Document Title	Specification of LIN Interface
Document Category	Software specification (SWS)
Document UID	73
Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R26-11

Document Change History			
Date	**Release**	**Changed by**	**Description**
2026-11-26	R26-11	AUTOSAR Release Management	• Resolved inconsistencies between ComM, LinSM and LinIf (both Master and Slave nodes) • Removed GetVersionInfo API and parameters • Changed the error category for Extended Production Errors to Runtime Errors • Added [SWS_LinIf_CONSTR_00903] for clarification. • Editorial changes
2025-11-27	R25-11	AUTOSAR Release Management	• Removal of obsolete elements • Editorial changes
2024-11-27	R24-11	AUTOSAR Release Management	• Support IEEE 1722 Tunneling – Changed the upper layer (in the PDU path) from PduR / CDD to LSduR • Editorial changes
2023-11-23	R23-11	AUTOSAR Release Management	• Added Extended Production Errors regarding `LinTp` timeouts and relevant errors • Added a note after [SWS_LinIf_00503] for clarification on implementation of `LinIf_CheckWakeup` API • Clarification of “Available via: Configurable” in API tables (Header File Cleanup) • Refined configuration structure • Editorial changes (incl. correcting typos in spec. items)

▽

△

2022-11-24	R22-11	AUTOSAR Release Management	• Changed upper limit of LinTpP2Timing and LinTpP2Max • Renamed the arguments "Schedule" ([SWS_LinIf_00202] LinIf_ScheduleRequest) and "schedule" ([SWS_LinIf_00520] <User>_ScheduleRequest-Confirmation)
2021-11-25	R21-11	AUTOSAR Release Management	• Added the API table of <User>_GotoSleepIndication • Removed inconsistent requirements regarding availability of LinIf_CheckWakeup and LinIf_WakeupConfirmation APIs
2020-11-30	R20-11	AUTOSAR Release Management	• Corrected behavior of LinTp-originated schedule table switch (with limitations) and LinTp P2 timeout monitoring • Updated the structure and tables of the error sections • Corrected header filename for the imported type LinTrcv_TrcvModeType • Changed Service ID of LinTp_Transmit API • Reformulated negative requirements which are not testable
2019-11-28	R19-11	AUTOSAR Release Management	• No content changes • Changed Document Status from Final to published
2018-10-31	4.4.0	AUTOSAR Release Management	• Changed TP Timers to channel specific • Removed dummy APIs (LinIf_CancelTransmit etc.) and replaced ChannelId with LinIfChannel.ShortName • Replaced references to LIN 2.1 by ISO 17987:2016 (with no functional modification) • LIN Slave Support (CONC_631) • Header file cleanup

▽

			• Minor corrections / clarifications / editorial changes; For details please refer to the ChangeDocumentation
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
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 LinIfFrame.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.
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 Mode Manager 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
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 APIs: `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

This work (specification and/or software implementation) and the material contained in it, as released by AUTOSAR, is for the purpose of information only. AUTOSAR and the companies that have contributed to it shall not be liable for any use of the work.

The material contained in this work is protected by copyright and other types of intellectual property rights. The commercial exploitation of the material contained in this work requires a license to such intellectual property rights.

This work may be utilized or reproduced without any modification, in any form or by any means, for informational purposes only. For any other purpose, no part of the work may be utilized or reproduced, in any form or by any means, without permission in writing from the publisher.

The work has been developed for automotive applications only. It has neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Table of Contents

1 Introduction and functional overview 12
1.1 Architectural overview . . . 13
1.2 Functional overview . . . 15
2 Acronyms and Abbreviations 16
3 Related documentation 18
3.1 Input documents & related standards and norms . . . 18
3.2 Related specification . . . 19
4 Constraints and assumptions 20
4.1 Limitations . . . 20
4.2 Applicability to car domains . . . 21
4.3 Clarification about other LIN standards . . . 21
5 Dependencies to other modules 22
5.1 Upper layers . . . 22
5.1.1 L-SDU Router and CDD . . . 22
5.1.2 PDU Router . . . 22
5.1.3 Bus Mirroring . . . 22
5.1.4 LIN State Manager . . . 23
5.1.5 BSW Mode Manager . . . 23
5.1.6 AUTOSAR COM . . . 23
5.2 Lower layers . . . 23
5.2.1 LIN Driver . . . 23
5.2.2 LIN Transceiver Driver . . . 24
5.3 File structure . . . 25
5.3.1 Header file structure . . . 25
6 Requirements Tracing 26
7 Functional specification 28
7.1 Frame Transfer . . . 28
7.1.1 Frame types . . . 28
7.1.1.1 Unconditional frame . . . 29
7.1.1.2 Event-triggered frame . . . 29
7.1.1.3 Sporadic frame (Master only) . . . 30
7.1.1.4 Diagnostic Frames MRF and SRF . . . 31
7.1.1.5 Reserved frames . . . 31
7.1.2 Frame reception . . . 32
7.1.2.1 Frame reception in master nodes . . . 32
7.1.2.2 Frame reception in slave nodes . . . 34
7.1.3 Frame transmission . . . 36
7.1.3.1 Frame transmission in master nodes . . . 36

7.1.3.2 Frame transmission in slave nodes . . . 37
7.1.4 Slave-to-slave communication (Master only) . . . 40
7.1.4.1 Header . . . 40
7.1.4.2 Response . . . 40
7.1.4.3 Status check . . . 40
7.1.5 Irrelevant communication (Slave only) . . . 40
7.2 Schedules (Master only) . . . 41
7.2.1 Schedule table manager . . . 41
7.3 Main function . . . 43
7.4 Network management . . . 43
7.4.1 Node Management . . . 43
7.4.1.1 LIN Interface state-machine . . . 43
7.4.1.2 LIN channel sub-state-machine . . . 44
7.4.2 Go to sleep process . . . 46
7.4.2.1 Go to sleep process in master nodes . . . 46
7.4.2.2 Go to sleep process in slave nodes . . . 47
7.4.3 Wake up process . . . 49
7.4.3.1 Wake up process in master nodes . . . 49
7.4.3.2 Wake up process in slave nodes . . . 50
7.5 Status Management . . . 51
7.5.1 response_error signal (Slave only) . . . 52
7.6 Diagnostics and Node configuration . . . 52
7.6.1 Node configuration in master nodes . . . 53
7.6.1.1 Node Configuration services . . . 53
7.6.1.2 Node Configuration in Schedule Table . . . 54
7.6.2 Node configuration in slave nodes . . . 54
7.6.2.1 Node Model . . . 55
7.6.2.2 Node Configuration services . . . 55
7.6.2.3 Diagnostic Frame Dispatcher . . . 56
7.6.2.4 Node Configuration Handler . . . 57
7.6.3 Diagnostics – Transport Protocol . . . 59
7.6.3.1 Schedule requests in master nodes . . . 59
7.6.3.2 State-machine . . . 60
7.6.3.3 LIN TP transmission . . . 62
7.6.3.4 LIN TP transmission error . . . 63
7.6.3.5 LIN TP reception . . . 65
7.6.3.6 Unavailability of receive buffer . . . 66
7.6.3.7 LIN TP reception error . . . 68
7.7 Handling multiple channels and drivers . . . 71
7.7.1 Multiple channels . . . 71
7.7.2 Multiple LIN drivers . . . 71
7.7.3 Multiple LIN transceiver drivers . . . 72
7.8 Error classification . . . 73

7.8.1 Development Errors . . . 73
7.8.2 Runtime Errors . . . 74
7.8.3 Production Errors . . . 74
8 API specification 75
8.1 Imported types . . . 75
8.1.1 Standard types . . . 75
8.2 Type definitions . . . 76
8.2.1 LinIf_SchHandleType . . . 76
8.2.2 LinIf_ConfigType . . . 76
8.2.3 LinTp_ConfigType . . . 77
8.2.4 LinTp_Mode . . . 77
8.3 Function definitions . . . 78
8.3.1 LinIf_Init . . . 78
8.3.2 LinIf_Transmit . . . 78
8.3.3 LinIf_ScheduleRequest . . . 80
8.3.4 LinIf_GotoSleep . . . 81
8.3.5 LinIf_Wakeup . . . 82
8.3.6 LinIf_SetTrcvMode . . . 83
8.3.7 LinIf_GetTrcvMode . . . 84
8.3.8 LinIf_GetTrcvWakeupReason . . . 85
8.3.9 LinIf_SetTrcvWakeupMode . . . 86
8.3.10 LinIf_GetPIDTable . . . 87
8.3.11 LinIf_SetPIDTable . . . 88
8.3.12 LinIf_GetConfiguredNAD . . . 89
8.3.13 LinIf_SetConfiguredNAD . . . 90
8.3.14 LinIf_CheckWakeup . . . 91
8.3.15 LinIf_EnableBusMirroring . . . 92
8.3.16 LinIf_WakeupConfirmation . . . 94
8.3.17 LinIf_HeaderIndication . . . 94
8.3.18 LinIf_RxIndication . . . 95
8.3.19 LinIf_TxConfirmation . . . 96
8.3.20 LinIf_LinErrorIndication . . . 97
8.3.21 LinTp_Init . . . 99
8.3.22 LinTp_Transmit . . . 99
8.3.23 LinTp_Shutdown . . . 101
8.3.24 LinTp_ChangeParameter . . . 102
8.4 Scheduled functions . . . 103
8.4.1 LinIf_MainFunction_<LinIfChannel.ShortName> . . . 103
8.5 Expected interfaces . . . 104
8.5.1 Mandatory Interfaces . . . 104
8.5.2 Optional interfaces . . . 104
8.5.3 Callout interfaces . . . 106
8.5.3.1 <User_ResponseErrorSignalChanged> . . . 106

8.5.3.2 <User_SaveConfigurationRequest> . . . 107
8.5.4 Configurable interfaces . . . 108
8.5.4.1 <User>_ScheduleRequestConfirmation . . . 108
8.5.4.2 <User>_GotoSleepConfirmation . . . 109
8.5.4.3 <User>_WakeupConfirmation . . . 109
8.5.4.4 <User>_GotoSleepIndication . . . 110
9 Sequence diagrams 111
9.1 Frame Transmission . . . 111
9.1.1 Frame transmission in master nodes . . . 111
9.1.2 Frame transmission in slave nodes . . . 113
9.2 Frame Reception . . . 114
9.2.1 Frame reception in master nodes . . . 114
9.2.2 Frame reception in slave nodes . . . 115
9.3 Slave-to-slave / Irrelevant communication . . . 116
9.3.1 Slave-to-slave communication in master nodes . . . 116
9.3.2 Irrelevant communication in slave nodes . . . 116
9.4 Sporadic frame (Master only) . . . 117
9.5 Event-triggered frame . . . 118
9.5.1 Event-triggered frame in master nodes . . . 118
9.5.1.1 With no answer . . . 119
9.5.1.2 With answer (No collision) . . . 120
9.5.1.3 With collision . . . 121
9.5.2 Event-triggered frame in slave nodes . . . 122
9.6 Transport Protocol message transmission . . . 123
9.7 Transport Protocol message reception . . . 125
9.8 Go to sleep process . . . 127
9.8.1 Go to sleep process in master nodes . . . 127
9.8.2 Go to sleep process in slave nodes . . . 129
9.9 Wake up request . . . 130
9.10 Internal wake-up . . . 130
10 Configuration specification 131
10.1 How to read this chapter . . . 131
10.2 Containers and configuration parameters . . . 131
10.2.1 Configuration Tool . . . 131
10.3 LinIf Configuration . . . 132
10.3.1 LinIf . . . 132
10.3.2 LinIfGlobalConfig . . . 133
10.3.3 LinIfGeneral . . . 134
10.3.4 LinIfChannel . . . 139
10.3.5 LinIfNodeType . . . 144
10.3.6 LinIfFrame . . . 145
10.3.7 LinIfFixedFrameSdu . . . 149

10.3.8 LinIfFixedFrameSduByte . . . 149
10.3.9 LinIfPduDirection . . . 150
10.3.10 LinIfSubstitutionFrames . . . 151
10.3.11 LinIfRxPdu . . . 153
10.3.12 LinIfTxPdu . . . 153
10.3.13 LinIfScheduleTable . . . 155
10.3.14 LinIfEntry . . . 157
10.3.15 LinIfMaster . . . 159
10.3.16 LinIfSlave . . . 160
10.3.17 LinIfNodeConfigurationIdentification . . . 161
10.3.18 LinIfSlaveToSlavePdu . . . 164
10.3.19 LinIfInternalPdu . . . 164
10.3.20 LinIfTransceiverDrvConfig . . . 165
10.4 LIN Transport Layer configuration . . . 166
10.4.1 LinTp . . . 168
10.4.2 LinTpGeneral . . . 168
10.4.3 LinTpGlobalConfig . . . 169
10.4.4 LinTpChannelConfig . . . 171
10.4.5 LinTpRxNSdu . . . 174
10.4.6 LinTpTxNSdu . . . 176
10.5 Configuration Constraints . . . 178
10.6 Published Information . . . 178
A Not applicable requirements 179
B Change history of AUTOSAR traceable items 180
B.1 Traceable item history of this document according to AUTOSAR Release R24-11 . . . 180
B.1.1 Added Specification Items in R24-11 . . . 180
B.1.2 Changed Specification Items in R24-11 . . . 181
B.1.3 Deleted Specification Items in R24-11 . . . 181
B.2 Traceable item history of this document according to AUTOSAR Release R25-11 . . . 182
B.2.1 Added Specification Items in R25-11 . . . 182
B.2.2 Changed Specification Items in R25-11 . . . 182
B.2.3 Deleted Specification Items in R25-11 . . . 182
B.3 Traceable item history of this document according to AUTOSAR Release R26-11 . . . 183
B.3.1 Added Specification Items in R26-11 . . . 183
B.3.2 Changed Specification Items in R26-11 . . . 183
B.3.3 Deleted Specification Items in R26-11 . . . 183
B.3.4 Added Constraints in R26-11 . . . 184
B.3.5 Changed Constraints in R26-11 . . . 184
B.3.6 Deleted Constraints in R26-11 . . . 184

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, LinTp). The LIN TP is a part of the LIN Interface.

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

This document is based on the ISO 17987 specifications [1]. It is assumed that the reader is familiar with the specifications. This document will not describe ISO 17987 LIN functionality again.

The LIN Interface module applies to ISO 17987 master and slave nodes (compatible with LIN 2.2 [2] and LIN 2.1 [3] master nodes). The LIN implementation in AUTOSAR deviates from the ISO 17987 specifications 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 nodes together with the AUTOSAR LIN implementation (i.e. the LIN Interface).

The LIN Interface module (LinIf) is designed to be hardware independent. The interfaces to upper (L-SDU Router for LinIf part and PDU Router for LinTp part) 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 [4], the LIN Interface is located within the BSW architecture as shown below (Figure 1.1). In this example, the LIN Interface is connected to two LIN Drivers. However, one LIN Driver is the most common configuration.

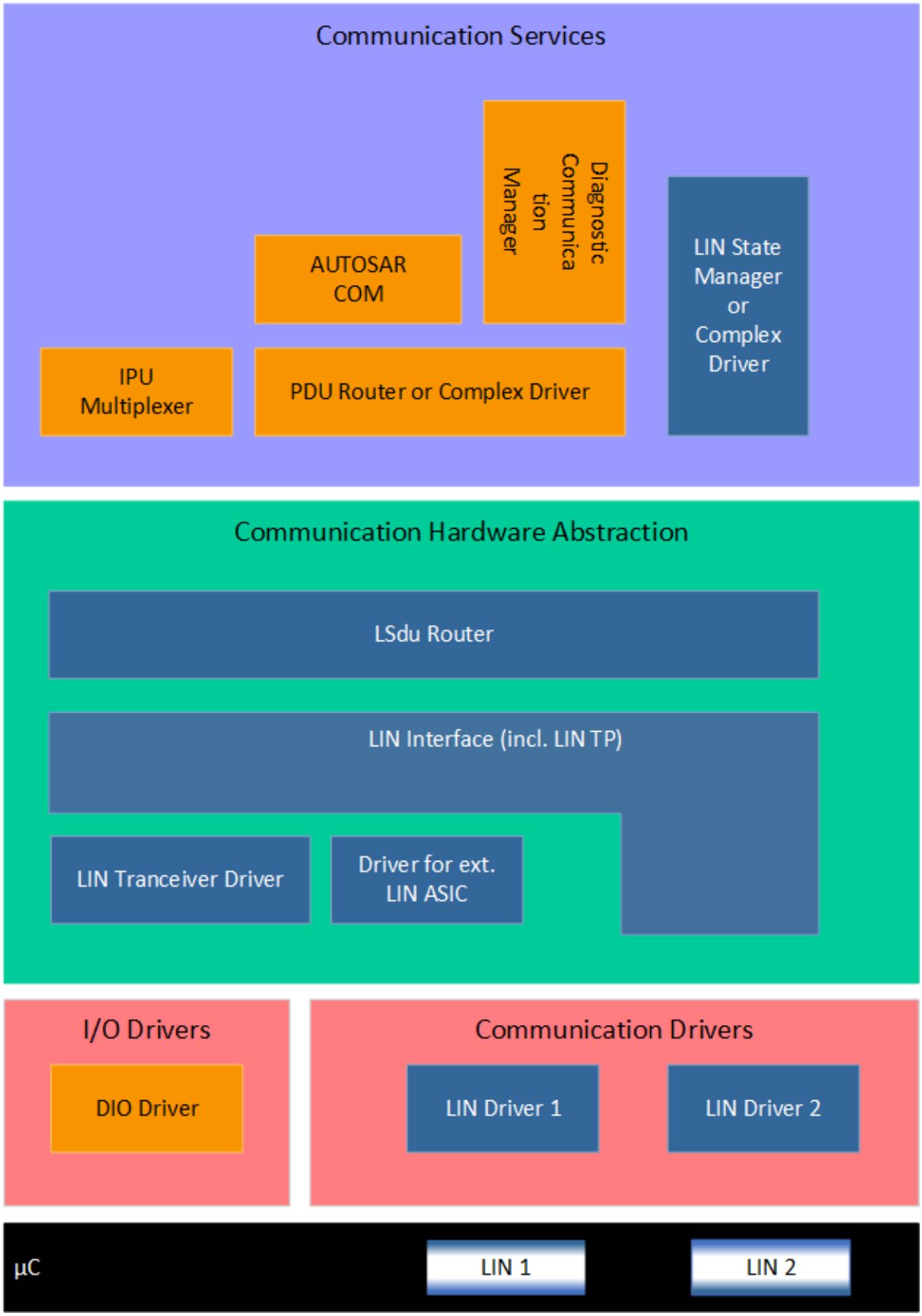


Figure 1.1: AUTOSAR BSW software architecture (LIN relevant modules)

1.2 Functional overview

The LIN Interface is responsible for providing ISO 17987 LIN functionality. This means:

- Executing the currently selected schedule for each LIN bus the ECU is connected to, as a master node (transmitting headers and transmitting / receiving responses).
- Switching schedule tables of master nodes 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.
- Node configuration and identification services of slave nodes.

2 Acronyms and Abbreviations

In addition to the acronyms and abbreviations found in the ISO 17987 LIN specifications [1], the following acronyms and abbreviations (that are not included in the [5, AUTOSAR glossary]) are used throughout this document. Some terms already defined in the ISO 17987 specifications have also been defined here in order to provide more clarification, especially for terms used very often in this document.

Abbreviation / Acronym:	Description:
CF	Consecutive Frame in LIN TP
FF	First Frame in LIN TP
ID	Identifier
LDF	LIN Description File
LIN TP (LinTp)	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 17987-2 [6]).
N_Cr	Time until reception of the next `Consecutive Frame` N-PDU (see ISO 17987-2 [6]).
N_Cs	Time until transmission of the next `Consecutive Frame` N-PDU (see ISO 17987-2 [6]).
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
SID	Service Identifier (of node configuration service)
SF	Single Frame in LIN TP
SRF	Slave Response Frame
SRS	Software Requirement Specification
TX	Transmission

Table 2.1: Acronyms and abbreviations used in the scope of this Document

Terms:	Description:
Bus idle timeout	Lapse of time duration with no bus activity
Irrelevant frame	From a LIN slave node point of view, there exist 3 different directions of frames on the LIN bus: Response transmitted by the slave, Response received by the slave and Response that is ignored by the slave (i.e. communication between master and another slave or between two other slaves). These ignored frames are named `Irrelevant frames` in this specification. This is not described explicitly in the ISO 17987 specifications.
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_MAX} as defined in the ISO 17987-3 [7] (i.e. The nominal frame length plus 40 %).
Relevant frame	From a LIN slave node point of view, a frame that is transmitted or received by the slave. Opposite of `Irrelevant frame`.
Schedule entry (or entry)	Corresponding to the term “Frame Slot” defined in the ISO 17987-3 [7].
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	From a LIN master node’s point of view, 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 ISO 17987 specifications, but mentioned in Figure 14 in ISO 17987-3: Three `Unconditional frame` transfers.
Slot Delay	The time between start of frames in a schedule table. The unit is in number of time-bases for the specific cluster.
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.

Table 2.2: Terms used in the scope of this Document

3 Related documentation

3.1 Input documents & related standards and norms

[1] ISO 17987:2016 (all parts), Road vehicles – Local Interconnect Network (LIN)
https://www.iso.org

[2] LIN Specification Package, Revision 2.2A
https://lin-cia.org/

[3] LIN Specification Package, Revision 2.1
https://lin-cia.org/

[4] Layered Software Architecture
AUTOSAR_CP_EXP_LayeredSoftwareArchitecture

[5] Glossary
AUTOSAR_FO_TR_Glossary

[6] ISO 17987-2:2016 Road vehicles – Local Interconnect Network (LIN) – Part 2: Transport protocol and network layer services
https://www.iso.org

[7] ISO 17987-3:2016 Road vehicles – Local Interconnect Network (LIN) – Part 3: Protocol specification
https://www.iso.org

[8] General Specification of Basic Software Modules
AUTOSAR_CP_SWS_BSWGeneral

[9] J2602-1:2012 LIN Network for Vehicle Applications

[10] Specification of Default Error Tracer
AUTOSAR_CP_SWS_DefaultErrorTracer

[11] Specification of ECU State Manager
AUTOSAR_CP_SWS_ECUStateManager

[12] Specification of PDU Router
AUTOSAR_CP_SWS_PDURouter

[13] Specification of Linklayer Sdu Routing Module
AUTOSAR_CP_SWS_LSduRouter

[14] Specification of LIN State Manager
AUTOSAR_CP_SWS_LINStateManager

[15] Specification of Basic Software Mode Manager
AUTOSAR_CP_SWS_BSWModeManager

[16] Specification of Communication
AUTOSAR_CP_SWS_COM

[17] Specification of LIN Driver
AUTOSAR_CP_SWS_LINDriver

[18] Specification of LIN Transceiver Driver
AUTOSAR_CP_SWS_LINTransceiverDriver

[19] Specification of Bus Mirroring
AUTOSAR_CP_SWS_BusMirroring

[20] General Requirements on Basic Software Modules
AUTOSAR_CP_RS_BSWGeneral

[21] ISO/TR 17987-5:2016 Road vehicles – Local Interconnect Network (LIN) – Part 5: Application programmers interface (API)
https://www.iso.org

[22] System Template
AUTOSAR_CP_TPS_SystemTemplate

[23] Specification of ECU Configuration
AUTOSAR_CP_TPS_ECUConfiguration

3.2 Related specification

AUTOSAR provides a General Specification on Basic Software modules [8] (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 be used as a LIN master or a LIN slave 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 acts on more than one cluster.

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

The LIN Interface module does not support

- `ConditionalChangeNAD` (SID 0xB3, defined in the LIN 2.1 specification [3]; obsolete in ISO 17987-3 [7])
- `DataDump` (SID 0xB4, optional in ISO 17987-3)

Note: Until LIN 2.2A, `ConditionalChangeNAD` was defined without definition on negative response, and it was set to obsolete in ISO 17987-3 without any definition of response behavior. Therefore, both ISO 17987 and LIN specifications do not define the response behavior when `ConditionalChangeNAD` is not supported. However, it is presumed that the clarification "A negative response shall never be sent by the slave node." for `AssignNAD` in ISO 17987-3 is also applicable to `ConditionalChangeNAD`.

For slave nodes, the LIN Interface module does not support

- `ReadByIdentifier` with identifier unequal to 0 and 2 (SID 0xB2, mandatory in ISO 17987-3)
- the `Serial Number` (defined in the ISO 17987-3, clause 6.2.2). It means that there's no corresponding configuration nor API for accessing Serial Number.
- `AutoAddressingSlave` (SID 0xB8, optional in ISO 17987-3), Slave node position detection (SID 0xB5, optional in LIN 2.x specification)

For master nodes, the LIN Interface module does not support

- `ReadByIdentifier` (SID 0xB2, mandatory in ISO 17987-3)

Note: The `ReadByIdentifier` is not considered as node configuration. It is more considered as an identification service. Therefore, it is senseless to support the `ReadByIdentifier` service as a schedule table command. It is the responsibility of the diagnostic layer to support the functionality of `ReadByIdentifier`.

Note: An ECU with a master node on a channel can be a slave node on another channel, too.

The LIN Interface module does not support transmission of `Reserved frames` (defined in the LIN 2.1 specification [3])

The LIN Interface module supports Post-Build Variant, but not directly in the way defined in ISO 17987-3 clause 6.3 Slave node model.

If LinTpScheduleChangeDiag was set to TRUE, simultaneous Schedule Table Switch requests originated from LinTp and from Non-LinTp (BswM or CDD) must be avoided, to prevent premature termination of diagnostic connections. This issue will be fixed in next release(s).

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 [9] and LIN 2.1 [3] are other standard manifestations of ISO 17987 [1]. These alternate standards are predecessors of ISO 17987 and share the concepts of ISO 17987.

An ISO 17987 node is compatible with a LIN 2.1 node (see ISO 17987-3, annex B.2.3).

AUTOSAR LinIf supports the above standards as far as they are identical to ISO 17987.

For legacy reasons, existing slave nodes based on older LIN standards than ISO 17987 (LIN 1.3, LIN 2.0, LIN 2.1 and LIN2.2) are supported as far as the standard is identical to ISO 17987.

5 Dependencies to other modules

This chapter 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 module to operate, the following modules are interfaced:

- Default Error Tracer – Det [10]
- ECU State Manager – EcuM [11]
- PDU Router – PduR [12]
- L-SDU Router – LSduR [13]
- LIN State Manager – LinSM [14]
- BSW Mode Manager – BswM [15]
- AUTOSAR COM – Com [16]
- LIN Driver – Lin [17]
- LIN Transceiver Driver – LinTrcv [18]

5.1 Upper layers

5.1.1 L-SDU Router and CDD

The LIN Interface (except `LinTp` part) connects to the L-SDU Router [13] 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.

5.1.2 PDU Router

In case of `LinTp` part, the PDU Router is the only module above and handles the TP messages buffers either as complete or fragmented messages.

5.1.3 Bus Mirroring

The LIN Interface also connects to the Bus Mirroring module [19]. The content of all received and transmitted LIN frames will be reported, if mirroring is enabled. TP messages are not reported to the Bus Mirroring module.

5.1.4 LIN State Manager

The LIN Interface connects to the LIN state manager [14] which is responsible for the control flow of the whole LIN stack. Therefore, it has the following purposes regarding the LIN Interface:

- For master nodes, the state manager forwards a schedule table request to the LIN Interface.
- The state manager requests the transmission of the wake-up and, for master nodes, the sleep command.

5.1.5 BSW Mode Manager

`LIN TP` that is a part of LIN Interface connects to BSW Mode Manager [15] for requesting the schedule table change when upper layer requests the `LIN TP` operation.

5.1.6 AUTOSAR COM

The LIN Interface used as LIN slave connects to the COM [16] to update the value of the `response_error` signal.

5.2 Lower layers

5.2.1 LIN Driver

The LIN Interface requires the services of the underlying LIN Driver specified by [17].

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`) for LIN master nodes. 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`) for LIN master nodes.
- 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`).
- Setting a LIN channel to state `LIN_CH_OPERATIONAL` without transmitting a wakeup command (`Lin_WakeupInternal`).
- Query of transmission status and reception of the response part of a frame (`Lin_GetStatus`) for LIN master nodes. 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).

For LIN slave nodes, the LIN Interface assumes the following primitives to be serviced by the LIN Driver in addition:

- Indication of a received header (`LinIf_HeaderIndication`). It is assumed that this primitive also tells the direction of the frame response (transmit, receive or `Slave-to-slave` communication).
- Indication of a received response (`LinIf_RxIndication`).
- Confirmation of a transmitted response (`LinIf_TxConfirmation`).
- Indication of a detected communication error event (`LinIf_LinErrorIndication`). The following cases are distinguished:
  - Error during header reception
  - Framing error in response
  - Checksum error
  - Bit error during response transmission
  - Incomplete response
  - No response

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 [18].

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

[SWS_LinIf_00889] Include the header file of LSduR for LinIf

Status: DRAFT

⌈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 L-SDU Router the file LSduR_LinIf.h (see [CP_SWS_LSduR_00001]).⌋

[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_00669] ⌈The LIN Interface shall include the header file(s) of the CDD(s) for callback declaration. The names of the header files are configurable via configuration parameter `LinIfPublicCddHeaderFile`.⌋

[SWS_LinIf_00872] ⌈The LIN Interface shall include the header file Mirror.h if Bus Mirroring is enabled (configuration parameter `LinIfBusMirroringSupported`).⌋

6 Requirements Tracing

The following tables reference the requirements specified in [20], and links to the fulfillment of these. Requirements that are not fulfilled by this document are linked to [SWS_LinIf_NA_99999].

Requirement	Description	Satisfied by
[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_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_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_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_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]	API handling in uninitialized state	[SWS_LinIf_00376]
[SRS_BSW_00413]	An index-based accessing of the instances of BSW modules shall be done	[SWS_LinIf_00197] [SWS_LinIf_00469]

△

Requirement	Description	Satisfied by
[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_00425]	The BSW module description template shall provide means to model the defined trigger conditions of schedulable objects	[SWS_LinIf_00248]
[SRS_BSW_00452]	Classification of runtime errors	[SWS_LinIf_00729]
[SRS_BSW_00480]	Null pointer errors shall follow a naming rule	[SWS_LinIf_00376]
[SRS_BSW_00481]	Invalid configuration set selection errors shall follow a naming rule	[SWS_LinIf_00376]

Table 6.1: Requirements Tracing

7 Functional specification

It is not required to reinvent the requirements already specified in the ISO 17987 specifications [1]. 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 master and slave in the ISO 17987 specifications. 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]

Upstream requirements: SRS_BSW_00425

⌈The LIN Interface shall support the behavior of the master and slave in the ISO 17987 specifications.⌋

The requirement above basically means that the communication from a ISO 17987 node and the LIN Interface node will be equal.

[SWS_LinIf_00249] ⌈The LIN Interface shall realize the LIN behavior so that existing LIN nodes can be reused.⌋

[SWS_LinIf_00386]

Upstream requirements: SRS_BSW_00328

⌈The LIN Interface shall be able to handle one or more LIN channels.⌋

7.1 Frame Transfer

All the functionality of the Protocol Specification in the ISO 17987 specifications [1] 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 ISO 17987 specifications [1]. The existing frame types are:

- Unconditional frame (Chapter 7.1.1.1)
- Event-triggered frame (Chapter 7.1.1.2)
- `Sporadic frame` (Chapter 7.1.1.3)
- Diagnostic frames `MRF` and `SRF` (Chapter 7.1.1.4)
- Reserved frames (Chapter 7.1.1.5)

The actual transmission/reception of the different frames is detailed in the Chapter 7.1.2 “Frame reception” and Chapter 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.

The requirements differentiates between master and slaves nodes depending on the realized node type.

7.1.1.2.1 Event-triggered frame in master nodes

This section is only applicable to LIN master nodes.

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`).⌋

7.1.1.2.2 Event-triggered frame in slave nodes

This section is only applicable to LIN slave nodes.

Upper layers decide the transmission of the response of an `Event-triggered frame`. Therefore, an API call must be available to set the `Event-triggered frame` response pending for transmission.

[SWS_LinIf_00730] ⌈The LIN Interface shall maintain a flag to keep the transfer state of each Event-triggered frame response (defined in the ISO 17987 specifications [1]).⌋

The first data byte of the Unconditional frame response allocated to an Event-triggered frame is reserved for the PID of the Unconditional frame.

[SWS_LinIf_00731] ⌈If the header of an Event-triggered frame is received and the associated response is pending, the LIN Interface shall transmit the PID of the Unconditional frame in the first byte of the response data. The payload of the Unconditional frame response shall be transmitted in the following bytes.⌋

[SWS_LinIf_00732] ⌈The LIN Interface shall clear the pending flag of an Event-triggered frame response once it has been transmitted successfully. This applies also to the case if the response is successfully transmitted as an Unconditional frame.⌋

7.1.1.3 Sporadic frame (Master only)

This section is only applicable to LIN master nodes. From a LIN slave point of view, a received Sporadic frame does not differ from a received Unconditional frame.

The ISO 17987 specifications [1] define 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 frame that are attached to a Sporadic slot.

[SWS_LinIf_00012] ⌈The master shall be the only allowed transmitter of a Sporadic frame (defined in the ISO 17987 specifications).⌋

[SWS_LinIf_00436] ⌈Only a Sporadic frame shall allocate a Sporadic slot (defined in the ISO 17987 specifications).⌋

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 ISO 17987 specifications) for transfer.⌋

[SWS_LinIf_00471] ⌈The LIN Interface shall transmit the specific Sporadic frame (defined in the ISO 17987 specifications) 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 ISO 17987 specifications) 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 ISO 17987 node configuration services and TP messages.

7.1.1.4.1 Diagnostic Frames MRF and SRF (Master only)

The ISO 17987 specifications [1] are 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 Interface module does not support transmission of Reserved frames.

Note: The ISO 17987 specifications [1] do not allow Reserved frames (not allowed since LIN 2.1 [3]).

7.1.2 Frame reception

7.1.2.1 Frame reception in master nodes

This section is only applicable to LIN master nodes.

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

The requirements in this section 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).

7.1.2.1.1 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.⌋

7.1.2.1.2 Response

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

7.1.2.1.3 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_<LinIfChannel.ShortName> function call after the maximum frame length has passed. In this case, the frame transmission is still going on (busy). Therefore the status determination shall be checked again within the next LinIf_MainFunction_<LinIfChannel.ShortName> function call (if the current LinIf_MainFunction_<LinIfChannel.ShortName> does not start a new frame, of course).

The Figure 7.1 shows an example of how the frame transmission is initiated and confirmed on the bus.

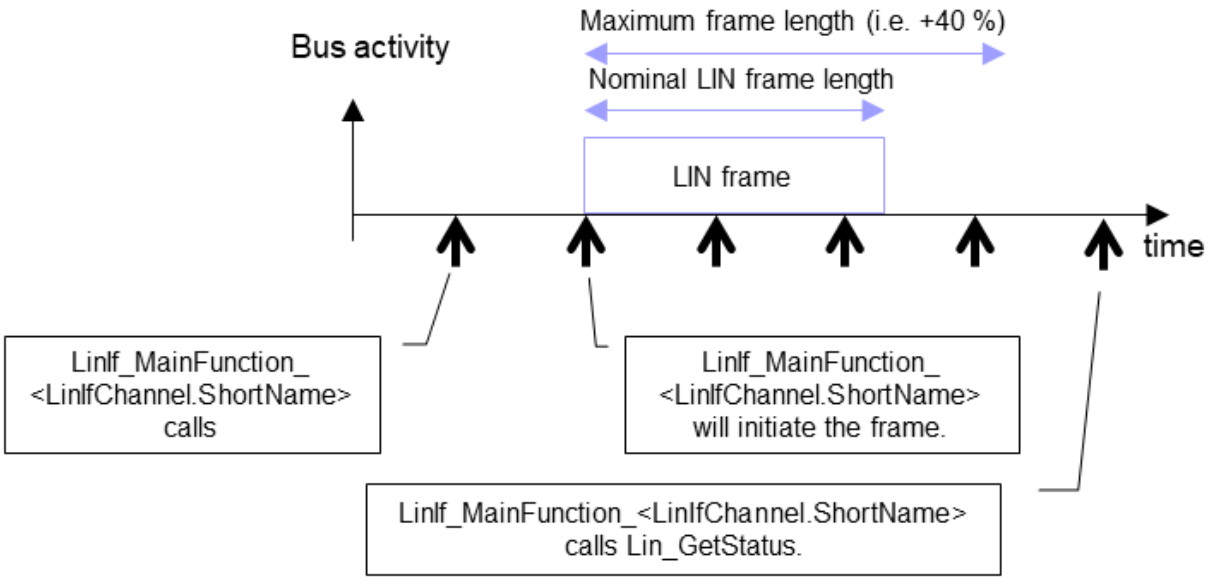


Figure 7.1: `Lin_GetStatus` call example

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_00873] ⌈If Bus Mirroring is enabled globally (configuration parameter `LinIfBusMirroringSupported`) and has been activated with a call to `LinIf_EnableBusMirroring()` for a LIN channel, the LIN Interface shall call `Mirror_ReportLinFrame()` each time after reading the LIN Driver's status for an Rx slot on that channel, providing the received data when the status is `LIN_RX_OK`, and otherwise a NULL pointer.⌋

[SWS_LinIf_00890] Limitation on invocation of `LSduR_LinIfRxIndication` in master nodes

Status: DRAFT

⌈The LIN Interface shall invoke `LSduR_LinIfRxIndication` with the received data only when LIN Interface determines the LIN Driver module's status is `LIN_RX_OK`.⌋

[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 ISO 17987 specifications [1] in this case is not reporting the error to upper layers. The same behavior applies also for transmitted and Slave-to-slave frames.

7.1.2.2 Frame reception in slave nodes

This section is only applicable to LIN slave nodes.

The LIN slave has no knowledge about the scheduling of the LIN master, it solely reacts to received LIN headers reported by the LIN Driver with the header indication callback function LinIf_HeaderIndication.

The requirements in this section are applicable to all frame types that are received by the slave. An exception is the MRF for which only the header handling in Chapter 7.1.2.2.1 applies, but the response handling is described in Chapter 7.6.2 and Chapter 7.6.3.

7.1.2.2.1 Header

[SWS_LinIf_00733] ⌈If the PID of a received header is evaluated and belongs to a configured receive frame, before returning from the callback LinIf_HeaderIndication the LIN Interface shall set the PduPtr->Cs and PduPtr->Dl to the configured values and shall set the PduPtr->Drc to LIN_FRAMERESPONSE_RX.⌋

7.1.2.2.2 Response

The completion of each response reception is notified to the LIN Interface. The LIN Driver indicates a successfully received response to the LIN Interface with the response indication callback function LinIf_RxIndication and an unsuccessful response with the error indication callback function LinIf_LinErrorIndication.

[SWS_LinIf_00891] Invocation of `LSduR_LinIfRxIndication` for Response in slave nodes

Status: DRAFT

⌈If the function `LinIf_RxIndication` is called, the LIN Interface shall invoke `LSduR_LinIfRxIndication` with the received data and payload length.⌋

[SWS_LinIf_00838] ⌈If Bus Mirroring is enabled globally (configuration parameter `LinIfBusMirroringSupported`) and has been activated with a call to `LinIf_EnableBusMirroring()` for a LIN channel, the LIN Interface shall call `Mirror_ReportLinFrame()` each time `LinIf_RxIndication` is called on that channel, with status code `LIN_RX_OK` and a pointer to the received data.⌋

[SWS_LinIf_00735] ⌈If the function `LinIf_LinErrorIndication` is called, the LIN Interface shall consider the response as lost. Therefore, the LIN Interface shall report the runtime error code `LINIF_E_RESPONSE` to the Default Error Tracer unless the error code of `LinIf_LinErrorIndication` is `LIN_ERR_HEADER`.⌋

[SWS_LinIf_00736] ⌈If the reported error is of type `LIN_ERR_RESP_STOPBIT`, `LIN_ERR_RESP_CHKSUM`, `LIN_ERR_RESP_DATABIT` or `LIN_ERR_INC_RESP`, the LIN Interface shall set the `response_error` signal (see [SWS_LinIf_00764]).⌋

[SWS_LinIf_00846] ⌈If `LinIf_HeaderIndication` is called while the indication of a response reception is expected, 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. Afterwards, the received LIN Header shall be processed.⌋

[SWS_LinIf_00869] ⌈If Bus Mirroring is enabled globally (configuration parameter `LinIfBusMirroringSupported`) and has been activated with a call to `LinIf_EnableBusMirroring()` for a LIN channel, the LIN Interface shall call `Mirror_ReportLinFrame()` each time `LinIf_LinErrorIndication` is called on that channel with any error code of `LinIf_LinErrorIndication` other than `LIN_ERR_HEADER`, providing the error status code and a NULL pointer for the frame content.⌋

[SWS_LinIf_00870] ⌈The LIN Interface shall translate the error code reported by `LinIf_LinErrorIndication` to an error code of `Lin_StatusType` before calling `Mirror_ReportLinFrame()`. The error code `LIN_ERR_RESP_STOPBIT` shall be mapped `LIN_TX_ERROR` or `LIN_RX_ERROR`, depending on the direction of the current frame. The error codes `LIN_ERR_RESP_CHKSUM` and `LIN_ERR_INC_RESP` shall be mapped to `LIN_RX_ERROR`. The error code `LIN_ERR_NO_RESP` shall be mapped to `LIN_RX_NO_RESPONSE`. The error code `LIN_ERR_RESP_DATABIT` shall be mapped to `LIN_TX_ERROR`.⌋

Rationale: `Mirror_ReportLinFrame()` expects a `Lin_StatusType` parameter.

7.1.3 Frame transmission

7.1.3.1 Frame transmission in master nodes

This section is only applicable to LIN master nodes.

A LIN frame is transmitted in the LinIf_MainFunction_<LinIfChannel.ShortName> when a new schedule entry is due.

The requirements in this section 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).

7.1.3.1.1 Header and response

[SWS_LinIf_00892] Invocation of LSduR_LinIfTriggerTransmit in master nodes

Status: DRAFT

⌈The LIN Interface shall call the function LSduR_LinIfTriggerTransmit 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_00893] Invocation of Lin_SendFrame after successful invocation of LSduR_LinIfTriggerTransmit in master nodes

Status: DRAFT

⌈After getting the data part of the frame (when the function LSduR_LinIfTriggerTransmit 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_00894] Behavior after failed invocation of LSduR_LinIfTriggerTransmit in master nodes

Status: DRAFT

⌈When the function LSduR_LinIfTriggerTransmit returns E_NOT_OK, the LIN Interface shall not transmit the Sporadic frame or Unconditional frame for which the data was requested.⌋

7.1.3.1.2 Status check

[SWS_LinIf_00874] ⌈If Bus Mirroring is enabled globally (configuration parameter LinIfBusMirroringSupported) and has been activated with a call to LinIf_EnableBusMirroring() for a LIN channel, the LIN Interface shall call Mirror_

ReportLinFrame() each time after reading the LIN Driver's status for a Tx slot on that channel, providing the transmitted data when the status is LIN_TX_OK, and otherwise a NULL pointer.⌋

[SWS_LinIf_00895] Limitation on invocation of LSduR_LinIfTxConfirmation in master nodes

Status: DRAFT

⌈If the return code of the function Lin_GetStatus is LIN_TX_OK, the LIN Interface shall issue a LSduR_LinIfTxConfirmation callback with result E_OK.⌋

[SWS_LinIf_00896] Invocation of LSduR_LinIfTxConfirmation at error or busy conditions in master nodes

Status: DRAFT

⌈If the return code of the function Lin_GetStatus is LIN_TX_ERROR or LIN_TX_BUSY, the LIN Interface shall issue a LSduR_LinIfTxConfirmation 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 (PduRQueueDepth > 1).

7.1.3.2 Frame transmission in slave nodes

This section is only applicable to LIN slave nodes.

The LIN slave has no knowledge about the scheduling, it solely reacts to received LIN headers reported by the LIN Driver with the header indication callback function LinIf_HeaderIndication.

The requirements in this section are applicable to all frame types that are transmitted by the slave. An exception is the SRF for which only the requirements [SWS_LinIf_00898] and [SWS_LinIf_00743] of this section apply, but the remaining handling is described

in Chapter 7.6.2 and Chapter 7.6.3. Event-triggered frames have also a special handling as described in Chapter 7.1.1.2.2.

7.1.3.2.1 Header

[SWS_LinIf_00897] Invocation of LSduR_LinIfTriggerTransmit after Header indication in slave nodes

Status: DRAFT

⌈If LinIf_HeaderIndication is called and the PID is evaluated and determined as a transmit frame, the LIN Interface shall call the function LSduR_LinIfTriggerTransmit with the PduInfoPtr->SduDataPtr set to the buffer provided as PduPtr->SduPtr and PduInfoPtr->SduLength set to the configured length to get the data part of the frame (data in the LIN frame response).⌋

If the frame type is an Event-triggered frame, see also [SWS_LinIf_00731].

[SWS_LinIf_00898] Handling of Cs, Dl and Drc in slave nodes

Status: DRAFT

⌈After getting the data part of the frame (when the function LSduR_LinIfTriggerTransmit returns E_OK or the SRF data is provided by node configuration handler or transport protocol), before returning from the callback LinIf_HeaderIndication, the LIN Interface shall set the PduPtr->Cs and PduPtr->Dl to the configured values and shall set the PduPtr->Drc to LIN_FRAMERESPONSE_TX.⌋

[SWS_LinIf_00899] Handling of Drc for failed invocation of LSduR_LinIfTriggerTransmit in slave nodes

Status: DRAFT

⌈When the function LSduR_LinIfTriggerTransmit returns E_NOT_OK, the LIN Interface shall set the PduPtr->Drc to LIN_FRAMERESPONSE_IGNORE before returning from the callback LinIf_HeaderIndication.⌋

Rationale: Avoid transmission of invalid data on the bus.

7.1.3.2.2 Response

The completion of each response transmission is notified to the LIN Interface. The LIN Driver confirms a successfully transmitted response to the LIN Interface with the response confirmation callback function LinIf_TxConfirmation and an unsuccessful response with the error indication callback function LinIf_LinErrorIndication.

[SWS_LinIf_00900] Invocation of `LSduR_LinIfTxConfirmation` after `LinIf_TxConfirmation` in slave nodes

Status: DRAFT

⌈If the function `LinIf_TxConfirmation` is called, the LIN Interface shall issue a `LSduR_LinIfTxConfirmation` callback with result `E_OK`.⌋

[SWS_LinIf_00747] ⌈If the function `LinIf_TxConfirmation` is called and the transmitted frame contains the `response_error` signal, the LIN Interface shall clear the `response_error` signal.⌋

If the frame type is an `Event-triggered frame` or `Unconditional frame`, consider also [SWS_LinIf_00732].

[SWS_LinIf_00847] ⌈If `LinIf_HeaderIndication` is called while the confirmation of a response transmission is expected, the LIN Interface shall consider the transmitted frame as lost. Therefore, the LIN Interface shall report the runtime error code `LINIF_E_RESPONSE` to the Default Error Tracer. Afterwards, the received LIN Header shall be processed.⌋

[SWS_LinIf_00839] ⌈If Bus Mirroring is enabled globally (configuration parameter `LinIfBusMirroringSupported`) and has been activated with a call to `LinIf_EnableBusMirroring()` for a LIN channel, the LIN Interface shall call `Mirror_ReportLinFrame()` each time `LinIf_TxConfirmation` is called on that channel, with status code `LIN_TX_OK` and a pointer to the transmitted data.⌋

[SWS_LinIf_00901] Invocation of `LSduR_LinIfTxConfirmation` after `LinIf_LinErrorIndication` in slave nodes

Status: DRAFT

⌈If the function `LinIf_LinErrorIndication` is called, the LIN Interface shall issue a `LSduR_LinIfTxConfirmation` callback with result `E_NOT_OK`.⌋

[SWS_LinIf_00743] ⌈If the function `LinIf_LinErrorIndication` is called, the LIN Interface shall consider the transmitted frame as lost and report the runtime error code `LINIF_E_RESPONSE` to the Default Error Tracer unless the error code of `LinIf_LinErrorIndication` is `LIN_ERR_HEADER`.⌋

[SWS_LinIf_00744] ⌈If the error reported in `LinIf_LinErrorIndication` is of type `LIN_ERR_RESP_STOPBIT`, `LIN_ERR_RESP_CHKSUM` or `LIN_ERR_RESP_DATABIT`, the LIN Interface shall set the `response_error` signal (see [SWS_LinIf_00764]).⌋

See [SWS_LinIf_00869] and [SWS_LinIf_00870] for the reporting to the Bus Mirroring module when `LinIf_LinErrorIndication` is called.

7.1.4 Slave-to-slave communication (Master only)

This section is only applicable to LIN master nodes.

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.⌋

7.1.5 Irrelevant communication (Slave only)

This section is only applicable to LIN slave nodes.

The third direction of a frame response is the response of an Irrelevant frame.

[SWS_LinIf_00748] ⌈If LinIf_HeaderIndication is called and the PID is evaluated and determined as a frame that is not relevant for the slave, before returning from the callback LinIf_HeaderIndication, the LIN Interface shall set the PduPtr->Drc to LIN_FRAMERESPONSE_IGNORE.⌋

The LIN Driver will not call LinIf_RxIndication or LinIf_TxConfirmation for Irrelevant frames.

7.2 Schedules (Master only)

This section is only applicable to LIN master nodes.

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 Schedule table manager

The schedule table manager is not defined in the ISO 17987 specifications [1].

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 an ISO 17987 node configuration
- poll Event-triggered frames 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.⌋

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].⌋

[SWS_LinIf_00495] ⌈If the switch to a requested schedule table has been performed, the schedule table manager shall call the function <User>_ScheduleRequest-Confirmation.⌋

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 LinIfResume-Position.⌋

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

7.3 Main function

The LinIf_MainFunction_<LinIfChannel.ShortName> is the central processing function in the LIN Interface. It has to be called periodically.

For LIN master nodes, the task of the function LinIf_MainFunction_<LinIfChannel.ShortName> is to poll the Schedule Table Manager, initiate frame transmission and receptions and interact with upper and lower layers.

For LIN slave nodes, the task of the function LinIf_MainFunction_<LinIfChannel.ShortName> is to supervise different timings. It is up to the implementer to decide if the frame handling and interaction with upper and lower layers is handled on task level or inside the LIN interface callback functions.

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

7.4 Network management

The network management described in this section is based on the ISO 17987 network management [6] 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 ISO 17987 specifications how a node operates.

7.4.1 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.

7.4.1.1 LIN Interface state-machine

[SWS_LinIf_00039] ⌈The LIN Interface shall have one state-machine. The state-machine is depicted in [SWS_LinIf_00887] (for master nodes) and [SWS_LinIf_00888] (for slave nodes).⌋

[SWS_LinIf_00438]

Upstream requirements: SRS_BSW_00335

⌈The LIN Interface state-machine shall have the state LINIF_UNINIT.⌋

[SWS_LinIf_00439]

Upstream requirements: SRS_BSW_00335

⌈The LIN Interface state-machine shall have the state LINIF_INIT.⌋

[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.⌋

7.4.1.2 LIN channel sub-state-machine

The sub-state-machine of the state LINIF_INIT is depicted in [SWS_LinIf_00887] (for master nodes) and [SWS_LinIf_00888] (for slave nodes).

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

[SWS_LinIf_00441]

Upstream requirements: SRS_BSW_00335

⌈The LIN channel sub-state-machine shall have the state LINIF_CHANNEL_OPERATIONAL.⌋

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 receive/transmit LIN frame headers and 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_<LinIfChannel.ShortName>. This requirement is only applicable to LIN master nodes.⌋

[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 external slave nodes will enter bus sleep mode earliest after 4s and latest 10s of bus inactivity (as specified in the ISO 17987 specifications, see ISO 17987-2 [6] clause 5.4). AUTOSAR slave nodes are initialized in sleep mode.

[SWS_LinIf_00442]

Upstream requirements: SRS_BSW_00335

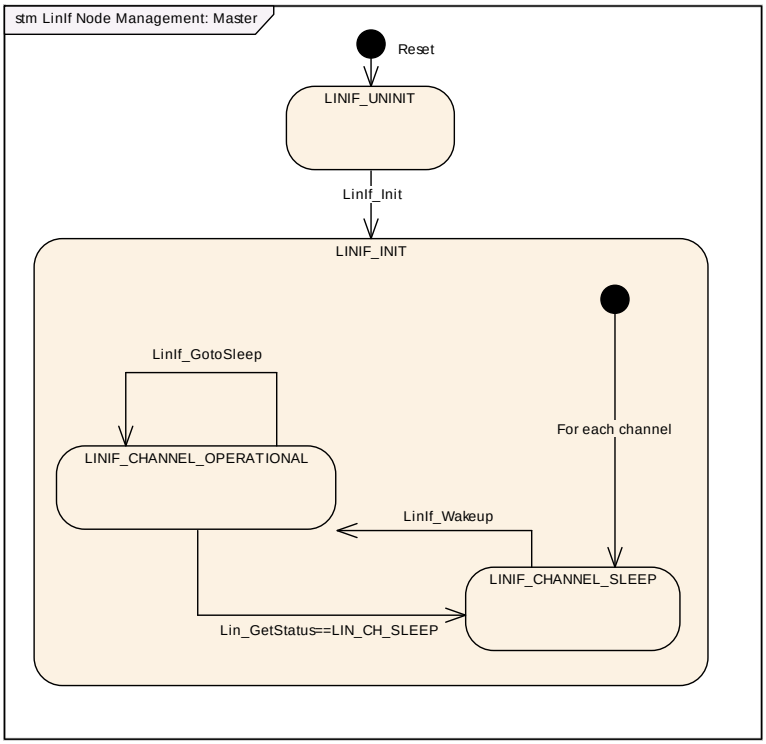
⌈The LIN channel sub-state-machine shall have the state `LINIF_CHANNEL_SLEEP`.⌋

[SWS_LinIf_00478] ⌈The LIN Interface shall transit from the channel state `LINIF_CHANNEL_SLEEP` to `LINIF_CHANNEL_OPERATIONAL` when wake up process was initiated by valid call of `LinIf_Wakeup` 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.

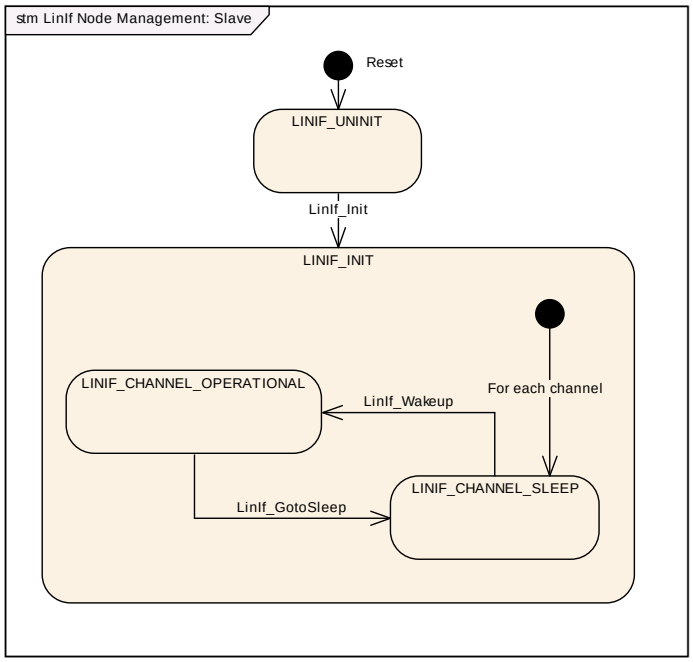
[SWS_LinIf_00043] ⌈When a channel is in the LIN channel state `LINIF_CHANNEL_SLEEP`, the function `LinIf_MainFunction_<LinIfChannel.ShortName>` shall not initiate any traffic on the bus for the corresponding LIN channel.⌋

[SWS_LinIf_00887] LIN Interface state-machine and channel sub-state-machine for LIN Master Nodes ⌈



⌋

[SWS_LinIf_00888] LIN Interface state-machine and channel sub-state-machine for LIN Slave Nodes ⌈



⌋

7.4.2 Go to sleep process

The transition into sleep mode significantly differs between master and slave nodes.

The LIN master node sends a go-to-sleep command when requested by upper layer to set all slave nodes on the bus to sleep mode.

The LIN slave node enters sleep mode either by reception of a go-to-sleep command or by detection of bus inactivity.

7.4.2.1 Go to sleep process in master nodes

This section is only applicable to LIN master nodes.

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 ISO 17987 specifications (see ISO 17987-2 [6] clause 5.4).

[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_<LinIfChannel.ShortName>` 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_<LinIfChannel.ShortName> 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_<LinIfChannel.ShortName> 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_<LinIfChannel.ShortName> 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_<LinIfChannel.ShortName> 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_<LinIfChannel.ShortName> 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_<LinIfChannel.ShortName> 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_<LinIfChannel.ShortName> shall switch the current used schedule table to the NULL_SCHEDULE.⌋

7.4.2.2 Go to sleep process in slave nodes

This section is only applicable to LIN slave nodes.

There are two distinct events in a slave that initiate the transition to sleep mode, the reception of a go-to-sleep command and the occurrence of a bus idle timeout.

7.4.2.2.1 Reception of go-to-sleep command

[SWS_LinIf_00750] ⌈If the function `LinIf_RxIndication` is called and the received frame is a `MRF` with the first data byte (NAD) equal to 0, a go-to-sleep command has been received and the transition to sleep mode shall be executed (see [SWS_LinIf_00756]).⌋

7.4.2.2.2 Bus idle

[SWS_LinIf_00751] ⌈The LIN Interface shall provide bus idle timeout observation (configuration parameter `LinIfBusIdleTimeoutPeriod`) for each channel in order to detect a sleep mode transition event caused by bus inactivity.⌋

[SWS_LinIf_00752] ⌈The LIN Interface shall start the bus idle timeout observation when the state `LINIF_CHANNEL_OPERATIONAL` is entered.⌋

[SWS_LinIf_00753] ⌈The LIN Interface shall stop the bus idle timeout observation when the state `LINIF_CHANNEL_SLEEP` is entered.⌋

[SWS_LinIf_00754] ⌈The LIN Interface shall reload the running bus idle timer each time when `LinIf_HeaderIndication`, `LinIf_RxIndication`, `LinIf_TxConfirmation` or `LinIf_LinErrorIndication` with any error code is called.⌋

[SWS_LinIf_00755] ⌈In case a bus idle timeout occurs, the sleep mode transition shall be executed (see [SWS_LinIf_00756]).⌋

7.4.2.2.3 Sleep mode transition

[SWS_LinIf_00756] ⌈In case of [SWS_LinIf_00750] or [SWS_LinIf_00755], the LIN Interface shall invoke the function `<User>_GotoSleepIndication`.⌋

[SWS_LinIf_00902] LinIf State Transition by Go-to-sleep ⌈The LIN Interface shall transit to the channel state `LINIF_CHANNEL_SLEEP` from `LINIF_CHANNEL_OPERATIONAL` when go-to-sleep process was initiated by valid call of `LinIf_GotoSleep` for the corresponding channel.⌋

[SWS_LinIf_00757] ⌈When the function `LinIf_GotoSleep` is called, the LIN Interface shall call the function `Lin_GoToSleepInternal` directly (and not wait for next main function call).⌋

Rationale: The LIN driver must be in LIN_CH_SLEEP state to be able to receive a wakeup frame on bus.

Note: LinIf_GotoSleep may be called in the context of <User>_GotoSleep-Indication.

[SWS_LinIf_00758] ⌈After calling the function Lin_GoToSleepInternal, the LIN Interface shall clear the wakeup flag of selected channel. (see [SWS_LinIf_00716]).⌋

[SWS_LinIf_00759] ⌈After calling the function Lin_GoToSleepInternal, the LIN Interface shall invoke the function <User>_GotoSleepConfirmation with the parameter TRUE.⌋

7.4.3 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_Check-Wakeup.

[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.⌋

7.4.3.1 Wake up process in master nodes

This section is only applicable to LIN master nodes.

[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.⌋

7.4.3.1.1 Wakeup during sleep transition in master nodes

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 (Figure 7.2):

[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`.⌋

[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`.⌋

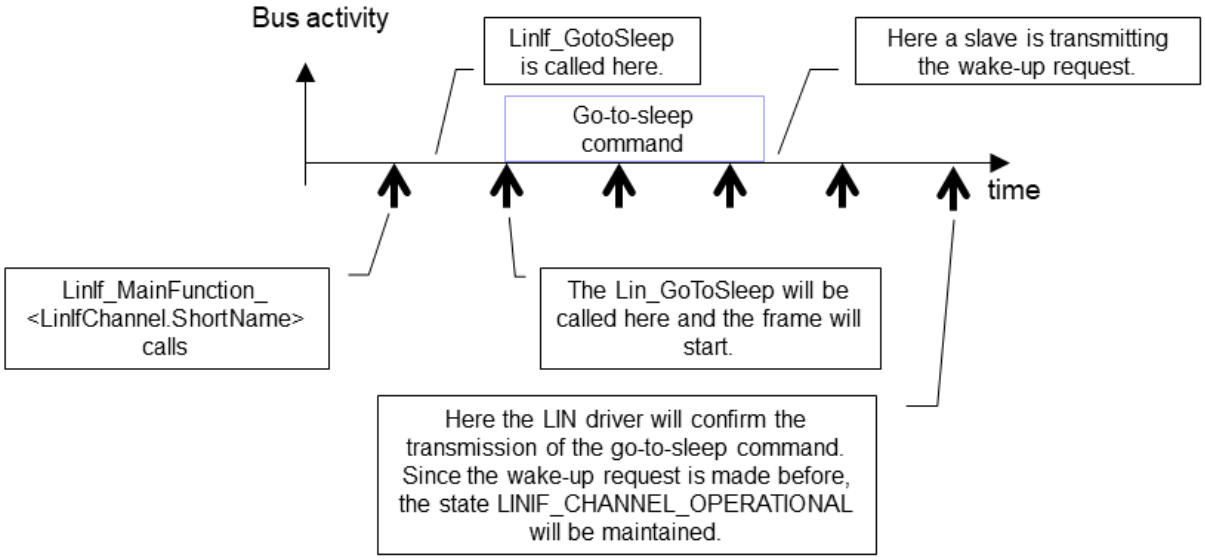


Figure 7.2: Wake up requested before confirmation of go-to-sleep command

7.4.3.2 Wake up process in slave nodes

This section is only applicable to LIN slave nodes.

If the wakeup is requested by upper layer without previous bus wake-up, the wakeup process is started by transmitting the wakeup frame and is completed when the master node starts scheduling (i.e. the first LIN header is received).

[SWS_LinIf_00761] ⌈When the function LinIf_HeaderIndication is called the first time after LinIf_Wakeup was called with return code E_OK, the LIN Interface shall issue the function <User>_WakeupConfirmation with the parameter TRUE.⌋

[SWS_LinIf_00762] ⌈Before returning code E_NOT_OK from the function LinIf_Wakeup, the LIN Interface shall call the function <User>_WakeupConfirmation with the parameter FALSE.⌋

Note: When LinIf_Wakeup returns E_OK but the LIN master node does not start scheduling LIN headers afterwards, the bus was not woken up successfully. In this case, <User>_WakeupConfirmation is not called causing a timeout in the LIN State Manager.

7.4.3.2.1 Wakeup during sleep transition in slave nodes

This section is only applicable to LIN slave nodes.

It may happen that the upper layer requests a wake-up during sleep mode transition, after Lin_GoToSleepInternal has been called and before the sleep mode is entered and the function <User>_GotoSleepConfirmation is called. In this case, the following shall apply:

[SWS_LinIf_00760] ⌈When the LIN Interface has started the sleep mode transition and the upper layer requests a wake-up before the sleep mode transition is completed, the LIN Interface shall not invoke the function <User>_GotoSleepConfirmation and restart the wakeup process by calling the function Lin_Wakeup to wake-up the channel again.⌋

7.5 Status Management

The LIN Interface has to be able to report communication errors on the bus in the same manner as the ISO 17987 specifications describe. 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 ISO 17987-5 specification [21]) that sets the Error_in_response (not to be confused with the slave signal response_error) and the Successful_transfer bits. The strategy here is only to report errors and not to monitor successful transfers.

The conditions for the Error_in_response will be set in the LIN Interface in the same way as described in the ISO 17987 specifications but not reported in the same way. How the Error_in_response is handled is described in chapters Chapter 7.1.2.1.3 and Chapter 7.1.3.1.2 for master nodes respectively Chapter 7.1.2.2.2 and Chapter 7.1.3.2.2 for slave nodes.

7.5.1 response_error signal (Slave only)

This section is only applicable to LIN slave nodes.

The response_error signal is a one bit scalar signal that is published by each slave to the master node in one of its transmitted Unconditional frames. It is used to report the communication status to the LIN cluster.

[SWS_LinIf_00763] ⌈The LIN Interface shall provide the autonomous handling of the response_error signal on each slave channel.⌋

Note: The configuration needs to ensure that LinIf is the only user that has write-access to the response_error signal.

[SWS_LinIf_00764] ⌈The LIN Interface shall call the function Com_SendSignal to update the value of the response_error signal.⌋

The conditions under which to set the response_error signal are defined in [SWS_LinIf_00736] and [SWS_LinIf_00744].

The condition under which to clear the response_error signal is defined in [SWS_LinIf_00747].

[SWS_LinIf_00765] ⌈Each time the response_error signal value has changed, the LIN Interface shall call the function <User_ResponseErrorSignalChanged> with the current value of the response_error signal.⌋

[SWS_LinIf_00766] ⌈The support of function <User_ResponseErrorSignalChanged> is optional and enabled at pre-compile time by the configuration parameter LinIfResponseErrorSignalChangedCallout.⌋

7.6 Diagnostics and Node configuration

Note that node configuration here means the configuration described in the ISO 17987 specifications [7] and has nothing to do with the AUTOSAR configuration.

The Node Configuration in the ISO 17987-3 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 configured NAD and frame ID's).

The Diagnostic Transport Layer and the Node Configuration in ISO 17987 specifications share the MRF and SRF. This will not be a conflict in master nodes since the Node Configuration is using the fixed frame types. For slave nodes, the received MRF and SRF must be evaluated and dispatched to the responsible user, either Transport Layer or Node Configuration handler.

7.6.1 Node configuration in master nodes

This section is only applicable to LIN master nodes.

The ISO 17987 specifications specify two ways for the LIN master to configure slaves:

- By using the ISO 17987 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 ISO 17987 specifications) by using services directly in the Schedule Table.⌋

7.6.1.1 Node Configuration services

The LIN Interface provides node configuration services as specified in the ISO 17987 specifications. 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 ISO 17987 specifications), "Unassign Frame ID" (defined in the LIN 2.0 specification) and "Save Configuration" (defined in the ISO 17987 specifications).⌋

[SWS_LinIf_00409] ⌈The LIN Interface shall support the `FreeFormat` (defined in the ISO 17987 specifications).⌋

The response of the `FreeFormat` is not defined within the ISO 17987 specifications. Therefore, a response from a slave cannot be processed.

[SWS_LinIf_00310]

Upstream requirements: SRS_BSW_00171

⌈The support for the Node Configuration request "Assign NAD" (defined in the ISO 17987-3 specifications) shall be pre-compile time configurable On/Off by the configuration parameter `LinIfNcOptionalRequestSupported`.⌋

Note: The LIN Interface does not support the Node Configuration request `DataDump`, as the ISO 17987 specifications state that the `DataDump` request shall not be used in operational clusters.

7.6.1.2 Node Configuration in Schedule Table

The ISO 17987 specifications allow 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 ISO 17987 specifications.

[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 ISO 17987 specifications.

[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 ISO 17987 specifications. 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 ISO 17987 specifications. 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.

7.6.2 Node configuration in slave nodes

This section is only applicable to LIN slave nodes.

7.6.2.1 Node Model

The LIN Interface uses the Node Model defined in the ISO 17987-3 specification that describes where the configuration is stored for slave nodes.

The LIN Interface manages the currently configured NAD and PIDs of the slave node.

The slave node shall have a valid configuration after reset in order to be addressed by node configuration services and to be able to process Relevant frames.

[SWS_LinIf_00767] ⌈The LIN Interface shall initialize the initial NAD and configured NAD of the slave node from the configuration data (configuration parameters LinIfInitialNAD and LinIfConfiguredNAD) in function LinIf_Init.⌋

Note: The initial NAD is statically configured, i.e. does not change during runtime and is used for "Assign NAD" requests. The configured NAD might change after initialization, either by an "Assign NAD" request or by upper layer, and is used for node configuration services (other than "Assign NAD") and transport protocol.

[SWS_LinIf_00768] ⌈The LIN Interface shall initialize the configured PIDs of the slave node from the configuration data (configuration parameters LinIfFrameId) in function LinIf_Init.⌋

Note: The current configuration of the node can be updated by upper layer using LinIf_SetConfiguredNAD and LinIf_SetPIDTable (e.g. with data loaded from non-volatile memory) or by the LIN master node using node configuration commands.

[SWS_LinIf_00769] ⌈The LIN Interface shall provide the LIN product identification (as described in the ISO 17987-3 specification) consisting of supplier ID, function ID and variant ID (configuration parameters LinIfSupplierId, LinIfFunctionId and LinIfVariantId).⌋

7.6.2.2 Node Configuration services

The LIN Interface provides node configuration services as specified in the ISO 17987-3 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_00810] ⌈The LIN Interface shall support the "Assign NAD" (SID 0xB0, defined in the ISO 17987-3 specification). The support is optional and pre-compile time configurable On/Off by the configuration parameter LinIfNcOptionalRequestSupported.⌋

[SWS_LinIf_00811] ⌈The LIN Interface shall support the "Assign Frame ID range" (SID 0xB7, defined in the ISO 17987-3 specification).⌋

[SWS_LinIf_00812] ⌈The LIN Interface shall support the “Save Configuration” (SID 0xB6, defined in the ISO 17987-3 specification). The support is optional and enabled at pre-compile time by the configuration parameter LinIfSaveConfiguration-Callout.⌋

[SWS_LinIf_00813] ⌈The LIN Interface shall support the “Read by Identifier” with identifier 0 (LIN Product Identification) (SID 0xB2, defined in the ISO 17987-3 specification).⌋

[SWS_LinIf_00840] ⌈The LIN Interface shall support the “Read by Identifier” with identifier 2 (Bit timing test) (SID 0xB2, defined in the ISO 17987-3 specification).⌋

Note: Node configuration services that are not directly supported by LIN Interface are forwarded over Transport Layer to upper layer and can be implemented by integration code.

7.6.2.3 Diagnostic Frame Dispatcher

The diagnostic communication frames (MRF and SRF) are shared by the two diagnostic users in the LIN Interface:

- Node Configuration Handler
- Transport Layer

A priority mechanism is used to dispatch the received diagnostic communication frames to the correct diagnostic user, in which the Node configuration is treated with higher priority than the Transport Protocol.

The LIN Interface dispatches each MRF at first to the Node configuration handler, afterwards to the Transport Protocol. Note that ISO 17987-2, clause 7.6.4 (Unexpected arrival of N_PDU) applies for both diagnostic users.

Similar, each received SRF header is at first passed to the Node configuration handler to transmit a pending response. If no node configuration response waits to be transmitted, the SRF is forwarded to Transport Layer.

[SWS_LinIf_00771] ⌈The LIN Interface shall evaluate the NAD, PCI and SID of a received MRF.⌋

[SWS_LinIf_00772] ⌈If a received MRF contains a valid node configuration request addressing the own slave node, the LIN interface shall accept the node configuration request and inform the Transport layer about the request.⌋

[SWS_LinIf_00773] ⌈If a received MRF does not contain a node configuration request but addresses the own slave node, the LIN interface shall forward the MRF to the Transport layer.⌋

[SWS_LinIf_00774] ⌈If the received MRF does not address the own slave node, the LIN interface shall inform the node configuration handler and the Transport Layer.⌋

Rationale: Any pending request must be aborted if a request addressing another slave node is detected. Of course, the request will not be handled by either diagnostic user.

[SWS_LinIf_00775] ⌈If the header of a SRF is received and the response of a node configuration command is pending for transmission, the response of the SRF shall be transmitted by the node configuration handler.⌋

[SWS_LinIf_00776] ⌈If the header of a SRF is received and no node configuration command response is pending for transmission, the SRF header shall be forwarded to the Transport layer.⌋

[SWS_LinIf_00837] ⌈If the function LinIf_LinErrorIndication is called and a MRF or SRF response is expected, the LIN interface shall inform the node configuration handler and the Transport Layer about the detected communication error.⌋

7.6.2.4 Node Configuration Handler

The Node configuration handler implements the functionality to evaluate, process and respond to node configuration requests supported by the LIN Interface.

A “valid node configuration request” is a MRF with NAD addressing the slave node (initial or broadcast/wildcard NAD for “Assign NAD” request, configured or broadcast/ wildcard NAD for the other supported services), PCI as defined in the ISO 17987-3 specification and a SID value of a supported node configuration service.

[SWS_LinIf_00778] ⌈The LIN Interface shall support wildcards for Function Id, Supplier Id and NAD in node configuration requests (as defined in the ISO 17987-3 specification).⌋

[SWS_LinIf_00780] ⌈If a positive response needs to be sent for a node configuration request, the LIN Interface shall transmit this response to the next scheduled SRF header.⌋

[SWS_LinIf_00779] ⌈If a valid “Assign NAD” request is received, the LIN Interface shall update its configured NAD value with the new NAD of the request and shall provide a positive response when a SRF header is transmitted by the master node.⌋

[SWS_LinIf_00781] ⌈If a valid “Assign Frame ID range” request is received, the LIN Interface shall update its PID configuration with the PIDs of the request (as defined in the ISO 17987-3 specification) and provide a positive response when a SRF header is transmitted by the master node.⌋

[SWS_LinIf_00809] ⌈It shall not be possible to change the PIDs of frames with identifier 0x3C and 0x3D (MRF and SRF).⌋

[SWS_LinIf_00782] ⌈If a valid “Save Configuration” request is received, the LIN Interface shall call the function <User_SaveConfigurationRequest>. Depending on the return value of this callout function, either a positive response or no response shall be provided.⌋

[SWS_LinIf_00783] ⌈If a valid “Read by Identifier” request with identifier 0 is received, the LIN interface shall provide a positive response to be transmitted for next received SRF header (as defined in the ISO 17987-3 specification).⌋

[SWS_LinIf_00841] ⌈If a valid “Read by Identifier” request with identifier 2 is received, the LIN interface shall provide a negative response to be transmitted for next received SRF header (as defined in the ISO 17987-3 specification).⌋

7.6.2.4.1 Node Configuration error

[SWS_LinIf_00784] ⌈The LIN Interface shall provide the N_As timeout observation (configuration parameter LinIfNasTimeout) for node configuration in order to abort a pending response if no SRF header is received.⌋

[SWS_LinIf_00785] ⌈The LIN Interface shall start the N_As timer after reception of a valid node configuration request and stop the timer if a pending node configuration response has been transmitted successfully.⌋

[SWS_LinIf_00786] ⌈In case of N_As timeout occurrence the LIN Interface shall abort the pending node configuration response.⌋

[SWS_LinIf_00787] ⌈If a MRF with an unknown NAD is received, the LIN interface shall reject the request and abort a pending node configuration response.⌋

[SWS_LinIf_00788] ⌈If node configuration request is received with the functional NAD (0x7E), the LIN interface shall ignore the request.⌋

[SWS_LinIf_00871] ⌈If a MRF with the functional NAD (0x7E) is received while a node configuration response is pending, the LIN interface shall ignore the request.⌋

[SWS_LinIf_00789] ⌈If a valid node configuration request is received while a node configuration response is pending, the LIN Interface shall abort the node configuration response and accept the new request.⌋

[SWS_LinIf_00790] ⌈If a node configuration request with an invalid or unknown PCI type is received, the LIN Interface shall ignore this LIN frame.⌋

[SWS_LinIf_00791] ⌈If a node configuration response is pending and new MRF is received with an error in the response (indicated by LinIf_LinErrorIndication), the LIN Interface shall keep the pending node configuration response.⌋

7.6.3 Diagnostics – Transport Protocol

In the ISO 17987 specifications, 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 ISO 17987 specifications) without the contained Diagnostic API which represents the LIN TP.⌋

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

[SWS_LinIf_00387]

Upstream requirements: SRS_BSW_00171

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

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

[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.⌋

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.⌋

7.6.3.1 Schedule requests in master nodes

This section is only applicable to LIN master nodes.

[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.⌋

7.6.3.2 State-machine

The following Figure 7.3 shows the state-machine of the LIN TP.

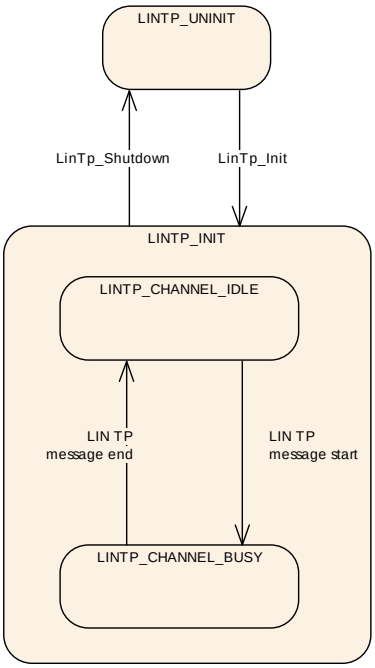


Figure 7.3: LIN Transport Protocol state-machine

[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]

Upstream requirements: SRS_BSW_00335

⌈The `LIN TP` state-machine shall have the state `LINTP_UNINIT`.⌋

[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]

Upstream requirements: SRS_BSW_00335

⌈The `LIN TP` state-machine shall have the state `LINTP_INIT`.⌋

[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 or a configuration request from a LIN master 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.⌋

7.6.3.3 LIN TP transmission

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

The LIN master use the MRF to transmit diagnostic data, while the LIN slave use the LIN response of the SRF to transmit diagnostic data.

[SWS_LinIf_00671] ⌈After a transmission request from the upper layer, the LIN Interface shall call the function PduR_LinTpCopyTxData with the info 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 info.

[SWS_LinIf_00329] ⌈If the function PduR_LinTpCopyTxData returns BUFREQ_E_BUSY, a LIN master node shall not send the next MRF and a LIN slave node shall not send a response to the next SRF header.⌋

[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. For a master node, the LIN Interface shall retry to copy the data during the next processing of the LinIf_MainFunction_<LinIfChannel.ShortName> until the transmit data is provided. For a slave node, the LIN Interface shall retry to copy the data after reception of a SRF header 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, a LIN master node shall resume the transmission of the MRF and a LIN slave node shall resume the response transmission to SRF header.⌋

[SWS_LinIf_00068] ⌈When the LIN Interface has transmitted a SF or the last CF as MRF (LIN master) or SRF response (LIN slave) 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.⌋

7.6.3.4 LIN TP transmission error

7.6.3.4.1 Transmission error handling common for master and slave nodes

[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.⌋

7.6.3.4.2 Transmission error handling for master nodes

This section is only applicable to LIN master nodes.

[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_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 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]). After successful completion or failure (e.g., since some other schedule table change is currently going on) of the

schedule table switch, the LIN Interface shall notify the upper layer by calling the function PduR_LinTpTxConfirmation with the result E_NOT_OK.⌋

[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. (The ISO 17987 specifications define 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 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]). After successful completion or failure (e.g., since some other schedule table change is currently going on) of the schedule table switch, the LIN Interface shall notify the upper layer by calling the function PduR_LinTpTxConfirmation with the result E_NOT_OK.⌋

7.6.3.4.3 Transmission error handling for slave nodes

This section is only applicable to LIN slave nodes.

[SWS_LinIf_00796] ⌈If a LIN error on the SRF response occurs (LinIf_LinErrorIndication is called after reception of a SRF header), 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_00797] ⌈If the start of a new physical request (SF or FF) is received while transmission of a previously triggered physical request is ongoing, the LIN Interface shall abort the ongoing transmission. If the NAD matches the configured NAD of the slave node or the broadcast NAD, the LIN Interface shall accept the new physical request.⌋

[SWS_LinIf_00798] ⌈If a functional request is received while transmission of a previously triggered physical request is ongoing, the LIN Interface shall ignore the functional request.⌋

[SWS_LinIf_00799] ⌈The LIN Interface shall provide the N_As timeout observation (configuration parameter LinTpNas) in order to abort a requested transmission if no SRF header is received.⌋

[SWS_LinIf_00800] ⌈The LIN Interface shall start the N_As timer for a SF or FF after invocation of the function LinTp_Transmit with return value E_OK and for a CF after the LIN driver indicates the reception of a SRF header with invocation of callback function LinIf_HeaderIndication and shall stop the N_As timer after the LIN driver confirms the response transmission for a SRF header with invocation of callback function LinIf_TxConfirmation.⌋

[SWS_LinIf_00801] ⌈In case of N_As timeout occurrence the LIN Interface shall abort the transmission and notify the upper layer by calling the function PduR_LinTpTx-Confirmation with the result E_NOT_OK.⌋

[SWS_LinIf_00802] ⌈The LIN Interface shall provide the N_Cs timeout observation (configuration parameter LinTpNcs) in order to abort an active transmission if no further SRF header is received.⌋

Note: ISO 17987-2 specification [6] defines the following requirement: (N_Cs+N_As) < 0.9*N_Cr timeout

[SWS_LinIf_00803] ⌈The LIN Interface shall start the N_Cs timer after the LIN driver confirms the response transmission for a SRF header with invocation of callback function LinIf_TxConfirmation and stop after the LIN driver indicates the reception of a SRF header with invocation of callback function LinIf_HeaderIndication.⌋

7.6.3.5 LIN TP reception

The LIN Interface shall be prepared to receive TP messages anytime.

The LIN master node receives the TP message from the slaves in one or several SRFs. The first SRF in the TP message will always be a FF or a SF.

The LIN slave node receives the TP message from the master in in one or several MRFs. The first MRF in the TP message will always be a FF or a SF.

Since the LIN Interface does not know when an external node is starting a TP message that the LIN Interface shall receive, it must have the possibility to store part of the TP message.

[SWS_LinIf_00075] ⌈The LIN Interface shall call the function PduR_LinTpStartOf-Reception with an info pointer and TpSduLength when the start of a TP message reception is indicated by the reception of a FF or a SF.

info 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 received NAD from the transmitting LIN node to an N-SDU Id that the upper layer understands.⌋

[SWS_LinIf_00674] ⌈After reception of each frame of a TP message (SF, FF and CF), the LIN Interface shall call the function PduR_LinTpCopyRxData with an info 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 SF or the last CF of a TP message successfully, 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]). After successful completion or failure (e.g., since some other schedule table change is currently going on) of the schedule table switch, the LIN Interface shall notify the upper layer by calling the function PduR_LinTpRxIndication with the result E_OK.⌋

7.6.3.6 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.

7.6.3.6.1 Unavailability of receive buffer common for master and slave nodes

[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 Frame of a TP message (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.⌋

7.6.3.6.2 Unavailability of receive buffer for master nodes

[SWS_LinIf_00792] ⌈If the function PduR_LinTpCopyRxData returns BUFREQ_E_NOT_OK, 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 (see [SWS_LinIf_00646]).⌋

[SWS_LinIf_00879] ⌈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, after successful completion or failure (e.g., since some other schedule table change is currently going on) of required schedule table switch (see [SWS_LinIf_00646] and [SWS_LinIf_00792]).⌋

[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 LinIf_MainFunction_<LinIfChannel.ShortName> 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).⌋

7.6.3.6.3 Unavailability of receive buffer for slave nodes

[SWS_LinIf_00793] ⌈If the function PduR_LinTpCopyRxData returns BUFREQ_OK with a smaller available buffer size than needed for the next CF, the LIN Interface shall call the function PduR_LinTpCopyRxData with a data length (SduLength) 0 (zero) again during the next processing of the LinIf_MainFunction_<LinIfChannel.ShortName> until the available buffer size is big enough or the next CF is received.⌋

[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.⌋

[SWS_LinIf_00794] ⌈In case of [SWS_LinIf_00793], when the buffer of sufficient size is available, the LIN Interface shall copy the received data via the function PduR_LinTpCopyRxData.⌋

[SWS_LinIf_00795] ⌈In case of [SWS_LinIf_00793], when the next CF is received before the data of the current CF could be copied, the LIN Interface shall abort the reception and notify the upper layer by calling the function PduR_LinTpRxIndication with the result E_NOT_OK.⌋

7.6.3.7 LIN TP reception error

7.6.3.7.1 LIN TP reception error common for master and slave nodes

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

[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 which is not the last CF, the LIN Interface shall stop the current LIN TP message reception.⌋

[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.⌋

7.6.3.7.2 LIN TP reception error for master nodes

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 SRFs.

[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 the error that is specified in [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 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]). After successful completion or failure (e.g., since some other schedule table change is currently going on) of the schedule table switch, the LIN Interface shall notify the upper layer by calling the function PduR_LinTpRxIndication with the result E_NOT_OK.⌋

[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 after a reception has been successfully started (i.e., call to PduR_LinTpStartOfReception() has been called and returned BUFREQ_OK), the LIN Interface shall abort the reception 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]). After successful completion or failure (e.g., since some other schedule table change is currently going on) of the schedule table switch, the LIN Interface shall notify the upper layer by calling the function PduR_LinTpRxIndication() with the result E_NOT_OK.⌋

[SWS_LinIf_00877] ⌈In case of P2 timeout occurrence before a reception has been successfully started (i.e., call to PduR_LinTpStartOfReception() did not take place or returned something different from BUFREQ_OK), 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 (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 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]). After successful completion or failure (e.g., since some other schedule table change is currently going on) of the schedule table switch, the LIN Interface shall notify the upper layer by calling the function PduR_LinTpRxIndication with the result E_NOT_OK.⌋

7.6.3.7.3 LIN TP reception error for slave nodes

[SWS_LinIf_00804] ⌈The LIN Interface shall provide the N_Cr timeout observation (configuration parameter LinTpNcr) in order to abort a running reception in case the no further MRF are received.⌋

[SWS_LinIf_00805] ⌈The LIN Interface shall start/restart the N_Cr timer after the LIN driver indicates the reception of a MRF (FF or CF except last CF) with invocation of callback function LinIf_RxIndication and stop after the LIN driver indicates the reception of a MRF (last CF) with invocation of callback function LinIf_RxIndication.⌋

[SWS_LinIf_00806] ⌈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.⌋

[SWS_LinIf_00807] ⌈If a new functional request is received while reception of a physical request is ongoing, the LIN Interface shall ignore the functional request.⌋

[SWS_LinIf_00808] ⌈If the start of new physical request (SF or FF) is received while reception of a physical request is ongoing, the LIN Interface shall stop the current LIN TP message reception (see also [SWS_LinIf_00651]). If the received NAD matches the configured NAD or broadcast NAD, the new request shall be accepted.⌋

7.7 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.

7.7.1 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).⌋

The LIN node type of a LIN channel is determined by the choice container LinIfNodeType.

7.7.2 Multiple LIN drivers

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

[SWS_LinIf_00462] ⌈The allocation of each channel to a LIN Driver shall be precompile time configurable by the configuration parameter LinIfMultipleDriversSupported.⌋

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

7.7.3 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 `vendorId` and a `vendorApiInfix`.

[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.

7.8 Error classification

Chapter [8, General Specification of Basic Software Modules] 7.2 *"Error Handling"* describes the error handling of the Basic Software in detail. Above all, it constitutes a classification scheme consisting of five error types which may occur in BSW modules.

Based on this foundation, the following section specifies particular errors arranged in the respective subsections below.

7.8.1 Development Errors

[SWS_LinIf_00376] Definition of development errors in module LinIf

Upstream requirements: SRS_BSW_00406, SRS_BSW_00337, SRS_BSW_00385, SRS_BSW_00327, SRS_BSW_00480, SRS_BSW_00481

⌈

Type of error	*Related error code*	*Error value*
API called without initialization of LIN Interface	LINIF_E_UNINIT	0x00
Initialization failed, e.g. selected configuration set doesn't exist	LINIF_E_INIT_FAILED	0x10
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

⌋

Note: The table covers the error codes for the LIN Interface and the LIN TP.

Note: The error code `LINIF_E_SCHEDULE_REQUEST_ERROR` is only used by a LIN master node.

7.8.2 Runtime Errors

[SWS_LinIf_00729] Definition of runtime errors in module LinIf

Upstream requirements: SRS_BSW_00452, SRS_BSW_00385, SRS_BSW_00327

⌈

Type of error	*Related error code*	*Error value*
LIN frame error detected	LINIF_E_RESPONSE	0x60
Slave did not answer on a node configuration request	LINIF_E_NC_NO_RESPONSE	0x61
A N_As timeout is detected	LINTP_E_LINTPNAS_TIMEOUT_OCCURRED	0x62
A N_Cs timeout is detected	LINTP_E_LINTPNCS_TIMEOUT_OCCURRED	0x63
A N_Cr timeout is detected	LINTP_E_LINTPNCR_TIMEOUT_OCCURRED	0x64
A swapped Consecutive Frame is received	LINTP_E_SWAPPED_CONSECUTIVE_FRAMES_RECEIVED	0x65
Missing Consecutive Frame is detected	LINTP_E_DROPPED_CONSECUTIVE_FRAMES_DETECTED	0x66
Schedule table switch request not accepted	LINIF_E_SCHEDULE_TABLE_SWITCH_REQUEST_NOT_ACCEPTED	0x67

⌋

Note: The table covers the error codes for the LIN Interface and the LIN TP.

7.8.3 Production Errors

There are no Production Errors.

8 API specification

8.1 Imported types

8.1.1 Standard types

In this section, all types included from the following modules are listed:

[SWS_LinIf_00469] Definition of imported datatypes of module LinIf

Upstream requirements: SRS_BSW_00413

⌈

Module	*Header File*	*Imported Type*
Com	Com.h	Com_SignalIdType
Comtype	ComStack_Types.h	BufReq_ReturnType
	ComStack_Types.h	NetworkHandleType
	ComStack_Types.h	PduIdType
	ComStack_Types.h	PduInfoType
	ComStack_Types.h	PduLengthType
	ComStack_Types.h	RetryInfoType
	ComStack_Types.h	TPParameterType
	ComStack_Types.h	TpDataStateType
EcuM	EcuM.h	EcuM_WakeupSourceType
Lin	Lin_GeneralTypes.h	Lin_FrameCsModelType
	Lin_GeneralTypes.h	Lin_FrameDlType
	Lin_GeneralTypes.h	Lin_FramePidType
	Lin_GeneralTypes.h	Lin_FrameResponseType
	Lin_GeneralTypes.h	Lin_PduType
	Lin_GeneralTypes.h	Lin_SlaveErrorType
	Lin_GeneralTypes.h	Lin_StatusType
LinTrcv	Lin_GeneralTypes.h	LinTrcv_TrcvModeType
	Lin_GeneralTypes.h	LinTrcv_TrcvWakeupModeType
	Lin_GeneralTypes.h	LinTrcv_TrcvWakeupReasonType
Std	Std_Types.h	Std_ReturnType

⌋

8.2 Type definitions

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

8.2.1 LinIf_SchHandleType

This type is only applicable for LIN master nodes.

[SWS_LinIf_00197] Definition of datatype LinIf_SchHandleType

Upstream requirements: SRS_BSW_00413

⌈

Name	LinIf_SchHandleType		
Kind	Type		
Derived from	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		
Available via	LinIf.h		

⌋

8.2.2 LinIf_ConfigType

[SWS_LinIf_00668] Definition of datatype LinIf_ConfigType ⌈

Name	LinIf_ConfigType	
Kind	Structure	
Elements	implementation specific	
	Type	–
	Comment	–
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.	
Available via	LinIf.h	

⌋

8.2.3 LinTp_ConfigType

[SWS_LinIf_00426] Definition of datatype LinTp_ConfigType ⌈

Name	LinTp_ConfigType	
Kind	Structure	
Elements	implementation specific	
	Type	–
	Comment	–
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	
Available via	LinIf.h	

⌋

8.2.4 LinTp_Mode

This type is only applicable for LIN master nodes.

[SWS_LinIf_00629] Definition of datatype LinTp_Mode ⌈

Name	LinTp_Mode		
Kind	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		
Available via	LinIf.h		

⌋

8.3 Function definitions

This chapter contains all APIs provided by the module.

8.3.1 LinIf_Init

[SWS_LinIf_00198] Definition of API function LinIf_Init

Upstream requirements: SRS_BSW_00101, SRS_BSW_00416, SRS_BSW_00358, SRS_BSW_00414

⌈

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.	
Available via	LinIf.h	

⌋

[SWS_LinIf_00373]

Upstream requirements: SRS_BSW_00344, SRS_BSW_00404, SRS_BSW_00405, SRS_BSW_00170

⌈The function LinIf_Init shall accept a parameter that references to a LIN Interface configuration descriptor.⌋

[SWS_LinIf_00233] ⌈The function LinIf_Init shall set the schedule type NULL_SCHEDULE for each configured channel. This requirement is only applicable to LIN master nodes.⌋

8.3.2 LinIf_Transmit

[SWS_LinIf_00201] Definition of API function LinIf_Transmit ⌈

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.	
Available via	LinIf.h	

⌋

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 (LIN master node) or an Event-triggered frame (LIN slave node) as pending for transmission and shall ignore other frame types.⌋

[SWS_LinIf_00700] ⌈The function LinIf_Transmit shall also return E_OK in case the Pdu belongs to a non-Sporadic frame (LIN master node) or non-Event-triggered frame (LIN slave node) and LIN Interface is initialized.⌋

[SWS_LinIf_00106] ⌈The function LinIf_Transmit shall tolerate repeated invocations while the Sporadic frame (LIN master node) or Event-triggered frame (LIN slave node) 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. This requirement is only applicable to LIN master nodes.⌋

Caveats: The LIN Interface has to be initialized with a call of LinIf_Init before this API service may be called, see [SRS_BSW_00487].

8.3.3 LinIf_ScheduleRequest

The service LinIf_ScheduleRequest is only applicable to LIN master node (available only if the ECU has any LIN master channel).

[SWS_LinIf_00202] Definition of API function LinIf_ScheduleRequest ⌈

Service Name	LinIf_ScheduleRequest	
Syntax	`Std_ReturnType LinIf_ScheduleRequest ( NetworkHandleType Channel, LinIf_SchHandleType ScheduleTableIdx )`	
Service ID [hex]	0x05	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Parameters (in)	Channel	Channel index.
	ScheduleTableIdx	Index of the scheduled table
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. Only used for LIN master nodes.	
Available via	LinIf.h	

⌋

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.⌋

[SWS_LinIf_00858] ⌈The function LinIf_ScheduleRequest is only available if the LinIf module is configured as LIN master node on at least one channel. In a pure LIN

slave configuration, this function is not available. This depends on the configuration parameter LinIfNodeType.⌋

Caveats: The LIN Interface has to be initialized with a call of LinIf_Init before this API service may be called, see [SRS_BSW_00487].

8.3.4 LinIf_GotoSleep

[SWS_LinIf_00204] Definition of API function LinIf_GotoSleep ⌈

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.	
Available via	LinIf.h	

⌋

[SWS_LinIf_00488] ⌈The function LinIf_GotoSleep shall initiate a transition into sleep mode on the selected Channel. (see [SWS_LinIf_00453], [SWS_LinIf_00597] and [SWS_LinIf_00757])⌋

[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.⌋

For LIN master nodes, the function LinIf_GotoSleep will start the process of putting the cluster into sleep and not do it immediately.

For LIN slave nodes, the function LinIf_GotoSleep sets the cluster directly into sleep after sleep indication by the master node.

Caveats: The LIN Interface has to be initialized with a call of LinIf_Init before this API service may be called, see [SRS_BSW_00487].

8.3.5 LinIf_Wakeup

[SWS_LinIf_00205] Definition of API function LinIf_Wakeup ⌈

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.	
Available via	LinIf.h	

⌋

[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_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 [SRS_BSW_00487].

8.3.6 LinIf_SetTrcvMode

[SWS_LinIf_00544] Definition of API function LinIf_SetTrcvMode ⌈

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.	
Available via	LinIf.h	

⌋

[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 [SRS_BSW_00487].

8.3.7 LinIf_GetTrcvMode

[SWS_LinIf_00545] Definition of API function LinIf_GetTrcvMode ⌈

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.	
Available via	LinIf.h	

⌋

[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 [SRS_BSW_00487].

8.3.8 LinIf_GetTrcvWakeupReason

[SWS_LinIf_00547] Definition of API function LinIf_GetTrcvWakeupReason ⌈

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.	
Available via	LinIf.h	

⌋

[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 precompile 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 [SRS_BSW_00487].

- 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_WU_POWER_ON, whereas the other may state e.g. LINTRCV_WU_RESET.

It is up to the EcuM to decide how to handle that wake up information.

8.3.9 LinIf_SetTrcvWakeupMode

[SWS_LinIf_00550] Definition of API function LinIf_SetTrcvWakeupMode ⌈

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	
Available via	LinIf.h	

⌋

[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 LinIfTrcvDriver-Supported.⌋

Caveats:

- The LIN Interface has to be initialized with a call of `LinIf_Init` before this API service may be called, see [SRS_BSW_00487].
- 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.

8.3.10 `LinIf_GetPIDTable`

The service `LinIf_GetPIDTable` is only applicable to LIN slave node (available only if the ECU has any LIN slave channel).

[SWS_LinIf_91002] Definition of API function LinIf_GetPIDTable ⌈

Service Name	LinIf_GetPIDTable	
Syntax	`Std_ReturnType LinIf_GetPIDTable ( NetworkHandleType Channel, Lin_FramePidType* PidBuffer, uint8* PidBufferLength )`	
Service ID [hex]	0x72	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Channel	Identification of the LIN channel.
Parameters (inout)	PidBuffer	Pointer to existing buffer to which the current assigned PID values are copied to
	PidBufferLength	Pointer to actual length of provided buffer. After successful return, it contains the number of copied PID values.
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: Request has been accepted. `E_NOT_OK`: Request has not been accepted, development or production error occurred.
Description	Retrieves all assigned PID values. The order is congruent to the LIN frame index. Only applicable for LIN slave nodes.	
Available via	LinIf.h	

⌋

[SWS_LinIf_00816] ⌈This API shall return the configured PIDs of all frames relevant for the slave node. The order shall be ascending corresponding to the configuration parameter `LinIfFrameIndex`. The value of `PidBufferLength` shall be updated with the actual number of configured PIDs.⌋

[SWS_LinIf_00817] ⌈The returned PID list shall not include the PIDs for `MRF` and `SRF`.⌋

Rationale: `MRF` and `SRF` are always implicitly assigned to each slave node and their PIDs shall not be changed (see also [SWS_LinIf_00809]).

[SWS_LinIf_00828] ⌈If the length of the buffer (provided by parameter PidBuffer-Length) is 0, the function shall return the number of configured PIDs of the slave node in parameter PidBufferLength, without updating the PID buffer with PIDs.⌋

[SWS_LinIf_00818] ⌈If development error detection is enabled and an invalid value for Channel is given, the function LinIf_GetPIDTable shall report LINIF_E_NONEXISTENT_CHANNEL to the default error tracer.⌋

[SWS_LinIf_00819] ⌈If development error detection is enabled and an invalid value for PidBuffer is given, the function LinIf_GetPIDTable shall raise the development error code LINIF_E_PARAM_POINTER.⌋

[SWS_LinIf_00820] ⌈If development error detection is enabled and an invalid value for PidBufferLength is given, the function LinIf_GetPIDTable shall raise the development error code LINIF_E_PARAM_POINTER.⌋

[SWS_LinIf_00821] ⌈If development error detection is enabled and length of the buffer (provided by parameter PidBufferLength) is smaller than the number of configured PIDs of the slave node (except 0, see [SWS_LinIf_00828]), the function LinIf_Get-PIDTable shall raise the development error code LINIF_E_PARAMETER.⌋

[SWS_LinIf_00859] ⌈The function LinIf_GetPIDTable is only available if the LinIf module is configured as LIN slave node on at least one channel. In a pure LIN master configuration, this function is not available. This depends on the configuration parameter LinIfNodeType.⌋

8.3.11 LinIf_SetPIDTable

The service LinIf_SetPIDTable is only applicable to LIN slave node (available only if the ECU has any LIN slave channel).

[SWS_LinIf_91003] Definition of API function LinIf_SetPIDTable ⌈

Service Name	LinIf_SetPIDTable	
Syntax	`Std_ReturnType LinIf_SetPIDTable ( NetworkHandleType Channel, Lin_FramePidType* PidBuffer, uint8 PidBufferLength )`	
Service ID [hex]	0x73	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Channel	Identification of the LIN channel.
	PidBuffer	Pointer to buffer which contains the PID values to configure.
	PidBufferLength	Number of PID values in the provided buffer
Parameters (inout)	None	

▽

△

Parameters (out)	None	
Return value	Std_ReturnType	E_OK: Request has been accepted. E_NOT_OK: Request has not been accepted, development or production error occurred.
Description	Sets all assigned PID values. The order is congruent to the LIN frame index. Only applicable for LIN slave nodes.	
Available via	LinIf.h	

⌋

[SWS_LinIf_00823] ⌈This API shall update the internal configured PID list in LIN Interface with the given PID list.⌋

Note: The user is responsible that the order of the PIDs in the provided buffer is ascending corresponding to the configuration parameter LinIfFrameIndex.

[SWS_LinIf_00824] ⌈The provided PID list shall not include the PIDs for MRF and SRF.⌋

Rationale: MRF and SRF are always implicitly assigned to each slave node and their PIDs shall not be changed (see also [SWS_LinIf_00809]).

[SWS_LinIf_00825] ⌈If development error detection is enabled and an invalid value for Channel is given, the function LinIf_SetPIDTable shall report LINIF_E_NONEXISTENT_CHANNEL to the default error tracer.⌋

[SWS_LinIf_00826] ⌈If development error detection is enabled and an invalid value for PidBuffer is given, the function LinIf_SetPIDTable shall raise the development error code LINIF_E_PARAM_POINTER.⌋

[SWS_LinIf_00827] ⌈If development error detection is enabled and the value of PidBufferLength is smaller than the number of the configured PIDs of the slave node, the function LinIf_SetPIDTable shall raise the development error code LINIF_E_PARAMETER.⌋

[SWS_LinIf_00860] ⌈The function LinIf_SetPIDTable is only available if the LinIf module is configured as LIN slave node on at least one channel. In a pure LIN master configuration, this function is not available. This depends on the configuration parameter LinIfNodeType.⌋

8.3.12 LinIf_GetConfiguredNAD

The service LinIf_GetConfiguredNAD is only applicable to LIN slave node (available only if the ECU has any LIN slave channel).

[SWS_LinIf_00829] Definition of API function LinIf_GetConfiguredNAD ⌈

Service Name	LinIf_GetConfiguredNAD	
Syntax	`Std_ReturnType LinIf_GetConfiguredNAD ( NetworkHandleType Channel, uint8* Nad )`	
Service ID [hex]	0x70	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Channel	Identification of the LIN channel.
Parameters (inout)	None	
Parameters (out)	Nad	Configured NAD of slave
Return value	Std_ReturnType	`E_OK`: Request has been accepted. `E_NOT_OK`: Request has not been accepted, development or production error occurred.
Description	Reports the current configured NAD. Only applicable for LIN slave nodes.	
Available via	LinIf.h	

⌋

[SWS_LinIf_00830] ⌈This API shall return the configured NAD of the slave node.⌋

[SWS_LinIf_00831] ⌈If development error detection is enabled and an invalid value for `Channel` is given, the function `LinIf_GetConfiguredNAD` shall report `LINIF_E_NONEXISTENT_CHANNEL` to the default error tracer.⌋

[SWS_LinIf_00832] ⌈If development error detection is enabled and an invalid value for `Nad` is given, the function `LinIf_GetConfiguredNAD` shall raise the development error code `LINIF_E_PARAM_POINTER`.⌋

[SWS_LinIf_00861] ⌈The function `LinIf_GetConfiguredNAD` is only available if the LinIf module is configured as LIN slave node on at least one channel. In a pure LIN master configuration, this function is not available. This depends on the configuration parameter `LinIfNodeType`.⌋

8.3.13 `LinIf_SetConfiguredNAD`

The service `LinIf_SetConfiguredNAD` is only applicable to LIN slave node (available only if the ECU has any LIN slave channel).

[SWS_LinIf_00833] Definition of API function LinIf_SetConfiguredNAD ⌈

Service Name	LinIf_SetConfiguredNAD
Syntax	`Std_ReturnType LinIf_SetConfiguredNAD ( NetworkHandleType Channel, uint8 Nad )`

▽

△

Service ID [hex]	0x71	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Channel	Identification of the LIN channel.
	Nad	Configured NAD to set as new slave NAD
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: Request has been accepted. `E_NOT_OK`: Request has not been accepted, development or production error occurred.
Description	Sets the current configured NAD. Only applicable for LIN slave nodes.	
Available via	LinIf.h	

⌋

[SWS_LinIf_00834] ⌈This API shall update the configured NAD of the slave node.⌋

[SWS_LinIf_00835] ⌈If development error detection is enabled and an invalid value for `Channel` is given, the function `LinIf_SetConfiguredNAD` shall report `LINIF_E_NONEXISTENT_CHANNEL` to the default error tracer.⌋

[SWS_LinIf_00836] ⌈If development error detection is enabled and the value 0 for `Nad` is given, the function `LinIf_SetConfiguredNAD` shall raise the development error code `LINIF_E_PARAMETER`.⌋

[SWS_LinIf_00862] ⌈The function `LinIf_SetConfiguredNAD` is only available if the LinIf module is configured as LIN slave node on at least one channel. In a pure LIN master configuration, this function is not available. This depends on the configuration parameter `LinIfNodeType`.⌋

8.3.14 `LinIf_CheckWakeup`

[SWS_LinIf_00378] Definition of API function LinIf_CheckWakeup

Upstream requirements: SRS_BSW_00375

⌈

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.	
Available via	LinIf.h	

⌋

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`.⌋

Note: It is implementation specific, which controllers and transceivers are queried. LinIf just has to find out the exact LIN hardware device.

[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`.⌋

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

8.3.15 `LinIf_EnableBusMirroring`

[SWS_LinIf_00876] Definition of API function LinIf_EnableBusMirroring ⌈

Service Name	LinIf_EnableBusMirroring	
Syntax	`Std_ReturnType LinIf_EnableBusMirroring ( NetworkHandleType Channel, boolean MirroringActive )`	
Service ID [hex]	0x7f	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Channel	Identification of the LIN channel.
	MirroringActive	TRUE: Mirror_ReportLinFrame will be called for each frame received or transmitted on the given channel. FALSE: Mirror_ReportLinFrame will not be called for the given channel.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: Mirroring mode was changed. E_NOT_OK: Wrong Channel, or mirroring globally disabled (see LinIfBusMirroringSupport).
Description	Enables or disables mirroring for a LIN channel.	

▽

△

Available via	LinIf.h

⌋

[SWS_LinIf_00875] ⌈If Bus Mirroring is not enabled (configuration parameter `LinIfBusMirroringSupported`), the API `LinIf_EnableBusMirroring` can be omitted.⌋

8.3.16 LinIf_WakeupConfirmation

[SWS_LinIf_00715] Definition of callback function LinIf_WakeupConfirmation ⌈

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.	
Available via	LinIf.h	

⌋

[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.⌋

8.3.17 LinIf_HeaderIndication

The callback function LinIf_HeaderIndication is only applicable for LIN slave node.

[SWS_LinIf_91004] Definition of callback function LinIf_HeaderIndication ⌈

Service Name	LinIf_HeaderIndication	
Syntax	`Std_ReturnType LinIf_HeaderIndication ( NetworkHandleType Channel, Lin_PduType* PduPtr )`	
Service ID [hex]	0x78	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different Channels. Non reentrant for the same Channel.	
Parameters (in)	Channel	Identification of the LIN channel

▽

△

Parameters (inout)	PduPtr	Pointer to PDU providing the received PID and pointer to the SDU data buffer as in parameter. Upon return, the length, checksum type and frame response type are received as out parameter. If the frame response type is LIN_FRAMERESPONSE_TX, then the SDU data buffer contains the transmission data.
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: Request has been accepted. `E_NOT_OK`: Request has not been accepted, development or production error occurred.
Description	The LIN Driver will call this function to report a received LIN header. This function is only applicable for LIN slave nodes (available only if the ECU has any LIN slave channel).	
Available via	LinIf.h	

⌋

[SWS_LinIf_00843] ⌈If development error detection is enabled and an invalid value for `Channel` is given, the function `LinIf_HeaderIndication` shall report `LINIF_E_NONEXISTENT_CHANNEL` to the default error tracer.⌋

[SWS_LinIf_00844] ⌈If development error detection is enabled and the parameter `PduPtr` has an invalid value, the function `LinIf_HeaderIndication` shall raise the development error code `LINIF_E_PARAM_POINTER`.⌋

[SWS_LinIf_00845] ⌈If development error detection is enabled, the PID is evaluated and rated to belong to a transmit frame and the parameter `PduPtr->SduPtr` has an invalid value, the function `LinIf_HeaderIndication` shall raise the development error code `LINIF_E_PARAM_POINTER`.⌋

[SWS_LinIf_00863] ⌈The function `LinIf_HeaderIndication` is only available if the LinIf module is configured as LIN slave node on at least one channel. In a pure LIN master configuration, this function is not available. This depends on the configuration parameter `LinIfNodeType`.⌋

8.3.18 `LinIf_RxIndication`

The callback function `LinIf_RxIndication` is only applicable for LIN slave node.

[SWS_LinIf_91005] Definition of callback function LinIf_RxIndication ⌈

Service Name	LinIf_RxIndication	

Syntax	`void LinIf_RxIndication (`
	`  NetworkHandleType Channel,`
	`  uint8* Lin_SduPtr`
	`)`	

Service ID [hex]	0x79	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different Channels. Non reentrant for the same Channel.	
Parameters (in)	Channel	Identification of the LIN channel

▽

△

	Lin_SduPtr	Pointer to pointer to a shadow buffer or memory mapped LIN Hardware receive buffer where the current SDU is stored. This pointer is only valid if the response is received.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	The LIN Driver will call this function to report a successfully received response and provides the reception data to the LIN Interface. This function is only applicable for LIN slave nodes (available only if the ECU has any LIN slave channel).	
Available via	LinIf.h	

⌋

[SWS_LinIf_00848] ⌈If no header of a receive frame has been indicated before (no response reception is expected), the function `LinIf_RxIndication` shall return without further action.⌋

Rationale: Unexpected calls to `LinIf_RxIndication` shall be ignored.

[SWS_LinIf_00849] ⌈If development error detection is enabled and an invalid value for `Channel` is given, the function `LinIf_RxIndication` shall report `LINIF_E_NONEXISTENT_CHANNEL` to the default error tracer.⌋

[SWS_LinIf_00850] ⌈If development error detection is enabled and the parameter `Lin_SduPtr` has an invalid value, the function `LinIf_RxIndication` shall raise the development error code `LINIF_E_PARAM_POINTER`.⌋

[SWS_LinIf_00864] ⌈The function `LinIf_RxIndication` is only available if the LinIf module is configured as LIN slave node on at least one channel. In a pure LIN master configuration, this function is not available. This depends on the configuration parameter `LinIfNodeType`.⌋

8.3.19 `LinIf_TxConfirmation`

The callback function `LinIf_TxConfirmation` is only applicable for LIN slave node.

[SWS_LinIf_91006] Definition of callback function LinIf_TxConfirmation ⌈

Service Name	LinIf_TxConfirmation	
Syntax	`void LinIf_TxConfirmation ( NetworkHandleType Channel )`	
Service ID [hex]	0x7a	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different Channels. Non reentrant for the same Channel.	
Parameters (in)	Channel	Identification of the LIN channel
Parameters (inout)	None	

▽

△

Parameters (out)	None
Return value	None
Description	The LIN Driver will call this function to report a successfully transmitted response. This function is only applicable for LIN slave nodes (available only if the ECU has any LIN slave channel).
Available via	LinIf.h

⌋

[SWS_LinIf_00852] ⌈If no header of a transmit frame has been indicated before (no response transmission is expected), the function `LinIf_TxConfirmation` shall return without further action.⌋

Rationale: Unexpected calls to `LinIf_TxConfirmation` shall be ignored.

[SWS_LinIf_00853] ⌈If development error detection is enabled and an invalid value for `Channel` is given, the function `LinIf_TxConfirmation` shall report `LINIF_E_NONEXISTENT_CHANNEL` to the default error tracer.⌋

[SWS_LinIf_00865] ⌈The function `LinIf_TxConfirmation` is only available if the LinIf module is configured as LIN slave node on at least one channel. In a pure LIN master configuration, this function is not available. This depends on the configuration parameter `LinIfNodeType`.⌋

8.3.20 `LinIf_LinErrorIndication`

The callback function `LinIf_LinErrorIndication` is only applicable for LIN slave node.

[SWS_LinIf_91007] Definition of callback function LinIf_LinErrorIndication ⌈

Service Name	LinIf_LinErrorIndication	
Syntax	`void LinIf_LinErrorIndication ( NetworkHandleType Channel, Lin_SlaveErrorType ErrorStatus )`	
Service ID [hex]	0x7b	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different Channels. Non reentrant for the same Channel.	
Parameters (in)	Channel	Identification of the LIN channel
	ErrorStatus	Type of detected error
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	The LIN Driver will call this function to report a detected error event during header or response processing. This function is only applicable for LIN slave nodes (available only if the ECU has any LIN slave channel).	
Available via	LinIf.h	

⌋

[SWS_LinIf_00855] ⌈If development error detection is enabled and an invalid value for `Channel` is given, the function `LinIf_LinErrorIndication` shall report `LINIF_E_NONEXISTENT_CHANNEL` to the default error tracer.⌋

[SWS_LinIf_00866] ⌈The function `LinIf_LinErrorIndication` is only available if the LinIf module is configured as LIN slave node on at least one channel. In a pure LIN master configuration, this function is not available. This depends on the configuration parameter `LinIfNodeType`.⌋

8.3.21 LinTp_Init

[SWS_LinIf_00350] Definition of API function LinTp_Init

Upstream requirements: SRS_BSW_00101, SRS_BSW_00414, SRS_BSW_00416, SRS_BSW_00358

⌈

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.	
Available via	LinIf.h	

⌋

[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.⌋

8.3.22 LinTp_Transmit

[SWS_LinIf_00351] Definition of API function LinTp_Transmit ⌈

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

▽

△

Service ID [hex]	0x53	
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.	
Available via	LinIf.h	

⌋

[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. This requirement is only applicable to LIN master nodes.⌋

[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. This requirement is only applicable to LIN master nodes.⌋

[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.⌋

Note: According to the ISO 17987 specifications, the NAD 0x7E shall be used for a functional transmission request.

[SWS_LinIf_00586] ⌈The LIN Interface shall use the NAD 0x7E for transmission of functional requests by LIN master nodes.⌋

[SWS_LinIf_00702] ⌈When LinTp_Transmit was successful (returned E_OK), the LIN Interface shall call PduR_LinTpTxConfirmation after successful completion or failure (e.g., since some other schedule table change is currently going on) of required schedule table switch (see [SWS_LinIf_00646], [SWS_LinIf_00642], [SWS_LinIf_00643], [SWS_LinIf_00707]), with a negative or positive result. When LinTp_Transmit was not successful, PduR_LinTpTxConfirmation shall not be called.⌋

[SWS_LinIf_00878] ⌈The function LinTp_Transmit shall return E_NOT_OK when the corresponding LinIf_ScheduleRequest call (called from BswM_LinTp_RequestMode resulting from the LinTp_Transmit call, see [SWS_LinIf_00646] and [SWS_LinIf_00641]) returned E_NOT_OK for the Schedule Request.⌋

[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 [SRS_BSW_00487].

8.3.23 LinTp_Shutdown

[SWS_LinIf_00355] Definition of API function LinTp_Shutdown

Upstream requirements: SRS_BSW_00336

⌈

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.
Available via	LinIf.h

⌋

[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 [SRS_BSW_00487].

8.3.24 LinTp_ChangeParameter

[SWS_LinIf_00501] Definition of API function LinTp_ChangeParameter ⌈

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).	
Available via	LinIf.h	

⌋

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_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, [SRS_BSW_00487].

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_<LinIfChannel.ShortName>

[SWS_LinIf_00384] Definition of scheduled function LinIf_MainFunction_<LinIfChannel.ShortName>

Upstream requirements: SRS_BSW_00373

⌈

Service Name	LinIf_MainFunction_<LinIfChannel.ShortName>
Syntax	`void LinIf_MainFunction_<LinIfChannel.ShortName> ( void )`
Service ID [hex]	0x80
Description	The main processing function of the LIN Interface.
Available via	SchM_LinIf.h

⌋

Design hint: The function LinIf_MainFunction_<LinIfChannel.ShortName> 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_<LinIfChannel.ShortName> shall exist once per LIN channel of the LIN Interface module.⌋

[SWS_LinIf_00726] ⌈The function name of each instance of the LinIf MainFunction shall be LinIf_MainFunction_<LinIfChannel.ShortName> where <LinIfChannel.ShortName> corresponds to the Short Name of the respective LIN channel (LinIfChannel).⌋

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

[SWS_LinIf_00286] ⌈The function LinIf_MainFunction_<LinIfChannel.ShortName> shall poll the Schedule Table Manager which frame shall be transported. This requirement is only applicable to LIN master nodes.⌋

[SWS_LinIf_00287] ⌈Only the function LinIf_MainFunction_<LinIfChannel.ShortName> shall process the transportation (transmission and reception) of frames. This requirement is only applicable to LIN master nodes.⌋

8.5 Expected interfaces

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

8.5.1 Mandatory Interfaces

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

[SWS_LinIf_00359] Definition of mandatory interfaces required by module LinIf ⌈

API Function	*Header File*	*Description*
Det_ReportRuntimeError	Det.h	Service to report runtime errors. If a callout has been configured then this callout shall be called.
Lin_GoToSleepInternal	Lin.h	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_Wakeup	Lin.h	Generates a wake up pulse and sets the channel state to LIN_CH_OPERATIONAL.
Lin_WakeupInternal	Lin.h	Sets the channel state to LIN_CH_OPERATIONAL without generating a wake up pulse.

⌋

8.5.2 Optional interfaces

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

[SWS_LinIf_00360] Definition of optional interfaces requested by module LinIf ⌈

API Function	*Header File*	*Description*
BswM_LinTp_RequestMode	BswM_LinTp.h	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.
Com_SendSignal	Com.h	The service Com_SendSignal updates the signal object identified by SignalId with the signal referenced by the SignalDataPtr parameter.
Det_ReportError	Det.h	Service to report development errors.
Lin_CheckWakeup	Lin.h	This function checks if a wakeup has occurred on the addressed LIN channel.
Lin_GetStatus	Lin.h	Gets the status of the LIN driver. Only used for LIN master nodes.
Lin_GoToSleep	Lin.h	The service instructs the driver to transmit a go-to-sleep-command on the addressed LIN channel. Only used for LIN master nodes.

▽

△

API Function	Header File	Description
Lin_SendFrame	Lin.h	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. Only used for LIN master nodes.
LinSM_GotoSleepConfirmation	LinSM.h	The LinIf will call this callback when the go to sleep command is sent successfully or not sent successfully on the network.
LinSM_GotoSleepIndication	LinSM.h	The LinIf will call this callback when the go to sleep command is received on the network or a bus idle timeout occurs. Only applicable for LIN slave nodes.
LinSM_ScheduleRequestConfirmation	LinSM.h	The LinIf module will call this callback when the new requested schedule table is active.
LinSM_WakeupConfirmation	LinSM.h	The LinIf will call this callback when the wake up signal command is sent not successfully/ successfully on the network.
LinTrcv_CheckWakeup	LinTrcv.h	Notifies the calling function if a wakeup is detected.
LinTrcv_GetBusWuReason	LinTrcv.h	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	LinTrcv.h	API detects the actual software state of LIN transceiver driver.
LinTrcv_SetOpMode	LinTrcv.h	The internal state of the LIN transceiver driver is switched to mode given in the parameter OpMode.
LinTrcv_SetWakeupMode	LinTrcv.h	This API enables, disables and clears the notification for wakeup events on the addressed network.
LSduR_LinIfRxIndication (draft)	LSduR_LinIf.h	Indication of a received PDU from a lower layer communication interface module.
LSduR_LinIfTriggerTransmit (draft)	LSduR_LinIf.h	Within this API, the upper layer module (called module) shall check whether the available data fits into the buffer size reported by PduInfoPtr->Sdu Length. 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->Sdu Length. If not, it returns E_NOT_OK without changing PduInfoPtr.
LSduR_LinIfTxConfirmation (draft)	LSduR_LinIf.h	The lower layer communication interface module confirms the transmission of a PDU, or the failure to transmit a PDU.
Mirror_ReportLinFrame	Mirror.h	Reports a received or transmitted LIN frame.
PduR_LinTpCopyRxData	PduR_LinTp.h	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.

▽

△

API Function	Header File	Description
PduR_LinTpCopyTxData	PduR_LinTp.h	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	PduR_LinTp.h	Called after an I-PDU has been received via the TP API, the result indicates whether the reception was successful or not.
PduR_LinTpStartOfReception	PduR_LinTp.h	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 TpSdu Length equal to 0.
PduR_LinTpTxConfirmation	PduR_LinTp.h	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 Callout interfaces

8.5.3.1 <User_ResponseErrorSignalChanged>

The callout function <User_ResponseErrorSignalChanged> is only applicable for LIN slave node.

[SWS_LinIf_00856] Definition of configurable interface <User_ResponseErrorSignalChanged> ⌈

Service Name	<User_ResponseErrorSignalChanged>	
Syntax	`void <User_ResponseErrorSignalChanged> ( NetworkHandleType Channel, boolean RespErrSigValue )`	
Service ID [hex]	0x74	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different Channels. Non reentrant for the same Channel.	
Parameters (in)	Channel	Identification of the LIN channel.
	RespErrSigValue	Current value of the response error signal. True if the signal is set to 1, false otherwise. Only applicable for LIN slave nodes.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	

▽

△

Description	Notifies the change of the response error signal with the new value. Only applicable for LIN slave nodes.
Available via	LinIf_Externals.h

⌋

This callout is optional, see [SWS_LinIf_00766].

Configuration of `<User_ResponseErrorSignalChanged>`: The name of the API `<User_ResponseErrorSignalChanged>` which will be called by the LIN Interface module shall be configured for the LIN Interface module by parameter `LinIfResponseErrorSignalChangedCallout`.

8.5.3.2 `<User_SaveConfigurationRequest>`

The callout function `<User_SaveConfigurationRequest>` is only applicable for LIN slave node.

[SWS_LinIf_00857] Definition of configurable interface <User_SaveConfiguration Request> ⌈

Service Name	<User_SaveConfigurationRequest>	
Syntax	`boolean <User_SaveConfigurationRequest> ( NetworkHandleType Channel )`	
Service ID [hex]	0x75	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Channel	Identification of the LIN channel.
Parameters (inout)	None	
Parameters (out)	None	
Return value	boolean	True if a positive response shall be transmitted, false if no response shall be transmitted.
Description	Notifies the reception of a SaveConfiguration request. Only applicable for LIN slave nodes.	
Available via	LinIf_Externals.h	

⌋

This callout is optional, depending if the `SaveConfiguration` node configuration service is directly supported (configuration parameter `LinIfSaveConfigurationCallout`).

Configuration of `<User_SaveConfigurationRequest>`: The name of the API `<User_SaveConfigurationRequest>` which will be called by the LIN Interface module shall be configured for the LIN Interface module by parameter `LinIfSaveConfigurationCallout`.

Note: The functions `LinIf_GetConfiguredNAD` and `LinIf_GetPIDTable` are intended to be used after reception of a `SaveConfiguration` request to read the current LIN configuration.

8.5.4 Configurable interfaces

In this section, 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 section 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).

8.5.4.1 `<User>_ScheduleRequestConfirmation`

[SWS_LinIf_00520] Definition of configurable interface < User >_ScheduleRequestConfirmation ⌈

Service Name	< User >_ScheduleRequestConfirmation	
Syntax	`void < User >_ScheduleRequestConfirmation ( NetworkHandleType channel, LinIf_SchHandleType ScheduleTableIdx )`	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	channel	Identification of the LIN channel
	ScheduleTableIdx	Index of the scheduled 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.	
Available via	Configuration parameters LinIfScheduleRequestConfirmationUL and LinIfPublicCddHeaderFile of the corresponding LinIfChannel	

⌋

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.

8.5.4.2 <User>_GotoSleepConfirmation

[SWS_LinIf_00521] Definition of configurable interface < User >_GotoSleepConfirmation ⌈

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.	
Available via	Configuration parameters LinIfGotoSleepConfirmationUL and LinIfPublicCddHeaderFile of the corresponding LinIfChannel	

⌋

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.

8.5.4.3 <User>_WakeupConfirmation

[SWS_LinIf_00522] Definition of configurable interface < User >_WakeupConfirmation ⌈

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.	
Available via	Configuration parameters LinIfWakeupConfirmationUL and LinIfPublicCddHeaderFile of the corresponding LinIfChannel	

⌋

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`.

8.5.4.4 `<User>_GotoSleepIndication`

[SWS_LinIf_00880] Definition of configurable interface <User>_GotoSleepIndication ⌈

Service Name	<User>_GotoSleepIndication	
Syntax	`void <User>_GotoSleepIndication ( NetworkHandleType Channel )`	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different Channels	
Parameters (in)	Channel	Identification of the LIN channel
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	The LinIf will call this callback when the go to sleep command is received on the network or a bus idle timeout occurs. Only applicable for LIN slave nodes.	
Available via	Configuration parameters LinIfGotoSleepIndicationUL and LinIfPublicCddHeaderFile of the corresponding LinIfChannel	

⌋

Configuration of `<User>_GotoSleepIndication`: The name of the API `<User>_GotoSleepIndication` which will be called by the LIN Interface module shall be configured for the LIN Interface module by parameter `LinIfGotoSleepIndicationUL`.

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

9.1.1 Frame transmission in master nodes

This section is only applicable to LIN master nodes.

The following use case shows the transmission of a LIN frame. The first call of the `LinIf_MainFunction_<LinIfChannel.ShortName>` 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_<LinIfChannel.ShortName>`, the frame is finished.

The `RequestFrame` call in the diagram is the interface call to the Schedule Table Manager. The `LinIf_MainFunction_<LinIfChannel.ShortName>` 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 9.1 does not show any timing information. The timing information is depicted in Figure 9.2 following the diagram.

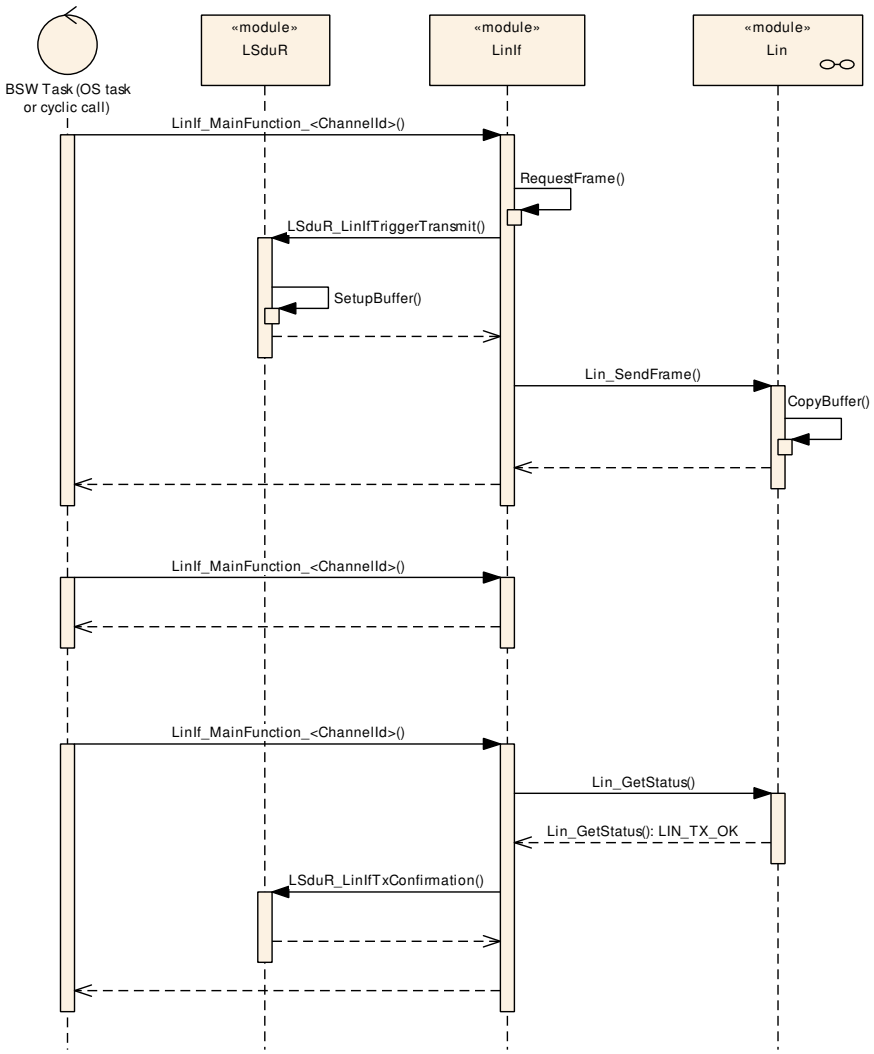


Figure 9.1: Frame transmission (master node)

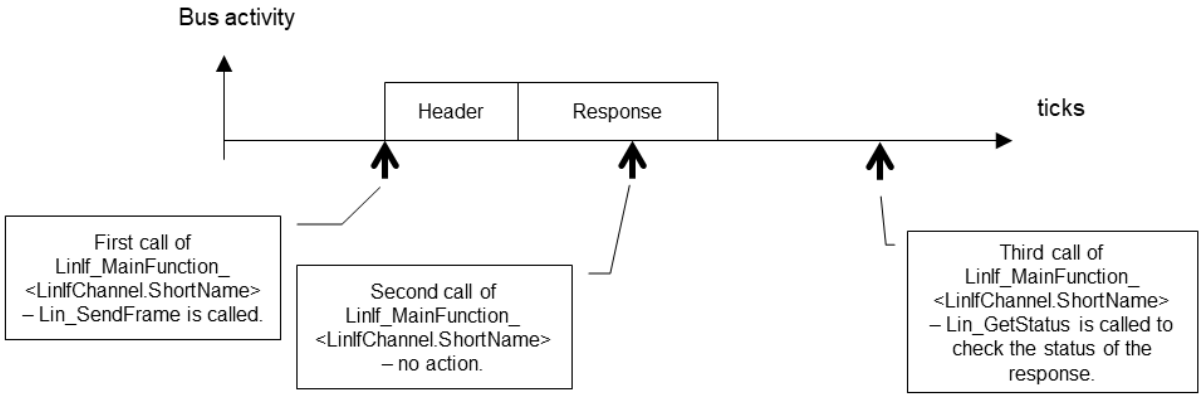


Figure 9.2: Timing information for transmitted frame

9.1.2 Frame transmission in slave nodes

This section is only applicable to LIN slave nodes.

The following use case (Figure 9.3) shows the transmission of a LIN frame. After a received LIN header is indicated by the LIN driver, the PID is evaluated (shown by function `EvaluatePID`). If it's determined to belong to a transmission frame, the transmission response data is requested from upper layer. The buffer to which `SduPtr` member in `PduPtr` parameter points to can be directly passed to let the upper layer copy the transmission data to the provided buffer. The LIN interface informs the driver about the response type by setting the appropriate members of `PduPtr` (shown by function `SetPduPtrValues`).

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

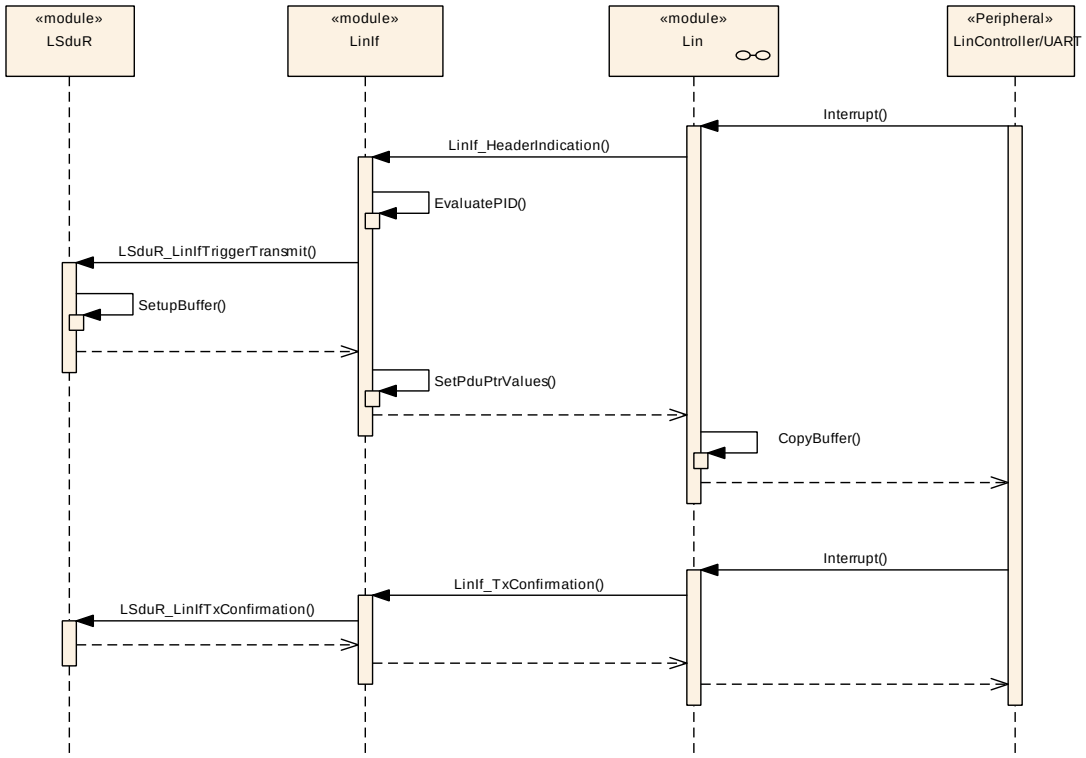


Figure 9.3: Frame transmission (slave node)

9.2 Frame Reception

9.2.1 Frame reception in master nodes

This section is only applicable to LIN master nodes.

The following use case (Figure 9.4) shows the reception of a LIN frame. The first call of the `LinIf_MainFunction_<LinIfChannel.ShortName>` 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_<LinIfChannel.ShortName>` 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.

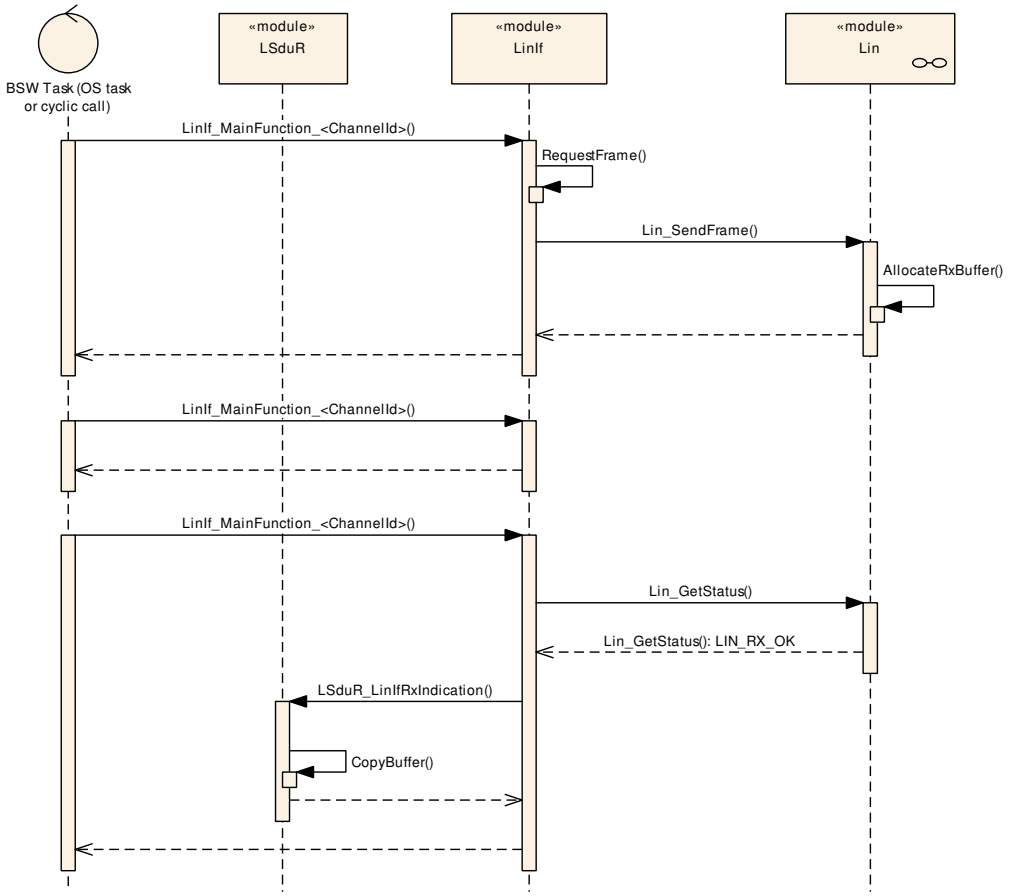


Figure 9.4: Frame reception (master node)

9.2.2 Frame reception in slave nodes

This section is only applicable to LIN slave nodes.

The following use case (Figure 9.5) shows the reception of a LIN frame. After a received LIN header is indicated by the LIN driver, the PID is evaluated (shown by function `EvaluatePID`). If it's determined to belong to a reception frame, the LIN interface informs the driver about the response type by setting the appropriate members of `PduPtr` (shown by function `SetPduPtrValues`).

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.

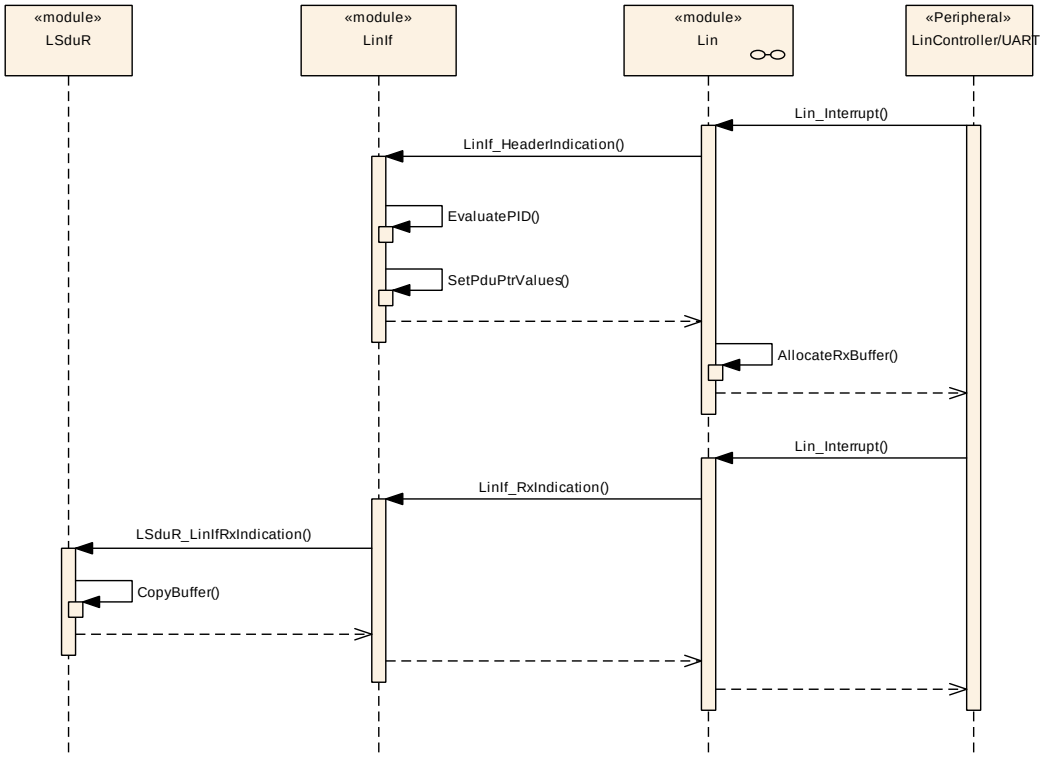


Figure 9.5: Frame reception (slave node)

9.3 Slave-to-slave / Irrelevant communication

9.3.1 Slave-to-slave communication in master nodes

This section is only applicable to LIN master nodes.

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 9.6).

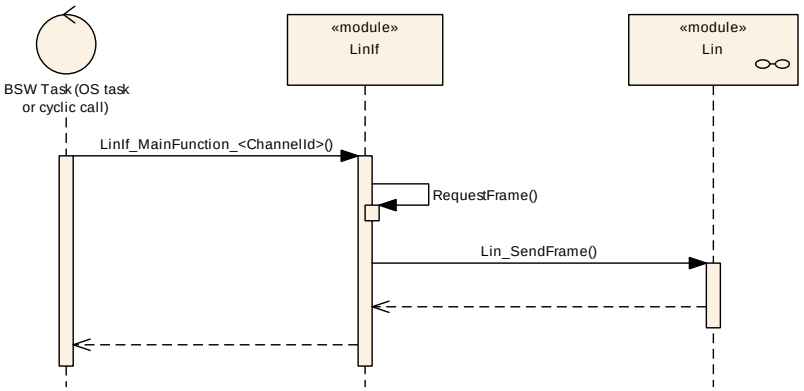


Figure 9.6: Slave-to-slave communication (master)

9.3.2 Irrelevant communication in slave nodes

This section is only applicable to LIN slave nodes.

The third direction for a LIN frame is that the frame is not relevant for the slave node and is ignored. After a received LIN header is indicated by the LIN driver, the PID is evaluated (shown by function `EvaluatePID`). If it's determined to belong to an `Irrelevant frame`, the LIN interface informs the driver about the response type by setting the appropriate members of `PduPtr` (shown by function `SetPduPtrValues`). No further action is made (Figure 9.7).

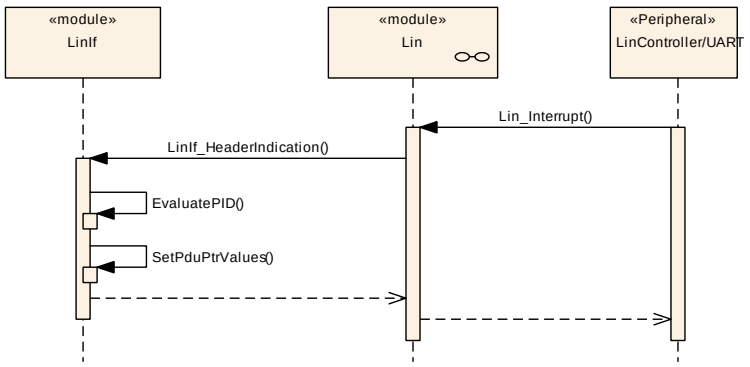


Figure 9.7: Irrelevant communication (slave node)

9.4 Sporadic frame (Master only)

This section is only applicable to LIN master nodes. For LIN slave nodes, Sporadic frames are handled like reception of Unconditional frames.

The following use case (Figure 9.8) 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_<LinIfChannel.ShortName> 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.

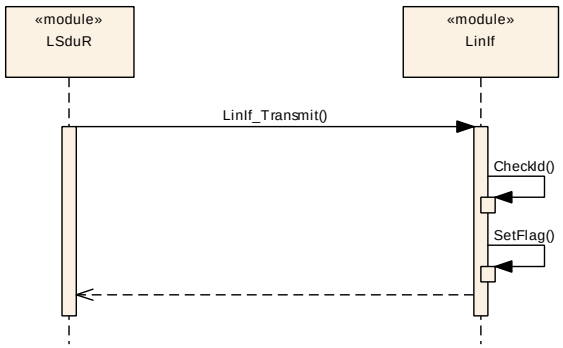


Figure 9.8: Sporadic frame (master node)

9.5 Event-triggered frame

9.5.1 Event-triggered frame in master nodes

This section is only applicable to LIN master nodes.

There are three results for an `Event-triggered frame`:

- No answer
- One slave node answers
- Two or more slaves answers so that there is a collision on the bus

All three use cases are shown below.

9.5.1.1 With no answer

The following use case (Figure 9.9) shows the transmission of an Event-triggered frame header and no response.

The first call of the LinIf_MainFunction_<LinIfChannel.ShortName> 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_<LinIfChannel.ShortName> 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_<LinIfChannel.ShortName> recognizes this situation and takes no action since this is not considered to be a communication error.

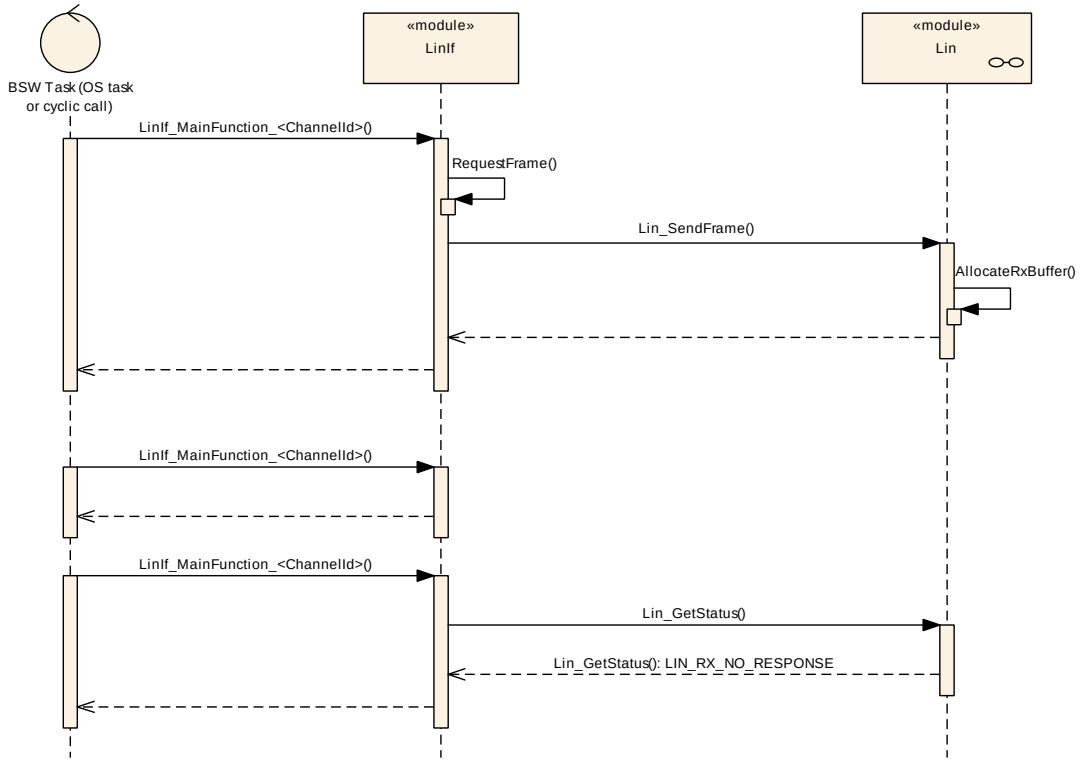


Figure 9.9: Event-triggered frame with no answer (master node)

9.5.1.2 With answer (No collision)

The following use case (Figure 9.10) shows the transmission of an `Event-triggered frame` header with a response from one slave.

The first call of the `LinIf_MainFunction_<LinIfChannel.ShortName>` 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_<LinIfChannel.ShortName>` 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.

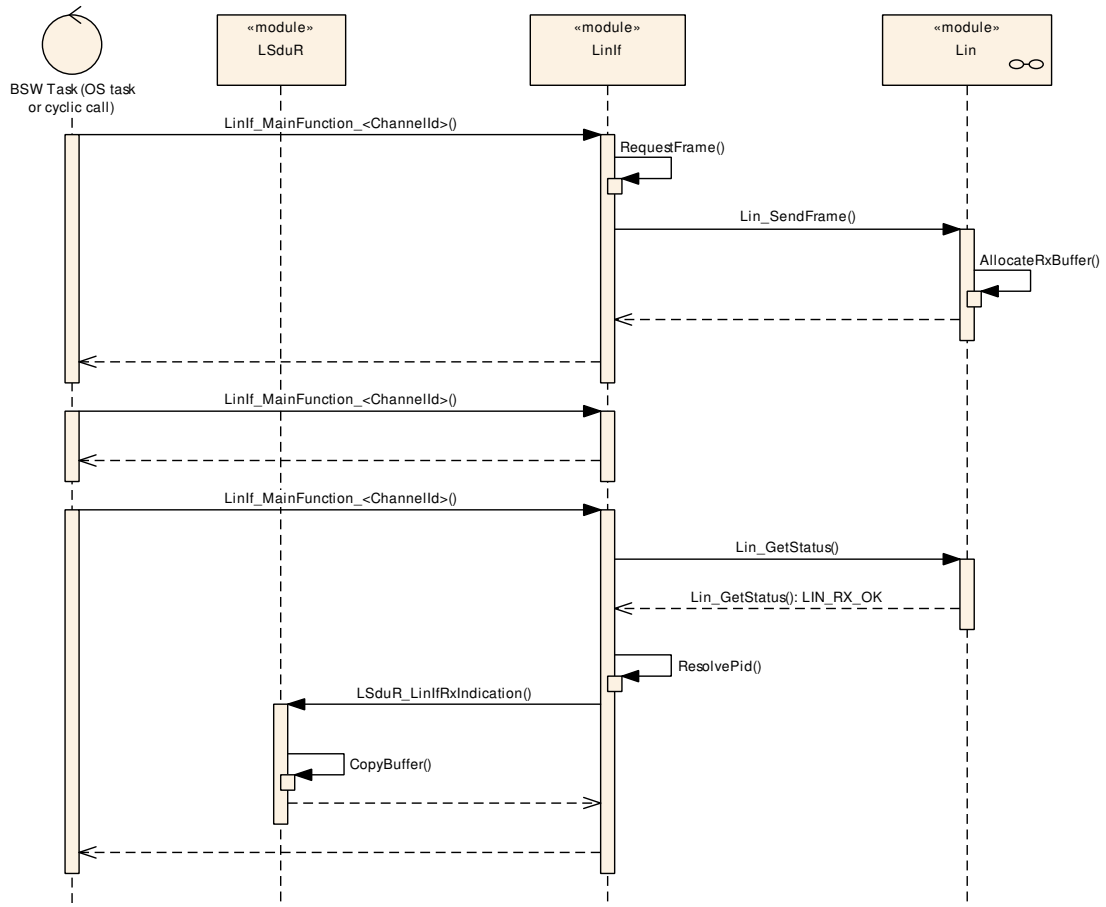


Figure 9.10: `Event-triggered frame` with answer (no collision) (master node)

9.5.1.3 With collision

The following use case (Figure 9.11) 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_<LinIfChannel.ShortName>` 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_<LinIfChannel.ShortName>` 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.

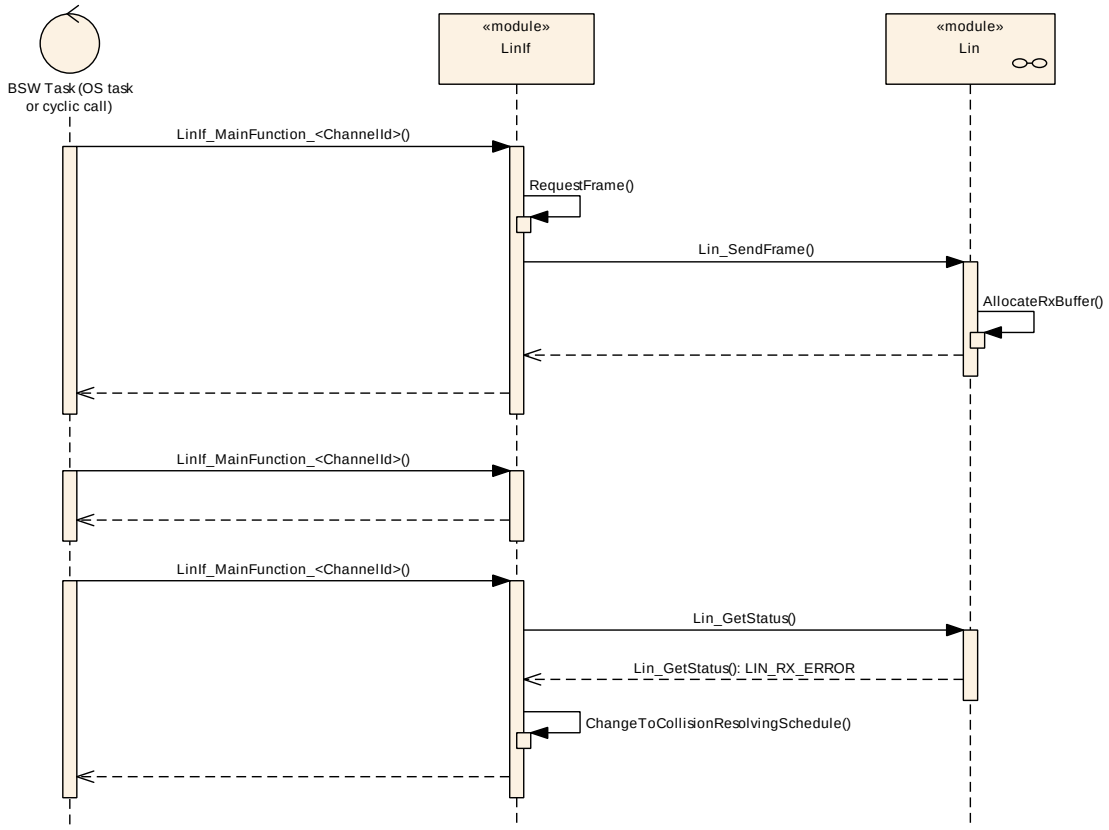


Figure 9.11: `Event-triggered frame` with collision (master node)

9.5.2 Event-triggered frame in slave nodes

This section is only applicable to LIN slave nodes.

The following use case (Figure 9.12) shows an upper layer requesting transmission of an Event-triggered frame. It just marks the frame for transmission. When the header of the Event-triggered frame is received for which the associated response is pending, the transmission is requested from upper layer like for an unconditional transmission frame.

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 Event-triggered frame for transmission in the LIN Interface. There is one flag for each Event-triggered frame.

The CopyPID function is to show that that the LIN Interface must copy the PID of the requested Event-triggered frame to the first byte of the payload data.

The ClearFlag function is a local function to clear the pending flag of an Event-triggered frame after successful transmission.

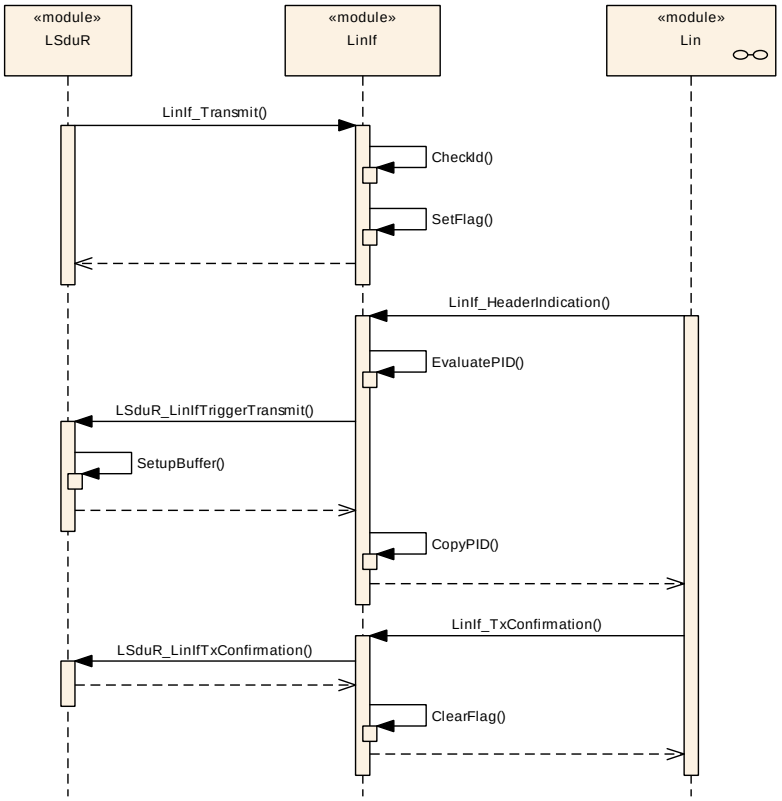


Figure 9.12: Event-triggered frame (slave node)

9.6 Transport Protocol message transmission

The following diagram (Figure 9.13) 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/SRF 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. For LIN master nodes, 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. For LIN master nodes, 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).

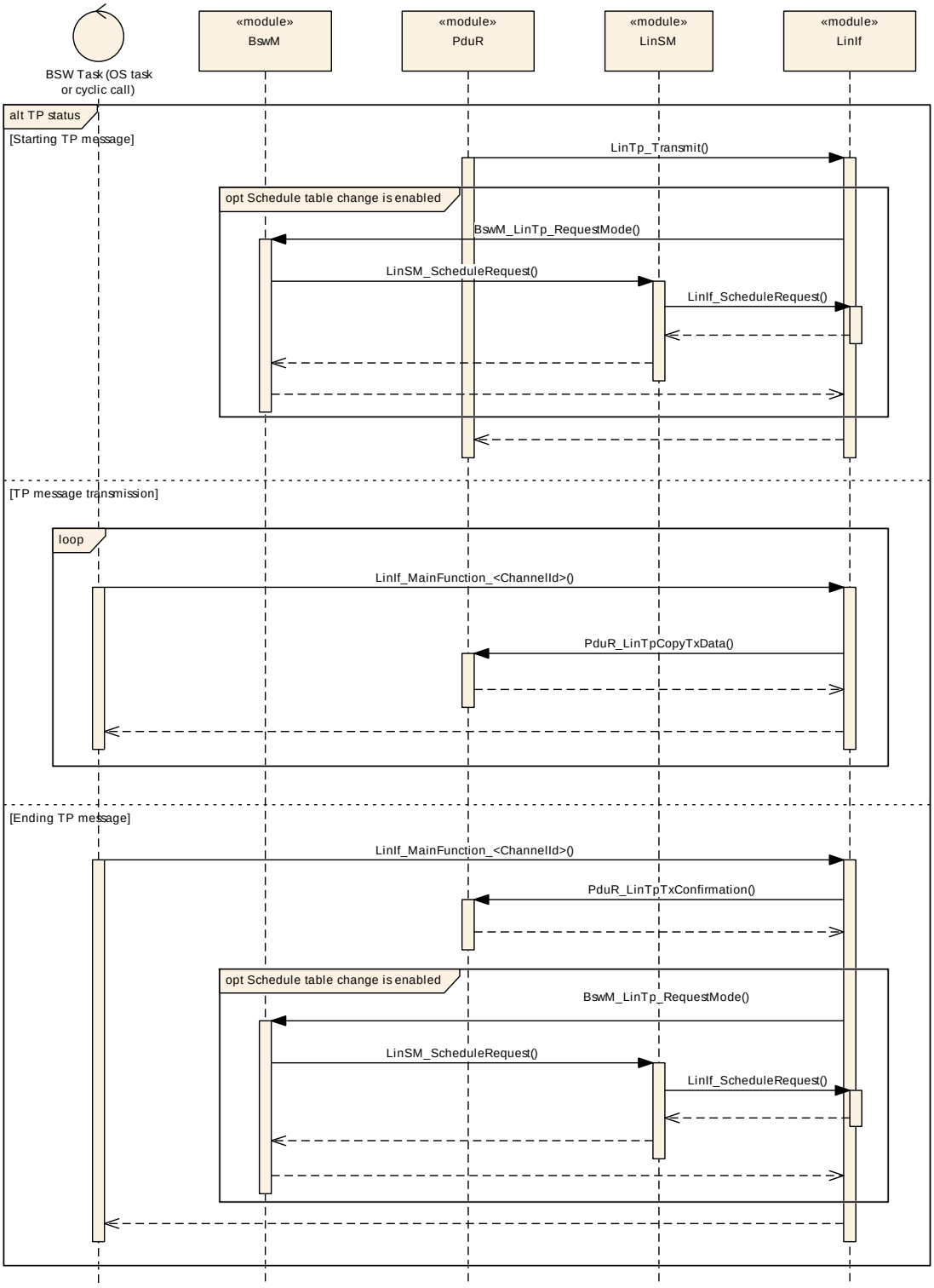


Figure 9.13: Transport Protocol message transmission

9.7 Transport Protocol message reception

The following diagram (Figure 9.14) 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 MRF/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. For LIN master nodes, 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)

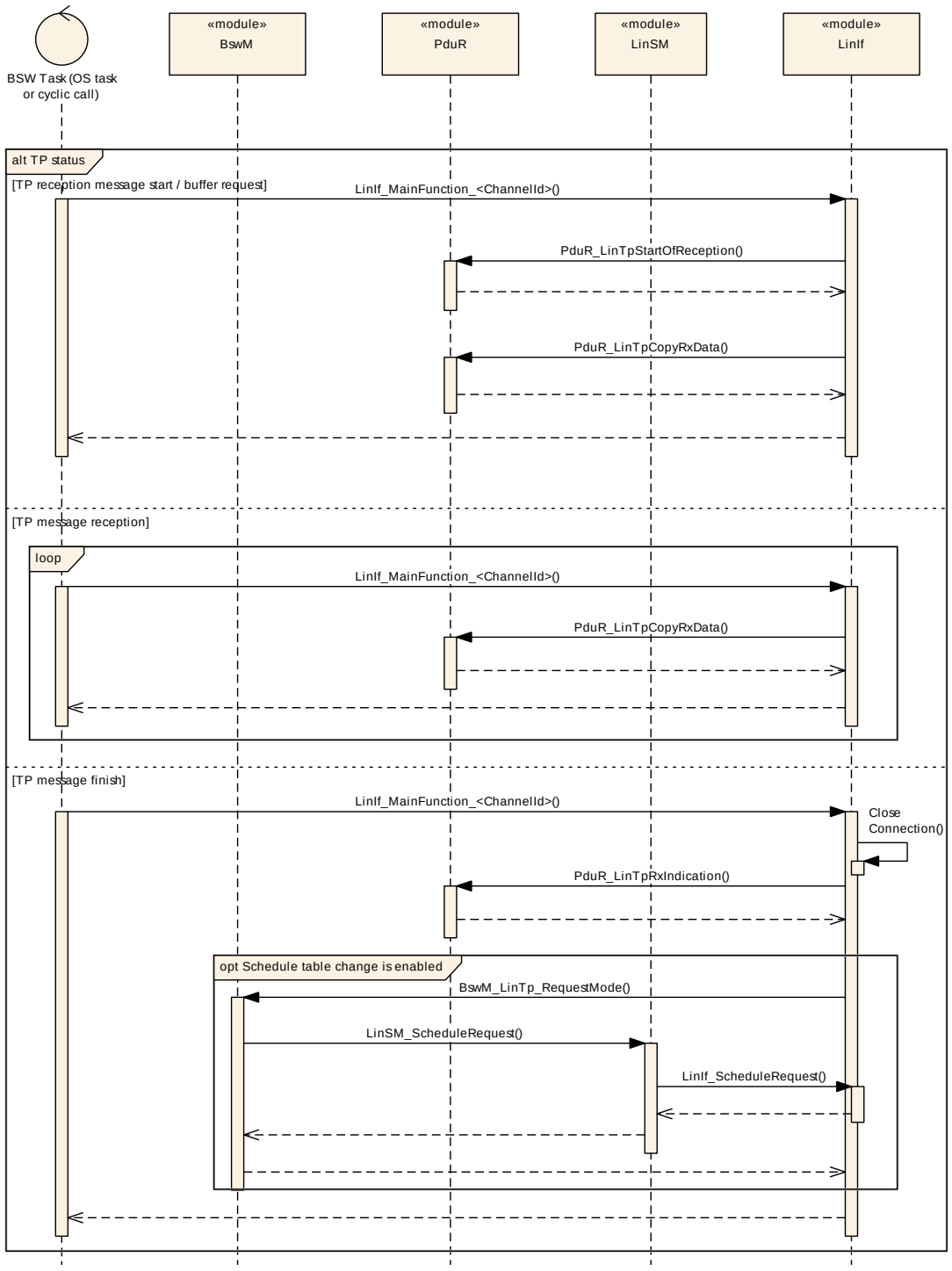


Figure 9.14: Transport Protocol message reception

9.8 Go to sleep process

9.8.1 Go to sleep process in master nodes

This section is only applicable to LIN master nodes.

This use case in Figure 9.15 shows the execution of the `LinIf_GotoSleep` command.

The `LinIf_MainFunction_<LinIfChannel.ShortName>` 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.

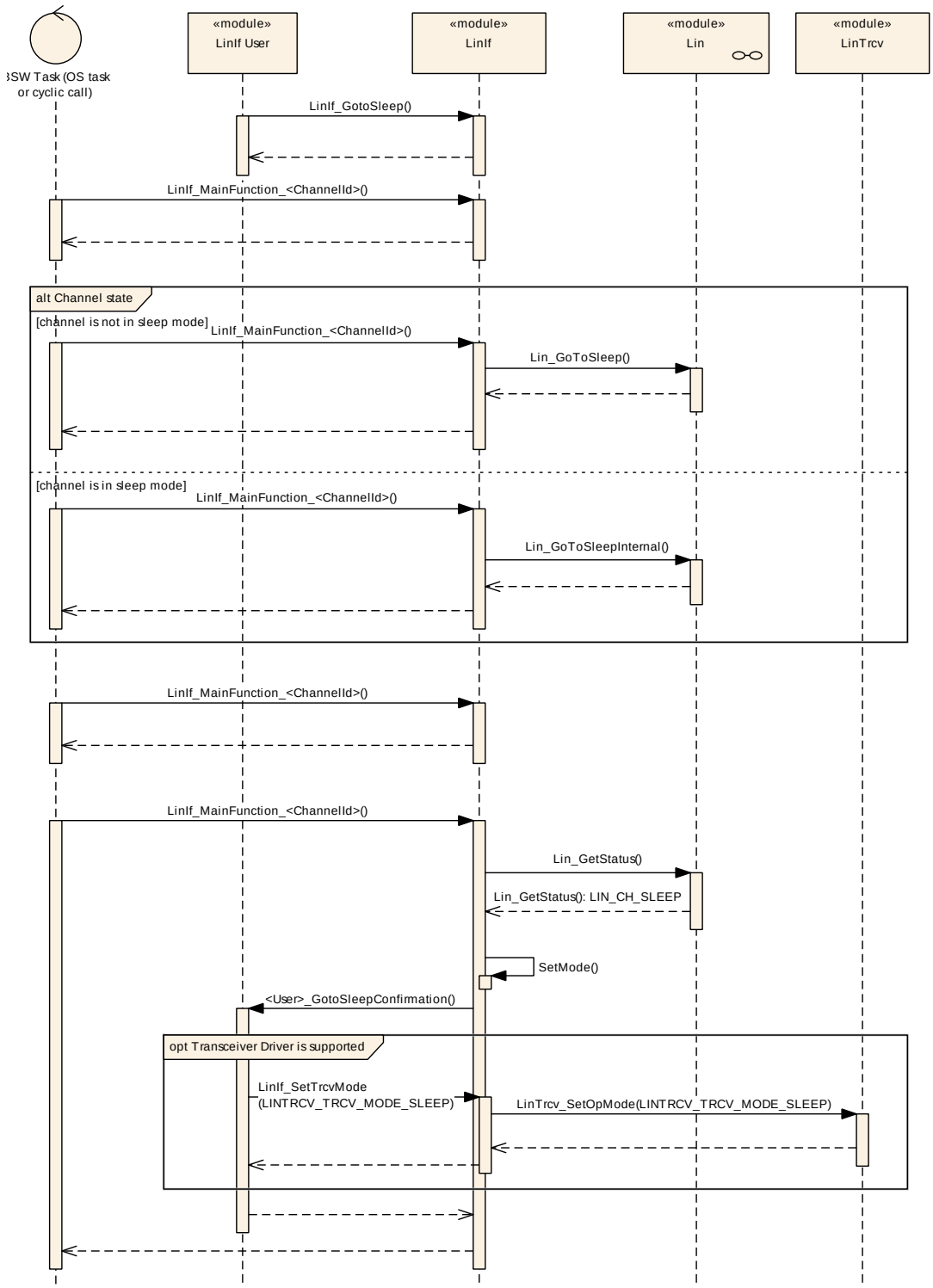


Figure 9.15: Go-to-sleep command process (master nodes)

9.8.2 Go to sleep process in slave nodes

This section is only applicable to LIN slave nodes.

This use case in Figure 9.16 shows the execution of the sleep transition, either caused by reception of a go-to-sleep command or by the detection of a bus idle condition.

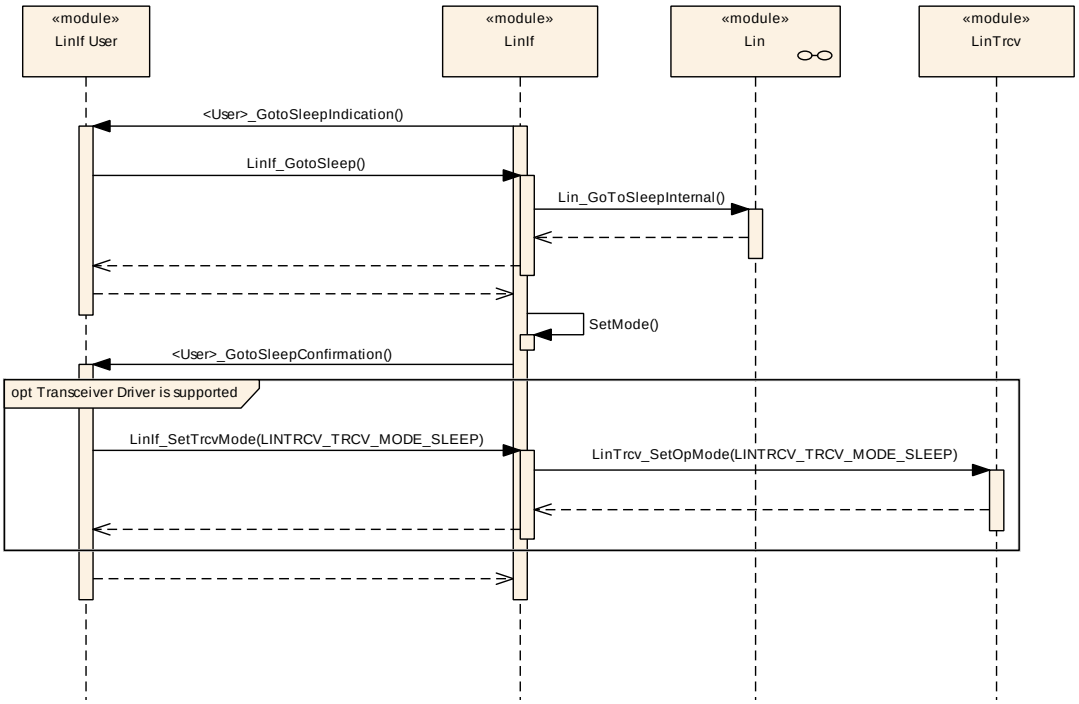


Figure 9.16: Go-to-sleep process (slave node)

9.9 Wake up request

The wake-up use cases are described in chapter 9 of the AUTOSAR Specification of the ECU State Manager [11].

9.10 Internal wake-up

There are two different use cases in Figure 9.17:

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

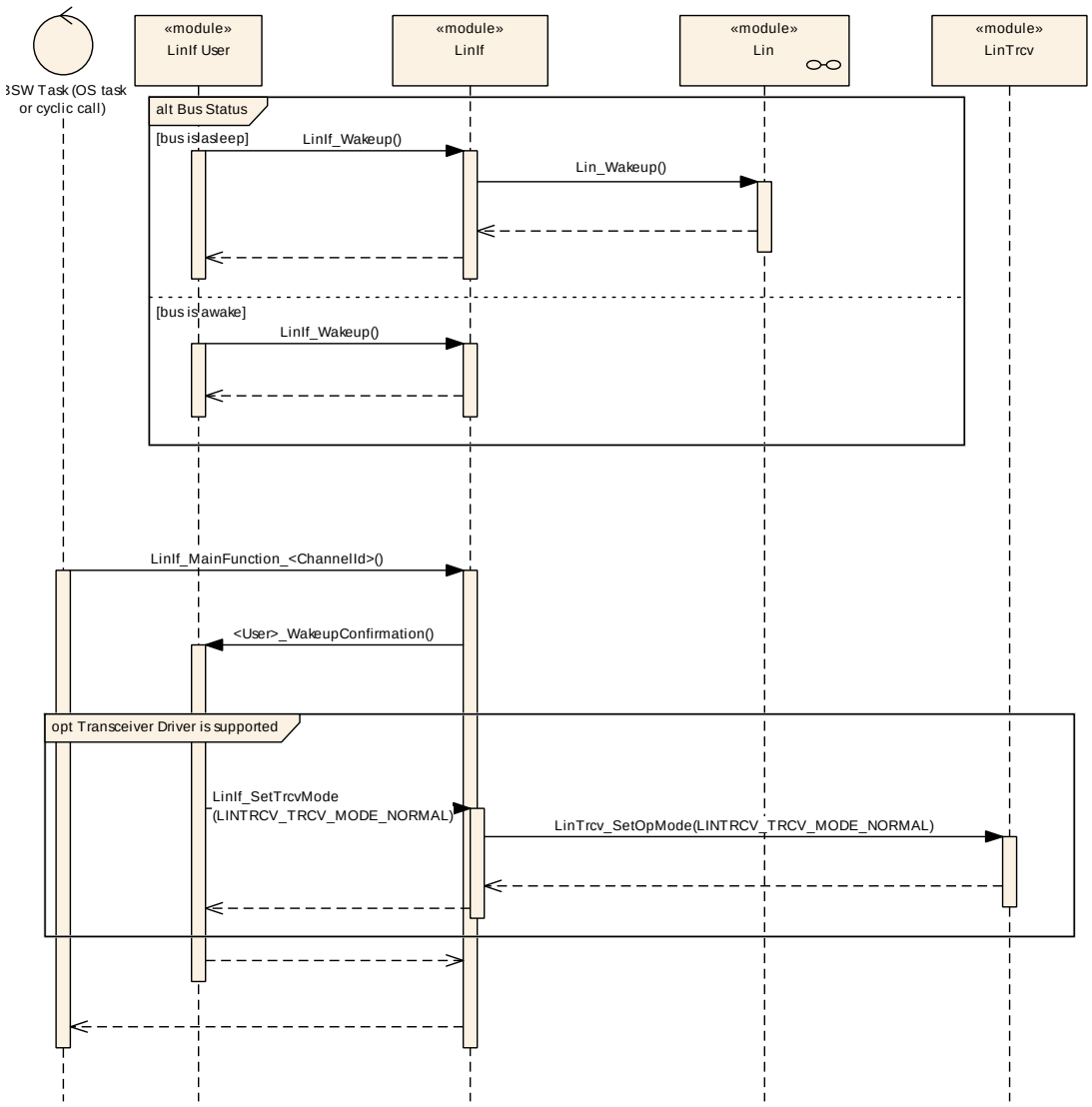


Figure 9.17: Internal wake-up

10 Configuration specification

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

The Chapter 10.3 (the LinIf part) and Chapter 10.4 (the LinTp part) specifiy the structure (containers) and the parameters of the module LIN Interface.

The Chapter 10.6 specifies published information of the module LIN Interface.

10.1 How to read this chapter

For details refer to [8] Chapter 10.1 *"Introduction to configuration specification"*.

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 structures `LinIf` and LIN Transport Layer configuration structures `LinTp` shall be constructed so that it may be exchangeable in memory.⌋

Example: The `LinIf` 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 ISO 17987 specifications 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]

Upstream requirements: SRS_BSW_00167

⌈The LIN Interface shall not make any consistency check of the configuration in runtime in production software. It may be done if the development error detection is enabled.⌋

10.3 LinIf Configuration

The Figure 10.1 depicts the LIN Interface configuration.

10.3.1 LinIf

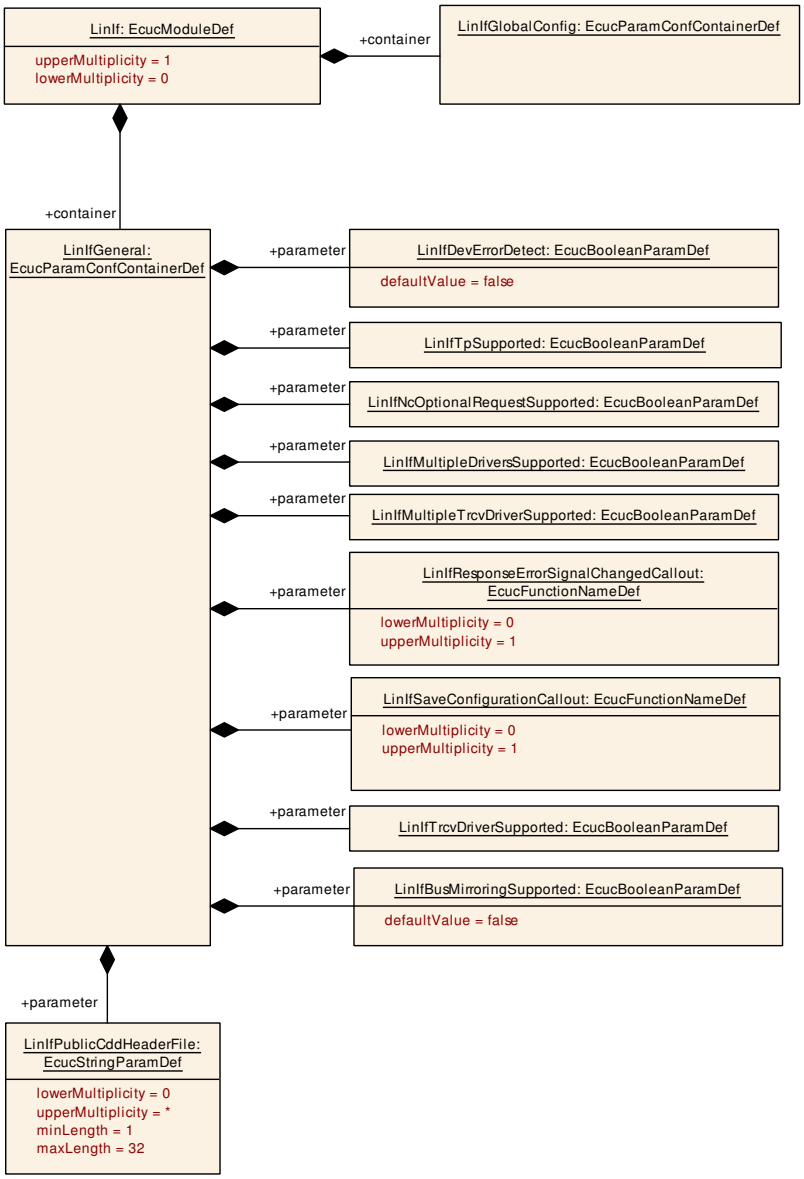


Figure 10.1: LIN Interface configuration

[ECUC_LinIf_00370] Definition of EcucModuleDef LinIf ⌈

Module Name	LinIf
Description	Configuration of the LinIf (LIN Interface) module.
Post-Build Selectable Support	true
Post-Build Loadable Support	true

Included Containers		
Container Name	Multiplicity	Description
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

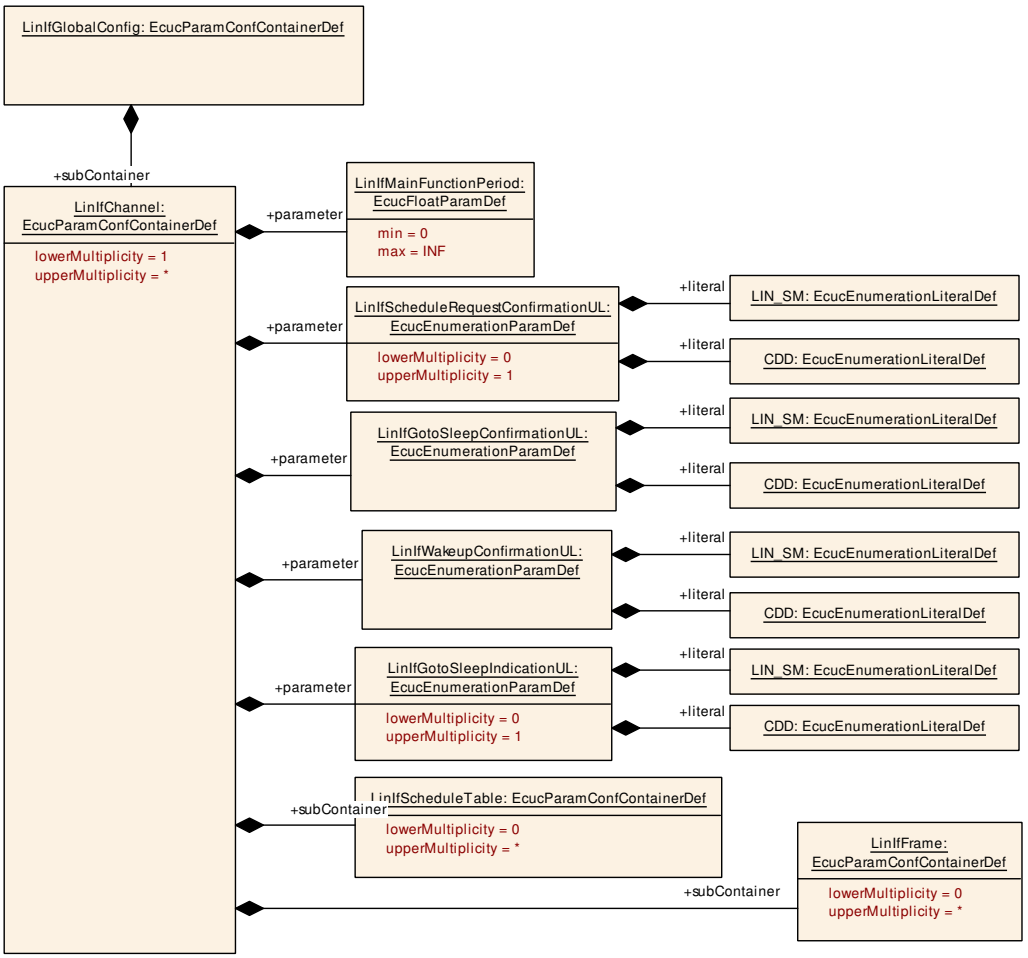


Figure 10.2: LIN Interface Global configuration

[ECUC_LinIf_00020] Definition of EcucParamConfContainerDef LinIfGlobalConfig ⌈

Container Name	LinIfGlobalConfig
Parent Container	LinIf
Upstream mapping defined	[22, CP_TPS_SystemTemplate]
Description	This container contains the global configuration parameters of the LinIf.
Multiplicity	1
Pre-Compile only	true
Configuration Parameters	

No Included Parameters

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

⌋

10.3.3 LinIfGeneral

[ECUC_LinIf_00019] Definition of EcucParamConfContainerDef LinIfGeneral ⌈

Container Name	LinIfGeneral
Parent Container	LinIf
Upstream mapping defined	[22, CP_TPS_SystemTemplate]
Description	This container contains the general parameters of LIN Interface module.
Multiplicity	1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	Multiplicity	ECUC ID
LinIfBusMirroringSupported	1	[ECUC_LinIf_00657]
LinIfDevErrorDetect	1	[ECUC_LinIf_00010]
LinIfMultipleDriversSupported	1	[ECUC_LinIf_00024]
LinIfMultipleTrcvDriverSupported	1	[ECUC_LinIf_00025]
LinIfNcOptionalRequestSupported	1	[ECUC_LinIf_00026]
LinIfPublicCddHeaderFile	0..*	[ECUC_LinIf_00631]
LinIfResponseErrorSignalChangedCallout	0..1	[ECUC_LinIf_00656]
LinIfSaveConfigurationCallout	0..1	[ECUC_LinIf_00651]
LinIfTpSupported	1	[ECUC_LinIf_00045]
LinIfTrcvDriverSupported	1	[ECUC_LinIf_00635]

No Included Containers

⌋

[ECUC_LinIf_00657] Definition of EcucBooleanParamDef LinIfBusMirroringSupported ⌈

Parameter Name	LinIfBusMirroringSupported
Parent Container	LinIfGeneral
Upstream mapping defined	–
Description	States if Bus Mirroring is enabled in the LIN Interface or not. The reason for this parameter is to reduce the size of LIN Interface if the Bus Mirroring is not used.
Multiplicity	1
Type	EcucBooleanParamDef
Default value	false
Pre-Compile only	true

⌋

[ECUC_LinIf_00010] Definition of EcucBooleanParamDef LinIfDevErrorDetect ⌈

Parameter Name	LinIfDevErrorDetect
Parent Container	LinIfGeneral
Upstream mapping defined	–
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
Pre-Compile only	true

⌋

[ECUC_LinIf_00024] Definition of EcucBooleanParamDef LinIfMultipleDriversSupported ⌈

Parameter Name	LinIfMultipleDriversSupported
Parent Container	LinIfGeneral
Upstream mapping defined	–
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	–
Pre-Compile only	true

⌋

[ECUC_LinIf_00025] Definition of EcucBooleanParamDef LinIfMultipleTrcvDriverSupported ⌈

Parameter Name	LinIfMultipleTrcvDriverSupported
Parent Container	LinIfGeneral
Upstream mapping defined	–

▽

△

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	–
Pre-Compile only	true

⌋

[ECUC_LinIf_00026] Definition of EcucBooleanParamDef LinIfNcOptionalRequestSupported ⌈

Parameter Name	LinIfNcOptionalRequestSupported
Parent Container	LinIfGeneral
Upstream mapping defined	–
Description	States if the node configuration commands Assign NAD and Conditional Change NAD are supported. Dependency: Only applicable when the channel contains a LinIfMaster container.
Multiplicity	1
Type	EcucBooleanParamDef
Default value	–
Pre-Compile only	true

⌋

[ECUC_LinIf_00631] Definition of EcucStringParamDef LinIfPublicCddHeaderFile ⌈

Parameter Name	LinIfPublicCddHeaderFile
Parent Container	LinIfGeneral
Upstream mapping defined	–
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	–
Length	1-32
Regular Expression	–
Pre-Compile only	true

⌋

[ECUC_LinIf_00656] Definition of EcucFunctionNameDef LinIfResponseErrorSignalChangedCallout ⌈

Parameter Name	LinIfResponseErrorSignalChangedCallout
Parent Container	LinIfGeneral
Upstream mapping defined	–

▽

△

Description	This parameter contains the name of the callout function that is called after a response error signal change. Only applicable for LIN slave nodes. Dependency: Only applicable when at least one channel contains a LinIfSlave container.
Multiplicity	0..1
Type	EcucFunctionNameDef
Default value	–
Regular Expression	–
Pre-Compile only	true

⌋

[ECUC_LinIf_00651] Definition of EcucFunctionNameDef LinIfSaveConfiguration Callout ⌈

Parameter Name	LinIfSaveConfigurationCallout
Parent Container	LinIfGeneral
Upstream mapping defined	–
Description	This parameter contains the name of the callout function that is called when a save configuration node configuration command is processed by this slave node. The service is only supported when this parameter is configured. Only applicable for LIN slave nodes. Dependency: Only applicable when at least one channel contains a LinIfSlave container.
Multiplicity	0..1
Type	EcucFunctionNameDef
Default value	–
Regular Expression	–
Pre-Compile only	true

⌋

[ECUC_LinIf_00045] Definition of EcucBooleanParamDef LinIfTpSupported ⌈

Parameter Name	LinIfTpSupported
Parent Container	LinIfGeneral
Upstream mapping defined	–
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	–
Pre-Compile only	true

⌋

[ECUC_LinIf_00635] Definition of EcucBooleanParamDef LinIfTrcvDriverSupported ⌈

Parameter Name	LinIfTrcvDriverSupported
Parent Container	LinIfGeneral
Upstream mapping defined	–
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	–
Pre-Compile only	true

⌋

10.3.4 LinIfChannel

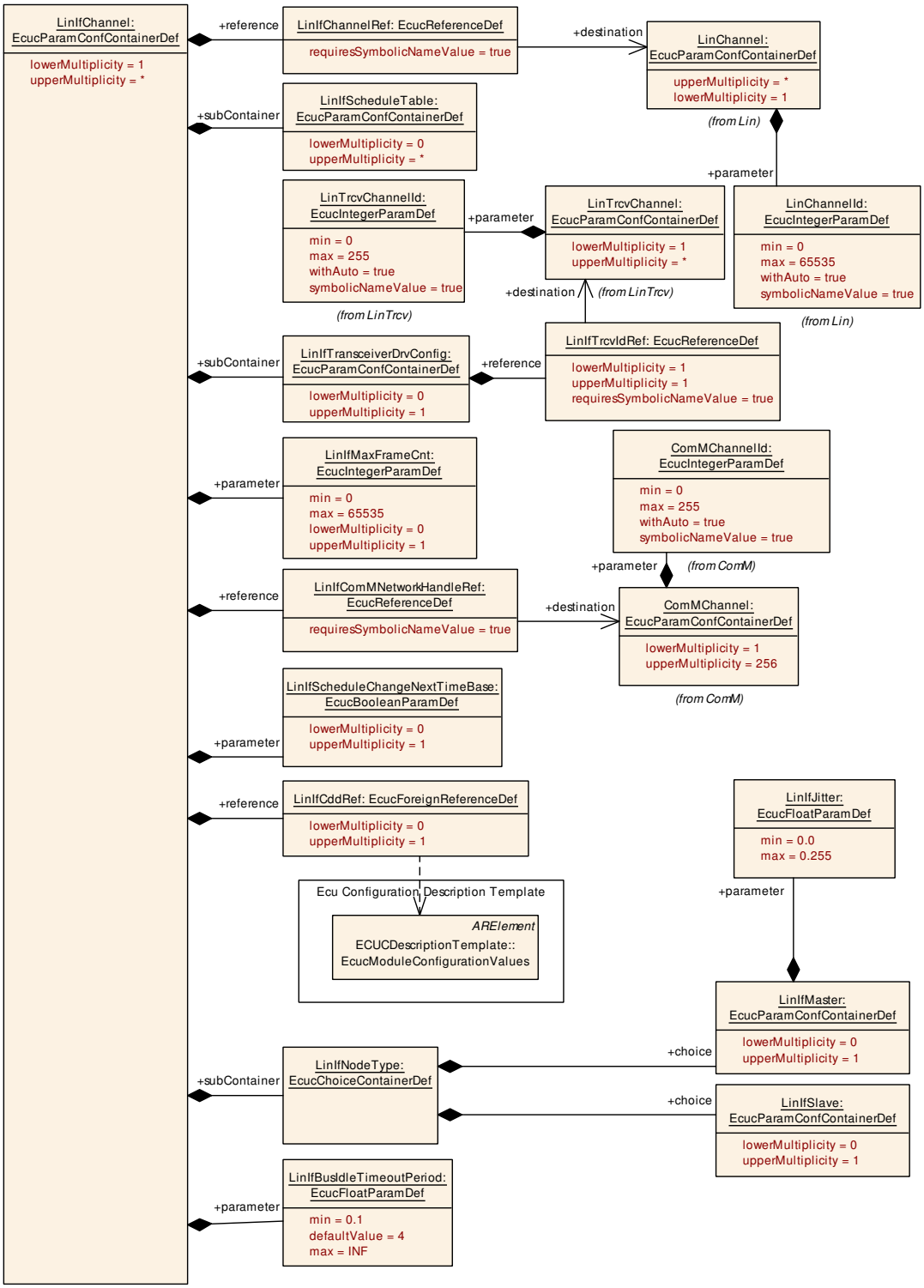


Figure 10.3: LIN Interface Channel configuration

[ECUC_LinIf_00364] Definition of EcucParamConfContainerDef LinIfChannel ⌈

Container Name	LinIfChannel
Parent Container	LinIfGlobalConfig
Upstream mapping defined	[22, CP_TPS_SystemTemplate]
Description	Describes each LIN channel the LinIf is connected to.
Multiplicity	1..*
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
LinIfBusIdleTimeoutPeriod	1	[ECUC_LinIf_00655]
LinIfGotoSleepConfirmationUL	1	[ECUC_LinIf_00601]
LinIfGotoSleepIndicationUL	0..1	[ECUC_LinIf_00652]
LinIfMainFunctionPeriod	1	[ECUC_LinIf_00639]
LinIfMaxFrameCnt	0..1	[ECUC_LinIf_00636]
LinIfScheduleChangeNextTimeBase	0..1	[ECUC_LinIf_00640]
LinIfScheduleRequestConfirmationUL	0..1	[ECUC_LinIf_00600]
LinIfWakeupConfirmationUL	1	[ECUC_LinIf_00602]
LinIfCddRef	0..1	[ECUC_LinIf_00637]
LinIfChannelRef	1	[ECUC_LinIf_00003]
LinIfComMNetworkHandleRef	1	[ECUC_LinIf_00626]

Included Containers		
Container Name	**Multiplicity**	**Description**
LinIfFrame	0..*	Generic container for all types of LIN frames.
LinIfNodeType	1	This container defines the LIN node type of this channel.
LinIfScheduleTable	0..*	Describes a schedule table. Each LinIfChannel may have several schedule tables. Each schedule table can only be connected to one channel. Mandatory for LIN Master nodes. 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.

⌋

[ECUC_LinIf_00655] Definition of EcucFloatParamDef LinIfBusIdleTimeoutPeriod ⌈

Parameter Name	LinIfBusIdleTimeoutPeriod	
Parent Container	LinIfChannel	
Upstream mapping defined	[22, CP_TPS_SystemTemplate]	
Description	Bus idle timeout in seconds. According to the LIN protocol specification, the bus idle timeout period shall be in range [4, 10] seconds.	
Multiplicity	1	
Type	EcucFloatParamDef	
Range	[0.1 .. INF]	

▽

△

Default value	4	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_LinIf_00601] Definition of EcucEnumerationParamDef LinIfGotoSleepConfirmationUL ⌈

Parameter Name	LinIfGotoSleepConfirmationUL	
Parent Container	LinIfChannel	
Upstream mapping defined	–	
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 Value	-	Selectable

⌋

[ECUC_LinIf_00652] Definition of EcucEnumerationParamDef LinIfGotoSleepIndicationUL ⌈

Parameter Name	LinIfGotoSleepIndicationUL	
Parent Container	LinIfChannel	
Upstream mapping defined	–	
Description	This parameter defines the upper layer (UL) module to which the indication of the goto-sleep command shall be sent. Only used for LIN Slave nodes, ignored for master nodes. Dependency: Only applicable when the channel contains a LinIfSlave container.	
Multiplicity	0..1	
Type	EcucEnumerationParamDef	
Range	CDD	Complex Driver
	LIN_SM	LIN State Manager
Post-Build Value	-	Selectable

⌋

[ECUC_LinIf_00639] Definition of EcucFloatParamDef LinIfMainFunctionPeriod ⌈

Parameter Name	LinIfMainFunctionPeriod	
Parent Container	LinIfChannel	
Upstream mapping defined	[22, CP_TPS_SystemTemplate]	
Description	Defines the interval of calls to main functions per channel in seconds.	
Multiplicity	1	
Type	EcucFloatParamDef	
Range	]0 .. INF[	
Default value	–	

▽

△

Pre-Compile only	true

⌋

[ECUC_LinIf_00636] Definition of EcucIntegerParamDef LinIfMaxFrameCnt ⌈

Parameter Name	LinIfMaxFrameCnt	
Parent Container	LinIfChannel	
Upstream mapping defined	–	
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	–	
Pre-Compile only	true	

⌋

[ECUC_LinIf_00640] Definition of EcucBooleanParamDef LinIfScheduleChange NextTimeBase ⌈

Parameter Name	LinIfScheduleChangeNextTimeBase	
Parent Container	LinIfChannel	
Upstream mapping defined	[22, CP_TPS_SystemTemplate]	
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. Only applicable for LIN Master nodes. Dependency: Only applicable when the channel contains a LinIfMaster container.	
Multiplicity	0..1	
Type	EcucBooleanParamDef	
Default value	–	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_LinIf_00600] Definition of EcucEnumerationParamDef LinIfScheduleRequestConfirmationUL ⌈

Parameter Name	LinIfScheduleRequestConfirmationUL	
Parent Container	LinIfChannel	
Upstream mapping defined	–	
Description	This parameter defines the upper layer (UL) module to which the confirmation of the successfully performed schedule table change shall be sent. Only applicable to LIN master nodes. Dependency: Only applicable when the channel contains a LinIfMaster container.	
Multiplicity	0..1	
Type	EcucEnumerationParamDef	
Range	CDD	Complex Driver
	LIN_SM	LIN State Manager

▽

△

Post-Build Value	-	Selectable

⌋

[ECUC_LinIf_00602] Definition of EcucEnumerationParamDef LinIfWakeupConfirmationUL ⌈

Parameter Name	LinIfWakeupConfirmationUL	
Parent Container	LinIfChannel	
Upstream mapping defined	–	
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 Value	-	Selectable

⌋

[ECUC_LinIf_00637] Definition of EcucForeignReferenceDef LinIfCddRef ⌈

Parameter Name	LinIfCddRef
Parent Container	LinIfChannel
Upstream mapping defined	[23, CP_TPS_ECUConfiguration]
Description	Reference to the CDD module description. This parameter is only required when LinIf WakeupConfirmationUL, LinIfScheduleRequestConfirmationUL, LinIfGotoSleep ConfirmationUL and/or LinIfGotoSleepIndicationUL is set to CDD.
Multiplicity	0..1
Type	Foreign reference to ECUC-MODULE-CONFIGURATION-VALUES
Pre-Compile only	true

⌋

[ECUC_LinIf_00003] Definition of EcucReferenceDef LinIfChannelRef ⌈

Parameter Name	LinIfChannelRef
Parent Container	LinIfChannel
Upstream mapping defined	–
Description	Reference to the channel definition in the LIN driver.
Multiplicity	1
Type	Symbolic name reference to LinChannel
Pre-Compile only	true

⌋

[ECUC_LinIf_00626] Definition of EcucReferenceDef LinIfComMNetworkHandleRef ⌈

Parameter Name	LinIfComMNetworkHandleRef
Parent Container	LinIfChannel
Upstream mapping defined	–
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
Pre-Compile only	true

⌋

10.3.5 LinIfNodeType

[ECUC_LinIf_00654] Definition of EcucChoiceContainerDef LinIfNodeType ⌈

Choice Container Name	LinIfNodeType
Parent Container	LinIfChannel
Upstream mapping defined	[22, CP_TPS_SystemTemplate]
Description	This container defines the LIN node type of this channel.
Multiplicity	1
Pre-Compile only	true

No Included Parameters

Container Choices		
Container Name	**Multiplicity**	**Description**
LinIfMaster	0..1	Each Master can only be connected to one physical channel. This could be compared to the Node parameter in a LDF file.
LinIfSlave	0..1	Describes all parameters which are only relevant for a LIN Slave node.

⌋

10.3.6 LinIfFrame

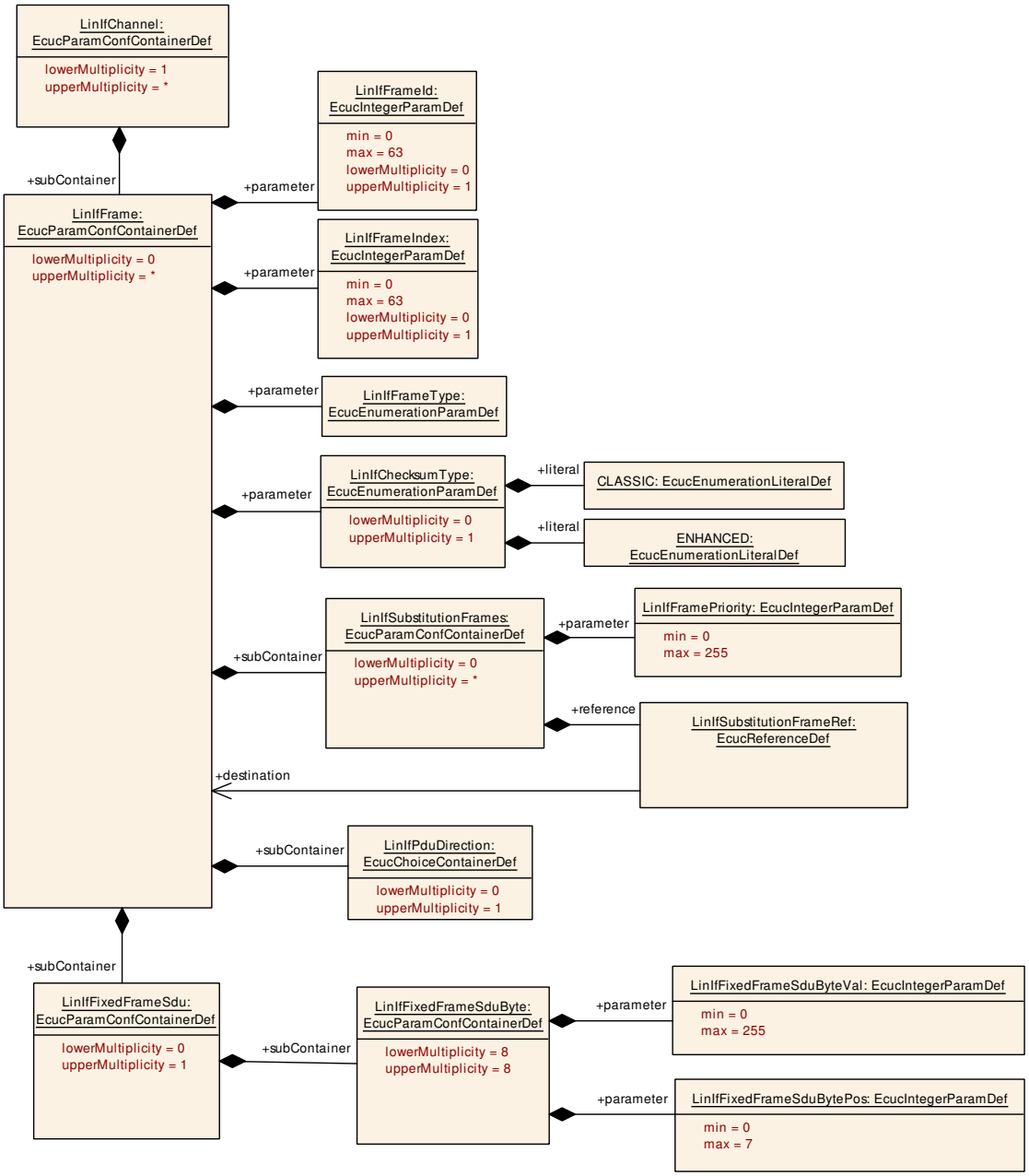


Figure 10.4: LIN Interface Frame configuration – (1) Overview

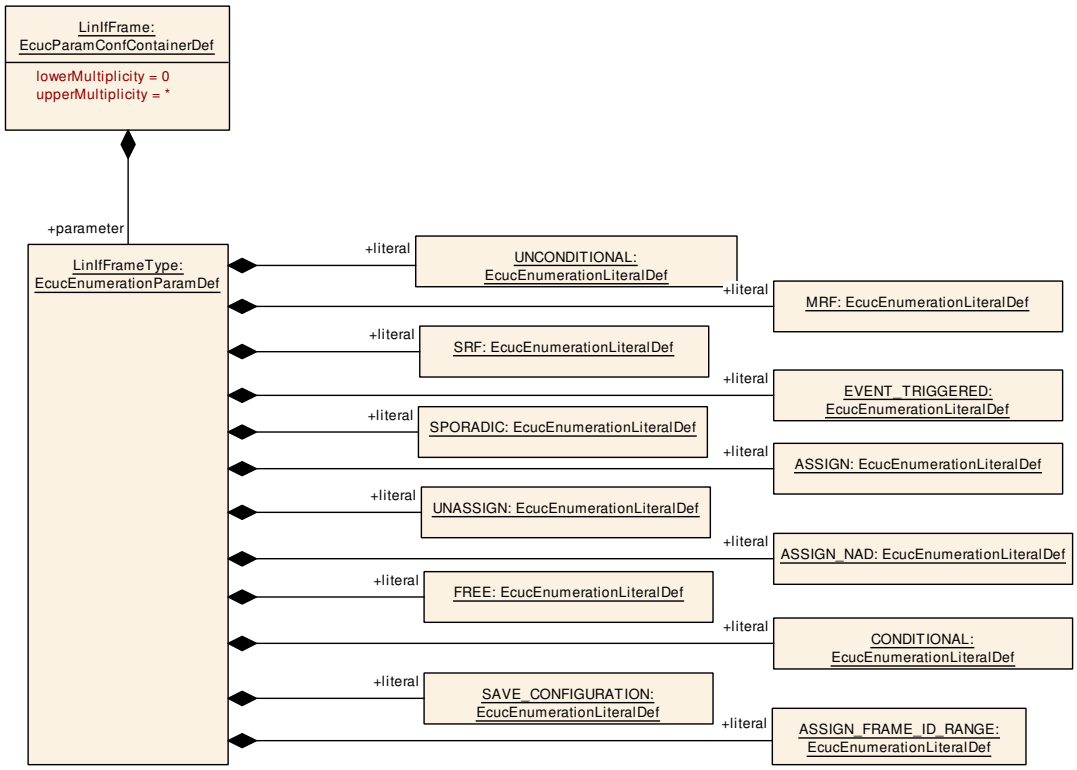


Figure 10.5: LIN Interface Frame configuration – (2) LinIfFrameType

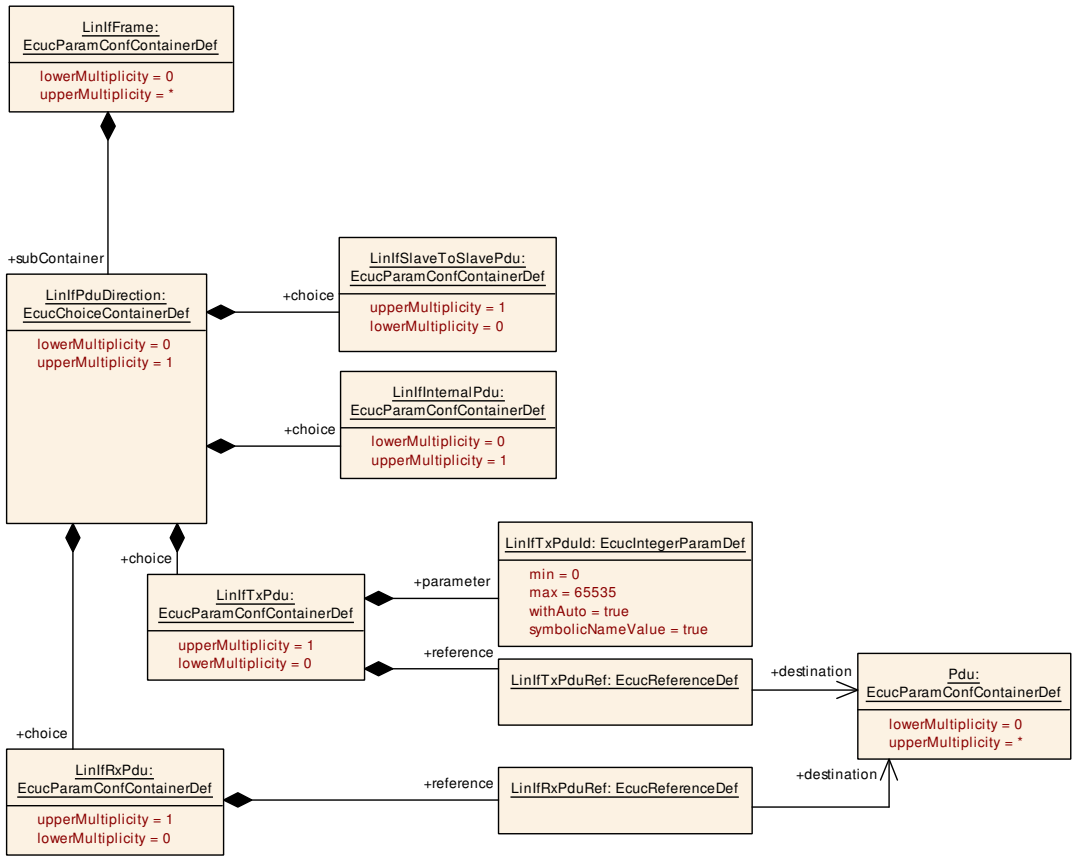


Figure 10.6: LIN Interface Frame configuration – (3) LinIfPduDirection

[ECUC_LinIf_00367] Definition of EcucParamConfContainerDef LinIfFrame ⌈

Container Name	LinIfFrame	
Parent Container	LinIfChannel	
Upstream mapping defined	[22, CP_TPS_SystemTemplate]	
Description	Generic container for all types of LIN frames.	
Multiplicity	0..*	
Post-Build Multiplicity	Loadable	Selectable
Configuration Parameters		

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
LinIfChecksumType	0..1	[ECUC_LinIf_00005]
LinIfFrameId	0..1	[ECUC_LinIf_00638]
LinIfFrameIndex	0..1	[ECUC_LinIf_00653]
LinIfFrameType	1	[ECUC_LinIf_00017]

Included Containers		
Container Name	**Multiplicity**	**Description**
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. Only applicable to LIN master nodes.
LinIfPduDirection	0..1	Direction of the frame
LinIfSubstitutionFrames	0..*	List of sporadic frames that can be sent in a sporadic frame slot (master node) or list of unconditional frames that can be sent in an event-triggered frame slot (slave node).

⌋

[ECUC_LinIf_00005] Definition of EcucEnumerationParamDef LinIfChecksumType ⌈

Parameter Name	LinIfChecksumType	
Parent Container	LinIfFrame	
Upstream mapping defined	[22, CP_TPS_SystemTemplate]	
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 Value	Loadable	Selectable

⌋

[ECUC_LinIf_00638] Definition of EcucIntegerParamDef LinIfFrameId ⌈

Parameter Name	LinIfFrameId
Parent Container	LinIfFrame
Upstream mapping defined	[22, CP_TPS_SystemTemplate]

▽

△

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 Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

[ECUC_LinIf_00653] Definition of EcucIntegerParamDef LinIfFrameIndex ⌈

Parameter Name	LinIfFrameIndex	
Parent Container	LinIfFrame	
Upstream mapping defined	[22, CP_TPS_SystemTemplate]	
Description	PID index of the frame. This index is used in the AssignFrameIdentifierRange node configuration service to identify the frame(s) to which a new PID shall be assigned. It corresponds to the order of the frames in the configurable frames list in the node attributes section of the LDF / NCF of the slave node. Only relevant for LIN slave nodes. Dependency: Only applicable when the channel contains a LinIfSlave container.	
Multiplicity	0..1	
Type	EcucIntegerParamDef	
Range	0 .. 63	
Default value	–	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

[ECUC_LinIf_00017] Definition of EcucEnumerationParamDef LinIfFrameType ⌈

Parameter Name	LinIfFrameType	
Parent Container	LinIfFrame	
Upstream mapping defined	[22, CP_TPS_SystemTemplate]	
Description	This parameter defines the type of frame (e.g. sporadic frame). For master nodes, all frame types are permitted. A sporadic slot may be used by a set of unconditional frames in the role of substitution frames. For slave nodes, only following types are permitted: Uncondtional, MRF, SRF, Event-triggered. An event-triggered 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 Value	Loadable	Selectable

⌋

10.3.7 LinIfFixedFrameSdu

[ECUC_LinIf_00012] Definition of EcucParamConfContainerDef LinIfFixedFrameSdu ⌈

Container Name	LinIfFixedFrameSdu
Parent Container	LinIfFrame
Upstream mapping defined	–
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. Only applicable to LIN master nodes.
Multiplicity	0..1
Pre-Compile only	true
Configuration Parameters	

No Included Parameters

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

⌋

10.3.8 LinIfFixedFrameSduByte

[ECUC_LinIf_00013] Definition of EcucParamConfContainerDef LinIfFixedFrameSduByte ⌈

Container Name	LinIfFixedFrameSduByte
Parent Container	LinIfFixedFrameSdu
Upstream mapping defined	[22, CP_TPS_SystemTemplate]
Description	This container represents a byte within the 8 byte array.
Multiplicity	8
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
LinIfFixedFrameSduBytePos	1	[ECUC_LinIf_00014]
LinIfFixedFrameSduByteVal	1	[ECUC_LinIf_00015]

No Included Containers

⌋

[ECUC_LinIf_00014] Definition of EcucIntegerParamDef LinIfFixedFrameSduBytePos ⌈

Parameter Name	LinIfFixedFrameSduBytePos	
Parent Container	LinIfFixedFrameSduByte	
Upstream mapping defined	–	
Description	Index of the Byte in the SDU (response) 8 byte array.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 7	
Default value	–	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_LinIf_00015] Definition of EcucIntegerParamDef LinIfFixedFrameSduByteVal ⌈

Parameter Name	LinIfFixedFrameSduByteVal	
Parent Container	LinIfFixedFrameSduByte	
Upstream mapping defined	–	
Description	Byte value in the SDU (response) 8-byte array.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 255	
Default value	–	
Post-Build Value	Loadable	Selectable

⌋

10.3.9 LinIfPduDirection

[ECUC_LinIf_00027] Definition of EcucChoiceContainerDef LinIfPduDirection ⌈

Choice Container Name	LinIfPduDirection
Parent Container	LinIfFrame
Upstream mapping defined	[22, CP_TPS_SystemTemplate]
Description	Direction of the frame

▽

△

Multiplicity	0..1
Pre-Compile only	true

No Included Parameters

Container Choices		
Container Name	**Multiplicity**	**Description**
LinIfInternalPdu	0..1	Represents a Diagnostic or Configuration frame : no Message ID (no PduId). Only applicable to LIN master nodes.
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. Only relevant for master nodes.
LinIfTxPdu	0..1	represents a transmitted PDU/frame

⌋

10.3.10 LinIfSubstitutionFrames

[ECUC_LinIf_00042] Definition of EcucParamConfContainerDef LinIfSubstitution Frames ⌈

Container Name	LinIfSubstitutionFrames
Parent Container	LinIfFrame
Upstream mapping defined	[22, CP_TPS_SystemTemplate]
Description	List of sporadic frames that can be sent in a sporadic frame slot (master node) or list of unconditional frames that can be sent in an event-triggered frame slot (slave node).
Multiplicity	0..*
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
LinIfFramePriority	1	[ECUC_LinIf_00513]
LinIfSubstitutionFrameRef	1	[ECUC_LinIf_00041]

No Included Containers

⌋

[ECUC_LinIf_00513] Definition of EcucIntegerParamDef LinIfFramePriority ⌈

Parameter Name	LinIfFramePriority
Parent Container	LinIfSubstitutionFrames
Upstream mapping defined	[22, CP_TPS_SystemTemplate]
Description	Priority of sporadic frame in a master node or of event-triggered frame in slave node.

▽

△

Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 255	
Default value	–	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_LinIf_00041] Definition of EcucReferenceDef LinIfSubstitutionFrameRef

⌈

Parameter Name	LinIfSubstitutionFrameRef	
Parent Container	LinIfSubstitutionFrames	
Upstream mapping defined	–	
Description	Reference to an unconditional Frame that is used as sporadic frame in a master node or event-triggered frame in a slave node.	
Multiplicity	1	
Type	Reference to LinIfFrame	
Post-Build Value	Loadable	Selectable

⌋

10.3.11 LinIfRxPdu

[ECUC_LinIf_00035] Definition of EcucParamConfContainerDef LinIfRxPdu ⌈

Container Name	LinIfRxPdu
Parent Container	LinIfPduDirection
Upstream mapping defined	[22, CP_TPS_SystemTemplate]
Description	represents a received PDU/frame
Multiplicity	0..1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
LinIfRxPduRef	1	[ECUC_LinIf_00036]

No Included Containers

⌋

[ECUC_LinIf_00036] Definition of EcucReferenceDef LinIfRxPduRef ⌈

Parameter Name	LinIfRxPduRef	
Parent Container	LinIfRxPdu	
Upstream mapping defined	–	
Description	Reference to the PDU that is received in this frame.	
Multiplicity	1	
Type	Reference to Pdu	
Post-Build Value	Loadable	Selectable

⌋

10.3.12 LinIfTxPdu

[ECUC_LinIf_00049] Definition of EcucParamConfContainerDef LinIfTxPdu ⌈

Container Name	LinIfTxPdu
Parent Container	LinIfPduDirection
Upstream mapping defined	[22, CP_TPS_SystemTemplate]
Description	represents a transmitted PDU/frame
Multiplicity	0..1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
LinIfTxPduId	1	[ECUC_LinIf_00050]
LinIfTxPduRef	1	[ECUC_LinIf_00051]

No Included Containers

⌋

[ECUC_LinIf_00050] Definition of EcucIntegerParamDef LinIfTxPduId ⌈

Parameter Name	LinIfTxPduId	
Parent Container	LinIfTxPdu	
Upstream mapping defined	–	
Description	Identifier of the Pdu for the upper layer.	
Multiplicity	1	
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)	
Range	0 .. 65535	
Default value	–	
With Auto	true	
Pre-Compile only	true	

⌋

[ECUC_LinIf_00051] Definition of EcucReferenceDef LinIfTxPduRef ⌈

Parameter Name	LinIfTxPduRef	
Parent Container	LinIfTxPdu	
Upstream mapping defined	–	
Description	Reference to the PDU that is transmitted in this frame.	
Multiplicity	1	
Type	Reference to Pdu	
Post-Build Value	Loadable	Selectable

⌋

10.3.13 LinIfScheduleTable

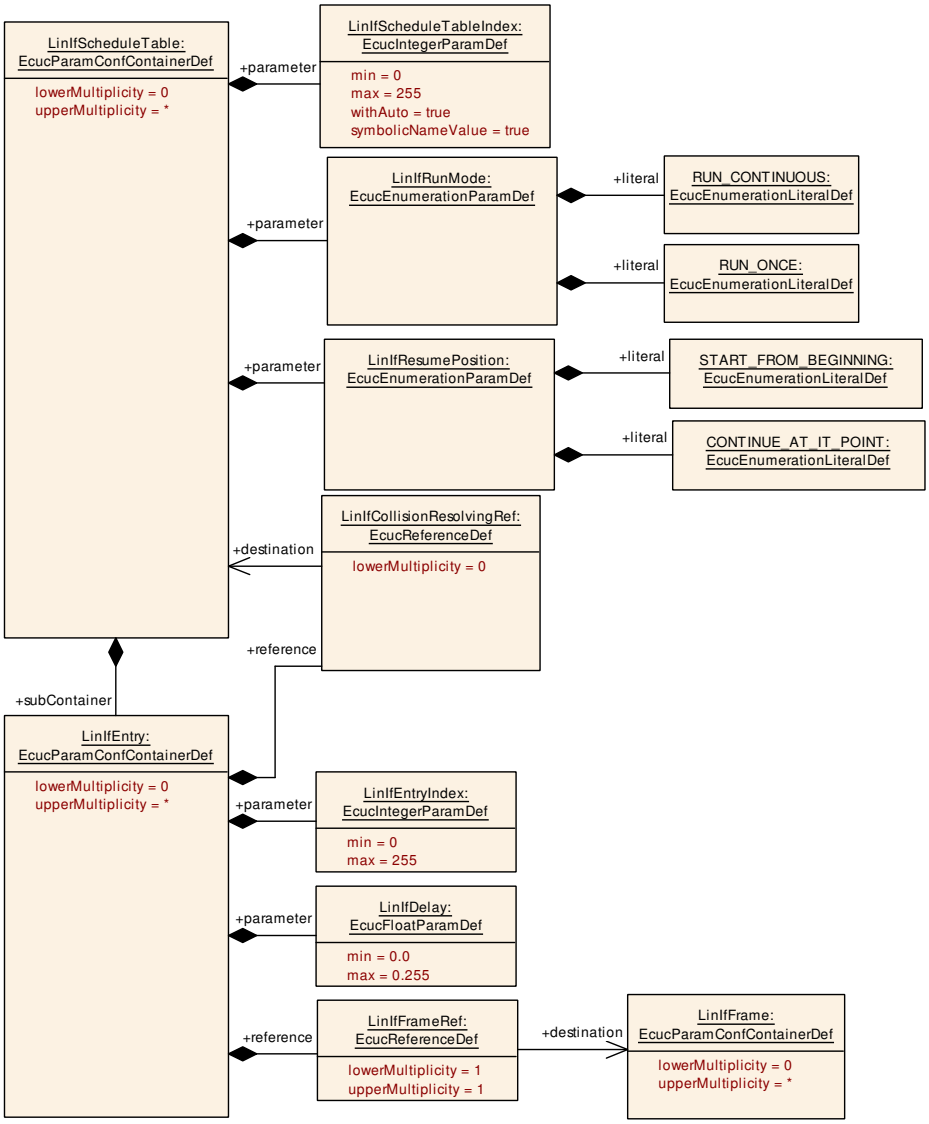


Figure 10.7: LIN Interface Schedule Table configuration

[ECUC_LinIf_00365] Definition of EcucParamConfContainerDef LinIfSchedule Table ⌈

Container Name	LinIfScheduleTable
Parent Container	LinIfChannel
Upstream mapping defined	[22, CP_TPS_SystemTemplate]

▽

△

Description	Describes a schedule table. Each LinIfChannel may have several schedule tables. Each schedule table can only be connected to one channel. Mandatory for LIN Master nodes. The SHORT-NAME of the LinIfScheduleTable container represents the symbolic name of the schedule table.	
Multiplicity	0..*	
Post-Build Multiplicity	Loadable	Selectable
Configuration Parameters		

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
LinIfResumePosition	1	[ECUC_LinIf_00033]
LinIfRunMode	1	[ECUC_LinIf_00034]
LinIfScheduleTableIndex	1	[ECUC_LinIf_00037]

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

⌋

[ECUC_LinIf_00033] Definition of EcucEnumerationParamDef LinIfResumePosition ⌈

Parameter Name	LinIfResumePosition	
Parent Container	LinIfScheduleTable	
Upstream mapping defined	[22, CP_TPS_SystemTemplate]	
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 Value	Loadable	Selectable

⌋

[ECUC_LinIf_00034] Definition of EcucEnumerationParamDef LinIfRunMode ⌈

Parameter Name	LinIfRunMode	
Parent Container	LinIfScheduleTable	
Upstream mapping defined	[22, CP_TPS_SystemTemplate]	
Description	The schedule table can be executed in two different modes.	
Multiplicity	1	
Type	EcucEnumerationParamDef	
Range	RUN_CONTINUOUS	Continuous
	RUN_ONCE	Once
Post-Build Value	Loadable	Selectable

⌋

[ECUC_LinIf_00037] Definition of EcucIntegerParamDef LinIfScheduleTableIndex ⌈

Parameter Name	LinIfScheduleTableIndex	
Parent Container	LinIfScheduleTable	
Upstream mapping defined	–	
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	–	
With Auto	true	
Pre-Compile only	true	

⌋

10.3.14 LinIfEntry

[ECUC_LinIf_00366] Definition of EcucParamConfContainerDef LinIfEntry ⌈

Container Name	LinIfEntry	
Parent Container	LinIfScheduleTable	
Upstream mapping defined	[22, CP_TPS_SystemTemplate]	
Description	Describes an entry in the schedule table (also known as Frame Slot).	
Multiplicity	0..*	
Post-Build Multiplicity	Loadable	Selectable
Configuration Parameters		

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
LinIfDelay	1	[ECUC_LinIf_00009]
LinIfEntryIndex	1	[ECUC_LinIf_00011]
LinIfCollisionResolvingRef	0..1	[ECUC_LinIf_00007]
LinIfFrameRef	1	[ECUC_LinIf_00016]

No Included Containers

⌋

[ECUC_LinIf_00009] Definition of EcucFloatParamDef LinIfDelay ⌈

Parameter Name	LinIfDelay
Parent Container	LinIfEntry
Upstream mapping defined	[22, CP_TPS_SystemTemplate]
Description	Delay to next entry in schedule table in seconds.
Multiplicity	1

▽

△

Type	EcucFloatParamDef	
Range	[0 .. 0.255]	
Default value	–	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_LinIf_00011] Definition of EcucIntegerParamDef LinIfEntryIndex ⌈

Parameter Name	LinIfEntryIndex	
Parent Container	LinIfEntry	
Upstream mapping defined	[22, CP_TPS_SystemTemplate]	
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 Value	Loadable	Selectable

⌋

[ECUC_LinIf_00007] Definition of EcucReferenceDef LinIfCollisionResolvingRef ⌈

Parameter Name	LinIfCollisionResolvingRef	
Parent Container	LinIfEntry	
Upstream mapping defined	[22, CP_TPS_SystemTemplate]	
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 Value	Loadable	Selectable

⌋

[ECUC_LinIf_00016] Definition of EcucReferenceDef LinIfFrameRef ⌈

Parameter Name	LinIfFrameRef	
Parent Container	LinIfEntry	
Upstream mapping defined	[22, CP_TPS_SystemTemplate]	
Description	Reference to the frames that belong to this schedule table entry.	
Multiplicity	1	
Type	Reference to LinIfFrame	
Post-Build Value	Loadable	Selectable

⌋

10.3.15 LinIfMaster

[ECUC_LinIf_00512] Definition of EcucParamConfContainerDef LinIfMaster ⌈

Container Name	LinIfMaster
Parent Container	LinIfNodeType
Upstream mapping defined	[22, CP_TPS_SystemTemplate]
Description	Each Master can only be connected to one physical channel. This could be compared to the Node parameter in a LDF file.
Multiplicity	0..1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
LinIfJitter	1	[ECUC_LinIf_00629]

No Included Containers

⌋

[ECUC_LinIf_00629] Definition of EcucFloatParamDef LinIfJitter ⌈

Parameter Name	LinIfJitter	
Parent Container	LinIfMaster	
Upstream mapping defined	[22, CP_TPS_SystemTemplate]	
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	–	
Pre-Compile only	true	

⌋

10.3.16 LinIfSlave

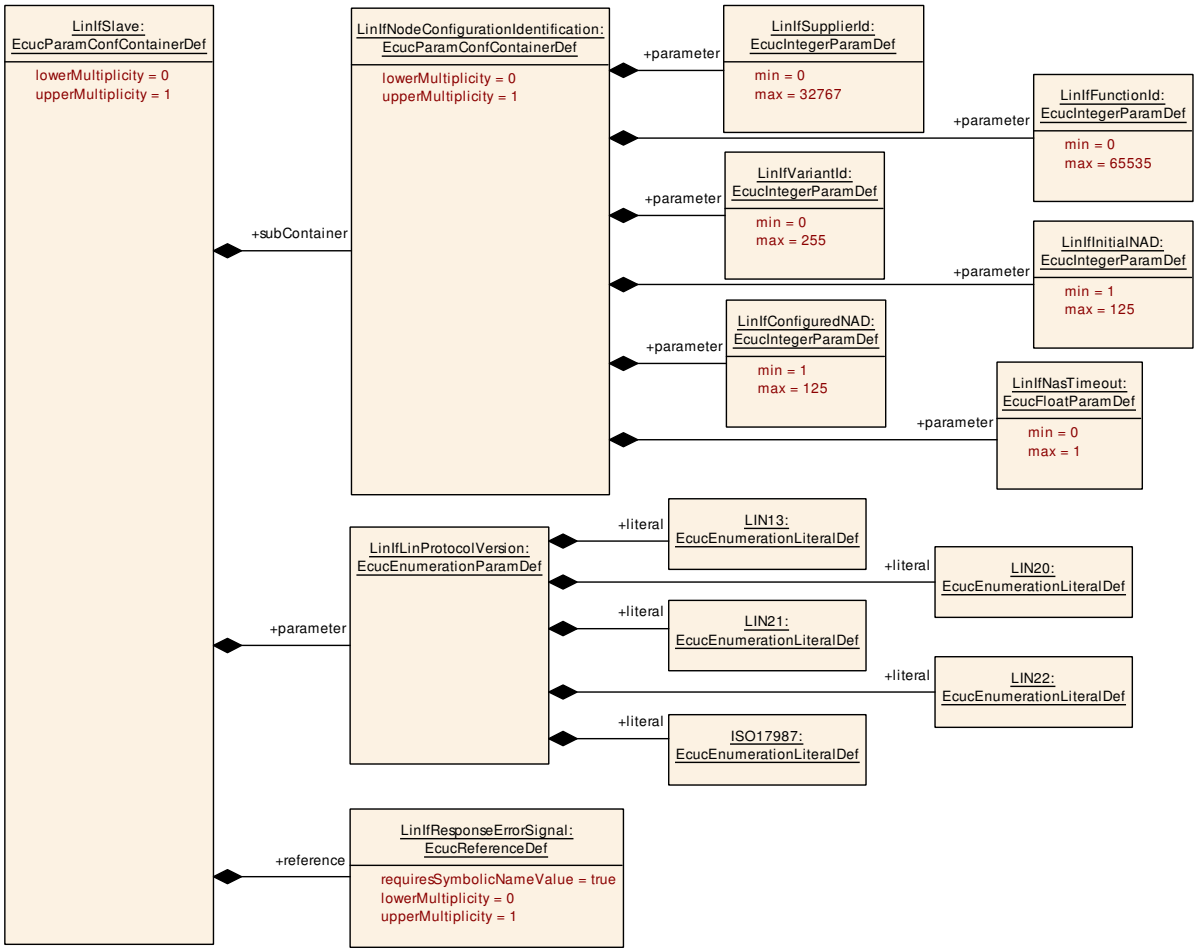


Figure 10.8: LIN Interface Slave configuration

[ECUC_LinIf_00649] Definition of EcucParamConfContainerDef LinIfSlave ⌈

Container Name	LinIfSlave
Parent Container	LinIfNodeType
Upstream mapping defined	[22, CP_TPS_SystemTemplate]
Description	Describes all parameters which are only relevant for a LIN Slave node.
Multiplicity	0..1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
LinIfLinProtocolVersion	1	[ECUC_LinIf_00647]
LinIfResponseErrorSignal	0..1	[ECUC_LinIf_00648]

Included Containers		
Container Name	**Multiplicity**	**Description**
LinIfNodeConfiguration Identification	0..1	This container is mandatory for all LIN 2.x and ISO17987 LIN slave nodes, and ignored for LIN 1.3 slave nodes and all master nodes,

⌋

[ECUC_LinIf_00647] Definition of EcucEnumerationParamDef LinIfLinProtocol Version ⌈

Parameter Name	LinIfLinProtocolVersion	
Parent Container	LinIfSlave	
Upstream mapping defined	[22, CP_TPS_SystemTemplate]	
Description	Defines the LIN protocol version of the slave node. This information is relevant for the LIN conformance test execution.	
Multiplicity	1	
Type	EcucEnumerationParamDef	
Range	ISO17987	ISO 17987 (first edition)
	LIN13	LIN 1.3
	LIN20	LIN 2.0
	LIN21	LIN 2.1
	LIN22	LIN 2.2
Pre-Compile only	true	

⌋

[ECUC_LinIf_00648] Definition of EcucReferenceDef LinIfResponseErrorSignal ⌈

Parameter Name	LinIfResponseErrorSignal
Parent Container	LinIfSlave
Upstream mapping defined	[22, CP_TPS_SystemTemplate]
Description	Reference to the response_error signal. Mandatory for all LIN 2.x and ISO LIN slave nodes, not relevant for LIN 1.3 slave nodes.
Multiplicity	0..1
Type	Symbolic name reference to ComSignal
Pre-Compile only	true

⌋

10.3.17 LinIfNodeConfigurationIdentification

[ECUC_LinIf_00650] Definition of EcucParamConfContainerDef LinIfNodeConfigurationIdentification ⌈

Container Name	LinIfNodeConfigurationIdentification
Parent Container	LinIfSlave
Upstream mapping defined	–

▽

△

Description	This container is mandatory for all LIN 2.x and ISO17987 LIN slave nodes, and ignored for LIN 1.3 slave nodes and all master nodes,
Multiplicity	0..1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
LinIfConfiguredNAD	1	[ECUC_LinIf_00643]
LinIfFunctionId	1	[ECUC_LinIf_00646]
LinIfInitialNAD	1	[ECUC_LinIf_00642]
LinIfNasTimeout	1	[ECUC_LinIf_00644]
LinIfSupplierId	1	[ECUC_LinIf_00645]
LinIfVariantId	1	[ECUC_LinIf_00641]

No Included Containers

⌋

[ECUC_LinIf_00643] Definition of EcucIntegerParamDef LinIfConfiguredNAD ⌈

Parameter Name	LinIfConfiguredNAD	
Parent Container	LinIfNodeConfigurationIdentification	
Upstream mapping defined	[22, CP_TPS_SystemTemplate]	
Description	Slave node configured NAD.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	1 .. 125	
Default value	–	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_LinIf_00646] Definition of EcucIntegerParamDef LinIfFunctionId ⌈

Parameter Name	LinIfFunctionId	
Parent Container	LinIfNodeConfigurationIdentification	
Upstream mapping defined	[22, CP_TPS_SystemTemplate]	
Description	LIN function Id.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 65535	
Default value	–	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_LinIf_00642] Definition of EcucIntegerParamDef LinIfInitialNAD ⌈

Parameter Name	LinIfInitialNAD	
Parent Container	LinIfNodeConfigurationIdentification	
Upstream mapping defined	[22, CP_TPS_SystemTemplate]	
Description	Slave node initial NAD.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	1 .. 125	
Default value	–	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_LinIf_00644] Definition of EcucFloatParamDef LinIfNasTimeout ⌈

Parameter Name	LinIfNasTimeout	
Parent Container	LinIfNodeConfigurationIdentification	
Upstream mapping defined	[22, CP_TPS_SystemTemplate]	
Description	N_As timeout in seconds.	
Multiplicity	1	
Type	EcucFloatParamDef	
Range	[0 .. 1]	
Default value	–	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_LinIf_00645] Definition of EcucIntegerParamDef LinIfSupplierId ⌈

Parameter Name	LinIfSupplierId	
Parent Container	LinIfNodeConfigurationIdentification	
Upstream mapping defined	[22, CP_TPS_SystemTemplate]	
Description	LIN consortium or ISO LIN supplier Id.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 32767	
Default value	–	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_LinIf_00641] Definition of EcucIntegerParamDef LinIfVariantId ⌈

Parameter Name	LinIfVariantId
Parent Container	LinIfNodeConfigurationIdentification
Upstream mapping defined	[22, CP_TPS_SystemTemplate]
Description	LIN variant Id.
Multiplicity	1
Type	EcucIntegerParamDef

▽

△

Range	0 .. 255	
Default value	–	
Post-Build Value	Loadable	Selectable

⌋

10.3.18 LinIfSlaveToSlavePdu

[ECUC_LinIf_00040] Definition of EcucParamConfContainerDef LinIfSlaveToSlavePdu ⌈

Container Name	LinIfSlaveToSlavePdu
Parent Container	LinIfPduDirection
Upstream mapping defined	–
Description	Represents a slave-to-slave PDU/frame. Master does only send the header but doesn't receive the response. Only relevant for master nodes.
Multiplicity	0..1
Pre-Compile only	true
Configuration Parameters	

No Included Parameters

No Included Containers

⌋

10.3.19 LinIfInternalPdu

[ECUC_LinIf_00021] Definition of EcucParamConfContainerDef LinIfInternalPdu ⌈

Container Name	LinIfInternalPdu
Parent Container	LinIfPduDirection
Upstream mapping defined	–
Description	Represents a Diagnostic or Configuration frame : no Message ID (no PduId). Only applicable to LIN master nodes.
Multiplicity	0..1
Pre-Compile only	true
Configuration Parameters	

No Included Parameters

No Included Containers

⌋

10.3.20 LinIfTransceiverDrvConfig

[ECUC_LinIf_00046] Definition of EcucParamConfContainerDef LinIfTransceiverDrvConfig ⌈

Container Name	LinIfTransceiverDrvConfig
Parent Container	LinIfChannel
Upstream mapping defined	–
Description	This container contains the configuration parameters of each underlying LIN Transceiver Driver.
Multiplicity	0..1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
LinIfTrcvIdRef	1	[ECUC_LinIf_00047]

No Included Containers

⌋

[ECUC_LinIf_00047] Definition of EcucReferenceDef LinIfTrcvIdRef ⌈

Parameter Name	LinIfTrcvIdRef
Parent Container	LinIfTransceiverDrvConfig
Upstream mapping defined	–
Description	Logical handle of the underlying LIN transceiver to be served by the LIN Interface.
Multiplicity	1
Type	Symbolic name reference to LinTrcvChannel
Pre-Compile only	true

⌋

10.4 LIN Transport Layer configuration

The Figure 10.9 shows the outline of the LIN Transport Protocol configuration.

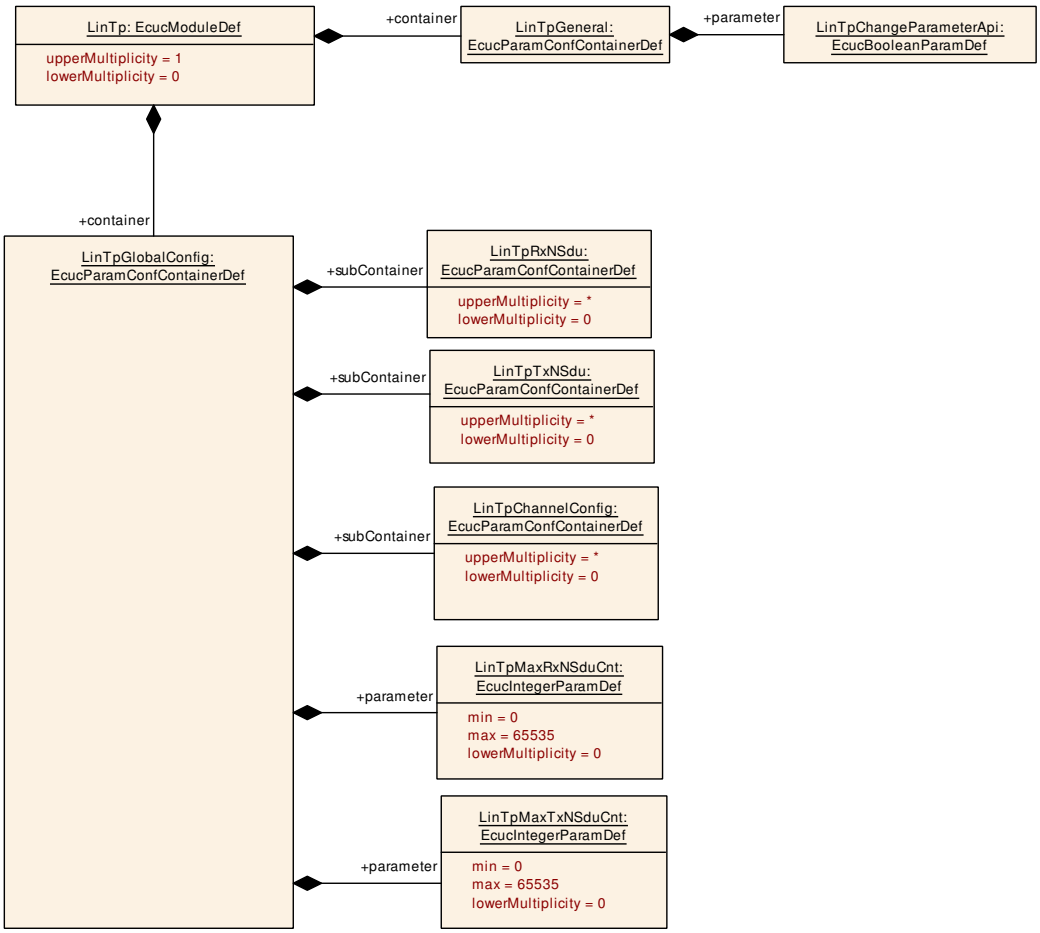


Figure 10.9: LIN Transport Protocol configuration – (1) Overview

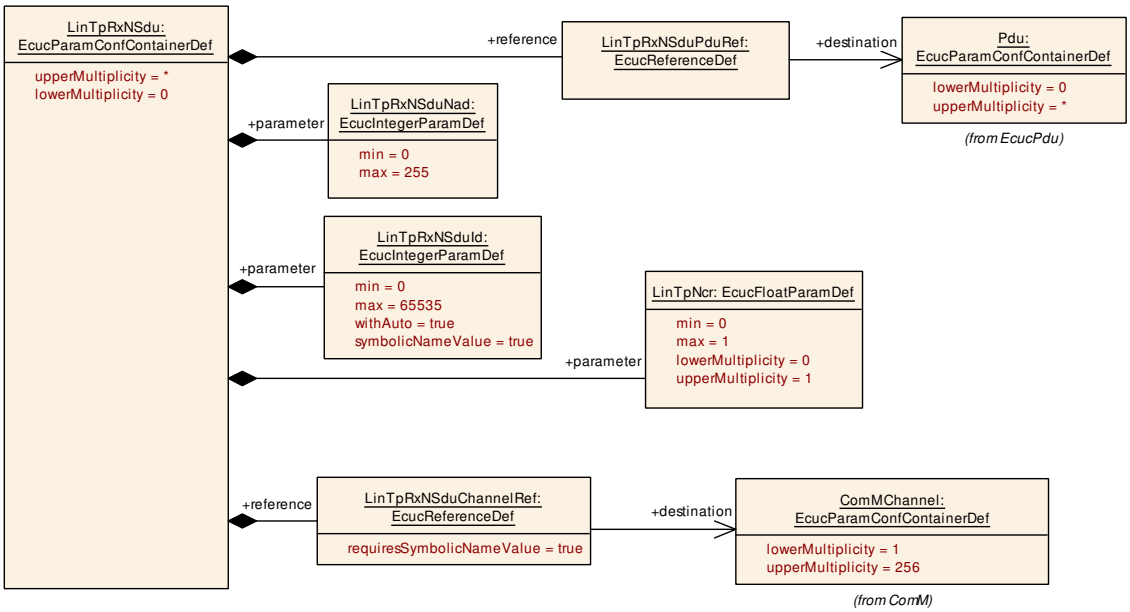


Figure 10.10: LIN Transport Protocol configuration – (2) LinTpRxNSdu

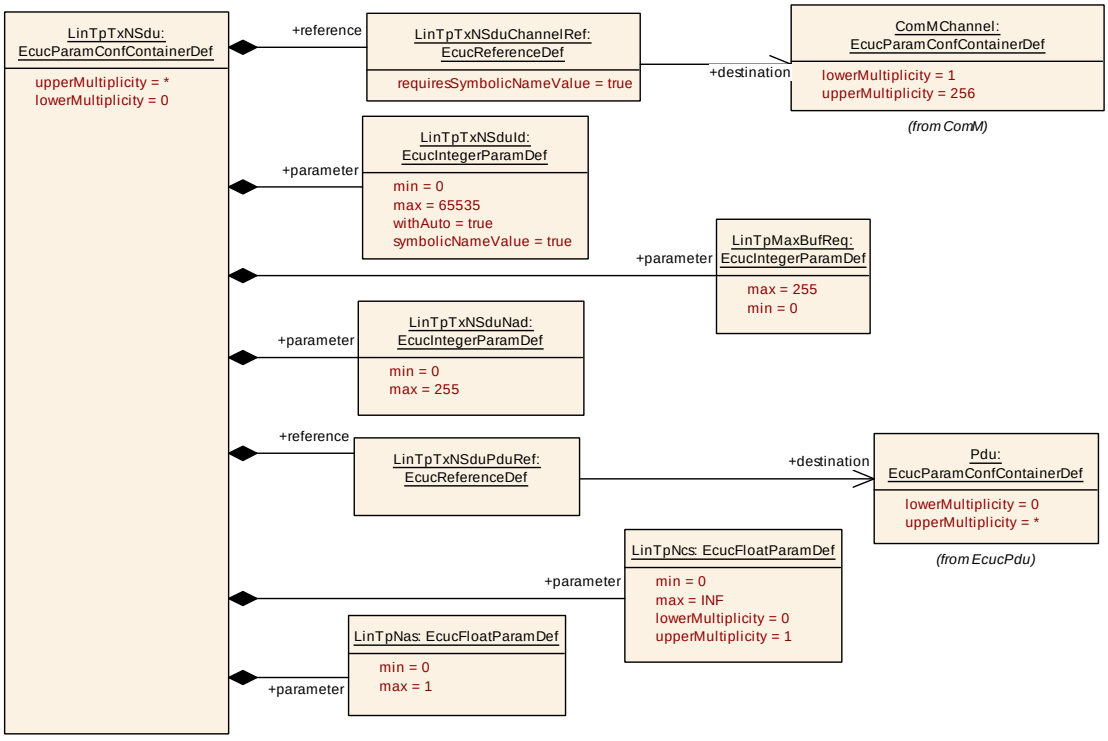


Figure 10.11: LIN Transport Protocol configuration – (3) LinTpTxNSdu

10.4.1 LinTp

[ECUC_LinTp_00425] Definition of EcucModuleDef LinTp ⌈

Module Name	LinTp
Description	Configuration of the LIN Transport Protocol.
Post-Build Selectable Support	true
Post-Build Loadable Support	true

Included Containers		
Container Name	**Multiplicity**	**Description**
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

[ECUC_LinTp_00617] Definition of EcucParamConfContainerDef LinTpGeneral ⌈

Container Name	LinTpGeneral
Parent Container	LinTp
Upstream mapping defined	–
Description	Container that holds all LIN transport protocol general parameters.
Multiplicity	1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
LinTpChangeParameterApi	1	[ECUC_LinTp_00638]

No Included Containers

⌋

[ECUC_LinTp_00638] Definition of EcucBooleanParamDef LinTpChangeParameterApi ⌈

Parameter Name	LinTpChangeParameterApi
Parent Container	LinTpGeneral
Upstream mapping defined	–
Description	This parameter, if set to true, enables the LinTp_ChangeParameter Api for this Module.
Multiplicity	1

▽

△

Type	EcucBooleanParamDef
Default value	–
Pre-Compile only	true

⌋

10.4.3 LinTpGlobalConfig

[ECUC_LinTp_00056] Definition of EcucParamConfContainerDef LinTpGlobalConfig ⌈

Container Name	LinTpGlobalConfig
Parent Container	LinTp
Upstream mapping defined	[22, CP_TPS_SystemTemplate]
Description	This container contains the global configuration parameters of the LinTp.
Multiplicity	1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
LinTpMaxRxNSduCnt	0..1	[ECUC_LinTp_00635]
LinTpMaxTxNSduCnt	0..1	[ECUC_LinTp_00636]

Included Containers		
Container Name	**Multiplicity**	**Description**
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.

⌋

[ECUC_LinTp_00635] Definition of EcucIntegerParamDef LinTpMaxRxNSduCnt ⌈

Parameter Name	LinTpMaxRxNSduCnt
Parent Container	LinTpGlobalConfig
Upstream mapping defined	–
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	–	
Pre-Compile only	true	

⌋

[ECUC_LinTp_00636] Definition of EcucIntegerParamDef LinTpMaxTxNSduCnt ⌈

Parameter Name	LinTpMaxTxNSduCnt	
Parent Container	LinTpGlobalConfig	
Upstream mapping defined	–	
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	–	
Pre-Compile only	true	

⌋

10.4.4 LinTpChannelConfig

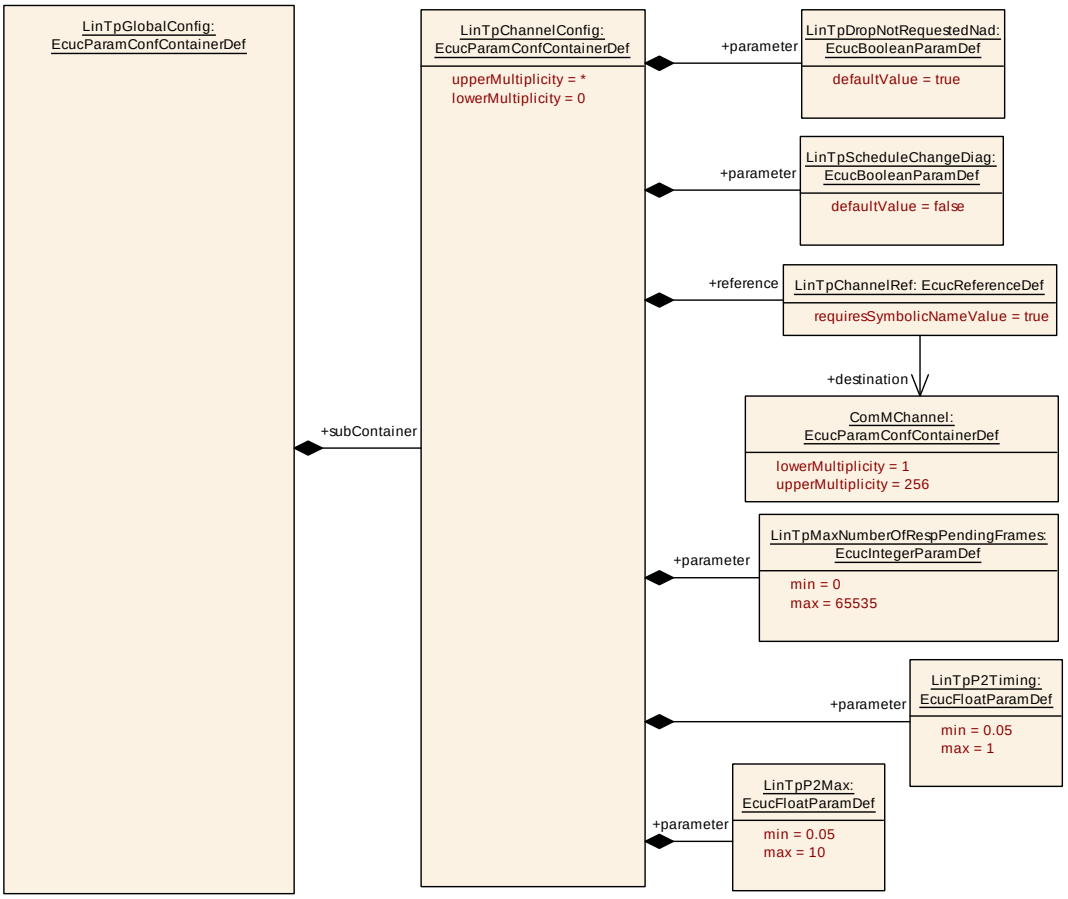


Figure 10.12: LIN Transport Protocol Channel configuration

[ECUC_LinTp_00071] Definition of EcucParamConfContainerDef LinTpChannel Config ⌈

Container Name	LinTpChannelConfig	
Parent Container	LinTpGlobalConfig	
Upstream mapping defined	[22, CP_TPS_SystemTemplate]	
Description	This container contains the channel specific configuration parameters of LinTp.	
Multiplicity	0..*	
Post-Build Multiplicity	Loadable	Selectable
Configuration Parameters		

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
LinTpDropNotRequestedNad	1	[ECUC_LinTp_00072]
LinTpMaxNumberOfRespPendingFrames	1	[ECUC_LinTp_00624]
LinTpP2Max	1	[ECUC_LinTp_00622]
LinTpP2Timing	1	[ECUC_LinTp_00625]
LinTpScheduleChangeDiag	1	[ECUC_LinTp_00070]
LinTpChannelRef	1	[ECUC_LinTp_00073]

No Included Containers

⌋

[ECUC_LinTp_00072] Definition of EcucBooleanParamDef LinTpDropNotRequestedNad ⌈

Parameter Name	LinTpDropNotRequestedNad	
Parent Container	LinTpChannelConfig	
Upstream mapping defined	[22, CP_TPS_SystemTemplate]	
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 Only used for LIN Master nodes, ignored for slave nodes.	
Multiplicity	1	
Type	EcucBooleanParamDef	
Default value	true	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_LinTp_00624] Definition of EcucIntegerParamDef LinTpMaxNumberOfRespPendingFrames ⌈

Parameter Name	LinTpMaxNumberOfRespPendingFrames	
Parent Container	LinTpChannelConfig	
Upstream mapping defined	[22, CP_TPS_SystemTemplate]	
Description	Configures the maximum number of allowed response pending frames. Only used for LIN Master nodes, ignored for slave nodes.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 65535	
Default value	–	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_LinTp_00622] Definition of EcucFloatParamDef LinTpP2Max ⌈

Parameter Name	LinTpP2Max	
Parent Container	LinTpChannelConfig	
Upstream mapping defined	[22, CP_TPS_SystemTemplate]	
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 LinTpP2 Timing. Only used for LIN Master nodes, ignored for slave nodes.	
Multiplicity	1	
Type	EcucFloatParamDef	
Range	[0.05 .. 10]	
Default value	–	

▽

△

Post-Build Value	Loadable	Selectable

⌋

[ECUC_LinTp_00625] Definition of EcucFloatParamDef LinTpP2Timing ⌈

Parameter Name	LinTpP2Timing	
Parent Container	LinTpChannelConfig	
Upstream mapping defined	[22, CP_TPS_SystemTemplate]	
Description	Definition of the P2max timeout observation parameter in seconds. Only used for LIN Master nodes, ignored for slave nodes.	
Multiplicity	1	
Type	EcucFloatParamDef	
Range	[0.05 .. 1]	
Default value	–	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_LinTp_00070] Definition of EcucBooleanParamDef LinTpScheduleChangeDiag ⌈

Parameter Name	LinTpScheduleChangeDiag	
Parent Container	LinTpChannelConfig	
Upstream mapping defined	–	
Description	Enables or disables the call of BswM_LinTp_RequestMode() to diagnostic request/ response schedule. false: BswM is not called true: BswM is called Only used for LIN Master nodes, ignored for slave nodes.	
Multiplicity	1	
Type	EcucBooleanParamDef	
Default value	false	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_LinTp_00073] Definition of EcucReferenceDef LinTpChannelRef ⌈

Parameter Name	LinTpChannelRef	
Parent Container	LinTpChannelConfig	
Upstream mapping defined	–	
Description	Index of the channel this LinTp channel belongs to.	
Multiplicity	1	
Type	Symbolic name reference to ComMChannel	
Post-Build Value	Loadable	Selectable

⌋

10.4.5 LinTpRxNSdu

[ECUC_LinTp_00428] Definition of EcucParamConfContainerDef LinTpRxNSdu ⌈

Container Name	LinTpRxNSdu	
Parent Container	LinTpGlobalConfig	
Upstream mapping defined	[22, CP_TPS_SystemTemplate]	
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.	
Multiplicity	0..*	
Post-Build Multiplicity	Loadable	Selectable
Configuration Parameters		

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
LinTpNcr	0..1	[ECUC_LinTp_00632]
LinTpRxNSduId	1	[ECUC_LinTp_00061]
LinTpRxNSduNad	1	[ECUC_LinTp_00062]
LinTpRxNSduChannelRef	1	[ECUC_LinTp_00060]
LinTpRxNSduPduRef	1	[ECUC_LinTp_00063]

No Included Containers

⌋

[ECUC_LinTp_00632] Definition of EcucFloatParamDef LinTpNcr ⌈

Parameter Name	LinTpNcr	
Parent Container	LinTpRxNSdu	
Upstream mapping defined	[22, CP_TPS_SystemTemplate]	
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 Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

[ECUC_LinTp_00061] Definition of EcucIntegerParamDef LinTpRxNSduId ⌈

Parameter Name	LinTpRxNSduId
Parent Container	LinTpRxNSdu
Upstream mapping defined	–
Description	The identifier of the Transport Protocol message. This ID will be used by upper layers to call LinTp_ChangeParameter.
Multiplicity	1

▽

△

Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)	
Range	0 .. 65535	
Default value	–	
With Auto	true	
Pre-Compile only	true	

⌋

[ECUC_LinTp_00062] Definition of EcucIntegerParamDef LinTpRxNSduNad ⌈

Parameter Name	LinTpRxNSduNad	
Parent Container	LinTpRxNSdu	
Upstream mapping defined	[22, CP_TPS_SystemTemplate]	
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 Value	Loadable	Selectable

⌋

[ECUC_LinTp_00060] Definition of EcucReferenceDef LinTpRxNSduChannelRef ⌈

Parameter Name	LinTpRxNSduChannelRef
Parent Container	LinTpRxNSdu
Upstream mapping defined	–
Description	Index of the channel this N-SDU belongs to.
Multiplicity	1
Type	Symbolic name reference to ComMChannel
Pre-Compile only	true

⌋

[ECUC_LinTp_00063] Definition of EcucReferenceDef LinTpRxNSduPduRef ⌈

Parameter Name	LinTpRxNSduPduRef	
Parent Container	LinTpRxNSdu	
Upstream mapping defined	–	
Description	Reference to the global PDU	
Multiplicity	1	
Type	Reference to Pdu	
Post-Build Value	Loadable	Selectable

⌋

10.4.6 LinTpTxNSdu

[ECUC_LinTp_00511] Definition of EcucParamConfContainerDef LinTpTxNSdu ⌈

Container Name	LinTpTxNSdu	
Parent Container	LinTpGlobalConfig	
Upstream mapping defined	[22, CP_TPS_SystemTemplate]	
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.	
Multiplicity	0..*	
Post-Build Multiplicity	Loadable	Selectable
Configuration Parameters		

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
LinTpMaxBufReq	1	[ECUC_LinTp_00637]
LinTpNas	1	[ECUC_LinTp_00633]
LinTpNcs	0..1	[ECUC_LinTp_00634]
LinTpTxNSduId	1	[ECUC_LinTp_00065]
LinTpTxNSduNad	1	[ECUC_LinTp_00066]
LinTpTxNSduChannelRef	1	[ECUC_LinTp_00064]
LinTpTxNSduPduRef	1	[ECUC_LinTp_00067]

No Included Containers

⌋

[ECUC_LinTp_00637] Definition of EcucIntegerParamDef LinTpMaxBufReq ⌈

Parameter Name	LinTpMaxBufReq	
Parent Container	LinTpTxNSdu	
Upstream mapping defined	–	
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_LinTpCopy TxData when no timer is active.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 255	
Default value	–	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_LinTp_00633] Definition of EcucFloatParamDef LinTpNas ⌈

Parameter Name	LinTpNas	
Parent Container	LinTpTxNSdu	
Upstream mapping defined	[22, CP_TPS_SystemTemplate]	
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 Value	Loadable	Selectable

⌋

[ECUC_LinTp_00634] Definition of EcucFloatParamDef LinTpNcs ⌈

Parameter Name	LinTpNcs	
Parent Container	LinTpTxNSdu	
Upstream mapping defined	[22, CP_TPS_SystemTemplate]	
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 Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

[ECUC_LinTp_00065] Definition of EcucIntegerParamDef LinTpTxNSduId ⌈

Parameter Name	LinTpTxNSduId	
Parent Container	LinTpTxNSdu	
Upstream mapping defined	–	
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	–	
With Auto	true	
Pre-Compile only	true	

⌋

[ECUC_LinTp_00066] Definition of EcucIntegerParamDef LinTpTxNSduNad ⌈

Parameter Name	LinTpTxNSduNad
Parent Container	LinTpTxNSdu
Upstream mapping defined	[22, CP_TPS_SystemTemplate]
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 Value	Loadable	Selectable

⌋

[ECUC_LinTp_00064] Definition of EcucReferenceDef LinTpTxNSduChannelRef
⌈

Parameter Name	LinTpTxNSduChannelRef
Parent Container	LinTpTxNSdu
Upstream mapping defined	–
Description	Index of the channel this N-SDU belongs to.
Multiplicity	1
Type	Symbolic name reference to ComMChannel
Pre-Compile only	true

⌋

[ECUC_LinTp_00067] Definition of EcucReferenceDef LinTpTxNSduPduRef ⌈

Parameter Name	LinTpTxNSduPduRef	
Parent Container	LinTpTxNSdu	
Upstream mapping defined	–	
Description	Reference to the global PDU	
Multiplicity	1	
Type	Reference to Pdu	
Post-Build Value	Loadable	Selectable

⌋

10.5 Configuration Constraints

[SWS_LinIf_CONSTR_00903] `LinIfTxPdu` and `LinIfRxPdu` constraint for keeping the local buffer ⌈Each `LinIfTxPdu` and `LinIfRxPdu` shall refer to a global PDU that has `KeepLocalPduBuffer` set to FALSE.⌋

10.6 Published Information

For details refer to [8] Chapter 10.3 *"Published Information"*.

A Not applicable requirements

[SWS_LinIf_NA_99999]

Upstream requirements: SRS_BSW_00432, SRS_BSW_00433, SRS_BSW_00417, SRS_BSW_00437, SRS_BSW_00422

⌈These requirements are not applicable to this specification.⌋

B Change history of AUTOSAR traceable items

Please note that the lists in this chapter also include traceable items that have been removed from the specification in a later version. These items do not appear as hyperlinks in the document.

B.1 Traceable item history of this document according to AUTOSAR Release R24-11

B.1.1 Added Specification Items in R24-11

Number	Heading
[SWS_LinIf_00887]	LIN Interface state-machine and channel sub-state-machine for LIN Master Nodes
[SWS_LinIf_00888]	LIN Interface state-machine and channel sub-state-machine for LIN Slave Nodes
[SWS_LinIf_00889]	Include the header file of LSduR for LinIf
[SWS_LinIf_00890]	Limitation on invocation of `LSduR_LinIfRxIndication` in master nodes
[SWS_LinIf_00891]	Invocation of `LSduR_LinIfRxIndication` for Response in slave nodes
[SWS_LinIf_00892]	Invocation of `LSduR_LinIfTriggerTransmit` in master nodes
[SWS_LinIf_00893]	Invocation of `Lin_SendFrame` after successful invocation of `LSduR_LinIfTriggerTransmit` in master nodes
[SWS_LinIf_00894]	Behavior after failed invocation of `LSduR_LinIfTriggerTransmit` in master nodes
[SWS_LinIf_00895]	Limitation on invocation of `LSduR_LinIfTxConfirmation` in master nodes
[SWS_LinIf_00896]	Invocation of `LSduR_LinIfTxConfirmation` at error or busy conditions in master nodes
[SWS_LinIf_00897]	Invocation of `LSduR_LinIfTriggerTransmit` after Header indication in slave nodes
[SWS_LinIf_00898]	Handling of Cs, DI and Drc in slave nodes
[SWS_LinIf_00899]	Handling of Drc for failed invocation of `LSduR_LinIfTriggerTransmit` in slave nodes
[SWS_LinIf_00900]	Invocation of `LSduR_LinIfTxConfirmation` after `LinIf_TxConfirmation` in slave nodes
[SWS_LinIf_00901]	Invocation of `LSduR_LinIfTxConfirmation` after `LinIf_LinErrorIndication` in slave nodes

Table B.1: Added Specification Items in R24-11

B.1.2 Changed Specification Items in R24-11

Number	Heading
[ECUC_LinIf_00035]	Definition of EcucParamConfContainerDef LinIfRxPdu
[ECUC_LinIf_00049]	Definition of EcucParamConfContainerDef LinIfTxPdu
[SWS_LinIf_00033]	
[SWS_LinIf_00128]	
[SWS_LinIf_00225]	
[SWS_LinIf_00226]	
[SWS_LinIf_00360]	Definition of optional interfaces requested by module LinIf
[SWS_LinIf_00497]	
[SWS_LinIf_00669]	
[SWS_LinIf_00706]	
[SWS_LinIf_00722]	
[SWS_LinIf_00723]	
[SWS_LinIf_00724]	
[SWS_LinIf_00728]	
[SWS_LinIf_00734]	
[SWS_LinIf_00738]	
[SWS_LinIf_00739]	
[SWS_LinIf_00740]	
[SWS_LinIf_00741]	
[SWS_LinIf_00742]	

Table B.2: Changed Specification Items in R24-11

B.1.3 Deleted Specification Items in R24-11

Number	Heading
[ECUC_LinIf_00054]	Definition of EcucFunctionNameDef LinIfTxConfirmationUL
[ECUC_LinIf_00055]	Definition of EcucFunctionNameDef LinIfRxIndicationUL
[ECUC_LinIf_00609]	Definition of EcucEnumerationParamDef LinIfUserTxUL
[ECUC_LinIf_00610]	Definition of EcucEnumerationParamDef LinIfUserRxIndicationUL
[ECUC_LinIf_00628]	Definition of EcucFunctionNameDef LinIfTxTriggerTransmitUL
[SWS_LinIf_00528]	Definition of configurable interface <User_TriggerTransmit>
[SWS_LinIf_00529]	Definition of configurable interface <User_TxConfirmation>
[SWS_LinIf_00530]	Definition of configurable interface <User_RxIndication>

Table B.3: Deleted Specification Items in R24-11

B.2 Traceable item history of this document according to AUTOSAR Release R25-11

B.2.1 Added Specification Items in R25-11

none

B.2.2 Changed Specification Items in R25-11

none

B.2.3 Deleted Specification Items in R25-11

Number	Heading
[SWS_LinIf_00033]	
[SWS_LinIf_00128]	
[SWS_LinIf_00225]	
[SWS_LinIf_00226]	
[SWS_LinIf_00497]	
[SWS_LinIf_00706]	
[SWS_LinIf_00722]	
[SWS_LinIf_00723]	
[SWS_LinIf_00724]	
[SWS_LinIf_00728]	
[SWS_LinIf_00734]	
[SWS_LinIf_00738]	
[SWS_LinIf_00739]	
[SWS_LinIf_00740]	
[SWS_LinIf_00741]	
[SWS_LinIf_00742]	

Table B.4: Deleted Specification Items in R25-11

B.3 Traceable item history of this document according to AUTOSAR Release R26-11

B.3.1 Added Specification Items in R26-11

Number	Heading
[SWS_LinIf_00902]	LinIf State Transition by Go-to-sleep

Table B.5: Added Specification Items in R26-11

B.3.2 Changed Specification Items in R26-11

Number	Heading
[ECUC_LinIf_00019]	Definition of EcucParamConfContainerDef LinIfGeneral
[ECUC_LinTp_00056]	Definition of EcucParamConfContainerDef LinTpGlobalConfig
[ECUC_LinTp_00617]	Definition of EcucParamConfContainerDef LinTpGeneral
[SWS_LinIf_00360]	Definition of optional interfaces requested by module LinIf
[SWS_LinIf_00469]	Definition of imported datatypes of module LinIf
[SWS_LinIf_00729]	Definition of runtime errors in module LinIf
[SWS_LinIf_00750]	
[SWS_LinIf_00755]	

Table B.6: Changed Specification Items in R26-11

B.3.3 Deleted Specification Items in R26-11

Number	Heading
[ECUC_LinIf_00053]	Definition of EcucBooleanParamDef LinIfVersionInfoApi
[ECUC_LinTp_00068]	Definition of EcucBooleanParamDef LinTpVersionInfoApi
[ECUC_LinTp_00639]	Definition of EcucParamConfContainerDef LinTpDemEventParameterRefs
[ECUC_LinTp_00640]	Definition of EcucReferenceDef LINTP_E_LINTPNAS_TIMEOUT_OCCURRED
[ECUC_LinTp_00641]	Definition of EcucReferenceDef LINTP_E_LINTPNCS_TIMEOUT_OCCURRED
[ECUC_LinTp_00642]	Definition of EcucReferenceDef LINTP_E_LINTPNCR_TIMEOUT_OCCURRED
[ECUC_LinTp_00643]	Definition of EcucReferenceDef LINTP_E_SWAPPED_CONSECUTIVE_FRAMES_RECEIVED
[ECUC_LinTp_00644]	Definition of EcucReferenceDef LINTP_E_DROPPED_CONSECUTIVE_FRAMES_DETECTED

▽

△

Number	Heading
[ECUC_LinTp_00645]	Definition of EcucReferenceDef LINIF_E_SCHEDULE_TABLE_SWITCH_REQUEST_NOT_ACCEPTED
[SWS_LinIf_00340]	Definition of API function LinIf_GetVersionInfo
[SWS_LinIf_00352]	Definition of API function LinTp_GetVersionInfo
[SWS_LinIf_00639]	
[SWS_LinIf_00640]	
[SWS_LinIf_00881]	
[SWS_LinIf_00882]	
[SWS_LinIf_00883]	
[SWS_LinIf_00884]	
[SWS_LinIf_00885]	
[SWS_LinIf_00886]	

Table B.7: Deleted Specification Items in R26-11

B.3.4 Added Constraints in R26-11

Number	Heading
[SWS_LinIf_CONSTR_00903]	`LinIfTxPdu` and `LinIfRxPdu` constraint for keeping the local buffer

Table B.8: Added Constraints in R26-11

B.3.5 Changed Constraints in R26-11

none

B.3.6 Deleted Constraints in R26-11

none