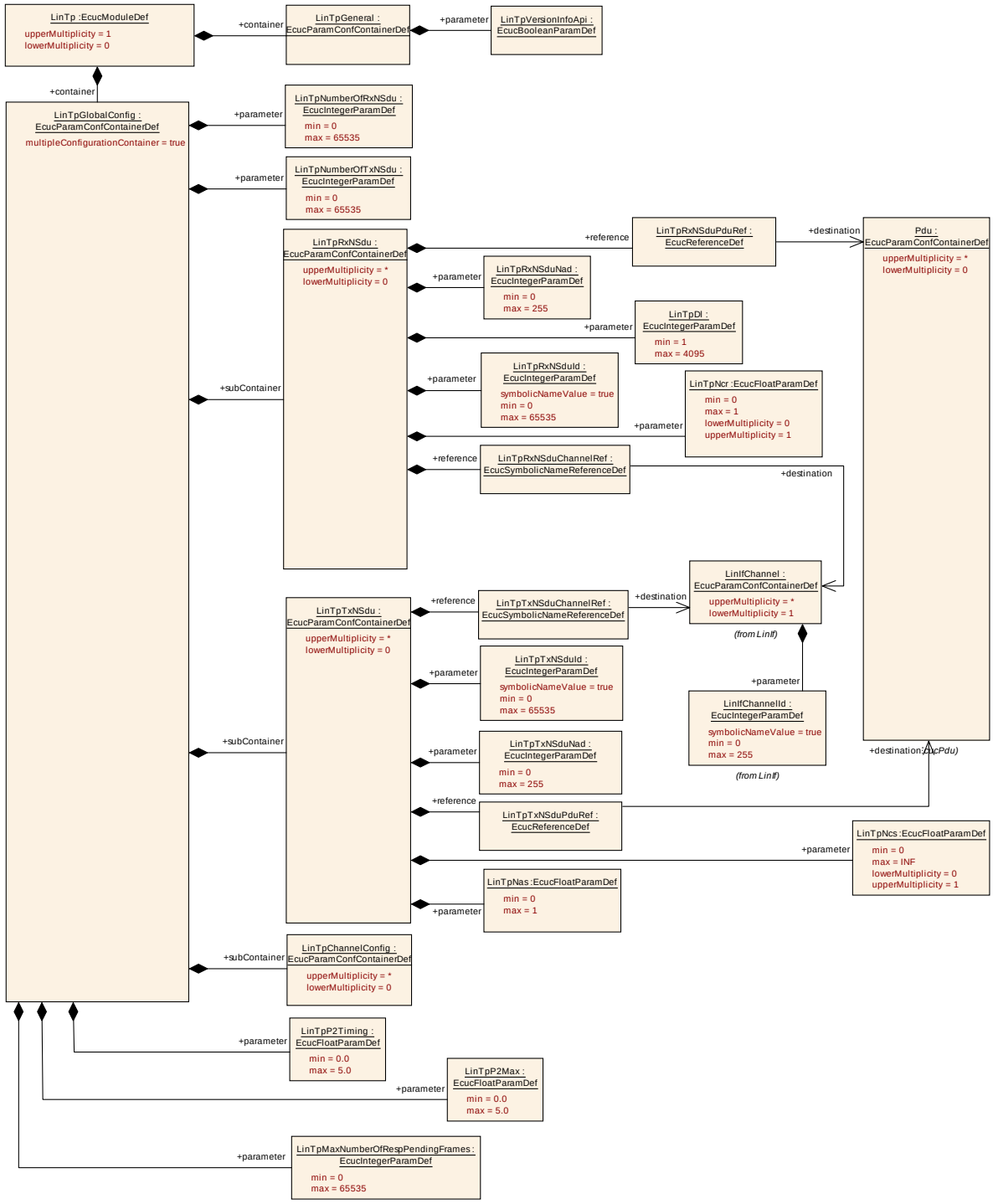

Figure 23 – LIN Transport Protocol configuration

10.4.1 LinTp

SWS Item	**LINIF425_Conf :**
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	**LINIF617_Conf :**
Container Name	LinTpGeneral
Description	Container that holds all LIN transport protocol general parameters.
Configuration Parameters	

SWS Item	**LINIF068_Conf :**		
Name	LinTpVersionInfoApi {LINTP_VERSION_INFO_API}		
Description	Switches the LinTp_GetVersionInfo function ON or OFF.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	**--**		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

No Included Containers

10.4.3 LinTpGlobalConfig

SWS Item	**LINIF056_Conf :**
Container Name	LinTpGlobalConfig [Multi Config Container]
Description	This container contains the global configuration parameter of the LinTp. It is a MultipleConfigurationContainer, i.e. this container and its sub-containers exit once per configuration set.
Configuration Parameters	

SWS Item	**LINIF624_Conf :**	
Name	LinTpMaxNumberOfRespPendingFrames	
Description	Configures the maximum number of allowed response pending frames.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 65535	

Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**LINIF057_Conf :**		
Name	LinTpNumberOfRxNSdu {LINTP_NUMBER_OF_RX_NSDU}		
Description	Number of transport protocol messages that can be received for all channels this node is connected to.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**LINIF058_Conf :**		
Name	LinTpNumberOfTxNSdu {LINTP_NUMBER_OF_TX_NSDU}		
Description	Number of transport protocol messages that can be transmitted for all channels this node is connected to.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**LINIF622_Conf :**		
Name	LinTpP2Max		
Description	P2 Timeout when a response pending frame is expected in seconds.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	0 .. 5		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**LINIF625_Conf :**		
Name	LinTpP2Timing		
Description	Definition of the P2 timeout observation parameter in seconds.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	0 .. 5		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

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

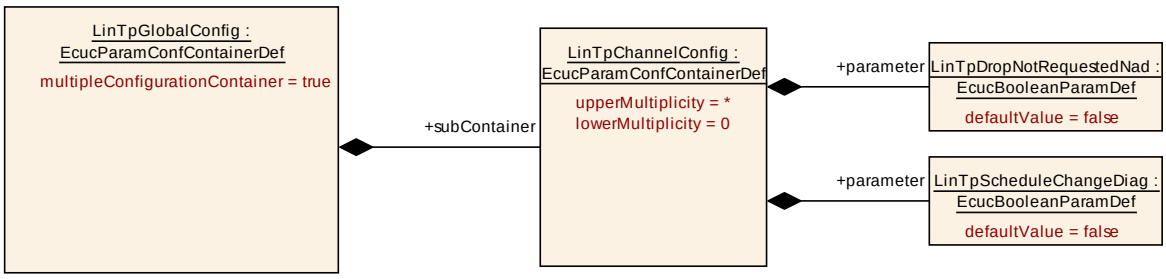

Figure 24 – LIN Transport Protocol Channel configuration

10.4.4 LinTpChannelConfig

SWS Item	**LINIF071_Conf :**
Container Name	LinTpChannelConfig{LINTP_CHANNEL_CONFIG}
Description	This container contains the channel specific configuration parameter of LinTp.
Configuration Parameters	

SWS Item	**LINIF072_Conf :**		
Name	LinTpDropNotRequestedNad {LINTP_DROP_NOT_REQUESTED_NAD}		
Description	Configures if TP Frames of not requested LIN-Slaves are dropped or not. false: Do drop TP Frames of Not requested LIN-Slaves true: Drop not TP Frames of Not requested LIN-Slaves		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**LINIF070_Conf :**		
Name	LinTpScheduleChangeDiag {LINTP_SCHEDULE_CHANNEL_DIAG}		
Description	Enables or disables the call of BswM_LinTp_RequestMode() to diagnostic request/response schedule. false: BswM is not called true: BswM is called		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

No Included Containers

10.4.5 LinTpRxNSdu

SWS Item	**LINIF428_Conf :**
Container Name	LinTpRxNSdu{LinTp_rx_NSDU}
Description	For each received N-SDU on any channel the node is connected to.
Configuration Parameters	

SWS Item	**LINIF616_Conf :**		
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	EcucIntegerParamDef		
Range	1 .. 4095		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**LINIF632_Conf :**		
Name	LinTpNcr {LINTP_NCR}		
Description	Value in seconds of the N_Cr timeout. N_Cr is the time until reception of the next Consecutive Frame N_PDU.		
Multiplicity	0..1		
Type	EcucFloatParamDef		
Range	0 .. 1		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**LINIF061_Conf :**		
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	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**LINIF062_Conf :**

Name	LinTpRxNSduNad {LINTP_NAD}		
Description	A N-SDU transported on LIN is identified using the NAD for the specific slave.		
Multiplicity	1		
Type	EcucIntegerParamDef		
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	scope: Module		

SWS Item	**LINIF060_Conf :**		
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	scope: Module		

SWS Item	**LINIF063_Conf :**		
Name	LinTpRxNSduPduRef		
Description	Reference to the global PDU		
Multiplicity	1		
Type	Reference to [Pdu]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: PduR		

No Included Containers

10.4.6 LinTpTxNSdu

SWS Item	**LINIF511_Conf :**
Container Name	LinTpTxNSdu{LinTp_tx_NSDU}
Description	For each transmitted N-SDU on any channel the node is connected to.
Configuration Parameters	

SWS Item	**LINIF633_Conf :**		
Name	LinTpNas {LINTP_NAS}		
Description	Value in second of the N_As timeout. N_As is the time for transmission of a LIN frame (any N_PDU) on the part of the sender.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	0 .. 1		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD

Scope / Dependency	scope: Module

SWS Item	**LINIF634_Conf :**		
Name	LinTpNcs {LINTP_NCS}		
Description	Value in seconds of the performance requirement of N_Cs. N_Cs is the time which elapses between the transmit request of a CF N-PDU until the transmit request of the next CF N-PDU.		
Multiplicity	0..1		
Type	EcucFloatParamDef		
Range	0 .. INF		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**LINIF065_Conf :**		
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	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**LINIF066_Conf :**		
Name	LinTpTxNSduNad {LINTP_NAD}		
Description	A N-SDU transported on LIN is identified using the NAD for the specific slave.		
Multiplicity	1		
Type	EcucIntegerParamDef		
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	scope: Module		

SWS Item	**LINIF064_Conf :**		
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	scope: Module		

SWS Item	**LINIF067_Conf :**
Name	LinTpTxNSduPduRef
Description	Reference to the global PDU
Multiplicity	1

Type	Reference to [Pdu]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: PduR		

No Included Containers

10.5 Published Information

[LINIF691] ⌈The standardized common published parameters as required by BSW00402 in the General Requirements on Basic Software Modules [3] shall be published within the header file of this module and need to be provided in the BSW Module Description. The according module abbreviation can be found in the List of Basic Software Modules [1].

Additional module-specific published parameters are listed below if applicable.⌋ (BSW00402, BSW00374, BSW00379, BSW00318)

11 Changes to Release 3

11.1 Deleted SWS Items

SWS Item	*Rationale*
LINIF470	
LINIF475	
LINIF480	
LINIF262	
LINIF114	
LINIF476	
LINIF390	
LINIF457	
LINIF380	
LINIF502	
LINIF391	
LINIF396	
LINIF392	
LINIF400	
LINIF395	
LINIF243	
LINIF407	
LINIF394	
LINIF505	
LINIF223	
LINIF294	
LINIF306	
LINIF598	
LINIF328	
LINIF077	
LINIF221	
LINIF587	
LINIF531	
LINIF504	
LINIF464	
LINIF523,603	
LINIF388	
LINIF532,533	
LINIF440	
LINIF591	
LINIF599	
LINIF524,525,526,527,583 LINIF604,605,606,607,611	
LINIF332	
LINIF325,430	
LINIF630 – 633	
LINIF582	
LINIF244	

11.2 Replaced SWS Items

SWS Item of Release 1	*replaced by SWS Item*	*Rationale*
LINIF504	LINIF506	

LINIF509	LINIF505	
LINIF508	LINIF504	
LINIF510	LINIF503	
LINIF463	LINIF502	
LINIF227	LINIF501	
LINIF469	LINIF495	
LINIF471	LINIF496	
LINIF453	LINIF497	
LINIF455	LINIF498	
LINIF458	LINIF499	
LINIF494	LINIF500	
LINIF500	LINIF561	Duplicate ID
LINIF495	LINIF589	Duplicate ID
LINIF496	LINIF590	Duplicate ID
LINIF501	LINIF591	Duplicate ID
LINIF506	LINIF592	Duplicate ID
LINIF382	LINIF383	

11.3 Changed SWS Items

SWS Item	***Rationale***
LINIF310	
LINIF075	
LINIF471	Support of LIN 2.1
LINIF470	Support of LIN 2.1
LINIF436	Support of LIN 2.1
LINIF012	Support of LIN 2.1
LINIF248	Support of LIN 2.1
LINIF014	Support of LIN 2.1
LINIF251	Support of LIN 2.1
LINIF472	Support of LIN 2.1
LINIF501	Support of LIN 2.1
LINIF237	Support of LIN 2.1
LINIF464	Support of LIN 2.1
LINIF401	Support of LIN 2.1
LINIF308	Support of LIN 2.1
LINIF309	Support of LIN 2.1
LINIF409	Support of LIN 2.1
LINIF310	Support of LIN 2.1
LINIF408	Support of LIN 2.1
LINIF090	Support of LIN 2.1
LINIF313	Support of LIN 2.1
LINIF430	
LINIF378	Harmonized wake-up validation
LINIF359	
LINIF488	
LINIF085	
LINIF087	
LINIF359	
LINIF436	
LINIF245	
LINIF245	
LINIF244	
LINIF434	
LINIF225	
LINIF226	

LINIF033	
LINIF186	
LINIF332	
LINIF376	Added development errors for transceiver driver
LINIF260	
LINIF014	
LINIF500	
LINIF485	
LINIF454	
LINIF444	
LINIF254	
LINIF105	
LINIF592	
LINIF497	
LINIF614	
LINIF578	
LINIF568	
LINIF350	
LINIF538	
LINIF383	
LINIF520	
LINIF378	
LINIF500	
LINIF532	
LINIF469,544,545	
LINIF205,547,550,522	
LINIF614,617,495	
LINIF320	
LINIF359,296,460,186,503	
LINIF533	
LINIF501	
LINIF618	
LINIF547	
LINIF405	
LINIF376,254,466,465,405	
LINIF500,501,580,592,625	
LINIF625	
LINIF351,613,616,628	
LINIF078	
LINIF469,547,550	
LINIF371,427	
LINIF432	
LINIF637	
LINIF360	
LINIF608	
LINIF556	
LINIF621	
LINIF628	
LINIF356	
LINIF079,081,613,614,648	
LINIF501,578	
LINIF278,353	
LINIF527,583	
LINIF520	
LINIF580	
LINIF525,527	
LINIF259	

LINIF069,073,615,617,618 LINIF619,620,623	
LINIF360	
LINIF197	
LINIF198	
LINIF202	
LINIF376,572	
LINIF432	
LINIF525	
LINIF068	
LINIF329,068,069,073,658 LINIF662,075,078,081,653 LINIF654,655,332,666,619 LINIF662,075,078,081,653 LINIF654,655,332,666,619 LINIF623,085,628,087,327 LINIF360	
LINIF329,330,068,069,073 LINIF075,078,086	
LINIF555	
LINIF634 – 637	
LINIF488	
LINIF535	
LINIF462,560	
LINIF359,360,376,387,469,503 LINIF566	

11.4 Added SWS Items

SWS Item	*Rationale*
LINIF515	
LINIF516	
LINIF517	
LINIF518	
LINIF519	
LINIF520	
LINIF521	
LINIF522	
LINIF523	
LINIF524	
LINIF525	
LINIF526	
LINIF527	
LINIF528	
LINIF529	
LINIF530	
LINIF531	LIN Transceiver Driver types added
LINIF532	LIN Transceiver Driver types added
LINIF533	LIN Transceiver Driver types added
LINIF534 – LINIF555	Added requirements for LIN Transceiver Driver support
LINIF556	
LINIF560	
LINIF567	
LINIF562	
LINIF568	
LINIF572	
LINIF563	

LINIF564	
LINIF565	
LINIF566	
LINIF575	
LINIF576	
LINIF577	
LINIF578	
LINIF569	
LINIF570	
LINIF571	
LINIF573	
LINIF574	
LINIF579	
LINIF580	
LINIF581	
LINIF582	Missing requirement for Lin_GoToSleepInternal
LINIF584	
LINIF586	
LINIF588	
LINIF593	
LINIF594	
LINIF595	
LINIF596	
LINIF597	
LINIF599	
LINIF600	
LINIF601	
LINIF602	
LINIF603	
LINIF604	
LINIF605	
LINIF606	
LINIF607	
LINIF608	
LINIF609	
LINIF610	
LINIF611	
LINIF612	
LINIF613	
LINIF614	
LINIF615	
LINIF616	
LINIF617	
LINIF618	
LINIF619	
LINIF620	
LINIF621	
LINIF622	
LINIF623	
LINIF624	
LINIF625	
LINIF626	
LINIF627	
LINIF628	
LINIF629	
LINIF630	
LINIF631	
LINIF632	

LINIF633	
LINIF634	
LINIF635	
LINIF636	
LINIF637	
LINIF691	Rework of Published Information
LINIF638	
LINIF639	
LINIF640	
LINIF641 – 648	
LINIF649	
LINIF650	
LINIF651 – 655	
LINIF656 – 667	
LINIF668	
LINIF669	
LINIF670	
LINIF671 – 680	
LINIF681 – 686	
LINIF687	
LINIF688	
LINIF689,690	
LINIF692	

12 Not applicable requirements

[LINIF692] ⌈These requirements are not applicable to this specification.⌋ (BSW00431, BSW00432, BSW00433, BSW00434, BSW00417, BSW00359, BSW00360, BSW00331, BSW010, BSW00333, BSW003, BSW00321, BSW00341, BSW00334, BSW00437, BSW00422, BSW00440, BSW00439)