INTERNATIONAL STANDARD

**ISO 17987-2**

First edition
2016-08-15

# Road vehicles — Local Interconnect Network (LIN) —

## Part 2: Transport protocol and network layer services

*Véhicules routiers — Réseau Internet local (LIN) —*

*Partie 2: Protocole de transport et couches de services réseau*

Reference number
ISO 17987-2:2016(E)

**COPYRIGHT PROTECTED DOCUMENT**



ISO copyright office
Ch. de Blandonnet 8 • CP 401
CH-1214 Vernier, Geneva, Switzerland
Tel. +41 22 749 01 11
Fax +41 22 749 09 47
copyright@iso.org
www.iso.org

# Contents

Page

# Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular the different approval criteria needed for the different types of ISO documents should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation on the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see the following URL: www.iso.org/iso/foreword.html.

The committee responsible for this document is ISO/TC 22, *Road vehicles*, SC 31, *Data communication*.

A list of all parts in the ISO 17987 series can be found on the ISO website.

# Introduction

This ISO 17987 (all parts) specifies the use cases, the communication protocol and physical layer requirements of an in-vehicle communication network called Local Interconnect Network (LIN).

The LIN protocol as proposed is an automotive focused low speed universal asynchronous receiver transmitter (UART) based network. Some of the key characteristics of the LIN protocol are signal based communication, schedule table based frame transfer, master/slave communication with error detection, node configuration and diagnostic service transportation.

The LIN protocol is for low cost automotive control applications, for example, door module and air condition systems. It serves as a communication infrastructure for low-speed control applications in vehicles by providing:

— signal based communication to exchange information between applications in different nodes;

— bitrate support from 1 kbit/s to 20 kbit/s;

— deterministic schedule table based frame communication;

— network management that wakes up and puts the LIN cluster into sleep mode in a controlled manner;

— status management that provides error handling and error signalling;

— transport layer that allows large amount of data to be transported (such as diagnostic services);

— specification of how to handle diagnostic services;

— electrical physical layer specifications;

— node description language describing properties of slave nodes;

— network description file describing behaviour of communication;

— application programmer's interface.

ISO 17987 (all parts) is based on the open systems interconnection (OSI) basic reference model as specified in ISO/IEC 7498-1 which structures communication systems into seven layers.

The OSI model structures data communication into seven layers called (top down) *application layer* (layer 7), *presentation layer*, *session layer*, *transport layer*, *network layer*, *data link layer* and *physical layer* (layer 1). A subset of these layers is used in ISO 17987 (all parts).

ISO 17987 (all parts) distinguishes between the services provided by a layer to the layer above it and the protocol used by the layer to send a message between the peer entities of that layer. The reason for this distinction is to make the services, especially the application layer services and the transport layer services, reusable also for other types of networks than LIN. In this way, the protocol is hidden from the service user and it is possible to change the protocol if special system requirements demand it.

ISO 17987 (all parts) provides all documents and references required to support the implementation of the requirements related to the following.

— ISO 17987-1: This part provides an overview of the ISO 17987 (all parts) and structure along with the use case definitions and a common set of resources (definitions, references) for use by all subsequent parts.

— ISO 17987-2: This part specifies the requirements related to the transport protocol and the network layer requirements to transport the PDU of a message between LIN nodes.

— ISO 17987-3: This part specifies the requirements for implementations of the LIN protocol on the logical level of abstraction. Hardware related properties are hidden in the defined constraints.

— ISO 17987-4: This part specifies the requirements for implementations of active hardware components which are necessary to interconnect the protocol implementation.

— ISO/TR 17987-5: This part specifies the LIN application programmers interface (API) and the node configuration and identification services. The node configuration and identification services are specified in the API and define how a slave node is configured and how a slave node uses the identification service.

— ISO 17987-6: This part specifies tests to check the conformance of the LIN protocol implementation according to ISO 17987-2 and ISO 17987-3. This comprises tests for the data link layer, the network layer and the transport layer.

— ISO 17987-7: This part specifies tests to check the conformance of the LIN electrical physical layer implementation (logical level of abstraction) according to ISO 17987-4.

# Road vehicles — Local Interconnect Network (LIN) —

# Part 2:
# Transport protocol and network layer services

## 1 Scope

This document specifies a transport protocol and network layer services tailored to meet the requirements of LIN-based vehicle network systems on local interconnect networks. The protocol specifies an unconfirmed communication.

The LIN protocol supports the standardized service primitive interface as specified in ISO 14229-2.

This document provides the transport protocol and network layer services to support different application layer implementations like

— normal communication messages, and

— diagnostic communication messages.

The transport layer defines transportation of data that is contained in one or more frames. The transport layer messages are transported by diagnostic frames. A standardized API is specified for the transport layer.

Use of the transport layer is targeting systems where diagnostics are performed on the backbone bus (e.g. CAN) and where the system builder wants to use the same diagnostic capabilities on the LIN sub-bus clusters. The messages are in fact identical to the ISO 15765-2 and the PDUs carrying the messages are very similar.

The goals of the transport layer are

— low load on LIN master node,

— to provide full (or a subset thereof) diagnostics directly on the LIN slave nodes, and

— targeting clusters built with powerful LIN nodes (not the mainstream low cost).

A typical system configuration is shown in Figure 1.

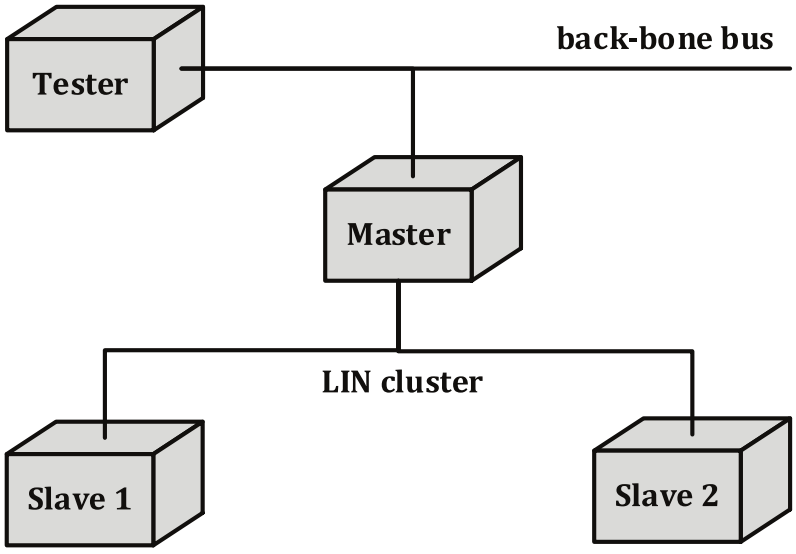



**Figure 1 — Typical system setup for a LIN cluster using the transport layer**

## 2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO 14229-1, *Road vehicles — Unified diagnostic services (UDS) — Part 1: Specification and requirements*

ISO 14229-2, *Road vehicles — Unified diagnostic services (UDS) — Part 2: Session layer services*

ISO 14229-7:2015, *Road vehicles — Unified diagnostic services (UDS) — Part 7: UDS on local interconnect network (UDSonLIN)*

ISO 17987-3:2016, *Road vehicles — Local Interconnect Network (LIN) — Part 3: Protocol specification*

## 3 Terms, definitions, symbols and abbreviated terms

### 3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 7498-1 and the following apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

— IEC Electropedia: available at http://www.electropedia.org/

— ISO Online browsing platform: available at http://www.iso.org/obp

**3.1.1**
**broadcast NAD**
slave node receiving a message with a NAD equal to the broadcast NAD $7F_{16}$ the message is received and processed

**3.1.2**
**configured NAD**
value in the range of ($01_{16}$ to $7D_{16}$) which is assigned to each slave node

Note 1 to entry: The assignment of configured NAD to each slave node is defined in the LDF. The configured NAD is used for node configuration and identification services, as well as UDS services according to ISO 14229-7.

Note 2 to entry: When communication is initialized configured NADs of slave nodes may be identical. The master shall assign unique configured NADs to all slave nodes before diagnostic communication begins.

Note 3 to entry: Setting or altering the configured NAD in a slave node can be done by the following ways:

— the master node assigns a new configured NAD to a slave node supporting the “Assign NAD” service;

— an API call in a slave node assigns the configured NAD;

— the configured NAD is assigned with a static configuration.

**3.1.3**
**functional NAD**
($7E_{16}$)
used to broadcast diagnostic requests

**3.1.4**
**initial NAD**
constant/static value in the range of ($01_{16}$ to $7D_{16}$)

Note 1 to entry: initial NAD value may be derived from a pin configuration, EEPROM or slave node position detection algorithm before entering the operational state (regular LIN communication).

Note 2 to entry: The combination of initial NAD, Supplier ID and Function ID unique for each slave node is used in the “Assign NAD” command allowing an unambiguous configured NAD assignment.

Note 3 to entry: If no initial NAD is defined for a slave node (LDF, NCF) the value is identical to the configured NAD.

**3.1.5**
**P2 timing parameter**
application timing parameter for the ECU(s) and the external test equipment

**3.1.6**
**P2* timing parameter**
enhanced response timing parameter for the ECU(s) application after response pending frame transmission

**3.1.7**
**P4 timing parameter**
timing parameter for the ECU(s) application defining the time between reception of a request and the final response

**3.1.8**
**proprietary NAD**
NAD values in the range [$80_{16}$ – $FF_{16}$] are used for not standardized communication purpose, such as Tier-1 slave node diagnostics

## 3.2 Symbols

% percentage

µs microsecond

ms millisecond

| The vertical bar indicates choice. Either the left hand side or the right hand side of the vertical bar shall appear

## 3.3 Abbreviated terms

API application programmers interface

BNF Bachus-Naur format

CAN Controller Area Network

CF ConsecutiveFrame

FF FirstFrame

LDF LIN description file

L_Data data link data

MRF master request frame

N_AI network address information

N_As network layer timing parameter As

$N\_As_{max}$ timeout on As

N_Cr network layer timing parameter Cr

$N\_Cr_{max}$ timeout on Cr

N_Cs network layer timing parameter Cs

$N\_Cs_{max}$ timeout on Cs

N_Data network data

N_PCI network protocol control information

N_PCItype network protocol control information type

N_PDU network protocol data unit

N_SA network source address

N_SDU network service data unit

N_TAtype network target address type

N_USData network layer LIN data transfer service name

NAD node address for slave nodes

| | |
|---|---|
| NCF | node capability file |
| NCL | node capability language |
| NRC | negative response code |
| NWL | network layer |
| OBD | on-board diagnostics |
| OSI | Open Systems Interconnection |
| PDU | protocol data unit |
| PID | protected identifier |
| RSID | response service identifier |
| SF | SingleFrame |
| SID | service identifier |
| SN | SequenceNumber |
| SRF | slave response frame |
| $ST_{min}$ | SeparationTime minimum |

# 4 Conventions

ISO 17987 (all parts) and ISO 14229-7 are based on the conventions specified in the OSI Service Conventions (ISO/IEC 10731) as they apply for physical layer, protocol, transport protocol and network layer services and diagnostic services.

# 5 Network management

## 5.1 Network management general information

Network management in a LIN cluster refers to cluster wake up and go-to-sleep only. Other network management features, for example, configuration detection and limp home management are left to the application.

## 5.2 LIN node communication state diagram

The state diagram in Figure 2 shows the behaviour model for LIN communication state.

— Bus Sleep

  Bus Sleep state is entered after first connection to power source and the system initialization, reset or when a go-to-sleep command is transmitted by the master or received by the slave node. The level on the bus is set to recessive. Only the wake up signal may be transmitted on the cluster.

— Operational

  The protocol behaviour (transmitting and receiving frames) specified in this document only applies to the Operational state.

NOTE The LIN “Bus Sleep” state does not necessarily correlate to the node’s power state.

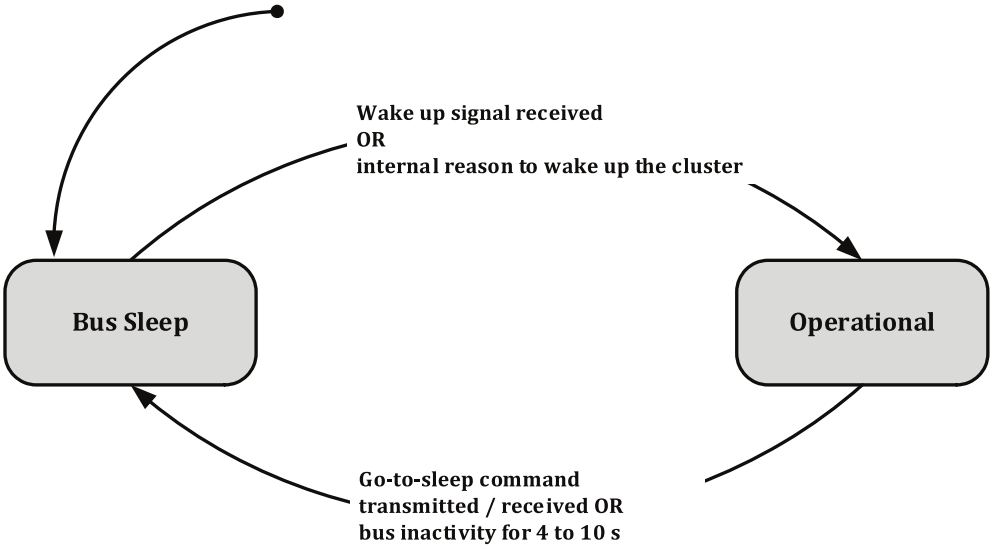



**Figure 2 — LIN node communication state diagram**

## 5.3 Wake up

### 5.3.1 Wake up general information

Any node in a sleeping LIN cluster may request a wake up, by transmitting a wake up signal. The wake up signal is started by forcing the bus to the dominant state for 250 µs to 5 ms, and is valid with the return of the bus signal to the recessive state.

### 5.3.2 Master generated wake up

The master node may issue a break field, e.g. by issuing an ordinary header since the break acts as a wake up signal (in this case, the master shall be aware of that this frame may not be processed by the slave nodes since they may not yet awake and ready to listen to headers).

Every slave node (connected to power) should detect the wake up signal (a dominant pulse longer than 150 µs followed by a rising edge of the bus signal) and be ready to listen to bus commands within 100 ms, measured from the ending edge of the dominant pulse (see Figure 3). The check for the rising edge shall be done by the transceiver and also could be done by the microcontroller LIN interface.

A detection threshold of 150 µs combined with a 250 µs pulse generation gives a detection margin that is enough for uncalibrated slave nodes. Following the detection of the wake up pulse, the Slave task machine (ISO 17987-3:2016, 7.5.3) shall start and enter into the idle state. During the idle state, the slave shall never issue a dominant level pulse on the bus until the state machine enters into an active state.

### 5.3.3 Slave generated wake up

If the node that transmitted the wake up signal is a slave node, it shall be ready to receive or transmit frames immediately. The master node shall also wake up and, when the slave nodes are ready (>100 ms), start transmitting headers to find out the cause (using signals) of the wake up.

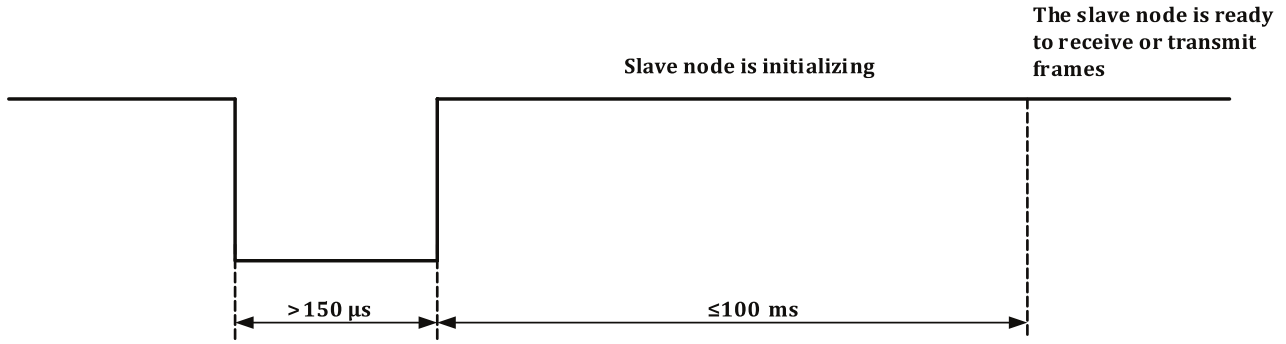



**Figure 3 — Wake up signal reception in slave nodes**

The master node shall detect the wake up signal (a dominant pulse longer than 150 µs followed by a rising edge of the bus signal) and be ready to start communication within a time that is decided by the cluster designer or application specific. The check for the rising edge shall be done by the transceiver and also could be done by the microcontroller LIN interface.

If the master node does not transmit a break field (i.e. starts to transmit a frame) or if the node issuing the wake up signal does not receive a wakeup signal (from another node) within 150 ms to 250 ms from the wake up signal, the node issuing the wake up signal shall transmit a new wake up signal (see Figure 4). In case the slave node transmits a wake up signal in the same time as the master node transmits a break field, the slave shall receive and recognize this break field.

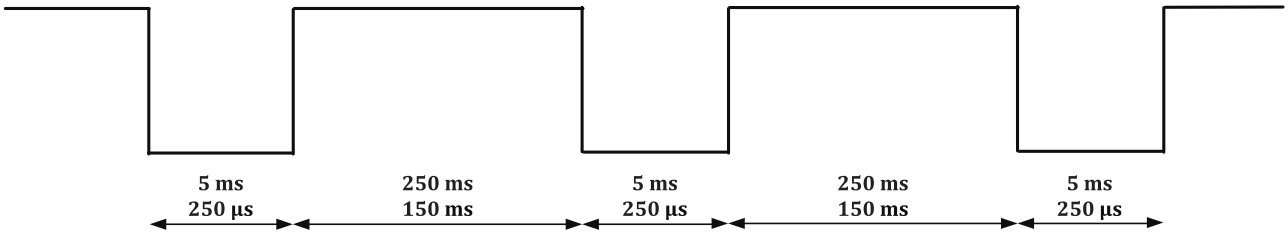



**Figure 4 — One block of wake up signals**

After three (failing) requests, the node shall wait minimum 1,5 s before issuing a fourth wake up signal. The reason for this longer duration is to allow the cluster to communicate in case the waking slave node has problems, e.g. if the slave node has problems with reading the bus it probably retransmit the wake up signal infinitely. There is no restriction of how many times a slave may transmit the wake up signal. However, it is recommended that a slave node transmits not more than one block of three wake up signals for each wake up condition. Figure 5 shows how wake up signals are transmitted over a longer time

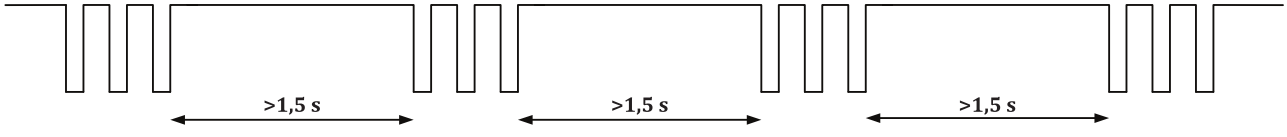



**Figure 5 — Wake up signals over long time**

### 5.4 Go-to-sleep

The master sets each node in the cluster instantly into bus sleep state by transmitting a go-to-sleep command. The request does not necessarily enforce the slave nodes into a low-power mode. The slave node application may still be active after the go-to-sleep command has been received. This behaviour is application specific. The go-to-sleep command is a master request frame with the first data field (Byte 1) set to $00_{16}$ and the rest set to $FF_{16}$ (see Table 1). The slave nodes shall ignore the data fields Byte 2 to Byte 8 and interpret only the first data field (Byte 1).

**Table 1 — Go-to-sleep command**

| LIN frame | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Byte 1** | **Byte 2** | **Byte 3** | **Byte 4** | **Byte 5** | **Byte 6** | **Byte 7** | **Byte 8** |
| $00_{16}$ | $FF_{16}$ | $FF_{16}$ | $FF_{16}$ | $FF_{16}$ | $FF_{16}$ | $FF_{16}$ | $FF_{16}$ |

The normal way for setting the cluster to sleep is that the master node transmits the go-to-sleep command. In case of bus inactivity, a slave node shall be able to receive/transmit frames for 4 s. The slave node shall automatically enter bus sleep mode earliest 4 s and latest 10 s of bus inactivity. Bus inactivity is defined as no transitions between recessive and dominant bit values. Bus activity is the inverse.

NOTE LIN transceivers normally have filters to remove short spikes on the bus. The transition here refers to the signal after this filter.

## 6 Network layer overview

### 6.1 General

This document specifies an unconfirmed network layer communication protocol for the exchange of data between network nodes, e.g. from ECU to ECU, or between external test equipment and an ECU. If the data to be transferred do not fit into a single LIN frame, a segmentation method is provided.

In order to describe the function of the network layer, services provided to higher layers and the internal operation of the network layer shall be considered.

All network layer services have the same general structure. To define the services, three types of service primitives are specified:

— a service request primitive, used by higher communication layers or the application to pass control information and data required to be transmitted to the network layer;

— a service indication primitive, used by the network layer to pass status information and received data to upper communication layers or the application;

— a service confirmation primitive, used by the network layer to pass status information to higher communication layers or the application.

This service specification does not specify an application programming interface, but only a set of service primitives that are independent of any implementation.

## 6.2 Format description of network layer services

All network layer services have the same general format. Service primitives are written in the form:

```
service_name.type     (
                      parameter A,
                      parameter B
                      [,parameter C, ...]
                      )
```

where service_name is the name of the service, e.g. N_USData, type indicates the type of the service primitive, and "parameter A, parameter B [parameter C, ...]" are the N_SDU as a list of values passed by the service primitive. The brackets indicate that this part of the parameter list may be empty.

The service primitives define how a service user (e.g. diagnostic application) cooperates with a service provider (e.g. network layer). The following service primitives are specified in this document: request, indication and confirm.

— Using the service primitive request (`service_name.request`), a service user requests a service from the service provider.

— Using the service primitive indication (`service_name.indication`), the service provider informs a service user about an internal event of the network layer or the service request of a peer protocol layer entity service user.

— With the service primitive confirm (`service_name.confirm`), the service provider informs the service user about the result of a preceding service request of the service user.

## 6.3 Internal operation of network layer

The internal operation of the network layer provides methods for segmentation and reassembly. The main purpose of the network layer is to transfer messages that might or might not fit in a single LIN frame. Messages that do not fit into a single LIN frame are segmented into multiple parts, where each can be transmitted in a LIN frame.

Figures 6 and 7 show, respectively, an example of an unsegmented message transmission and of a segmented message transmission.

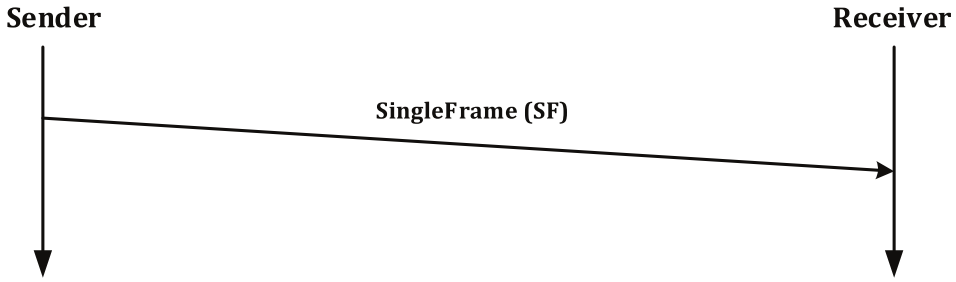



**Figure 6 — Example of an unsegmented message**

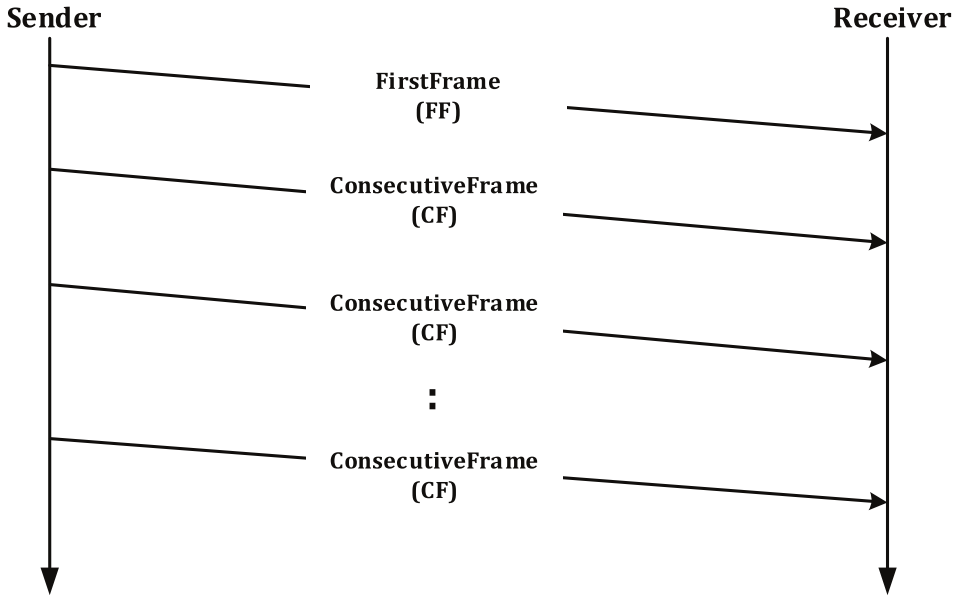



**Figure 7 — Example of a segmented message**

## 6.4 Service data unit specification

In the following subclauses, all network layer service parameters are described which are used by the LIN network layer services.

### 6.4.1 N_AI, address information

#### 6.4.1.1 N_AI description

The `N_AI` is used to identify the communicating peer entities of the network layer. `N_AI` consists of a LIN ID and a `N_NAD`. Two dedicated LIN IDs are used for any transport layer protocol communication.

ID $3C_{16}$ (MasterReq) is assigned to the message transmissions sent by the master node.

ID $3D_{16}$ (SlaveResp) is assigned to the message transmissions sent by any slave node.

Because the node type (Master or Slave) is always fixed for all nodes in a LIN cluster, the LIN ID used is no subject of the service primitive parameters (6.5.1) but assigned to each node by this document.

Additionally, a `N_NAD` comprises the slave node addressed by the request/response and also the addressing type (physical, functional and broadcast).

These parameters refer to addressing information. As a whole, the `N_AI` parameters are used to identify the message senders and recipients as well as the communication model for the message (`N_TAtype`).

#### 6.4.1.2 N_NAD, Network NAD

Type: 1 byte unsigned integer value

Range: $01_{16}$ – $FF_{16}$

Description: Two communication models are specified

— 1 to 1 communication, called physical addressing, and

— 1 to n communication called functional addressing or broadcast addressing.

Depending on the N_NAD value the communication model is given

— $00_{16}$: not applicable, reserved for sleep command frame,

— $01_{16}$ – $7D_{16}$: physical addressing type,

— $7E_{16}$: functional addressing type,

— $7F_{16}$: broadcast addressing type, and

— $80_{16}$ – $FF_{16}$: proprietary addressing type.

Physical addressing shall be supported for all types of network layer messages. Responses on requests are expected. The N_NAD value shall always be validated against the configured NAD in a slave node.

Functional addressing shall only be supported for SingleFrame communication. No responses shall be transmitted on functional requests. The master node is the only node transmitting functional messages.

Broadcast addressing shall be supported for all types of network layer messages. Responses are unexpected, but supported, even if they could raise collisions. The master node shall be the only node transmitting broadcast messages.

The addressing type of NAD values in range $80_{16}$ – $FF_{16}$ is unassigned by this document and shall be assigned by users when this range of NAD is used.

### 6.4.2 <Length>

Type: 12 bit

Range: $001_{16}$ to $FFF_{16}$

Description: This parameter includes the length of data to be transmitted/received.

### 6.4.3 <MessageData>

Type: string of bytes

Range: not applicable

Description: This parameter includes all data the higher layer entities exchange.

### 6.4.4 <N_Result>

Type: enumeration

Range: N_OK, N_TIMEOUT_As, N_TIMEOUT_Cs, N_TIMEOUT_Cr, N_WRONG_SN, N_UNEXP_PDU, N_ERROR, N_UNEXP_NEW_REQ

Description: This parameter contains the status relating to the outcome of a service execution. If two or more errors are discovered at the same time, then the network layer entity shall use the parameter value first found in this list in the error indication to the higher layers.

— N_OK

This value means that the service execution has completed successfully, it can be issued to a service user on both the sender and receiver side.

— N_TIMEOUT_As

This value is issued to the service user on the sender side when the timer N_As has passed its timeout value $N\_As_{max}$.

— N_TIMEOUT_Cs

This value is issued to the protocol user on the sender side when the timer N_Cs has passed its timeout value $N\_Cs_{max}$.

— N_TIMEOUT_Cr

This value is issued to the service user on the sender side when the timer N_Cr has passed its timeout value $N\_Cr_{max}$.

— N_WRONG_SN

This value is issued to the service user upon reception of an unexpected SequenceNumber (N_PCI.SN) value, it can be issued to the service user on the receiver side only.

— N_UNEXP_PDU

This value is issued to the service user upon reception of an unexpected protocol data unit, it can be issued to the service user on the receiver or transmitter side.

— N_ERROR

This is the general error value. It shall be issued to the service user when an error has been detected by the network layer and no other parameter value can be used to better describe the error. It can be issued to the service user on both the sender and receiver side.

— N_UNEXP_NEW_REQ

This value is issued

a) to the lower priority service instances with an active diagnostic communication as parameter of N_USData.indication() upon service call of a new higher priority call of N_USData.request,

b) when a N_USData.request can't be accepted due to a currently active diagnostic communication with a higher priority, and

c) this value is issued to the service user upon service call of a new N_USData.request with the same priority.

## 6.5 Services provided by network layer to higher layers

### 6.5.1 Specification of network layer service primitives

The service interface defines a set of services that are needed to access the functions offered by the network layer, i.e. transmission/reception of data and setting of protocol parameters.

The communication services, of which the following are defined, enable the transfer of up to 4 095 bytes of data.

a) N_USData.request

This service is used to request the transfer of data. If necessary, the network layer segments the data.

b) N_USData_FF.indication

This service is used to signal the beginning of a segmented message reception to the upper layer.

c) N_USData.indication

This service is used to provide received data to the higher layers.

d) N_USData.confirm

This service confirms to the higher layers that the requested service has been carried out (successfully or not).

### 6.5.2 N_USData.request

The service primitive requests transmission of <MessageData> with <Length> bytes from the sender to the receiver peer entities identified by the address information in N_NAD (see 6.4.1.2 for parameter definition).

Each time the N_USData.request service is called, the network layer shall signal the completion (or failure) of the message transmission to the service user by means of the issuing of a N_USData.confirm service call:

```
N_USData.request     (
                     N_NAD
                     <MessageData>
                     <Length>
                     )
```

### 6.5.3 N_USData.confirm

The N_USData.confirm service is issued by the network layer. The service primitive confirms the completion of a N_USData.request service identified by the address information in N_NAD. The parameter <N_Result> provides the status of the service request (see 6.4.4 for parameter definition).

```
N_USData.confirm      (
                       N_NAD
                       <N_Result>
                       )
```

### 6.5.4 N_USData_FF.indication

The N_USData_FF.indication service is issued by the network layer. The service primitive indicates to the adjacent upper layer the arrival of a FirstFrame (FF) of a segmented message received from a peer protocol entity, identified by the address information in N_NAD (see 6.4.1.2 for parameter definition). This indication shall take place upon reception of the FirstFrame (FF) of a segmented message.

```
N_USData_FF.indication      (
                            N_NAD
                            <Length>
                            )
```

The N_USData_FF.indication service shall always be followed by an N_USData.indication service call from the network layer, indicating the completion (or failure) of the message reception.

A N_USData_FF.indication service call shall only be issued by the network layer if a correct FirstFrame (FF) message segment has been received.

If the network layer detects any type of error in a FirstFrame (FF), then the message shall be ignored by the network layer and no N_USData_FF.indication shall be issued to the adjacent upper layer.

If the network layer receives a FirstFrame (FF) with a data length value (FF_DL) that is greater than the available receiver buffer size, then this shall be considered as an error condition and no N_USData_FF.indication shall be issued to the adjacent upper layer.

### 6.5.5 N_USData.indication

The N_USData.indication service is issued by the network layer. The service primitive indicates <N_Result> events and delivers <MessageData> with <Length> bytes received from a peer protocol entity identified by the address information in N_NAD to the adjacent upper layer (see 6.4.1.2 for parameter definition).

The parameters <MessageData> and <Length> are only valid if <N_Result> equals N_OK.

```
N_USData.indication     (
                        N_NAD
                        <MessageData>
                        <Length>
                        <N_Result>
                        )
```

The N_USData.indication service call is issued after the reception of a SingleFrame (SF) message or as an indication of the completion (or failure) of a segmented message reception.

If the network layer detects any type of error in a SingleFrame (SF), then the message shall be ignored by the network layer and no N_USData.indication shall be issued to the adjacent upper layer.

# 7 Transport layer protocol

## 7.1 Protocol functions

The transport layer protocol performs the following functions:

— transmission/reception of messages up to 4 095 data bytes;

— reporting of transmission/reception completion/failure occurrence.

## 7.2 Single frame transmission

Figure 8 shows the transmission of a SingleFrame (SF) called LIN frame, which consists of 8 bytes (Byte 1 … Byte 8) with Byte 1 = N_ NAD, Byte 2 = N_PCI and Byte 3 … Byte 8 = N_Data. The actual message payload of a SF (see 7.4.2) is up to 6 data bytes (see also 8.3).

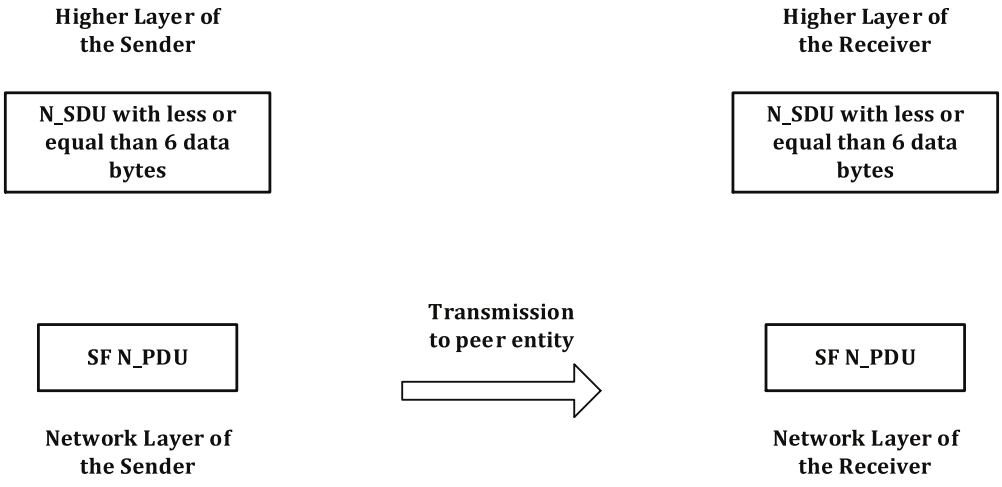



**Figure 8 — Example of a SingleFrame (SF) transmission**

## 7.3 Multiple frame transmission

Transmission of longer messages is performed via segmentation of the message into LIN frames (transmission of multiple N_PDUs). Reception of longer messages is performed via reception of multiple LIN frames (N_PDUs) and reassembly of the received data bytes (concatenation) into a message. The multiple N_PDUs are called FirstFrame (for the first N_PDU of the message) and ConsecutiveFrame (for all the following N_PDUs).

Messages longer than the allowed number of data bytes contained in a SF N_PDU are segmented into

— a FirstFrame protocol data unit (FF N_PDU), containing Byte 1 = N_ NAD, Byte 2 + Byte 3 = N_PCI and Byte 4 .. Byte 8 = N_Data. The actual payload of a FF (see 7.4.3 and 8.3) is 5 data bytes, and

— one or more ConsecutiveFrame protocol data units (CF N_PDU), containing Byte 1 = N_NAD, Byte 2 = N_PCI and Byte 3 .. Byte 8 = N_Data. The actual payload of a CF (see 7.4.4 and 8.3) is 6 data bytes. The last CF N_PDU contains the remaining valid payload data bytes N_Data. Unused data bytes in the last CF N_PDU are stuffed with value $FF_{16}$.

Figure 9 shows segmentation at the sender side and reassembly at the receiver side.

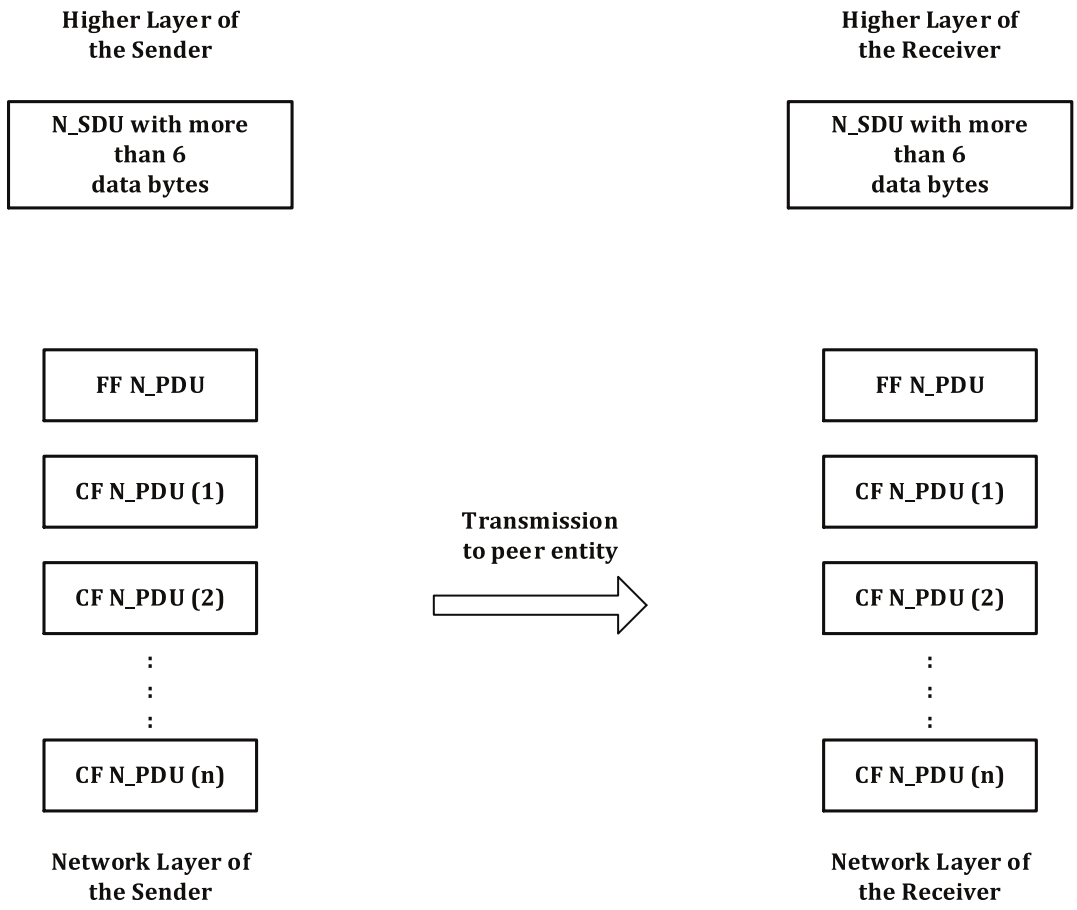



**Figure 9 — Example of a multiple frame transmission (segmentation and reassembly)**

The message length is transmitted in the FF N_PDU. All CF N_PDUs are numbered by the sender to support verification of correct order of transmission.

The master node application layer shall conform to the slave nodes capabilities. This can be realized by schedule table design, scheduling mode (diagnostic only vs. interleaved mode) or by application specific scheduling control. The transport layer of receiving slave nodes does not observe the $ST_{min}$.

These slave node capabilities are defined as follows.

SeparationTime minimum ($ST_{min}$) is the minimum time the master node shall wait between the transmissions of a FF and its first CF, as well as two CF. The separation time is statically specified for each slave node (LDF/NCF) when supporting transport layer communication. The measurement of the $ST_{min}$ starts after completion of transmission of a ConsecutiveFrame (CF) and ends at the request for the transmission of the next CF.

A slave node shall not take care on the capabilities of the master node ($ST_{min}$) because it is the responsibility of the master node application layer to apply an appropriate scheduling.

Capabilities of all nodes is that, each time the sender/receiver waits for an N_PDU from the receiver/sender, a timeout mechanism allows detection of a transmission failure (see 7.6.2).

## 7.4 Transport layer protocol data units

### 7.4.1 Protocol data unit types

The communication between the peer protocol entities of the network layer in different nodes is done by means of exchanging N_PDUs.

This document specifies three different types of transport layer protocol data units:

— SingleFrame (SF N_PDU);

— FirstFrame (FF N_PDU);

— ConsecutiveFrame (CF N_PDU);

which are used to establish a communication path between the peer network layer entities, to exchange communication parameters, to transmit user data and to release communication resources.

### 7.4.2 SF N_PDU

The SF N_PDU is identified by the SingleFrame protocol control information (SF N_PCI). The SF N_PDU shall be sent out by the sending network entity and received by all network entities. Only the one entity that is addressed by the N_AI shall proceed this request for the duration of the segmented message transmission. Other nodes which are not addressed shall reject already pending connections. It shall be sent out to transfer a service data unit that can be transferred via a single service request to the data link layer, and to transfer unsegmented messages.

### 7.4.3 FF N_PDU

The FF N_PDU is identified by the FirstFrame protocol control information (FF N_PCI). The FF N_PDU shall be sent out by the sending network entity and received by all network entities. Only the one entity that is addressed by the N_AI shall proceed this request for the duration of the segmented message transmission. Other nodes which are not addressed shall reject already pending connections. It identifies the first N_PDU of a segmented message transmitted by a network sending entity and received by a receiving network entity. The receiving network layer entity shall start assembling the segmented message on receipt of a FF N_PDU.

### 7.4.4 CF N_PDU

The CF N_PDU is identified by the ConsecutiveFrame protocol control information (CF N_PCI). The CF N_PDU transfers segments (N_Data) of the service data unit message data (`<MessageData>`). All N_PDUs transmitted by the sending entity after the FF N_PDU shall be encoded as CF N_PDUs. The receiving entity shall pass the assembled message to the service user of the network receiving entity after the last CF N_PDU has been received. The CF N_PDU shall be sent out by the sending network entity and received by a unique receiving network entity for the duration of the segmented message transmission.

### 7.4.5 Protocol data unit field description

#### 7.4.5.1 N_PDU format

The protocol data unit (N_PDU) enables the transfer of data between the network layer in one node and the network layer in one or more other nodes (peer protocol entities). All N_PDUs consist of three fields, as specified in Table 2.

**Table 2 — N_PDU format**

| Address information | Protocol control information | Data field |
|---|---|---|
| N_AI | N_PCI | N_Data |

#### 7.4.5.2 Address information (N_AI)

The N_AI is used to identify the communicating peer entities of the network layer. The N_AI information received in the N_SDU — N_NAD shall be copied and included in the N_PDU. If the message data (`<MessageData>` and `<Length>`) received in the N_SDU is so long that segmentation is needed for the network layer to transmit the complete message, the N_NAD shall be copied and included (repeated) in every N_PDU that is transmitted.

This field contains address information that identifies the type of message exchanged and the recipient(s) and sender between whom data exchange takes place.

NOTE For a detailed description of address information, see 6.4.1.

#### 7.4.5.3 Protocol control information (N_PCI)

This field identifies the type of N_PDUs exchanged. It is also used to exchange other control parameters between the communicating network layer entities.

NOTE For a detailed specification of all N_PCI parameters, see 7.5.

#### 7.4.5.4 Data field (N_Data)

The N_Data in the N_PDU is used to transmit the service user data received in the `<MessageData>` parameter in the `N_USData.request` service call. The `<MessageData>`, if needed, is segmented into smaller parts that each fit into the N_PDU data field before they are transmitted over the network.

The size of N_Data depends on the N_PDU type and on the number of remaining `MessageData` in case of a SF or the last CF belonging to the message.

## 7.5 Protocol control information specification

### 7.5.1 N_PCI

Each N_PDU is identified by means of an N_PCI (see Tables 3 and 4).

**Table 3 — Summary of N_PCI bytes**

| N_PDU name | N_PCI bytes | | |
|---|---|---|---|
| | Byte #1 | | Byte #2 |
| | Bits 7 – 4 | Bits 3 – 0 | Bits 7 – 0 |
| SingleFrame (SF) | N_PCItype = 0 | SingleFrame_DataLength (SF_DL) | N/A |
| FirstFrame (FF) | N_PCItype = 1 | FirstFrame_DataLength (FF_DL) | |
| ConsecutiveFrame (CF) | N_PCItype = 2 | SequenceNumber (SN) | N/A |

**Table 4 — Definition of N_PCItype values**

| Value | Description |
|---|---|
| $0_{16}$ | **SingleFrame (SF)**<br>For unsegmented messages, the network layer protocol provides an optimized implementation of the network protocol with the message length embedded in the N_PCI byte only. SF shall be used to support the transmission of messages that can fit in a single LIN frame. |
| $1_{16}$ | **FirstFrame (FF)**<br>A FF shall only be used to support the transmission of messages that cannot fit in a single LIN frame, i.e. segmented messages. The FF of a segmented message is encoded as a FF. On receipt of a FF, the receiving network layer entity shall start assembling the segmented message. |
| $2_{16}$ | **ConsecutiveFrame (CF)**<br>When sending segmented data, all CFs following the FF are encoded as CF. On receipt of a CF, the receiving network layer entity shall assemble the received data bytes until the whole message is received. The receiving entity shall pass the assembled message to the adjacent upper protocol layer after the last frame of the message has been received without error. |
| $3_{16}$ | **Reserved**<br>This range of values is reserved by this document. A FlowControl is not supported with LIN. |
| $4_{16}$ – $F_{16}$ | **Reserved**<br>This range of values is reserved by ISO 15765-2. |

### 7.5.2 SingleFrame N_PCI parameter definition

#### 7.5.2.1 SF N_PCI byte

Table 5 provides an overview of the SF N_PCI byte.

**Table 5 — Overview of SF N_PCI byte**

| N_PDU name | SF N_PCI byte | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Byte #1 | | | | | | | |
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| SingleFrame | 0 | 0 | 0 | 0 | SingleFrame_DataLength (SF_DL) | | | |

#### 7.5.2.2 SingleFrame_Data Length (SF_DL) parameter definition

The parameter SingleFrame Data Length (SF_DL) is used in the SF N_PDU to specify the number of service user data bytes. See Table 6.

**Table 6 — Definition of SF_DL values**

| Value | Description |
|---|---|
| $0_{16}$ | **Invalid**<br>This value is invalid. |
| $1_{16}$ – $6_{16}$ | **SingleFrame_Data Length (SF_DL)**<br>The SF_DL is encoded in the low nibble of N_PCI byte #1 value. It shall be assigned the value of the service parameter <Length>. |
| $7_{16}$ – $F_{16}$ | **Invalid**<br>This range of values is invalid. |

### 7.5.3 FirstFrame N_PCI parameter definition

#### 7.5.3.1 FF N_PCI bytes

Table 7 provides an overview of the FF N_PCI bytes.

**Table 7 — Overview of FF N_PCI bytes**

| N_PDU name | FF N_PCI bytes | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Byte #1 | | | | | | | | Byte #2 |
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 – 0 |
| FirstFrame | 0 | 0 | 0 | 1 | FirstFrame_DataLength (FF_DL) | | | | |

#### 7.5.3.2 FirstFrame_DataLength (FF_DL) parameter definition

The parameter FF_DL is used in the FF N_PDU to specify the number of service user data bytes. See Table 8.

**Table 8 — Definition of FF_DL values**

| Value | Description |
|---|---|
| $0_{16}$ – $6_{16}$ | **Invalid**<br>This range of values is invalid. |
| $7_{16}$ – $FFF_{16}$ | **FirstFrame_DataLength (FF_DL)**<br>The encoding of the segmented message length results in a twelve bit length value (FF_DL) where the least significant bit (LSB) is specified to be bit "0" of the N_PCI byte #2 and the most significant bit (MSB) is bit three of the N_PCI byte #1. The maximum segmented message length supported is equal to 4095 bytes of user data. It shall be assigned the value of the service parameter <Length>. |

### 7.5.4 ConsecutiveFrame N_PCI parameter definition

#### 7.5.4.1 CF N_PCI byte

Table 9 provides an overview of the CF N_PCI byte.

**Table 9 — Overview of CF N_PCI byte**

| N_PDU name | CF N_PCI byte | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Byte #1 | | | | | | | |
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ConsecutiveFrame | 0 | 0 | 1 | 0 | SequenceNumber (SN) | | | |

#### 7.5.4.2 SequenceNumber (SN) parameter definition

The parameter SN is used in the ConsecutiveFrame (CF) N_PDU to specify the following.

— The numeric ascending order of the ConsecutiveFrames (CF).

— The SN shall start with zero for all segmented messages. The FirstFrame (FF) shall be assigned the value zero. It does not include an explicit SequenceNumber (SN) in the N_PCI field, but shall be treated as the segment number zero.

— The SN of the first ConsecutiveFrame (CF) that immediately follows the FirstFrame (FF) shall be set to one.

— The SN shall be incremented by one for each new ConsecutiveFrame (CF) that is transmitted during a segmented message transmission.

— When the SN reaches the value of $F_{16}$, it shall wrap around and be set to $0_{16}$ for the next ConsecutiveFrame (CF).

This shall lead to the sequence given in Table 10.

**Table 10 — Summary of SN definition**

| N_PDU | FF | CF | CF | CF | CF | CF | CF | CF |
|---|---|---|---|---|---|---|---|---|
| SN | $0_{16}$ | $1_{16}$ | ... | $E_{16}$ | $F_{16}$ | $0_{16}$ | $1_{16}$ | ... |

See Table 11 for definition of SN values.

**Table 11 — Definition of SN values**

| Value | Description |
|---|---|
| $0_{16}$ to $F_{16}$ | **SequenceNumber (SN)**<br>The SN shall be encoded in the lower nibble bits of N_PCI byte #1. The SN shall be set to a value within the range of $0_{16}$ – $F_{16}$. |

## 7.6 Network layer timing

### 7.6.1 Timing constraints

The timing constraints for the transport layer (based on ISO 15765-2) are defined in Table 12. The properties shall be within a defined range. Since LIN is slower than CAN, the values shall be adjusted accordingly. These properties are part of the transport layer and shall not have any constraint on the node configuration.

Performance requirement values are the binding communication requirements to be met by each communication peer in order to be compliant with the specification. Since the LIN schedules vary with the specific use case, a certain application may define specific performance requirements within the ranges defined in Table 12. The time between transport layer N_SDUData.request and data link layer start of frame transmission depends on the schedule table definition and the schedule mode (see 9.6.4 for more information).

Timeout values are defined to be higher than the values for the performance requirements to ensure a working system and to overcome communication conditions where the performance requirement can absolutely not be met (e.g. high bus load). Specified timeout values in Table 12 or values given by the node (i.e. the NCF) shall be treated as the upper limit for any given implementation.

The network layer shall issue an appropriate service primitive to the network layer service user upon detection of an error condition.

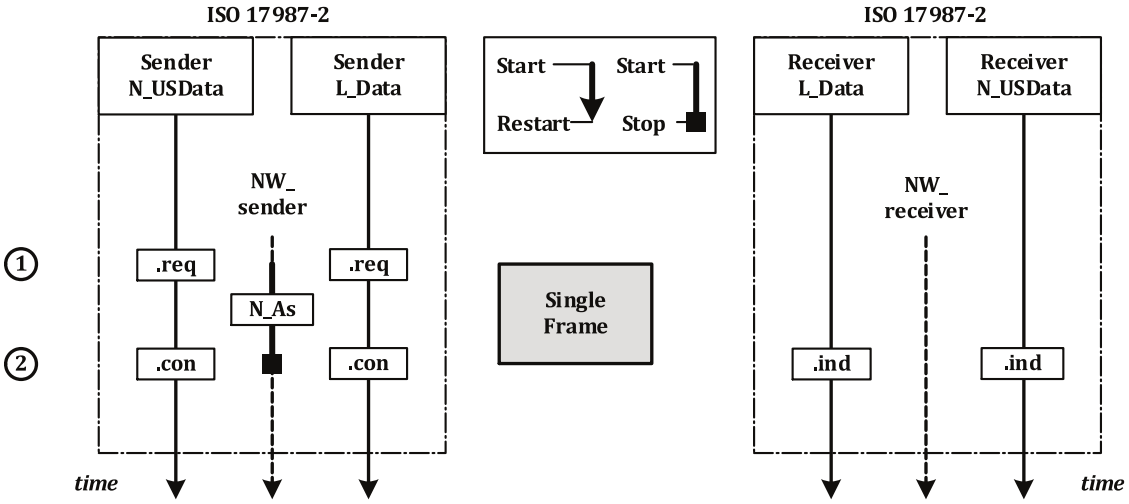
If a communication path is established between peer protocol entities, identified by N_AI (see 6.4.1 and 7.4.5.2 for further details), a single set of network layer timing parameters is assigned statically to this communication path. For the selection of the network layer timing parameters no other information beside N_AI is used.

**Table 12 — Network layer timing parameter definition**

| Timing parameter | Description | Data link layer service | | Timeout ms | Performance requirement ms |
|---|---|---|---|---|---|
| | | Start | End | | |
| **N_As** | Time for transmission of the LIN frame (MRF or SRF) on the transmitter side | When the transport layer requests a diagnostic frame to be transmitted | When the diagnostic frame has been confirmed as transmitted | 1 000 | N/A |
| **N_Cs** | Time until request to transmit the next Consecutive-Frame (CF) | When the previous diagnostic frame in the same message has been confirmed as transmitted. | When the transport layer requests the CF to be transmitted | N/A | (N_Cs+N_As) < (0,9*$N\_Cr_{max}$) |
| **N_Cr** | Time until reception of the next Consecutive-Frame (CF) | When the previous diagnostic frame in the message has been indicated as received. | When the next diagnostic frame in the message has been indicated as received. | 1 000 | — |

The N_Cs parameter does not require a timeout monitoring in the transmitting node since N_As ensures the correct timeout behaviour. However, N_Cs shall be considered in the system design (scheduling and transmitter software design) so that a timeout on the receiver's side (N_Cr) can be avoided.

Figure 10 shows the network layer timing parameters of an unsegmented message, while Table 12 defines the network layer timing parameter values and their corresponding start and end positions based on the data link layer services.




**Key**

1 Sender N_USData.req: the session layer issues an unsegmented message to the transport layer.

Sender L_Data.req: the transport layer requests the SingleFrame transmission to the data link layer and starts the N_As timer. This diagram shows the timing relationship between the data link and the transport layers and does not imply actual bus message scheduling.

2 Receiver L_Data.ind: the data link layer issues to the transport layer the reception of the LIN frame.

Receiver N_USData.ind: the transport layer issues to the session layer the completion of the unsegmented massage.

Sender L_Data.con: the data link layer confirms to the transport layer that the transport layer LIN frame has been transmitted successfully. The sender stops the N_As timer.

Sender N_USData.con: the transport layer issues to the session layer the completion of the unsegmented message.

**Figure 10 — Placement of network layer timing parameters — unsegmented message**

Figures 11 and 12 illustrate the parameters in the time domain. The intention of the figures is to show the transport layer timing parameters, not to require a certain implementation. The behaviour for the master node and the slave node in the lower layers are generalized.

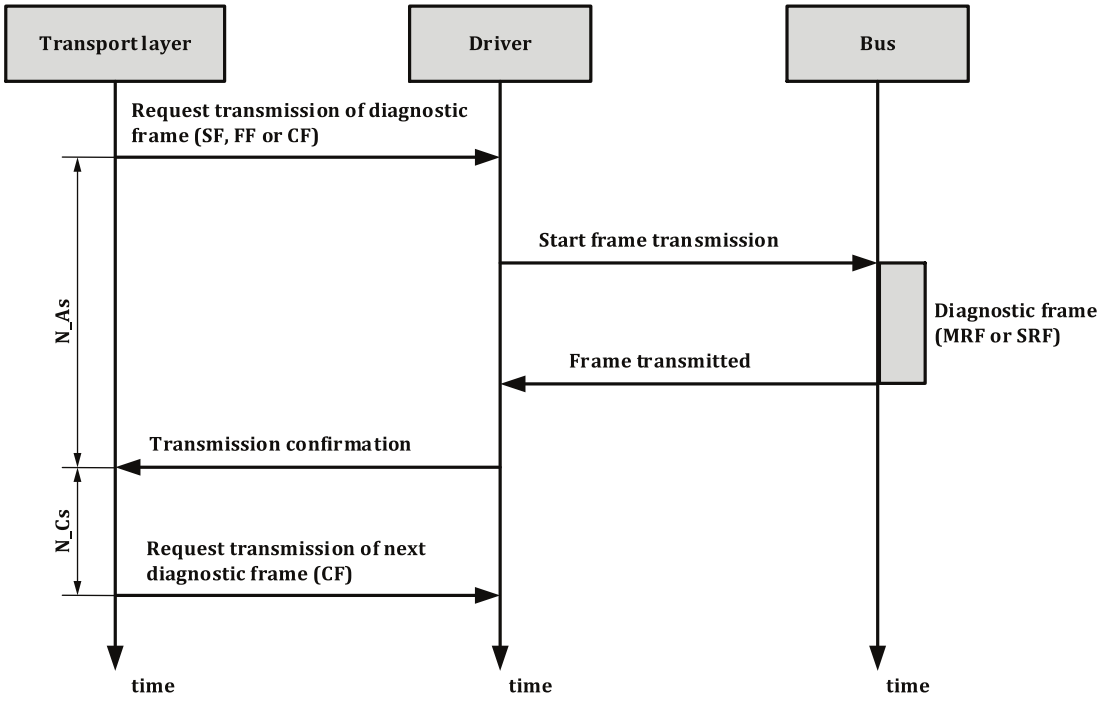



**Figure 11 — Transport layer timing on transmitter side**

Figure 12 shows the transport layer timing on receiver side.

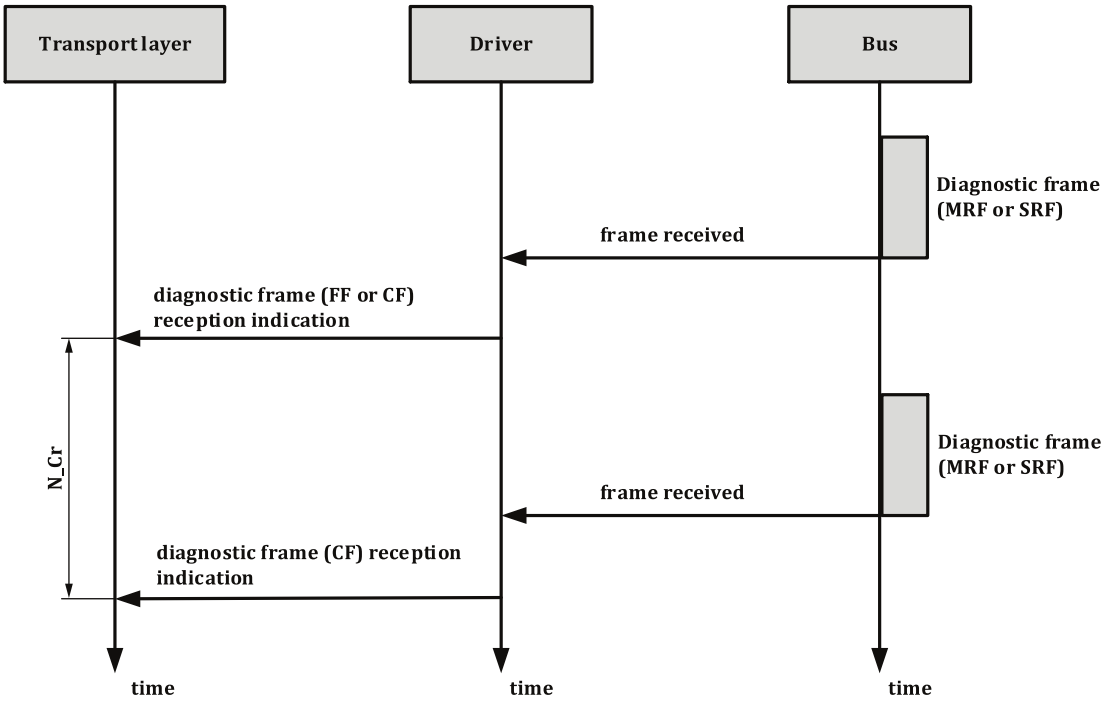
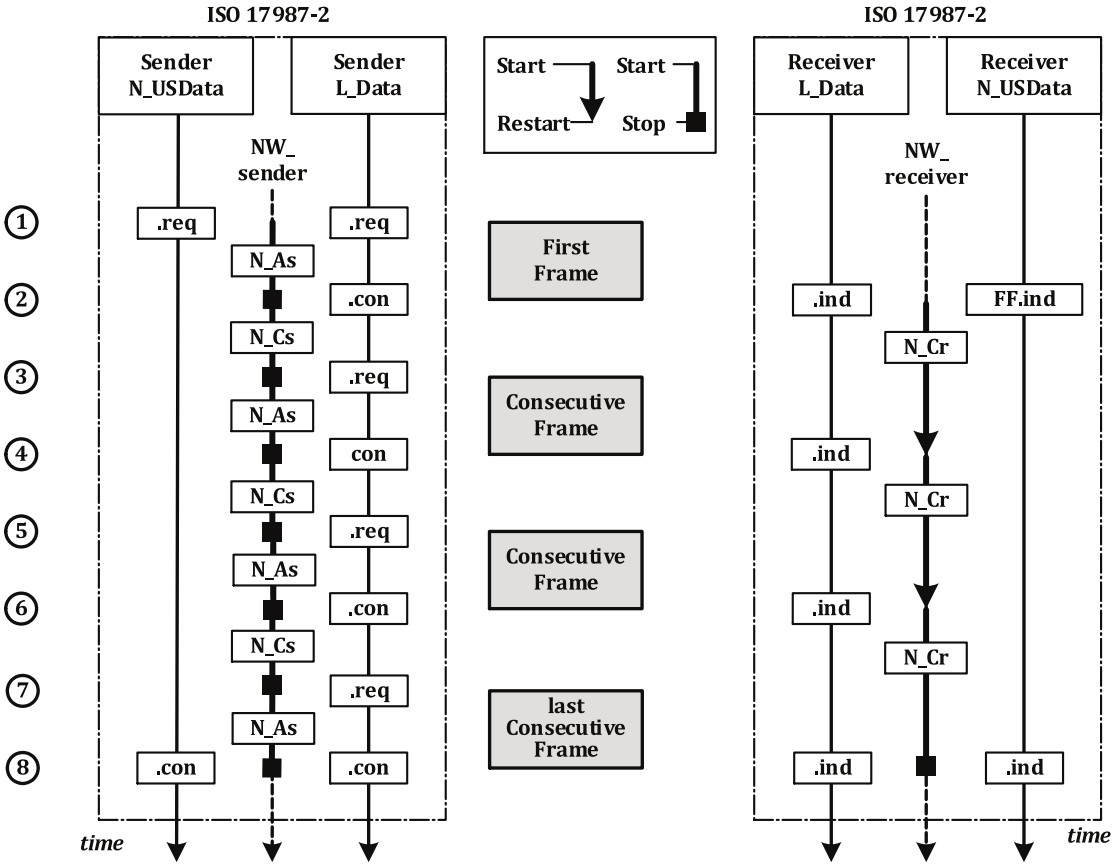



**Figure 12 — Transport layer timing on receiver side**

Figure 13 shows the network layer timing parameters of a segmented message, while Table 12 defines the network layer timing parameter values and their corresponding start and end positions based on the data link layer services.




**Key**

1 Sender N_USData.req: the session layer issues a segmented message to the transport layer.

Sender L_Data.req: the transport layer requests the FirstFrame transmission to the data link layer and starts the N_As timer. This diagram shows the timing relationship between the data link and the transport layers and does not imply actual bus message scheduling.

2 Receiver L_Data.ind: the data link layer issues to the transport layer the reception of the LIN transport layer frame. The receiver starts the N_Cr timer.

Receiver N_USData_FF.ind: the transport layer issues to the session layer the reception of a FirstFrame of a segmented message.

Sender L_Data.con: the data link layer confirms to the transport layer that the LIN frame has been transmitted successfully. The sender stops the N_As timer and starts the N_Cs timer.

3 Sender L_Data.req: the transport layer requests the first ConsecutiveFrame transmission to the data link layer and starts the N_As timer.

4, 6 Receiver L_Data.ind: the data link layer issues to the transport layer the reception of the LIN frame. The receiver restarts the N_Cr timer.

Sender L_Data.con: the data link layer confirms to the transport layer that the LIN frame has been transmitted successfully. The sender stops the N_As timer and starts the N_Cs timer according to the separation time value ($ST_{min}$).

5, 7 Sender L_Data.req: when the N_Cs timer is elapsed ($ST_{min}$) the transport layer requests the next ConsecutiveFrame transmission to the data link layer and starts the N_As timer.

8 Receiver L_Data.ind: the data link layer issues to the transport layer the reception of the LIN frame. The receiver stops the N_Cr timer.

Sender L_Data.con: the data link layer confirms to the transport layer that the LIN frame has been transmitted successfully. The sender stops the N_As timer.

Receiver N_USData.ind: the transport layer issues to the session layer the completion of the segmented massage.

Sender N_USData.con: the transport layer issues to session layer the completion of the segmented message.

**Figure 13 — Placement of network layer timing parameters — segmented message**

### 7.6.2 Network layer timeouts

Table 13 defines the cause and action in a network layer timeout.

**Table 13 — Network layer timeout error handling**

| Timeout | Cause | Action |
|---|---|---|
| **N_As** | Any N_PDU not transmitted in time on the sender side. | Abort message transmission and issue N_USData.confirm with <N_Result> = N_TIMEOUT_As. |
| **N_Cs** | ConsecutiveFrame N_PDU not transmitted in time on the sender side. | Abort message transmission and issue N_USData.confirm with <N_Result> = N_TIMEOUT_Cs. |
| **N_Cr** | ConsecutiveFrame N_PDU not received (disturbed or not transmitted) on the receiver side. | Abort message reception and issue N_USData.indication with <N_Result> = N_TIMEOUT_Cr. |

### 7.6.3 Network layer error handling

The following error handling applies.

— A SingleFrame PDU (SF) with a DataLength (DL) value greater than six bytes shall be ignored by the receiver.

— A FirstFrame PDU (FF) with a DataLength (DL) value less than seven bytes shall be ignored by the receiver.

— A FirstFrame PDU (FF) with a DataLength (DL) value greater than the maximum available receive buffer size of the slave node shall be ignored by the receiver and the receiver shall not start the reception of a segmented message. This implies of course that the receiving node is receiving the complete message (the target node) and not a fragmented (in case of gateway).

— If the network layer receives a SF with a SF_DL equal to zero then the network layer shall ignore the received SF N_PDU.

— PDUs with unexpected N_PCI types from any node shall be ignored except SingleFrame (SF) and FirstFrame PDUs (FF).

— When a SingleFrame (SF) or FirstFrame (FF) PDU is received in a slave node, with a NAD not equal to the functional NAD, while a message reception (request) or transmission (response) is already ongoing the current reception or transmission shall be aborted. Reception of the new message shall be started if the NAD equals the node's configured NAD or broadcast NAD.

— When a SingleFrame (SF) or FirstFrame (FF) PDU is received in a master node while a message reception (response) is already ongoing the current reception shall be aborted.

— Reception of the new message shall only be started if the NAD equals the NAD that was addressed in the appropriate request.

— If a CF N_PDU message is received with an SequenceNumber (SN) not according to the definition in 7.5.4.2, the message reception shall be aborted, and the network layer shall make a `N_USData.indication` service call with the parameter `<N_Result> = N_WRONG_SN` to the adjacent upper layer.

— The message reception shall be aborted by the receiver after occurrence of an $N\_Cr_{max}$ timeout.

— The message transmission shall be aborted by the transmitter after occurrence of an N_As timeout (see Table 13).

— If the time between two subsequent CFs of a segmented data transmission (N_As + N_Cs) is smaller than the $ST_{min}$ value or the receiving node there is no guarantee that the receiver of the segmented data transmission correctly receives and process all frames. In any case, the receiver of the segmented data transmission is not required to monitor the adherence of the $ST_{min}$ value.

### 7.6.4 Unexpected arrival of N_PDU

An unexpected N_PDU is defined as one that has been received by a node outside the expected order of N_PDUs. It could be an N_PDU defined by this document (SF N_PDU, FF N_PDU, CF N_PDU) that is received out of the normal expected order or with a NAD that is different to the currently used NAD. As described in 7.6.3 already a N_PDU with unknown PCI type is ignored.

Tables 14 and 15 define the network layer behaviour in the case of the reception of an unexpected N_PDU for Master and Slave node. It considers the current internal network layer status (NWL status) and different possible NAD values received.

When the specified action is to ignore an unexpected N_PDU, this means that the network layer shall not notify the upper layers of its arrival.

**Table 14 — Master node handling of unexpected N_PDUs**

| NWL status | Reception of SF N_PDU | Reception of FF N_PDU | Reception of CF N_PDU |
|---|---|---|---|
| Segmented transmission in progress | Ignore | Ignore | Ignore |
| Segmented reception in progress | **New NAD equals current NAD**:<br>Terminate the current reception, report a N_USData.indication, with <N_Result> set to N_UNEXP_PDU, to the upper layer, and process the SF N_PDU as the start of a new reception.<br>**Other NAD**:<br>Ignored | **New NAD equals current NAD**:<br>Terminate the current reception, report a N_USData.indication, with <N_Result> set to N_UNEXP_PDU, to the upper layer, and process the FF N_PDU as the start of a new reception.<br>**Other NAD**:<br>Ignored | **New NAD equals current NAD**:<br>Terminate the current reception, report a N_USData.indication, with <N_Result> set to N_WRONG_SN, to the upper layer. (Only a wrong SequenceNumber can be unexpected)<br>**Other NAD**:<br>Ignored |

**Table 15 — Slave node handling of unexpected N_PDUs**

| NWL status | Reception of SF N_PDU | Reception of FF N_PDU | Reception of CF N_PDU |
|---|---|---|---|
| Segmented transmission in progress | **New NAD equals configured NAD or broadcast NAD**:<br>Terminate the current transmission, report a N_USData.confirm with <N_Result> set to N_UNEXP_PDU, to the upper layer, and process the SF N_PDU as the start of a new reception.<br>**Functional NAD**:<br>Ignore<br>**Other NAD:**<br>Terminate the current transmission, report a N_USData.confirm with <N_Result> set to N_UNEXP_PDU, to the upper layer. A different slave is addressed. | **New NAD equals configured NAD or broadcast NAD**:<br>Terminate the current transmission, report a N_USData.confirm with <N_Result> set to N_UNEXP_PDU, to the upper layer, and process the FF N_PDU as the start of a new reception.<br>**Functional NAD**:<br>Ignore (invalid N_PDU format)<br>**Other NAD**:<br>Terminate the current transmission, report a N_USData.confirm with <N_Result> set to N_UNEXP_PDU, to the upper layer. A different slave is addressed. | Ignore |
| Segmented reception in progress | **New NAD equals configured NAD or broadcast NAD**:<br>Terminate the current reception, report a N_USData.indication, with <N_Result> set to N_UNEXP_PDU, to the upper layer, and process the SF N_PDU as the start of a new reception.<br>**Functional NAD**:<br>Ignore<br>**Other NAD**:<br>Terminate the current reception, report a N_USData.indication, with <N_Result> set to N_UNEXP_PDU, to the upper layer. A different slave is addressed. | **New NAD equals configured NAD or broadcast NAD**:<br>Terminate the current reception, report a N_USData.indication, with <N_Result> set to N_UNEXP_PDU, to the upper layer, and process the FF N_PDU as the start of a new reception.<br>**Functional NAD**:<br>Ignore (invalid N_PDU format)<br>**Other NAD**:<br>Terminate the current reception, report a N_USData.indication, with <N_Result> set to N_UNEXP_PDU, to the upper layer. A different slave is addressed. | **New NAD equals configured NAD**:<br>Terminate the current reception, report a N_USData.indication, with <N_Result> set to N_WRONG_SN, to the upper layer. (Only a wrong Sequence-Number can be unexpected)<br>**Other NAD**:<br>Ignored |

## 8 Data link layer usage

The data link layer services defined in this document are providing an interface for transport/network layer services only. Any data exchange on LIN is controlled by the usage of schedule tables in the master node. Service defined therefore always requires a LIN master node application which serves the current communication demand. In case of diagnostic communication using transport layer services the master nodes application shall activate appropriate schedule tables that contain MRF and SRF.

## 8.1 Data link layer service parameters

The following data link layer service parameters are defined in ISO 17987-3.

— <Data>: LIN frame data

— <Transfer_Status>: status of a transmission

## 8.2 Data link layer interface services

### 8.2.1 L_Data.request

The service primitive requests transmission of `<Data>` that shall be mapped into a MRF/SRF. `<Data>` consist of the N_NAD, N_PCI and N_Data field. In case of the SF or last CF additional stuffing data may also be provided.

```
L_Data.request      (
                    <Data>
                    )
```

### 8.2.2 L_Data.confirm

The service primitive confirms the completion of an `L_Data.request` service.

The parameter `<Transfer_Status>` provides the status of the service request:

```
L_Data.confirm      (
                     <Transfer_Status>
                      )
```

### 8.2.3 L_Data.indication

The service primitive indicates data link layer event to the adjacent upper layer and delivers `<Data>` identified:

```
L_Data.indication     (
                      <Data>
                      )
```

## 8.3 Mapping of the N_PDU fields

N_PCI and N_Data are placed in the LIN frame data field (see Table 16). If the N_PDU to be transmitted in a SF or last CF is shorter than SF N_PDU/CF N_PDU filed the sender shall pad any unused bytes in the frame with $FF_{16}$.

**Table 16 — Mapping of N_PDU parameters into LIN frame**

| | LIN frame | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | **Byte 1** | **Byte 2** | **Byte 3** | **Byte 4** | **Byte 5** | **Byte 6** | **Byte 7** | **Byte 8** |
| **N_PDU type** | **LIN frame address field (N_NAD)** | **LIN frame data field** | | | | | | |
| | | **N_PCI** | **LIN SF N_PDU field** | | | | | |
| **SingleFrame (SF)** | NAD | SF | N_Data/Byte Padding | | | | | |
| | | **N_PCI** | **LIN FF N_PDU field** | | | | | |
| **FirstFrame (FF)** | NAD | FF | N_Data | | | | | |
| | | **N_PCI** | **LIN CF N_PDU field** | | | | | |
| **ConsecutiveFrame (CF)** | NAD | CF | N_Data | | | | | |
| | | **N_PCI** | **LIN CF N_PDU field** | | | | | |
| **last CF** | NAD | CF | N_Data/Byte Padding | | | | | |

## 8.4 Transport layer PDU structure and communication

### 8.4.1 PDU structure

#### 8.4.1.1 Mapping of CAN to LIN and LIN to CAN frame data field

The units that are transported in a transport layer frame are called PDU. A PDU can be a complete message or part of a message; in the latter case, multiple concatenated PDUs form the complete message.

Messages issued by the client (tester, master node) are called requests and messages issued by the server (master node, slave node) are called responses.

The first byte in the payload is used as a node address (NAD). The transport layer frames have fixed frame IDs, since the diagnostic frames are used. This means that the addressing of a node (or function) is made using the NAD.

**IMPORTANT — No FlowControl frames are supported in LIN clusters. If the backbone bus, for example, CAN test equipment needs FlowControl PDUs, these shall be generated by the master node on the backbone (CAN) side.**

A routing of backbone bus transport layer frames (such as defined in ISO 15765-2) to a LIN cluster is possible on data link layer already when the mapping of N_PDU fields is similar to LIN. This means that Byte 1 of the back bone bus N_PDU is used to identify the LIN subscriber(s) and can be converted by the LIN master into a N_NAD.

In case of CAN, this condition is fulfilled for

— 11 bit CAN ID mixed addressing Tp,

— 29 bit CAN ID mixed addressing Tp, and

— extended addressing.

NOTE Extended addressing uses a target address N_TA definition. Only in case each slave node represents an own unambiguous N_TA value in the master node a conversion into N_NAD is possible. Otherwise, data link layer routing is not supported by mixed addressing.

This data link layer routing is depicted in Figure 14 and called Tp raw.

If data mapping of the back bone bus does not match the LIN format, routing is supported on application/session layer level. The message data reception shall be confirmed by a N_USData.indication

service call issued by the back bone bus network layer. The master node application/session layer shall forward this message as a new N_USData.request service request to the LIN network layer. Reception is handled accordingly.

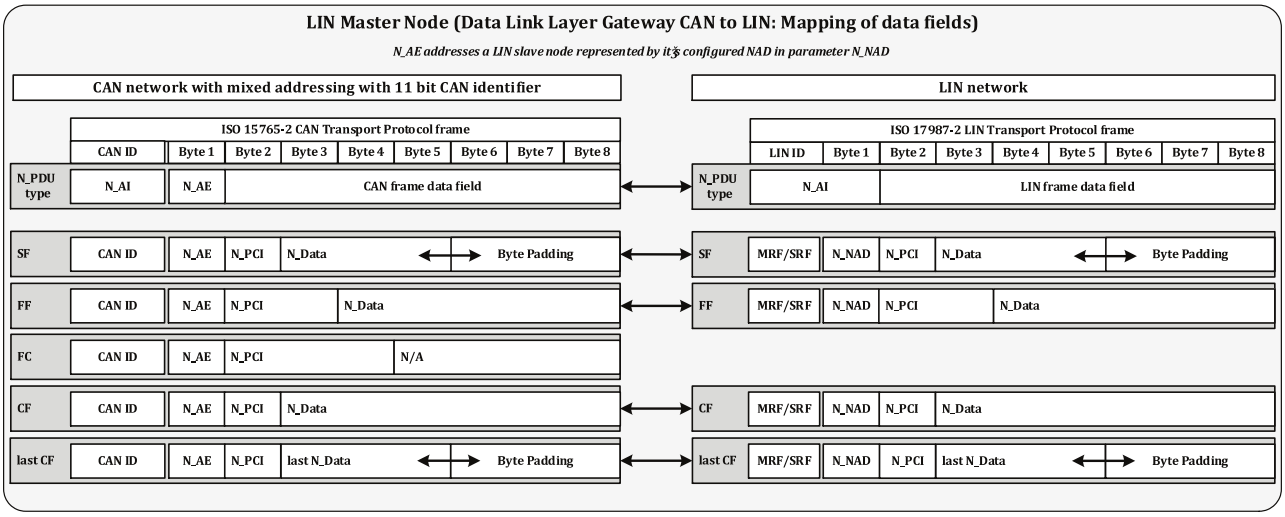



**Figure 14 — Mapping of data fields of 11 bit CAN ID with mixed addressing**

To simplify conversion between ISO 15765-2 transport layer frames with 29 bit CAN IDs and transport layer frames of this document a very similar structure is defined which supports the PDU types shown in Figure 15.

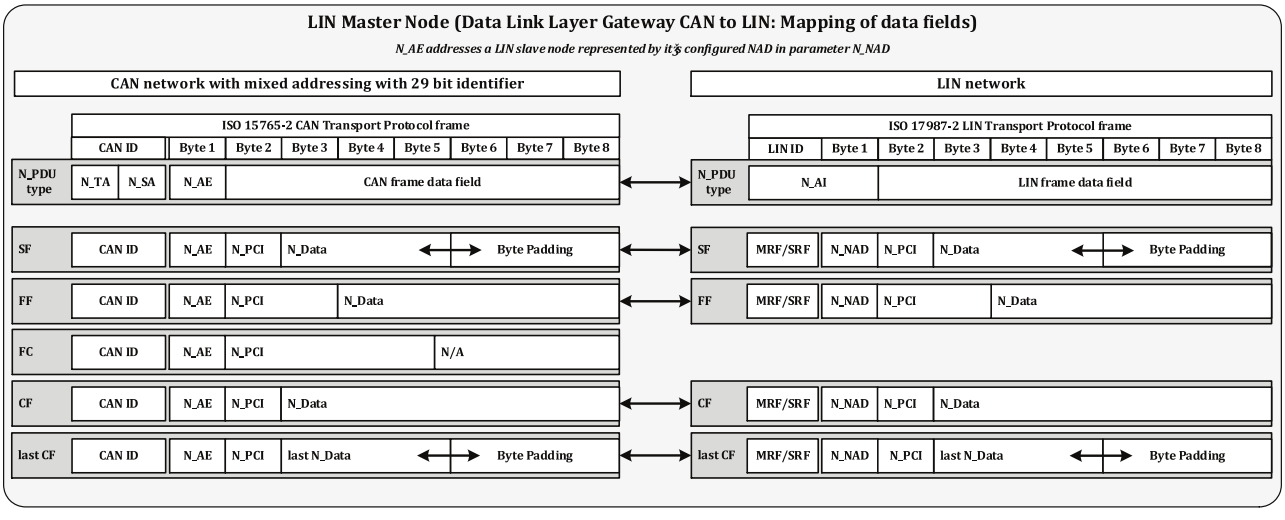



**Figure 15 — Mapping of data fields of 29 bit CAN ID with mixed addressing**

Requests are always sent in master request frames and responses are always sent in slave response frames.

### 8.4.1.2 Service identifier (SID)

The service identifier (SID) specifies the request that shall be performed by the slave node addressed. $00_{16}$ .. $AF_{16}$ and $C0_{16}$ .. $FE_{16}$ are used for diagnostics while $B0_{16}$ .. $BF_{16}$ are used for node configuration and node identification (see ISO 17987-3). The Response Service Identifier (RSID), SID+$40_{16}$, specifies to which requested service the response belongs to.

The Response Service Identifier (RSID), SID + $40_{16}$, specifies the content of the response message.

#### 8.4.1.3 Data field (N_Data)

The interpretation of the data bytes depends on the SID or RSID. In multi-PDU messages, all data bytes in all PDUs of the message shall be concatenated into a complete message, before being parsed.

If a PDU is not completely filled with payload data (applies to last CF and SF PDUs only), the publisher of the PDU shall pad the unused bytes with $FF_{16}$.

### 8.4.2 Communication

#### 8.4.2.1 General

It is required that a transport layer message is exclusive on one bus. This means that no inter-leaving frames are allowed to the “request receiving” and “response transmission” of a slave during a multi-frame transport message with the exception of a functional request.

## 9 Diagnostic communication requirements

### 9.1 Definition of diagnostic classes

#### 9.1.1 General

Architectural, diagnostic communication performance and transport protocol needs of slave nodes are accommodated by dividing diagnostic services functionality into three diagnostic classes.

Therefore, a diagnostic class is assigned to each slave node according to its level of diagnostic functionality and complexity.

#### 9.1.2 Diagnostic class I

Smart and simple devices like intelligent sensors and actuators, requiring none or very low amount of diagnostic functionality. Actuator control, sensor reading and fault memory handling is done by the master node, using signal carrying frames. Therefore, specific diagnostic support for these tasks is not required. Fault indication is always signal based.

#### 9.1.3 Diagnostic class II

A diagnostic class II slave node is similar to a diagnostic class I slave node, but it provides node identification support. The extended node identification is normally required by vehicle manufacturers. Testers or master nodes use ISO 14229-1 diagnostic services to request the extended node identification information. Actuator control, sensor reading and fault memory handling is done by the master node, using signal carrying frames. Therefore, specific diagnostic support for these tasks is not required. Fault indication is always signal based.

#### 9.1.4 Diagnostic class III

Diagnostic class III slave nodes are devices with enhanced application functions, typically performing their own local information processing (e.g. function controllers, local sensor/actuator loops). The slave nodes execute tasks beyond the basic sensor/actuator functionality, and therefore require extended diagnostic support. Direct actuator control and raw sensor data is often not exchanged with the master node, and therefore not included in signal carrying frames. ISO 14229-1 diagnostic services for I/O control, sensor value reading and parameter configuration (beyond node configuration) are required.

Diagnostic class III slave nodes have internal fault memory, along with associated reading and clearing services. Optionally, reprogramming (flash/NVRAM reprogramming) of the slave node is possible. This requires an implementation of a boot loader and necessary diagnostic services to unlock the device initiate downloads and transfer data, etc.

The primary difference between diagnostic class II and diagnostic class III is the distribution of diagnostic capabilities between the LIN master node and the LIN slave node for diagnostic class II while for a diagnostic class III LIN slave node no diagnostic application features of the LIN slave node are implemented in the LIN master node.

### 9.1.5 Summary of slave node diagnostic classes

Table 17 shows a list of diagnostic services supported by the different diagnostic classes. All supported configuration and diagnostic services of a slave node are listed in the node capability file, see Node Capability Language Specification. See ISO 17987-3:2016, 6.3.4.4 for a detailed description of node configuration and identification services. ISO 14229-7 provides the list of UDS based services for LIN.

**Table 17 — Slave node diagnostic properties**

| Slave diagnostic class | I | II | III | Service identifier |
|---|---|---|---|---|
| **Diagnostic transport protocol requirements** | | | | |
| SingleFrame (SF) transport only | mandatory | N/A | N/A | N/A |
| Full function transport protocol (multi-segment) | N/A | mandatory | mandatory | N/A |
| **Required configuration services** | | | | |
| AssignNAD | optional | optional | optional | $B0_{16}$ |
| AssignFrameIdentifier (legacy) | optional | optional | optional | $B1_{16}$ |
| ReadByIdentifier ($00_{16}$ = Product ID) | mandatory | mandatory | mandatory | $B2_{16}$ $00_{16}$ |
| ReadByIdentifier (all except Product ID) | optional | optional | mandatory | $B2_{16}$ $XX_{16}$ |
| DataDump | optional | optional | optional | $B4_{16}$ |
| AutoAddressingSlave (legacy SID) | optional | optional | optional | $B5_{16}$ |
| Targeted Reset (SAE J2602 nodes only) | mandatory | mandatory | mandatory | $B5_{16}$ |
| SaveConfiguration | optional | optional | optional | $B6_{16}$ |
| AssignFrameIdentifierRange | mandatory | mandatory | mandatory | $B7_{16}$ |
| AutoAddressingSlave | optional | optional | optional | $B8_{16}$ |

## 9.2 Diagnostic messages

The LIN transport layer uses the same diagnostic messages as defined in ISO 14229-7. The SID and RSID shall be according to ISO 14229-7. A node may implement a subset of the services defined in ISO 14229-7.

Some very simple LIN slaves (ASIC-based) may implement a fault reporting form of diagnostics using custom parameters in the data fields. Since these parameters may be vehicle manufacturer or application-specific, they should be processed by the LIN node application.

## 9.3 Using the transport layer

Two communication cases exist using the transport layer, the master node wants to transmit a diagnostic request to a slave node or the slave node wants to transmit a diagnostic response. There are several use cases for a master node why a request message is sent

— diagnostic request from backbone,

— slave node self-diagnostics, and

— slave node configuration and identification.

Figures 16 and 17 show the message flow in these two cases.

It is important that the unit controlling the communication (the tester or the master) avoids requesting multiple slaves to respond simultaneously as each new request is rejecting any pending response from a slave node addressed before.

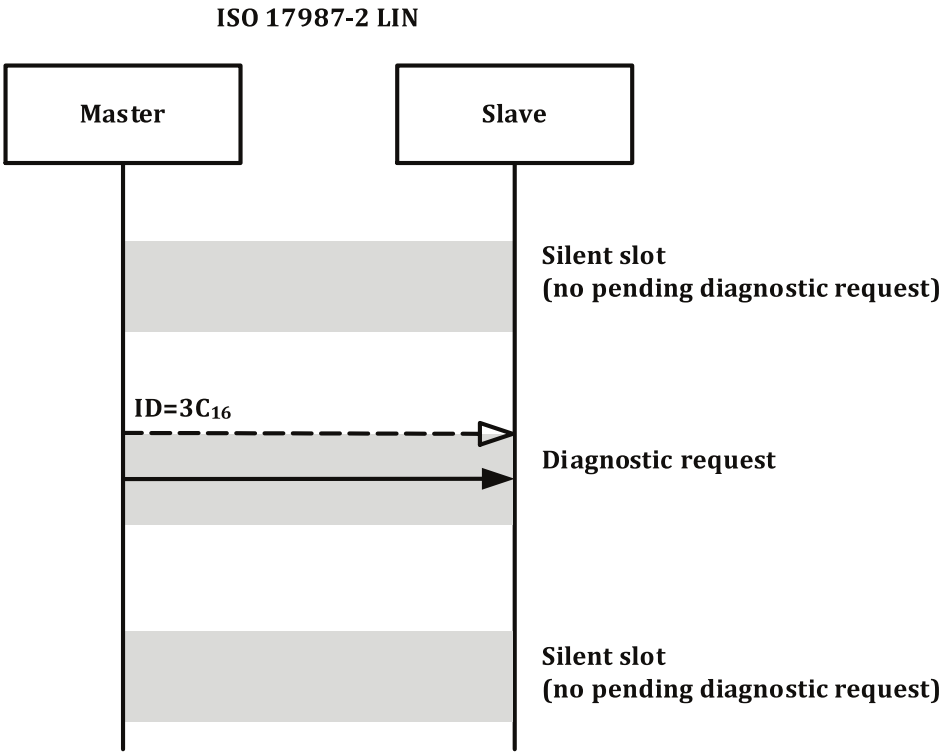



**Figure 16 — Diagnostic request**

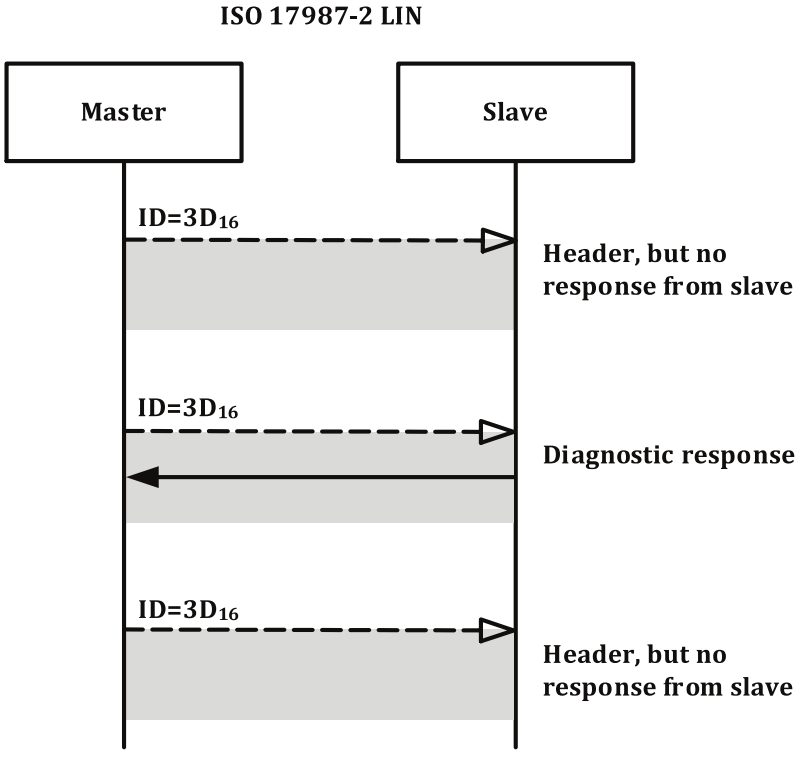



**Figure 17 — Diagnostic response**

## 9.4 Slave node diagnostic timing requirements

This subclause contains the timing parameter requirements that shall be taken into account when designing the LIN cluster. The monitoring for diagnostic class II and III slave nodes shall be implemented by the slave node itself.

Table 18 defines the diagnostic communication timings.

**Table 18 — Diagnostic communication timings**

| Parameter | Affected device | Description | Minimum value /performance requirement | Maximum value/ timeout |
|---|---|---|---|---|
| P2 | master node | 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.<br>The maximum value defines the time after which a slave node shall have received a slave response header before it discards its response.<br>Each slave node defines this minimum value in the NCF or LDF, see 11.3.2 and 12.3.4. | 50 ms | 500 ms |
| $ST_{min}$ | master node | The minimum time the slave node needs to prepare the reception of the next frame of a diagnostic request or the transmission of the next frame of a diagnostic response.<br>Each slave node defines this minimum value in the NCF or LDF, see 11.3.2 and 12.3.4. | 0 ms | n/a |
| P2* | master node | Time between sending a NRC $78_{16}$ and the LIN-slave being able to provide data for a response. | P2 | 2 000 ms |

A timing sequence chart of the diagnostic communication is shown in Figure 18. The external test equipment is shown only as an example.

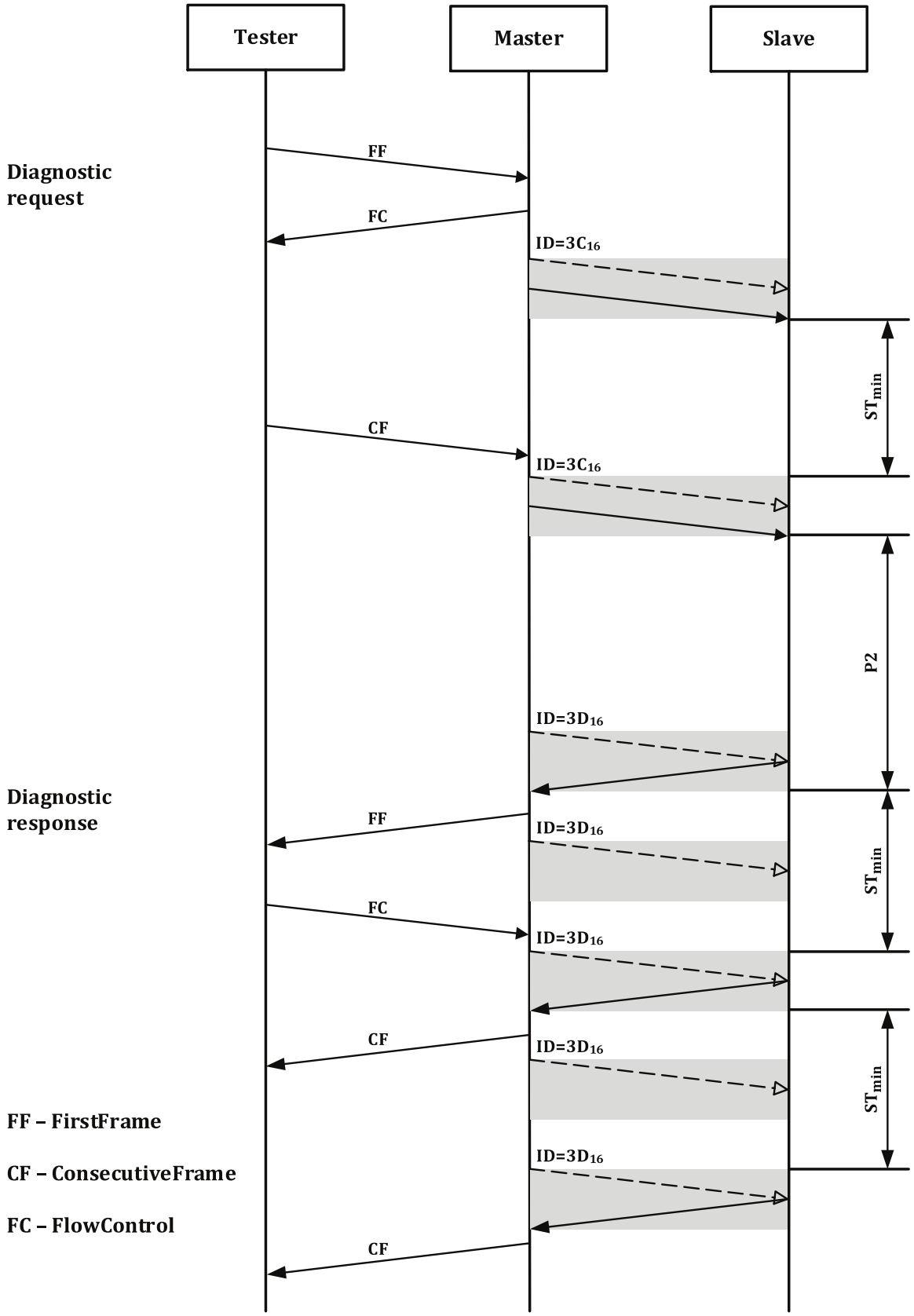



**Figure 18 — Timing sequence chart of the diagnostic communication from the tester to LIN via a backbone bus**

## 9.5 Response pending

Slave nodes of diagnostic class II and III are supporting ISO 14229-7 UDSonLIN based diagnostic services. In case a service requires more time than P2 for processing, a response pending frame according to ISO 14229-1 and ISO 14229-2 is transmitted by the slave node to extend the maximum response time. A response pending frame may be repeated within P4 timing. Whenever a response pending frame is used a final positive or negative response is mandatory for this request. A master node receiving a response pending frame extends the response timeout to P2* and continues waiting for the final slave response.

The response pending frame is defined as a SingleFrame negative response with NRC $78_{16}$.

Table 19 defines the response pending frame format.

**Table 19 — Response pending frame format**

| NAD | PCI | RSID | D1 | D2 | D3 | D4 | D5 |
|---|---|---|---|---|---|---|---|
| NAD | $03_{16}$ | $7F_{16}$ | SID | $7F_{16}$ | $FF_{16}$ | $FF_{16}$ | $FF_{16}$ |

## 9.6 Transport protocol handling in LIN master

### 9.6.1 General

The LIN master is responsible for handling the scheduling according to the currently active diagnostic transmissions. This subclause defines the requirements for schedules and schedule handling which shall be implemented to enable diagnostic communication with any slave node. The master node acts as a network layer router between the backbone bus and the LIN cluster, implying that the transport protocols on the backbone bus and on the LIN cluster are handled by the master.

Within the next chapters several communication sequence diagrams are used. Refer to Figure 19 for the explanation of communication diagrams. For clearness of the communication sequence charts, the size of the interleaved normal communication schedules as well as the size of the diagnostic schedules do not represent the correct delays in the schedules and is only a schematic graphical representation.

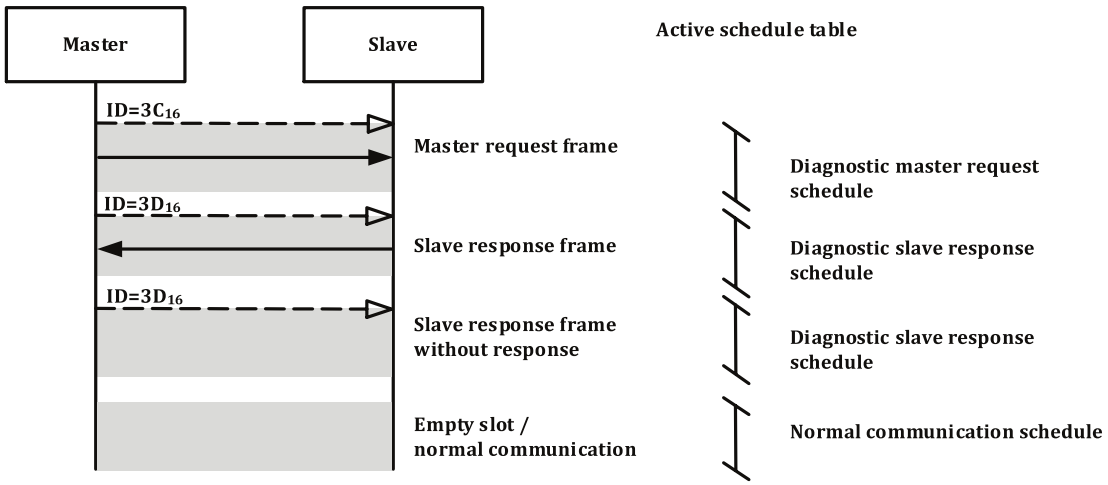



**Figure 19 — Legend of communication sequence charts**

### 9.6.2 Diagnostic master request schedule

The master node shall support a diagnostic master request schedule table that contains a single master request frame. It is up to the LDF designer to add this specific schedule table.

The diagnostic master request schedule table shall be executed whenever a master request frame shall be transmitted (see Figure 20).

NOTE By insertion of an execution of the diagnostic master request schedule table, the overall timing of the normal communication schedule is affected.

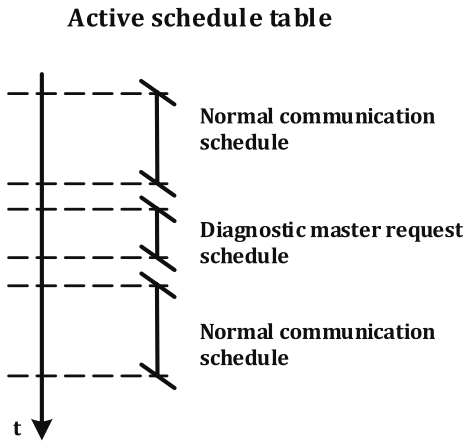



**Figure 20 — Interleaving of normal communication schedule table**

### 9.6.3 Diagnostic slave response schedule

The master node shall support a diagnostic slave response schedule table that contains a single slave response frame. It is up to the LDF designer to add this specific schedule table.

The diagnostic slave response schedule table shall be inserted between the executions of the normal communication schedules whenever a slave response frame shall be transmitted (see Figure 21).

NOTE Note that by insertion of an additional execution of the diagnostic slave response schedule table the overall timing of the normal communication is affected.

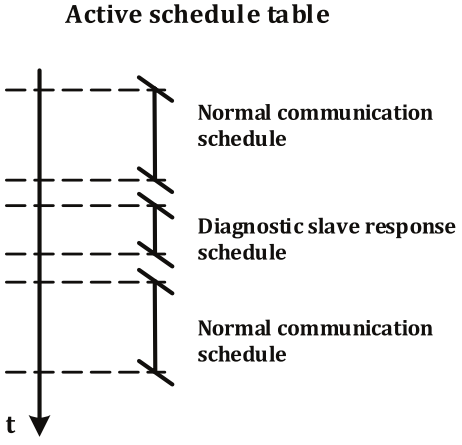



**Figure 21 — Interleaving of a diagnostic slave response schedule table**

### 9.6.4 Diagnostic schedule execution

#### 9.6.4.1 General

When no diagnostic communication is active, the master node shall not execute diagnostic schedules tables (see Figure 22). This shall be the default behaviour of the master node.

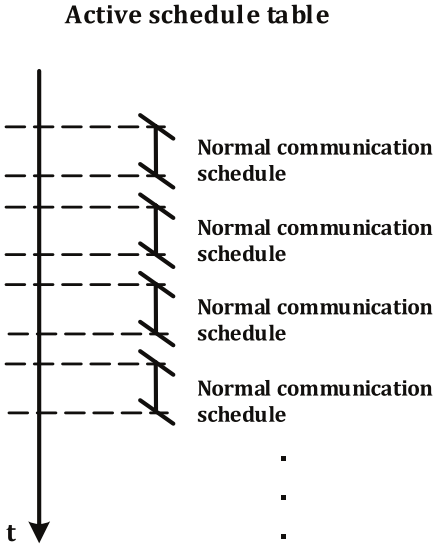



**Figure 22 — No diagnostic communication**

A master node supports the following different scheduling modes:

— interleaved diagnostics mode (mandatory);

— diagnostics only mode (optional).

The two modes are defined in greater detail in 9.6.4.2 and 9.6.4.3. The master node shall support to operate each of its connected LIN clusters in the one or the other mode upon request from an external diagnostic test tool.

### 9.6.4.2 Diagnostic interleaved mode

When diagnostic schedules need to be executed, the master node shall finish the currently running normal communication schedule with the last defined frame entry and then switch to the required diagnostic schedule to perform the transmission (see Figures 20 and 21). After execution of a diagnostic schedule, the master node starts again the previous normal communication table from its beginning or changes to another schedule table and frame index if another schedule table has been requested in the meantime. A new diagnostic schedule table is not executed until the end of the interleaved normal communication schedule table.

When using the diagnostic interleaved mode, it shall be ensured (via normal communication schedule design) that the time between two subsequent diagnostic schedules fulfils the vehicle manufacturer-specific diagnostic requirements.

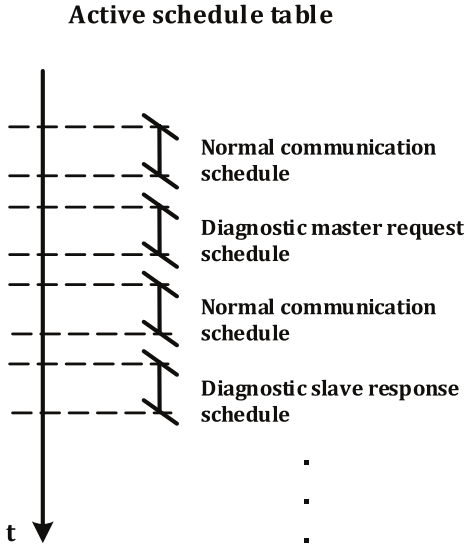



**Figure 23 — Normal diagnostic communication mode**

The number of executions of the diagnostic master request schedule table depends on the amount of data that needs to be transmitted and shall be determined by the master node considering the LIN transport protocol (e.g. two executions of the schedule for transmitting 10 user data bytes using the LIN transport protocol).

The subsequent interleaved execution of the diagnostic slave response schedule table depends on the amount of data to be transferred and shall therefore be performed by the master node until the transmission has been successfully finished or a transport protocol timeout occurs.

If a diagnostic transmission from a slave node to the master node has been started, the master node shall keep executing the diagnostic slave response schedule even when one or several slave response frame headers have not been answered (see Figure 24) until

— a $P2_{max}$/$P2^*_{max}$ timeout occurs (see 9.4), and

— a transport protocol timeout occurs (see Table 12).

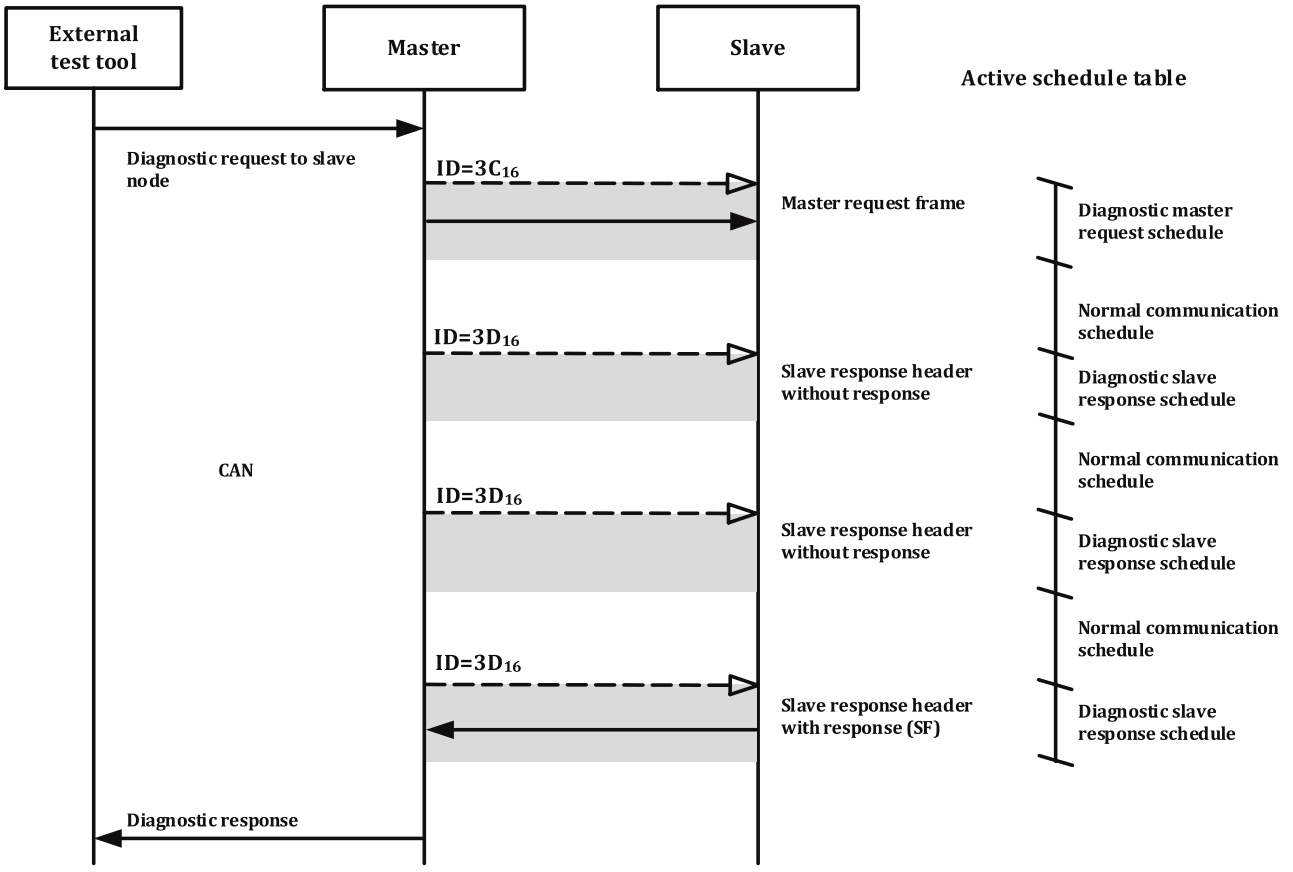
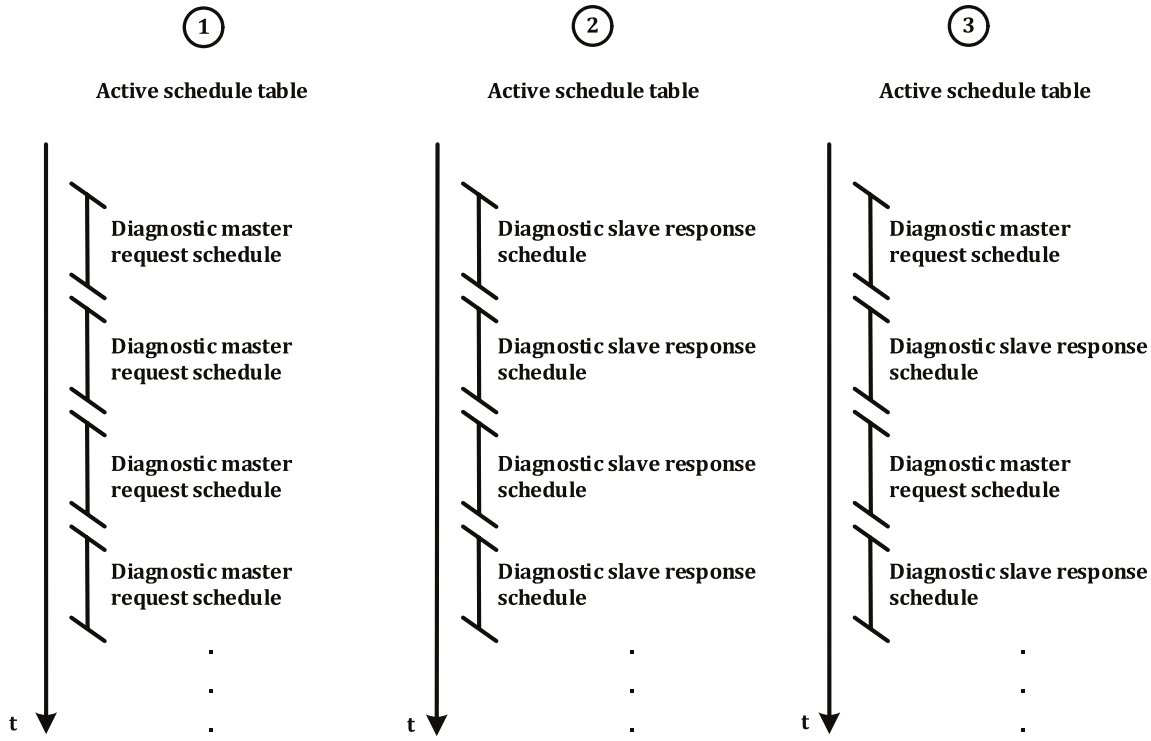



**Figure 24 — Continued execution of diagnostic slave response schedule table until response is received**

### 9.6.4.3 Diagnostic only mode

The master node may optionally implement a “diagnostics only mode” in which only the diagnostic schedules and no normal communication schedules are executed. The basic principles to use master request frame schedule tables and slave response frame schedule tables are the same as for the diagnostics interleaved mode except that no normal communication schedules are interleaved between the diagnostic schedule tables.

This is to allow for optimized diagnostic data transmission (e.g. when reading slave node identifications or during flash reprogramming, see Figure 25 for the different use cases).




**Key**

1 long transmission to diagnostic slave

2 long transmission from diagnostic slave

3 subsequent requests with responses from diagnostic slave nodes

**Figure 25 — Use cases for diagnostic only mode**

The diagnostic only mode shall be enabled and disabled via diagnostic service request from external test tool (e.g. service “communication control” in ISO 14229-7 to disable normal communication on the LIN cluster leads to the activation of the “diagnostic only mode”). When operating in the diagnostic only mode without any active transmission the master node shall execute diagnostic slave response schedule tables to prevent slave nodes to enter sleep mode (see Figure 26).

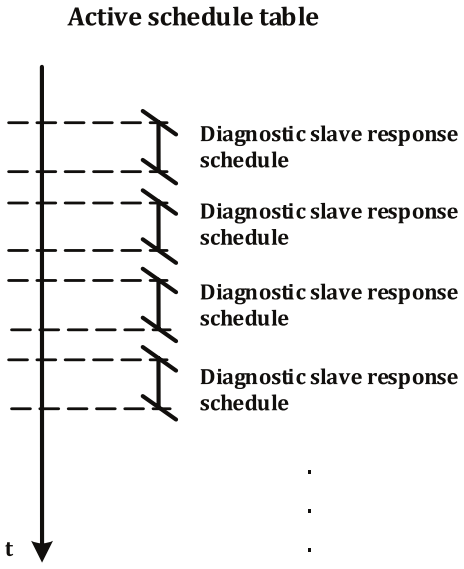



**Figure 26 — Default schedule in the diagnostic only mode**

## 9.7 Transmission handler requirements

### 9.7.1 General

For each LIN cluster, the master node implements one instance of the transmission handler specified in 9.7.2. The transmission handler shall be capable to operate in either interleaved diagnostics mode or diagnostics only mode.

At least one active master to slave node physical transmission plus one functional transmission can be handled per cluster.

Broadcasting to all LIN clusters of a master node shall always be possible regardless of the currently active connections.

Asynchronous responses from slave nodes without any prior request are prohibited. A slave node with boot loader capability may however confirm a programming request that has not been received by the transport layer/diagnostic layer in the same runtime context. When entering a programming session, the positive response is transmitted after the slave node has performed a physical reset to enter the boot loader.

### 9.7.2 Master node transmission handler

A transmission handler shall be implemented by the master node according to Figure 27. Scheduling is completely under the control of the application. The following states are defined for both modes:

— Idle:

  In this state, the master node is neither receiving nor transmitting any transmission on the LIN cluster. It is continuously available for any new transmission request.

— Tx functional active:

  In this state, the master node is routing a functional addressed request from the backbone bus to the LIN cluster. This can only be a SingleFrame (SF) according to restrictions for the transport protocol on LIN.

— Tx physical active:

In this state, the master node is currently routing data from the backbone bus to one slave node in the LIN cluster. The master node is consequently occupied and cannot route any other physical transmission from the backbone bus to the LIN cluster. Also, no response from a slave node can be routed to the backbone bus.

— Rx physical active:

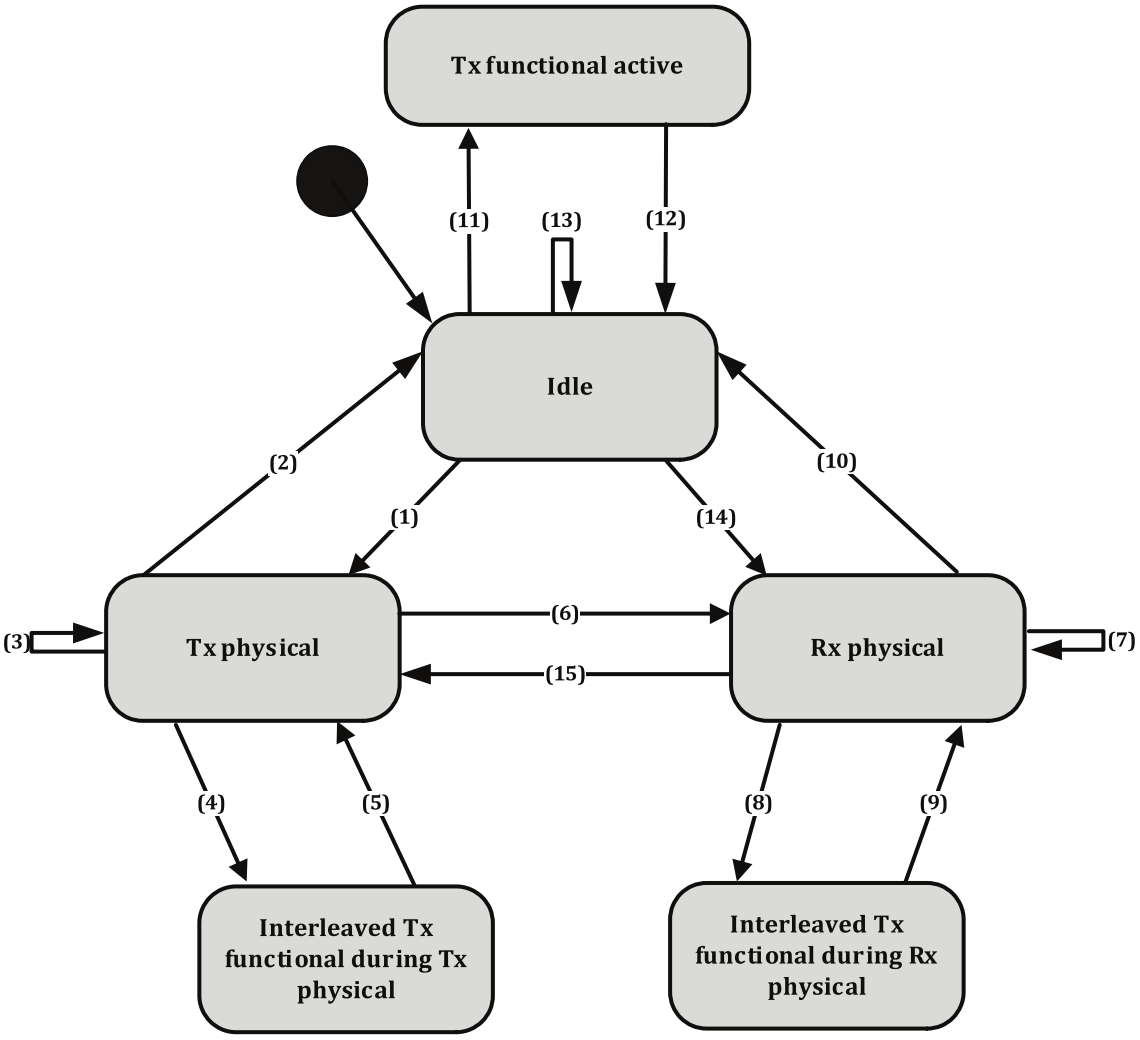
In this state, the master node is routing a transmission from a slave node to the backbone bus. It is possible to transmit functional addressed requests to the LIN cluster but cannot handle further physical transmissions to a slave node.

— Interleaved functional during Tx:

This is the state in which the master node routes a functional addressed request from the backbone bus to the LIN cluster, while a transmission to a slave node is currently active. Functional addressed SingleFrames (SF) can be transmitted, but shall be ignored by the active slave node while receiving a physically addressed transmission.

— Interleaved functional during Rx:

In this state, the master node routes a functional addressed request from the backbone bus to the cluster, while a reception from a slave node is currently active. Functional addressed SingleFrames (SF) can be transmitted, but shall be ignored by the active slave node while transmitting the physically addressed response.




**Key**

1 Idle state → Tx physical active state

— Trigger: Start of a physical transmission from the backbone bus to a slave node.

— Effect: Start executing diagnostic master request frame schedule tables and handle the transport protocol.

2 Tx physical active state → Idle state

— Condition: Routing of the physical transmission from the backbone bus to the cluster has finished or a transport protocol transmission error (e.g. timeout) on the backbone bus occurred.

— Action: Stop executing master request frame schedule tables.

3 Tx physical active state → Tx physical active state

— Condition: A new physical request is triggered while a previous one is operated.

— Action: A new transmission request is accepted and the previous pending physical request is discontinued.

4 Tx physical active state → Interleaved functional request during Tx physical state

— Condition: Functional addressed request from the backbone bus has been received.

— Action: Interrupt the physical transmission to the slave node and execute a single master request frame schedule to transmit the functional addressed request onto the cluster.

5 Interleaved functional during Tx physical state → Tx physical active state

— Condition: Functional addressed request has been routed onto the cluster.

— Action: Continue the interrupted physical transmission to the slave node.

6 Tx physical active state → Rx physical active state

— Condition: The physical transmission to the slave node has been successfully completed.

— Action: Stop executing master request frame schedule tables, start executing slave response frame schedule tables and handle the incoming response from the previously addressed slave node.

7 Rx physical active state → Rx physical active state

— Condition: The response from the slave node has not been started or has not finished yet or a response pending frame has been received.

— Action: Transmit slave response frame schedule tables and handle the LIN transport protocol (i.e. route transmission from the slave node to the backbone bus).

8 Rx physical active state → Interleaved functional request during Rx physical state

— Condition: Functional addressed request from the backbone bus has been received.

— Action: Interrupt the scheduling of a slave response frame schedule and execute a single master request frame schedule table to transmit the functional addressed request onto the cluster.

9 Interleaved functional during Rx physical state → Rx physical active state

— Condition: Functional addressed request has been routed onto the cluster.

— Action: Restart executing slave response frame schedules and continue the interrupted reception from the slave node.

10 Rx physical active state → Idle state

— Condition: Reception from the slave node has been completed or a transport protocol error on the backbone bus has occurred or the timeout P2 max respective P2* max has elapsed (according to the NRC $78_{16}$ handling as defined in ISO 14229-2).

— Action: In "diagnostic interleaved mode", stop executing slave response frame schedules and recover application schedule table. In "diagnostic only mode", slave response frame scheduling is continued until a new transmission is started from IDLE mode.

11 Idle state → Tx functional active state

— Condition: Functional addressed request from the backbone bus has been received.

— Action: Execute a single master request frame schedule table to transmit the functional addressed request onto the cluster.

12 Tx functional active state → Idle state

— Condition: Functional addressed request has been routed onto the cluster.

— Action: Stop executing master request frame schedules.

13 Idle state → Idle state

— Condition: Neither physical transmission from the backbone bus to be routed to a slave node nor any response to be routed from a slave node to the backbone bus.

— Action: Diagnostics interleaved mode. Do not execute any master request frame schedule tables or slave response frame schedule tables. Diagnostics only mode: execute slave response frame schedule tables.

14 Idle state → Rx physical active state (for "diagnostics only mode" only)

— Condition: A slave node is Broadcasting to all LIN clusters of a master node initiated a transmission via one of the slave response frame schedule tables.

— Action: Handle the incoming response from the responding slave node and start routing to the backbone bus.

15 Broadcasting to all LIN clusters of a master node or transmitting a new request while waiting for a response shall always be possible regardless of the currently active connections. Any pending physical reception is rejected when a new physical transmission is requested.

**Figure 27 — Master node transmission handler**

The master shall only accept a physical reception if the received NAD in the SF or FF matches the transmitted NAD of the previous physical request.

In case of an error condition during reception, another response with this NAD shall be ignored until renewed physical request.

Several valid physical responses with the same NAD shall be accepted to support response pending frames defined in UDSonLIN as specified in ISO 14229-1 and ISO 14229-2.

### 9.7.3 Slave node transmission handler

Each slave node shall implement a transmission handler as defined in Figure 28. This is to allow for diagnostic communication without frame collisions on the cluster. During diagnostics, the broadcast NAD is normally not used. If this happens, the slave node processes the requests with broadcast NAD ($7F_{16}$) in the same way as if it is the slave node's configured NAD.

The following states are defined.

— Idle:

  In this state, the slave node is neither receiving nor transmitting any messages in the cluster. It is accepting incoming request from the master node. It shall not respond to slave response headers.

— Receive physical request:

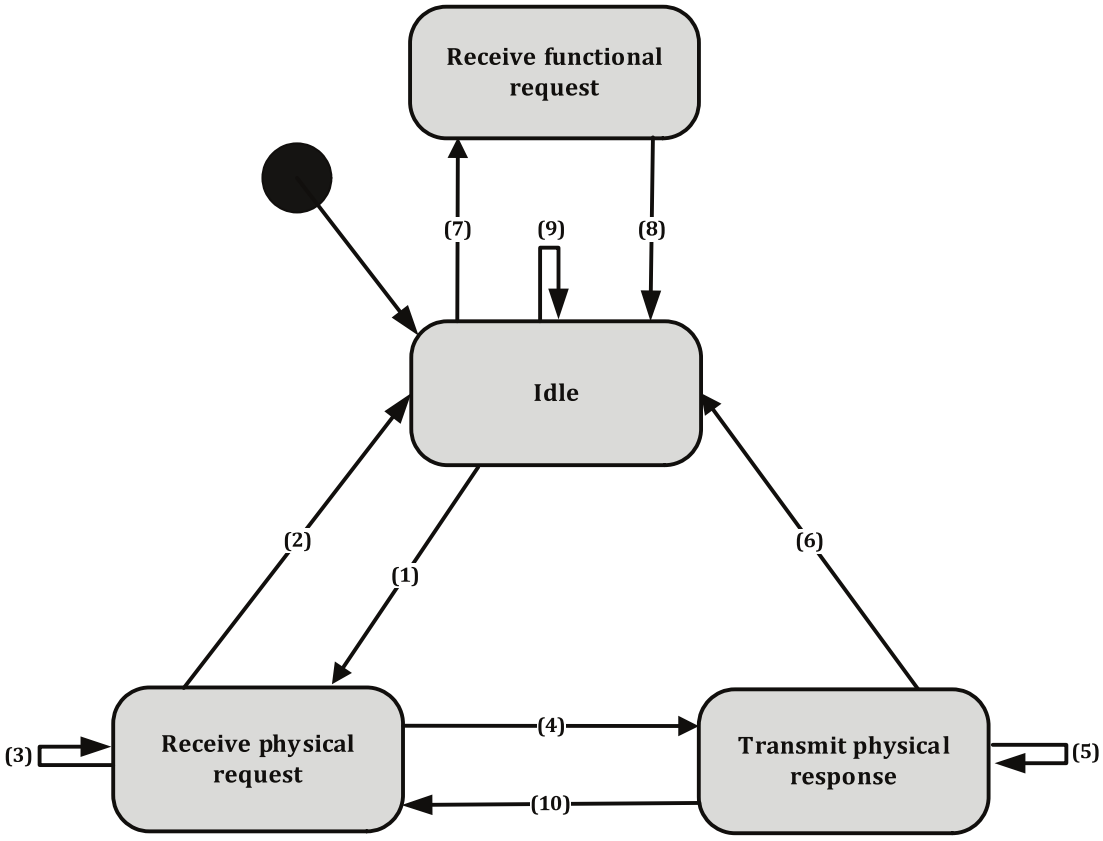
  In this state, the slave node is receiving and processing transport layer frames from the master node. The slave node shall ignore any interleaved functional addressed requests from the master node.

— Transmit physical response:

  In this state, a slave node is currently still processing the previously received request, is ready to transmit a physical response or is actually transmitting the response to the previously received request. A slave node shall ignore interleaved functional addressed (NAD $7E_{16}$) requests from the master node. New physical requests shall be received and make the slave node discard the current request or response. If the new request is addressed to the slave node, the request shall be processed.

— Receive functional request:

  In this state, a slave node is receiving a functional transmission from the master node. The slave node shall not respond to any slave response header.




**Key**

1 Idle state → Receive physical request state

— Condition: A master request frame has been received with the NAD matching the slave node's configured NAD.

— Action: Start processing the physical request according to the transport layer requirements.

2 Receive physical request state → Idle state

— Condition: A transport layer error has occurred or a master request frame with a NAD different from the slave node's configured NAD has been received or a physical request has been received completely where no response is expected.

— Action: Stop processing the physical request. Do not respond to slave response headers.

3 Receive physical request state → Receive physical request state

— Condition: The physical request has not been completely received yet and a new master request frame (CF) is received with the NAD set to the slave node's configured NAD. The physical request has not been completely received yet and a new physical request is received with the NAD set to the slave node's configured NAD. A functional addressed request shall be ignored.

— Action: Continue/restart the physical request.

4 Receive physical request state → Transmit physical response state

— Condition: The physical request has been completely received.

— Action: Process the diagnostic request. If a new physical request with the NAD set to the slave node's configured NAD is received while processing the previous request, the slave node shall discard the current request or response data and shall start receiving the new request.

5 Transmit physical response state → Transmit physical response state

— Condition: The physical response has not been completely transmitted yet. A functional addressed request shall be ignored.

— Action: Keep responding to slave response frames according to the transport layer requirements.

6 Transmit physical response state → Idle state

— Condition: The physical response has been completely transmitted, a LIN transport layer error occurred or a request with the NAD set to a different as the slave node's configured NAD has been received.

— Action: Discard the request and response data. Stop responding to slave response frames.

7 Idle state → Receive functional request state

— Condition: A master request frame with the NAD parameter set to the functional NAD has been received.

— Action: Receive and process the master request frame according to the transport layer. Do not respond to the slave response frame headers.

8 Receive functional request state → Idle state

— Condition: The functional request was processed.

— Action: Discard any response data. Stop responding to slave response frames.

9 Idle state → Idle state

— Condition: No request is received and no response is pending.

— Action: Do not respond to any slave response frames.

10 Transmit physical response state → Receive physical request state

— Condition: The previous response transmission has not been completely processed and a new physical diagnostic master request frame with the slave node's configured NAD is received.

— Action: Discard the response data. Start receiving and processing the physical request according to the LIN transport protocol requirements.

**Figure 28 — Slave node transmission handler**

## 9.8 Diagnostic service prioritization

In LIN networks MasterReq and SlaveResp frames are used for diagnostic communication where the Slave node is addressed using a configured NAD. Beside the session layer UDS diagnostic services, this document specifies also node configuration and identification services which are provided on data link layer and also proprietary services in the NAD range $80_{16}$ – $FF_{16}$ on session/application layer. As all of these three, "service instances" access the same frame resources MasterReq and SlaveResp and additionally LIN is limited to half-duplex communication conflicts are possible where an active service from one instance is interrupted by another instance request.

On network design level, those conflicts should be addressed by defining distinctive time lots for the usage of the different service instances, e.g. ISO 14229-7 UDSonLIN services should not be used if slave nodes are reconfigured. Proprietary services should not be used during normal live cycle of LIN clusters.

To resolve these conflicts on the data link layer in master and slave nodes, a priority scheme is provided meaning that a service with a higher priority interrupts a service with a lower priority.

a) Node configuration and identification services have the highest priority. Any other diagnostic communication is interrupted. Please note that using an AssignNAD service a new configured NAD is assigned to a slave node. This

b) ISO 14229-7 UDSonLIN diagnostics services have medium priority. Those are operated when no node configuration and identification service is active.

c) Proprietary services (NAD ≥ $80_{16}$) have low priority. Those are operated only if no other diagnostic communication is active.

# 10 LIN node capability language (NCL)

## 10.1 General

The intention of a node capability language is to be able to describe the possibilities of a slave node in a standardized, machine readable syntax.

The availability of premade off-the-shelf slave nodes is expected to grow in the next years. If they are all accompanied by a node capability file (NCF), it is possible to generate both the LIN description file

(LDF), see Clause 12 and initialization code (configuring the cluster, e.g. reconfigure conflicting frame identifiers) for the master node.

If the setup and configuration of any cluster is fully automatic, a great step towards plug-and-play development with LIN is taken. In other words, it is just as easy to use distributed nodes in a cluster as a single CPU node with the physical devices connected directly to the node.

## 10.2 Plug and play workflow concept

### 10.2.1 General

The LIN workflow concept allows for the implementation of a seamless chain of design and development tools and it enhances the speed of development and the reliability of the LIN cluster.

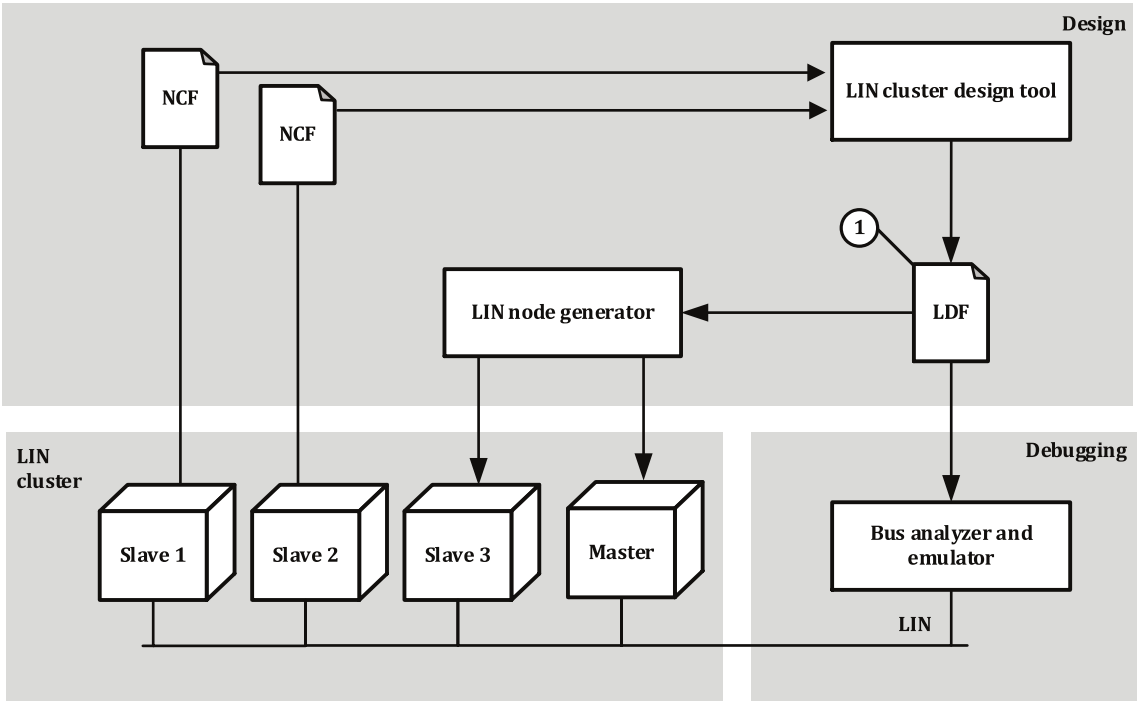
The LDF specification allows for safe sub-contracting of nodes without jeopardizing the LIN system functionality by, e.g. message incompatibility or network overload. It is also a powerful tool for debugging of a LIN cluster, including emulation of non-finished nodes.

The Node Capability Language Specification provides a standardized syntax for specification of off-the-shelves slave nodes. This simplifies procurement of standard slave nodes, as well as provides possibilities for tools that automate node generation. Thus, true Plug-and-Play with slave nodes in a cluster becomes reality.

The slave nodes are connected to the master node forming a LIN cluster. The corresponding node capability files are parsed by the LIN cluster design tool to generate a LIN description file (LDF) in the LIN cluster design process. The LDF is parsed by the LIN node generator to automatically generate LIN related functions in the desired nodes (the Master node and Slave 3 node in the example shown in Figure 29).

The LDF is also used by a LIN bus analyser/emulator tool to allow for cluster debugging.

Figure 29 shows the development of a cluster split in three areas; design, debugging and the LIN physical cluster. This specification focuses on the design phase.




**Key**

1 LIN description file

**Figure 29 — Development of a LIN cluster**

### 10.2.2 LIN node generation

The core description file of a LIN cluster is the LIN description file (LDF). Based on this file, it is possible to generate communication drivers of all nodes in the cluster, a process named LIN node generation. All signals, frames and in case of a master node also schedule tables are declared in this files.

### 10.2.3 LIN cluster design

The process of creating the LDF file is named LIN cluster design. When you design a completely new cluster, writing the LDF file (by hand or with computer aid) is an efficient way to define the communication of your cluster.

However, when you have existing slave nodes and want to create a cluster of them starting from scratch is not that convenient. This is especially true if the defined cluster contains slave node address conflicts or frame identifier conflicts.

By receiving a node capability file, NCF, with every existing slave node, the LIN cluster design step is automatic: Just add the NCF files to your project in the LIN cluster design tool and it produces the LDF file.

If you want to create new slave nodes as well, Figure 29, Slave 3, the process becomes somewhat more complicated. The steps to perform depend on the LIN cluster design tool being used, which is not part of the LIN specification. A useful tool allows entering additional information before generating the LDF file. (It is always possible to write a fictive NCF file for the non-existent slave node and thus, it is included.)

It is worth noticing that the generated LDF file reflects the configured network; any conflicts originally between slave nodes or frames shall have been resolved before activating the cluster traffic or by means of node configuration services ahead of application communication (initialization schedule table containing schedule table commands to resolve existing conflicts).

### 10.2.4 Debugging

Debugging and node emulation is based on the LDF file produced in the LIN cluster design.

Emulation of the master adds the requirement that the cluster shall be configured to be conflict free. Hence, the emulator tool shall be able to read reconfiguration data produced by the LIN cluster design tool.

## 11 Node capability file (NCF)

The NCF provides a capability definition for at least one slave node.

### 11.1 Overview of NCF syntax

The syntax is described using a modified BNF (Bachus-Naur Format), as summarized in Table 20.

**Table 20 — BNF syntax used for NCF**

| Symbol | Meaning |
|---|---|
| ::= | A name on the left of the ::= is expressed using the syntax on its right |
| <> | Used to mark objects specified later |
| \| | The vertical bar indicates choice. Either the left-hand side or the right hand side of the vertical bar shall appear |
| bold | The text in bold is reserved, either because it is a reserved word, or mandatory punctuation |
| [ ] | The text between the square brackets shall appear once or multiple times |
| ( ) | The text between the parenthesis are optional, i.e. shall appear once or zero times |
| `char_string` | Any character string enclosed in quotes “like this” |
| identifier | An identifier. Typically used to name objects. Identifiers shall follow the normal C rules for variable declaration |
| integer | An integer. Integers can be in decimal or hexadecimal ($_{16}$) format. |
| `real_or_integer` | A real or integer number. A real number is always in decimal and has an embedded decimal point. |

Within files using this syntax, comments are allowed anywhere. The comment syntax is the same as that for C++ where anything from // to the end of a line and anything enclosed in /* and */ delimiters shall be ignored.

The reserved text and identifiers are case sensitive.

### 11.2 Global structure definition

The syntax of a node capability file is defined as follows.

```
<node_capability_file_def>
<language_version_def>
(NCF_file_revision_def)
(LIN_sig_byte_order_big_endian_def)
[<node_definition>]
```

#### 11.2.1 Node capability file marker

```
<node_capability_file_def> ::= node_capability_file;
```

`node_capability_file` tag is used to declare the file as node capability file (NCF).

### 11.2.2 Language version number definition

```
<language_version_def> ::=
   LIN_language_version = char_string;
```

`LIN_language_version` shall be “ISO17987:2015”.

### 11.2.3 NCF revision

```
NCF_file_revision_def ::=
  NCF_file_revision = <ncf_revision>;
<ncf_revision> ::= "int_major.int_minor.int_sub";
```

This revision number ncf_revision is used to check consistency of slave nodes against a LIN cluster. Updates in the communication definition (new signal, changed signal mapping, ...) are represented by an increment in the unsigned 8-bit integer value(s) of `int_major, int_minor` or `int_sub`. The data link layer provides this revision version to the network via ReadByIdentifier $B2_{16}$ service and to the application. For more details, refer to ISO 17987-3:2016, 6.3.6.6.

### 11.2.4 Big-endian signal encoding variant

```
<LIN_sig_byte_order_big_endian_def> ::=
   LIN_sig_byte_order_big_endian;
```

If this optional tag is defined in the NCF, signals are mapped in big-endian order to the frames. See ISO 17987-3:2016, 5.2.2.6 for more information.

## 11.3 Node definition

```
<node_definition> ::=
  node <node_name>{
    <general_definition>
    <diagnostic_definition>
    <frame_definition>
    <encoding_definition>
    <status_management>
    (<free_text_definition>)
  }
<node_name> ::= identifier
```

If a node capability file contains more than one slave node, the `node_name` shall be unique within the file. The declared slave nodes shall be seen as classes (templates) for physical slave node instances.

The properties of a `node_definition` are defined in the following subclauses.

### 11.3.1 General node definition

```
<general_definition> ::=
  general {
    LIN_protocol_version = <protocol_version>;
    supplier = <supplier_id>;
    function = <function_id>;
    variant = <variant_id>;
    bitrate = <bitrate_definition>;
    sends_wake_up_signal = "yes" | "no";
  }
```

The `general_definition` declares the properties that specify the general compatibility with the cluster and also general node properties.

#### 11.3.1.1 LIN protocol version number definition

```
<protocol_version> ::= char_string
```

This specifies the protocol used by the slave node and shall be in the range of “ISO17987:2015”, “2.2”, “2.1”, “2.0”, “<SAE J2602 protocol version tag>”

NOTE LIN 1.3 is not in the list since NCF definitions start with LIN 2.0.

#### 11.3.1.2 LIN product identification

```
<supplier_id> ::= integer
<function_id> ::= integer
<variant_id>  ::= integer
```

The `supplier_id` is assigned to each supplier as a 16 bit number. The `function_id` is a 16 bit number assigned to the product by the supplier to make it unique. Finally, `variant_id` is an 8 bit value specifying the variant, see ISO 17987-3.

#### 11.3.1.3 Bit rate

```
<bitrate_definition> ::=
  automatic (min <bitrate>) (max <bitrate>) |
  select {<bitrate> [, <bitrate>]} |
  <bitrate>
```

Three kinds of `bitrate_definition` are possible:

— automatic,

the slave node can adopt to any legal bit rate used on the bus. If the words min and/or max is added, any bit rate starting from/up to the provided bit rate can be used.

— select,

the slave node can detect the bit rate if one of the listed bit rates are used, otherwise it fails.

— fixed,

only one bit rate can be used.

Manufacturers of standardized, off-the-shelf, slave nodes are encouraged to build automatic slave nodes since this gives the most flexibility to the cluster builder.

```
<bitrate> ::= real_or_integer kbps
```

The bit rates are specified in the range of 1 kbit/s to 20 kbit/s.

#### 11.3.1.4 Sends wake up signal

This parameter is set to “yes”, if the slave is able to transmit the wake up signal. Otherwise, it is set to “no”.

### 11.3.2 Diagnostic definition

```
<diagnostic_definition> ::=
  diagnostic {
    NAD = integer ([, integer]); | (integer to integer);
    diagnostic_class = integer;
    (P2min = real_or_integer ms;)
    (STmin = real_or_integer ms;)
    (N_As_timeout = real_or_integer ms;)
    (N_Cr_timeout = real_or_integer ms;)
    (support_sid { integer ([, integer]) };)
    (max_message_length = integer;)
}
```

The `diagnostic_definition` specifies the properties for transport layer and configuration.

The NAD property defines the initial node address; the value shall be set according to ISO 17987-3:2016, 6.3.4.2. Either a list of values or a range can be given. The range is inclusive, i.e. both values are included in the range. If more than one value is given, the slave shall dynamically select one of the values within the given NAD set based on a physical property.

The diagnostic class defines the supported class I, II or III.

— The default value of $P2_{min}$ is specified in ISO 14229-2 as parameter $P2_{Server}$.

— The default value of $ST_{min}$ is specified in 9.4.

— The default values of `N_As_timeout`, `N_Cs_timeout` and `N_Cr_timeout` are defined in Table 12.

Above timing parameters are only relevant for diagnostic class II and class III slave nodes.

The `max_message_length` property only applies to the diagnostic transport layer; it defines the maximum length of a diagnostic message. Default value is defined in 4095.

The `support_sid` lists all SID values (node configuration, identification and diagnostic services) that are supported by the slave node. Default values are mandatory node configuration and identification services $B2_{16}$ and $B7_{16}$ according to ISO 17987-3.

### 11.3.3 Frame definition

```
<frame_definition> ::=
  frames {
    [<single_frame>]
  }
```

The frames listed shall be all unconditional frames and event triggered frames processed by the slave node. Event triggered frames refer to the event triggered frame header, it therefore does not contain any signals. The diagnostic frames shall always be supported and therefore are not listed.

```
<single_frame> ::=
  <frame_kind> <frame_name> {
    <frame_properties>
    (<signal_definition>)
  }
<frame_kind> ::= publish | subscribe
<frame_name> ::= identifier
```

Each frame published or subscribed is declared as defined above. The `frame_name` is the symbolic name of the frame. The `frame_kind` is determined from the slave node point of view (e.g. a transmitted frame shall be a published frame).

#### 11.3.3.1 Frame properties

```
<frame_properties> ::=
  frame_length = integer;
  (min_period = integer ms;)
  (max_period = integer ms;)
  (event_triggered_frame = identifier;)
```

The `frame_length` is the length of a LIN frame (1 to 8).

The optional values for `min_period` and `max_period` are used to guide the tool in generation of the schedule table.

The `event_triggered_frame` refers to an event triggered frame, in case that the described frame is associated with it.

Several restrictions apply when a frame is also event triggered, see ISO 17987-3:2016, 5.2.4.3.

#### 11.3.3.2 Signal definition

```
<signal_definition> ::=
  signals {
    [<signal_name> { <signal_properties> }]
  }
<signal_name> ::= identifier
```

All frames (except diagnostic frames) carry signals, which are declared in according to the `signal_definition`.

```
<signal_properties> ::=
  <init_value>
  size = integer;
  offset = integer;
  (<encoding_name>;)
<init_value> ::= <init_value_scalar> | <init_value_array>
<init_value_scalar> ::= init_value = integer
<init_value_array> ::= init_value = {integer ([, integer ])}
```

The `init_value` specifies the value used for the signal from power on until first set by the publishing application. The `init_value_scalar` is used for scalar signals and the `init_value_array` is used for byte array signals. The `init_value_array` is given in big-endian order.

The `size` is the number of bits reserved for the signal and the `offset` specifies the position of the signal in the frame (LIN default signal encoding: number of bits in the frame response to the LSB of the signal, see ISO 17987-3: 2016, Figure 12, LIN big-endian signal encoding variant: number of bits in the frame response to the MSB of the signal, see ISO 17987-3:2016, Figure 13).

For a byte array, both size and offset shall be multiples of eight.

The only way to describe if a signal with size 8 or 16 is a byte array with one or two elements or a scalar signal is by analysing the `init_value`, i.e. the curly parenthesis are very important to distinguish between arrays and scalar values.

The `encoding_name` is a reference to an encoding defined in encoding clausal, see 11.3.4.

### 11.3.4 Signal encoding type definition

The encoding is intended for providing representation and scaling properties of signals.

```
<encoding_definition> ::=
  encoding {
    [<encoding_name> {
      [<logical_value> |
      <physical_range> |
      <bcd_value>      |
      <ascii_value>]
    }]
  }
<encoding_name>  ::= identifier
<logical_value>  ::= logical_value, <signal_value> (, <text_info>);
<physical_range> ::= physical_value, <min_value>, <max_value>, <scale>,
                     <offset> (, <text_info>);
<bcd_value>      ::= bcd_value;
<ascii_value>    ::= ascii_value;
<signal_value>   ::= integer
<min_value>      ::= integer
<max_value>      ::= integer
<scale>          ::= real_or_integer
<offset>         ::= real_or_integer
<text_info>      ::= char_string
```

The `signal_value` the `min_value` and the `max_value` shall be in range of 0 to 65 535.

The max_value shall be greater than or equal to min_value. If the raw value is within the range defined by the min and max value, the physical value shall be calculated as defined in Formula (1):

$$\text{physical\_value} = (\text{scale} * \text{raw\_value}) + \text{offset} \quad (1)$$

### 11.3.5 Status management

```
<status_management> ::=
  status_management {
    response_error = identifier;
    (fault_state_signals = identifier ([, identifier]);)
  }
<published_signal> ::= identifier
```

The status_management specifies which published signal the master node shall monitor to determine if the slave node is operating as expected.

The identifiers above refer each to one unique published signal in the signal definition, see 12.3.2. See the definition of the response_error signal in ISO 17987-3:2016, 5.5.4 and the fault state signals in ISO 14229-7:2015, 6.4.1.

Legacy LIN 2.0 slave nodes set the response_error signal also if the response of an unconditional frame is fully absent. This is in conjunction to later LIN standard revisions [LIN 2.1, LIN 2.2(A) and ISO 17987]. This may be considered in the network design.

### 11.3.6 Free text definition

```
<free_text_definition> ::=
  free_text {
    char_string
  }
```

The free_text_definition is used to bring up help text, limitations, etc., in the LIN cluster design tool, if desired.

Typical information provided in the free text definition is

— slave node purpose and physical world interaction, e.g. motor speed, power consumption etc., and

— deviations from the LIN standard.

## 11.4 NCF example

```
node_capability_file;
LIN_language_version = "ISO17987:2015";
NCF_file_revision = "1.07.04";
node step_motor {
  general {
    LIN_protocol_version = "ISO17987:2015";
    supplier = 0x0005; function = 0x0020; variant = 1;
    bitrate = automatic min 10 kbps max 20 kbps;
    sends_wake_up_signal = "yes";
  }
  diagnostic {
    NAD = 1 to 3;
    diagnostic_class = 2;
    P2_min = 100 ms;
    ST_min = 40 ms;
    support_sid { 0xB0, 0xB2, 0xB7 };
  }
  frames {
    publish node_status {
      length = 4; min_period = 10 ms; max_period = 100 ms;
      signals {
        state       {init_value = 0; size = 8; offset = 0;}
        fault_state {init_value = 0; size = 2; offset = 9; fault_enc;}
        error_bit   {init_value = 0; size = 1; offset = 8;}
```

```
        angle       {init_value = {0x22, 0x11}; size = 16; offset = 16;}
      }
    }
    subscribe control {
      length = 1; max_period = 100 ms;
      signals {
        command {init_value = 0; size = 8; offset = 0; position;}
      }
    }
  }
  encoding {
    position {physical_value, 0, 199, 1.8, 0, "deg";}
    fault_enc {logical_value, 0, "no result";
               logical_value, 1, "failed";
               logical_value, 2, "passed";}
  }
  status_management { response_error = error_bit;
                      fault_state_signals = fault_state; }
  free_text { "step_motor signal values outside 0 - 199 are ignored" }
}
```

# 12 LIN description file (LDF)

## 12.1 General

The language described in this document is used in order to create a LIN description file. The LIN description file describes a complete cluster and also contains all information necessary to monitor the cluster. This information is sufficient to make a limited emulation of one or multiple nodes if it/they are not available.

The LIN description file can be one component used in order to write software for an electronic control unit which shall be part of the cluster. An application program interface (API) is described in Reference [2], in order to have a uniform way to access the cluster from within different application programs. However, the functional behaviour of the application program is not addressed by the LIN description file.

The syntax of a LIN description file is simple enough to be entered manually, but the development and use of computer based tools is encouraged. Node capability files as described in 11 provide one way to (almost) automatically generate LIN description files. The same specification also gives an example of a possible workflow in the development of a cluster.

## 12.2 Overview of LDF syntax

The syntax is described using a modified BNF (Bachus-Naur Format), as summarized in Table 21.

**Table 21 — BNF syntax used in LDF**

| Symbol | Definition |
|---|---|
| ::= | A name on the left of the ::= is expressed using the syntax on its right |
| <> | Used to mark objects specified later |
| \| | The vertical bar indicates choice. Either the left-hand side or the right hand side of the vertical bar shall appear |
| bold | The text in bold is reserved - either because it is a reserved word, or mandatory punctuation |
| [ ] | The text between the square brackets shall appear once or multiple times |
| ( ) | The text between the parenthesis are optional, i.e. shall appear once or zero times |
| `char_string` | Any character string enclosed in quotes “like this” |

**Table 21** *(continued)*

| Symbol | Definition |
|---|---|
| identifier | An identifier. Typically used to name objects. Identifiers shall follow the normal C rules for variable declaration |
| integer | An integer. Integers can be in decimal or hexadecimal ($_{16}$) format. |
| `real_or_integer` | A real or integer number. A real number is always in decimal and has an embedded decimal point. |

Within files using this syntax, comments are allowed anywhere. The comment syntax is the same as that for C++ where anything from // to the end of a line and anything enclosed in /* and */ delimiters shall be ignored.

The reserved text and identifiers are case sensitive.

## 12.3 LDF definition

The syntax of the LDF is specified in 12.2.

### 12.3.1 Global structure definition

```
<LIN_description_file_def>
<LIN_protocol_version_def>
<LIN_language_version_def>
<LDF_file_revision_def>
<LIN_speed_def>
(<Channel_name_def>)
(<LIN_sig_byte_order_big_endian_def>)
<Node_def>
<Signal_def>
(<Diagnostic_signal_def>)
<Frame_def>
(<Sporadic_frame_def>)
(<Event_triggered_frame_def>)
(<Diagnostic_frame_def>)
<Node_attributes_def>
<Schedule_table_def>
(<Signal_encoding_type_def>)
(<Signal_representation_def>)
```

The overall syntax of a LIN description file shall be as above.

#### 12.3.1.1 LIN description file marker

```
<LIN_description_file_def> ::=  LIN_description_file;
```

`LIN_description_file` tag is used to declare the file as LIN description file (LDF).

#### 12.3.1.2 LIN protocol version number definition

```
<LIN_protocol_version_def> ::=
  LIN_protocol_version = char_string;
```

`LIN_protocol_version` defines the LIN master protocol version and shall be in the range of “ISO17987:2015”, “2.2”, “2.1”, “2.0”, “1.3”, “<J2602 protocol version tag>”. The master protocol version shall be equal or higher than any slave node defined in the cluster.

#### 12.3.1.3 LIN language version number definition

```
<LIN_language_version_def> ::=
  LIN_language_version = char_string;
```

`LIN_language_version` shall be “ISO17987:2015”.

#### 12.3.1.4 LIN file revision number

```
<LDF_file_revision_def> ::=
  LDF_file_revision = "int_major.int_minor.int_sub";
```

This revision number is used to check consistency of master and slave nodes against a LIN cluster. Updates in the communication definition (new signal, changed signal mapping, ...) are represented by an increment in the unsigned 8-bit integer value(s) of int_major, int_minor or int_sub. The data link layer provides this revision version to the network via ReadByIdentifier $B2_{16}$ service (slave nodes) and to the application (master and slave nodes). For more details, refer to ISO 17987-3:2016, 6.3.6.6.

#### 12.3.1.5 LIN speed definition

```
<LIN_speed_def> ::=
  LIN_speed = real_or_integer kbps;
```

This sets the nominal bit rate for the cluster. It shall be in the range of 1 kbit/s to 20 kbit/s.

#### 12.3.1.6 Channel postfix name definition

```
<Channel_name_def> ::=
  Channel_name = identifier;
```

Postfix for all named objects in the LDF. The postfix is mandatory for master nodes that are connected too more than one cluster. It is used to avoid naming collision if a node is connected to several clusters (i.e. using several LDFs). If given all named objects shall add this postfix to its name.

The postfix name shall be added with an underscore "_" to the named object.

Example: If signal name is "signal1" and `Channel_name` = "net1" in the LDF, then generated signal name is "`signal1_net1`".

#### 12.3.1.7 Big-endian signal encoding variant

```
<LIN_sig_byte_order_big_endian_def> ::=
  LIN_sig_byte_order_big_endian;
```

If this optional tag is defined in the LDF, signals are mapped in big-endian order to the frames. See ISO 17987-3:2016, 5.2.2.6 for more information.

### 12.3.2 Signal definition

#### 12.3.2.1 General

The signal definition subclauses identify the name of all signals in the cluster and their properties. The definitions in this subclause create a signal identifier set. All identifiers in this set shall be unique.

#### 12.3.2.2 Standard signals

```
<Signal_def> ::=
  Signals {
    [<signal_name>: <signal_size>, <init_value>, <published_by>[
                  ,<subscribed_by>];]
  }
<signal_name> ::= identifier
```

All `signal_name` identifiers shall be unique within the signal identifier set.

```
<signal_size> ::= integer
```

The `signal_size` specifies the size of the signal. It shall be in the range 1 bits to 16 bits for scalar signals and 8, 16, 24, 32, 40, 48, 56 or 64 for byte array signals.

```
<init_value>        ::= <init_value_scalar> | <init_value_array>
<init_value_scalar> ::= integer
<init_value_array>  ::= {integer ([, integer])}
```

The `init_value` specifies the signal value that shall be used by all subscriber nodes until the frame containing the signal is received. The `init_value_scalar` is used for scalar signals and the `init_`

value_array is used for byte array signals. The initial_value for byte arrays shall be arranged in big endian order (i.e. with the most significant byte first).

The only way to describe if a signal with size 8 or 16 is a byte array with one or two elements or a scalar signal is by analysing the init_value, i.e. the curly parenthesis are very important to distinguish between arrays and scalar values.

```
<published_by>  ::= identifier
<subscribed_by> ::= identifier
```
The published_by identifier and the subscribed_by identifier shall all exist in the node identifier set.

#### 12.3.2.3 Diagnostic signals

```
<Diagnostic_signal_def> ::=
  Diagnostic_signals {
    MasterReqB0: 8, 0;
    MasterReqB1: 8, 0;
    MasterReqB2: 8, 0;
    MasterReqB3: 8, 0;
    MasterReqB4: 8, 0;
    MasterReqB5: 8, 0;
    MasterReqB6: 8, 0;
    MasterReqB7: 8, 0;
    SlaveRespB0: 8, 0;
    SlaveRespB1: 8, 0;
    SlaveRespB2: 8, 0;
    SlaveRespB3: 8, 0;
    SlaveRespB4: 8, 0;
    SlaveRespB5: 8, 0;
    SlaveRespB6: 8, 0;
    SlaveRespB7: 8, 0;
  }
```
Diagnostic signals have a separate subclause in the LIN description file due to the fact that the publisher/subscriber information is predefined.

### 12.3.3 Frame definition

#### 12.3.3.1 General

The frame definition identifies the name of all frames in the cluster as well as their properties. The definitions create a frame identifier set (their symbolic name) and an associated frame ID set (the frame identifier). All members in these sets shall be unique.

#### 12.3.3.2 Unconditional frames

```
<Frame_def> ::=
  Frames {
    [<frame_name>: <frame_id>, <published_by>, <frame_size> {
      [<signal_name>, <signal_offset>;]
    }]
  }
<frame_name> ::= identifier
```
All frame_name identifiers shall be unique within the frame identifier set.

```
<frame_id> ::= integer
```
The frame_id specifies the frame identifier number in range 0 to 59. The frame identifier shall be unique for all frames within the frames identifier set.

```
<published_by> ::= identifier
```
The published_by identifier shall exist in the node identifier set.

```
<frame_size> ::= integer
```
The frame_size specifies the size of the frame in range 1 bytes to 8 bytes.

```
<signal_name> ::= identifier
```

The signal_name identifier shall exist in the signal identifier set.

All signals within one frame definition shall be published by the same node as specified in the published_by identifier for that frame.

<signal_offset> ::= integer
This value is in the range of 0 to (8 * frame_size - 1). signal_offset depends on the used signal encoding type, see 12.3.1.7.

LIN default signal encoding (little-endian) according to ISO 17987-3:2016, Figure 12: The signal_offset value specifies the position of the least significant bit of the signal in the frame. The least significant bit of the signal is transmitted first.

LIN big-endian signal encoding variant according to ISO 17987-3:2016, Figure 13: The signal_offset value specifies the position of the most significant bit of the signal in the frame.

EXAMPLE (LIN default signal encoding) Table 22 shows a ten bit signal mapped in a frame with a four byte data field. The LSB of signal S is at offset 16 (signal_offset) and the MSB is at offset 25.

**Table 22 — Packing of a signal**

| Byte0 | | | | | | | | Byte1 | | | | | | | | Byte2 | | | | | | | | Byte3 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | S | S | S | S | S | S | S | S | S | S | | | | | | |
| 0 | | | | | | | 7 | 8 | | | | | | | 15 | **16** | | | | | | | 23 | 24 | | | | | | | 31 |
| Transmitted first | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Transmitted last | |

### 12.3.3.3 Sporadic frames

```
<Sporadic_frame_def> ::=
  Sporadic_frames {
    [<sporadic_frame_name>: <frame_name> ([, <frame_name>]);]
  }
<sporadic_frame_name> ::= identifier
```
All sporadic_frame_name identifiers shall be unique within the frame identifier set.

<frame_name> ::= identifier
All frame_name identifiers shall exist in the frame identifier set and refer to unconditional frames. In the case that more than one of the declared frames needs to be transferred, the one first listed shall be chosen (prioritization). All frame_name identifiers shall be unconditional frames published by the master node. Furthermore, they shall not be scheduled as unconditional frames directly in the same schedule table as the sporadic_frame_name.

### 12.3.3.4 Event triggered frames

```
<Event_triggered_frame_def> ::=
  Event_triggered_frames {
    [<event_trig_frm_name>:
      <collision_resolving_schedule_table>,
      <frame_id>
      [, <frame_name>];]
  }
<event_trig_frm_name> ::= identifier
```
All event_trig_frm_name identifiers shall be unique within the frame identifier set.

<collision_resolving_schedule_table> ::= identifier
This refers to a schedule table in the schedule table set. This schedule shall automatically be activated after the collision. It shall minimum contain the associated unconditional frames.

<frame_id> ::= integer
The frame_id specifies the frame ID number in the range 0 to 59. The ID shall be unique for all frames within the frames ID set.

<frame_name> ::= identifier

All `frame_name` identifiers shall exist in the frame identifier set and refer to unconditional frames.

Remark

The first byte of the frame carries the protected identifier (PID) of the associated frame and, hence, cannot be used for other purposes.

#### 12.3.3.5 Diagnostic frames

```
< Diagnostic_frame_def> ::=
  Diagnostic_frames {
    MasterReq: 60 {
      MasterReqB0, 0;
      MasterReqB1, 8;
      MasterReqB2, 16;
      MasterReqB3, 24;
      MasterReqB4, 32;
      MasterReqB5, 40;
      MasterReqB6, 48;
      MasterReqB7, 56;
    }
    SlaveResp: 61 {
      SlaveRespB0, 0;
      SlaveRespB1, 8;
      SlaveRespB2, 16;
      SlaveRespB3, 24;
      SlaveRespB4, 32;
      SlaveRespB5, 40;
      SlaveRespB6, 48;
      SlaveRespB7, 56;
    }
  }
```

The MasterReq and SlaveResp reserved frame names are identifying the diagnostic frames (see ISO 17987-3:2016, 5.2.4.5) and shall be unique in the frame identifier set.

If the LIN big-endian signal encoding variant according to ISO 17987-3:2016, 5.2.2.6.2 is used the MSB of each signal is used for frame signal mapping:

```
< Diagnostic_frame_def> ::=
  Diagnostic_frames {
    MasterReq: 60 {
      MasterReqB0, 7;
      MasterReqB1, 15;
      …
      MasterReqB7, 63;
    }
    SlaveResp: 61 {
      SlaveRespB0, 7;
      SlaveRespB1, 15;
      …
      SlaveRespB7, 63;
    }
  }
```

### 12.3.4 Node definition

#### 12.3.4.1 General

The node definition subclauses identify the name of all participating nodes, as well as specifying time base and jitter for the master. The definitions in this subclause create a node identifier set. All identifiers in this set shall be unique.

#### 12.3.4.2 Participating nodes

```
<node_def> ::=
  Nodes {
    Master: <node_name>, <time_base> ms, <jitter> ms;
```

```
    Slaves: <node_name>([, <node_name>]);
  }
<node_name> ::= identifier
```

The nodes clause lists the physical nodes participating in the cluster. All `node_name` identifiers shall be unique within the node identifier set.

The `node_name` identifier after the master reserved word specifies the master node.

```
<time_base> ::= real_or_integer
```

The `time_base` value specifies the used time base in the master node to generate the maximum allowed frame transfer time. The time base shall be specified in milliseconds.

```
<jitter> ::= real_or_integer
```

The jitter shall be specified in milliseconds. For more information on `time_base` and jitter, see 12.3.4.

#### 12.3.4.3 Node attributes for ISO 17987:2015, LIN 2.2, LIN 2.1 slave nodes

Node attributes provides all necessary information on the behaviour of a single slave node.

```
<Node_attributes_def> ::=
  Node_attributes {
    [<node_name> {
      LIN_protocol = <protocol_version>;
      configured_NAD = <diag_address>;
      (initial_NAD = <diag_address>;)
      <attributes_def>;
    }]
  }
<node_name> ::= identifier
```

All `node_name` identifiers shall exist within the node identifier set and refer to a slave node.

```
<protocol_version> ::= char_string
```

Shall be in the range of “ISO17987:2015”, “2.2”, “2.1”.

```
<diag_address> ::= integer
```

The `diag_address` specifies the diagnostic address for the identified slave node in the range as defined in 3.1.2. It shall specify the unique NAD used for the slave node after resolving any cluster conflicts, i.e. it shall be unique within the cluster.

In case the `initial_NAD` is not given the `initial_NAD` is the same as the `configured_NAD`.

```
<attributes_def> ::=
  product_id = <supplier_id>, <function_id> (, <variant>);
  response_error = <signal_name>;
  (fault_state_signals = <signal_name>([, <signal_name>]);)
  (P2min = real_or_integer ms;)
  (STmin = real_or_integer ms;)
  (N_As_timeout = real_or_integer ms;)
  (N_Cr_timeout = real_or_integer ms;)
  <configurable_frames_def>
<supplier_id> ::= integer
<function_id> ::= integer
<variant>     ::= integer
```

The `supplier_id`, `function_id` and `variant_id` ranges are defined in 11.3.1.2.

The variant ID is optional since it is a property of the slave node and not the cluster.

```
<signal_name> ::= identifier
```

The `signal_name identifiers` for the `response_error` signal shall exist within the signal identifier set and refer to a one bit standard signal, see 12.3.2.2. The `response_error` signal shall be published by the specified slave node. For more information refer to status management in 11.3.5.

The `fault_state_signals` is a property of LIN slave nodes and are used for diagnostic class I and II, see ISO 14229-7:2015, 6.4.1.

The default value of $P2_{min}$ is 50 ms and $ST_{min}$ is 0 ms, see ISO 14229-7 as parameter $P2_{Server}$.

The default values of N_As_timeout and N_Cr_timeout are defined in Table 12.

Configurable frames shall list all frames (unconditional frames, event triggered frames and sporadic frames) processed by the slave node.

Definition of configurable_frames_def:

```
<configurable_frames_def> ::=
  configurable_frames {
     [<frame_name>;]
  }
```

The order of the frames are important since the node configuration request AssignFrameIdentifierRange dependent on the order, see ISO 17987-3:2016 6.3.6.5.

#### 12.3.4.4 Node attributes for LIN 2.0 slave nodes

Node attributes provides all necessary information on the behaviour of a single node.

```
<Node_attributes_def> ::=
Node_attributes {
  [<node_name> {
    LIN_protocol = <protocol_version>;
    configured_NAD = <diag_address>;
    (product_id = <supplier_id>, <function_id>, <variant>;)
    (response_error = <signal_name>;)
    (P2_min = real_or_integer ms;)
    (ST_min = real_or_integer ms;)
    (configurable_frames {
      [<frame_name> = <message_id>;]
    })
  }]
}

<node_name>        ::= identifier
<protocol_version> ::= "2.0"
<supplier_id>      ::= integer
<function_id>      ::= integer
<variant>          ::= integer
<message_id>       ::= integer
```

All node_name identifiers shall exist within the node identifier set and refer to a slave node. supplier_id shall be in the range $0_{16}$ .. $7FFE_{16}$, function_id in the range $0_{16}$ .. $FFFE_{16}$, variant in the range $0_{10}$ .. $255_{10}$ and message_id in the range $0_{16}$ ..$FFFE_{16}$.

```
<signal_name> ::= identifier
```

The signal_name identifiers for the response_error signal shall exist within the signal identifier set and refer to a one bit standard signal, see 12.3.2.2. The response_error signal shall be published by the specified slave node. For more information refer to status management in 11.3.5.

```
<diag_address> ::= integer
```

The diag_address specifies the diagnostic address for the identified node in the range of $1_{10}$ to $127_{10}$ as further defined in LIN Diagnostic and Configuration Specification. It shall specify the unique NAD used for the node after resolving any cluster conflicts, i.e. it shall be unique within the cluster.

configurable_frames shall list all frames processed by the node and their associated unique identifier message_id.

#### 12.3.4.5 Node attributes for LIN 1.3 slave nodes

Node attributes provides all necessary information on the behaviour of a single node.

```
<Node_attributes_def> ::=
Node_attributes {
  [<node_name> {
    LIN_protocol = <protocol_version>;
    configured_NAD = <diag_address>;
  }]
```

```
}
<node_name>        ::= identifier
<protocol_version> ::= "1.3"
```

#### 12.3.4.6 Node attributes for J2602 slave nodes

Node attributes provides all necessary information on the behaviour of a single node.

```
<Node_attributes_def> ::=
Node_attributes {
  [<node_name> {
    LIN_protocol = <protocol_version>;
    configured_NAD = <diag_address>;
    product_id = <supplier_id>, <function_id>, <variant>;
    response_error = <signal_name>;
    (P2_min = real_or_integer ms;)
    (ST_min = real_or_integer ms;)
    (configurable_frames {
      [<frame_name> = <message_id>;]
    })
    (response_tolerance = real_or_integer %;)
    (wakeup_time = integer ms;)
    (poweron_time = integer ms;)
  }]
}
<node_name>        ::= identifier
<protocol_version> ::= "J2602_1_1.0"
<supplier_id>      ::= integer
<function_id>      ::= integer
<variant>          ::= integer
<message_id>       ::= integer
```

All `node_name` identifiers shall exist within the node identifier set and refer to a slave node. `supplier_id` shall be in the range $0_{16}$ .. $7FFE_{16}$, `function_id` in the range $0_{16}$ .. $FFFE_{16}$, `variant` in the range $0_{10}$ .. $255_{10}$ and `message_id` in the range $0_{16}$ ..$FFFE_{16}$.

```
<signal_name> ::= identifier
```

The `signal_name` identifier shall exist within the signal identifier set and refer to a three bit signal. The signal shall be published by the specified node in each non-diagnostic transmit frame. Refer to the SAE J2602 standard for more information.

```
<diag_address> ::= integer
```

The `diag_address` specifies the diagnostic address for the identified node in the range of $1_{10}$ to $127_{10}$ as further defined in LIN Diagnostic and Configuration Specification. It shall specify the unique NAD used for the node after resolving any cluster conflicts, i.e. it shall be unique within the cluster.

`configurable_frames` shall list all frames processed by the node and their associated unique identifier `message_id`.

The `response_tolerance` specifies the node specific frame reception tolerance. Default is 40 %.

### 12.3.5 Schedule table definition

The schedule table describes the frames and the timing of the frames transmitted on the bus. Valid frames in the schedule tables are the frame types defined in see ISO 17987-3:2016, 5.2.4 (with the exception of the reserved frames) and the node configuration commands listed below.

```
<Schedule_table_def> ::=
  Schedule_tables {
    [<schedule_table_name> {
      [<command> delay <frame_time> ms;]
    }]
  }
<schedule_table_name> ::= identifier
```

All `schedule_table_name` identifiers shall be unique within the schedule table identifier set.

```
<command> ::=
  <frame_name> |
```

```
  MasterReq |
  SlaveResp |
  AssignNAD {<node_name>} |
  DataDump {<node_name>, <D1>, <D2>, <D3>, <D4>, <D5>} |
  SaveConfiguration {<node_name>} |
  AssignFrameId {<node_name>, <frame_name>} |
  AssignFrameIdRange {<node_name>, <frame_index> (, <frame_PID>, <frame_PID>,
                     <frame_PID>, <frame_PID>)} |
  FreeFormat {<D1>, <D2>, <D3>, <D4>, <D5>, <D6>, <D7>, <D8>}
```

The command specifies what shall be done in the frame slot. Providing a frame name transfers the specified frame.

`<frame_name> ::= identifier`
The `frame_name` identifier shall exist in the frame identifier set. If the `frame_name` refers to an event triggered frame or a sporadic frame, the associated unconditional frames may not be used in the same schedule table.

`<node_name> ::= identifier`
The `node_name` refers to one slave node, see 11.3.

`MasterReq` and `SlaveResp` are either defined as frames in 12.3.3.5 or, if this subclause is left out, automatically defined. The content of these frames is provided via the services and specified in ISO 17987-3:2016, Clause 6.

`AssignNAD` generates an AssignNAD request, see ISO 17987-3:2016, 6.3.4.2.

`DataDump` generates a DataDump request, see ISO 17987-3:2016, 6.3.6.3.

`SaveConfiguration` generates a SaveConfiguration request, see ISO 17987-3:2016, 6.3.6.4. `configured_NAD` parameter is taken from the node attributes section.

`AssignFrameId` generates an AssignFrameIdentifier request with a contents based on the parameters: NAD, `supplier_id` and `message_id` are taken from the node attributes of the `node_name`, and the `protected_id` is taken from the frame definition for `frame_name`, see 12.3.3.

`AssignFrameIdRange` generates an AssignFrameIdentifierRange request with the contents based on the parameters: NAD and the order of the frames (`frame_index`) are taken from the node attributes, see 12.3.4.3.

`<frame_index> ::= integer`
The `frame_index` sets the index to the first frame to assign a PID, see 12.3.4.3.

`<frame_PID> ::= integer`
If the optional four `frame_PID` are given the request include these values. If `frame_PID` are not given the PIDs for the four frames are taken from the frame definition for `frame_name`, see 12.3.3.

All data in this frame is fixed and determined during the processing of the LDF file. This service is only supported if the master node also supports this configuration service.

`FreeFormat` transmits a fixed master request frame with the eight data bytes provided. This may for instance be used to issue user specific fixed frames.

`<frame_time> ::= real_or_integer`
The `frame_time` specifies the duration of the frame slot, see ISO 17987-3:2016, 5.3.3. The `frame_time` value shall be specified in milliseconds.

The handling and switching of schedule table is controlled by the master application program, see description in I ISO 17987-3:2016, 5.3 and the schedule table handling API in Reference [2].

EXAMPLE Figure 30 shows a time line that corresponds to the schedule table VL1_ST1. It is assumed that the `time_base` (see 12.3.4.2) is set to 5 ms.

```
schedule_tables {
 VL1_ST1 {
 VL1_CEM_Frm1 delay 15 ms;
```

```
 VL1_LSM_Frm1 delay 15 ms;
 VL1_CPM_Frm1 delay 15 ms;
 VL1_CPM_Frm2 delay 20 ms;
 }
}
```

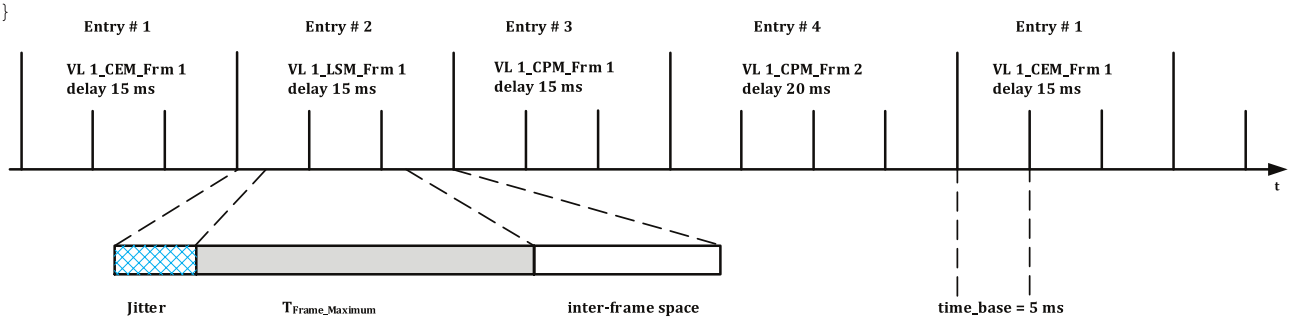



**Figure 30 — Time line for the VL1_ST1 schedule table**

The delay specified for every schedule entry shall be longer than the jitter and the worst case frame transfer time.

### 12.3.6 Signal encoding type definition

#### 12.3.6.1 General

The signal encoding type is intended for providing representation and scaling properties of signals. Although this information may be used to generate automatically scaling API routines in the node application, those API routines would require quite powerful nodes. The main purpose of the signal encoding type declarations is in bus traffic analysing tools, which can present the recorded traffic in an easily accessed way.

#### 12.3.6.2 ASCII (ASC)

ASCII data uses a one byte code to represent a text character. ASCII data is most often used where the consumer of the data is a display device which recognizes ASCII characters and can therefore display the data without further conversion. The least significant 7 bits represent the standard ASCII codes from 0 to 127. The most significant bit is reserved at this time but may be assigned a special function in the future. All ASCII signals shall have a length in bits which is a multiple of 8.

#### 12.3.6.3 Binary coded decimal (BCD)

Binary coded decimal (BCD) encoding is used when it is desirable to report decimal data in a nibble, and is often used where the data consumer is a display device. A BCD encoded signal shall be 4 bits long. Valid BCD data are the hex characters 0-9 with the encoding specified in Table 23.

**Table 23 — BCD to decimal value conversion**

| BCD value | Decimal value |
|---|---|
| $0_{16}$ | 0 |
| $1_{16}$ | 1 |
| $2_{16}$ | 2 |
| $3_{16}$ | 3 |
| $4_{16}$ | 4 |
| $5_{16}$ | 5 |
| $6_{16}$ | 6 |
| $7_{16}$ | 7 |
| $8_{16}$ | 8 |
| $9_{16}$ | 9 |
| $A_{16}$ to $F_{16}$ | invalid |

EXAMPLE $25_{16}$ would be interpreted as 37 decimal. As two BCD characters, the value is interpreted as 25 decimal.

#### 12.3.6.4 Signal encoding

```
<Signal_encoding_type_def> ::=
  Signal_encoding_types {
    [<signal_encoding_type_name> {
      [<logical_value> |
      <physical_range> |
      <bcd_value>      |
      <ascii_value>]
    }]
  }
<signal_encoding_type_name> ::= identifier
```

All `signal_encoding_type_name` identifier shall be unique within the signal encoding type identifier set.

```
<logical_value>  ::= logical_value, <signal_value> (, <text_info>);
<physical_range> ::= physical_value, <min_value>, <max_value>, <scale>,
                       <offset> (, <text_info>);
<bcd_value>      ::= bcd_value;
<ascii_value>    ::= ascii_value;
<signal_value>   ::= integer
<min_value>      ::= integer
<max_value>      ::= integer
<scale>          ::= real_or_integer
<offset>         ::= real_or_integer
<text_info>      ::= char_string
```

The `signal_value` the `min_value` and the `max_value` shall be in range of 0 to 65 535. The `max_value` shall be greater than or equal to `min_value`. If the raw value is within the range defined by the min and max value, the physical value shall be calculated as defined in Formula (1).

EXAMPLE The V_battery signal is an eight bit representation that follows the graph Figure 31, i.e. the resolution is high around 12 V and has three special values for out of range values.

```
signal_encoding_types {
  power_state {
    logical_value, 0, "off";
    logical_value, 1, "on";
  }
  V_battery {
    logical_value, 0, "under voltage";
```

```
    physical_value, 1, 63, 0.0625, 7.0, "Volt";
    physical_value, 64, 191, 0.0104, 11.0, "Volt";
    physical_value, 192, 253, 0.0625, 1.3, "Volt";
    logical_value, 254, "over voltage";
    logical_value, 255, "invalid";
  }
}
```

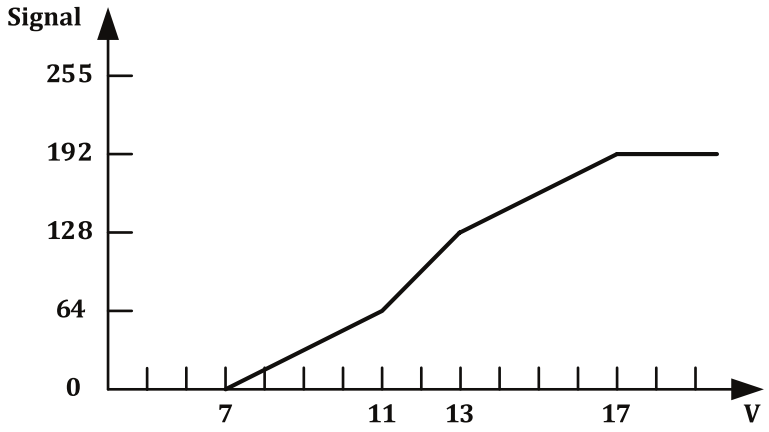



**Figure 31 — Representation of V_battery**

### 12.3.7 Signal representation definition

The signal representation declaration is used to associate signals with the corresponding signal encoding type.

```
<Signal_representation_def> ::=
  Signal_representation {
    [<signal_encoding_type_name>: <signal_name> ([, <signal_name>]);]
  }
<signal_encoding_type_name> ::= identifier
```

The `signal_encoding_type_name` identifier shall exist in the signal encoding type identifier set.

```
<signal_name> ::= identifier
```

The `signal_name` identifier shall exist in the signal identifier set (both scalar and byte array signals are applicable). Each signal may only be associated with one `signal_encoding_type_name` and may not be nested in a `signal_group_name`.

## 12.4 LDF example

```
LIN_description_file;
LIN_protocol_version = "ISO17987:2015";
LIN_language_version = "ISO17987:2015";
LDF_file_revision = "14.23.01";
LIN_speed = 19.2 kbps;
Channel_name = "DB";
Nodes {
  Master: CEM, 5 ms, 0.1 ms;
  Slaves: LSM, RSM;
}
Signals {
  InternalLightsRequest: 2, 0, CEM, LSM, RSM;
  RightIntLightsSwitch: 8, 0, RSM, CEM;
  LeftIntLightsSwitch: 8, 0, LSM, CEM;
  LSMerror: 1, 0, LSM, CEM;
  RSMerror: 1, 0, RSM, CEM;
  IntTest: 2, 0, LSM, CEM;
}
```

```
Frames {
  CEM_Frm1: 0x01, CEM, 1 {
    InternalLightsRequest, 0;
  }
  LSM_Frm1: 0x02, LSM, 2 {
    LeftIntLightsSwitch, 8;
  }
  LSM_Frm2: 0x03, LSM, 1 {
    LSMerror, 0;
    IntTest, 1;
  }
  RSM_Frm1: 0x04, RSM, 2 {
    RightIntLightsSwitch, 8;
  }
  RSM_Frm2: 0x05, RSM, 1 {
    RSMerror, 0;
  }
}
Event_triggered_frames {
  Node_Status_Event : Collision_resolver, 0x06, RSM_Frm1, LSM_Frm1;
}
Node_attributes {
  RSM {
    LIN_protocol = "2.1";
    configured_NAD = 0x20;
    product_id = 0x4E4E, 0x4553;
    response_error = RSMerror;
    P2_min = 150 ms;
    ST_min = 50 ms;
    configurable_frames {
      Node_Status_Event;
      CEM_Frm1;
      RSM_Frm1;
      RSM_Frm2;
    }
  }
  LSM {
    LIN_protocol = "ISO17987:2015";
    configured_NAD = 0x21;
    initial_NAD = 0x01;
    product_id = 0x4A4F, 0x4841;
    response_error = LSMerror;
    fault_state_signals = IntTest;
    P2_min = 150 ms;
    ST_min = 50 ms;
    configurable_frames {
      Node_Status_Event;
      CEM_Frm1;
      LSM_Frm1;
      LSM_Frm2;
    }
  }
}
Schedule_tables {
  Configuration_Schedule {
    AssignNAD {LSM} delay 15 ms;
    AssignFrameIdRange {LSM, 0} delay 15 ms;
    AssignFrameIdRange {RSM, 0} delay 15 ms;
    SaveConfiguration {LSM} delay 10 ms;
    SaveConfiguration {RSM} delay 10 ms;
  }
  Normal_Schedule {
    CEM_Frm1 delay 15 ms;
    LSM_Frm2 delay 15 ms;
    RSM_Frm2 delay 15 ms;
    Node_Status_Event delay 10 ms;
  }
  MRF_schedule {
    MasterReq delay 10 ms;
  }
  SRF_schedule {
```

```
    SlaveResp delay 10 ms;
  }
  Collision_resolver { // Keep timing of other frames if collision
    CEM_Frm1 delay 15 ms;
    LSM_Frm2 delay 15 ms;
    RSM_Frm2 delay 15 ms;
    RSM_Frm1 delay 10 ms;  // Poll the RSM node
    CEM_Frm1 delay 15 ms;
    LSM_Frm2 delay 15 ms;
    RSM_Frm2 delay 15 ms;
    LSM_Frm1 delay 10 ms;  // Poll the LSM node
  }
}
Signal_encoding_types {
  Dig2Bit {
    logical_value, 0, "off";
    logical_value, 1, "on";
    logical_value, 2, "error";
    logical_value, 3, "void";
  }
  ErrorEncoding {
    logical_value, 0, "OK";
    logical_value, 1, "error";
  }
  FaultStateEncoding {
    logical_value, 0, "No test result";
    logical_value, 1, "failed";
    logical_value, 2, "passed";
    logical_value, 3, "not used";
  }
  LightEncoding {
    logical_value, 0, "Off";
    physical_value, 1, 254, 1, 100, "lux";
    logical_value, 255, "error";
  }
}
Signal_representation {
  Dig2Bit: InternalLightsRequest;
  ErrorEncoding: RSMerror, LSMerror;
  FaultStateEncoding: IntTest;
  LightEncoding: RightIntLightsSwitch, LeftIntLightsSwitch;
}
```

# Bibliography

[1] ISO 17987-4, *Road vehicles — Local Interconnect Network (LIN) — Part 4: Electrical Physical Layer (EPL) specification (12V/24V)*

[2] ISO/TR 17987-5, *Road vehicles — Local Interconnect Network (LIN) — Part 5: Application Programmers Interface (API)*

[3] ISO 17987-6, *Road vehicles — Local Interconnect Network (LIN) — Part 6: Protocol conformance test specification*

[4] ISO 17987-7, *Road vehicles — Local Interconnect Network (LIN) — Part 7: Electrical Physical Layer (EPL) conformance test specification*

[5] ISO/IEC 7498-1, *Information processing systems — Open Systems Interconnection — Basic Reference Model: The Basic Model — Part 1*

[6] ISO/IEC 10731, *Information technology — Open Systems Interconnection — Basic Reference Model — Conventions for the definition of OSI services*

**ICS 35.240.60; 43.040.15**

Price based on 72 pages