Document Title	Specification of LIN Interface
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	073
Document Status	Final
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	4.3.1

Document Change History

Date	Release	Changed by	Change Description
2017-12-08	4.3.1	AUTOSAR Release Management	• Rollout of Runtime Errors • Clarification of SRF handling for Node Configuration Request • Resolve inconsistency on channel state upon initialization • Clarification of LIN schedule table switch behavior
2016-11-30	4.3.0	AUTOSAR Release Management	• Changed the call of MainFunction_<ChannelId> of each channel • Added the new function for schedule table change • Changed the signature of User_TxConfirmation
2015-07-31	4.2.2	AUTOSAR Release Management	• Removed PostBuildTime from the configuration class of optional interfaces • Changed to call the <User_TriggerTransmit> with the buffer length • Changed to Default Error Tracer from Development Error Tracer

Document Change History			
Date	**Release**	**Changed by**	**Change Description**
2014-10-31	4.2.1	AUTOSAR Release Management	• Changed the description of return value E_NOT_OK for LinIf_Wakeup • Changed the parameter LinIfFrameRef.upperMultiplicity from '*' to '1' • Revised the typo in SWS_LinIf_00614 • Editorial changes
2014-03-31	4.1.3	AUTOSAR Release Management	• Set the parameter LinIfSlave and LinIfLength to obsolete • Changed the signature of <User_RxIndication> • Editorial changes
2013-10-31	4.1.2	AUTOSAR Release Management	• Added the parallel handling for physical and functional request of LINTP • Changed the wakeup handling by LIN bus • Removed the type NotifResultType • Editorial changes • Removed chapter(s) on change documentation
2013-03-15	4.1.1	AUTOSAR Administration	• Changed the buffer handling of the retry and failure for LinTp • Changed the reception error handling of the unexpected PDU for LinTp • Changed the wakeup operation during the transition to sleep state
2011-12-22	4.0.3	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.

Document Change History			
Date	**Release**	**Changed by**	**Change Description**
2010-09-30	3.1.5	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”
2010-02-02	3.1.4	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
2008-08-13	3.1.1	AUTOSAR Administration	• Legal disclaimer revised
2007-12-21	3.0.1	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

Document Change History			
Date	**Release**	**Changed by**	**Change Description**
2007-01-24	2.1.15	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
2006-05-16	2.0	AUTOSAR Administration	• Initial release

Disclaimer

Table of Contents

1 Introduction and functional overview

This document specifies the functionality, API and the configuration of the AUTOSAR Basic Software module LIN Interface (LinIf) and the LIN Transport Protocol (LIN TP). 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 [17]. 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 upper (PDU Router) and lower (LIN Driver) modules 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.

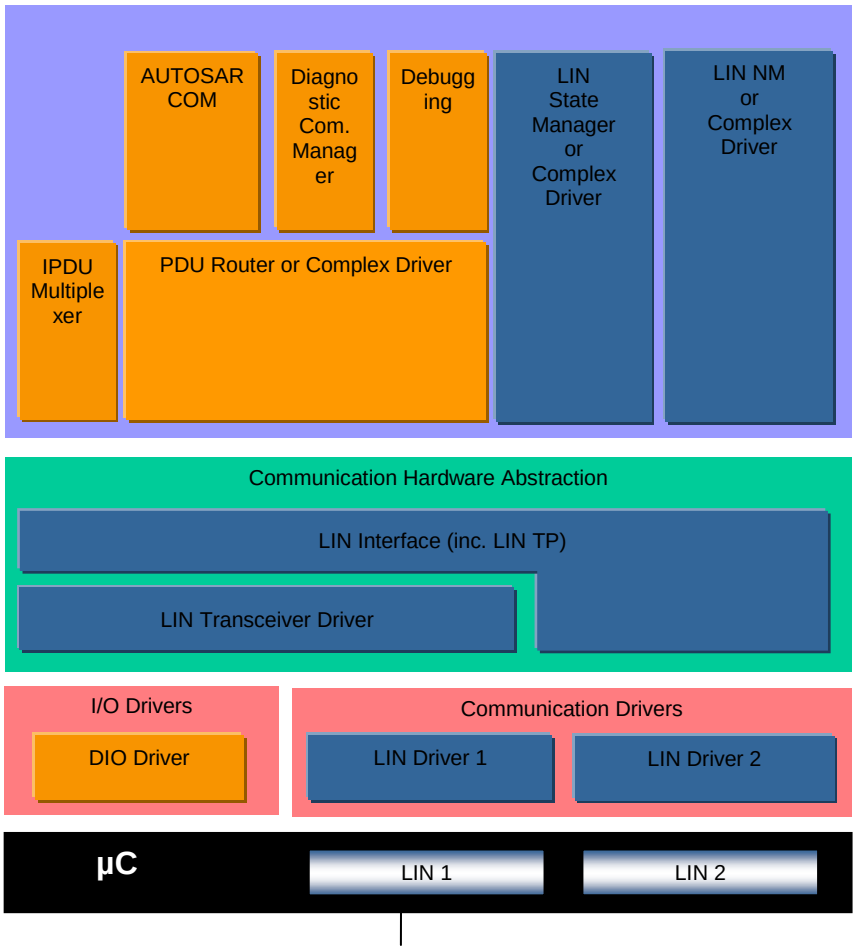

Figure 1 – AUTOSAR BSW software architecture (LIN relevant modules)

1.2 Functional overview

The LIN Interface is responsible for providing LIN 2.1 master functionality. 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 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:***
CF	Continuous Frame in LIN TP
FF	First Frame in LIN TP
ID	Identifier
LDF	LIN Description File
LIN TP	LIN Transport Protocol (Part of the LIN Interface)
MRF	Master Request Frame
NAD	Node Address. Each slave in LIN must have a unique NAD.
NC	Node Configuration
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).
P2	Time between reception of the last frame of a diagnostic request on the LIN bus and the slave node being able to provide data for a response.
P2*	Time between sending a response pending frame (0x78) and the LIN-slave being able to provide data for a response.
PID	Protected ID
RX	Reception
SF	Single Frame in LIN TP
SRF	Slave Response Frame
SRS	Software Requirement Specification
TX	Transmission

Term:	***Description:***
Slot Delay	The time between start of frames in a schedule table. The unit is in number of time-bases for the specific cluster.
Jitter	Difference between longest delay and shortest delay (e.g. Worst case execution time – Best case execution time)
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 %).
Schedule entry is due	This means that the LIN Interface has arrived at a new entry in the schedule table and a frame (received or transmitted) will be initiated.
Slave-to-slave	There exist 3 different directions of frames on the LIN bus: Response transmitted by the master, Response received by the master 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.
Tick	The tick is the smallest time entity to handle the communication on all channels.

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 Default Error Tracer
AUTOSAR_SWS_DefaultErrorTracer.pdf

[6] Requirements on LIN
AUTOSAR_SRS_LIN.pdf

[7] Specification of LIN Driver
AUTOSAR_SWS_LINDriver.pdf

[8] Specification of ECU Configuration
AUTOSAR_TPS_ECUConfiguration.pdf

[9] Specification of ECU State Manager
AUTOSAR_SWS_ECUStateManager.pdf

[10] Specification of LIN State Manager
AUTOSAR_SWS_LINStateManager.pdf

[11] Basic Software Module Description Template
AUTOSAR_TPS_BSWModuleDescriptionTemplate.pdf

[12] Specification of LIN Transceiver Driver
AUTOSAR_SWS_LINTransceiverDriver.pdf

[13] Specification of PDU Router
AUTOSAR_SWS_PDURouter.pdf

[14] Specification of Communication Stack Types
AUTOSAR_SWS_CommunicationStackTypes.pdf

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

[16] General Specification of Basic Software Modules
AUTOSAR_SWS_BSWGeneral.pdf

3.2 Related standards and norms

[17] LIN Specification Package Revision 2.1, November 24, 2006
http://www.lin-subbus.org/

[18] SAE J2602-1 (2012-11), LIN Network for Vehicle Applications

[19] ISO 17987:2016 (all parts), Road vehicles – Local Interconnect Network (LIN)

3.3 Related specification

AUTOSAR provides a General Specification on Basic Software modules [16] (SWS BSW General), which is also valid for LIN Interface.

Thus, the specification SWS BSW General shall be considered as additional and required specification for LIN Interface.

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.

It's assumed that all of connected LIN Slave ECUs can receive a wakeup frame when they are already operational (as LIN Master ECU starts with LINIF_CHANNEL_SLEEP state).

4.2 Applicability to car domains

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

4.3 Clarification about other LIN standards

J2602 [18] and ISO 17987 [19] are other standard manifestations of LIN 2.1 [17]. These alternate standards are based on the concepts of LIN 2.1.
AUTOSAR LinIf supports the above standards as far as they are identical to LIN 2.1.

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
- Default Error Tracer – DET
- ECU State Manager – EcuM
- LIN State Manager – LinSM
- BSW Mode Manager – BswM

5.1 Upper layers

5.1.1 PDU Router and CDD

The LIN Interface connects to the PDU Router and/or alternative modules above (e.g. Complex Driver) for transmission and reception of frames. It is assumed that these modules are responsible for the copying of the data of the frames for reception and transmission. In case of TP, the PDU Router is the only module above and handles the TP messages buffers either as complete or fragmented messages.

5.1.2 LIN State Manager

The LIN Interface connects to the LIN state manager 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.

5.1.3 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 and response 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 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 transmission status and reception of the response part of a frame (Lin_GetStatus). The following cases are 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).

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

5.2.2 LIN Transceiver Driver

Optionally, the LIN Interface requires the services of the underlying LIN Transceiver Driver specified by [12].

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.

The LIN Interface does 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 Header file structure

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

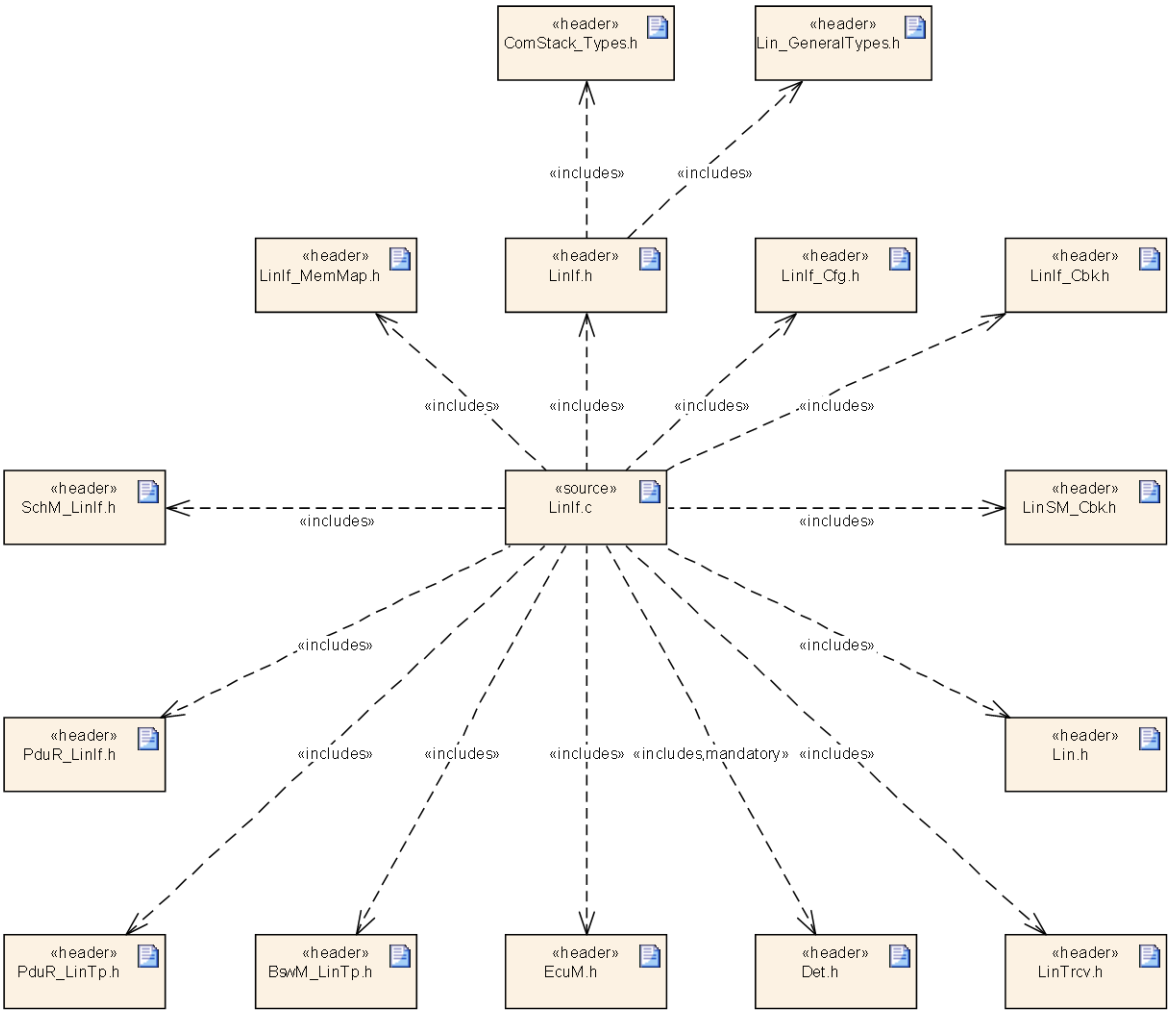

Figure 2 – Header file structure

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.

[SWS_LinIf_00434] ⌈The LIN Interface shall include the file Lin.h. ⌋ ()

[SWS_LinIf_00497] ⌈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. ⌋ ()

[SWS_LinIf_00498] ⌈The LIN Interface shall include the file Det.h. ⌋ ()

[SWS_LinIf_00499] ⌈The LIN Interface shall include the file ComStack_Types.h. ⌋()

[SWS_LinIf_00638] ⌈The LinIf.h shall include the file Lin_GeneralTypes.h for the include of general LIN type declarations. ⌋()

[SWS_LinIf_00561] ⌈The LIN Interface shall include the file PduR_LinTp.h, if the LIN TP is enabled (configuration parameter LinIfTpSupported). ⌋()

[SWS_LinIf_00555] ⌈The LIN Interface shall include the file LinTrcv.h, if the configuration parameter LinIfTrcvDriverSupported is set to TRUE. ⌋()

[SWS_LinIf_00556] ⌈The LIN Interface shall include the file LinSM_Cbk.h. ⌋()

[SWS_LinIf_00650] ⌈The LIN Interface shall include the file BswM_LinTp.h. ⌋()

[SWS_LinIf_00690] ⌈ The LIN Interface shall include the file EcuM.h. ⌋()

[SWS_LinIf_00669] ⌈The LIN Interface shall include the file <CDD_Cbk.h> for callback declaration of CDD. <CDD_Cbk.h> is configurable via configuration parameter LinIfPublicCddHeaderFile. ⌋()

[SWS_LinIf_00711] ⌈A header file LinIf_Cbk.h shall contain the function declarations for the callback functions in the LIN Interface. ⌋()

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	Description	Satisfied by
SRS_BSW_00003	All software modules shall provide version and identification information	SWS_LinIf_99999
SRS_BSW_00010	The memory consumption of all Basic SW Modules shall be documented for a defined configuration for all supported platforms.	SWS_LinIf_99999
SRS_BSW_00101	The Basic Software Module shall be able to initialize variables and hardware in a separate initialization function	SWS_LinIf_00198, SWS_LinIf_00350
SRS_BSW_00167	All AUTOSAR Basic Software Modules shall provide configuration rules and constraints to enable plausibility checks	SWS_LinIf_00375
SRS_BSW_00170	The AUTOSAR SW Components shall provide information about their dependency from faults, signal qualities, driver demands	SWS_LinIf_00373
SRS_BSW_00171	Optional functionality of a Basic-SW component that is not required in the ECU shall be configurable at pre-compile-time	SWS_LinIf_00310, SWS_LinIf_00387
SRS_BSW_00321	The version numbers of AUTOSAR Basic Software Modules shall be enumerated according specific rules	SWS_LinIf_99999
SRS_BSW_00327	Error values naming convention	SWS_LinIf_00376, SWS_LinIf_00729
SRS_BSW_00328	All AUTOSAR Basic Software Modules shall avoid the duplication of code	SWS_LinIf_00386
SRS_BSW_00331	All Basic Software Modules shall strictly separate error and status information	SWS_LinIf_99999
SRS_BSW_00333	For each callback function it shall be specified if it is called from interrupt context or not	SWS_LinIf_99999
SRS_BSW_00334	All Basic Software Modules shall provide an XML file that contains the meta data	SWS_LinIf_99999
SRS_BSW_00335	Status values naming convention	SWS_LinIf_00316, SWS_LinIf_00319, SWS_LinIf_00438, SWS_LinIf_00439, SWS_LinIf_00441, SWS_LinIf_00442
SRS_BSW_00336	Basic SW module shall be able to shutdown	SWS_LinIf_00355
SRS_BSW_00337	Classification of development errors	SWS_LinIf_00376
SRS_BSW_00341	Module documentation shall contains all needed informations	SWS_LinIf_99999
SRS_BSW_00344	BSW Modules shall support link-time configuration	SWS_LinIf_00373

SRS_BSW_00358	The return type of init() functions implemented by AUTOSAR Basic Software Modules shall be void	SWS_LinIf_00198, SWS_LinIf_00350
SRS_BSW_00359	All AUTOSAR Basic Software Modules callback functions shall avoid return types other than void if possible	SWS_LinIf_99999
SRS_BSW_00360	AUTOSAR Basic Software Modules callback functions are allowed to have parameters	SWS_LinIf_99999
SRS_BSW_00373	The main processing function of each AUTOSAR Basic Software Module shall be named according the defined convention	SWS_LinIf_00384
SRS_BSW_00375	Basic Software Modules shall report wake-up reasons	SWS_LinIf_00378
SRS_BSW_00385	List possible error notifications	SWS_LinIf_00376, SWS_LinIf_00729
SRS_BSW_00404	BSW Modules shall support post-build configuration	SWS_LinIf_00373
SRS_BSW_00405	BSW Modules shall support multiple configuration sets	SWS_LinIf_00373
SRS_BSW_00406	A static status variable denoting if a BSW module is initialized shall be initialized with value 0 before any APIs of the BSW module is called	SWS_LinIf_00376, SWS_LinIf_00535, SWS_LinIf_00687
SRS_BSW_00407	Each BSW module shall provide a function to read out the version information of a dedicated module implementation	SWS_LinIf_00340, SWS_LinIf_00352
SRS_BSW_00413	An index-based accessing of the instances of BSW modules shall be done	SWS_LinIf_00197, SWS_LinIf_00469
SRS_BSW_00414	Init functions shall have a pointer to a configuration structure as single parameter	SWS_LinIf_00198, SWS_LinIf_00350
SRS_BSW_00416	The sequence of modules to be initialized shall be configurable	SWS_LinIf_00198, SWS_LinIf_00350
SRS_BSW_00417	Software which is not part of the SW-C shall report error events only after the DEM is fully operational.	SWS_LinIf_99999
SRS_BSW_00422	Pre-de-bouncing of error status information is done within the DEM	SWS_LinIf_99999
SRS_BSW_00425	The BSW module description template shall provide means to model the defined trigger conditions of schedulable objects	SWS_LinIf_00248
SRS_BSW_00432	Modules should have separate main processing functions for read/receive and write/transmit data path	SWS_LinIf_99999
SRS_BSW_00433	Main processing functions are only allowed to be called from task bodies provided by the BSW Scheduler	SWS_LinIf_99999
SRS_BSW_00437	Memory mapping shall provide the	SWS_LinIf_99999

	possibility to define RAM segments which are not to be initialized during startup	
SRS_BSW_00439	Enable BSW modules to handle interrupts	SWS_LinIf_99999
SRS_BSW_00440	The callback function invocation by the BSW module shall follow the signature provided by RTE to invoke servers via Rte_Call API	SWS_LinIf_99999
SRS_BSW_00452	Classification of runtime errors	SWS_LinIf_00729
SRS_Lin_01502	The LIN Interface shall support an API for RX/TX notifications.	SWS_LinIf_00033, SWS_LinIf_00128
SRS_Lin_01504	The usage of AUTOSAR architecture shall be mandatory only in LIN master nodes	SWS_LinIf_00248
SRS_Lin_01514	The LIN Interface shall inform an upper layer about wake-up events	SWS_LinIf_00378
SRS_Lin_01515	The LIN Interface shall provide an API to wake-up a LIN channel cluster	SWS_LinIf_00205
SRS_Lin_01523	There shall be an API call to send the LIN bus to sleep-mode.	SWS_LinIf_00204
SRS_Lin_01534	The AUTOSAR LIN Transport Layer shall support half-duplex physical connections.	SWS_LinIf_00062
SRS_Lin_01540	The LIN Transport Layer shall provide an API for initialization.	SWS_LinIf_00350
SRS_Lin_01544	Errors shall be handled	SWS_LinIf_00079, SWS_LinIf_00651
SRS_Lin_01546	The LIN Interface shall contain a Schedule Table Handler.	SWS_LinIf_00028, SWS_LinIf_00384, SWS_LinIf_00393
SRS_Lin_01551	One LIN Interface shall support one or more LIN Drivers.	SWS_LinIf_00386
SRS_Lin_01555	The LIN driver shall have an API which the driver shall use to poll for transmit / receive notifications.	SWS_LinIf_00384
SRS_Lin_01558	The LIN Interface shall check for successful data transfer	SWS_LinIf_00033, SWS_LinIf_00128
SRS_Lin_01560	If a wakeup occurs during transition to sleep-mode, this channel shall go back to the running mode	SWS_LinIf_00459
SRS_Lin_01561	The LIN Interface shall define a main function	SWS_LinIf_00384
SRS_Lin_01564	A Schedule Table Manager shall be available	SWS_LinIf_00202, SWS_LinIf_00495
SRS_Lin_01569	The LIN Interface shall support initialization of each LIN channel separately	SWS_LinIf_00198
SRS_Lin_01571	Transmission request service shall be provided	SWS_LinIf_00201
SRS_Lin_01574	It shall be possible to have one instance of the TP for each channel	SWS_LinIf_00314

SRS_Lin_01576	The LIN 2.1 specification shall be reused as far as possible	SWS_LinIf_00248
SRS_Lin_01577	It shall be compatible to LIN protocol specification	SWS_LinIf_00248
SRS_Lin_01579	The AUTOSAR LIN Transport Layer shall be based on the Diagnostic Transport Layer for LIN 2.1.	SWS_LinIf_00313
SRS_Lin_01584	The bus transceiver driver shall support an API to send the addressed transceiver into its Standby mode.	SWS_LinIf_00544
SRS_Lin_01585	The bus transceiver driver shall support an API to send the addressed transceiver into its Sleep mode.	SWS_LinIf_00544
SRS_Lin_01586	The bus transceiver driver shall support an API to send the addressed transceiver into its Normal mode.	SWS_LinIf_00544
SRS_Lin_01587	The LIN Transceiver Driver shall support an API to read out the current operation mode.	SWS_LinIf_00545
SRS_Lin_01588	The LIN Transceiver Driver shall support an API to read out the the reason of the last wakeup.	SWS_LinIf_00547
SRS_Lin_01589	The bus transceiver driver shall support an API to enable and disable the wakeup notification for each bus separately.	SWS_LinIf_00550
SRS_Lin_01590	The node configuration of LIN slaves shall only be done via defined schedule table(s).	SWS_LinIf_00401
SRS_Lin_01592	The AUTOSAR LIN Transport Layer shall support the transmission of functional requests at any time.	SWS_LinIf_00062

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 these base requirements.

[SWS_LinIf_00248] ⌈The LIN Interface shall support the behavior of the master in the LIN 2.1 specification. ⌋(SRS_BSW_00425, SRS_Lin_01576, SRS_Lin_01504, SRS_Lin_01577)

The requirement above basically means that the communication from a LIN 2.1 master and the LIN Interface master will be equal.

[SWS_LinIf_00249] ⌈The LIN Interface shall realize the master behavior so that existing slaves can be reused. ⌋()

[SWS_LinIf_00386] ⌈The LIN Interface shall be able to handle one or more LIN channels. ⌋(SRS_BSW_00328, SRS_Lin_01551)

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.

[SWS_LinIf_00588] ⌈If a collision occurs in an event-triggered frame response, then the LIN Interface shall switch to the corresponding collision resolving schedule table. ⌋()

[SWS_LinIf_00176] ⌈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. ⌋()

[SWS_LinIf_00519] ⌈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.

[SWS_LinIf_00012] ⌈The master shall be the only allowed transmitter of a sporadic frame (defined in the LIN 2.1 specification). ⌋()

[SWS_LinIf_00436] ⌈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.

[SWS_LinIf_00470] ⌈The LIN Interface shall flag the specific sporadic frame (defined in the LIN 2.1 specification) for transfer.⌋()

[SWS_LinIf_00471] ⌈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.

[SWS_LinIf_00014] ⌈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:

[SWS_LinIf_00066] ⌈The LIN Interface shall send an MRF if there is an ongoing TP transmission, when a schedule entry is due, 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.

[SWS_LinIf_00023] ⌈The LIN Interface shall always send an SRF header when a 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.

[SWS_LinIf_00472] ⌈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 an SRF can be silent or pending for transportation).

7.1.2.1 Header

[SWS_LinIf_00419] ⌈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

[SWS_LinIf_00030] ⌈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_<ChannelId> 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_<ChannelId> function call (if the current LinIf_MainFunction_<ChannelId> 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.

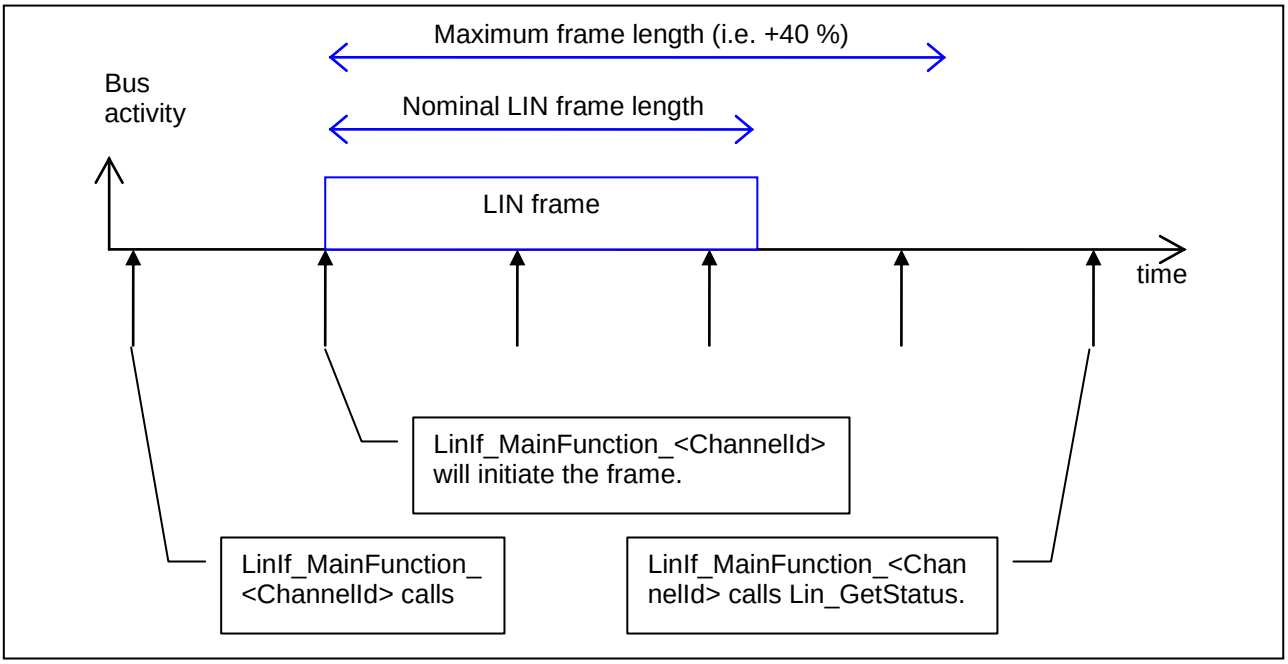

Figure 3 – Lin_GetStatus call example

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:

[SWS_LinIf_00033] ⌈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. ⌋ (SRS_Lin_01502, SRS_Lin_01558)

[SWS_LinIf_00259] ⌈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 that a collision may occur, which is handled as described in chapter 7.1.1.2. The following shall apply, if none of the slave reply on the event-triggered frame header.

[SWS_LinIf_00258] ⌈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. ⌋ ()

[SWS_LinIf_00254] ⌈When the LIN Interface has determined the LIN Driver module's status as LIN_RX_BUSY or LIN_RX_ERROR, the LIN Interface shall consider the received frame as lost. Therefore, the LIN Interface shall report the runtime error code LINIF_E_RESPONSE to the Default Error Tracer, if this frame is an unconditional frame. ⌋ ()

[SWS_LinIf_00466] ⌈When the LIN Interface has determined the LIN Driver module's status as LIN_RX_NO_RESPONSE, the LIN Interface shall consider the

expected frame as lost. Therefore, the LIN Interface shall report the runtime error code LINIF_E_RESPONSE to the Default Error Tracer, if this frame is an unconditional frame. ⌋()

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

[SWS_LinIf_00458] ⌈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_<ChannelId> when a new schedule entry is due.

The requirements in this chapter are applicable to all 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 and response

[SWS_LinIf_00225] ⌈ The LIN Interface shall call the function <User_TriggerTransmit> with the PduInfoPtr pointer containing data buffer (SduDataPtr) and buffer length (SduLength) to get the data part of the frame (data in the LIN frame response) when a schedule entry for a frame transmission is due. ⌋()

[SWS_LinIf_00226] ⌈After getting the data part of the frame (when the function <User_TriggerTransmit> returns E_OK), the LIN Interface shall call the LIN Driver module's function Lin_SendFrame to provide the LIN Driver a pointer to the data part. ⌋()

[SWS_LinIf_00706] ⌈When the function <User_TriggerTransmit> returns E_NOT_OK, the LIN Interface shall not transmit the sporadic or unconditional frame for which the data was requested. ⌋()

7.1.3.2 Status check

[SWS_LinIf_00128] ⌈If the return code of the function Lin_GetStatus is LIN_TX_OK, the LIN Interface shall issue a <User_TxConfirmation> callback with result E_OK. ⌋ (SRS_Lin_01502, SRS_Lin_01558)

[SWS_LinIf_00728] ⌈If the return code of the function Lin_GetStatus is LIN_TX_ERROR or LIN_TX_BUSY, the LIN Interface shall issue a <User_TxConfirmation> callback with result E_NOT_OK.⌋()

[SWS_LinIf_00036] ⌈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 and report the runtime error code LINIF_E_RESPONSE to the Default Error Tracer.⌋()

[SWS_LinIf_00465] ⌈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 report the runtime error code LINIF_E_RESPONSE to the Default Error Tracer.⌋()

[SWS_LinIf_00463] ⌈If the LIN Interface has transmitted a sporadic frame successfully, it shall reset the pending flag.⌋()

Note that sporadic frames should not be used in combination with a PduR FiFo (PduRTxBufferDepth > 1).

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

[SWS_LinIf_00416] ⌈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

[SWS_LinIf_00417] ⌈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

[SWS_LinIf_00418] ⌈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.

[SWS_LinIf_00261] ⌈The delay between processing two frames shall be a multiple of a period which is given by configuration parameter LinIfMainFunctionPeriod.⌋()

[SWS_LinIf_00231] ⌈The LIN Interface shall provide a predefined schedule table per channel (named NULL_SCHEDULE).⌋()

[SWS_LinIf_00263] ⌈The schedule table NULL_SCHEDULE shall contain no entries.⌋()

7.2.1 LinIf_MainFunction

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

The SchM will call the function LinIf_MainFunction_<ChannelId> periodically with a period which is given by the configuration parameter LinIfMainFunctionPeriod.

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 frame transmission and reception occurs.

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

[SWS_LinIf_00727] ⌈The point in time where a schedule table switch is performed depends on the optional configuration parameter LinIfScheduleChangeNextTimeBase. If LinIfScheduleChangeNextTimeBase is disabled or absent, the schedule table shall be switched after the current entry of the active schedule table is ended. If LinIfScheduleChangeNextTimeBase is enabled, the schedule table shall be switched when message transmission or reception within an entry has been completed, ensured by status checks for transmission and reception. ⌋()

Note: The conditions under which schedule table switches can take place are given by SWS_LinIf_00176, SWS_LinIf_00293, SWS_LinIf_00393, SWS_LinIf_00588, SWS_LinIf_00617, SWS_LinIf_00656, SWS_LinIf_00660, and SWS_LinIf_00664.

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

[SWS_LinIf_00444] ⌈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 at the next possible time (even if the current is RUN_ONCE). ⌋()

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

[SWS_LinIf_00028] ⌈The LIN Interface shall start the newly requested schedule table at the next possible time (e.g. at start of a frame slot) if the current schedule is RUN_CONTINUOUS. ⌋(SRS_Lin_01546)
Note: It is possible to request the same schedule table again. In this case, the table is restarted.

[SWS_LinIf_00393] ⌈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. But, if a collision occurs in an event-triggered frame response, the LIN Interface shall switch to a collision resolving schedule table according to **SWS_LinIf_00176**. ⌋(SRS_Lin_01546)

[SWS_LinIf_00495] ⌈If the switch to a requested schedule table has been performed, the schedule table manager shall call the function <User>_ScheduleRequestConfirmation. ⌋(SRS_Lin_01564)

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

[SWS_LinIf_00029] ⌈The state of sporadic frames shall not be cleared when the schedule table is changed.⌋()

[SWS_LinIf_00397] ⌈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.⌋()

[SWS_LinIf_00485] ⌈The definition where the execution of a RUN_CONTINUOUS 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.

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

[SWS_LinIf_00039] ⌈The LIN Interface shall have one state-machine.

The state-machine is depicted in Figure 4.⌋()

[SWS_LinIf_00438] ⌈The LIN Interface state-machine shall have the state LINIF_UNINIT.⌋(SRS_BSW_00335)

[SWS_LinIf_00439] ⌈The LIN Interface state-machine shall have the state LINIF_INIT.⌋(SRS_BSW_00335)

[SWS_LinIf_00381] ⌈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.

[SWS_LinIf_00290] ⌈Each LIN channel shall have a separate channel state-machine. ⌋ ()

[SWS_LinIf_00441] ⌈The LIN channel sub-state-machine shall have the state LINIF_CHANNEL_OPERATIONAL. ⌋ (SRS_BSW_00335)

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

[SWS_LinIf_00189] ⌈The LIN Interface shall transmit LIN frame headers and receive/transmit responses only when the corresponding LIN channel is in the state LINIF_CHANNEL_OPERATIONAL. ⌋ ()

[SWS_LinIf_00053] ⌈In the state LINIF_CHANNEL_OPERATIONAL, the LIN Interface shall process the currently selected schedule table within the function LinIf_MainFunction_<ChannelId>. ⌋ ()

[SWS_LinIf_00507] ⌈The LIN Interface shall transit from LINIF_UNINIT to LINIF_CHANNEL_SLEEP without sending go-to-sleep command, when the function LinIf_Init is called. ⌋ ()

Note: it is assumed that automatically slave nodes will enter bus sleep mode earliest after 4s and latest 10s of bus inactivity (as specified in LIN 2.1).

[SWS_LinIf_00442] ⌈The LIN channel sub-state-machine shall have the state LINIF_CHANNEL_SLEEP. ⌋ (SRS_BSW_00335)

[SWS_LinIf_00478] ⌈The LIN Interface shall transit from the channel state LINIF_CHANNEL_SLEEP to LINIF_CHANNEL_OPERATIONAL when it has detected a wake-up request for the corresponding channel. ⌋ ()

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

[SWS_LinIf_00043] ⌈When a channel is in the LIN channel state LINIF_CHANNEL_SLEEP, the function LinIf_MainFunction_<ChannelId> shall not initiate any traffic on the bus for the corresponding LIN channel.⌋()

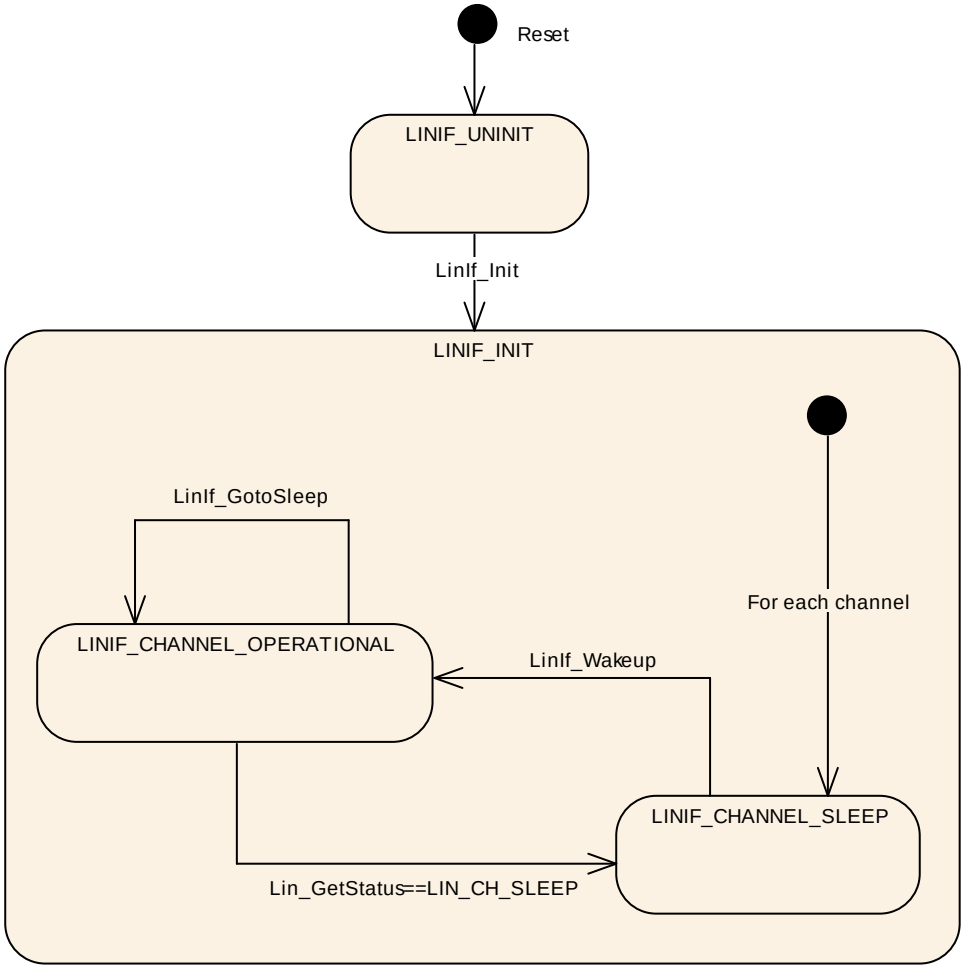

Figure 4 – LIN Interface state-machine and LIN Interface channel sub-state-machine

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.

[SWS_LinIf_00453] ⌈When processing the go-to-sleep command and the channel is not in the state LINIF_CHANNEL_SLEEP, the function LinIf_MainFunction_<ChannelId> shall call the function Lin_GoToSleep instead of the scheduled frame latest when the next schedule entry is due.⌋()

[SWS_LinIf_00597] ⌈When processing the go-to-sleep command and the channel is in the state LINIF_CHANNEL_SLEEP, the function LinIf_MainFunction_<ChannelId>

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_<ChannelId> 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.

[SWS_LinIf_00712] ⌈When the function Lin_GoToSleep or Lin_GotoSleepInternal is called, the function LinIf_MainFunction_<ChannelId> shall clear the wakeup flag of selected channel. (see **SWS_LinIf_00716**) ⌋()

[SWS_LinIf_00455] ⌈When processing the go-to-sleep command, the function LinIf_MainFunction_<ChannelId> 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_<ChannelId> shall set the channel state of the affected channel to LINIF_CHANNEL_SLEEP. In this case, the go-to-sleep command transmission has successfully been performed. ⌋()

[SWS_LinIf_00454] ⌈When processing the go-to-sleep command, the function LinIf_MainFunction_<ChannelId> 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. ⌋()

[SWS_LinIf_00557] ⌈When the go-to-sleep command was sent successful or the function Lin_GoToSleepInternal was called, the LIN Interface shall invoke the function <User>_GotoSleepConfirmation with the parameter TRUE. ⌋()

[SWS_LinIf_00558] ⌈When the go-to-sleep command was not sent successful, the LIN Interface shall invoke the function <User>_GotoSleepConfirmation with the parameter FALSE. ⌋()

[SWS_LinIf_00293] ⌈When entering the LINIF_CHANNEL_SLEEP state during the go-to-sleep command process, the function LinIf_MainFunction_<ChannelId> shall switch the current used schedule table to the NULL_SCHEDULE. ⌋()

7.3.3 Wake up process

There are different possibilities to wake-up a LIN channel. Either the upper layer requests a wake-up through the LinIf_Wakeup call or a bus wake-up is detected. If a bus wake-up is detected, LinIf_Wakeup is also called when the upper layer enters

the FULL_COM mode after a successful validation through the function LinIf_CheckWakeup.

[SWS_LinIf_00496] ⌈When the return code of the function LinIf_Wakeup is E_OK, the LIN Interface shall issue the function <User>_WakeupConfirmation with the parameter TRUE. ⌋ ()

[SWS_LinIf_00670] ⌈When the return code of the function LinIf_Wakeup is E_NOT_OK, the LIN Interface shall issue the function <User>_WakeupConfirmation with the parameter FALSE. ⌋ ()

7.3.3.1 Wakeup during sleep transition

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

[SWS_LinIf_00459] ⌈If the go-to-sleep command is requested and the upper layer requests a wake-up before the go-to-sleep command is executed, the LIN Interface shall neither send the pending go-to-sleep command nor a wake-up on the bus and shall maintain the LIN channel state LINIF_CHANNEL_OPERATIONAL. ⌋ (SRS_Lin_01560)

[SWS_LinIf_00460] ⌈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. ⌋ ()

[SWS_LinIf_00699] ⌈In case of **SWS_LinIf_00460**, LIN Interface shall not invoke the function <User>_GotoSleepConfirmation. ⌋ ()

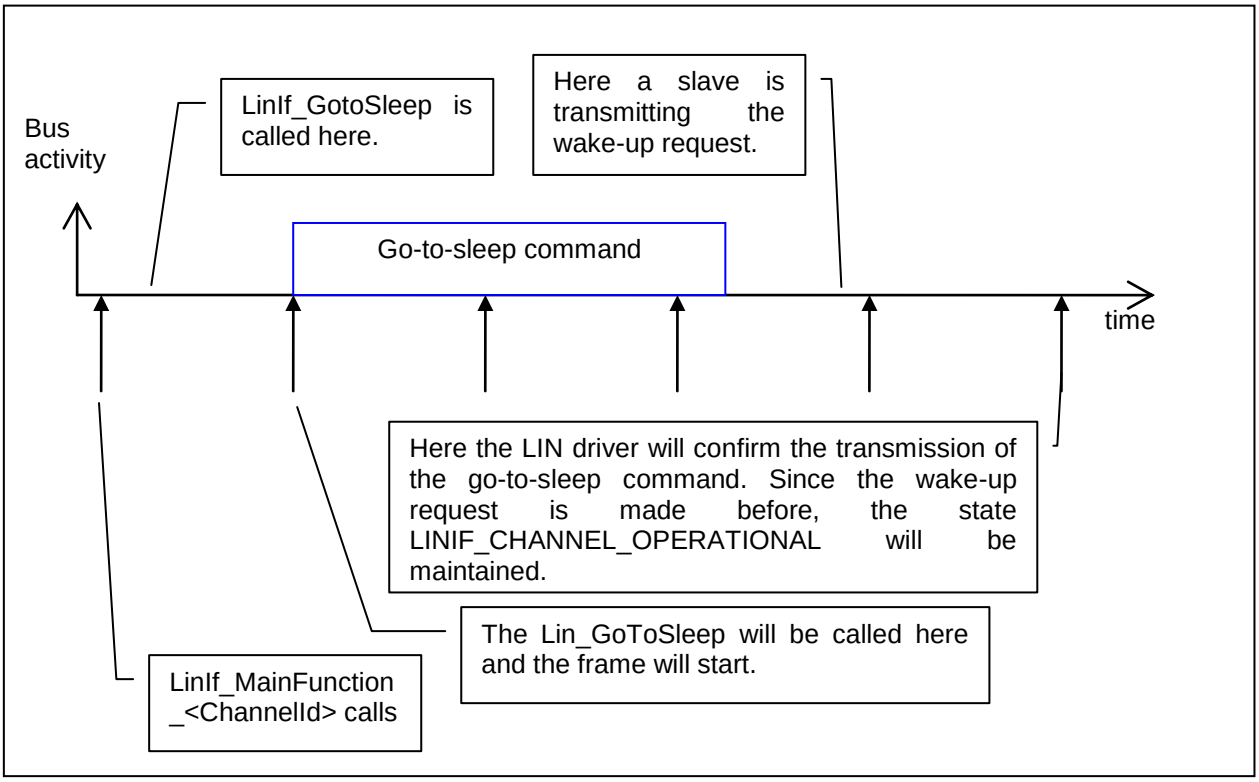

Figure 5 – Wake up requested before confirmation of go-to-sleep command

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 7.1.3.2.

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.

[SWS_LinIf_00401] ⌈The LIN Interface shall only do the Node Configuration (defined in the LIN 2.1 specification) by using services directly in the Schedule Table. ⌋ (SRS_Lin_01590)

7.5.1.1 Node Model

The LIN 2.1 specification defines a Node Model that describes where the configuration is stored.

[SWS_LinIf_00308] ⌈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.

[SWS_LinIf_00309] ⌈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). ⌋ ()

[SWS_LinIf_00409] ⌈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.

[SWS_LinIf_00310] ⌈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. ⌋(SRS_BSW_00171)

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

[SWS_LinIf_00408] ⌈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. It is the responsibility of the diagnostic layer to support the function Read-by-Identifier.

[SWS_LinIf_00090] ⌈The LIN Interface shall not support the function Read-by-Identifier (defined in the LIN 2.1 specification). ⌋()

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.

[SWS_LinIf_00479] ⌈The LIN Interface shall process the fixed MRF entries without the interaction with an upper layer. ⌋()

[SWS_LinIf_00709] ⌈The LIN Interface shall not send the SRF header when the transmission of a fixed MRF failed. ⌋()

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.

[SWS_LinIf_00404] ⌈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:

[SWS_LinIf_00405] ⌈The LIN Interface shall report the runtime error code LINIF_E_NC_NO_RESPONSE to the Default Error Tracer, if it has put a SRF in the schedule table after a node configuration command and if there's no response from any slaves (timed out). The error shall always be reported, even if the previous configuration command was not transmitted successfully.⌋()

Note: The LIN Interface will not report the runtime error code LINIF_E_NC_NO_RESPONSE, if there's any slave response (regardless of its contents, e.g. RSID).

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.

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.

[SWS_LinIf_00313] ⌈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.⌋(SRS_Lin_01579)

The support of the LIN TP shall be configurable on/off to make the LIN Interface smaller when LIN TP is not used.

[SWS_LinIf_00387] ⌈The support for the LIN TP shall be pre-compile time configurable by the configuration parameter LinIfTpSupported.⌋(SRS_BSW_00171)

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

[SWS_LinIf_00314] ⌈The LIN Interface shall support the transfer of a LIN TP message on each separate channel and they shall be independent of each other. ⌋ (SRS_Lin_01574)

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 service requests and responses. Functional diagnostic requests are possible in parallel to physical requests or responses.

[SWS_LinIf_00062] ⌈LIN Interface shall support physical (only half-duplex) and functional TP connections on one channel at the same time, while only one physical TP connection can be active at a time. ⌋ (SRS_Lin_01534, SRS_Lin_01592)

[SWS_LinIf_00646] ⌈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. ⌋ ()

[SWS_LinIf_00641] ⌈When the transmission of a physical or functional request 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. ⌋ ()

Note that the P2 timer is not restarted for the transmission of a functional request.

[SWS_LinIf_00642] ⌈When the transmission of physical request 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. ⌋ ()

[SWS_LinIf_00643] ⌈When the transmission of physical 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. ⌋ ()

[SWS_LinIf_00707] ⌈When the transmission of functional request is completed, the LIN Interface shall request a schedule table change to the previous schedule (applicative, diagnostic request or diagnostic response schedule) by calling the function BswM_LinTp_RequestMode. ⌋ ()

This ensures that the interrupted transmission or reception of a physical TP message is continued afterwards.

[SWS_LinIf_00708] ⌈If the transmission for a further physical request is triggered by the function LinTp_Transmit while the LIN Interface waits for a physical response or receives a physical response, LIN Interface shall terminate the current TP handling (reception, N_Cr timeout supervision or P2 timeout supervision) and accept the new physical request. ⌋()

7.5.2.1 State-machine

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

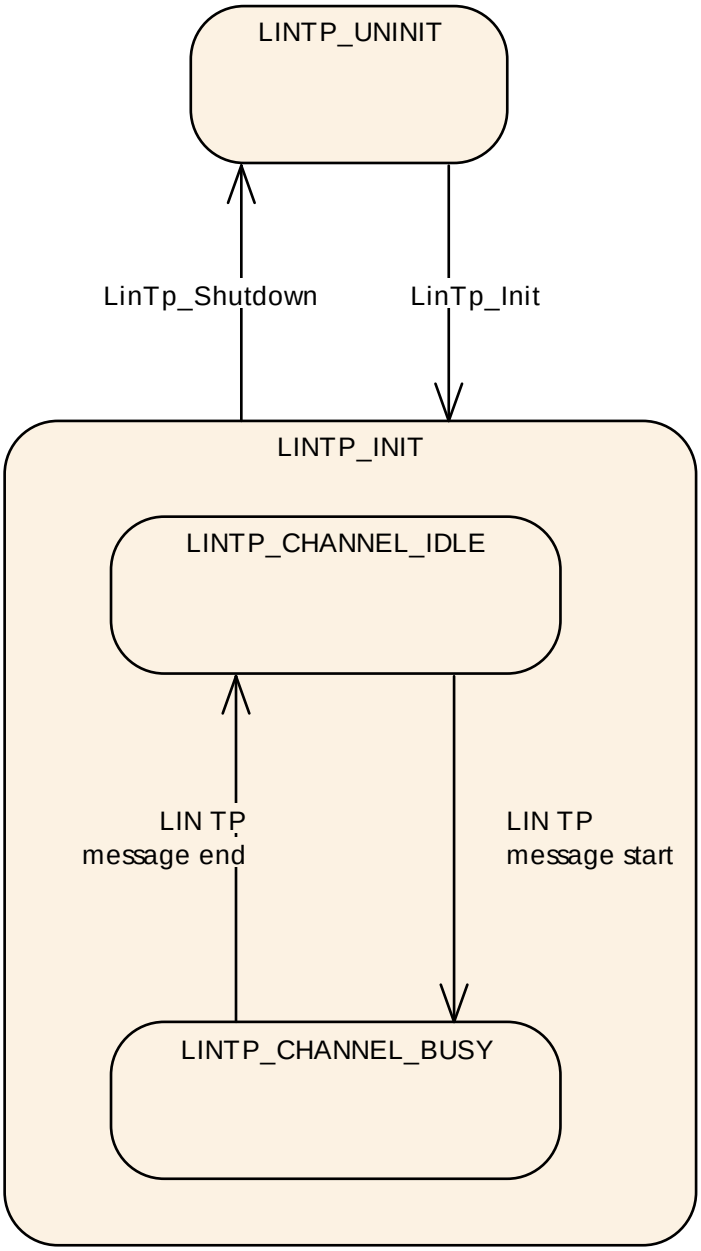

Figure 6 – LIN Transport Protocol state-machine

[SWS_LinIf_00315] ⌈Each channel of the LIN Interface shall have one instance of the LIN TP state-machine which is called LIN TP channel state-machine.⌋()

[SWS_LinIf_00316] ⌈The LIN TP state-machine shall have the state LINTP_UNINIT.⌋(SRS_BSW_00335)

[SWS_LinIf_00483] ⌈The LIN Interface shall set the LIN TP state to LINTP_UNINIT for all corresponding channels after a reset.⌋()

[SWS_LinIf_00319] ⌈The LIN TP state-machine shall have the state LINTP_INIT. ⌋ (SRS_BSW_00335)

[SWS_LinIf_00412] ⌈The LIN TP state-machine shall have the sub-state-machines of the state LINTP_INIT for each channel, that track the state of channel separately. ⌋ ()

[SWS_LinIf_00450] ⌈The sub-state-machine of the state LINTP_INIT shall have the state LINTP_CHANNEL_IDLE. ⌋ ()

[SWS_LinIf_00710] ⌈The LIN Interface shall set the sub-state of a channel to LINTP_CHANNEL_IDLE when the LIN TP state-machine is set to the state LINTP_INIT. ⌋ ()

[SWS_LinIf_00321] ⌈The LIN Interface shall start only a transmission of a TP message if the channel is in the sub-state LINTP_CHANNEL_IDLE. ⌋ ()

[SWS_LinIf_00322] ⌈The sub-state-machine of the state LINTP_INIT shall have the state LINTP_CHANNEL_BUSY. ⌋ ()

[SWS_LinIf_00323] ⌈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). ⌋ ()

[SWS_LinIf_00414] ⌈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. ⌋ ()

[SWS_LinIf_00688] ⌈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. ⌋ ()

7.5.2.2 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.

[SWS_LinIf_00671] ⌈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. 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 copies the transmit data to the PduInfo.

[SWS_LinIf_00329] ⌈If the function PduR_LinTpCopyTxData returns BUFREQ_E_BUSY, the LIN Interface shall not send the next MRF. ⌋()

[SWS_LinIf_00330] ⌈If the function PduR_LinTpCopyTxData returns BUFREQ_E_BUSY, the LIN Interface shall retry to copy the data via the function PduR_LinTpCopyTxData again during the next processing of the MainFunction until the transmit data is provided. For the number of retries, refer to the configuration parameter LinTpMaxBufReq. ⌋()

[SWS_LinIf_00672] ⌈When the function PduR_LinTpCopyTxData returns BUFREQ_OK, the LIN Interface shall resume the transmission of the MRF. ⌋()

[SWS_LinIf_00068] ⌈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 E_OK. ⌋()

The LIN Interface does not support retransmission of corrupted data.

[SWS_LinIf_00705] ⌈When calling PduR_LinTpCopyTxData, the LIN Interface shall always set the parameter retry to NULL. ⌋()

7.5.2.3 LIN TP transmission error

[SWS_LinIf_00069] ⌈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 E_NOT_OK. ⌋()

[SWS_LinIf_00073] ⌈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 E_NOT_OK. ⌋()

[SWS_LinIf_00673] ⌈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 **SWS_LinIf_00646**). ⌋()

[SWS_LinIf_00656] ⌈The LIN Interface shall provide the N_As timeout observation (configuration parameter LinTpNas) in order to switch a schedule table from

diagnostic schedule to applicative schedule in case the transmission for MRF is not successful.」()

[SWS_LinIf_00657] ⌈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.」()

[SWS_LinIf_00658] ⌈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 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 **SWS_LinIf_00646**).」()

[SWS_LinIf_00660] ⌈The LIN Interface shall provide the N_Cs timeout observation (configuration parameter LinTpNcs) in order to switch a schedule table from diagnostic schedule to applicative schedule in case the transmission for MRF is not successful. (LIN 2.1 specification defines the following requirement: (N_Cs+N_As) < 0.9*N_Cr timeout)」()

[SWS_LinIf_00661] ⌈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).」()

[SWS_LinIf_00662] ⌈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 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 **SWS_LinIf_00646**).」()

7.5.2.4 LIN TP reception

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

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

[SWS_LinIf_00075] ⌈The LIN Interface shall call the function PduR_LinTpStartOfReception with a PduInfo pointer and TpSduLength when the start of a SRF is indicated by the reception of a FF or a SF. PduInfo is pointer to the buffer containing the received data (SduDataPtr) and data length (SduLength). The

data length (including SID) is 5 bytes for FF and up to 6 bytes for SF. TpSduLength is the total length of the Sdu. ⌋()

The output pointer parameter provides the LIN Interface with currently available receive buffer size.

[SWS_LinIf_00076] ⌈The LIN Interface shall convert the NAD from the transmitting LIN slave to an N-SDU Id that the upper layer understands. ⌋()

[SWS_LinIf_00674] ⌈ After reception of each SRF (SF, FF and CF), the LIN Interface shall call the function PduR_LinTpCopyRxData with a PduInfo pointer containing received data (SduDataPtr) and data length (SduLength). 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.

[SWS_LinIf_00078] ⌈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 E_OK. ⌋()

7.5.2.5 Unavailability of receive buffer

The function PduR_LinTpStartOfReception and PduR_LinTpCopyRxData may indicate that the required buffer is not available.

The LIN Interface handles this case differently.

[SWS_LinIf_00676] ⌈If the function PduR_LinTpStartOfReception returns BUFREQ_E_NOT_OK or BUFREQ_E_OVFL, the LIN Interface shall abort the reception without any further calls to PduR. ⌋()

[SWS_LinIf_00701] ⌈If the function PduR_LinTpStartOfReception returns BUFREQ_OK with a smaller available buffer size than needed for the data received in the first SRF (SF or FF), the LIN Interface shall abort the reception and notify the upper layer by calling the function PduR_LinTpRxIndication with result E_NOT_OK. ⌋()

[SWS_LinIf_00677] ⌈If the function PduR_LinTpCopyRxData returns BUFREQ_E_NOT_OK, the LIN Interface shall abort the reception and notify the upper layer by calling the function PduR_LinTpRxIndication with the result 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 **SWS_LinIf_00646**). ⌋()

[SWS_LinIf_00679] ⌈If the function PduR_LinTpCopyRxData returns BUFREQ_OK with a smaller available buffer size than needed for the next CF, the LIN Interface shall suspend the transmission of LIN headers for next SRF (CF) ⌋()

[SWS_LinIf_00086] ⌈In case of **SWS_LinIf_00679**, the LIN Interface shall call the function PduR_LinTpCopyRxData with a data length (SduLength) 0 (zero) again during the next processing of the MainFunction until the available buffer size is big enough. ⌋()

[SWS_LinIf_00680] ⌈In case of **SWS_LinIf_00086**, when the buffer of sufficient size is available, the LIN Interface shall copy the received data via the function PduR_LinTpCopyRxData and resume the transmission of LIN headers for SRF (CF). ⌋()

7.5.2.6 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.

[SWS_LinIf_00079] ⌈In case an incorrect sequence number is received, the LIN Interface shall stop the current LIN TP message reception. ⌋(SRS_Lin_01544)

[SWS_LinIf_00081] ⌈In case an incorrect sequence number is received, the LIN Interface shall report this failure to PDU Router by calling the function PduR_LinTpRxIndication with the result E_NOT_OK. ⌋()

[SWS_LinIf_00651] ⌈In case a FF or a SF is received after a CF, the LIN Interface shall stop the current LIN TP message reception. ⌋(SRS_Lin_01544)

[SWS_LinIf_00653] ⌈In case a FF or a SF is received after a CF, the LIN Interface shall report this failure to PDU Router by calling the function PduR_LinTpRxIndication with the result E_NOT_OK. ⌋()

[SWS_LinIf_00696] ⌈In case a CF is received instead of a FF or a SF, the LIN Interface shall ignore this LIN frame. ⌋()

[SWS_LinIf_00697] ⌈In case an unknown PCI type is received, the LIN Interface shall ignore this LIN frame. ⌋()

[SWS_LinIf_00652] ⌈In case an invalid data length is received (a SF with a length of 0 (zero) or greater than 6, a FF with a length of less than 7), the LIN Interface shall ignore the LIN TP message. ⌋ ()

[SWS_LinIf_00612] ⌈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. ⌋ ()

[SWS_LinIf_00613] ⌈In case an incorrect NAD is received and the configuration parameter LinTpDropNotRequestedNad is TRUE, the LIN Interface shall stop the current LIN TP message reception. ⌋ ()

[SWS_LinIf_00655] ⌈In case an incorrect NAD is received and the configuration parameter LinTpDropNotRequestedNad is TRUE, the LIN Interface shall report this failure to PDU Router by calling the function PduR_LinTpRxIndication with the result E_NOT_OK. ⌋ ()

[SWS_LinIf_00648] ⌈In case an incorrect NAD is received and the configuration parameter LinTpDropNotRequestedNad is FALSE, the LIN Interface shall continue the current LIN TP message reception. ⌋ ()

[SWS_LinIf_00614] ⌈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 are specified in **SWS_LinIf_00079** and **SWS_LinIf_00613** (see **SWS_LinIf_00646**). ⌋ ()

[SWS_LinIf_00080] ⌈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.

[SWS_LinIf_00664] ⌈The LIN Interface shall provide the N_Cr timeout observation (configuration parameter LinTpNcr) in order to switch a schedule table from diagnostic schedule to applicative schedule in case the reception for SRF is not successful. ⌋ ()

[SWS_LinIf_00665] ⌈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.⌋()

[SWS_LinIf_00666] ⌈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 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 **SWS_LinIf_00646**).⌋()

[SWS_LinIf_00617] ⌈The LIN Interface shall provide the P2 timeout observation (configuration parameter LinTpP2Timing) in order to switch a schedule table from diagnostic schedule to applicative schedule in case the reception for SRF is not successful.⌋()

[SWS_LinIf_00618] ⌈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.⌋()

[SWS_LinIf_00619] ⌈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 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 **SWS_LinIf_00646**).⌋()

[SWS_LinIf_00621] ⌈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 timer is reloaded with the timeout time P2*max (configuration parameter LinTpP2Max).⌋()

[SWS_LinIf_00623] ⌈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 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 **SWS_LinIf_00646**).⌋()

7.6 Handling multiple channels and drivers

Normally, only one LIN driver (supporting multiple channels) is needed for the LIN Interface. However, in rare cases the ECU contains different LIN hardware. In such cases, multiple LIN drivers are used.

7.6.1 Multiple channels

[SWS_LinIf_00461] ⌈Each channel of the LIN Interface shall have a unique internal channel index even when the LIN channels are located on different LIN Drivers. The channel index is derived from ComM channel (LinIfComMNetworkHandleRef). ⌋ ()

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.

[SWS_LinIf_00462] ⌈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.

[SWS_LinIf_00560] ⌈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

7.7.1 Development Errors

[SWS_LinIf_00376] ⌈The following Table 1 shows the available error codes, which shall be detected by the LIN Interface and the LIN TP:

Type or error	*Related error code*	*Value [hex]*
API called without initialization of LIN Interface	LINIF_E_UNINIT	0x00
Referenced channel does not exist (identification is out of range)	LINIF_E_NONEXISTENT_CHANNEL	0x20
API service called with wrong parameter	LINIF_E_PARAMETER	0x30
API service called with invalid pointer	LINIF_E_PARAM_POINTER	0x40
Schedule request made in channel sleep	LINIF_E_SCHEDULE_REQUEST_ERROR	0x51
API service called with invalid parameter for LIN transceiver operation mode	LINIF_E_TRCV_INV_MODE	0x53
Referenced transceiver state is not normal	LINIF_E_TRCV_NOT_NORMAL	0x54
API service called with invalid parameter for WakeupSource	LINIF_E_PARAM_WAKEUPSOURCE	0x55

Table 1 – Development Error codes for DET

⌋(SRS_BSW_00406, SRS_BSW_00337, SRS_BSW_00385, SRS_BSW_00327)

7.7.2 Runtime Errors

[SWS_LinIf_00729] [The following Table 2 shows the available error codes, which shall be detected by the LIN Interface and the LIN TP:

Type or error	*Related error code*	*Value [hex]*
LIN frame error detected	LINIF_E_RESPONSE	0x60
Slave did not answer on a node configuration request	LINIF_E_NC_NO_RESPONSE	0x61

Table 2 – Runtime Error codes for DET

] (SRS_BSW_00452, SRS_BSW_00385, SRS_BSW_00327)

7.7.3 Transient Faults

There are no Transient Faults.

7.7.4 Production Errors

There are no Production Errors.

7.7.5 Extended Production Errors

There are no Extended Production Errors.

7.8 Error detection

[SWS_LinIf_00535] ⌈All LIN Interface API services other than LinIf_Init and LinIf_GetVersionInfo shall:
- not execute their normal operation,
- report to the default 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. ⌋ (SRS_BSW_00406)

[SWS_LinIf_00687] ⌈All LIN TP API services other than LinTp_Init and LinTp_GetVersionInfo shall:
- not execute their normal operation,
- report to the default 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. ⌋ (SRS_BSW_00406)

8 API specification

8.1 Imported types

8.1.1 Standard types

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

[SWS_LinIf_00469] ⌈

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

⌋ (SRS_BSW_00413)

8.1.2 Type definitions

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

8.1.2.1 LinIf_SchHandleType

[SWS_LinIf_00197] ⌈

Name:	`LinIf_SchHandleType`		
Type:	`uint8`		
Range:	`NULL_SCHEDULE`	`0x00`	The NULL_SCHEDULE.
	`range`	`1..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.
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		

⌋ (SRS_BSW_00413)

8.1.2.2 LinIf_ConfigType

[SWS_LinIf_00668] ⌈

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

[SWS_LinIf_00426] ⌈

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

[SWS_LinIf_00629] ⌈

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

[SWS_LinIf_00198] ⌈

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.

⌋ (SRS_BSW_00101, SRS_BSW_00416, SRS_BSW_00358, SRS_Lin_01569, SRS_BSW_00414)

[SWS_LinIf_00373] ⌈The function LinIf_Init shall accept a parameter that references to a LIN Interface configuration descriptor. ⌋ (SRS_BSW_00344, SRS_BSW_00404, SRS_BSW_00405, SRS_BSW_00170)

[SWS_LinIf_00233] ⌈The function LinIf_Init shall set the schedule type NULL_SCHEDULE for each configured channel. ⌋ ()

8.2.2 LinIf_GetVersionInfo

[SWS_LinIf_00340] ⌈

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.

⌋ (SRS_BSW_00407)

[SWS_LinIf_00640] ⌈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_PARAM_POINTER. ⌋ ()

8.2.3 LinIf_Transmit

[SWS_LinIf_00201] ⌈

Service name:	LinIf_Transmit

Syntax:	`Std_ReturnType LinIf_Transmit(`
	`  PduIdType TxPduId,`
	`  const PduInfoType* PduInfoPtr`

	)	
Service ID[hex]:	0x49	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant for different PduIds. Non reentrant for the same PduId.	
Parameters (in):	TxPduId	Identifier of the PDU to be transmitted
	PduInfoPtr	Length of and pointer to the PDU data and pointer to MetaData.
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.
Description:	Requests transmission of a PDU.	

」(SRS_Lin_01571)

Note: TxPduId is the identifier of LIN frame for upper layer (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.

[SWS_LinIf_00105] ⌈The function LinIf_Transmit shall indicate a request from an upper layer to transmit a frame specified by the parameter TxPduId. ⌋()

[SWS_LinIf_00341] ⌈The function LinIf_Transmit shall only mark a sporadic frame as pending for transmission and shall ignore non-sporadic frames. ⌋()

[SWS_LinIf_00700] ⌈The function LinIf_Transmit shall also return E_OK in case the Pdu belongs to a non-sporadic frame and LIN Interface is initialized. ⌋()

[SWS_LinIf_00106] ⌈The function LinIf_Transmit shall tolerate repeated invocations while the sporadic frame is still pending. ⌋()

[SWS_LinIf_00570] ⌈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_PARAM_POINTER. ⌋()

[SWS_LinIf_00575] ⌈If development error detection is enabled and the parameter TxPduId has an invalid value, the function LinIf_Transmit shall raise the development error code LINIF_E_PARAMETER. ⌋()

[SWS_LinIf_00719] ⌈LinIf_Transmit() shall return E_NOT_OK in case the LinIf's current schedule is NULL_SCHEDULE. ⌋()

Caveats: The LIN Interface has to be initialized with a call of LinIf_Init before this API service may be called, see **SWS_LinIf_00535**.

8.2.4 LinIf_ScheduleRequest

[SWS_LinIf_00202] ⌈

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.	

⌋ (SRS_Lin_01564)

The schedule tables are configured by the LinIfScheduleTable container in the LIN Interface configuration.

[SWS_LinIf_00389] ⌈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.

[SWS_LinIf_00563] ⌈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. ⌋ ()

[SWS_LinIf_00567] ⌈If development error detection is enabled and an invalid schedule table is given or the corresponding channel is in the state LINIF_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 **SWS_LinIf_00535**.

8.2.5 LinIf_GotoSleep

[SWS_LinIf_00204] ⌈

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.	

⌋ (SRS_Lin_01523)

[SWS_LinIf_00488] ⌈The function LinIf_GotoSleep shall initiate a transition into sleep mode on the selected channel. (see **SWS_LinIf_00453** and **SWS_LinIf_00597**) ⌋ ()

[SWS_LinIf_00564] ⌈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. ⌋ ()

[SWS_LinIf_00113] ⌈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 **SWS_LinIf_00535**.

8.2.6 LinIf_Wakeup

[SWS_LinIf_00205] ⌈

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) - Lin_Wakeup has returned E_NOT_OK - Lin_WakeupInternal has returned E_NOT_OK
Description:	Initiates the wake up process.	

⌋(SRS_Lin_01515)

[SWS_LinIf_00432] ⌈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.⌋()

[SWS_LinIf_00296] ⌈ 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 LINIF_CHANNEL_SLEEP and the wakeup flag of the selected channel is not set. (see **SWS_LinIf_00716**)⌋()

[SWS_LinIf_00713] ⌈The function LinIf_Wakeup shall call the function Lin_WakeupInternal of the LIN Driver module to set selected channel to the wakeup state, if the channel is in the channel state LINIF_CHANNEL_SLEEP and the wakeup flag of the selected channel is set. (see **SWS_LinIf_00716**)⌋()

[SWS_LinIf_00714] ⌈The function LinIf_Wakeup shall clear the wakeup flag of the selected channel.⌋()

[SWS_LinIf_00720] ⌈If the function Lin_Wakeup returns E_NOT_OK, the function LinIf_Wakeup shall return E_NOT_OK and not change the status of the wakeup flag.⌋()

[SWS_LinIf_00721] ⌈If the function Lin_WakeupInternal returns E_NOT_OK, the function LinIf_Wakeup shall return E_NOT_OK and not change the status of the wakeup flag.⌋()

[SWS_LinIf_00565] ⌈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 **SWS_LinIf_00535**.

8.2.7 LinIf_SetTrcvMode

[SWS_LinIf_00544] ⌈

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.	

⌋ (SRS_Lin_01584, SRS_Lin_01585, SRS_Lin_01586)

[SWS_LinIf_00536] ⌈This service shall call the underlying function LinTrcv_SetOpMode(LinNetwork, OpMode) for the corresponding requested LIN transceiver. ⌋ ()

[SWS_LinIf_00537] ⌈This API shall be applicable to all LIN transceivers with all values independent if the transceiver hardware supports these modes or not. ⌋ ()

[SWS_LinIf_00538] ⌈The API LinIf_SetTrcvMode returns the value that is returned by LinTrcv_SetOpMode. ⌋ ()

[SWS_LinIf_00539] ⌈If development error detection is enabled and an invalid value for Channel is given, the function LinIf_SetTrcvMode shall report LINIF_E_NONEXISTENT_CHANNEL to the default error tracer. ⌋ ()

[SWS_LinIf_00540] ⌈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 default error tracer. ⌋ ()

[SWS_LinIf_00634] ⌈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 **SWS_LinIf_00535**.

8.2.8 LinIf_GetTrcvMode

[SWS_LinIf_00545] ⌈

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 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 or channel parameter is invalid or pointer is NULL.
Description:	Returns the actual state of a LIN Transceiver Driver.	

⌋(SRS_Lin_01587)

[SWS_LinIf_00541] ⌈This service shall invoke the underlying function LinTrcv_GetOpMode(LinNetwork, OpMode) for the corresponding requested LIN transceiver. ⌋()

[SWS_LinIf_00546] ⌈If development error detection is enabled and an invalid value for Channel is given, the function LinIf_GetTrcvMode shall report LINIF_E_NONEXISTENT_CHANNEL to the default error tracer. ⌋()

[SWS_LinIf_00571] ⌈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_PARAM_POINTER. ⌋()

[SWS_LinIf_00635] ⌈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 **SWS_LinIf_00535**.

8.2.9 LinIf_GetTrcvWakeupReason

[SWS_LinIf_00547] ⌈

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 value 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 or channel parameter is invalid or pointer is NULL.
Description:	Returns the reason for the wake up that has been detected by the LIN Transceiver Driver.	

⌋ (SRS_Lin_01588)

[SWS_LinIf_00548] ⌈This API shall return the reason for the wake up that the LIN Transceiver Driver has detected by invoking the underlying function LinTrcv_GetBusWuReason(LinNetwork, Reason) for the corresponding requested LIN transceiver. ⌋ ()

[SWS_LinIf_00549] ⌈If development error detection is enabled and an invalid value for Channel is given, the function LinIf_GetTrcvWakeupReason shall report LINIF_E_NONEXISTENT_CHANNEL to the default error tracer. ⌋ ()

[SWS_LinIf_00573] ⌈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_PARAM_POINTER. ⌋ ()

[SWS_LinIf_00572] ⌈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 default error tracer. ⌋ ()

[SWS_LinIf_00636] ⌈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 **SWS_LinIf_00535**.
- 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

[SWS_LinIf_00550] ⌈

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 or channel or mode parameter is invalid.
Description:	This API enables, disables and clears the notification for wakeup events on the addressed network	

⌋ (SRS_Lin_01589)

[SWS_LinIf_00551] ⌈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). ⌋ ()

[SWS_LinIf_00595] ⌈If development error detection is enabled and an invalid value for Channel is given, the function LinIf_SetTrcvWakeupMode shall report LINIF_E_NONEXISTENT_CHANNEL to the default error tracer. ⌋ ()

[SWS_LinIf_00596] ⌈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 default error tracer. ⌋ ()

[SWS_LinIf_00637] ⌈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 **SWS_LinIf_00535**.
- 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

[SWS_LinIf_00580] ⌈

Service name:	LinIf_CancelTransmit	
Syntax:	`Std_ReturnType LinIf_CancelTransmit( PduIdType TxPduId )`	
Service ID[hex]:	0x4a	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant for different PduIds. Non reentrant for the same PduId.	
Parameters (in):	TxPduId	Identification of the PDU to be cancelled.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Cancellation was executed successfully by the destination module. E_NOT_OK: Cancellation was rejected by the destination module.
Description:	Requests cancellation of an ongoing transmission of a PDU in a lower layer communication module.	

⌋ ()

Note: This function is an empty implementation to comply with upper layer specification.

[SWS_LinIf_00649] ⌈The cancellation request shall always be rejected by returning E_NOT_OK. ⌋ ()

[SWS_LinIf_00581] ⌈The function LinIf_CancelTransmit shall be pre-compile time configurable On/Off by the configuration parameter LinIfCancelTransmitSupported. ⌋ ()

[SWS_LinIf_00594] ⌈If development error detection is enabled and an invalid value for TxPduId is given, the function LinIf_CancelTransmit shall report LINIF_E_PARAMETER to the default error tracer. ⌋ ()

Note: The TxConfirmation is not invoked in the call of the LinIf_CancelTransmit.

8.2.12 LinTp_Init

[SWS_LinIf_00350] ⌈

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.	

⌋ (SRS_BSW_00101, SRS_BSW_00414, SRS_BSW_00416, SRS_BSW_00358, SRS_Lin_01540)

[SWS_LinIf_00427] ⌈The parameter ConfigPtr of the function LinTp_Init is 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. ⌋ ()

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

[SWS_LinIf_00320] ⌈The function LinTp_Init shall set the state LINTP_INIT and sub-state LINTP_CHANNEL_IDLE for each configured channel of the LIN TP channel state-machine. ⌋ ()

[SWS_LinIf_00681] ⌈The function LinTp_Init shall be pre-compile time configurable On/Off by the configuration parameter LinIfTpSupported. ⌋ ()

8.2.13 LinTp_Transmit

[SWS_LinIf_00351] ⌈

Service name:	LinTp_Transmit	

Syntax:	`Std_ReturnType LinTp_Transmit(`
	`  PduIdType TxPduId,`
	`  const PduInfoType* PduInfoPtr`
	`)`	

Service ID[hex]:	0x49	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant for different PduIds. Non reentrant for the same PduId.	
Parameters (in):	TxPduId	Identifier of the PDU to be transmitted

	PduInfoPtr	Length of and pointer to the PDU data and pointer to MetaData.
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.
Description:	Requests transmission of a PDU.	

⌋()

[SWS_LinIf_00326] ⌈The function LinTp_Transmit shall prepare a LIN TP message for transmission. ⌋()

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

[SWS_LinIf_00413] ⌈The function LinTp_Transmit shall set the sub-state of the referenced channel to LINTP_CHANNEL_BUSY. ⌋()

[SWS_LinIf_00422] ⌈The function LinTp_Transmit shall convert the N-SDU Id (given by the parameter TxPduId) to a specific channel and a destination NAD for the slave. ⌋()

[SWS_LinIf_00584] ⌈The function LinTp_Transmit shall accept a functional transmission request also when a TP message is currently ongoing on the selected channel. ⌋()

[SWS_LinIf_00616] ⌈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. ⌋()

[SWS_LinIf_00586] ⌈According to the LIN 2.1 specification, the NAD 0x7E shall be used for a functional transmission request. ⌋()

[SWS_LinIf_00702] ⌈When LinTp_Transmit was successful (returned E_OK), the LIN Interface shall ensure that PduR_LinTpTxConfirmation is always called, with a negative or positive result. When LinTp_Transmit was not successful, PduR_LinTpTxConfirmation shall not be called. ⌋()

[SWS_LinIf_00574] ⌈If development error detection is enabled and the parameter PduInfoPtr has an invalid value, the function LinTp_Transmit shall raise the development error code LINIF_E_PARAM_POINTER. ⌋()

[SWS_LinIf_00576] ⌈If development error detection is enabled and the parameter TxPduId has an invalid value, the function LinTp_Transmit shall raise the development error code LINIF_E_PARAMETER. ⌋()

[SWS_LinIf_00682] ⌈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 **SWS_LinIf_00535** and **SWS_LinIf_00687**.

8.2.14 LinTp_GetVersionInfo

[SWS_LinIf_00352] ⌈

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.

⌋ (SRS_BSW_00407)

[SWS_LinIf_00639] ⌈If development error detection is enabled and the parameter versioninfo has an invalid value, the function LinTp_GetVersionInfo shall raise the development error code LINIF_E_PARAM_POINTER.⌋ ()

8.2.15 LinTp_Shutdown

[SWS_LinIf_00355] ⌈

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.

⌋ (SRS_BSW_00336)

[SWS_LinIf_00356] ⌈The function LinTp_Shutdown shall close all pending transport protocol connection of the LIN TP and free all resources of the LIN TP. ⌋()

[SWS_LinIf_00433] ⌈The function LinTp_Shutdown shall affect all configured channels. ⌋()

[SWS_LinIf_00484] ⌈The function LinTp_Shutdown shall set the LIN TP state of all channels to LINTP_UNINIT. ⌋()

[SWS_LinIf_00683] ⌈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 **SWS_LinIf_00535** and **SWS_LinIf_00687**.

8.2.16 LinTp_CancelTransmit

[SWS_LinIf_00500] ⌈

Service name:	LinTp_CancelTransmit	
Syntax:	`Std_ReturnType LinTp_CancelTransmit( PduIdType TxPduId )`	
Service ID[hex]:	0x4a	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant for different PduIds. Non reentrant for the same PduId.	
Parameters (in):	TxPduId	Identification of the PDU to be cancelled.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Cancellation was executed successfully by the destination module. E_NOT_OK: Cancellation was rejected by the destination module.
Description:	Requests cancellation of an ongoing transmission of a PDU in a lower layer communication module.	

⌋()

Note: This function is an empty implementation to comply with upper layer specification.

[SWS_LinIf_00490] ⌈The cancellation request shall always be rejected by returning E_NOT_OK. ⌋()

[SWS_LinIf_00577] ⌈If development error detection is enabled and the parameter TxPduId has an invalid value, the function LinTp_CancelTransmit shall raise the development error code LINIF_E_PARAMETER. ⌋()

[SWS_LinIf_00684] ⌈The function LinTp_CancelTransmit shall be pre-compile time configurable On/Off by the configuration parameter LinIfTpSupported. ⌋()

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

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 **SWS_LinIf_00535** and **SWS_LinIf_00687**.

8.2.17 LinTp_ChangeParameter

[SWS_LinIf_00501] ⌈

Service name:	LinTp_ChangeParameter	
Syntax:	`Std_ReturnType LinTp_ChangeParameter( PduIdType id, TPParameterType parameter, uint16 value )`	
Service ID[hex]:	0x4b	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	id	Identification of the PDU which the parameter change shall affect.
	parameter	ID of the parameter that shall be changed.
	value	The new value of the parameter.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: The parameter was changed successfully. E_NOT_OK: The parameter change was rejected.
Description:	Request to change a specific transport protocol parameter (e.g. block size).	

⌋()

Note: This function is an empty implementation to comply with upper layer specification.

[SWS_LinIf_00592] ⌈The change parameter request shall always be rejected by returning E_NOT_OK. ⌋()

[SWS_LinIf_00578] ⌈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. ⌋()

[SWS_LinIf_00685] ⌈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 **SWS_LinIf_00535** and **SWS_LinIf_00687**.

8.2.18 LinIf_CheckWakeup

[SWS_LinIf_00378] ⌈

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 or invalid WakeupSource
Description:	Will be called when the EcuM has been notified about a wakeup on a specific LIN channel.	

⌋ (SRS_Lin_01514, SRS_BSW_00375)

The LIN Interface will recognize the source of the wakeup and thus the destination of this call by the parameter of the function LinIf_CheckWakeup.

[SWS_LinIf_00503] ⌈The function LinIf_CheckWakeup shall issue the call of function Lin_CheckWakeup or LinTrcv_CheckWakeup depending on the given parameter WakeupSource. ⌋ ()

[SWS_LinIf_00566] ⌈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. ⌋ ()

[SWS_LinIf_00689] ⌈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

[SWS_LinIf_00625] ⌈

Service name:	LinTp_CancelReceive	
Syntax:	`Std_ReturnType LinTp_CancelReceive( PduIdType RxPduId )`	
Service ID[hex]:	0x4c	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	RxPduId	Identification of the PDU to be cancelled.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Cancellation was executed successfully by the destination module. E_NOT_OK: Cancellation was rejected by the destination module.
Description:	Requests cancellation of an ongoing reception of a PDU in a lower layer transport protocol module.	

⌋()

Note: This function is an empty implementation to comply with upper layer specification.

[SWS_LinIf_00626] ⌈The cancellation request shall always be rejected by returning E_NOT_OK.⌋()

[SWS_LinIf_00627] ⌈If development error detection is enabled and the parameter RxPduId has an invalid value, the function LinTp_CancelReceive shall raise the development error code LINIF_E_PARAMETER.⌋()

[SWS_LinIf_00686] ⌈The function LinTp_CancelReceive shall be pre-compile time configurable On/Off by the configuration parameter LinIfTpSupported.⌋()

Note: The RxIndication is not invoked in the call of the LinTp_CancelReceive.

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 **SWS_LinIf_00535** and **SWS_LinIf_00687**.

8.3 Call-back notifications

This is a list of functions provided for other modules.

8.3.1 LinIf_WakeupConfirmation

[SWS_LinIf_00715] ⌈

Service name:	LinIf_WakeupConfirmation
Syntax:	`void LinIf_WakeupConfirmation( EcuM_WakeupSourceType WakeupSource )`

Service ID[hex]:	0x61	
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:	None	
Description:	The LIN Driver or LIN Transceiver Driver will call this function to report the wake up source after the successful wakeup detection during CheckWakeup or after power on by bus.	

⌋ ()

[SWS_LinIf_00716] ⌈The function LinIf_WakeupConfirmation shall set the wakeup flag for the channel depending on the given parameter WakeupSource. The wakeup flags shall be provided for each channel. ⌋ ()

[SWS_LinIf_00717] ⌈If development error detection is enabled and the parameter WakeupSource has an invalid value, the function LinIf_WakeupConfirmation shall raise the development error code LINIF_E_PARAM_WAKEUPSOURCE. ⌋ ()

[SWS_LinIf_00718] ⌈The function LinIf_WakeupConfirmation 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. ⌋ ()

8.4 Scheduled functions

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

8.4.1 LinIf_MainFunction_<ChannelId>

[SWS_LinIf_00384] ⌈

Service name:	LinIf_MainFunction_<ChannelId>

Syntax:	`void LinIf_MainFunction_<ChannelId>(`
	`  void`
	`)`

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

⌋ (SRS_BSW_00373, SRS_Lin_01546, SRS_Lin_01561, SRS_Lin_01555)

Design hint: The function LinIf_MainFunction_<ChannelId> 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, LinIf_Transmit, LinIf_ScheduleRequest and LinIf_Wakeup, and potentially also LinIf_Init, LinTp_Init and LinTp_Shutdown.

[SWS_LinIf_00725] ⌈The function LinIf_MainFunction_<ChannelId> shall exist once per LIN channel of the LIN Interface module. ⌋()

[SWS_LinIf_00726] ⌈The function name of each instance of LinIf_MainFunction_<ChannelId> shall contain the index of the respective LIN channel (ChannelId).
i.e.) LinIf_MainFunction_0, LinIf_MainFunction_1... ⌋()

[SWS_LinIf_00473] ⌈The function LinIf_MainFunction_<ChannelId> shall operate per LIN channel of the LIN Interface module. ⌋()

[SWS_LinIf_00286] ⌈The function LinIf_MainFunction_<ChannelId> shall poll the Schedule Table Manager which frame shall be transported. ⌋()

[SWS_LinIf_00287] ⌈Only the function LinIf_MainFunction_<ChannelId> shall process the transportation (transmission and reception) of frames. ⌋()

8.5 Expected Interfaces

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

8.5.1 Mandatory Interfaces

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

[SWS_LinIf_00359] ⌈

API function	*Description*
BswM_LinTp_RequestMode	Function called by LinTP to request a mode for the corresponding LIN channel. The LinTp_Mode correlates to the LIN schedule table that should be used.
Det_ReportRuntimeError	Service to report runtime errors. If a callout has been configured then this callout shall be called.
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 to reduced power operation mode (if supported by HW).
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 and sets the channel state to LIN_CH_OPERATIONAL.
Lin_WakeupInternal	Sets the channel state to LIN_CH_OPERATIONAL without generating 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.

[SWS_LinIf_00360] [

API function	*Description*
Det_ReportError	Service to report development errors.
Lin_CheckWakeup	This function checks if a wakeup has occurred on the addressed LIN channel.
LinSM_GotoSleepConfirmation	The LinIf will call this callback when the go to sleep command is sent successfully or not sent successfully on the network.
LinSM_ScheduleRequestConfirmation	The LinIf module will call this callback when the new requested schedule table is active.
LinSM_WakeupConfirmation	The LinIf will call this callback when the wake up signal command is sent not successfully/successfully on the network.
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.
PduR_LinIfRxIndication	Indication of a received PDU from a lower layer communication interface module.
PduR_LinIfTriggerTransmit	Within this API, the upper layer module (called module) shall check whether the available data fits into the buffer size reported by PduInfoPtr->SduLength. If it fits, it shall copy its data into the buffer provided by PduInfoPtr->SduDataPtr and update the length of the actual copied data in PduInfoPtr->SduLength. If not, it returns E_NOT_OK without changing PduInfoPtr.
PduR_LinIfTxConfirmation	The lower layer communication interface module confirms the transmission of a PDU, or the failure to transmit a PDU.
PduR_LinTpCopyRxData	This function is called to provide the received data of an I-PDU segment (N-PDU) to the upper layer. Each call to this function provides the next part of the I-PDU data. The size of the remaining buffer is written to the position indicated by bufferSizePtr.
PduR_LinTpCopyTxData	This function is called to acquire the transmit data of an I-PDU segment (N-PDU). Each call to this function provides the next part of the I-PDU data unless retry->TpDataState is TP_DATARETRY. In this case the function restarts to copy the data beginning at the offset from the current position indicated by retry->TxTpDataCnt. The size of the remaining data is written to the position indicated by availableDataPtr.
PduR_LinTpRxIndication	Called after an I-PDU has been received via the TP API, the

	result indicates whether the transmission was successful or not.
PduR_LinTpStartOfReception	This function is called at the start of receiving an N-SDU. The N-SDU 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 after the I-PDU has been transmitted on its network, the result indicates whether 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

[SWS_LinIf_00520] [

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

] ()

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

[SWS_LinIf_00521] ⌈

Service name:	< User >_GotoSleepConfirmation	

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

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

⌋ ()

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

[SWS_LinIf_00522] ⌈

Service name:	< User >_WakeupConfirmation	

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

Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	channel	Identification of the LIN channel
	success	True if wakeup was successfully sent, false otherwise
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	The LinIf will call this function when the wake up signal command is sent not successfully/successfully on the bus.	

⌋ ()

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>

[SWS_LinIf_00528] ⌈

Service name:	<User_TriggerTransmit>	

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

Sync/Async:	Synchronous	
Reentrancy:	Reentrant for different PduIds. Non reentrant for the same PduId.	
Parameters (in):	TxPduId	ID of the SDU that is requested to be transmitted.
Parameters (inout):	PduInfoPtr	Contains a pointer to a buffer (SduDataPtr) to where the SDU data shall be copied, and the available buffer size in SduLengh. On return, the service will indicate the length of the copied SDU data in SduLength.
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: SDU has been copied and SduLength indicates the number of copied bytes. E_NOT_OK: No SDU data has been copied. PduInfoPtr must not be used since it may contain a NULL pointer or point to invalid data.
Description:	Within this API, the upper layer module (called module) shall check whether the available data fits into the buffer size reported by PduInfoPtr->SduLength. If it fits, it shall copy its data into the buffer provided by PduInfoPtr->SduDataPtr and update the length of the actual copied data in PduInfoPtr->SduLength. If not, it returns E_NOT_OK without changing PduInfoPtr.	

⌋ ()

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.

[SWS_LinIf_00722] ⌈Configuration of <User_TriggerTransmit>: If LinIfUserTxUL is set to PDUR, LinIfTxTriggerTransmitUL must be PduR_LinIfTriggerTransmit. ⌋ ()

8.5.3.5 <User_TxConfirmation>

[SWS_LinIf_00529] ⌈

Service name:	<User_TxConfirmation>	

Syntax:	`void <User_TxConfirmation>(`
	`  PduIdType TxPduId,`
	`  Std_ReturnType result`
	`)`	

Sync/Async:	Synchronous	
Reentrancy:	Reentrant for different PduIds. Non reentrant for the same PduId.	
Parameters (in):	TxPduId	ID of the PDU that has been transmitted.
	result	E_OK: The PDU was transmitted. E_NOT_OK: Transmission of the PDU failed.
Parameters	None	

(inout):	
Parameters (out):	None
Return value:	None
Description:	The lower layer communication interface module confirms the transmission of a PDU, or the failure to transmit a PDU.

⌋ ()

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.

[SWS_LinIf_00723] ⌈Configuration of <User_TxConfirmation>: If LinIfUserTxUL is set to PDUR, LinIfTxConfirmationUL must be PduR_LinIfTxConfirmation. ⌋ ()

8.5.3.6 <User_RxIndication>

[SWS_LinIf_00530] ⌈

Service name:	<User_RxIndication>	

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

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

⌋ ()

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.

[SWS_LinIf_00724] ⌈Configuration of <User_RxIndication>: If LinIfUserRxIndicationUL is set to PDUR, LinIfRxIndicationUL must be PduR_LinIfRxIndication. ⌋ ()

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_<ChannelId> 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_<ChannelId>, the frame is finished.

The RequestFrame call in the diagram is the interface call to the Schedule Table Manager. The LinIf_MainFunction_<ChannelId> 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.

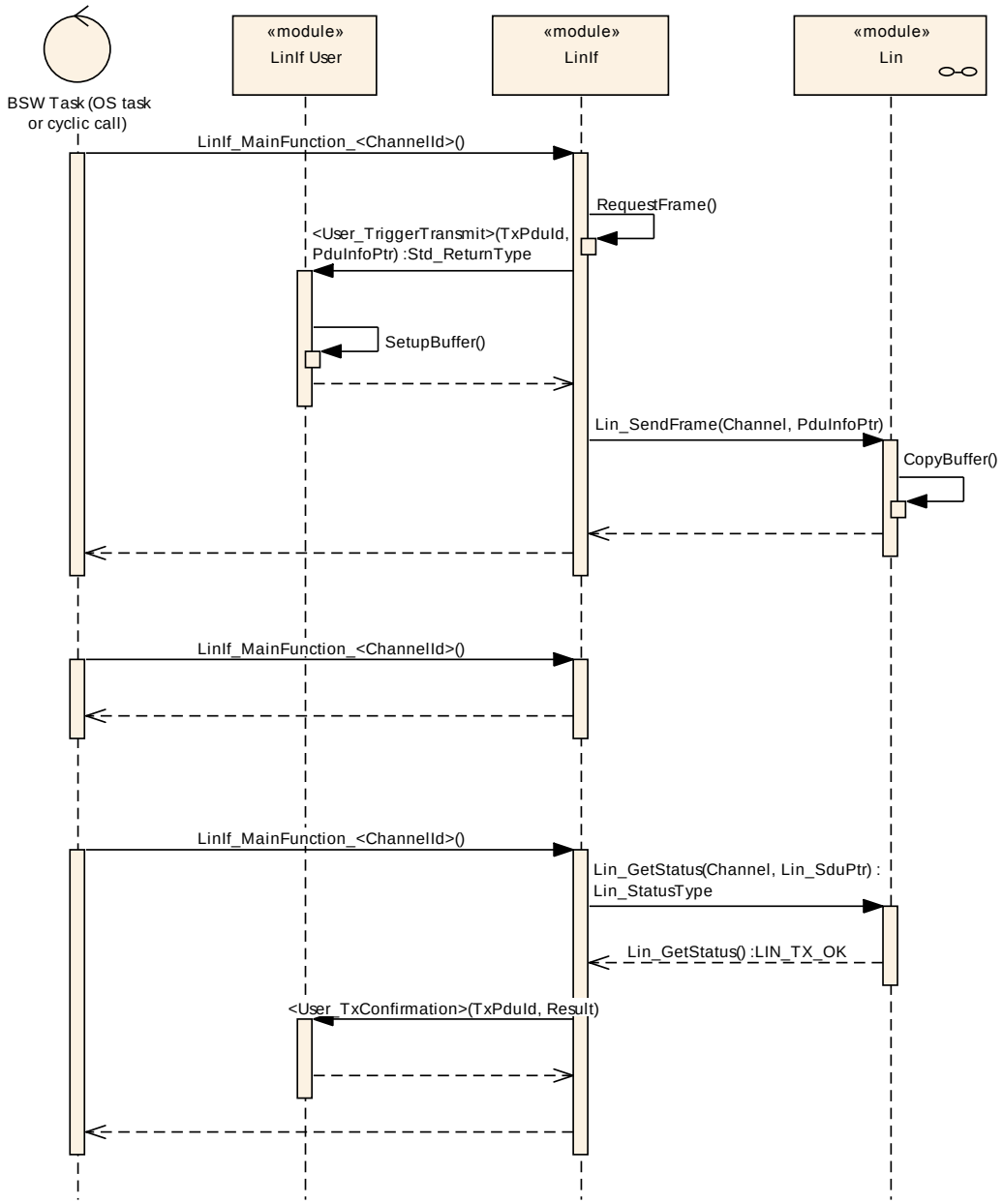

Figure 7 – Frame transmission

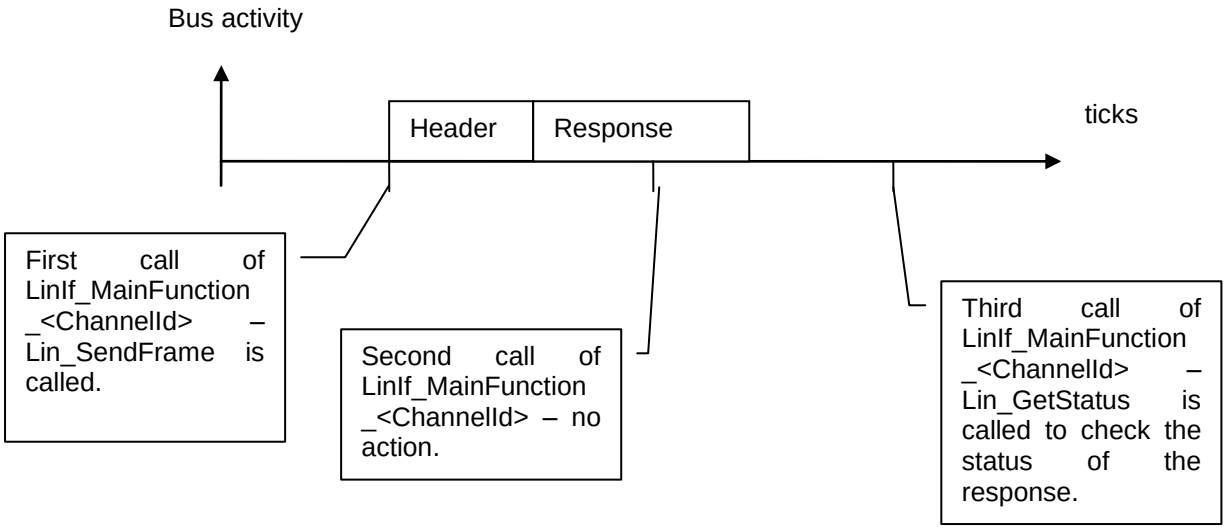

Figure 8 – Timing information for transmitted frame

9.2 Frame Reception

The following use case shows the reception of a LIN frame. The first call of the LinIf_MainFunction_<ChannelId> 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_<ChannelId> 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.

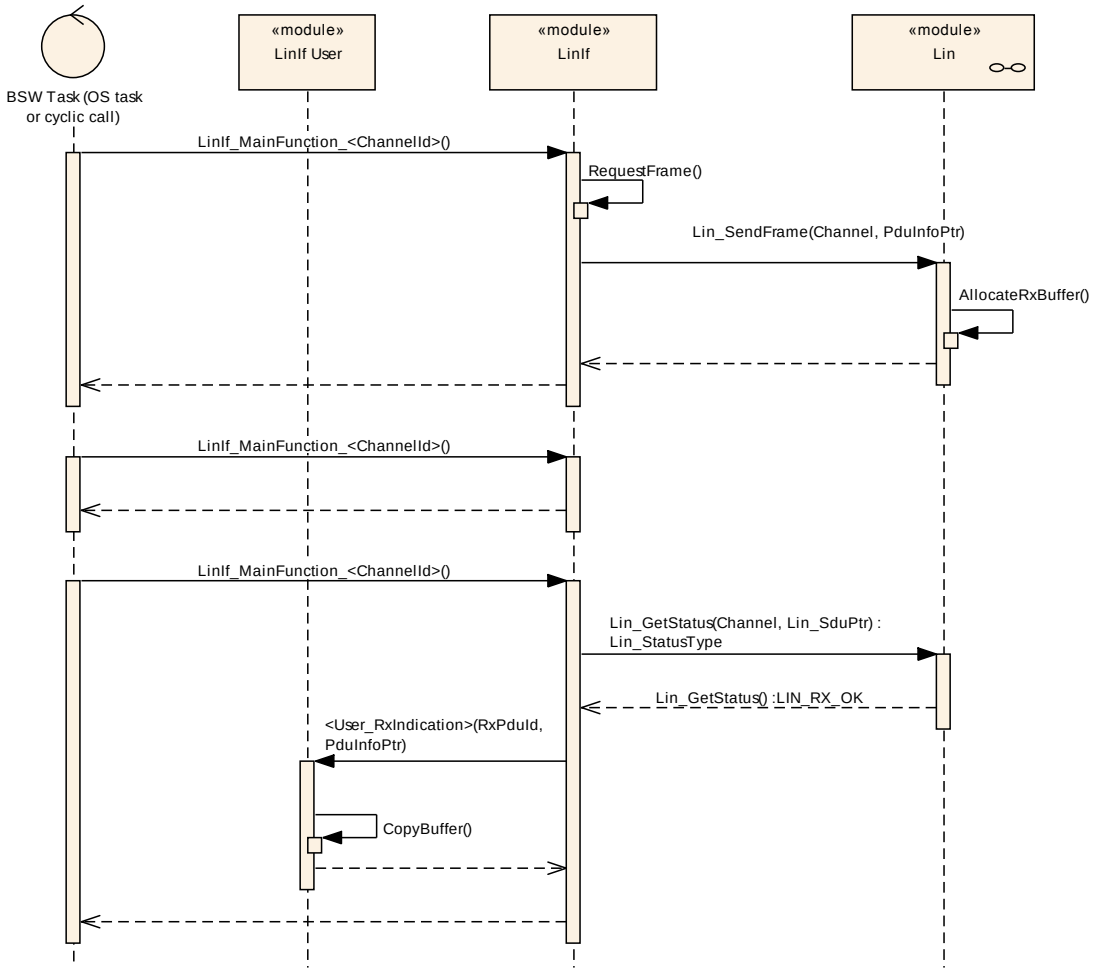

Figure 9 – Frame reception

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.

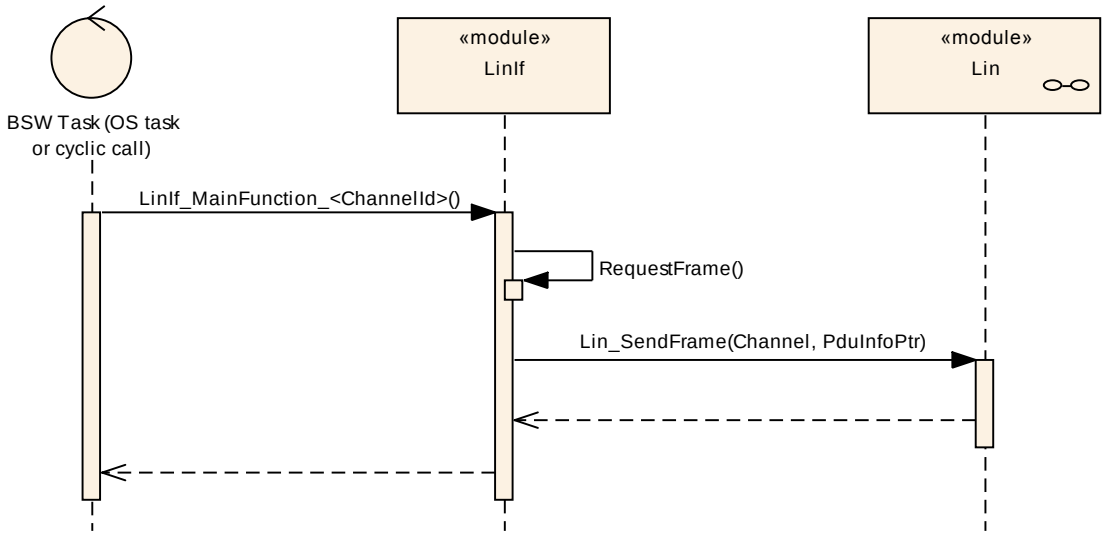

Figure 10 – Slave to slave communication

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_<ChannelId> 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.

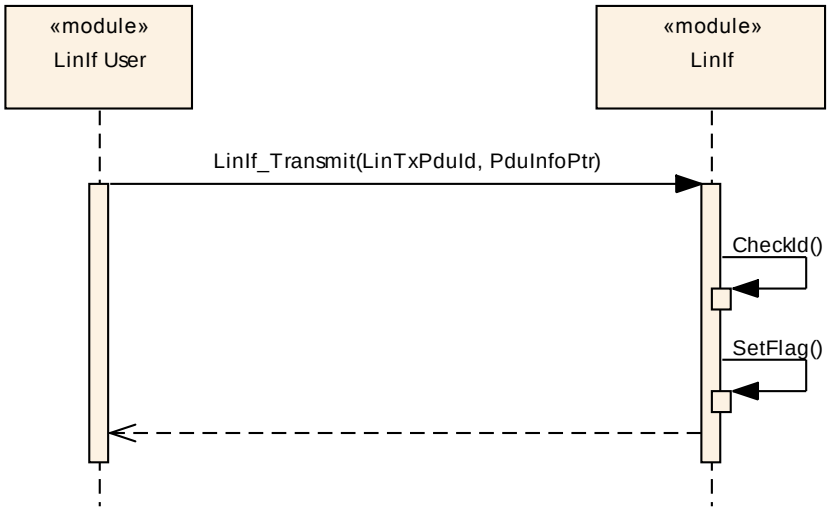

Figure 11 – Sporadic frame

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_<ChannelId> 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_<ChannelId> 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_<ChannelId> recognizes this situation and takes no action since this is not considered to be a communication error.

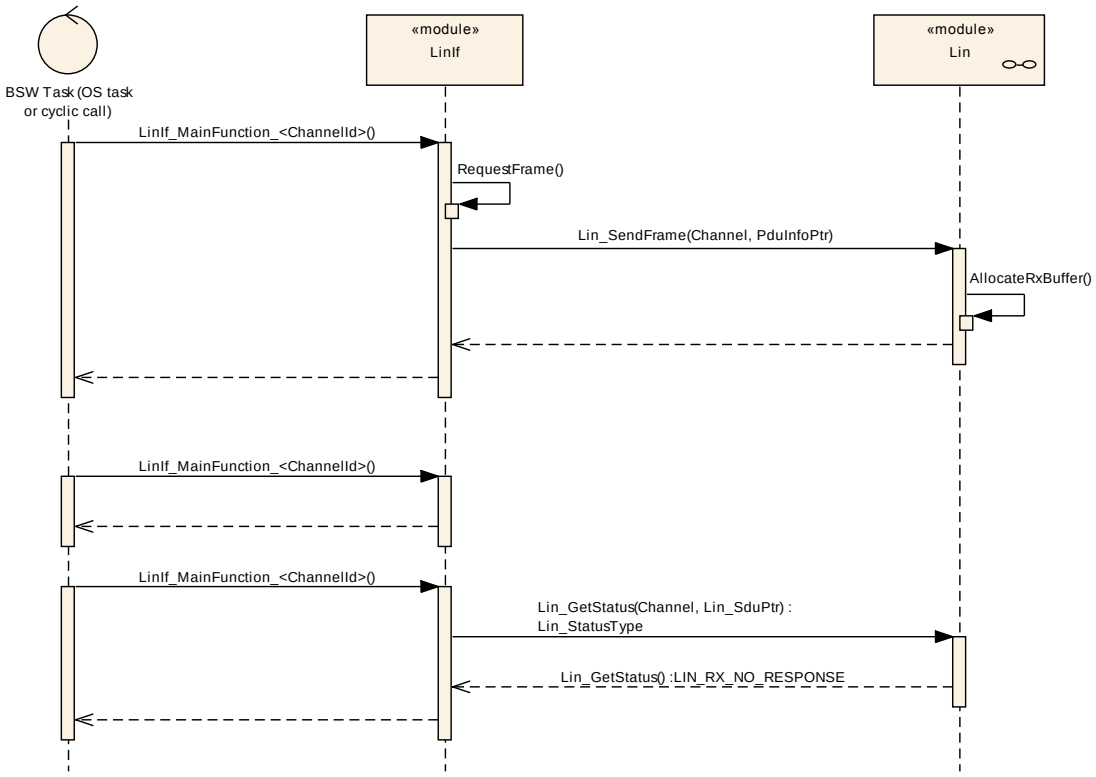

Figure 12 – Event-triggered frame with no answer

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_<ChannelId> 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_<ChannelId> gets the frame to send and the delay to the next frame.

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

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

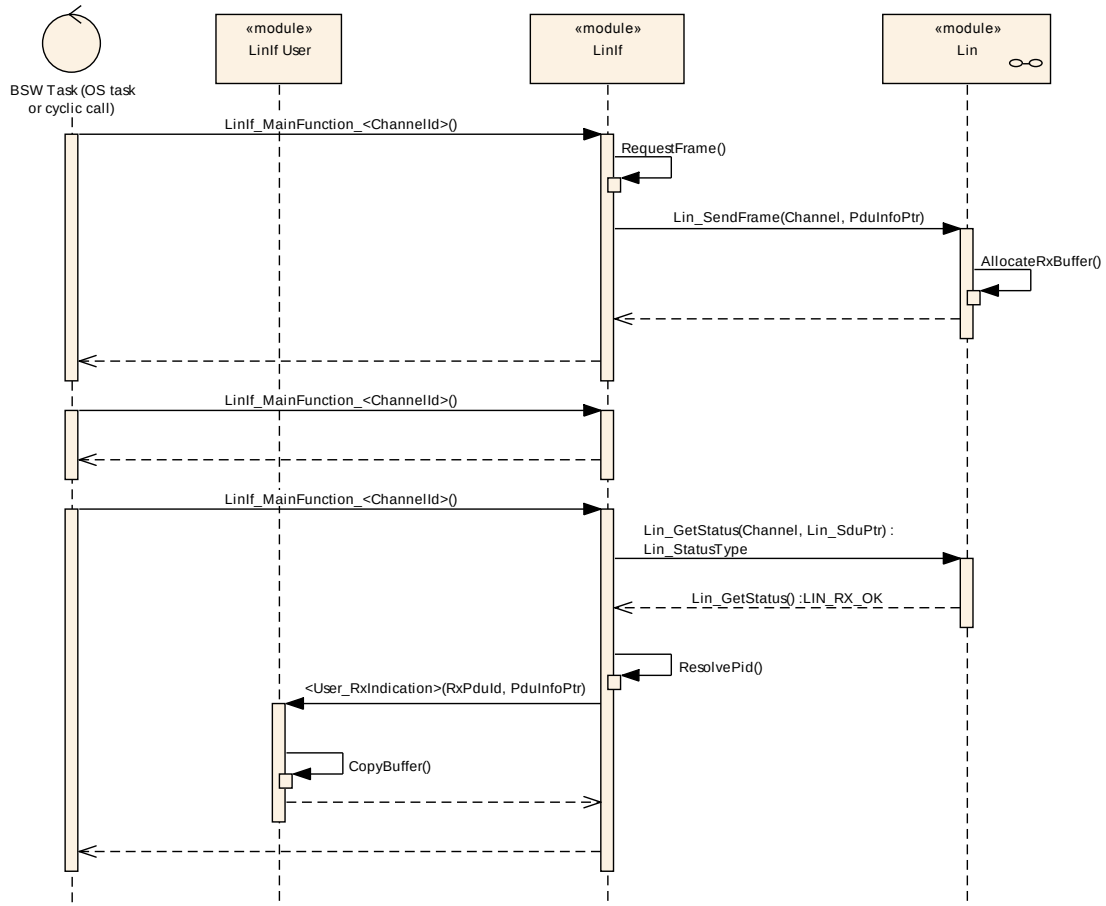

Figure 13 – Event-triggered frame with answer (no collision)

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_<ChannelId> 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_<ChannelId> 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.

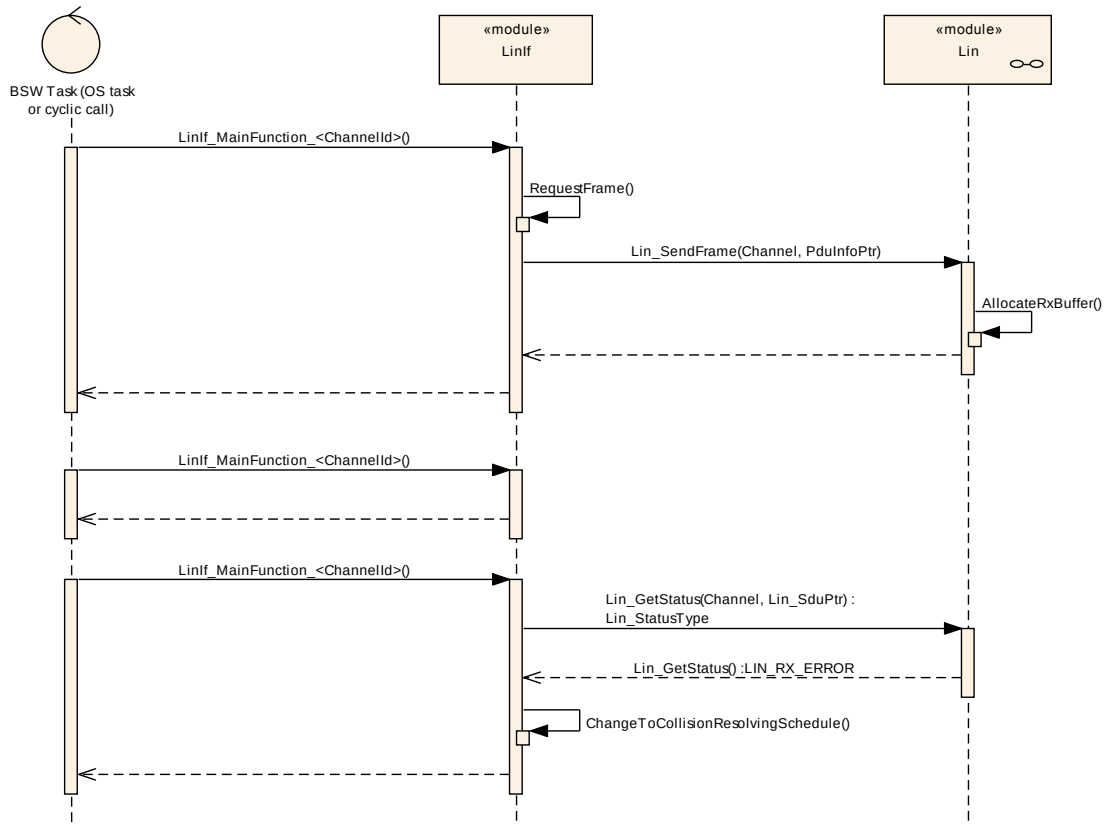

Figure 14 – Event-triggered frame with collision

9.6 Transport Protocol message transmission

The following diagram Figure 15 shows the transmission of a TP message. The initiation of the message, the continuous copying of the data and the finish of the message are shown. The actual transmission of the MRF is not shown in the diagram and it has the same behavior as frame transmission.

The TP message start is always initiated by requesting to send the TP message from the PDU Router. The schedule table change to the diagnostic request schedule is requested if a schedule table change is enabled by the configuration parameter. (see the parameter LinTpScheduleChangeDiag)

The TP message is finished after the last N-PDU (SF or CF) is transmitted. The PDU Router is notified of the completion of the message transmission. The schedule table change to the diagnostic response schedule is requested if a schedule table change is enabled by the configuration parameter. (see the parameter LinTpScheduleChangeDiag)

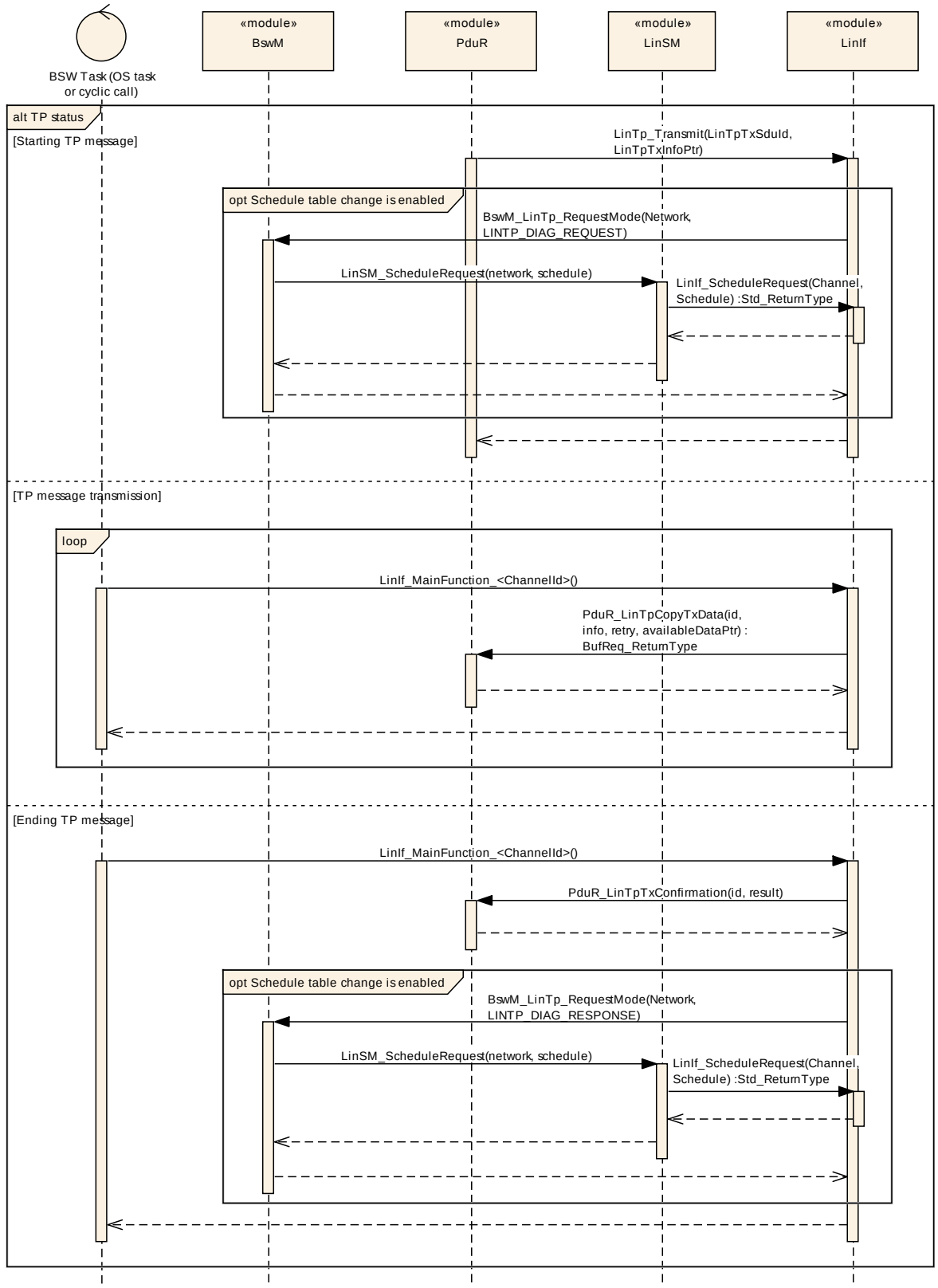

Figure 15 –Transport Protocol message transmission

9.7 Transport Protocol message reception

The following diagram Figure 16 shows the reception of a TP message. The initiation of the message, the continuous copying of the data 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. Incoming data is provided to the PDU Router via the API PduR_LinTpCopyRxData.

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

The TP message is finished after the last N-PDU (SF or CF) is received. The PDU Router is notified of the reception of the complete message. The schedule table change to the applicative schedule is requested if a schedule table change is enabled by the configuration parameter. (see the parameter LinTpScheduleChangeDiag)

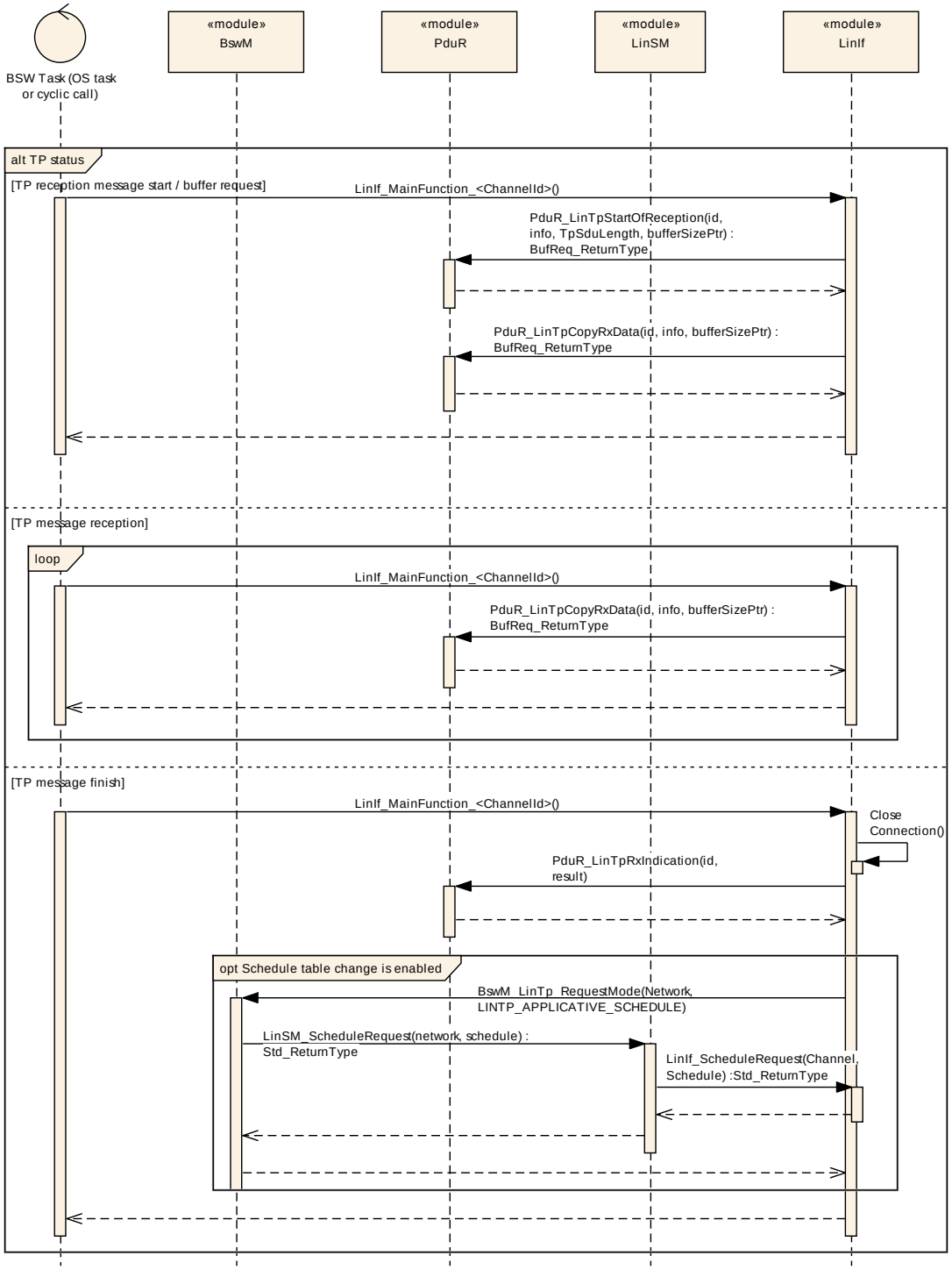

Figure 16 – Transport Protocol message reception

9.8 Go to sleep process

This use case in Figure 17 shows the execution of the LinIf_GotoSleep command.

The LinIf_MainFunction_<ChannelId> 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.

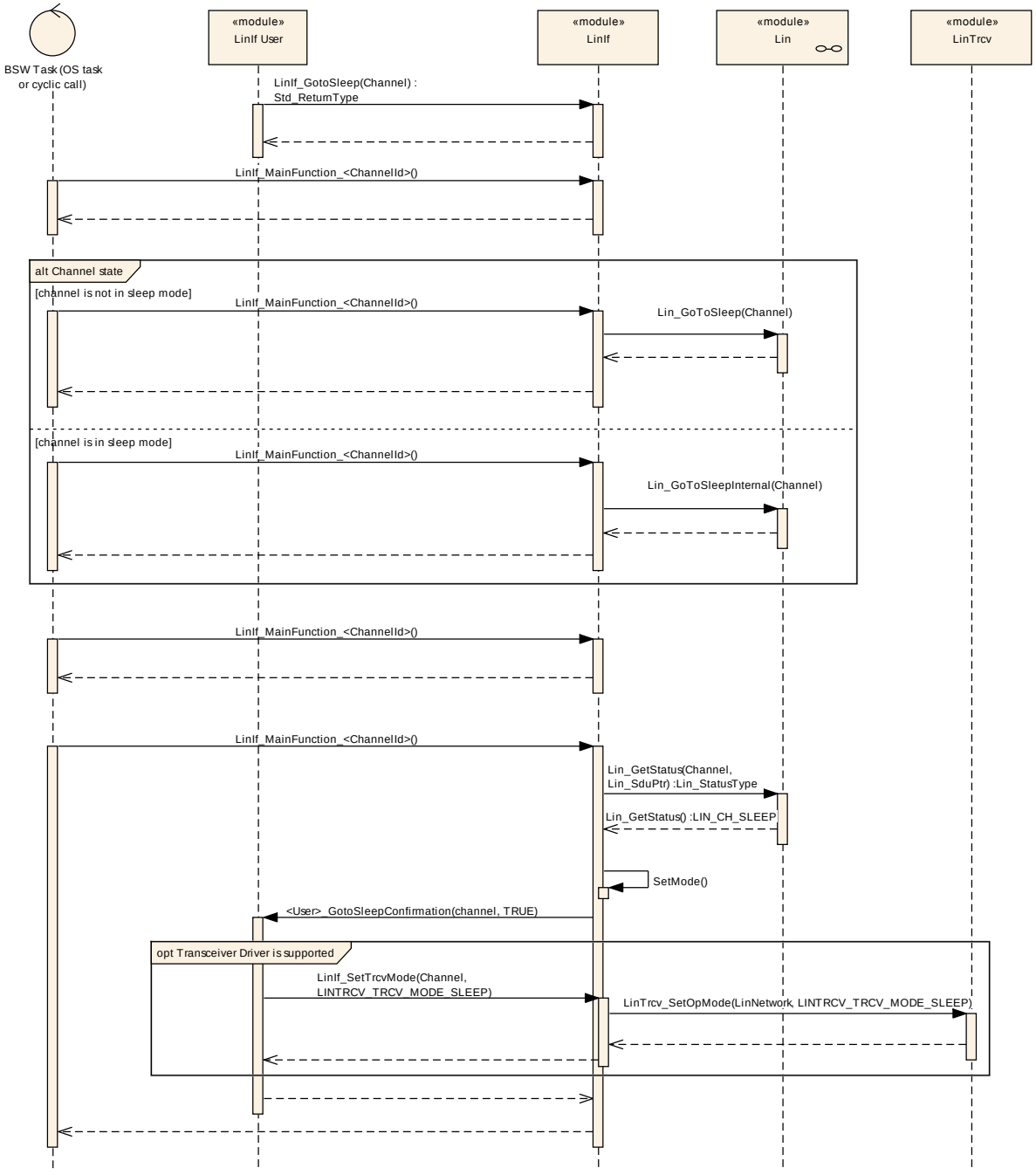

Figure 17 – Go-to-sleep command process

9.9 Wake up request

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

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.

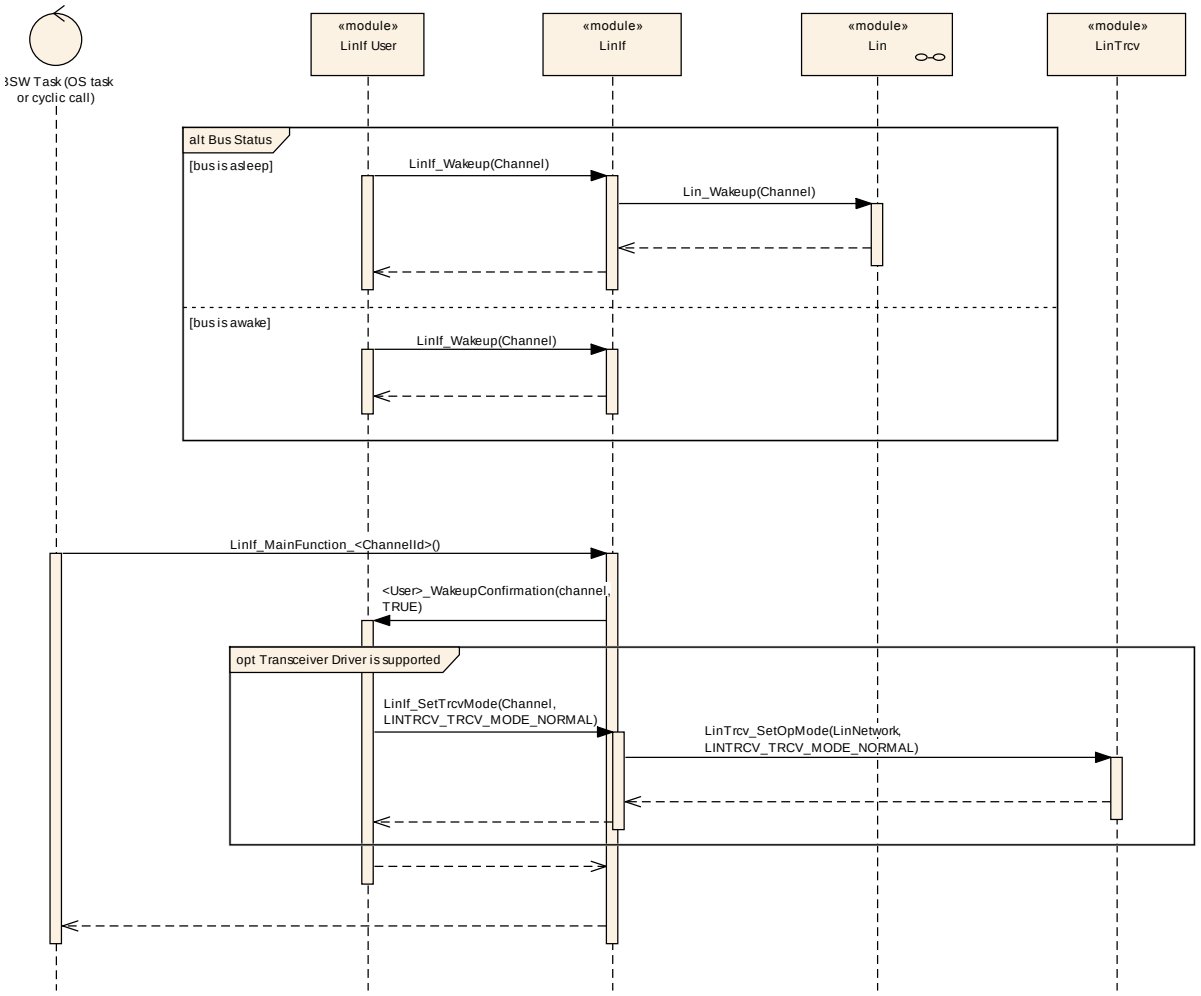

Figure 18 – Internal wake-up

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

For details refer to the chapter 10.1 “Introduction to configuration specification” in *SWS_BSWGeneral.*

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.

[SWS_LinIf_00374] ⌈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.

[SWS_LinIf_00375] ⌈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. ⌋(SRS_BSW_00167)

10.3 LinIf_Configuration

The Figure 19 depicts the LIN Interface configuration.

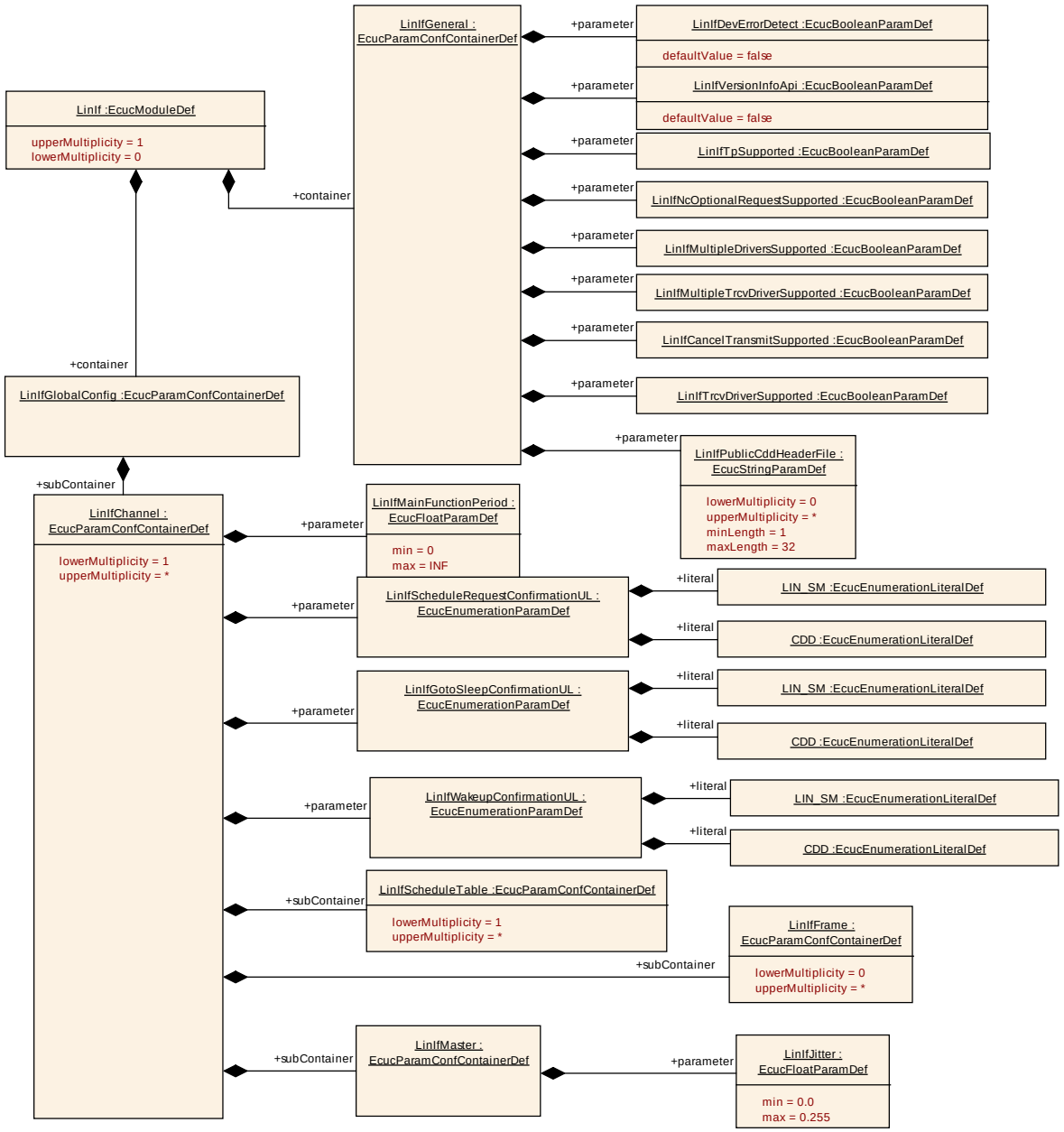

Figure 19 – LIN Interface configuration

10.3.1 LinIf

SWS Item	**ECUC_LinIf_00370 :**
Module Name	*LinIf*
Module Description	Configuration of the LinIf (LIN Interface) module.
Post-Build Variant Support	true
Supported Config Variants	VARIANT-LINK-TIME, VARIANT-POST-BUILD, VARIANT-PRE-COMPILE

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
LinIfGeneral	1	This container contains the general parameters of LIN Interface module.
LinIfGlobalConfig	1	This container contains the global configuration parameters of

		the LinIf.

10.3.2 LinIfGlobalConfig

SWS Item	**ECUC_LinIf_00020 :**
Container Name	LinIfGlobalConfig
Description	This container contains the global configuration parameters of the LinIf.
Configuration Parameters	

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
LinIfChannel	1..*	Describes each LIN channel the LinIf is connected to.

10.3.3 LinIfGeneral

SWS Item	**ECUC_LinIf_00019 :**
Container Name	LinIfGeneral
Description	This container contains the general parameters of LIN Interface module.
Configuration Parameters	

SWS Item	**ECUC_LinIf_00001 :**		
Name	LinIfCancelTransmitSupported		
Parent Container	LinIfGeneral		
Description	Global Pre-Compile Switch to enable/disable the APIs LinIf_CancelTransmit/LinTp_CancelReceive.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_LinIf_00010 :**		
Name	LinIfDevErrorDetect		
Parent Container	LinIfGeneral		
Description	Switches the development error detection and notification on or off. • true: detection and notification is enabled. • false: detection and notification is disabled.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	

Scope / Dependency	scope: local

SWS Item	**ECUC_LinIf_00024 :**		
Name	LinIfMultipleDriversSupported		
Parent Container	LinIfGeneral		
Description	States if multiple drivers are supported by 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	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_LinIf_00025 :**		
Name	LinIfMultipleTrcvDriverSupported		
Parent Container	LinIfGeneral		
Description	States if multiple transceiver drivers are supported by 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	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_LinIf_00026 :**		
Name	LinIfNcOptionalRequestSupported		
Parent Container	LinIfGeneral		
Description	States if the node configuration commands Assign NAD and Conditional Change NAD are supported.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_LinIf_00631 :**
Name	LinIfPublicCddHeaderFile
Parent Container	LinIfGeneral
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	--		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_LinIf_00045 :**		
Name	LinIfTpSupported		
Parent Container	LinIfGeneral		
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	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_LinIf_00635 :**		
Name	LinIfTrcvDriverSupported		
Parent Container	LinIfGeneral		
Description	States if transceiver driver support is 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	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_LinIf_00053 :**		
Name	LinIfVersionInfoApi		
Parent Container	LinIfGeneral		
Description	Switches the LinIf_GetVersionInfo function ON or OFF.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

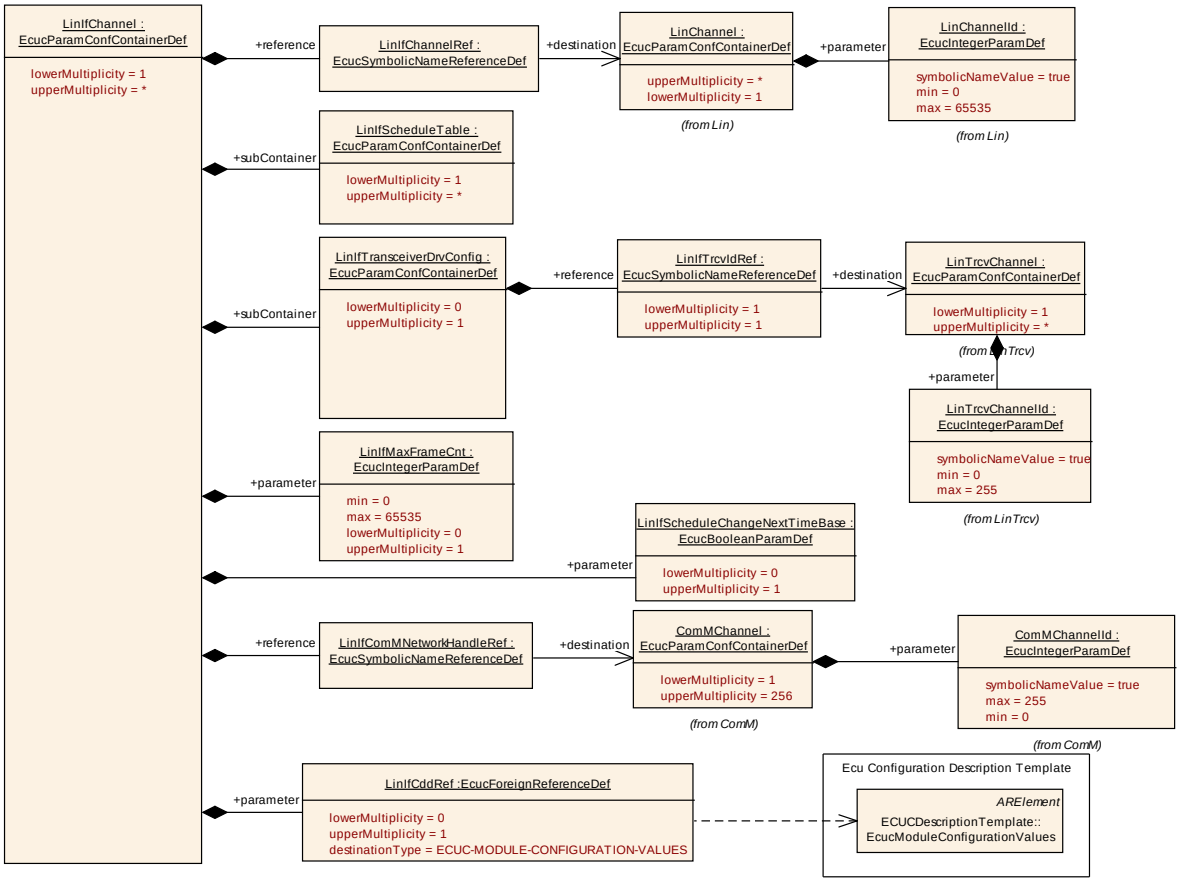

Figure 20 – LIN Interface Channel configuration

10.3.4 LinIfChannel

SWS Item	**ECUC_LinIf_00364 :**		
Container Name	LinIfChannel		
Description	Describes each LIN channel the LinIf is connected to.		
Post-Build Variant Multiplicity	false		
Multiplicity Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE, VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Configuration Parameters			

SWS Item	**ECUC_LinIf_00601 :**	
Name	LinIfGotoSleepConfirmationUL	
Parent Container	LinIfChannel	
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 Driver
	LIN_SM	LIN State Manager
Post-Build Variant	true	

Value			
Value Configuration Class	***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	**ECUC_LinIf_00639 :**		
Name	LinIfMainFunctionPeriod		
Parent Container	LinIfChannel		
Description	Defines the interval of calls to main functions per channel in seconds.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	]0 .. INF[		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_LinIf_00636 :**		
Name	LinIfMaxFrameCnt		
Parent Container	LinIfChannel		
Description	Maximum number of Frames. This parameter is needed only in case of post-build loadable implementation using static memory allocation.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_LinIf_00640 :**
Name	LinIfScheduleChangeNextTimeBase
Parent Container	LinIfChannel
Description	Enables/disables the switch to a new schedule table at the start of the next time base after status check. True: LinIf selects a new schedule table in next main function.
Multiplicity	0..1
Type	EcucBooleanParamDef
Default value	--
Post-Build Variant Multiplicity	false

Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_LinIf_00600 :**		
Name	LinIfScheduleRequestConfirmationUL		
Parent Container	LinIfChannel		
Description	This parameter defines the upper layer (UL) module to which the confirmation of the successfully performed schedule table change shall be sent.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	CDD	Complex Driver	
	LIN_SM	LIN State Manager	
Post-Build Variant Value	true		
Value Configuration Class	***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	**ECUC_LinIf_00069 : (Obsolete)**		
Name	LinIfStartupState		
Parent Container	LinIfChannel		
Description	Defines the state of each LIN channel after startup **Tags:** atp.Status=obsolete		
Multiplicity	0..1		
Type	EcucEnumerationParamDef		
Range	NORMAL	The state of the channel shall be LINIF_CHANNEL_OPERATIONAL after startup. **Tags:** atp.Status=obsolete	
	SLEEP	The state of the channel shall be LINIF_CHANNEL_SLEEP after startup. **Tags:** atp.Status=obsolete	
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_LinIf_00602 :**	
Name	LinIfWakeupConfirmationUL	
Parent Container	LinIfChannel	
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 Driver

	LIN_SM	LIN State Manager	
Post-Build Variant Value	true		
Value Configuration Class	***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	**ECUC_LinIf_00637 :**		
Name	LinIfCddRef		
Parent Container	LinIfChannel		
Description	Reference to the CDD module description. This parameter is only required when LinIfWakeupConfirmationUL, LinIfScheduleRequestConfirmationUL, and/or LinIfGotoSleepConfirmationUL is set to CDD.		
Multiplicity	0..1		
Type	Foreign reference to [ECUC-MODULE-CONFIGURATION-VALUES]		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_LinIf_00003 :**		
Name	LinIfChannelRef		
Parent Container	LinIfChannel		
Description	Reference to the channel definition in the LIN driver.		
Multiplicity	1		
Type	Symbolic name reference to [LinChannel]		
Post-Build Variant Value	false		
Value Configuration Class	***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	**ECUC_LinIf_00626 :**		
Name	LinIfComMNetworkHandleRef		
Parent Container	LinIfChannel		
Description	Unique handle to identify one LIN network. Reference to one of the network handles configured for the ComM.		
Multiplicity	1		
Type	Symbolic name reference to [ComMChannel]		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: local		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
LinIfFrame	0..*	Generic container for all types of LIN frames.
LinIfMaster	1	Each Master can only be connected to one physical channel. This could be compared to the Node parameter in a LDF file.
LinIfScheduleTable	1..*	Describes a schedule table. Each LinIfChannel may have several schedule tables. Each schedule table can only be connected to one channel. The SHORT-NAME of the LinIfScheduleTable container represents the symbolic name of the schedule table.
LinIfTransceiverDrvConfig	0..1	This container contains the configuration parameters of each underlying LIN Transceiver Driver.

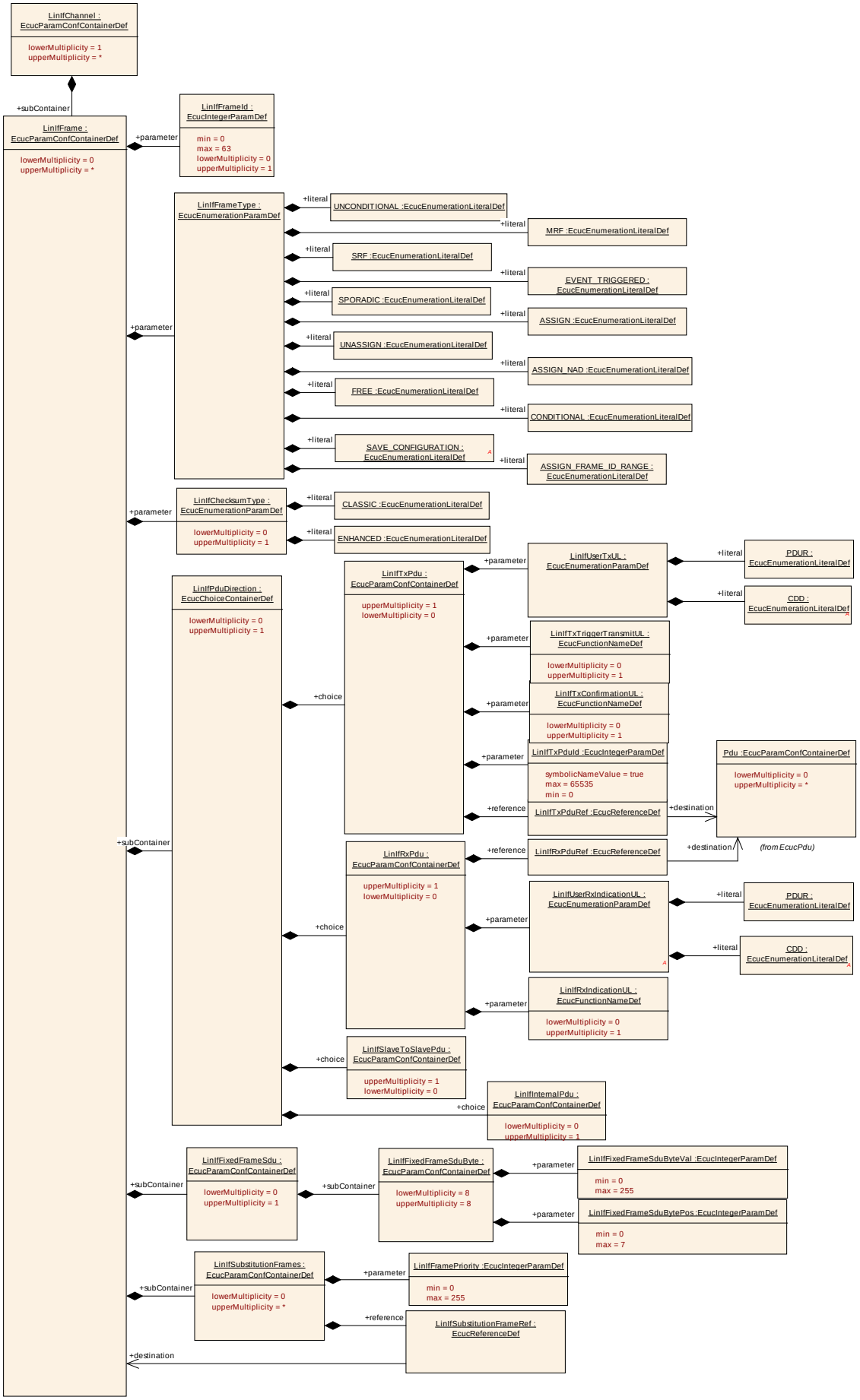
LinIfChannel : EcucParamConfContainerDef
lowerMultiplicity = 1
upperMultiplicity = *
+subContainer
LinIfFrame : EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *
+parameter
LinIfFrameId : EcucIntegerParamDef
min = 0
max = 63
lowerMultiplicity = 0
upperMultiplicity = 1
LinIfFrameType : 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
lowerMultiplicity = 0
upperMultiplicity = 1
CLASSIC :EcucEnumerationLiteralDef
ENHANCED :EcucEnumerationLiteralDef
LinIfPduDirection : EcucChoiceContainerDef
lowerMultiplicity = 0
upperMultiplicity = 1
+choice
+subContainer
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
+destination
Pdu :EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *
LinIfRxPdu : EcucParamConfContainerDef
upperMultiplicity = 1
lowerMultiplicity = 0
LinIfRxPduRef :EcucReferenceDef
(from EcucPdu)
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

Figure 21 – LIN Interface Frame configuration

10.3.5 LinIfFrame

SWS Item	**ECUC_LinIf_00367 :**		
Container Name	LinIfFrame		
Description	Generic container for all types of LIN frames.		
Post-Build Variant Multiplicity	true		
Multiplicity Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Configuration Parameters			

SWS Item	**ECUC_LinIf_00005 :**		
Name	LinIfChecksumType		
Parent Container	LinIfFrame		
Description	Type of checksum that the frame is using. This parameter is optional because in case of sporadic frames it should not be set.		
Multiplicity	0..1		
Type	EcucEnumerationParamDef		
Range	CLASSIC	Classic	
	ENHANCED	Enhanced	
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_LinIf_00638 :**		
Name	LinIfFrameId		
Parent Container	LinIfFrame		
Description	ID of the LIN frame. The Protected ID including parity is calculated by the generation tool.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 63		
Default value	--		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_LinIf_00017 :**
Name	LinIfFrameType
Parent Container	LinIfFrame
Description	Type of frame/slot. A sporadic slot may be used by a set of unconditional frames in

	the role of substitution 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	Master Request Frame	
	SAVE_CONFIGURATION	SaveConfiguration	
	SPORADIC	Sporadic slot	
	SRF	Slave Response Frame	
	UNASSIGN	UnassignFrameId	
	UNCONDITIONAL	Unconditional Frame	
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
LinIfFixedFrameSdu	0..1	In case this is a fixed frame this is the SDU (response). This container represents an eight byte array. The Byte order shall be MSB first.
LinIfPduDirection	0..1	Direction of the frame
LinIfSubstitutionFrames	0..*	List of sporadic frames that can be sent in a sporadic frame slot.

10.3.6 LinIfFixedFrameSdu

SWS Item	**ECUC_LinIf_00012 :**
Container Name	LinIfFixedFrameSdu
Description	In case this is a fixed frame this is the SDU (response). This container represents an eight byte array. The Byte order shall be MSB first.
Configuration Parameters	

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

10.3.7 LinIfFixedFrameSduByte

SWS Item	**ECUC_LinIf_00013 :**
Container Name	LinIfFixedFrameSduByte
Description	This container represents a byte within the 8 byte array.

Configuration Parameters

SWS Item	**ECUC_LinIf_00014 :**		
Name	LinIfFixedFrameSduBytePos		
Parent Container	LinIfFixedFrameSduByte		
Description	Index of the Byte in the SDU (response) 8 byte array.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 7		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_LinIf_00015 :**		
Name	LinIfFixedFrameSduByteVal		
Parent Container	LinIfFixedFrameSduByte		
Description	Byte value in the SDU (response) 8-byte array.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers

10.3.8 LinIfPduDirection

SWS Item	**ECUC_LinIf_00027 :**
Choice container Name	LinIfPduDirection
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.
LinIfTxPdu	0..1	represents a transmitted PDU/frame

10.3.9 LinIfSubstitutionFrames

SWS Item	**ECUC_LinIf_00042 :**

Container Name	LinIfSubstitutionFrames
Description	List of sporadic frames that can be sent in a sporadic frame slot.
Configuration Parameters	

SWS Item	**ECUC_LinIf_00513 :**		
Name	LinIfFramePriority		
Parent Container	LinIfSubstitutionFrames		
Description	Priority of sporadic frame.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_LinIf_00041 :**		
Name	LinIfSubstitutionFrameRef		
Parent Container	LinIfSubstitutionFrames		
Description	Reference to an unconditional Frame that is used as sporadic frame.		
Multiplicity	1		
Type	Reference to [LinIfFrame]		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers

10.3.10 LinIfRxPdu

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

SWS Item	**ECUC_LinIf_00055 :**
Name	LinIfRxIndicationUL
Parent Container	LinIfRxPdu
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	--		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Value Configuration Class	***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	**ECUC_LinIf_00610 :**		
Name	LinIfUserRxIndicationUL		
Parent Container	LinIfRxPdu		
Description	This parameter defines the upper layer (UL) module to which the indication of the successfully received LinIfRxPdu has to be routed via <User_RxIndication>.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	CDD	Complex Driver	
	PDUR	Pdu Router	
Post-Build Variant Value	true		
Value Configuration Class	***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	**ECUC_LinIf_00036 :**		
Name	LinIfRxPduRef		
Parent Container	LinIfRxPdu		
Description	Reference to the PDU that is received in this frame.		
Multiplicity	1		
Type	Reference to [Pdu]		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

No Included Containers

10.3.11 LinIfTxPdu

SWS Item	**ECUC_LinIf_00049 :**
Container Name	LinIfTxPdu
Description	represents a transmitted PDU/frame
Configuration Parameters	

SWS Item	**ECUC_LinIf_00054 :**		
Name	LinIfTxConfirmationUL		
Parent Container	LinIfTxPdu		
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	--		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Value Configuration Class	***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	**ECUC_LinIf_00050 :**		
Name	LinIfTxPduId		
Parent Container	LinIfTxPdu		
Description	Identifier of the Pdu for the upper layer.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 65535		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_LinIf_00628 :**		
Name	LinIfTxTriggerTransmitUL		
Parent Container	LinIfTxPdu		
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	--		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Value Configuration Class	***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	**ECUC_LinIf_00609 :**		
Name	LinIfUserTxUL		
Parent Container	LinIfTxPdu		
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 Driver	
	PDUR	Pdu Router	
Post-Build Variant Value	true		
Value Configuration Class	***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	**ECUC_LinIf_00051 :**		
Name	LinIfTxPduRef		
Parent Container	LinIfTxPdu		
Description	Reference to the PDU that is transmitted in this frame.		
Multiplicity	1		
Type	Reference to [Pdu]		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

No Included Containers

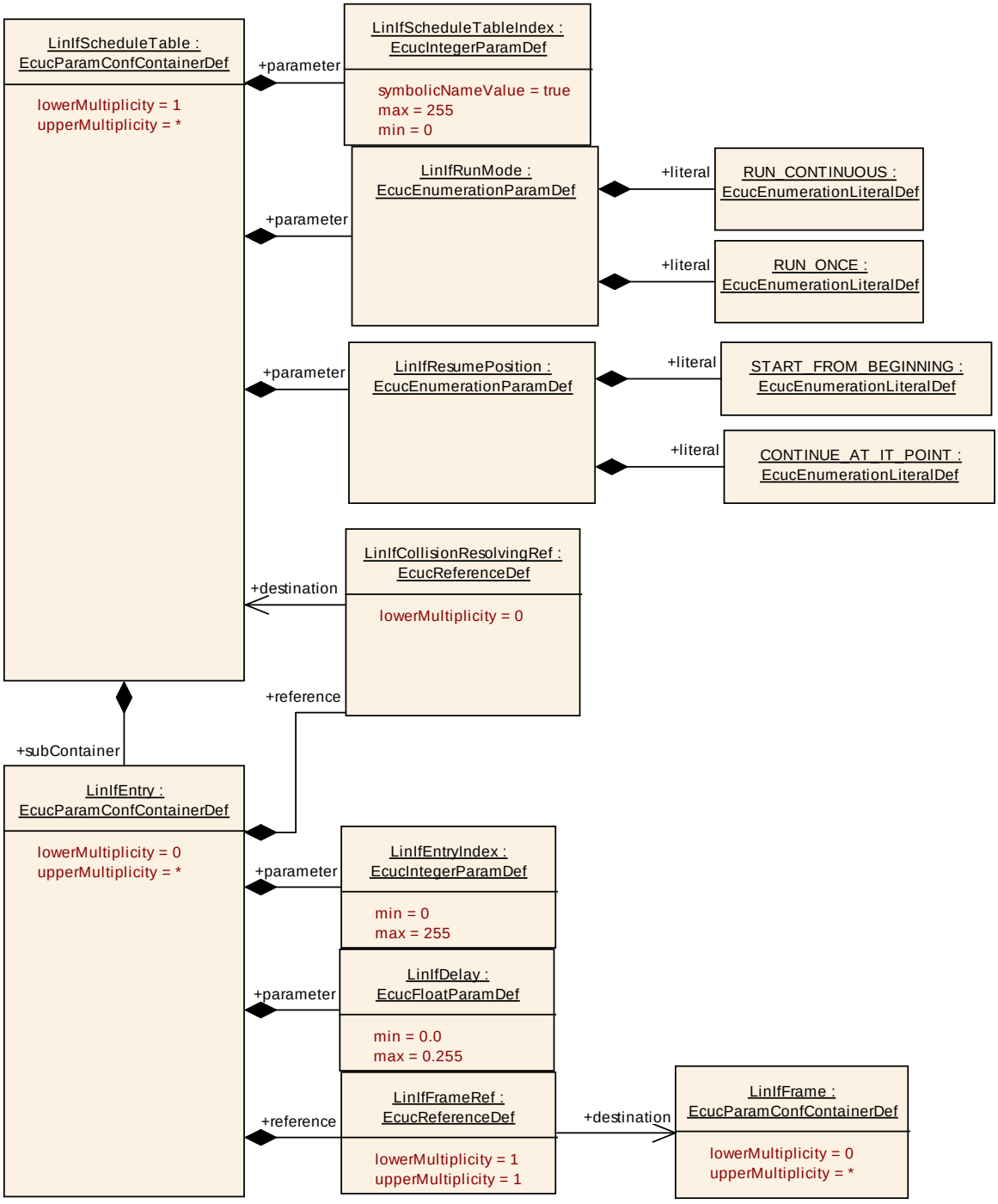

Figure 22 – LIN Interface Schedule Table configuration

10.3.12 LinIfScheduleTable

SWS Item	**ECUC_LinIf_00365 :**
Container Name	LinIfScheduleTable
Description	Describes a schedule table. Each LinIfChannel may have several schedule tables. Each schedule table can only be connected to one channel. The SHORT-NAME of the LinIfScheduleTable container represents the symbolic name of the schedule table.

Post-Build Variant Multiplicity	true		
Multiplicity Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Configuration Parameters			

SWS Item	**ECUC_LinIf_00033 :**		
Name	LinIfResumePosition		
Parent Container	LinIfScheduleTable		
Description	Defines where a RUN_CONTINUOUS schedule table shall proceed in case it has been interrupted by a RUN_ONCE table.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	CONTINUE_AT_IT_POINT	Continue schedule table where it was interrupted.	
	START_FROM_BEGINNING	Start schedule table from the beginning.	
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_LinIf_00034 :**		
Name	LinIfRunMode		
Parent Container	LinIfScheduleTable		
Description	The schedule table can be executed in two different modes.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	RUN_CONTINUOUS	--	
	RUN_ONCE	--	
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_LinIf_00037 :**		
Name	LinIfScheduleTableIndex		
Parent Container	LinIfScheduleTable		
Description	This is the unique index used by upper layers to identify a schedule. Note that the NULL_SCHEDULE for each channel must have index 0.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 255		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

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	**ECUC_LinIf_00366 :**		
Container Name	LinIfEntry		
Description	Describes an entry in the schedule table (also known as Frame Slot).		
Post-Build Variant Multiplicity	true		
Multiplicity Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Configuration Parameters			

SWS Item	**ECUC_LinIf_00009 :**		
Name	LinIfDelay		
Parent Container	LinIfEntry		
Description	Delay to next entry in schedule table in seconds.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	[0 .. 0.255]		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_LinIf_00011 :**		
Name	LinIfEntryIndex		
Parent Container	LinIfEntry		
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	--		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_LinIf_00007 :**
Name	LinIfCollisionResolvingRef
Parent Container	LinIfEntry
Description	Reference to the schedule table, which resolves the collision. This parameter is only used if the referenced frames are event triggered frames.

Multiplicity	0..1		
Type	Reference to [LinIfScheduleTable]		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_LinIf_00016 :**		
Name	LinIfFrameRef		
Parent Container	LinIfEntry		
Description	Reference to the frames that belong to this schedule table entry.		
Multiplicity	1		
Type	Reference to [LinIfFrame]		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers

10.3.14 LinIfMaster

SWS Item	**ECUC_LinIf_00512 :**
Container Name	LinIfMaster
Description	Each Master can only be connected to one physical channel. This could be compared to the Node parameter in a LDF file.
Configuration Parameters	

SWS Item	**ECUC_LinIf_00629 :**		
Name	LinIfJitter		
Parent Container	LinIfMaster		
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	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

10.3.15 LinIfSlaveToSlavePdu

SWS Item	**ECUC_LinIf_00040 :**

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

No Included Containers

10.3.16 LinIfInternalPdu

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

No Included Containers

10.3.17 LinIfTransceiverDrvConfig

SWS Item	**ECUC_LinIf_00046 :**
Container Name	LinIfTransceiverDrvConfig
Description	This container contains the configuration parameters of each underlying LIN Transceiver Driver.
Configuration Parameters	

SWS Item	**ECUC_LinIf_00047 :**		
Name	LinIfTrcvIdRef		
Parent Container	LinIfTransceiverDrvConfig		
Description	Logical handle of the underlying LIN transceiver to be served by the LIN Interface.		
Multiplicity	1		
Type	Symbolic name reference to [LinTrcvChannel]		
Post-Build Variant Value	false		
Value Configuration Class	***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.

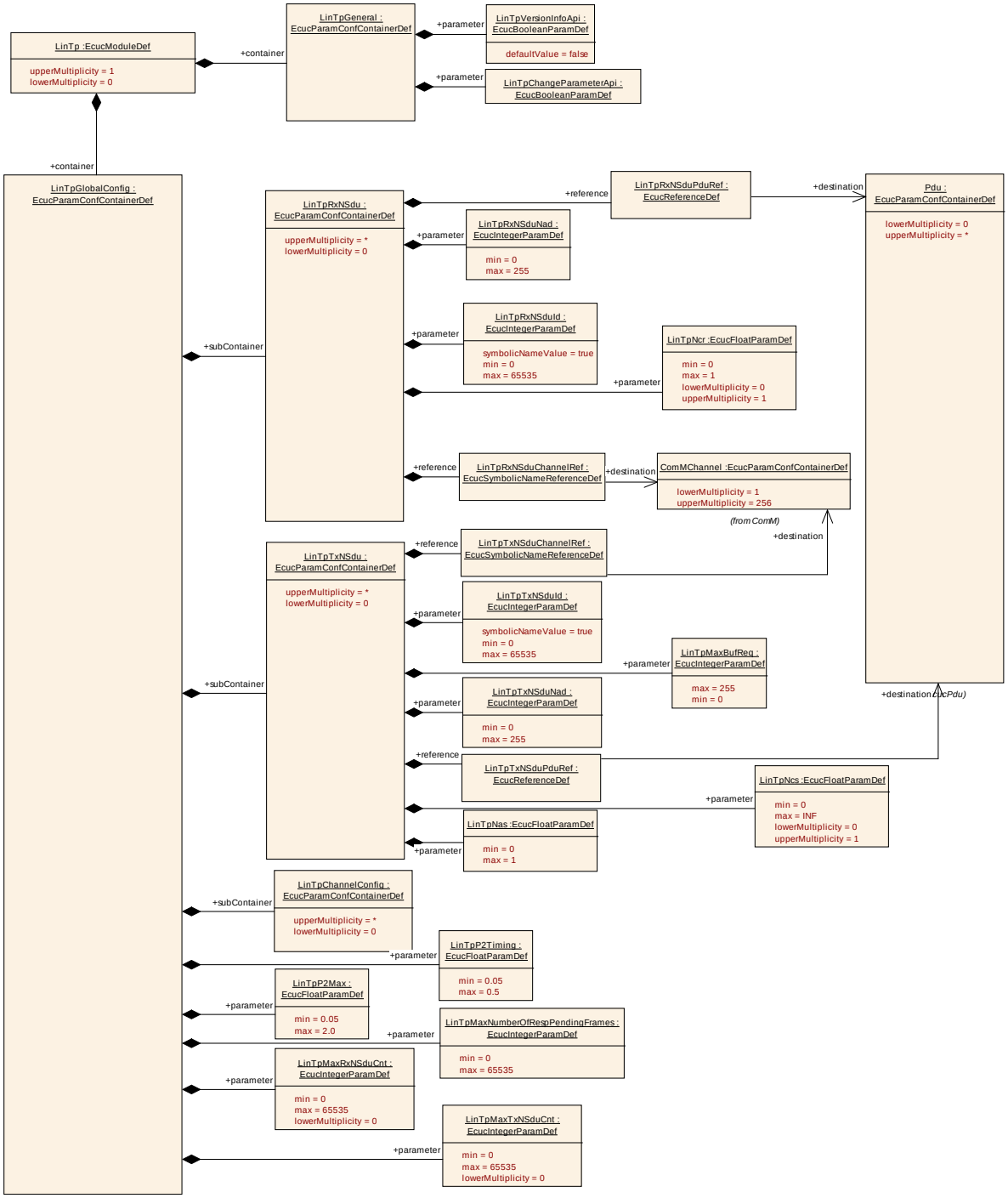

Figure 23 – LIN Transport Protocol configuration

10.4.1 LinTp

SWS Item	**ECUC_LinTp_00425 :**
Module Name	*LinTp*
Module Description	Configuration of the LIN Transport Protocol.
Post-Build Variant Support	true
Supported Config Variants	VARIANT-LINK-TIME, VARIANT-POST-BUILD, VARIANT-PRE-COMPILE

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 parameters of the LinTp.

10.4.2 LinTpGeneral

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

SWS Item	**ECUC_LinTp_00638 :**		
Name	LinTpChangeParameterApi		
Parent Container	LinTpGeneral		
Description	This parameter, if set to true, enables the LinTp_ChangeParameterRequest Api for this Module.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_LinTp_00068 :**		
Name	LinTpVersionInfoApi		
Parent Container	LinTpGeneral		
Description	Switches the LinTp_GetVersionInfo function ON or OFF.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

10.4.3 LinTpGlobalConfig

SWS Item	**ECUC_LinTp_00056 :**
Container Name	LinTpGlobalConfig
Description	This container contains the global configuration parameters of the LinTp.

Configuration Parameters

SWS Item	**ECUC_LinTp_00624 :**		
Name	LinTpMaxNumberOfRespPendingFrames		
Parent Container	LinTpGlobalConfig		
Description	Configures the maximum number of allowed response pending frames.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	*Pre-compile time*	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_LinTp_00635 :**		
Name	LinTpMaxRxNSduCnt		
Parent Container	LinTpGlobalConfig		
Description	Maximum number of NSdus. This parameter is needed only in case of post-build loadable implementation using static memory allocation.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	*Pre-compile time*	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_LinTp_00636 :**		
Name	LinTpMaxTxNSduCnt		
Parent Container	LinTpGlobalConfig		
Description	Maximum number of NSdus. This parameter is needed only in case of post-build loadable implementation using static memory allocation.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	*Pre-compile time*	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_LinTp_00622 :**
Name	LinTpP2Max
Parent Container	LinTpGlobalConfig
Description	P2*max timeout when a response pending frame is expected in seconds. Note that the minimum value of LinTpP2Max shall be more than or equal to the value of LinTpP2Timing.
Multiplicity	1
Type	EcucFloatParamDef

Range	[0.05 .. 2]		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_LinTp_00625 :**		
Name	LinTpP2Timing		
Parent Container	LinTpGlobalConfig		
Description	Definition of the P2max timeout observation parameter in seconds.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	[0.05 .. 0.5]		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
LinTpChannelConfig	0..*	This container contains the channel specific configuration parameters of LinTp.
LinTpRxNSdu	0..*	This container exists once for each received N-SDU on any channel the node is connected to. This N-SDU produces meta data items of type LIN_NAD_8.
LinTpTxNSdu	0..*	This container exists once for each transmitted N-SDU on any channel the node is connected to. This N-SDU consumes meta data items of type LIN_NAD_8.

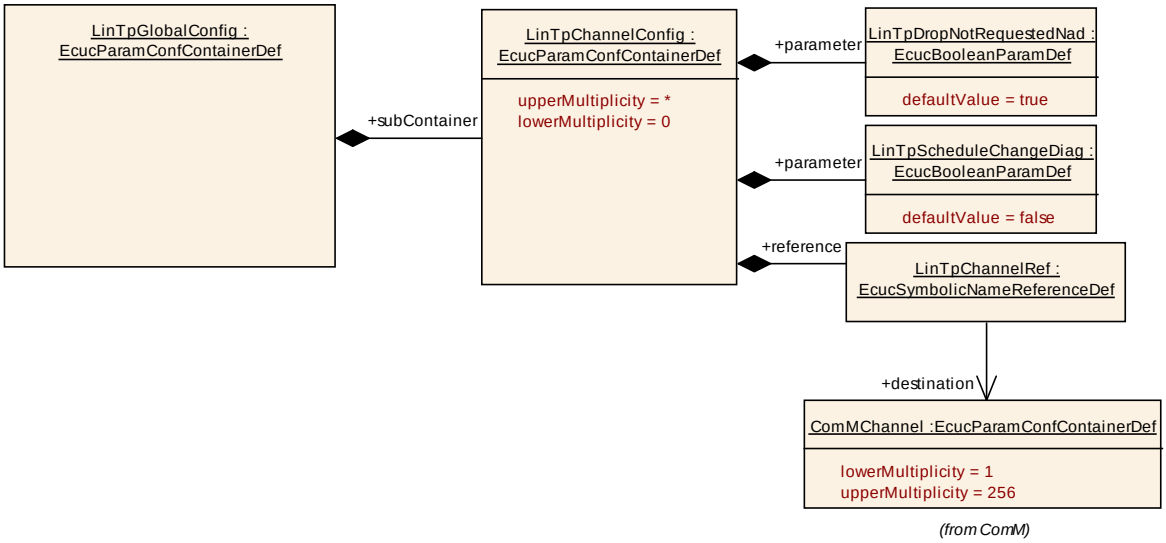

Figure 24 – LIN Transport Protocol Channel configuration

10.4.4 LinTpChannelConfig

SWS Item	**ECUC_LinTp_00071 :**		
Container Name	LinTpChannelConfig		
Description	This container contains the channel specific configuration parameters of LinTp.		
Post-Build Variant Multiplicity	true		
Multiplicity Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Configuration Parameters			

SWS Item	**ECUC_LinTp_00072 :**		
Name	LinTpDropNotRequestedNad		
Parent Container	LinTpChannelConfig		
Description	Configures if TP Frames of not requested LIN-Slaves are dropped or not. TRUE: Drop TP Frames of not requested LIN-Slaves FALSE: Keep TP Frames of not requested LIN-Slaves		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	true		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_LinTp_00070 :**		
Name	LinTpScheduleChangeDiag		
Parent Container	LinTpChannelConfig		
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		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_LinTp_00073 :**		
Name	LinTpChannelRef		
Parent Container	LinTpChannelConfig		
Description	Index of the channel this LinTp channel belongs to.		
Multiplicity	1		
Type	Symbolic name reference to [ComMChannel]		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers

10.4.5 LinTpRxNSdu

SWS Item	**ECUC_LinTp_00428 :**		
Container Name	LinTpRxNSdu		
Description	This container exists once for each received N-SDU on any channel the node is connected to. This N-SDU produces meta data items of type LIN_NAD_8.		
Post-Build Variant Multiplicity	true		
Multiplicity Configuration Class	*Pre-compile time*	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Configuration Parameters			

SWS Item	**ECUC_LinTp_00632 :**		
Name	LinTpNcr		
Parent Container	LinTpRxNSdu		
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	--		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	*Pre-compile time*	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	*Pre-compile time*	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_LinTp_00061 :**		
Name	LinTpRxNSduId		
Parent Container	LinTpRxNSdu		
Description	The identifier of the Transport Protocol message. This ID will be used by upper layers to call LinTp_ChangeParameter and LinTp_CancelReceive.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 65535		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	*Pre-compile time*	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_LinTp_00062 :**
Name	LinTpRxNSduNad

Parent Container	LinTpRxNSdu		
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	--		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_LinTp_00060 :**		
Name	LinTpRxNSduChannelRef		
Parent Container	LinTpRxNSdu		
Description	Index of the channel this N-SDU belongs to.		
Multiplicity	1		
Type	Symbolic name reference to [ComMChannel]		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_LinTp_00063 :**		
Name	LinTpRxNSduPduRef		
Parent Container	LinTpRxNSdu		
Description	Reference to the global PDU		
Multiplicity	1		
Type	Reference to [Pdu]		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

No Included Containers

10.4.6 LinTpTxNSdu

SWS Item	**ECUC_LinTp_00511 :**		
Container Name	LinTpTxNSdu		
Description	This container exists once for each transmitted N-SDU on any channel the node is connected to. This N-SDU consumes meta data items of type LIN_NAD_8.		
Post-Build Variant Multiplicity	true		
Multiplicity Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Configuration Parameters			

SWS Item	**ECUC_LinTp_00637 :**		
Name	LinTpMaxBufReq		
Parent Container	LinTpTxNSdu		
Description	This parameter defines the maximum number of times the LinTp should request upper layer for the Tx Buffer. It is also used to limit the number of retries for PduR_LinTpCopyTxData when no timer is active.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_LinTp_00633 :**		
Name	LinTpNas		
Parent Container	LinTpTxNSdu		
Description	Value in seconds 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	--		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_LinTp_00634 :**		
Name	LinTpNcs		
Parent Container	LinTpTxNSdu		
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	--		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_LinTp_00065 :**
Name	LinTpTxNSduId
Parent Container	LinTpTxNSdu

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	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_LinTp_00066 :**		
Name	LinTpTxNSduNad		
Parent Container	LinTpTxNSdu		
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	--		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_LinTp_00064 :**		
Name	LinTpTxNSduChannelRef		
Parent Container	LinTpTxNSdu		
Description	Index of the channel this N-SDU belongs to.		
Multiplicity	1		
Type	Symbolic name reference to [ComMChannel]		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_LinTp_00067 :**		
Name	LinTpTxNSduPduRef		
Parent Container	LinTpTxNSdu		
Description	Reference to the global PDU		
Multiplicity	1		
Type	Reference to [Pdu]		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

No Included Containers

10.5 Published Information

For details refer to the chapter 10.3 “Published Information” in *SWS_BSWGeneral.*

11 Not applicable requirements

[SWS_LinIf_99999] ⌈These requirements are not applicable to this specification.⌋ (SRS_BSW_00432, SRS_BSW_00433, SRS_BSW_00417, SRS_BSW_00359, SRS_BSW_00360, SRS_BSW_00331, SRS_BSW_00010, SRS_BSW_00333, SRS_BSW_00003, SRS_BSW_00321, SRS_BSW_00341, SRS_BSW_00334, SRS_BSW_00437, SRS_BSW_00422, SRS_BSW_00440, SRS_BSW_00439)