INTERNATIONAL STANDARD

ISO 14230-2

Second edition
2013-03-15

Road vehicles — Diagnostic communication over K-Line (DoK-Line) —

Part 2: Data link layer

Véhicules routiers — Communication de diagnostic sur la ligne K (DoK-Line) —

Partie 2: Couche de liaison de données

Reference number
ISO 14230-2:2013(E)

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

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.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of technical committees is to prepare International Standards. Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75 % of the member bodies casting a vote.

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.

ISO 14230-2 was prepared by Technical Committee ISO/TC 22, *Road vehicles*, Subcommittee SC 3, *Electrical and electronic equipment*.

This second edition cancels and replaces the first edition (ISO 14230-2:1999), which has been technically revised.

ISO 14230 consists of the following parts, under the general title *Road vehicles — Diagnostic communication over K-Line (DoK-Line)*:

— *Part 1: Physical layer*

— *Part 2: Data link layer*

— *Part 3: Application layer*

— *Part 4: Requirements for emission-related systems*

Introduction

This part of ISO 14230 has been established in order to define common requirements for vehicle diagnostic systems implemented on K-Line (UART-based) communication link, as specified in ISO 14230-1.

To achieve this, it is based on the Open Systems Interconnection (OSI) Basic Reference Model in accordance with ISO/IEC 7498-1:1994 and ISO/IEC 10731, which structures communication systems into seven layers. When mapped on this model, the services specified by ISO 14230 are broken into:

— Diagnostic services (layer 7), specified in ISO 14229-1, ISO 14229-6,

— Presentation layer (layer 6),

  — vehicle manufacturer specific,

  — legislated WWH-OBD: specified in ISO 27145-2, SAE J1930-DA, SAE J1979-DA, SAE J2012-DA, SAE J1939, Companion Spreadsheet (SPNs), SAE J1939-73:2010, Appendix A (FMIs),

— Session layer services (layer 5),

  — legislated OBD: specified in ISO 14229-2,

  — legislated WWH-OBD: specified in ISO 14229-2,

— Transport layer services (layer 4), specified in ISO 14230-2,

— Network layer services (layer 3), specified in ISO 14230-2,

— Data link layer (layer 2), specified in ISO 14230-4, ISO 14230-2,

— Physical layer (layer 1), specified in ISO 14230-1.

This breakdown is shown in Table 1.

Table 1 — Enhanced and legislated OBD diagnostic specifications applicable to the OSI layers

<table>
<tr><th>Applicability</th><th>OSI seven layer</th><th>Enhanced diagnostics</th><th colspan="2">Legislated OBD (On-Board Diagnostics)</th><th colspan="2">Legislated WWH-OBD (On-Board Diagnostics)</th></tr>
<tr><td rowspan="7">Seven layer according to ISO/IEC 7498-1 and ISO/IEC 10731</td><td>Application (layer 7)</td><td>ISO 14229-1, ISO 14229-6</td><td colspan="2">ISO 15031-5</td><td colspan="2">ISO 14229-1, ISO 27145-3</td></tr>
<tr><td>Presentation (layer 6)</td><td>vehicle manufacturer specific</td><td colspan="2">ISO 15031-2, ISO 15031-5, ISO 15031-6, SAE J1930-DA, SAE J1979-DA, SAE J2012-DA</td><td colspan="2">ISO 27145-2, SAE 1930-DA, SAE J1939 Companion Spread-sheet (SPNs), SAE J1939-73:2010, Appendix A (FMIs), SAE J1979-DA, SAE J2012-DA,</td></tr>
<tr><td>Session (layer 5)</td><td colspan="5">ISO 14229-2</td></tr>
<tr><td>Transport (layer 4)</td><td rowspan="2">ISO 14230-2</td><td rowspan="2">ISO 15765-2</td><td rowspan="4">ISO 15765-4</td><td rowspan="2">ISO 15765-4, ISO 15765-2</td><td rowspan="4">ISO 27145-4</td></tr>
<tr><td>Network (layer 3)</td></tr>
<tr><td>Data link (layer 2)</td><td>ISO 14230-2</td><td rowspan="2">ISO 11898-1, ISO 11898-2</td><td rowspan="2">ISO 15765-4, ISO 11898-1, ISO 11898-2</td></tr>
<tr><td>Physical (layer 1)</td><td>ISO 14230-1</td></tr>
</table>

The application layer services covered by ISO 14229-6 have been defined in compliance with diagnostic services established in ISO 14229-1 and ISO 15031-5, but are not limited to use only with them.

ISO 14229-6 is also compatible with most diagnostic services defined in national standards or vehicle manufacturer's specifications.

Road vehicles — Diagnostic communication over K-Line (DoK-Line) —

Part 2: Data link layer

1 Scope

This part of ISO 14230 specifies data link layer services tailored to meet the requirements of UART-based vehicle communication systems on K-Line as specified in ISO 14230-1. It has been defined in accordance with the diagnostic services established in ISO 14229-1 and ISO 15031-5, but is not limited to use with them, and is also compatible with most other communication needs for in-vehicle networks. The protocol specifies an unconfirmed communication.

The diagnostic communication over K-Line (DoK-Line) protocol supports the standardized service primitive interface as specified in ISO 14229-2.

This part of ISO 14230 provides the data link layer services to support different application layer implementations like:

— enhanced vehicle diagnostics (emissions-related system diagnostics beyond legislated functionality, non-emissions-related system diagnostics);

— emissions-related OBD as specified in ISO 15031, SAE J1979-DA, and SAE J2012-DA.

In addition, this part of ISO 14230 clarifies the differences in initialization for K-line protocols defined in ISO 9141 and ISO 14230. This is important since a server supports only one of the protocols mentioned above and the client has to handle the coexistence of all protocols during the protocol-determination procedure.

2 Normative references

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO 14230-1, *Road vehicles — Diagnostic communication over K-Line (DoK-Line) — Part 1: Physical layer*

ISO 14230-4, *Road vehicles — Diagnostic systems — Keyword Protocol 2000 — Part 4: Requirements for emission-related systems*

3 Terms, definitions, symbols, and abbreviated terms

3.1 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

3.1.1
5-baud initialization
5-BAUD_INIT
starts with bus idle and ends with inverted address byte sent by the server

3.1.2
fast initialization
FAST_INIT
starts with bus idle and ends with the reception of all positive responses of the StartCommunication service from all addressed servers

3.1.3
topology
serial link between client and servers that consists of a K-Line and an optional L-Line

3.1.4
server
function that is part of an electronic control unit and that provides the diagnostic services

3.1.5
client
function that is part of the tester and that makes use of the diagnostic services

Note 1 to entry: A tester normally makes use of other functions such as data base management, specific interpretation, human-machine interface.

3.2 Abbreviated terms

5-BAUD_INIT	5-baud initialization
ISO 9141-2, 5-BAUD_INIT	protocol on K-Line according to ISO 9141-2 including 5-BAUD_INIT
ISO 14230-2, 5-BAUD_INIT	protocol on K-Line according to ISO 14230-2 including 5-BAUD_INIT
ISO 14230-2 FAST_INIT	protocol on K-Line according to ISO 14230-2 including FAST_INIT
ISO 14230-4, 5-BAUD_INIT	protocol on K-Line according to ISO 14230-4 including 5-BAUD_INIT
ISO 14230-4 FAST_INIT	protocol on K-Line according to ISO 14230-4 including FAST_INIT
bus converter	electronic control unit that links bus systems
client	external test equipment
confirm	confirmation service primitive
Cvt	M = mandatory, C = conditional, U = user-optional
ECU	electronic control unit
FAST_INIT	fast initialization
FB	first byte
FMT	format byte
gateway	linking hardware between bus systems
DA	destination address
DoK-Line	diagnostic communication over K-Line
DoK-Line_SA	data link source address
DoK-Line_TA	data link target address

DoK-Line_TAtype	data link target address type
indication	indication service primitive
LEN	length byte
Mtype	message type
request	request service primitive
DL_Data	data link data
DoK-Line_PCI	data link protocol control information
DoK-Line_PCItype	data link protocol control information type
DoK-Line_PDU	data link protocol data unit
DoK-Line_SA	data link source address
DoK-Line_SDU	data link service data unit
$P1_{Receiver}$	inter-byte timing parameter of the server
$P2_{Server}$	time between client request and server response or two server responses
$P3_{Client}$	time between end of server responses and start of new client request
$P4_{Sender}$	inter-byte timing parameter of the client
SA	source address
server	electronic control unit (ECU)
TA	target address
UART	universal asynchronous receiver and transmitter
WuP	wake-up pattern

4 Conventions

This part of ISO 14230 is based on the conventions discussed in the OSI Service Conventions (ISO/IEC 10731:1994) as they apply for diagnostic services.

These conventions specify the interactions between the service user and the service provider. Information is passed between the service user and the service provider by service primitives, which may convey parameters.

Figure 1 summarizes the distinction between service and protocol.

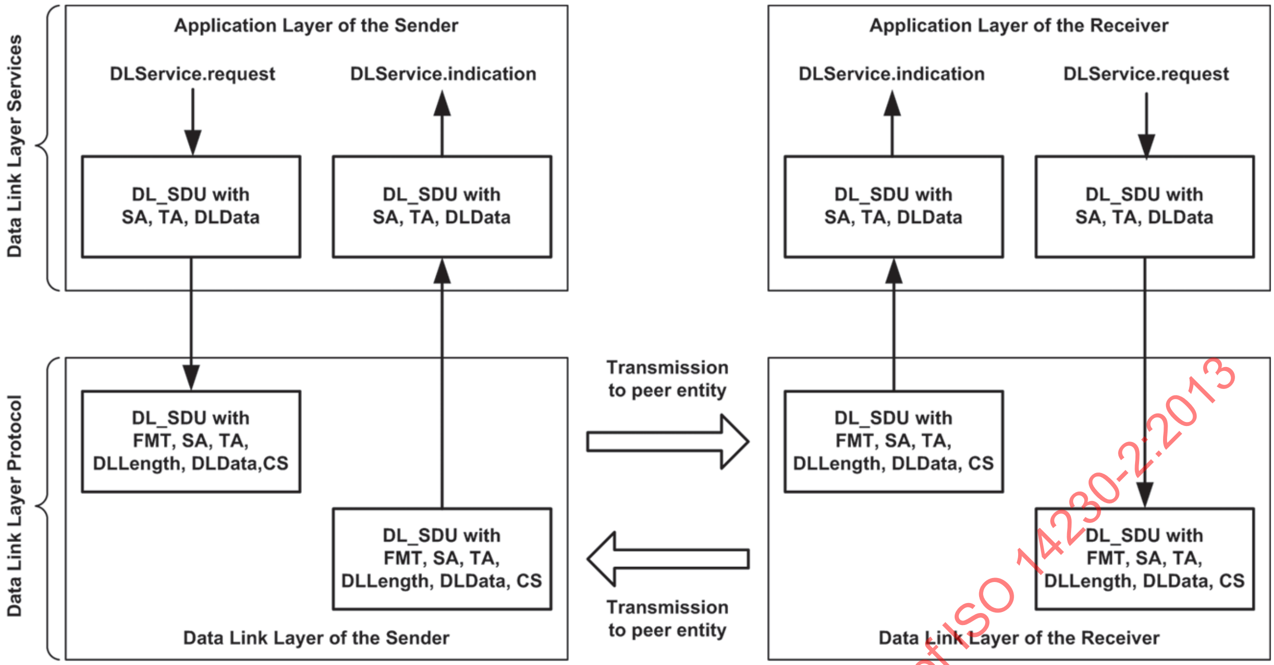


Figure 1 — The services and the protocol

NOTE The figure above does not show the confirmation generated on the transmitter side of the message.

ISO 14230-2 defines confirmed services. The confirmed services use the three service primitives: request, indication, and confirmation.

For all services defined in ISO 14230-2, the request and indication service primitives always have the same format and parameters.

5 Document overview

Figure 2 shows the diagnostic communication over K-Line document reference according to OSI model.

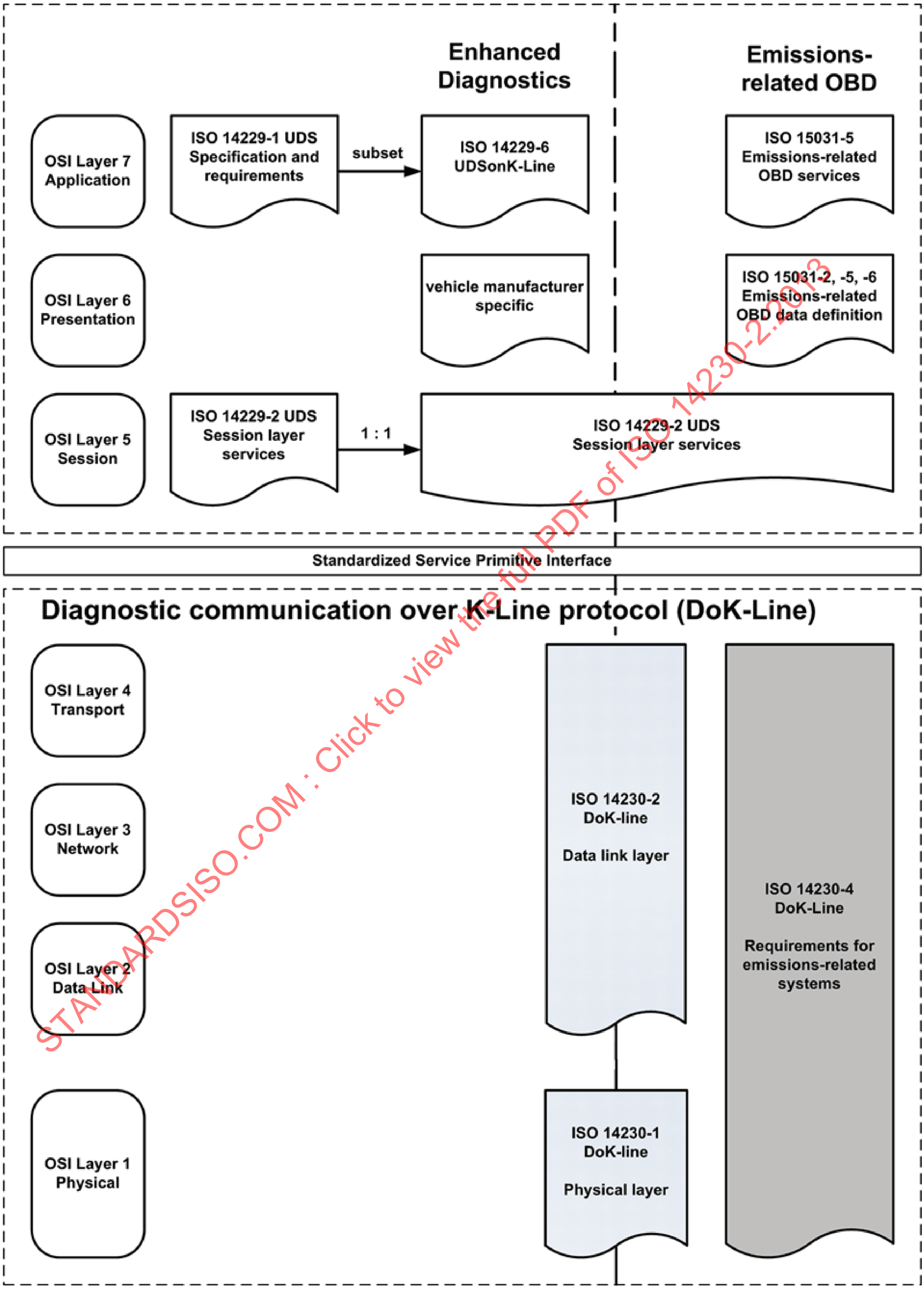
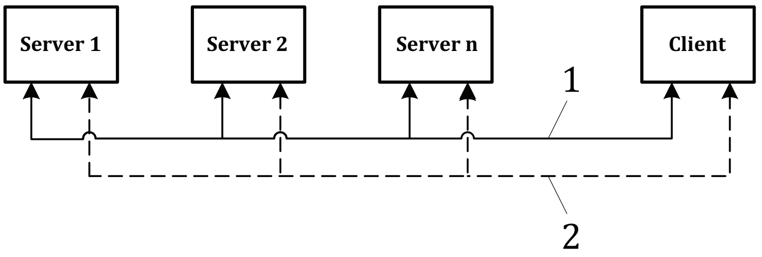


Figure 2 — DoK-Line document reference according to OSI model

6 Physical bus topology

DoK-Line is a bus concept based on a serial link consisting of one or two physical lines.

Figure 3 shows the server and client topology.



Key

1 K-Line

2 L-Line (optional)

Figure 3 — Server and client topology

"K-Line" is used for communication and initialization, whereas "L-Line" (optional) is used for initialization only. Special cases are node-to-node connection that means only one server (ECU) is on the line, which also can be a bus converter.

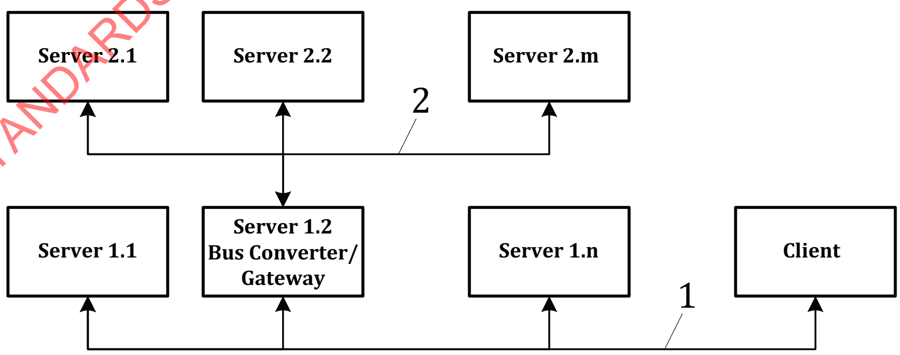
The following recommendations apply:

— It is recommended to no longer support the L-Line in server (ECU) hardware.

— Client (external test equipment) hardware shall support the L-Line if compliance to ISO 15031-4 is required.

For more details, refer to ISO 14230-1 "K-/L-line configurations".

Figure 4 illustrates an example of multiple servers (ECUs) connected with the K-Line to the client (external test equipment). Server 1.2 (ECU 1.2) functions as a gateway (bus converter) and is operating on a bus system (e.g. ISO 15765, SAE J1850).



Key

1 K-Line

2 arbitrary bus system

Figure 4 — Gateway topology example

7 Data link layer overview

7.1 General

This part of ISO 14230 specifies the data link layer services which are used in client-server based systems to transmit data from one to the other entity. The client, referred to as the external test equipment, uses the data link layer services to transfer diagnostic request data to one or more servers, referred to as an ECU. The server, usually a function that is part of an ECU, uses the data link layer services to send response data, provided by the requested diagnostic service, back to the client. The client is usually the external test equipment but can in some systems also be an on-board test equipment. The usage of data link layer services is independent from the external test equipment being an off-board or on-board test equipment. It is possible to have more than one client (test equipment) in the same vehicle system.

In order to describe the function of the data link layer, services provided to higher layers and the internal operation of the data link layer have to be considered.

7.2 Format description of data link layer services

All data link layer services have the same general format. Service primitives are written in the form

```
service_name.type    (
                     [parameter 1, parameter 2, parameter 3, ...]
                     )
```

where

— "service_name" is the name of the diagnostic service (e.g. DL_Data),

— "type" indicates the type of the service primitive (e.g. request),

— "[parameter 1, ...]" are parameters that depend on the specific service (e.g. parameter 1 can be the source address of the sender). The brackets indicate that this part of the parameter list may be empty.

7.3 Services provided by the data link layer to higher layers

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

The service access point of the data link layer provides the following service primitives as specified:

— 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.

The three types of services are defined:

a) Initialization services

These services, of which the following are defined, provide the functionality to perform the initialization of the DoK-Line communication.

— DoK-Line_Initialize.request

This service is used to request the DoK-Line communication.

— DoK-Line_Initialize.confirm

This service confirms to the higher layers that the DoK-Line communication has been carried out (successfully or not).

b) Communication services

These services, of which the following are defined, enable the transfer of up to 255 bytes of data.

— DL_Data.request

This service is used to request the transfer of data.

— DL_Data_FB.indication

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

— DL_Data.indication

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

— DL_Data.confirm

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

c) InputOutputControl services

These services, of which the following are defined, provide the functionality to perform certain fixed sequences (e.g. 5-baud initialization, wake-up pattern generation).

— DoK-Line_IOControl.request

This service is used to request the execution of a specific data link layer sequence.

— DoK-Line_IOControl.confirm

This service confirms to the upper layer that the request to execute a specific data link layer sequence has been done (successfully or not).

d) Protocol parameter setting services

These services, of which the following are defined, enable the dynamic setting of protocol parameters.

— DoK-Line_ChangeParameter.request

This service is used to request the dynamic setting of specific internal parameters (e.g. timing parameters).

— DoK-Line_ChangeParameter.confirm

This service confirms to the upper layer that the request to change a specific protocol parameter has been carried out (successfully or not).

7.4 Specification of DoK-Line data link layer service primitives

7.4.1 DL_Data.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 SA and TA.

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

```
DL_Data.request        (
                       SA
                       TA
                       TAtype
                       <MessageData>
                       <Length>
                       )
```

7.4.2 DL_Data.confirm

The data link layer issues the DL_Data.confirm service. The service primitive confirms the completion of a DL_Data.request service identified by the address information in SA and TA. The parameter < Result_DoK-Line > provides the status of the service request.

```
DL_Data.confirm        (
                       SA
                       TA
                       TAtype
                       <Result_DoK-Line>
                       )
```

7.4.3 DL_Data_FB.indication

The data link layer issues the DL_Data_FB.indication service. The service primitive indicates to the adjacent upper layer the arrival of the first byte (FB) of a message received from a peer protocol entity identified. This indication shall take place upon reception of the first byte of a message.

The DL_Data_FB.indication service shall always be followed by a DL_Data.indication service call from the data link layer to indicate the completion (or failure) of the message reception.

```
DL_Data_FB.indication        (
                             SA
                             TA
                             TAtype
                             <Length>
                             <Result_DoK-Line>
                             )
```

There is no address information contained in the indication, because the first byte only indicates the start of a message. There can only be one message transmitted on the data link layer at a time (no multiple messages can be pending in the data link layer at a time); therefore, the first byte indication does not require any address information. The final indication of the message reception will contain the address information for the received message.

7.4.4 DL_Data.indication

The data link layer issues the DL_Data.indication service. The service primitive indicates < Result_DoK-Line > events and delivers < MessageData > with < Length > bytes received from a peer protocol entity identified by the address information in SA and TA to the adjacent upper layer.

The parameters < MessageData > and < Length > are only valid if < Result_DoK-Line > equals DoK-Line_OK.

```
DL_Data.indication           (
                             SA
                             TA
                             TAtype
                             <MessageData>
                             <Length>
                             <Result_DoK-Line>
                             )
```

7.4.5 DoK-Line_Init.request

The service primitive requests the initialization of the data link layer.

Each time the DoK-Line_Initialize.request service is called, the data link layer shall signal the completion (or failure) of the message transmission to the service user by means of issuing a DoK-Line_Initialize.confirm service call.

```
DoK-Line_Initialize.request        (
                                   SA
                                   TA
                                   <InitializationModeIdentifier>
                                   )
```

7.4.6 DoK-Line_Initialize.confirm

The data link layer issues the DoK-Line_Initialize.confirm service. The service primitive confirms the completion of a DoK-Line_Initialize.request service. The parameter < Result_Initialize > provides the status of the service request, and the parameter < InitializeResultData > provides result data of the performed input output control (e.g. key bytes).

```
DoK-Line_Initialize.confirm        (
                                   <Result_Initialize>
                                   <InitializeResultData>
                                   )
```

7.4.7 DoK-Line_ChangeParameter.request

The service primitive is used to request the change of an internal parameter's value on the local protocol entity. The < Parameter_Value > is assigned to the < Parameter > (see 10.2 for parameter definition).

A parameter change is always possible except after reception of the first byte (DL_Data_FB.indication) until the end of the reception of the corresponding message (DL_Data.indication).

```
DoK-Line_ChangeParameter.request   (
                                   <Parameter>
                                   <Parameter_Value>
                                   )
```

This is an optional service that can be replaced by implementation of fixed parameter values.

7.4.8 DoK-Line_ChangeParameter.confirm

The service primitive confirms the completion of a DoK-Line_ChangeParameter.Confirmation service (see 10.2 for parameter definition).

```
DoK-Line_ChangeParameter.confirm   (
                                   <Parameter>
                                   <Result_ChangeParameter>
                                   )
```

7.5 Service data unit specification

7.5.1 SA, Source Address

Type: 1 byte unsigned integer value

Range: 0x00 – 0xFF

Description:

The parameter SA shall be used to encode client and server identifiers, and it shall be used to represent the physical location of a client or server.

For the transmission of data from the client to the server, SA represents the client identifier in the service request, service indication, and service confirmation.

For the transmission of data from the server to the client, SA represents the server identifier in the service request, service indication, and service confirmation.

The client shall always be located in one external test equipment only. There shall be a strict, one to one, relation between client identifiers and source addresses. Each client identifier shall be encoded with one SA value. If more than one client is implemented in the same external test equipment, then each client shall have its own client identifier and corresponding SA value.

A server may be implemented in one ECU only or be distributed and implemented in several ECUs. If a server is implemented in one ECU only, then it shall be encoded with one SA value only. If a server is distributed and implemented in several ECUs, then the server identifier shall be encoded with one SA value for each physical location of the server.

7.5.2 TA, Target Address

Type: 1 byte unsigned integer value

Range: 0x00 – 0xFF

Description:

The parameter TA shall be used to encode client and server identifiers.

For the transmission of data from the client to the server, TA represents the server identifier in the service request, service indication, and service confirmation.

For the transmission of data from the server to the client, TA represents the client identifier in the service request, service indication, and service confirmation.

TA may be a physical or a functional address. Physical addresses may be the 5-baud address byte (see ISO 9141:1989, Annex A and Annex B).

For emissions-related messages, this byte is defined in ISO 14230-4.

7.5.3 TAtype, target address type

Type: enumeration

Range: physical, functional

Description:

The parameter TAtype is an extension to the TA parameter. It shall be used to encode the communication model used by the communicating peer entities of the data link layer. Two communication models are specified: 1-to-1 communication called physical addressing and 1-to-n communication called functional addressing. (For the format of the format byte in the DoK-Line_PDU to handle the two addressing types, see 9.2.1.)

7.5.4 < Length >

Type: 1 byte

Range: 0x00 – 0xFF

Description:

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

7.5.5 < MessageData >

Type: string of bytes

Range: not applicable

Description:

This parameter includes all data exchanged by the higher layer entities.

7.5.6 < Result_DoK-Line >

Type: enumeration

Range: DoK-Line_OK, DoK-Line_TIMEOUT_P1, DoK-Line_TIMEOUT_P4, DoK-Line_UNEXP_PDU

Description:

This parameter contains the status relating to the outcome of a service execution. If more than one error is discovered at the same time, then the data link layer entity shall use the parameter value first found in this list in the error indication to the higher layers.

— DoK-Line_OK

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

— DoK-Line_TIMEOUT_P1

  This parameter is issued to the protocol user when the timer DoK-Line_P1 has passed its timeout value DoK-Line_P1$_{max}$. This parameter can be issued to the service user on the server side.

— DoK-Line_TIMEOUT_P4

  This parameter is issued to the protocol user when the timer DoK-Line_P4 has passed its timeout value DoK-Line_P4$_{max}$. This parameter can be issued to the service user on the client side.

— DoK-Line_UNEXP_PDU

  This parameter is issued to the service user upon reception of an unexpected protocol data unit. This parameter can be issued to the service user on both the sender and receiver side.

NOTE The status parameters DoK-Line_TIMEOUT_P1 and DoK-Line_TIMEOUT_P4 are equivalent for the upper layer.

7.5.7 < InitializationModeIdentifier >

Type: 1 byte unsigned integer value

Range: 0x00 – 0xFF

Description:

This parameter identifies the type of initialization to be performed by the data link layer.

— Perform 5-BAUD_INIT initialization sequence and provide the resulting key bytes.

— Perform FAST_INIT initialization sequence and provide the resulting key bytes.

NOTE The above listed functionality is only required to be supported by the client (external test equipment).

7.5.8 < InitializationResultData >

Type: string of bytes

Range: not applicable

Description:

This parameter includes all data provided by the initialization procedure (e.g. key bytes).

7.5.9 < Result_Initialization >

Type: enumeration

Range: DoK-Line_OK, DoK-Line_RX_ON, DoK-Line_WRONG_PARAMETER, DoK-Line_WRONG_VALUE

Description:

This parameter contains the status relating to the outcome of a service execution.

— DoK-Line_OK

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

— DoK-Line_RX_ON

  This parameter is issued to the service user to indicate that the service did not execute, since a reception of the message identified by < AI > was taking place. This parameter can be issued to the service user on the receiver side only.

— DoK-Line_WRONG_PARAMETER

  This parameter is issued to the service user to indicate that the service did not execute due to an undefined < Parameter > . This parameter can be issued to the service user on both the receiver and sender side.

— DoK-Line_WRONG_VALUE

  This parameter is issued to the service user to indicate that the service did not execute due to an out of range < Parameter_Value > . This parameter can be issued to the service user on both the receiver and sender side.

7.5.10 < Parameter_Value >

Type: 1 byte unsigned integer value

Range: 0x00 – 0xFF

Description:

This parameter is assigned to a protocol parameter < Parameter > as indicated in the service section of this document. The upper layer can, for instance, configure what kind of DoK-Line_FMT shall be placed in the DoK-Line_SDU when transmitting a message (see Clause 9).

7.5.11 < Result_ChangeParameter >

Type: enumeration

Range: DoK-Line_OK, DoK-Line_RX_ON, DoK-Line_WRONG_PARAMETER, DoK-Line_WRONG_VALUE

Description:

This parameter contains the status relating to the outcome of a service execution.

— DoK-Line_OK

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

— DoK-Line_RX_ON

This parameter is issued to the service user to indicate that the service did not execute, since a reception of the message identified by < AI > was taking place. This parameter can be issued to the service user on the receiver side only.

— DoK-Line_WRONG_PARAMETER

This parameter is issued to the service user to indicate that the service did not execute due to an undefined < Parameter >. This parameter can be issued to the service user on both the receiver and sender side.

— DoK-Line_WRONG_VALUE

This parameter is issued to the service user to indicate that the service did not execute due to an out of range < Parameter_Value > . This parameter can be issued to the service user on both the receiver and sender side.

8 Protocol initialization

8.1 General

ISO 9141-2 and ISO 14230-2 define three different methods to accomplish asynchronous-to-synchronous communications.

The three protocols are separate and distinct:

— ISO 9141-2 with 5-BAUD_INIT initialization,

— ISO 14230-2 with 5-BAUD_INIT initialization,

— ISO 14230-2 with FAST_INIT initialization.

ISO 14230-4 states that all emissions-related OBD ECUs on a single vehicle shall only support one of either the 5-BAUD_INIT or the FAST_INIT. ISO 9141-2 also defines a 5-BAUD_INIT sequence, which is differentiated from the ISO 14230-2, 5-BAUD_INIT sequence by the key bytes that the vehicle responds with.

8.2 Timing parameters for 5-BAUD_INIT

Table 2 shows timing values for 5-baud initialization. These are fixed values. They cannot be changed by the AccessCommunicationParameter service.

Table 2 — Timing parameters for 5_BAUD_INIT

Timing parameter	Values [ms]		Description
	min	max	
W1	60	300	Time from end of the address byte to start of synchronization pattern.
W2	5	20	Time from end of the synchronization pattern to the start of key byte 1.
W3	0	20	Time between key byte 1 and key byte 2.
W4	25	50	Time between key byte 2 (from the server) and its inversion from the client. Also the time from the inverted key byte 2 from the client and the inverted address from the server.
W5	300	-	Time before the client starts to transmit the address byte.

8.3 Protocol determination

8.3.1 5-BAUD_INIT according to ISO 9141

ISO 9141 5-BAUD_INIT initialization starts with a sequence by the client (external test equipment) issuing the address byte at 5 bits per second.

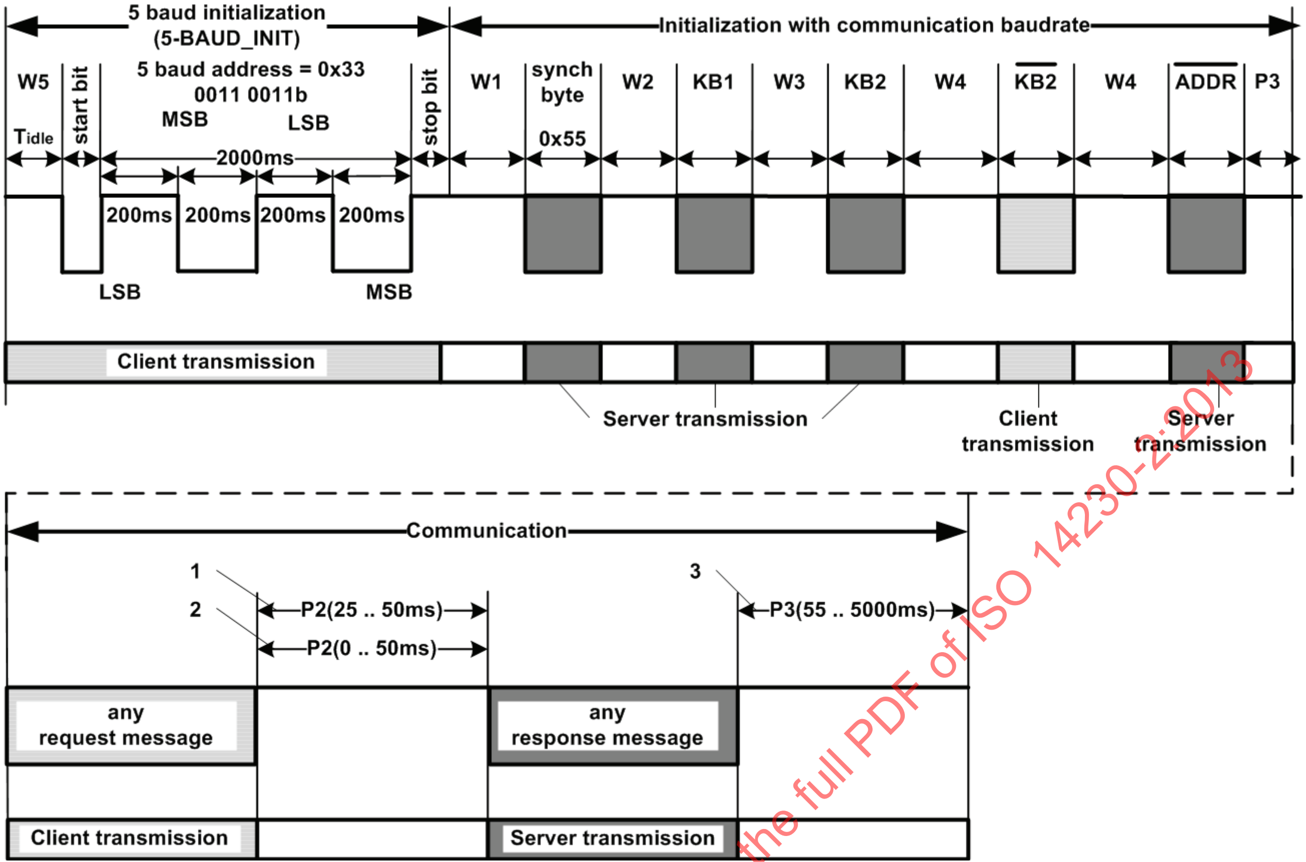
The address byte has a preceding start bit (level low) and a following stop bit (level high). This gives a total length of 10 bits to be transmitted at 5 baud (see Figure 5 and Figure 6).

Table 3 defines the initialization procedure.

Table 3 — ISO 9141 initialization procedure with 5-BAUD_INIT

#	Step	Client/ Server	Description
1	Address byte transmission	client	Address byte at 5 baud including start and stop bit takes 2 seconds.
2	Address byte validation	server	Validation of the address byte internally in the vehicle servers, which takes the time W1 (20 .. 300 ms).
3	Synchronization byte transmission	server	Exactly one vehicle server will respond with the synchronization byte 0x55 informing the external test equipment of the new baud rate.
4	Synchronization byte validation & set new baud rate	client	Reconfiguration has to be done within 5 ms.
5	Key bytes transmission	server	The vehicle server which has sent the synchronization byte shall wait the time W2 (5 .. 20 ms) for the client to reconfigure to the new baud rate. Then this vehicle server will send the two key bytes.
6	Key bytes validation	client	See 8.4 for protocol-specific key bytes. According to the received key bytes, the external test equipment (client) has to configure: — ISO 9141 protocol, — header format, — timing ($P2_{min}$).
7	Inverted key byte 2 transmission	client	As an acknowledgement of reception of the key bytes, the client, after waiting W4 (25 .. 50 ms), shall then send the inverted key byte 2 to the vehicle servers.
8	Validation of inverted key byte	server	Evaluation of inverted key byte.
9	Inverted address byte transmission	server	After waiting another period equal to W4, the vehicle server which has sent the synchronization byte shall then invert the initialization address byte and send it to the client as the “ready to communicate” signal. This ends the initialization sequence from the viewpoint of the server.
10	Validation of inverted address byte	client	Evaluation of inverted address byte. This ends the initialization sequence from the viewpoint of the client.

Figure 5 shows the initialization with 5-BAUD_INIT according to ISO 9141-2.



Key

1 $P2_{Server}$ timing value (25 .. 50 ms) depends on the key bytes (normal timing).

2 $P2_{Server}$ timing value (0 .. 50 ms) depends on the key bytes (extended timing).

3 $P3_{Client}$ timing value (55 .. 5 000 ms) depends on the key bytes.

Figure 5 — Initialization with 5-BAUD_INIT according to ISO 9141-2

8.3.2 5-BAUD_INIT according to ISO 14230-2

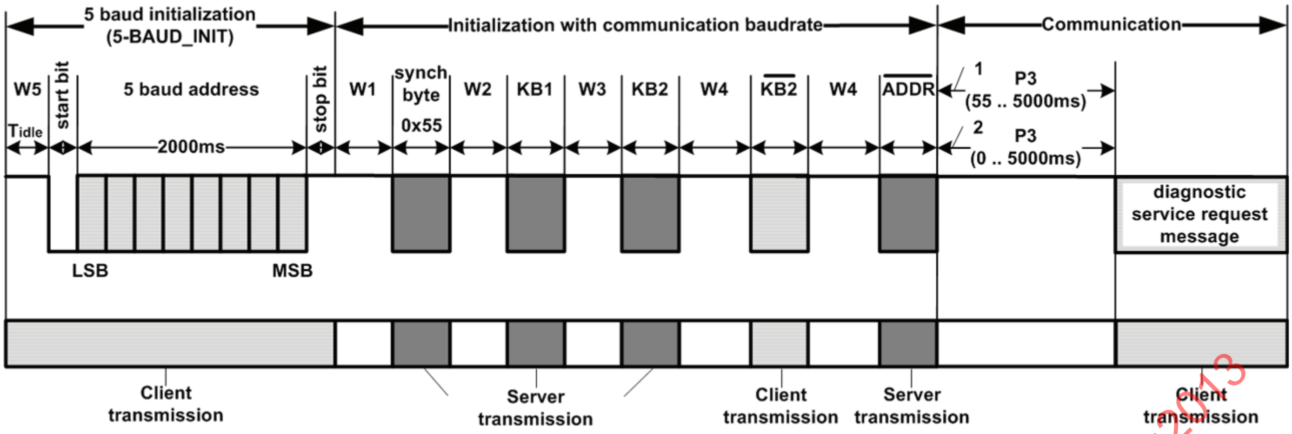
ISO 14230 5-BAUD_INIT is identical to the ISO 9141 5-BAUD_INIT, except for the key bytes sent from the vehicle to the external test equipment. The definition in 8.4.1 Table 11 is valid for both protocols. The key byte sets are defined in 8.4.2 and 8.4.4.

Table 4 defines the ISO 14230-2 initialization procedure with 5-BAUD_INIT.

Table 4 — ISO 14230-2 initialization procedure with 5-BAUD_INIT

#	Step	Client/Server	Description
1	Address byte transmission	client	Address byte at 5 baud including start and stop bit takes 2 seconds.
2	Address byte validation	server	Validation of the address byte internally in the vehicle servers, which takes the time W1 (20 .. 300 ms).
3	Synchronization byte transmission	server	Exactly one vehicle server will respond with the synchronization byte 0x55 informing the external test equipment of the new baud rate.
4	Synchronization byte validation & set new baud rate	client	Reconfiguration has to be done within 5 ms.
5	Key bytes transmission	server	The vehicle server which has sent the synchronization byte shall wait the time W2 (5 .. 20 ms) for the client to reconfigure to the new baud rate. Then this vehicle server will send the two key bytes.
6	Key bytes validation	client	See 8.4 for protocol-specific key bytes. According to the received key bytes, the external test equipment (client) has to configure: — ISO 14230-2 protocol, — header format, — timing ($P2_{min}$).
7	Inverted key byte 2 transmission	client	As an acknowledgement of reception of the key bytes, the client, after waiting W4 (25 .. 50 ms), shall then send the inverted key byte 2 to the vehicle servers.
8	Validation of inverted key byte	server	Evaluation of inverted key byte.
9	Inverted address byte transmission	server	After waiting another period equal to W4, the vehicle server which has sent the synchronization byte shall then invert the initialization address byte and send it to the client as the “ready to communicate” signal. This ends the initialization sequence from the viewpoint of the server.
10	Validation of inverted address byte	client	Evaluation of inverted address byte. This ends the initialization sequence from the viewpoint of the client.

Figure 6 shows the initialization with 5-BAUD_INIT according ISO 14230-2.



Key

1 $P3_{Client}$ timing value (55 .. 5 000 ms) depends on the key bytes (normal timing).

2 $P3_{Client}$ timing value (0 .. 5 000 ms) depends on the key bytes (extended timing).

Figure 6 — Initialization with 5-BAUD_INIT according to ISO 14230-2

NOTE Depending on the address byte, the high periods may be longer than W5 and may not be interpreted as idle times.

8.3.3 FAST_INIT according to ISO 14230-2

8.3.3.1 General

All servers (ECUs) which are initialized shall use a baud rate of 10 400 baud for initialization and communication.

The client (external test equipment) transmits a wake-up pattern (WuP) on K- and L-Line synchronously. The pattern begins after an idle time on the K-line with a low time of T_{iniL}. The client (external test equipment) transmits the first bit of the StartCommunication service after a time of t_{WuP} following the first falling edge.

8.3.3.2 Timing values of FAST_INIT according to ISO 14230-2

Table 5 defines the timing values of FAST_INIT according to ISO 14230-2.

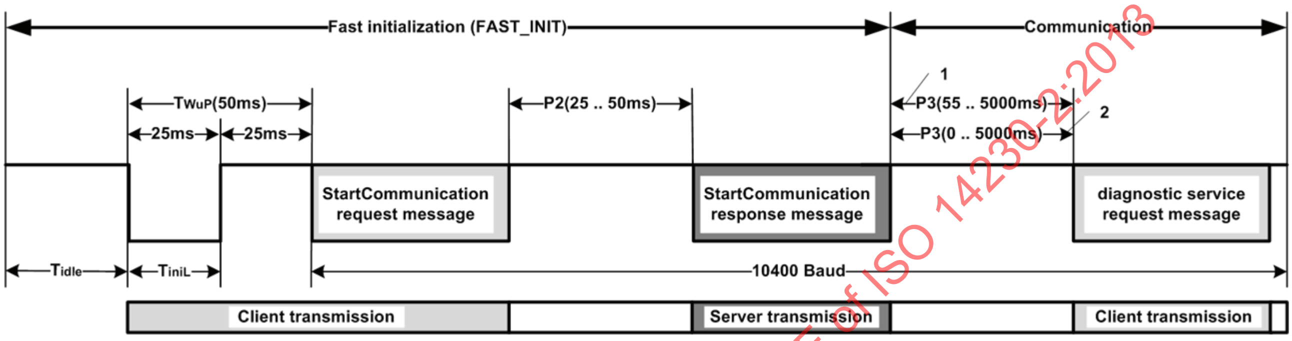
Table 5 — Timing values of FAST_INIT according to ISO 14230-2

Timing parameter name	Minimum timing value [ms]	Timing value [ms]	Maximum timing value [ms]
T_{Idle}	First transmission after power on	W5	see Table 2
	After completion of StopCommunication service	$P3_{min}$	see Table 17
	After stopping communication by timeout $P3_{max}$ (starting with a falling edge)	0	0
T_{iniL}	24	25 ± 1	26
T_{WuP}	49	50 ± 1	51

8.3.3.3 Initialization sequence of FAST_INIT according to ISO 14230-2

The transfer of a wake-up pattern WuP as described above is followed by a StartCommunication request message from the client (external test equipment) and a response message from the server (ECU). The first message of a fast initialization always uses a header with target and source address and without additional length byte. A server (ECU) may answer back with or without address information and length byte and tells its supported modes within the key bytes. The pattern begins with a falling edge after an idle time on K-line. While communication is running, it is not necessary for a server (ECU) to react when receiving a wake-up pattern WuP.

Figure 7 shows the initialization with FAST_INIT according to ISO 14230-2.



Key

1 $P3_{Client}$ timing value (55 .. 5 000 ms) depends on the key bytes (normal timing).

2 $P3_{Client}$ timing value (0 .. 5 000 ms) depends on the key bytes (extended timing).

Figure 7 — Initialization with FAST_INIT according to ISO 14230-2

IMPORTANT — The ISO 14230 FAST_INIT initialization is successful if the vehicle has responded with a StartCommunication positive response message on a StartCommunication request message PDU = [SID] 0x81 request message. If there is no vehicle StartCommunication positive response message, the initialization sequence has failed.

8.3.3.4 Message sequence of FAST_INIT according to ISO 14230-2

ISO 14230-2 FAST_INIT begins with the WuP transmitted by the client to the vehicle that is 50 ms long. This pattern is followed immediately by a StartCommunication request message from the client to the vehicle. The vehicle should then respond to the external test equipment with a StartCommunication response message, including a pair of key bytes.

The StartCommunication request message comprises a format byte, target address byte, source address byte, and a service identifier byte of 0x81.

Table 6 defines the StartCommunication request message for ISO 14230-2.

Table 6 — StartCommunication request message for ISO 14230-2

Byte	Parameter name	Cvt	Value	Mnemonic
1	Format byte = [physical addressing or functional addressing]	M	0xXX = [0x81 or 0xC1]	FMT
2	Target address byte	M	0xXX	TGT
3	Source address byte	M	0xXX	SRC
4	StartCommunication Request Service Identification	M	0x81	STC
5	Checksum	M	0xXX	CS

The StartCommunication response message is also comprised of a format byte, target address byte, source address byte, and a service identifier byte of 0xC1.

Table 7 defines the StartCommunication response message for ISO 14230-2.

Table 7 — StartCommunication response message for ISO 14230-2

Byte	Parameter name	Cvt	Value	Mnemonic
1	Format byte	M	0xXX	FMT
2	Target address byte	M	0xXX	TGT
3	Source address byte	M	0xXX	SRC
4	StartCommunication Response Service Identification	M	0xC1	STC
5	Key byte 1 [a]	M	0xXX	KB1
6	Key byte 2 [a]	M	0xXX	KB2
7	Checksum	M	0xXX	CS

[a] The key bytes are defined in 8.4.

The reception of the StartCommunication response message terminates the initialization sequence.

8.3.4 FAST_INIT according to ISO 14230-4

8.3.4.1 General

All emissions-related servers (ECUs), which are initialized, shall use a baud rate of 10 400 baud for initialization and communication.

The client (external test equipment) transmits a wake-up pattern (WuP) on K- and L-Line synchronously. The pattern begins after an idle time on the K-line with a low time of T_{iniL}. The client (external test equipment) transmits the first bit of the StartCommunication service after a time of t_{WuP} following the first falling edge.

8.3.4.2 Message sequence of FAST_INIT according to ISO 14230-4

ISO 14230-4 FAST_INIT begins with the WuP transmitted by the client to the vehicle that is 50 ms long. This pattern is followed immediately by a StartCommunication request from the client to the vehicle. The vehicle should then respond to the client with a StartCommunication response, including a pair of key bytes.

Since functional addressing is required for legislated OBD communication, the StartCommunication request message shall have the format byte 0xC1, the target address 0x33, the source address (client) 0xF1. The StartCommunication request service identifier is 0x81.

Table 8 defines the StartCommunication request message for ISO 14230-4.

Table 8 — StartCommunication request message for ISO 14230-4

Byte	Parameter name	Cvt	Value	Mnemonic
1	Format byte	M	0xC1	FMT
2	Target address byte	M	0x33	TGT
3	Source address byte	M	0xF1	SRC
4	StartCommunication Request Service Identification	M	0x81	STC
5	Checksum	M	0x66	CS

There is no vehicle response allowed between the "wake-up" pattern and the StartCommunication request message. The first vehicle response will be a StartCommunication positive response message. A StartCommunication response message is similarly comprised of a format byte, a target address byte, a source address byte, a service ID byte, and the key bytes. The format byte is defined in ISO 15031-5. Thus, in this StartCommunication response message with three data bytes, the format byte value is 0x83.

Table 9 defines the StartCommunication response message for ISO 14230-4.

Table 9 — StartCommunication response message for ISO 14230-4

Byte	Parameter name	Cvt	Value	Mnemonic
1	Format byte	M	0x83	FMT
2	Target address byte	M	0xF1	TGT
3	Source address byte	M	0xXX	SRC
4	StartCommunication Response Service Identification	M	0xC1	STC
5	Key byte 1 [a]	M	0xXX	KB1
6	Key byte 2 [a]	M	0xXX	KB2
7	Checksum	M	0xXX	CS

[a] The key bytes are defined in 8.4.

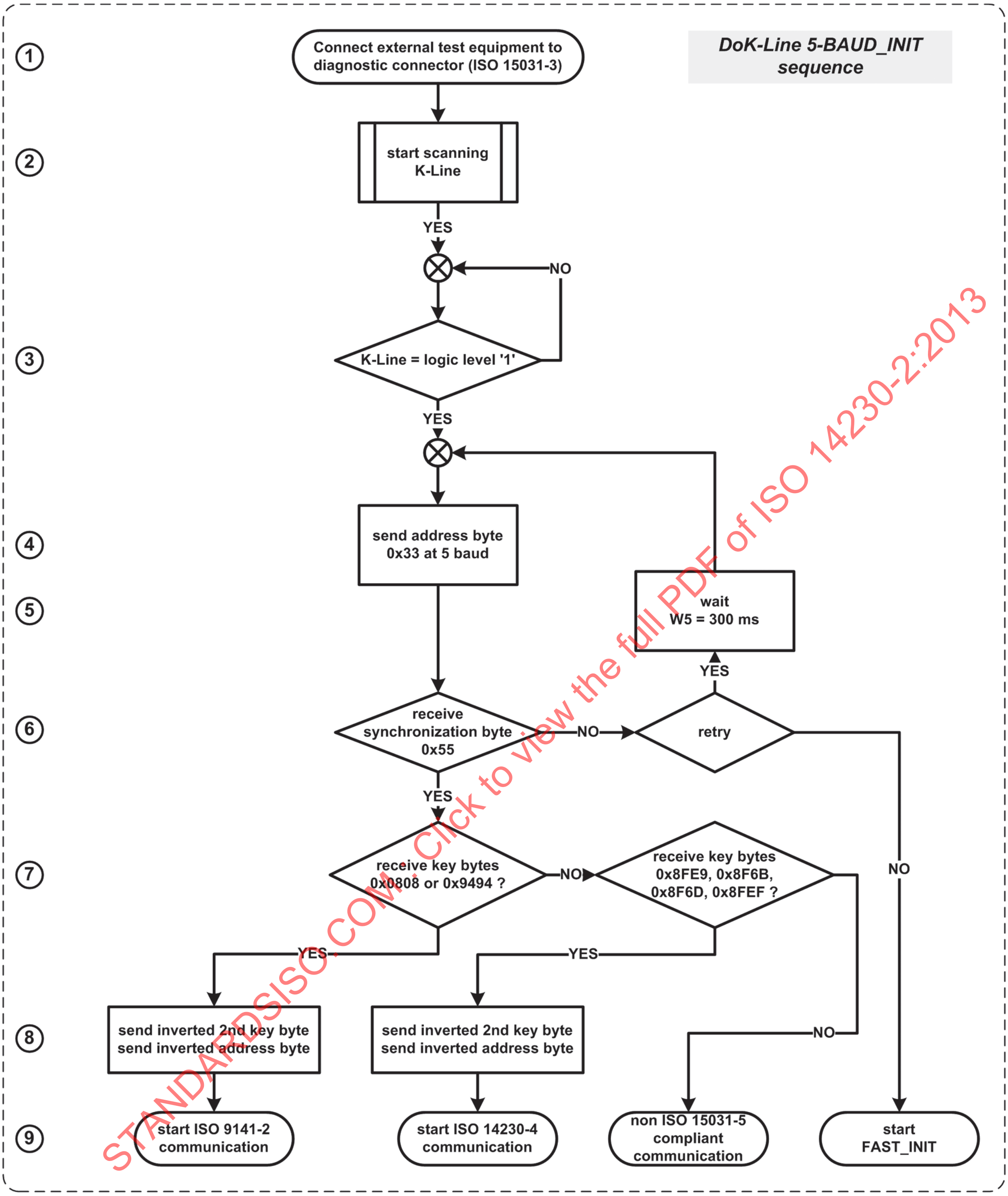
The reception of the StartCommunication response message terminates the initialization sequence.

8.3.5 Client protocol determination by server (ECU) key bytes

The key bytes are the important differentiator during 5-BAUD_INIT attempts to determine whether the ISO 9141 or ISO 14230 protocol is to be used for this communication session.

NOTE Due to this key byte differentiation, only one 5-BAUD_INIT sequence by the client is necessary to determine the protocol. This speeds up the initialization process.

Figure 8 shows the client protocol determination by server key bytes for legislated communication as an example.



Key

1 Connect external test equipment (ISO 15031-4) to vehicle diagnostic connector (ISO 15031-3).

2 External test equipment scans (detects logic level of) K-Line. If "YES", then branch to step 3.

3 External test equipment checks if K-Line is at logic level '1' (B+ = battery supply voltage). If "NO", then branch to step 2. If "YES", then branch to step 4.

4 External test equipment sends address byte 0x33 at 5 baud onto K- and L-Line. Then branch to step 6.

5 External test equipment waits W5 (300 ms) before it continues with step 4 again.

6 External test equipment waits for synchronization byte 0x55 from the vehicle. If “NO”, then retry the 5-baud initialization. If number of retries reached (“NO”), then branch to step 9 (start FAST-INIT).

7 External test equipment has received key bytes and evaluates whether the value of the key bytes is either 0x0808 or 0x9494. If “YES”, then branch to step 8. If “NO”, then evaluate whether the value of the key bytes is one of the following: 0x8FE9, 0x8F6B, 0x8F6D, 0x8FEF. If “NO”, then branch to step 9 (non ISO 15031-5-compliant communication). If “YES”, then branch to step 8.

8 External test equipment sends inverted 2nd key byte and inverted address byte onto the K-Line to the vehicle. Then branch to step 9.

9 Result of client protocol determination is either ISO 9141-2 communication or ISO 14230-4 communication or is not ISO 15031-5 compliant communication or to start the FAST_INIT of ISO 14230-4 communication.

Figure 8 — Client protocol determination by server key bytes for legislated communication

There are two recommended initialization methodologies that will be described here:

— Attempt a 5-BAUD_INIT initialization, followed by a FAST_INIT initialization. This guarantees that the correct timings have been maintained.

— Attempt a FAST_INIT initialization, followed by a 5-BAUD_INIT initialization. For external test equipment (clients) which try ISO 14230-2 FAST_INIT prior to the combined ISO 9141-2/ISO 14230-2, 5-BAUD_INIT sequence, a wait time shall be implemented as specified in Table 10 before sending the address byte at 5 baud after unsuccessful FAST_INIT.

Table 10 — Wait time between FAST_INIT and ISO 9141/ISO 14230-2, 5-BAUD_INIT sequence

Item	Operation	Duration
Transmission of address byte		2,0 s
W1 validation period	+	0,3 s
W5 external test equipment wait period	+	0,3 s
Total wait time	sum	2,6 s

8.3.6 Initial data exchange after successful completion of initialization

8.3.1 to 8.3.5 specify the complete initialization sequence for each protocol. Any subsequent communication after the initialization sequence is specified as communication. Communication comprises request and response messages.

For legislated OBD, communication starts with a service request message PDU = ([SID] 0x01, [PID] 0x00) for vehicle-supported emissions-related data. The service request message shall be performed after any initialization sequence and shall be compliant with the ISO 15031-5 and shall have the correct header in relation to the responded key bytes from the vehicle.

8.4 Protocol-specific key bytes

8.4.1 Format of key bytes

With the key bytes, a server informs the client about the supported header, timing, and length information. A server not necessarily has to support all possibilities. The decoding of the key bytes is defined in ISO 9141:1989. KB1 = Low Byte, KB2 = High Byte, 7 bit, odd parity.

Figure 9 shows the key bytes.

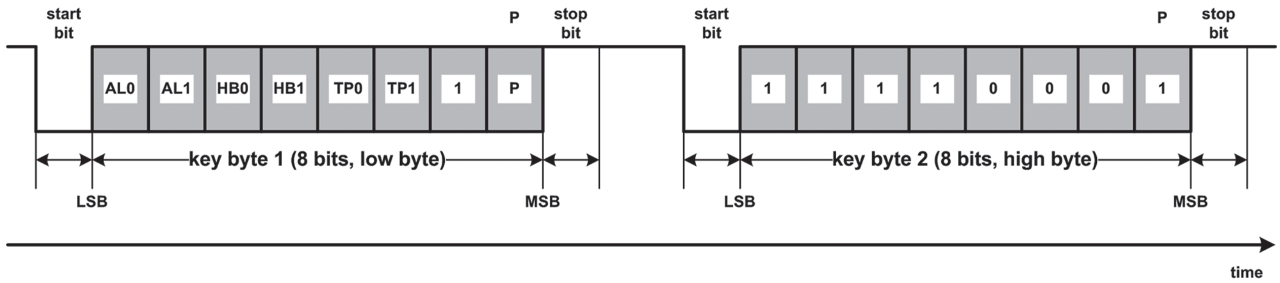


Figure 9 — Format of key bytes

Table 11 defines the key byte.

Table 11 — Definition of key byte

FMT	= 0	= 1
AL0	length information in format byte not supported	length information in format byte supported
AL1	additional length byte not supported	additional length byte supported
HB0	1 byte header not supported	1 byte header supported
HB1	Target/Source address in header not supported	Target/Source address in header supported
TP0 [a]	normal timing parameter set	extended timing parameter set
TP1 [a]	extended timing parameter set	normal timing parameter set

[a] Only TP0, TP1 = 0, 1 and 1, 0 allowed.

8.4.2 Key bytes for emissions-related OBD protocols of ISO 9141-2

ISO 9141-2 protocol defines two different pairs of key bytes which can be implemented by the server (ECU). These are either [KB1] 0x08 and [KB2] 0x08 or [KB1] 0x94 and [KB2] 0x94. Both require the same protocol message header in the diagnostic communication session. The difference between both pairs is the value of $P2_{min}$ according to Table 12.

Table 12 — Key bytes for emissions-related OBD of ISO 9141-2 protocol and the required $P2_{min}$ timing

Key byte 1	Key byte 2	Key byte [dec]	$P2_{min}$ [ms]	Timing	Description
0x08	0x08	1 032$_d$	25	Normal	Request Header: [FMT: 0x68] [TA: 0x6A] [SA: 0xF1]
0x94	0x94	2 580$_d$	0	Extended	Response Header: [FMT: 0x48] [TA: 0x6B] [SA: 0xXX] Response PDU: [SID] [DATA] (maximum 7 data bytes) Response Trailer: [CS]

8.4.3 Key bytes for emissions-related OBD protocol ISO 14230-4

ISO 14230-4 defines four different pairs of key bytes which can be implemented in the server (ECU). The header format of the protocol messages is different for these four pairs according to Table 11. Normal timing is specified for all pairs of key bytes.

IMPORTANT — For legislated emissions-related OBD communication, the client and the server shall always use the functionality of key bytes [KB1] 0x8F, [KB2] 0xE9 (i.e. 3-byte header, no additional length byte, normal timing).

The difference between the key bytes is specified in Table 13.

Table 13 — Key bytes for emissions-related OBD of ISO 14230-4 protocol and the required header format

Key byte 1	Key byte 2	Key byte [dec]	$P2_{min}$ [ms]	Timing	Description
0x8F	**0xE9**	**2 025$_d$**	25	Normal	**Request Header: [FMT: 0xC2] [TA: 0x33] [SA: 0xF1]** **Response Header: [FMT: 0x8X] [TA: 0xF1] [SA: 0xXX]** **Response PDU: [SID] [DATA] (maximum 7 data bytes)** **Response Trailor: [CS]**
0x8F	0x6B	2 027$_d$	25	Normal	Request Header: [FMT: 0xC2] [TA: 0x33] [SA: 0xF1] [optional LEN] Response Header: [FMT: 0x8X] [TA: 0xF1] [SA: 0xXX] [optional LEN] Response PDU: [SID] [DATA] Response Trailor: [CS]
0x8F	0x6D	2 029$_d$	25	Normal	Request Header: [FMT: 0xC2] Response Header: [FMT: 0x8X] Response PDU: [SID] [DATA] Response Trailor: [CS] or Response Header: [FMT: 0xC2] [TA: 0x33] [SA: 0xF1] Response PDU: [SID] [DATA] Response Trailor: [CS]
0x8F	0xEF	2 031$_d$	25	Normal	Request Header: [FMT: 0xC2] [optional LEN] Response Header: [FMT: 0x8X] [optional LEN] Response PDU: [SID] [DATA] Response Trailor: [CS] or Response Header: [FMT: 0xC2] [TA: 0x33] [SA: 0xF1] [optional LEN] Response PDU: [SID] [DATA] Response Trailor: [CS]

8.4.4 Key bytes for enhanced diagnostics with support of ISO 14230-4

ISO 14230-2 protocol defines 19 different pairs of key bytes which can be implemented by the server (ECU).

For OBDonK-Line implementations (ISO 15031-5 on ISO 14230-4), only one of the four key bytes as defined in 8.4.2 are allowed to be supported.

Table 14 defines the valid key bytes of ISO 14230-2 protocol and the required $P2_{min}$ timing.

Table 14 — Valid key bytes of ISO 14230-2 protocol and the required $P2_{min}$ timing

Key byte 1	Key byte 2	Key byte [dec]	$P2_{min}$ [ms]	Timing	Header ISO 14230-2 **protocol message**	PDU
0x8F	0xD0	$2\ 000_d$	—	no further information	[FMT] no further information	[SID] [DATA]
0x8F	0xD5	$2\ 005_d$	0	Extended (see Table 18)	[FMT]	
0x8F	0xD6	$2\ 006_d$			[FMT] [LEN]	
0x8F	0x57	$2\ 007_d$			[FMT] [optional LEN]	
0x8F	0xD9	$2\ 009_d$			[FMT] [TA] [SA]	
0x8F	0xDA	$2\ 010_d$			[FMT] [TA] [SA] [LEN]	
0x8F	0x5B	$2\ 011_d$			[FMT] [TA] [SA] [optional LEN]	
0x8F	0x5D	$2\ 013_d$			[FMT] [optional TA] [optional SA]	
0x8F	0x5E	$2\ 014_d$			[FMT] [optional TA] [optional SA] [LEN]	
0x8F	0xDF	$2\ 015_d$			[FMT] [optional TA] [optional SA] [optional LEN]	
0x8F	0xE5	$2\ 021_d$	25	Normal (see Table 17)	[FMT]	
0x8F	0xE6	$2\ 022_d$			[FMT] [LEN]	
0x8F	0x67	$2\ 023_d$			[FMT] [optional LEN]	
0x8F [a]	0xE9 [a]	$2\ 025_d$			**[FMT] [TA] [SA]** [b]	
0x8F	0xEA	$2\ 026_d$			[FMT] [TA] [SA] [LEN]	
0x8F [a]	0x6B [a]	$2\ 027_d$			[FMT] [TA] [SA] [optional LEN]	
0x8F [a]	0x6D [a]	$2\ 029_d$			[FMT] [optional TA] [optional SA]	
0x8F	0x6E	$2\ 030_d$			[FMT] [optional TA] [optional SA] [LEN]	
0x8F [a]	0xEF [a]	$2\ 031_d$			[FMT] [optional TA] [optional SA] [optional LEN]	

[a] Allowed key bytes for ISO 14230-4.

[b] The client and server shall always use this message format for legislated emissions-related OBD communication.

8.4.5 Calculation of decimal value of key bytes

To calculate the decimal value, clear the parity bit of both key bytes and then multiply key byte 2 by 27 and add key byte 1.

The key byte value of $2\ 000_d$ is out of scope of this part of ISO 14230.

9 Message definition

9.1 Message structure

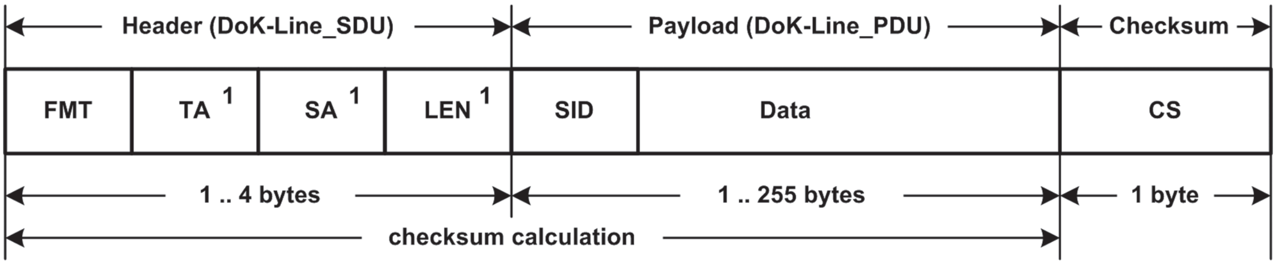
The message structure consists of three parts:

— header;

— protocol data unit (data bytes);

— checksum.

Figure 10 illustrates the content of a message.



Key

1 optional, depends on FMT (format byte value)

Figure 10 — Message structure

9.2 Message header

9.2.1 Format byte (FMT)

The format byte (FMT) contains six-bit length (L) information and two-bit address mode (A1, A0) information. The client is informed about the use of header bytes by the key bytes.

Figure 11 illustrates the definition of the address mode (A) and the length (L) information.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
A1	A0	L5	L4	L3	L2	L1	L0
		← L5 .. L0 = 0xXX: 1 .. 63 bytes → ← L5 .. L0 = 0x00: additonal length byte →					
0_b	0_b	no address information included in header					
0_b	1_b	ISO 9141-2: fixed header bytes: 0x68, 0x48					
1_b	0_b	with address information incl. in header, physical addressing					
1_b	1_b	with address information incl. in header, functional addressing					

Figure 11 — Format byte structure

The format byte structure includes address mode and length information and is defined as follows:

— L5 .. L0:

Defines the length of a message from the beginning of the PDU (SID and data) to checksum byte (not included). A message length of 1 to 63 bytes is possible. If L0 to L5 = 00000_b, then the additional length byte is included in the header as the fourth byte.

— A1, A0 = ‘0, 1’ ISO 9141-2 mode is an exception mode. This mode is not specified in this part of ISO 14230. ISO 9141-2 uses the format bytes 0x68 (0110 1000_b) and 0x48 (0100 1000_b). For further details, refer to ISO 9141-2 and ISO 15031-5.

9.2.2 Target address byte (TA)

The target address of the header defines the target node for the message to be received and is always used together with the source address byte.

The target address value specifies either physical or functional addressing. There are reserved address values defined to address the messages to specific functional systems, e.g. emissions-related systems.

— Physical addresses may be the address byte at 5 baud as specified in ISO 9141 and ISO 9141-2 (see Annex A.1 and Annex B).

— Functional address ranges are given in Annex A.2.

— For emissions-related messages, this byte is defined in ISO 14230-4 or ISO 9141-2.

NOTE The target address is an optional byte and only necessary on multi-node bus topologies. For node-to-node connections, it may be omitted.

9.2.3 Source address byte (SA)

The source address of the header defines the source address of the transmitting device (server or client). The source address value shall always be interpreted as a physical address.

There are the same possibilities for the values as described for physical target address bytes.

Available addresses for the external test equipment are specified in Annex B.

The source address is an optional byte (always used together with the target address byte) and only necessary on multi-node bus topologies. For node-to-node connections, it may be omitted.

9.2.4 Length byte (LEN)

The additional length byte is provided in the header of the message if the length in the header format byte (L0 to L5) is set to 0 0000_b. It allows the transmitting node to send messages with data fields longer than 63 bytes. With shorter messages, it may be omitted.

This byte defines the length of a message from the beginning to the end of the PDU [header byte(s) and checksum byte not included]. A data length of 1 to 255 bytes is possible. The longest message consists of a maximum of 260 bytes.

For messages with data fields of less than 64 bytes, there are two possibilities:

— length may be included in the format byte;

— length may be included in the additional length byte.

A server is not required to support both possibilities. The external test equipment is informed about the capability of a server via the key bytes whether an additional length byte is generally supported. The server has to support the features according to transmitted key bytes.

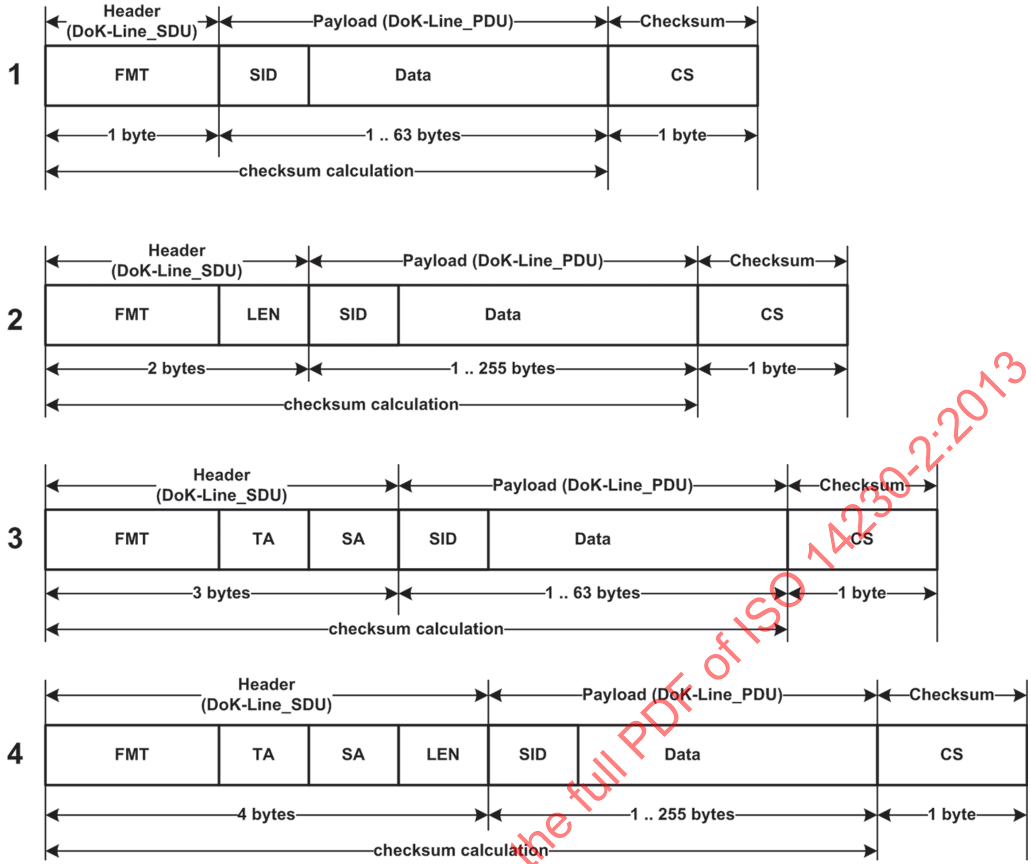
Table 15 defines the usage of the length byte.

Table 15 — Length byte definition

Length	Length provided in	
	Format byte (FMT)	Length byte (LEN)
$< 64_d$	XX00 0000b	present
$< 64_d$	XXLL LLLLb	not present
$\geq 64_d$	XX00 0000b	present

9.2.5 Message header configurations

The format byte in the header defines two bits (A1, A0) which allow four different header configurations. Figure 12 shows the message header configurations.



Key

1 Header = format byte (FMT)

2 Header = format byte (FMT) + length byte (LEN)

3 Header = format byte (FMT) + target address (TA) + source address (SA)

4 Header = format byte (FMT) + target address (TA) + source address (SA) + length byte (LEN)

Figure 12 — Message header configurations

9.3 Protocol data unit (PDU)

The PDU shall contain 1 to 63 bytes or 1 to 255 bytes of data depending on the value of the format byte (FMT: A1, A0) and the availability of the additional length (LEN) byte. The first byte of the data field is the Service Identifier (SID). All additional data bytes depend on the selected service. The PDUs of each service are defined in ISO 14229-6, UDSonK-Line or ISO 14230-3 Keyword protocol 2000.

9.4 Checksum byte (CS)

The checksum byte (CS) inserted at the end of the message block is defined as the simple 8-bit sum series of all bytes in the message, excluding the checksum.

Definition 1:

If the message is $< 1 > \ < 2 > \ < 3 > \ldots < N > , < CS >$

where $< i > (1 <= i <= N)$ is the numeric value of the i^{th} byte of the message,

then $< CS > \ = \ < CS >_N$

where $< CS >_i \ (i = 2 \text{ to } N)$

is defined as $< CS >_i = \{ < CS >_{i-1} + < i > \}$ Modulo 256 and $< CS >_1 = < 1 >$

10 Protocol timing requirements

10.1 General timing measurement requirements

A UART provides interrupt capabilities in order to notify the communication driver software to perform necessary operations to ensure reliable communication. The interrupt is generated by the UART at the end of a valid stop bit sampling.

The UART communication driver will generate

— a DL_Data.con (in case of sending the byte),

— a DL_DataFB.ind (in case of receiving the first byte of a message), or

— a DL_Data.ind (in case of receiving any byte after the first byte of a message).

10.2 Protocol timing parameter definition

10.2.1 Inter-byte and inter-message timing parameters

Table 16 defines the inter-byte and inter-message protocol timing parameters. The inter-byte timing parameters are only active between the bytes of a message (request and response message).

The inter-message timing parameters refer to the time between messages, either from the client to the server or the time between several messages from one or more server(s).

Table 16 — DoK-Line protocol timing parameter

Value	Description
P1	Inter-byte time for receiver of a message
P2	Inter-message time between client request and server response or multiple server responses
P3	Inter-message time between end of server responses and start of new client request
P4	Inter-byte time for a sender of a message

10.2.2 Inter-byte timing parameter set

Table 17 defines the normal protocol timing parameters minimum value, default value, maximum value, and the resolution which shall be used to set a new value by using the communication service AccessTimingParameter.

Table 17 — Normal inter-byte timing parameter set (for functional and physical addressing)

Timing parameter	Minimum values [ms]			Maximum values [ms]		
	default	lower limit (min)	resolution	default	upper limit (max)	resolution
P1	0	0	—	20	20	—
P2	25	0	0,5	50	calculation see Table 19	see Table 19
P3	55	0	0,5	5 000	∞ (0xFF)	250
P4	5	0	0,5	20	20	—

Table 18 defines the extended protocol timing parameters.

Table 18 — Extended inter-byte timing parameter set (for functional and physical addressing)

Timing parameter	Minimum values [ms]			Maximum values [ms]		
	default	lower limit (min)	resolution	default	upper limit (max)	resolution
P1	0	0	—	20	20	—
P2	0	0	0.5	1 000	calculation see Table 19	see Table 19
P3	0	0	0.5	5 000	(0xFF)	250
P4	5	0	0.5	20	20	—

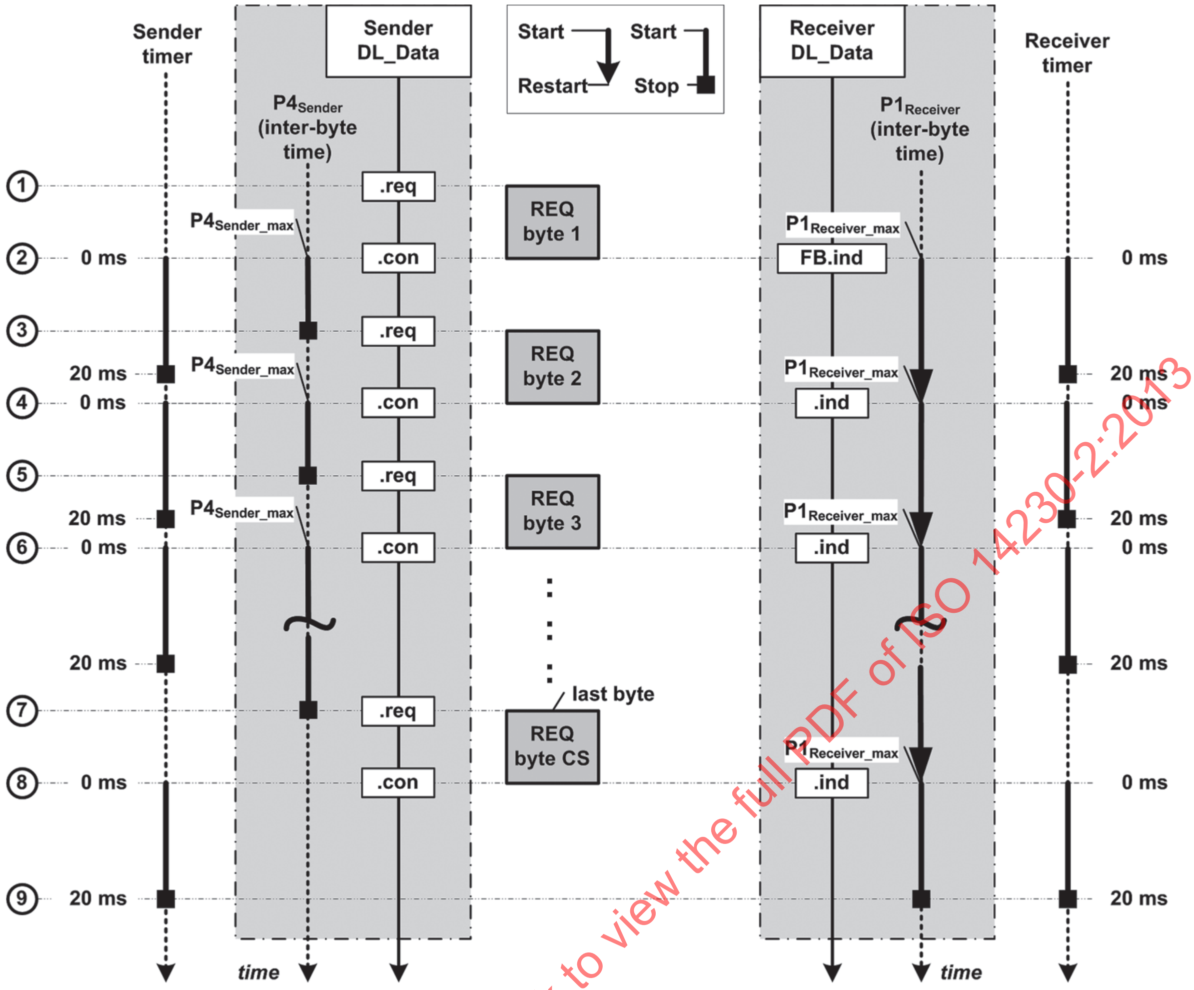
Table 19 defines the $P2_{max}$ timing parameter calculation.

Table 19 — $P2_{max}$ timing parameter calculation

Timing parameter	Value [hex]	Resolution	Maximum value [ms]	Maximum value calculation method [ms]
$P2_{max}$	0x01 – 0xF0	25	25 to 6 000	(hex value) x (resolution)
	0xF1 0xF2 0xF3 0xF4 0xF5 0xF6 0xF7 0xF8 0xF9 0xFA 0xFB 0xFC 0xFD 0xFE	see maximum value calculation method	6 400 12 800 19 200 25 600 32 000 38 400 44 800 51 200 57 600 64 000 70 400 76 800 83 200 89 600	(low nibble of hex value) x 256 x 25 Example of 0xFA: (0xFA x 0x0100) x 25 = 64 000
	0xFF	—	—	not applicable

10.3 Inter-byte message timing

Figure 13 illustrates the service primitive interface of the data link layer and the inter-byte timing parameter of the sender (P4) and the receiver (P1).



Key

1 Sender DL_Data.req: data link layer sends byte 1.

2 Receiver DL_DataFB.ind: data link layer indicates a first byte reception. The server starts the P4 timer using the value of $P1_{Receiver} = P1_{Receiver\_max}$.
Sender DL_Data.con: data link layer issues a confirmation on reception of transmitted byte via K-Line read back. The client starts the $P4_{Sender}$ timer using the value of $P4_{Sender} = P4_{Sender\_max}$.

3 Sender DL_Data.req: data link layer sends byte 2.

4 Receiver DL_Data.ind: data link layer indicates a byte reception. The server starts the P4 timer using the value of $P1_{Receiver} = P1_{Receiver\_max}$.
Sender DL_Data.con: data link layer issues a confirmation on reception of transmitted byte via K-Line read back. The client starts the P4 timer using the value of $P4_{Client} = P4_{Client\_max}$.

5 Sender DL_Data.req: data link layer sends byte 3.

6 Receiver DL_Data.ind: data link layer indicates a byte reception. The server starts the P4 timer using the value of $P1_{Receiver} = P1_{Receiver\_max}$.
Sender DL_Data.con: data link layer issues a confirmation on reception of transmitted byte via K-Line read back. The client starts the P4 timer using the value of $P4_{Client} = P4_{Client\_max}$.

7 Same routine with consecutive bytes.

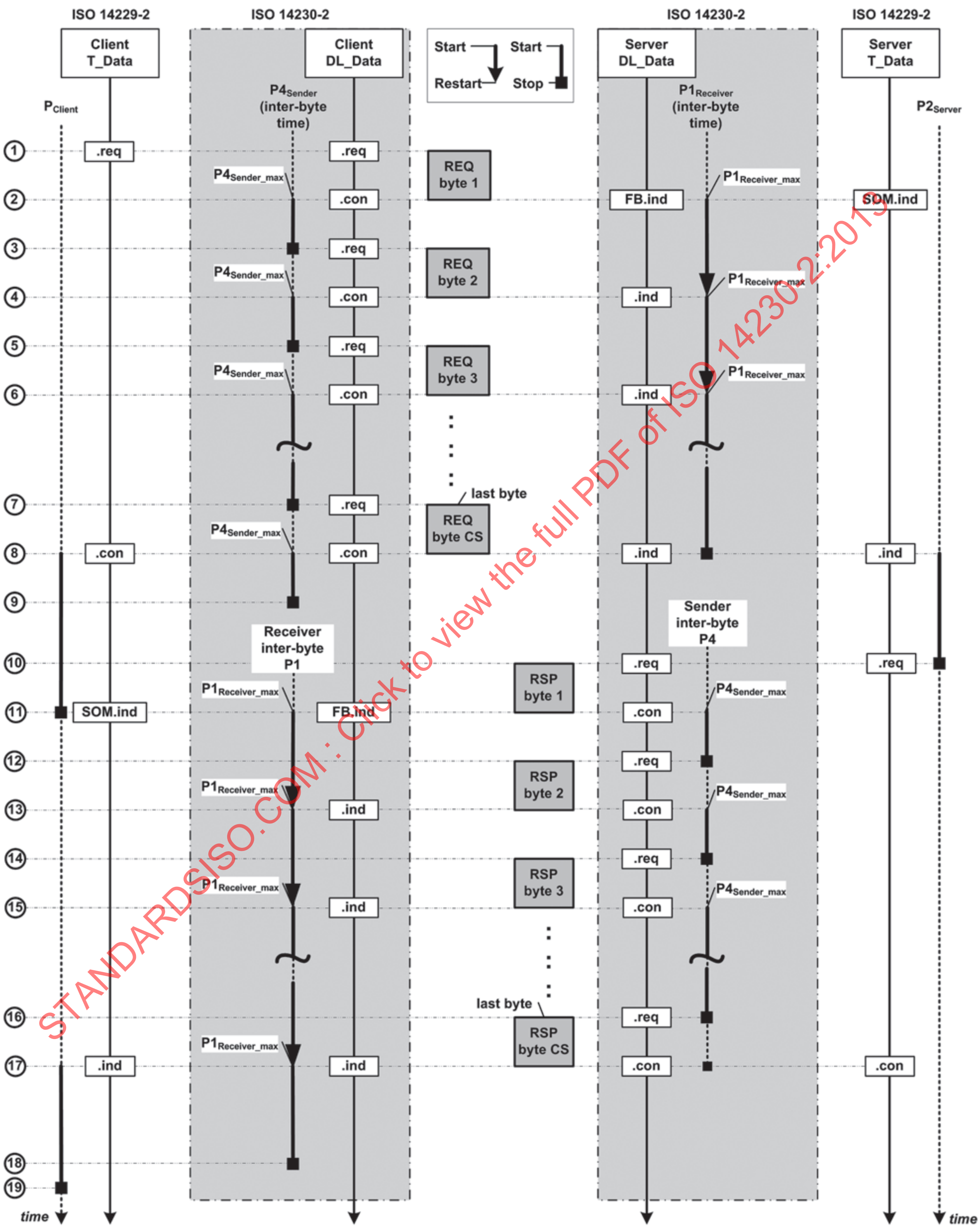
8 Sender DL_Data.req: data link layer sends last byte (i.e. CS).
Receiver DL_Data.ind: data link layer indicates a byte reception. The server starts the P4 timer using the value of $P1_{Receiver} = P1_{Receiver\_max}$.
Sender DL_Data.con: data link layer issues a confirmation on reception of transmitted byte via K-Line read back.

9 Receiver: reception of message complete upon P1 timer indicates $P1_{Receiver} > P1_{Receiver\_max}$.

Figure 13 — Data link layer request message timing

10.4 Data link layer timing at T-Data interface

Figure 14 shows the data link layer timing at the T-Data interface.



Key

1 Client_T_Data.req, Client_N_Data.req, Client_DL_Data.req: send byte 1 of request message.

2 Client_DL_Data.con, Client_N_Data.con: request byte 1 confirmation.

Server_T_Data, Server_N_DataFB.ind, Server_DL_DataFB.ind: request byte 1 (first byte) indication and start of message indication.

3 Client_N_Data.req, Client_DL_Data.req: send byte 2 of request message.

4 Client_DL_Data.con, Client_N_Data.con: request message byte 2 confirmation.

Server_DL_Data.ind, Server_N_Data.ind: request message byte 2 indication.

5 Client_N_Data.req, Client_DL_Data.req: send byte 3 of request message.

6 Client_DL_Data.con, Client_N_Data.con: request message byte 3 confirmation.

Server_DL_Data.ind, Server_N_Data.ind: request message byte 3 indication.

7 Client_N_Data.req, Client_DL_Data.req: send checksum byte(CS) of request message.

8 Client_T_Data.con, Client_DL_Data.con, Client_N_Data.con: request message checksum byte (CS) confirmation.

Start $P2_{Client}$ timer.

Server_DL_Data.ind, Server_N_Data.ind, Server_T_Data: request message checksum byte (CS) indication.
Start $P2_{Server}$ timer.

9 Sender P4 timeout expired.

10 Server_T_Data.req, Server_N_Data.req, Server_DL_Data.req: send byte 1 of response message.
Stop $P2_{Server}$ timer.

11 Client_T_DataSOM.ind, Client_N_DataFB.ind, Client_DL_DataFB.ind: response message byte 1 indication and start of message indication.

Stop $P2_{Client}$ timer.

12 Server_N_Data.req, Server_DL_Data.req: send byte 2 of response message.

13 Client_DL_DataFB.ind, Client_N_DataFB.ind: response message byte 2 indication.

14 Server_N_Data.req, Server_DL_Data.req: send byte 3 of response message.

15 Client_DL_DataFB.ind, Client_N_DataFB.ind: response message byte 3 indication.

16 Server_N_Data.req, Server_DL_Data.req: send checksum byte (CS) of response message.

17 Client_DL_DataFB.ind, Client_N_DataFB.ind: response message checksum byte (CS) indication.

Start $P2_{Client}$ timer.

18 Receiver P1 timeout expired.

19 $P2_{Client}$ timeout expired.

Figure 14 — Data link layer timing at T-Data interface

11 Communication services

11.1 StartCommunication service

11.1.1 Service definition

11.1.1.1 Service purpose

The purpose of this communication layer service is to establish communication to one or more servers after successful transmission of the WuP (see 8.3.3).

11.1.1.2 Service procedure

Upon reception of a StartCommunication request service from the client, the server shall respond either with a positive or negative response message. Any positive response message shall contain the appropriate key bytes for the header format and timing valid for this server. The first message of a FAST_INIT always uses a header with target and source address and without additional length byte. A server may answer back with or without address information and length byte and tells its supported mode within the key bytes.

The negative response message shall not include the key bytes. For ISO 14229-6 implementations, the negative response codes are defined in ISO 14229-1, and for Keyword Protocol 2000 implementations, the negative response codes are defined in ISO 14230-3.

11.1.2 Implementation

Table 20 defines the StartCommunication request message implementation.

Table 20 — StartCommunication request message implementation

Byte	Parameter name	Cvt	Value	Mnemonic
1	Format byte	M	0xXX	FMT
2	Target address byte	C [1]	0xXX	TGT
3	Source address byte	C [1]	0xXX	SRC
4	Additional length byte	C [2]	0xXX	LEN
5	StartCommunication Request Service Identification	M	0x81	STC
6	Checksum	M	0xXX	CS

[1] The presence of the target and source address byte depends on the content of the format byte: '10xx xxxx$_b$' or '11xx xxxx$_b$'.

[2] The presence of the additional length byte depends on the content of the format byte: 'xx00 0000$_b$'.

Table 21 defines the StartCommunication positive response message implementation.

Table 21 — StartCommunication positive response message implementation

Byte	Parameter name	Cvt	Value	Mnemonic
1	Format byte	M	0xXX	FMT
2	Target address byte	C 1	0xXX	TGT
3	Source address byte	C 1	0xXX	SRC
4	Additional length byte	C 2	0xXX	LEN
5	StartCommunication Positive Response Service Identification	M	0xC1	STCPR
6	Key byte 1 (see 8.4) Key byte 2	M	0xXX 0xXX	KB1 KB2
7	Checksum	M	0xXX	CS

1 The presence of the target and source address byte depends on the content of the format byte: '10xx xxxx$_b$' or '11xx xxxx$_b$'.

2 The presence of the additional length byte depends on the content of the format byte: 'xx00 0000$_b$'.

Table 22 defines the StartCommunication negative response message implementation.

Table 22 — StartCommunication negative response message implementation

Byte	Parameter name	Cvt	Value	Mnemonic
1	Format byte	M	0xXX	FMT
2	Target address byte	C 1	0xXX	TGT
3	Source address byte	C 1	0xXX	SRC
4	Additional length byte	C 2	0xXX	LEN
5	Negative Response Service Identification	M	0x7F	STCNR
6	StartCommunication Request Service Identification	M	0x81	STC
7	ResponseCode [a] = generalReject	M	0x10	RC
8	Checksum	M	0xXX	CS

a For ISO 14229-6 implementations, other negative response codes are possible and are defined in ISO 14229-1. For other implementations, the negative response codes apply as defined in ISO 14230-3.

1 The presence of the target and source address byte depends on the content of the format byte: '10xx xxxx$_b$' or '11xx xxxx$_b$'.

2 The presence of the additional length byte depends on the content of the format byte: 'xx00 0000$_b$'.

11.2 StopCommunication service

11.2.1 Service definition

11.2.1.1 Service purpose

The purpose of this communication layer service is to terminate a diagnostic communication.

11.2.1.2 Service procedure

Upon receiving a StopCommunication indication primitive, the server (ECU) shall check if the current conditions allow terminating the communication. In this case, the server shall perform all actions necessary to terminate this communication.

If it is possible to terminate the communication, the server (ECU) shall issue a StopCommunication response primitive with the positive response parameters selected before the communication is terminated. If the communication cannot be terminated by any reason, the server shall issue a StopCommunication response primitive with the negative response parameter selected.

11.2.2 Implementation

Table 23 defines the StopCommunication request message implementation.

Table 23 — StopCommunication request message implementation

Byte	Parameter name	Cvt	Value	Mnemonic
1	Format byte	M	0xXX	FMT
2	Target address byte	C[1]	0xXX	TGT
3	Source address byte	C[1]	0xXX	SRC
4	Additional length byte	C[2]	0xXX	LEN
5	StopCommunication Request Service Identification	M	0x82	SPC
6	Checksum	M	0xXX	CS

[1] The presence of the target and source address byte depends on the content of the format byte: '10xx xxxx$_b$' or '11xx xxxx$_b$'.

[2] The presence of the additional length byte depends on the content of the format byte: 'xx00 0000$_b$'.

Table 24 defines the StopCommunication positive response message implementation.

Table 24 — StopCommunication positive response message implementation

Byte	Parameter name	Cvt	Value	Mnemonic
1	Format byte	M	0xXX	FMT
2	Target address byte	C[1]	0xXX	TGT
3	Source address byte	C[1]	0xXX	SRC
4	Additional length byte	C[2]	0xXX	LEN
5	StopCommunication Positive Response Service Identification	S	0xC2	SPCPR
6	Checksum	M	0xXX	CS

[1] The presence of the target and source address byte depends on the content of the format byte: '10xx xxxx$_b$' or '11xx xxxx$_b$'.

[2] The presence of the additional length byte depends on the content of the format byte: 'xx00 0000$_b$'.

Table 25 defines the StopCommunication negative response message implementation.

Table 25 — StopCommunication negative response message implementation

Byte	Parameter name	Cvt	Value	Mnemonic
1	Format byte	M	0xXX	FMT
2	Target address byte	C[1]	0xXX	TGT
3	Source address byte	C[1]	0xXX	SRC
4	Additional length byte	C[2]	0xXX	LEN
5	Negative Response Service Identification	M	0x7F	SPCNR
6	StopCommunication Request Service Identification	M	0x82	SPC
7	ResponseCode[a] = generalReject	M	0x10	RC
8	Checksum	M	0xXX	CS

[a] For ISO 14229-6 implementations, other negative response codes are possible and are defined in ISO 14229-1. For other implementations, the negative response codes apply as defined in ISO 14230-3.

[1] The presence of the target and source address byte depends on the content of the format byte: '10xx xxxx$_b$' or '11xx xxxx$_b$'.

[2] The presence of the additional length byte depends on the content of the format byte: 'xx00 0000$_b$'.

11.3 AccessTimingParameter service

11.3.1 Service definition

11.3.1.1 Service purpose

The purpose of this communication layer service is to read and to change the default timing parameters of a communication link for the duration this communication link is active.

NOTE The use of this service is complex and depends on the server's (ECU's) capability and the physical topology in the vehicle. The user of this service is responsible for proper communication on the K-line with other servers (ECUs).

11.3.1.2 Service procedure

This procedure has four different modes:

— readLimitsOfPossibleTimingParameters;

— setTimingParametersToDefaultValues;

— readCurrentlyActiveTimingParameters;

— setTimingParametersToGivenValues.

Upon receiving an AccessTimingParameter indication primitive with TPI = 00 (see 11.3.2), the server (ECU) shall read the timing parameter limits, i.e. the value that the server (ECU) is capable of supporting. If the read access to the timing parameters is successful, the server (ECU) shall send an AccessTimingParameter response primitive with the positive response parameters. If the read access to the timing parameters is not successful, the server (ECU) shall send an AccessTimingParameter response primitive with the negative response parameters.

Upon receiving an AccessTimingParameter indication primitive with TPI = 01, the server shall change all timing parameters to the default values and send an AccessTimingParameter response primitive with the positive response parameters before the default timing parameters become active. If the timing parameters cannot be changed to default values for any reason, the server (ECU) shall maintain the communication link and send an AccessTimingParameter response primitive with the negative response parameters.

Upon receiving an AccessTimingParameter indication primitive with TPI = 10, the server (ECU) shall read the currently used timing parameters. If the read access to the timing parameters is successful, the server (ECU) shall send an AccessTimingParameter response primitive with the positive response parameters.

If the read access to the currently used timing parameters is impossible for any reason, the server (ECU) shall send an AccessTimingParameter response primitive with the negative response parameters.

Upon receiving an AccessTimingParameter indication primitive with TPI = 11, the server (ECU) shall check if the timing parameters can be changed under the present conditions. If the conditions are valid, the server (ECU) shall perform all actions necessary to change the timing parameters and send an AccessTimingParameter response primitive with the positive response parameters before the new timing parameter limits become active.

If the timing parameters cannot be changed by any reason, the server (ECU) shall maintain the communication link and send an AccessTimingParameter response primitive with the negative response parameters.

11.3.2 Implementation

Selection of mode (read/write/current/limits) is effected by the TimingParameterIdentifier (TPI).

Table 26 defines the TimingParameterIdentifier values implementation.

Table 26 — TimingParameterIdentifier values implementation

Value	Description	Mnemonic
0x00	readLimitsOfPossibleTimingParameter	RLOPTP
0x01	setTimingParametersToDefaultValues	STPTDV
0x02	readCurrentlyActiveTimingParameters	RCATP
0x03	setTimingParametersToGivenValues	STPTGV
0x04 - 0xFF	reservedByDocument	RBD

Table 27 defines the AccessTimingParameter request message implementation.

Table 27 — AccessTimingParameter request message implementation

Byte	Parameter name	Cvt	Value	Mnemonic
1	Format byte	M	0xXX	FMT
2	Target address byte	C [1]	0xXX	TGT
3	Source address byte	C [1]	0xXX	SRC
4	Additional length byte	C [2]	0xXX	LEN
5	AccessTimingParameter Request Service Identification	M	0x83	ATP
6	TimingParameterIdentifier = [readLimitsOfPossibleTimingParameter, setTimingParametersToDefaultValues, readCurrentlyActiveTimingParameters, setTimingParametersToGivenValues]	M	0xXX =[0x00, 0x01, 0x02, 0x03]	TPI_ RLOPTP STPTDV RCATP STPTGV
7 8 9 10 11	$P2_{Server\_min}$ $P2_{Server\_max}$ $P3_{Client\_min}$ $P3_{Client\_max}$ $P4_{Sender\_min}$	C [3] C [3] C [3] C [3] C [3]	0xXX : : : 0xXX	P2SERVERMIN P2SERVERMAX P3CLIENTMIN P3CLIENTMAX P4SENDERMIN
12	Checksum	M	0xXX	CS

[1] The presence of the target and source address byte depends on the content of the format byte: '10xx xxxx$_b$' or '11xx xxxx$_b$'.

[2] The presence of the additional length byte depends on the content of the format byte: 'xx00 0000$_b$'.

[3] The presence of timing parameter bytes depends on the value of the TimingParameterIdentifier byte: TPI = setTimingParametersToGivenValues.

Table 28 defines the AccessTimingParameter positive response message implementation.

Table 28 — AccessTimingParameter positive response message implementation

Byte	Parameter name	Cvt	Value	Mnemonic
1	Format byte	M	0xXX	FMT
2	Target address byte	C [1]	0xXX	TGT
3	Source address byte	C [1]	0xXX	SRC
4	Additional length byte	C [2]	0xXX	LEN
5	AccessTimingParameter Positive Response Service Identification	M	0xC3	ATPPR
6	TimingParameterIdentifier = [readLimitsOfPossibleTimingParameter, setTimingParametersToDefaultValues, readCurrentlyActiveTimingParameters, setTimingParametersToGivenValues]	M	0xXX=[0x00, 0x01, 0x02, 0x03]	TPI_ RLOPTP STPTDV RCATP STPTGV
7	$P2_{Server\_min}$	C [3]	0xXX:	P2SERVERMIN
8	$P2_{Server\_max}$	C [3]	:	P2SERVERMAX
9	$P3_{Client\_min}$	C [3]	:	P3CLIENTMIN
10	$P3_{Client\_max}$	C [3]	:	P3CLIENTMAX
11	$P4_{Sender\_min}$	C [3]	0xXX	P4SENDERMIN
12	Checksum	M	0xXX	CS

[1] The presence of the target and source address byte depends on the content of the format byte: '10xx xxxx$_b$' or '11xx xxxx$_b$'.

[2] The presence of the additional length byte depends on the content of the format byte: 'xx00 0000$_b$'.

[3] The presence of timing parameter bytes depends on the value of the TimingParameterIdentifier byte: TPI = readLimitsOfPossibleTimingParameter, readCurrentlyActiveTimingParameters.

Table 29 defines the AccessTimingParameter negative response message implementation.

Table 29 — AccessTimingParameter negative response message implementation

Byte	Parameter name	Cvt	Value	Mnemonic
1	Format byte	M	0xXX	FMT
2	Target address byte	C [1]	0xXX	TGT
3	Source address byte	C [1]	0xXX	SRC
4	Additional length byte	C [2]	0xXX	LEN
5	Negative Response Service Identification	M	0x7F	ATPNR
6	AccessTimingParameter Request Service Identification	M	0x83	ATP
7	ResponseCode [a] = generalReject	M	0x10	RC
8	Checksum	M	0xXX	CS

[a] For ISO 14229-6 implementations, other negative response codes are possible and are defined in ISO 14229-1. For other implementations, the negative response codes apply as defined in ISO 14230-3.

[1] The presence of the target and source address byte depends on the content of the format byte: '10xx xxxx$_b$' or '11xx xxxx$_b$'.

[2] The presence of the additional length byte depends on the content of the format byte: 'xx00 0000$_b$'.