INTERNATIONAL STANDARD

ISO 14229-2

Second edition
2021-10

Road vehicles — Unified diagnostic services (UDS) —

Part 2: Session layer services

Véhicules routiers — Services de diagnostic unifiés (SDU) —

Partie 2: Services de la couche session

Reference number
ISO 14229-2:2021(E)

COPYRIGHT PROTECTED DOCUMENT

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: 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.

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 of the voluntary nature of standards, 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 www.iso.org/iso/foreword.html.

This document was prepared by Technical Committee ISO/TC 22, *Road vehicles*, Subcommittee SC 31, *Data communication*.

This second edition cancels and replaces the first edition (ISO 14229-2:2013), which has been technically revised.

The main changes are as follows:

— restructuration of the document;

— introduction of requirement numbers and names;

— technical content improvements based on implementation feedback from the automotive industry.

A list of all parts in the ISO 14229 series can be found on the ISO website.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

Introduction

The ISO 14229 series has been established in order to define common requirements for diagnostic systems, whatever the serial data link is.

To achieve this, the ISO 14229 series is based on the Open Systems Interconnection (OSI) Basic Reference Model in accordance with ISO/IEC 7498-1 and ISO/IEC 10731,[1] which structures communication systems into seven layers. When mapped on this model, the services used by a diagnostic tester (client) and an Electronic Control Unit (ECU, server) are structured into the following layers:

— application layer (layer 7) specified in ISO 14229-1;

— presentation layer (layer 6) specified in ISO 14229-1;

— session layer services (layer 5) specified in this document (ISO 14229-2).

Figure 1 illustrates the ISO 14229 series reference according to OSI model.

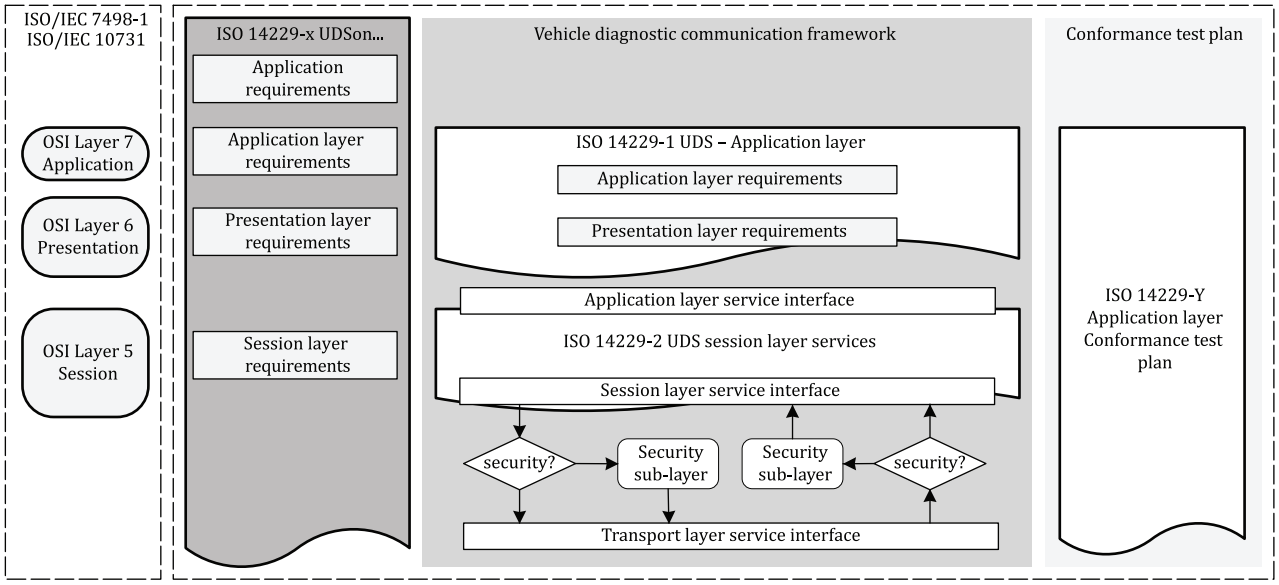


Figure 1 — ISO 14229 series reference according to OSI model

Road vehicles — Unified diagnostic services (UDS) —

Part 2:
Session layer services

1 Scope

This document specifies common session layer services and requirements to provide independence between unified diagnostic services (ISO 14229-1) and all transport protocols and network layer services (e.g. ISO 13400-2 DoIP, ISO 15765-2 DoCAN, ISO 10681-2 communication on FlexRay, ISO 14230-2 DoK-Line, and ISO 20794-3 CXPI).

This document specifies a common service primitive interface between OSI layer 5 (session) and layer 4 (transport) via so-called service request/indication/confirmation primitives.

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/IEC 7498-1, *Information technology — Open Systems Interconnection — Basic Reference Model: The Basic Model*

ISO 14229-1, *Road vehicles — Unified diagnostic services (UDS) — Part 1: Application layer*

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO 14229-1, ISO/IEC 7498-1 and the following apply.

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

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

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

3.1
gateway
networking device that transfers the PDU on different OSI layers

EXAMPLE A network device that enables communication between control module networks that uses different communication protocols, different communication rates, etc. and that includes, but is not limited to, gateway functionalities like bridge, *switch* (3.3), *router* (3.2) or application layer routing.

3.2
router
networking device that transfers the PDU on OSI layers 3 and 4

3.3
switch
networking device that transfers the PDU on OSI layer 2

4 Symbols and abbreviated terms

4.1 Symbols

—	empty cell/undefined
t	time

4.2 Abbreviated terms

Diag	diagnostics
ECU	electronic control unit
N/A	not applicable
OSI	open systems interconnection
RDiag	remote diagnostics
S_AE	session layer address extension
S_Data	session layer data transfer service name
S_Length	session layer length of data
S_Mtype	session layer message type
S_PDU	session layer protocol data unit
S_SA	session layer source address
S_TA	session layer target address
S_TAtype	session layer target address type
SecureDiag	secure diagnostics
SecureRDiag	secure remote diagnostics
SI	service identifier
SOM	start of message
SPP	service primitive parameter

5 Conventions

This document is based on the OSI service conventions as specified in ISO/IEC 10731[1].

Annex B describes vehicle diagnostic OSI layer architecture examples.

6 Session layer services

6.1 Service interface

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

The service primitives define how a service user (e.g. diagnostic application) cooperates with a service provider (e.g. session layer). To define the services, three types of service primitives are specified:

— a service request primitive `S_Data.request`, used by the higher application layer to pass control information or data required to be transmitted to the session layer (i.e. the service provider is being requested by the service user to process control information or to transmit data);

— a service indication primitive `S_Data.indication`, used by the session layer to pass status information and received data to the higher application layer (i.e. the service user is being informed by the service provider about an internal event of the session layer or the service request of a peer protocol layer entity service user);

— a service confirmation primitive `S_Data.confirm` used by the session layer to pass status information to the application layer (i.e. the service user is being informed by service provider about the result of a preceding service request of the service user).

6.2 Service interface parameters

The session layer services have the same general format. Service primitives are written in the form:

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

where

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

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

— "parameter A, ..." is the S_PDU (session layer protocol data unit) as a list of values passed by the service primitive (e.g. addressing information, data, length, result),

— "parameter A, parameter B, parameter C" are mandatory parameters that are included in all service calls, "[parameter X]" is an optional parameter that is included if specific conditions are fulfilled.

6.3 Service interface primitives

Figure 2 shows the session layer service primitives of a message transmission with a `T_Data.ind` reception at the session layer of the receiver side from the lower OSI layer.

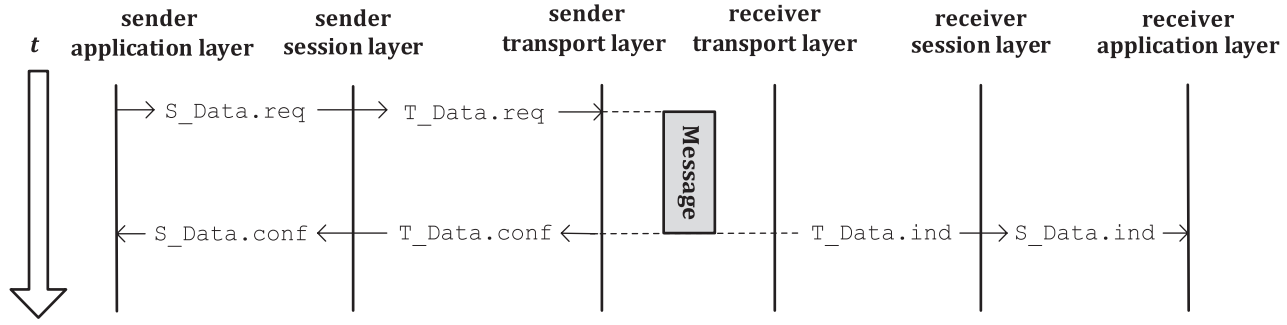
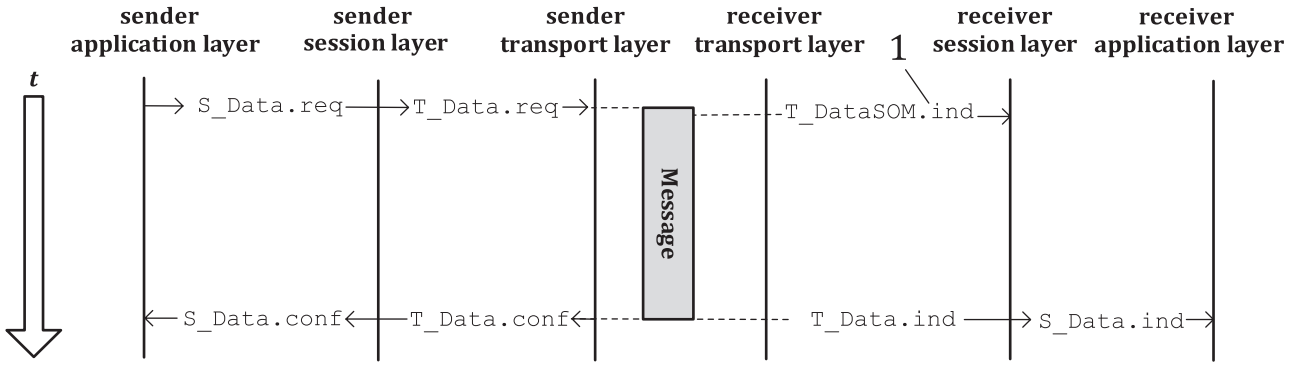


Figure 2 — Session layer service primitives – T_Data.ind message reception

Figure 3 shows the session layer service primitives of a message transmission with a `T_DataSOM.ind` reception at the beginning of the message and a `T_Data.ind` reception at the end of the message at the session layer of the receiver side from the lower OSI layer.



Key

1 transport layer StartOfMessage data indication, e.g. ISO 15765-2

Figure 3 — Session layer service primitives – T_DataSOM.ind and T_Data.ind reception

The following communication scenarios are distinguished:

a) physical communication during

   1) default session, and

   2) non-default session — session handling required;

b) functional communication during

   1) default session, and

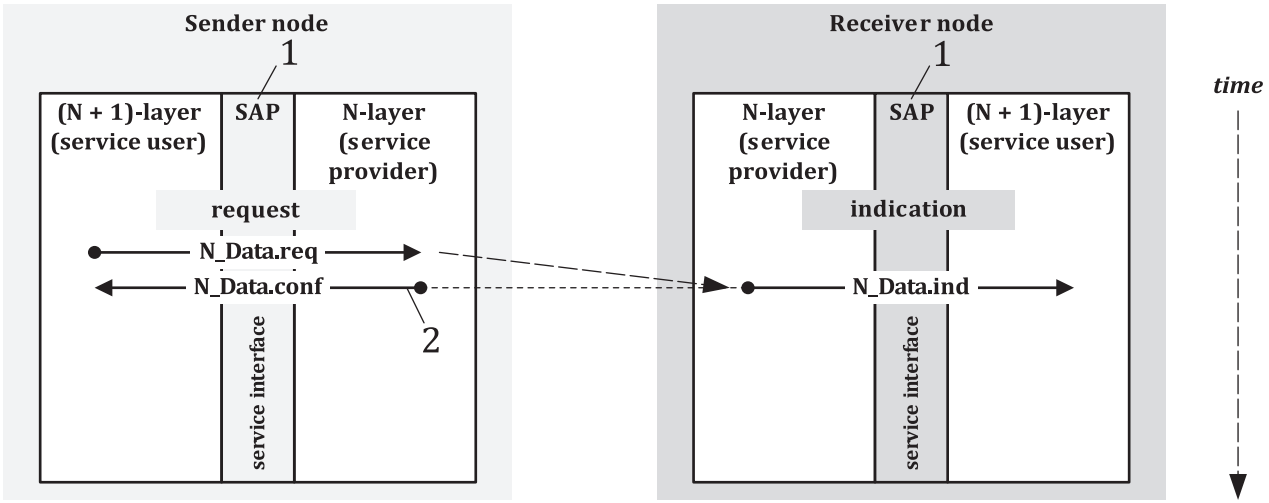
   2) non-default session — session handling required.

7 Service interface (SI) definition from application layer to session layer

7.1 SI — S_Data.req, S_Data.ind, and S_Data.conf service interface

The service interface defines the service and parameter mapping from the application layer to the session layer.

Figure 4 shows the `S_Data.req`, `S_Data.ind`, and `S_Data.conf` service interface.



Key

1 service access point

2 read back from N-layer service provider

Figure 4 — S_Data.req and S_Data.ind service interface

7.2 SI — S_Data.req, S_Data.ind, and S_Data.conf service interface parameter mapping

This requirement specifies the application service interface and parameter mapping between the session layer and the lower OSI layers.

REQ	0.1 SI — S_Data.req, S_Data.ind, and S_Data.conf service interface parameter mapping
The `S_Data.req`, `S_Data.ind`, and `S_Data.conf` service interface parameter mapping shall be implemented as specified in Table 1.	

Table 1 — S_Data.req and S_Data.ind service interface parameter mapping

Application layer (service user)	Session layer (service provider)	A_Data.req, A_Data.ind and A_Data.conf parameter validity		
		.req	.ind	.conf
A_Mtype	S_Mtype	X	X	X
A_AI[TAtype]	S_AI[TAtype]	X	X	X
A_AI[TA]	S_AI[TA]	X	X	X
A_AI[SA]	S_AI[SA]	X	X	X
A_AI[AE]	S_AI[AE]	X	X	X
A_Length	S_Length	X	X	—
A_Data	S_Data	X	X	—
A_Result	S_Result	—	X	X
Key X = supported — = not supported				

7.3 SI — S_PDU mapping onto T_PDU and vice versa for message transmission

The parameters of the session layer protocol data unit defined to request the transmission of a diagnostic service request/response are mapped onto the parameters of the transport layer protocol

data unit for the transmission of a message in the client/server. Annex A specifies the T_PDU interface and shall be followed.

The parameters of the transport layer protocol data unit defined for the reception of a message are mapped as follows onto the parameters of the session layer protocol data unit for the confirmation/ indication of the reception of a diagnostic response/request.

The transport layer confirmation of the successful transmission of the message (`T_Data.conf`) is forwarded to the application, because it is needed in the application for starting those actions, which shall be executed immediately after the transmission of the request/response message (e.g. ECUReset, bit rate change).

The transport layer indication for the reception of a StartOfMessage T_PDU (`T_DataSOM.ind`), e.g. ISO 15765-2 is not forwarded to the application layer, because it is only used within the session layer to perform the session layer timing (see Clause 9). Therefore, no mapping of the `T_DataSOM.ind` T_PDU onto an S_PDU is defined.

Table 2 defines the mapping of session layer S_PDU onto transport layer T_PDU and vice versa.

Table 2 — Mapping of session layer S_PDU onto transport layer T_PDU and vice versa

S_PDU parameter (session layer protocol data unit)	Description	T_PDU parameter (transport layer protocol data unit)	Description
`S_Mtype`	Session layer message type	`T_Ptype`	Transport layer segment type
`S_AI[TAtype]`	Session layer target address type	`T_AI[TAtype]`	Transport layer target address type
`S_AI[SA]`	Session layer source address	`T_AI[SA]`	Transport layer source address
`S_AI[TA]`	Session layer target address	`T_AI[TA]`	Transport layer target address
`S_AI[AE]`[a]	Session layer address extension	`T_AI[AE]`[a]	Transport layer address extension
`S_Data[1]` – `S_Data[n]`	Session layer data	`T_Data[1]` – `T_Data[n]`	Transport layer data
`S_Length`	Session layer data length	`T_Length`	Transport layer data length
`S_Result`	Session layer result	`T_Result`	Transport layer result

[a] If `Mtype` = diagnostics/secure diagnostics, then the address information shall consist of the parameters `SA`, `TA` and `TAtype`.

If `Mtype` = remote diagnostics/secure remote diagnostics, then the address information shall consist of the parameters `SA`, `TA`, `TAtype`, and `AE`.

7.4 SI — S_Data.req

The service primitive requests transmission of `S_Data` with `S_Length` number of bytes from the sender to the receiver peer entities identified by the address information in `S_AI[TAtype]`, `S_AI[SA]`, `S_AI[TA]`, and `S_AI[AE]`.

If the `S_Data.req` service is called, the session layer signals the completion (or failure) of the message transmission to the service user by means of the issuing of an `S_Data.conf` service call.

```
S_Data.req    (
              S_Mtype,
              S_AI[TAtype],
              S_AI[SA],
              S_AI[TA],
              [S_AI[AE]],
              S_Data[Data#1, Data#2, …, Data#n],
              S_Length
              )
```

7.5 SI — S_Data.ind

The S_Data.indication service is issued by the session layer. The service primitive shall indicate S_Result events and delivers S_Data with S_Length bytes received from a peer protocol entity identified by the address information in S_AI[TAtype], S_AI[SA], S_AI[TA], and S_AI[AE] to the adjacent upper layer. The parameters S_Data and S_Length shall only be valid if S_Result equals S_OK.

```
S_Data.ind    (
              S_Mtype,
              S_AI[TAtype],
              S_AI[SA],
              S_AI[TA],
              [S_AI[AE]],
              S_Data[Data#1, Data#2, …, Data#n],
              S_Length,
              S_Result
             )
```

7.6 SI — S_Data.conf

The S_Data.conf service is issued by the session layer. The service primitive confirms the completion of an S_Data.req service identified by the address information in S_AI[TAtype], S_AI[SA], S_AI[TA], and S_AI[AE]. The parameter S_Result provides the status of the service request.

```
S_Data.conf   (
              S_Mtype,
              S_AI[TAtype],
              S_AI[SA],
              S_AI[TA],
              [S_AI[AE]],
              S_Result
              )
```

8 Service primitive parameters (SPP)

8.1 SPP – General

Clause 8 specifies the service primitive parameters and data types, which are used by the application layer services.

8.2 SPP – Data type definitions

REQ	0.2 SPP – Data type definitions
The data types shall be in accordance to: — Enum = 8-bit enumeration, — Unsigned Byte = 8-bit unsigned numeric value, — Unsigned Word = 16-bit unsigned numeric value, — Unsigned Long = 32-bit unsigned numeric value, — Byte Array = sequence of 8-bit aligned data, — Bit String = 8-bit binary coded.	

8.3 SPP – S_Mtype, session layer message type

The parameter S_Mtype is used to identify the type and range of address information parameters included in a service call. This document specifies a range of two values for this parameter.

REQ	0.3 SPP – S_Mtype, session layer message type
The S_Mtype parameter shall be of data type Enum and shall be used to identify the message type and range of address information included in a service call. — If S_Mtype = diagnostics (Diag)/secure diagnostics (SecureDiag), then the address information shall consist of the parameters S_SA, S_TA and S_TAtype. — If S_Mtype = remote diagnostics (RDiag)/secure remote diagnostics (SecureRDiag), then the address information shall consist of the parameters S_SA, S_TA, S_TAtype and S_AE. Range: [Diag, RDiag, SecureDiag, SecureRDiag]	

8.4 SPP – S_TAtype, session layer target address type

The parameter S_TAtype is a configuration attribute to the S_TA parameter. It is used to encode the communication model used by the communicating peer entities. Two communication models are specified: '1 to 1' communication, called physical addressing, and '1 to n' communication, called functional addressing.

REQ	0.4 SPP – S_TAtype, session layer target address type
The S_TAtype parameter shall be of data type Enum and shall be used to identify the target address type to be used with the request address. Range: [physical, functional]	

8.5 SPP – S_TA, session layer target address

S_TA parameter is used to encode the receiving session layer protocol entity. The parameter S_TA is used to encode client and server identifiers.

REQ	0.5 SPP – S_TA, session layer target address
The S_TA parameter shall be of data type Unsigned Word and shall contain the target address of the node. Range: [0000_{16} to $FFFF_{16}$]	

8.6 SPP – S_SA, session layer source address

S_SA parameter is used to encode the sending session layer protocol entity. The parameter S_SA is used to encode client and server identifiers.

REQ	0.6 SPP – S_SA, session layer source address
The S_SA parameter shall be of data type Unsigned Word and shall contain the source address of the node. Range: [0000_{16} to $FFFF_{16}$]	

8.7 SPP – S_AE, session layer address extension

S_AE parameter is used to encode the sending session layer protocol entity. The parameter S_AE is used to encode client and server identifiers.

REQ	0.7 SPP – S_AE, session layer address extension
The S_AE parameter shall be of data type Unsigned Word and shall contain the extended address of the node. Range: [0000_{16} to $FFFF_{16}$]	

8.8 SPP – S_Length, session layer length of S_Data

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

REQ	0.8 SPP – S_Length, session layer length of S_Data
The S_Length parameter shall be of data type Unsigned Long and shall contain the length of the S_Data to be transmitted/received. Range: [0000 0000_{16} to FFFF $FFFF_{16}$]	

8.9 SPP – S_Data, session layer data of PDU

This parameter includes data to be exchanged by the higher OSI layer entities.

REQ	0.9 SPP – S_Data, session layer data of PDU
The S_Data parameter shall be of data type Byte Array and shall contain the message data content of the request or response message to be transmitted/received. Range: [00_{16} to FF_{16}]	

8.10 SPP – S_Result, session layer result

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

REQ	0.10 SPP – S_Result, session layer result
The S_Result parameter shall be of data type Enum and shall contain the status relating to the outcome of a service execution (request field and response field sequence). If two or more errors are discovered at the same time, then the application layer entity shall set the appropriate error bit in the Result parameter. Range: [OK, ERR_...] The result OK shall be issued to the service user when the service execution is successfully completed. The OK shall be issued to a service user on both, the sender and receiver side. The ERR_... shall be issued to the service user when an error is detected by a lower layer (provider). The ERR_... shall be issued to the service user on both, the sender and receiver side.	

9 Timing parameter definition

9.1 General application timing considerations

9.1.1 Server

REQ	5.1 Timing parameter definition – Server – t_{P2_Server}
A server shall use a single application timer (t_{P2_Server}) implementation, which is triggered (started and stopped) by the T_Data service primitive interface (T_Data.req, T_Data.conf, T_DataSOM.ind, T_Data.ind).	

REQ	5.2 Timing parameter definition – Server – $t_{P2\_Server\_Max}$/$t_{P2*\_Server\_Max}$
The t_{P2_Server} application timer shall be loaded with a $t_{P2\_Server\_Max}$/$t_{P2*\_Server\_Max}$ parameter value. Both parameters and values are specified in Table 3 and in Table 4.	

REQ	5.3 Timing parameter definition – Server – t_{P4_Server}
The parameter t_{P4_Server} is a performance requirement and shall be the time between the reception of a request (T_Data.ind) and the start of transmission of the final response (T_Data.req).	

The timing parameter t_{P4} is a performance parameter.

REQ	5.4 Timing parameter definition – Server – $t_{P4\_Server\_Max}$
The parameter $t_{P4\_Server\_Max}$ shall be the maximum value of t_{P4_Server} and if $t_{P4\_Server\_Max}$ is the same as $t_{P2\_Server\_Max}$, this shall be interpreted as that a negative response with negative response code 78_{16} "requestCorrectlyReceived-ResponsePending" is not allowed for the service in progress.	

REQ	5.5 Timing parameter definition – Server – Final response
A final response shall be a positive response or a negative response with a negative response code other than 78^{16} "requestCorrectlyReceived-ResponsePending".	

In case of a request to schedule periodic responses, the initial USDT positive or negative response that indicates the acceptance or non-acceptance of the request to schedule periodic responses is considered the final response.

REQ	5.6 Timing parameter definition – Server – Service not supported
Services not supported by the server shall utilize a $t_{P4\_Server\_Max}$ value equal to $t_{P2\_Server\_Max}$. A negative response with a negative response code 78_{16} "requestCorrectlyReceived-ResponsePending" shall not be allowed.	

9.1.2 Client

REQ	5.7 Timing parameter definition – Client – t_{P_Client}
A client shall use a single application timer (t_{P_Client}) implementation which is triggered (started, reloaded, and stopped) by the `T_Data` service primitive interface (`T_Data.req, T_Data.conf, T_DataSOM.ind, T_Data.ind`).	

REQ	5.8 Timing parameter definition – Client – $t_{P2\_Client\_Max}$/$t_{P2*\_Client\_Max}$
The t_{P_Client} application timer shall be loaded with a $t_{P2\_Client\_Max}$/$t_{P2*\_Client\_Max}$ for protocols, which support a `T_DataSOM.ind` service primitive (e.g. ISO 15765 DoCAN).	

REQ	5.9 Timing parameter definition – Client – t_{P_Client} start
The t_{P_Client} application timer shall be started, whenever the client application layer receives a `T_Data.conf` service primitive. Depending on the protocol type (with `T_DataSOM.ind` or without `T_DataSOM.ind`) the t_{P_Client} application timer shall be loaded with a $t_{P2\_Client\_Max}$ or a $t_{P6\_Client\_Max}$ parameter value.	

REQ	5.10 Timing parameter definition – Client – t_{P_Client} stop
Depending on the protocol type (with `T_DataSOM.ind` or without `T_DataSOM.ind`) the t_{P_Client} application timer shall be stopped, either when the `T_DataSOM.ind` or the `T_Data.ind` service primitive is received by the application.	

REQ	5.11 Timing parameter definition – Client – $t_{P6\_Client\_Max}$/$t_{P6*\_Client\_Max}$
The t_{P_Client} application timer shall be loaded with a $t_{P6\_Client\_Max}$/$t_{P6*\_Client\_Max}$ for protocols, which do not support a `T_DataSOM.ind` service primitive (e.g. ISO 13400 DoIP).	

REQ	5.12 Timing parameter definition – Client – Error condition detection
If no "`.ind`" is received while t_{P_Client} is smaller or equal to $t_{P2\_Client\_Max}$ an error condition shall be detected.	

REQ	5.13 Timing parameter definition – Client – Error indication to application layer
The error condition shall be flagged to the application layer with the parameters included in either T_DataSOM.ind or the T_Data.ind service primitive.	

REQ	5.14 Timing parameter definition – Client – Application layer protocols with support of T_DataSOM.ind
For application layer protocols, which support T_DataSOM.ind, the client application shall verify the correct application timing by comparing its actual t_{P_Client} application timer value with the $t_{P2\_Client\_Max}$ parameter value.	

If T_DataSOM.ind or T_Data.ind is received while t_{P_Client} is smaller or equal to $t_{P2\_Client\_Max}$ the timing fulfils the requirements established by this document.

REQ	5.15 Timing parameter definition – Client – Application layer protocols with no support of T_DataSOM.ind
For application layer protocols, which only support T_Data.ind, the client application shall verify the correct application timing by comparing its actual t_{P_Client} application timer value with the $t_{P6\_Client\_Max}$ parameter value.	

If T_Data.ind is received while t_{P_Client} is smaller or equal to $t_{P6\_Client\_Max}$ the timing fulfils the requirements established by this document.

REQ	5.16 Timing parameter definition – Client – Error condition detection and reporting
If no indication (.ind) is received while t_{P_Client} is smaller or equal to $t_{P6\_Client\_Max}$ an error condition shall be detected and shall be flagged to the application layer with the parameters included in the T_Data.ind service primitive.	

9.2 Application timing parameter definitions – defaultSession

REQ	5.17 Timing parameter definition – defaultSession start
A server shall start the defaultSession when powered up.	

REQ	5.18 Timing parameter definition – Timing parameter definition of defaultSession
The timing parameter definitions shall be in accordance with Table 3.	

Table 3 — Message timing parameter definitions for the defaultSession

Timing parameter	Definition	Type
Δt_{P2}	The Δt_{P2} parameter is defined to be the worst-case vehicle network design-dependent message transmission delay such as delays introduced by gateways and bus-load dependent arbitration. The value of Δt_{P2} is divided into the time to transmit the request to the addressed server/ECU ($\Delta t_{P2\_Request}$) and in case the protocol supports T_DataSOM.ind till the start of the response transmission indicated by T_DataSOM.ind or T_Data.ind if the response is a single frame message (e.g. ISO 15765 DoCAN).	Performance requirement

Table 3 *(continued)*

Timing parameter	Definition	Type
Δt_{P6}	The Δt_{P6} parameter is defined to be the worst-case vehicle network design-dependent message transmission delay such as delays introduced by gateways and bus-load dependent arbitration. The value of Δt_{P6} is divided into the time to transmit the request to the addressed server/ECU ($\Delta t_{P6\_Request}$) and the time to transmit the complete response to the client/tester ($\Delta t_{P6\_Response}$). The Δt_{P6} is independent of whether a protocol supports a `T_DataSOM.ind` (e.g. ISO 15765 DoCAN) or does not support a `T_DataSOM.ind` (e.g. ISO 13400 DoIP).	Performance requirement
t_{P2_Server}	Performance requirement for the server to start with the response message after the reception of a request message (indicated via `T_Data.ind`).	Performance requirement
t_{P2_Client}	Timeout for the client to wait after the successful transmission of a request message (indicated via `T_Data.conf`) for the start of incoming response messages (indicated via `T_DataSOM.ind` of a multi-frame message or `T_Data.ind` of a SingleFrame message).	Timer reload value
t_{P6_Client}	Timeout for the client to wait after the successful transmission of a request message (indicated via `T_Data.conf`) for the complete reception of the corresponding response message (indicated via `T_Data.ind`) e.g. ISO 13400.	Timer reload value
t_{P2*_Server}	Performance requirement for the server to start with the response message after the transmission of a negative response message (indicated via `T_Data.conf`) with negative response code 78_{16} (enhanced response timing).	Performance requirement
t_{P2*_Client}	Enhanced timeout for the client to wait after the reception of a negative response message with negative response code 78_{16} (indicated via `T_Data.ind`) for the start of incoming response messages (indicated via `T_DataSOM.ind` of a multi-frame message or `T_Data.ind` of a SingleFrame message).	Timer reload value
t_{P6*_Client}	Enhanced timeout for the client to wait after the reception of a negative response message with negative response code 78_{16} (indicated via `T_Data.ind`) for the complete reception of the corresponding response messages (indicated via `T_Data.ind`), e.g. ISO 13400 DoIP.	Timer reload value
$t_{P3\_Client\_Phys}$	Minimum time for the client to wait after the successful transmission of a physically-addressed request message (indicated via `T_Data.conf`) with no response required before it can transmit the next physically-addressed request message (see Figure 20).	Timer reload value
$t_{P3\_Client\_Func}$	Minimum time for the client to wait after the successful transmission of a functionally-addressed request message (indicated via `T_Data.conf`) before it can transmit the next functionally-addressed request message in case no response is required or the requested data are only supported by a subset of the functionally-addressed servers (see 10.3).	Timer reload value
t_{P4_Server}	This is the time between the reception of a request (`T_Data.ind`) and the start of the transmission of the final response (`T_Data.req`) at the server side.	Performance requirement

REQ	5.19 Timing parameter definition – New request message transmission
Each server/ECU shall be able to process a new request message immediately after the successful transmission of a response message (`T_Data.conf`) from the preceding request message.	

An exception to this requirement may be granted by the vehicle manufacturer for some use cases, where the server/ECU requires additional time after the execution of the previous service request, e.g. EcuReset service.

REQ	5.20 Timing parameter definition – Message timing parameter value specification of defaultSession
The timing parameter values for the defaultSession shall be in accordance with Table 4.	

Table 4 — Message timing parameter value specification for the defaultSession

Timing parameter	Minimum	Maximum
Δt_{P2}	0 ms	vehicle manufacturer specific value: see Formula (1)
Δt_{P6}	0 ms	vehicle manufacturer specific value: see Formula (2)
t_{P2_Server}	0 ms	server specific value: recommended value = 50 ms
t_{P2_Client} [a]	$t_{P2\_Server\_Max} + \Delta t_{P2\_Max}$	—
t_{P6_Client} [a]	$t_{P2\_Server\_Max} + \Delta t_{P6\_Max}$	—
t_{P2*_Server} [b]	0 ms	server specific value: recommended value = 5 000 ms
t_{P2*_Client} [c]	$t_{P2*\_Server\_Max} + \Delta t_{P2\_Response}$	—
t_{P6*_Client} [c]	$t_{P2*\_Server\_Max} + \Delta t_{P6\_Response}$	—
$t_{P3\_Client\_Phys}$ [d]	$t_{P2\_Server\_Max} + \Delta t_{P2\_Max}$	—
$t_{P3\_Client\_Func}$ [d]	$t_{P2\_Server\_Max} + \Delta t_{P2\_Max}$	—
$t_{P3\_Client\_Phys}$ [e]	$t_{P2\_Server\_Max} + \Delta t_{P6\_Max}$	—
$t_{P3\_Client\_Func}$ [e]	$t_{P2\_Server\_Max} + \Delta t_{P6\_Max}$	—
t_{P4_Server}	t_{P2_Server}	As defined by regulation for OBD diagnostic use cases. Vehicle manufacturer specific value for enhanced diagnostic use cases.

[a] The maximum time a client waits for a response message is at the discretion of the client, provided that t_{P2_Client}/t_{P6_Client} is greater than the specified minimum value of t_{P2_Client}/t_{P6_Client}.

[b] During the enhanced response timing, the minimum time between the transmission of consecutive negative messages (each with negative response code 78_{16}) shall be 0,3 × $t_{P2*\_Server\_Max}$, in order to avoid flooding the data link with unnecessary negative response code 78_{16} messages.

[c] The maximum value that a client uses for t_{P2*_Client}/t_{P6*_Client} is at the discretion of the client, provided it is greater than the specified minimum value of t_{P2*_Client}/t_{P6*_Client}.

[d] The maximum time a client waits until it transmits the next request message is at the discretion of the client, provided that for non-default sessions the t_{S3_Server} timing is kept active in the server(s).

[e] The maximum time a client waits until it transmits the next request message is at the discretion of the client, provided that for non-default sessions the t_{S3_Server} timing is kept active in the server(s).

REQ	5.21 Timing parameter definition – Δt_{P2}/Δt_{P6} definition
The parameters Δt_{P2}/Δt_{P6} shall be any system network design-dependent delays as introduced by gateways and bus bandwidth plus a safety margin (e.g. 50 % of worst-case).	

The worst-case scenario (transmission time necessary for one “round trip” from client to server and back from server to client) depends on system design and is impacted by:

— the number of gateways involved,

— frame transmission time (bit rate),

— bus utilization, and

— the device driver implementation method (polling vs. interrupt) and processing time of the transport layer.

The value of Δt_{P2} is the sum of the time to transmit the request message to the addressed server and the time to transmit the response message to the client.

REQ	5.22 Timing parameter definition – Δt_{P2} formula
Δt_{P2} shall be calculated in accordance with Formula (1).	

$$\Delta t_{P2} = \Delta t_{P2\_Request} + \Delta t_{P2\_Response} \quad (1)$$

where

Δt_{P2} is the sum of the time to transmit the request and the response message;

$\Delta t_{P2\_Request}$ is the time to transmit the request message;

$\Delta t_{P2\_Response}$ is the time to transmit the response message.

The value of Δt_{P6} is the sum of the time to transmit the request message to the addressed server and the time to transmit the response message to the client.

REQ	5.23 Timing parameter definition – Δt_{P6} formula
Δt_{P6} shall be calculated in accordance with Formula (2).	

$$\Delta t_{P6} = \Delta t_{P6\_Request} + \Delta t_{P6\_Response} \quad (2)$$

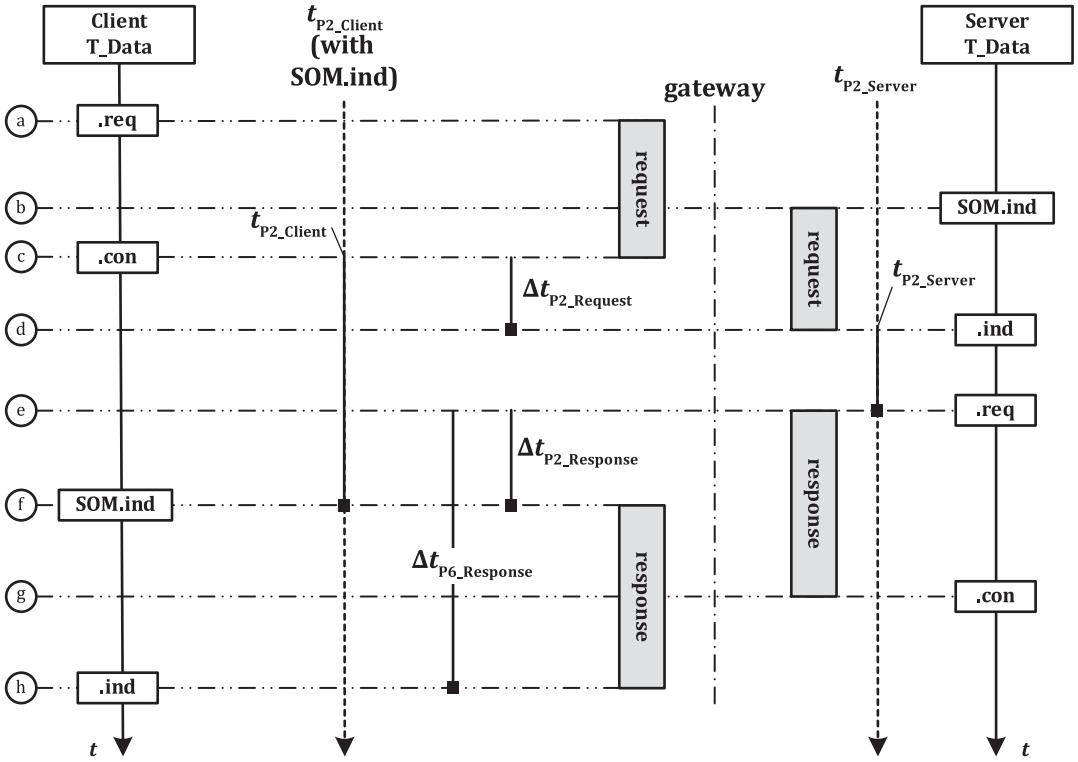
where

Δt_{P6} is the sum of the time to transmit the request and the response message;

$\Delta t_{P6\_Request}$ is the time to transmit the request message;

$\Delta t_{P6\_Response}$ is the time to transmit the response message.

Figure 5 shows an example of how Δt_{P2} can be composed.



Key

a Client `T_Data.req`: diagnostic application issues a request message to the transport layer.

b Server `T_DataSOM.ind`: transport layer issues to the diagnostic application the StartOfMessage of the request message.

c Client `T_Data.conf`: transport layer issues to the diagnostic application the confirmation of the completion of the request message. Client starts its t_{P_Client} timer using the default reload value $t_{P\_Client} = t_{P2\_Client\_Max}$. The value of the t_{P_Client} timer shall consider any latency that is involved based on the vehicle network design (communication over gateways, bus bandwidth, etc.).

d Server `T_Data.ind`: transport layer issues to the diagnostic application the completion of the request message. Server starts the t_{P2_Server} timer using the default value of $t_{P2\_Server} = t_{P2\_Server\_Max}$.

e Server `T_Data.req`: diagnostic application has prepared the response message and issues a `T_Data.req` to the transport layer within t_{P2_Server}. Server stops the t_{P2_Server} timer.

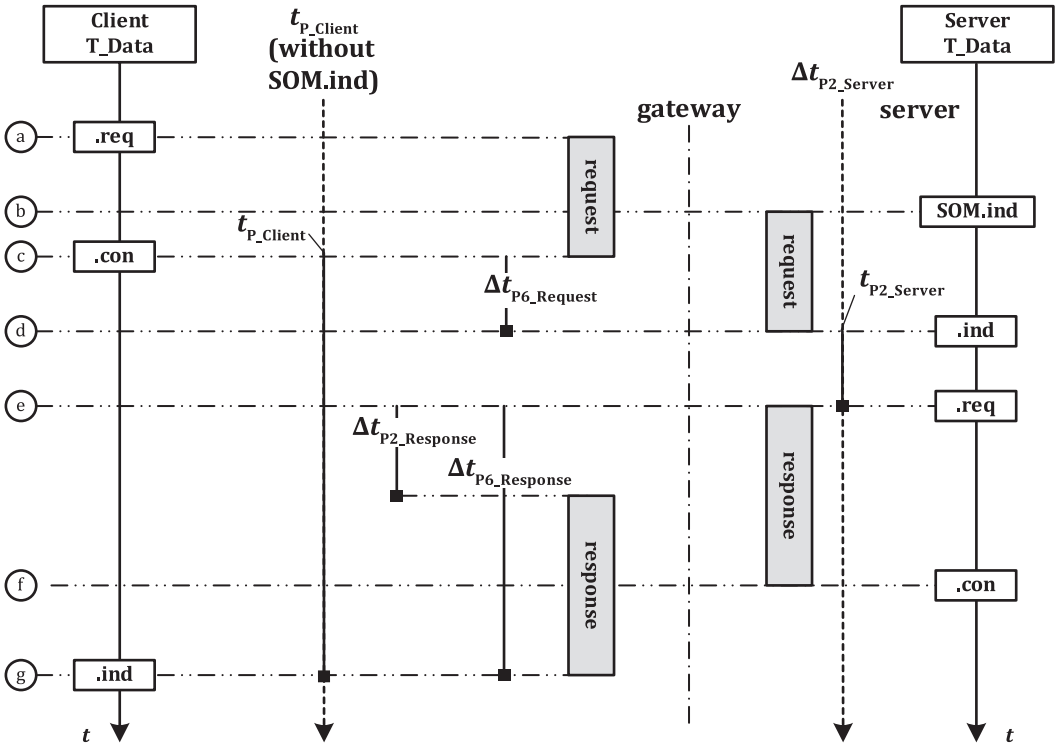
f Client `T_DataSOM.ind`: transport layer issues to the diagnostic application the StartOfMessage of the response message. Client stops its t_{P_Client} timer.

g Server `T_Data.conf`: transport layer issues to the diagnostic application the completion of the response message.

h Client `T_Data.ind`: transport layer issues a `T_Data.ind` to the diagnostic application to confirm the completion of the response message.

Figure 5 — Example for Δt_{P2} and Δt_{P6} – Response message with SOM.ind

Figure 6 shows an example of how Δt_{P6} can be composed.



Key

a Client `T_Data.req`: diagnostic application issues a request message to the transport layer.

b Server `T_DataSOM.ind`: transport layer issues to the diagnostic application the StartOfMessage of the request message.

c Client `T_Data.conf`: transport layer issues to the diagnostic application the confirmation of the completion of the request message. Client starts its t_{P_Client} timer using the default reload value $t_{P6\_Client} = t_{P6\_Client\_Max}$. The value of the t_{P_Client} timer shall consider any latency that is involved based on the vehicle network design (communication over gateways, bus bandwidth, etc.).

d Server `T_Data.ind`: transport layer issues to the diagnostic application the completion of the request message. Server starts the t_{P2_Server} timer using the default value of $t_{P2\_Server} = t_{P2\_Server\_Max}$.

e Server `T_Data.req`: diagnostic application has prepared the response message and issues a `T_Data.req` to the transport layer within t_{P2_Server}. Server stops the t_{P2_Server} timer.

f Server `T_Data.conf`: transport layer issues to the diagnostic application the completion of the response message.

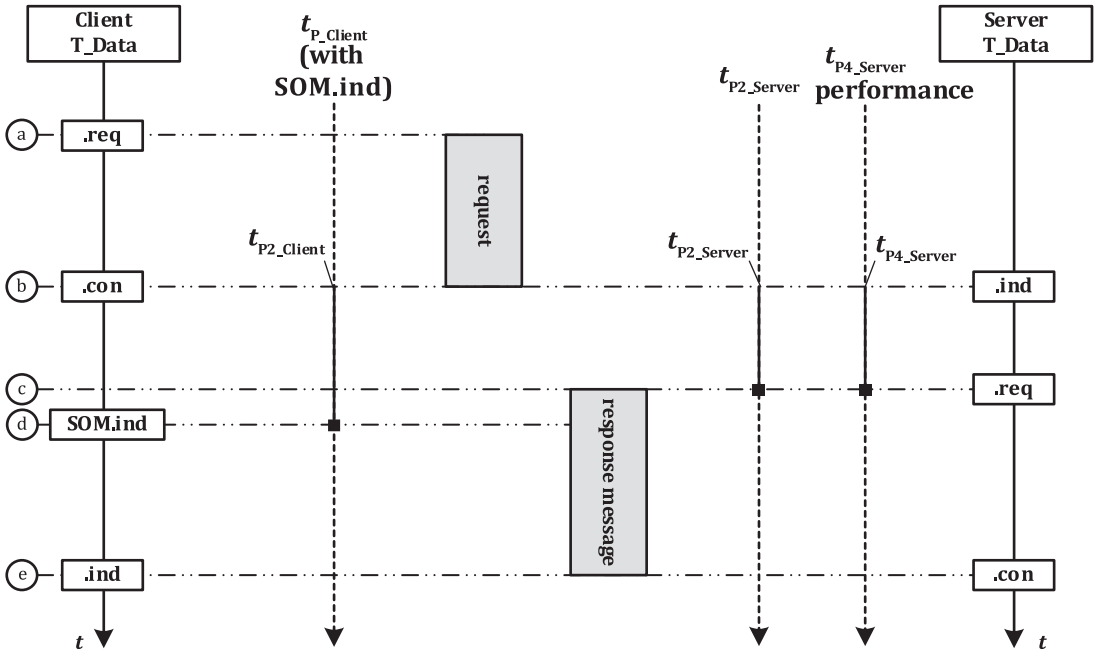
g Client `T_Data.ind`: transport layer issues a `T_Data.ind` to the diagnostic application to confirm the completion of the response message. Client stops its t_{P_Client} timer.

NOTE The client and the server are located on the same network. All descriptions and figures are presented in a time-related sequential order.

Figure 6 — Example for Δt_{P2} and Δt_{P6} – Response message without SOM.ind

9.3 Example for t_{P4_Server} without enhanced response timing

Figure 7 shows an example where $t_{P4\_Server\_Max} = t_{P2\_Server\_Max}$. In this scenario the server response performance timing parameter indicates that no negative responses including NRC 78_{16} are allowed.



Key

a Client `T_Data.req`: diagnostic application issues a request message to the transport layer.

b Server `T_Data.ind`: transport layer issues to the diagnostic application the completion of the request message. Server starts the t_{P2_Server} timer using the default value of $t_{P2\_Server} = t_{P2\_Server\_Max}$. If $t_{P4\_Server\_Max} = t_{P2\_Server\_Max}$ applies for a certain `T_Data.ind`, the server shall ensure that the final positive or negative response is started at the latest after t_{P2_Server} (i.e. no negative responses with NRC 78_{16} are allowed). Client `T_Data.conf`: transport layer issues to the diagnostic application the confirmation of the completion of the request message. Client starts its t_{P_Client} timer using the default reload value $t_{P2\_Client} = t_{P2\_Client\_Max}$. The value of the t_{P_Client} timer shall consider any latency that is involved based on the vehicle network design (communication over gateways, bus bandwidth, etc.).

c Server `T_Data.req`: diagnostic application has prepared the response message and issues a `T_Data.req` to the transport layer within t_{P2_Server}. Server stops the t_{P2_Server} timer. The t_{P4_Server} performance timer is stopped.

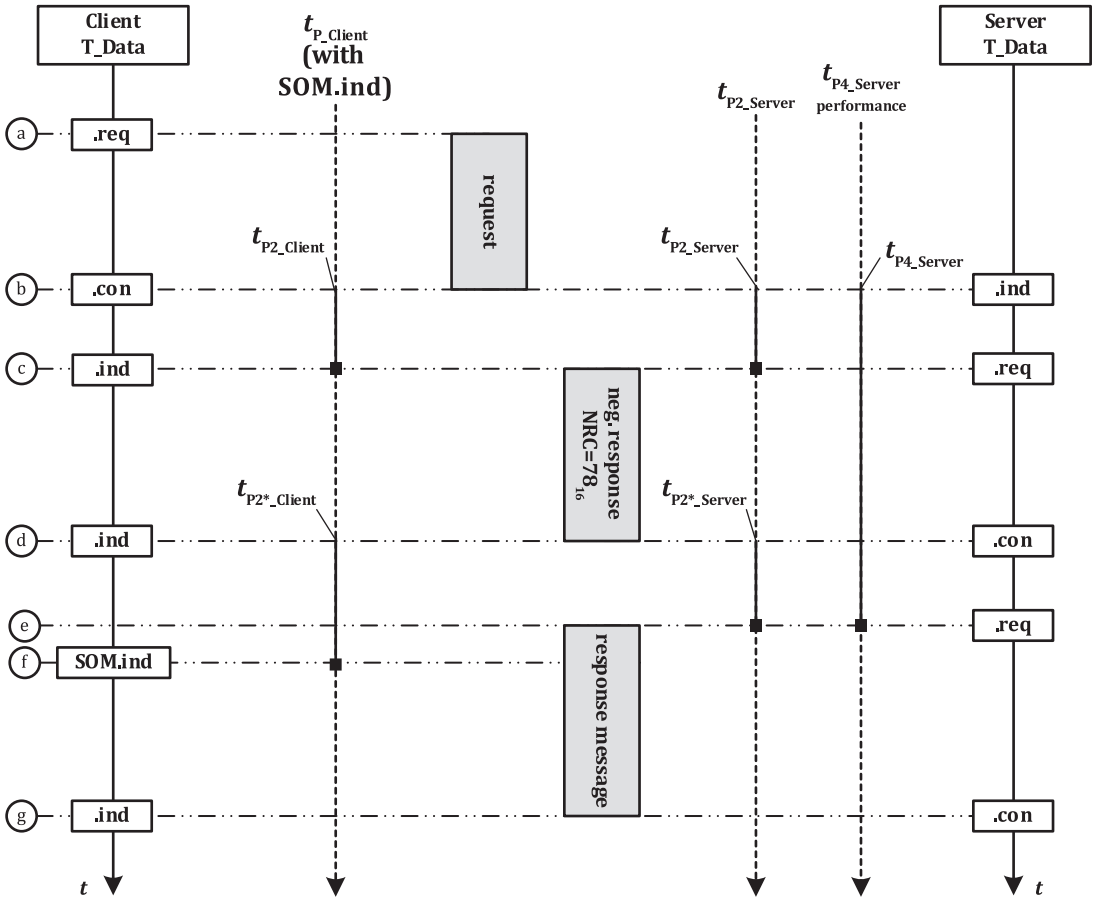
d Client `T_DataSOM.ind`: transport layer issues to the diagnostic application the reception of a StartOfMessage. Client stops the t_{P_Client} timer.

e Server `T_Data.conf`: transport layer issues to the diagnostic application the completion of the response message. Client `T_Data.ind`: transport layer issues to the diagnostic application the completion of the response message.

Figure 7 — Example for t_{P4_Server} without enhanced response timing

9.4 Example for t_{P4_Server} with enhanced response timing

Figure 8 shows an example where $t_{P4\_Server\_Max} > t_{P2\_Server\_Max}$. In this scenario the server response performance timing parameter indicates that negative responses including NRC 78_{16} are allowed as long as t_{P4_Server} is not exceeded.



Key

a Client `T_Data.req`: diagnostic application issues a request message to the transport layer.

b Server `T_Data.ind`: transport layer issues to the diagnostic application the completion of the request message. Server starts the t_{P2_Server} timer using the default value of $t_{P2\_Server} = t_{P2\_Server\_Max}$. If $t_{P4\_Server\_Max} > t_{P2\_Server\_Max}$ applies for a certain `T_Data.ind`, the server shall ensure that the final positive or negative response is started at the latest after $t_{P4\_Server\_Max}$ (i.e. negative responses with NRC 78_{16} are allowed as long as t_{P4_Server} is not exceeded). Client `T_Data.conf`: transport layer issues to the diagnostic application the confirmation of the completion of the request message. Client starts its t_{P_Client} timer using the default reload value $t_{P2\_Client} = t_{P2\_Client\_Max}$. The value of the t_{P_Client} timer shall consider any latency that is involved based on the vehicle network design (communication over gateways, bus bandwidth, etc.).

c Server `T_Data.req`: diagnostic application does not have the positive response message ready and issues negative response message with NRC = 78_{16} by a `T_Data.req` to the transport layer within t_{P2_Server}. Server stops the t_{P2_Server} timer. Client `T_DataSOM.ind`: transport layer issues to the diagnostic application the reception of a StartOfMessage. Client stops the t_{P_Client} timer.

d Server `T_Data.conf`: transport layer issues to the diagnostic application the completion of the response message. Server starts the t_{P2_Server} timer using the value of $t_{P2*\_Server} = t_{P2*\_Server:Max}$ (default enhanced timing). In case the server can still not provide the requested information within the enhanced t_{P2*_Server}, then a further negative response message including negative response code 78_{16} can be sent by the server. This causes the client to restart its t_{P_Client} timer, using the enhanced reload value t_{P2*_Client}. For simplicity, the figure only shows a single negative response message with negative response code 78_{16}. Client `T_Data.ind`: transport layer issues to the diagnostic application the reception of a response message. Client starts the t_{P_Client} timer with the value of $t_{P2*\_Client} = t_{P2*\_Client\_Max}$ (default enhanced timing).

e Server `T_Data.req`: diagnostic application has prepared the response message and issues a `T_Data.req` to the transport layer within t_{P2_Server}. Server stops the t_{P2_Server} timer. The t_{P4_Server} performance timer is stopped.

f Client `T_DataSOM.ind`: transport layer issues to the diagnostic application the reception of a StartOfMessage. Client stops the t_{P_Client} timer.

g Server T_Data.conf: transport layer issues to the diagnostic application the completion of the response message. Client T_Data.ind: transport layer issues to the diagnostic application the completion of the response message.

Figure 8 — Example for P4_Server with enhanced response timing

9.5 Session timing parameter definitions for the non-default session

When a diagnostic session other than the defaultSession is started, then a session handling is required.

REQ	5.24 Timing parameter definition – Session timing parameter definitions for the non-default session
The session timing parameter definitions for the non-default session values shall be in accordance with Table 5.	

Table 5 — Session timing parameter definitions for the non-default session

Timing parameter	Description	Type	Recommended reload (ms)	Timeout (ms)
t_{S3_Client}	Time between functionally-addressed TesterPresent ($3E_{16}$) request messages transmitted by the client to keep a diagnostic session other than the defaultSession active in multiple servers (functional communication) or maximum time between physically transmitted request messages to a single server (physical communication). The t_{S3_Client} timeout value includes the travel time of the message on the network (gateway delays, etc.).	Timer reload value	2 000	< t_{S3_Server}
t_{S3_Server}	Time for the server to keep a diagnostic session other than the defaultSession active while not receiving any diagnostic request message from the client which requested the transition to a non-default session. The tolerance of t_{S3_Server} is: 0 ms to 200 ms.	Timer reload value	N/A	5 000

The timing parameter definitions as specified in Tables 3 and 4 are also valid for the non-default session. Furthermore, the server can change its application layer timings t_{P2_Server} and t_{P2*_Server} when transitioning into a non-default session in order to achieve a certain performance or to compensate restrictions which can apply during a non-default diagnostic session. The applicable timing parameters for a non-default diagnostic session are reported in the DiagnosticSessionControl positive response message in the case where a response is required to be transmitted or known in advance by the client in case no response is required to be transmitted. When the client starts a non-default session functionally, then it adapts to the timing parameters of the responding servers.

Table 5 defines the conditions for the client and the server to start/restart its t_{S3_Client}/t_{S3_Server} timer. For the client a periodically-transmitted functionally-addressed TesterPresent ($3E_{16}$) request message shall be distinguished from a sequentially-transmitted physically-addressed TesterPresent ($3E_{16}$) request message, which is only transmitted in case of absence of any other diagnostic request message. For the server there is no need to distinguish between that kind of TesterPresent ($3E_{16}$) handling. Furthermore, Table 6 shows that the t_{S3_Server} timer handling is based on the transport layer service primitives from the client which requested the transition to a non-default session, which means that the t_{S3_Server} timer is also restarted upon the reception of a diagnostic request message that is not supported by the server. This functionality ensures that only the client that requested the non-default session controls it and no other client can affect the t_{S3_Server} timer and take over the session.

REQ	5.25 Timing parameter definition – Session layer timing in non-default session
During a non-default session, the additional timer resource requirements for the client and the server shall be in accordance with Table 6.	

Table 6 — Session layer timing start/stop conditions for the client and the server

Timing para meter	Action	Physical and functional communication, using functionally addressed, periodically-transmitted TesterPresent request message	Physical communication only, using a physically addressed, sequentially-transmitted TesterPresent request message
t_{S3_Client}	Initial start	T_Data.conf that indicates the completion of the DiagnosticSessionControl (10_{16}) request message. This is only true if the session type is a non-default session.	T_Data.conf that indicates the completion of the DiagnosticSessionControl (10_{16}) request message in case no response is required.
			T_Data.ind that indicates the reception of the DiagnosticSessionControl (10_{16}) response message in case a response is required.
	Sub-sequent start	T_Data.conf that indicates the completion of the functionally-addressed TesterPresent ($3E_{16}$) request message, which is transmitted each time the t_{S3_Client} timer times out.	T_Data.conf that indicates the completion of any request message in case no response is required.
			T_Data.conf that indicates an error during the transmission of either a single-frame or multi-frame request message.
			T_Data.ind that indicates the reception of any response message in case a response is required.
			T_Data.ind that indicates an error during the reception of a multi-frame response message.
t_{S3_Server}	Initial start	T_Data.conf that indicates the completion of the transmission of a DiagnosticSessionControl positive response message for a transition from the default session to a non-default session, in case a response message is required.	
		Successful completion of the requested action of the service DiagnosticSessionControl (10_{16}) for a transition from the default session to a non-default session, in case no response message is required/allowed.	
	Sub-sequent stop	T_DataSOM.ind that indicates the start of a multi-frame request message or T_Data.ind that indicates the reception of any SingleFrame from the client, which requested the transition to a non-default session request message. If the defaultSession is active, the t_{S3_Server} timer is disabled.	
	Sub-sequent start	T_Data.conf that indicates the completion of any response message that concludes a service execution (final response message) in case a response message is required/allowed to be transmitted (this includes positive and negative response messages). A negative response with negative response code 78_{16} does not restart the t_{S3_Server} timer.	
		Completion of the requested action (service conclusion) in case no response message (positive and negative) is required/allowed.	
		T_Data.ind that indicates an error during the reception of a multi-frame request message.	
		For further details regarding the t_{S3_Server} handling in the server when the server is requested to transmit unsolicited response message such as periodic data or responses based on an event, see the data link specific implementation documents of the ISO 14229 series.	

9.6 Client and server timer resource requirements

The session timer resource requirements are defined for the client and the server to fulfil the timing requirements during the default session and any non-default session.

REQ	5.26 Timing parameter definition – Session timer resource requirements
The session timer resource requirements for the client and the server to fulfil the timing requirements during the default session and any non-default session shall be in accordance with the specifications in Tables 7 and 8.	

Table 7 — Timer resources requirements during defaultSession

Timing parameter	Client	Server
t_{P_Client}	A single timer is required for each logical communication channel (physical and functional communication), e.g. each point-to-point communication requires a separate communication channel.	N/A
t_{P2_Server}	N/A	A single timer is required for the enhanced response timing in order to ensure that subsequent negative response messages with negative response code 78_{16} are transmitted prior to the expired $t_{P2*\ Server}$.
$t_{P3\_Client\_Phys}$	A single timer is required per logical physical communication channel.	N/A
$t_{P3\_Client\_Func}$	A single timer is required per logical functional communication channel.	N/A

Table 8 — Additional timer resources requirements during non-defaultSession

Timing parameter	Client	Server
t_{S3_Client}	A single timer is required when using a periodically transmitted, functionally-addressed TesterPresent ($3E_{16}$) request message to keep the servers in a non-defaultSession. There is no need for additional timers per activated diagnostic sessions.	N/A
	A single timer is required for each point-to-point communication channel when using a sequentially transmitted, physically-addressed TesterPresent ($3E_{16}$) request message to keep a single server in a non-defaultSession in case of the absence of another diagnostic request message then.	
t_{S3_Server}	N/A	A single timer is required in the server, because only a single diagnostic session can be active at a time in a single server.

9.7 Error handling

Error handling for the application layer and session management is performed by the client and the server during physical and functional communication.

REQ	5.27 Timing parameter definition – Client error handling
The client error handling for the application layer and session management during physical and functional communication shall be in accordance with Table 9.	

Table 9 — Client error handling

Communication phase	Client error type	Client error handling	
		Physical communication	Functional communication
Request transmission	T_Data.conf from transport layer with a negative result value	The client shall repeat the last request, after the time $t_{P3\_Client\_Phys}$ following the error indication. Restart t_{S3_Client} in the case of a physically addressed and sequentially-transmitted TesterPresent (because t_{S3_Client} has been stopped based on the request message transmission).	The client shall repeat the last request, after the time $t_{S3\_Client\_Func}$ following the error indication.
t_{P_Client} t_{P*_Client}	Timeout	The client shall repeat the last request. Restart t_{S3_Client} in the case of a physically addressed and sequentially-transmitted TesterPresent (because t_{S3_Client} has been stopped based on the request message transmission).	Where the client does not know the number of servers responding, then this is the indication for the client that no further response messages are expected. No retry of the request message is required. The client shall completely receive all response messages that are in progress until it can continue with further requests.
			Where the client knows the number of responding servers, then this is the indication for the client that not all expected servers responded. The client shall repeat the request after it has completely received any response message that is in progress at the point in time the timeout occurs.
Response reception	T_Data.ind from transport layer with a negative result value	The client shall repeat the last request. Restart t_{S3_Client} in the case of a physically addressed and sequentially-transmitted TesterPresent (because t_{S3_Client} has been stopped based on the request message transmission).	The client shall repeat the last request after it has completely received any response message that is in progress at the point in time the error has been indicated.
The client error handling defined shall be performed for a maximum of two times, which means that the worst-case of service request transmissions is three.			

REQ	5.28 Timing parameter definition – Server error handling
The server error handling for the application layer and session management during physical and functional communication shall be in accordance with Table 10.	

Table 10 — Server error handling

Communication phase	Server error type	Server error handling
Request reception	T_Data.ind from transport layer with a negative result value	Restart t_{S3_Server} timer (because it has been stopped based on the previously received StartOfMessage indication). The server shall ignore the request.

Table 10 *(continued)*

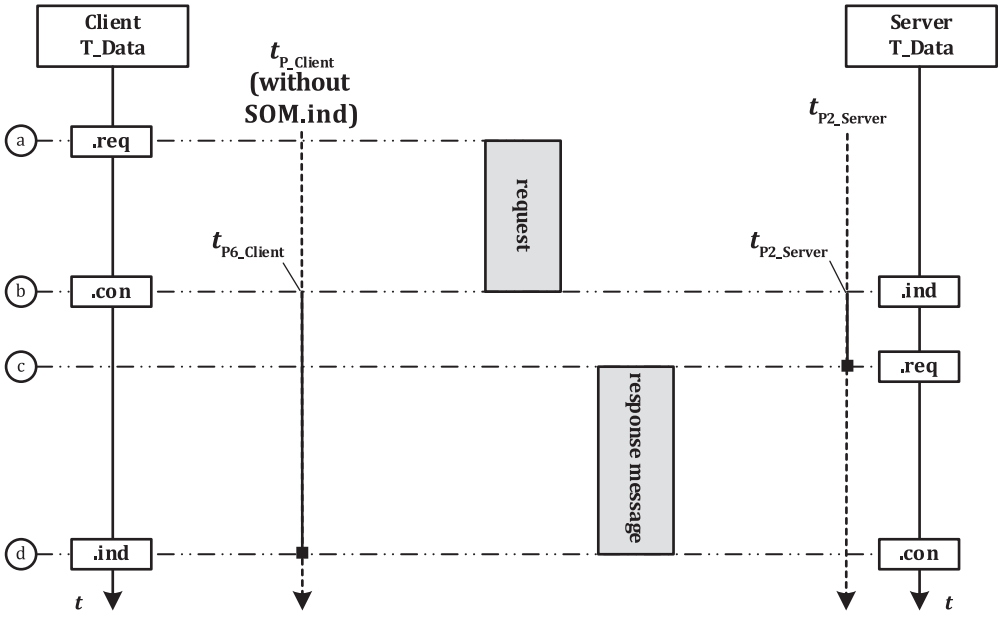
Communication phase	Server error type	Server error handling
Response transmission	`T_Data.conf` from transport layer with a negative result value	Restart t_{S3_Server} timer (because it has been stopped based on the previously received request message). The server shall not perform a retransmission of the response message.

10 Timing handling during communication

10.1 Physical communication

10.1.1 Physical communication during defaultSession – without SOM.ind

Figure 9 graphically depicts the timing handling in the client and the server for a physically-addressed request message without `SOM.ind` during the default session.



Key

a Client `T_Data.req`: diagnostic application issues a request message to the transport layer.

b Server `T_Data.ind`: transport layer issues to the diagnostic application the completion of the request message. Server starts the t_{P2_Server} timer using the default value of t_{P2_Server} = $t_{P2\_Server\_Max}$. Client `T_Data.conf`: transport layer issues to the diagnostic application the confirmation of the completion of the request message. Client starts its t_{P_Client} timer using the default reload value t_{P6_Client} = $t_{P6\_Client\_Max}$. The value of the t_{P_Client} timer shall consider any latency that is involved based on the vehicle network design (communication over gateways, bus bandwidth, etc.).

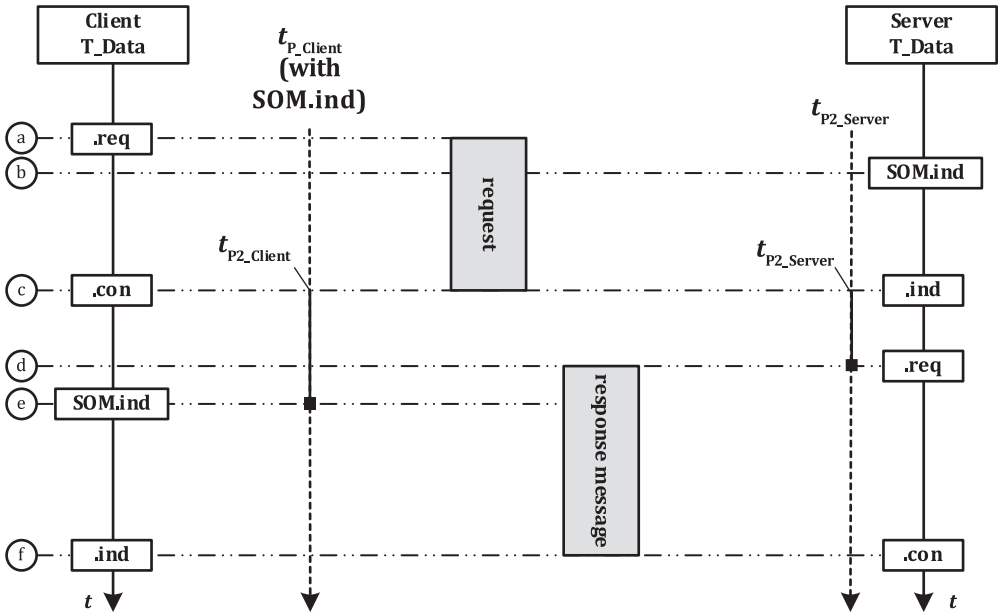
c Server `T_Data.req`: diagnostic application has prepared the response message and issues a `T_Data.req` to the transport layer within t_{P2_Server}. Server stops the t_{P2_Server} timer.

d Server `T_Data.conf`: transport layer issues to the diagnostic application the completion of the response message. Client `T_Data.ind`: transport layer issues to the diagnostic application the completion of the response message. The client stops its t_{P_Client} timer.

Figure 9 — Physical communication during default session - without SOM.ind

10.1.2 Physical communication during defaultSession – with SOM.ind

Figure 10 graphically depicts the timing handling in the client and the server for a physically-addressed request message with `SOM.ind` during the default session.



Key

a Client `T_Data.req`: diagnostic application issues a request message to the transport layer.

b Server `T_DataSOM.ind`: transport layer issues to the diagnostic application the reception of a StartOfMessage of a request message.

c Server `T_Data.ind`: transport layer issues to the diagnostic application the completion of the request message. Server starts the t_{P2_Server} timer using the default value of $t_{P2\_Server} = t_{P2\_Server\_Max}$. Client `T_Data.conf`: transport layer issues to the diagnostic application the confirmation of the completion of the request message. Client starts its t_{P_Client} timer using the default reload value $t_{P2\_Client} = t_{P2\_Client\_Max}$. The value of the t_{P_Client} timer shall consider any latency that is involved based on the vehicle network design (communication over gateways, bus bandwidth, etc.). For simplicity, the figure assumes that the client and the server are located on the same network.

d Server `T_Data.req`: diagnostic application has prepared the response message and issues a `T_Data.req` to the transport layer within t_{P2_Server}. Server stops the t_{P2_Server} timer.

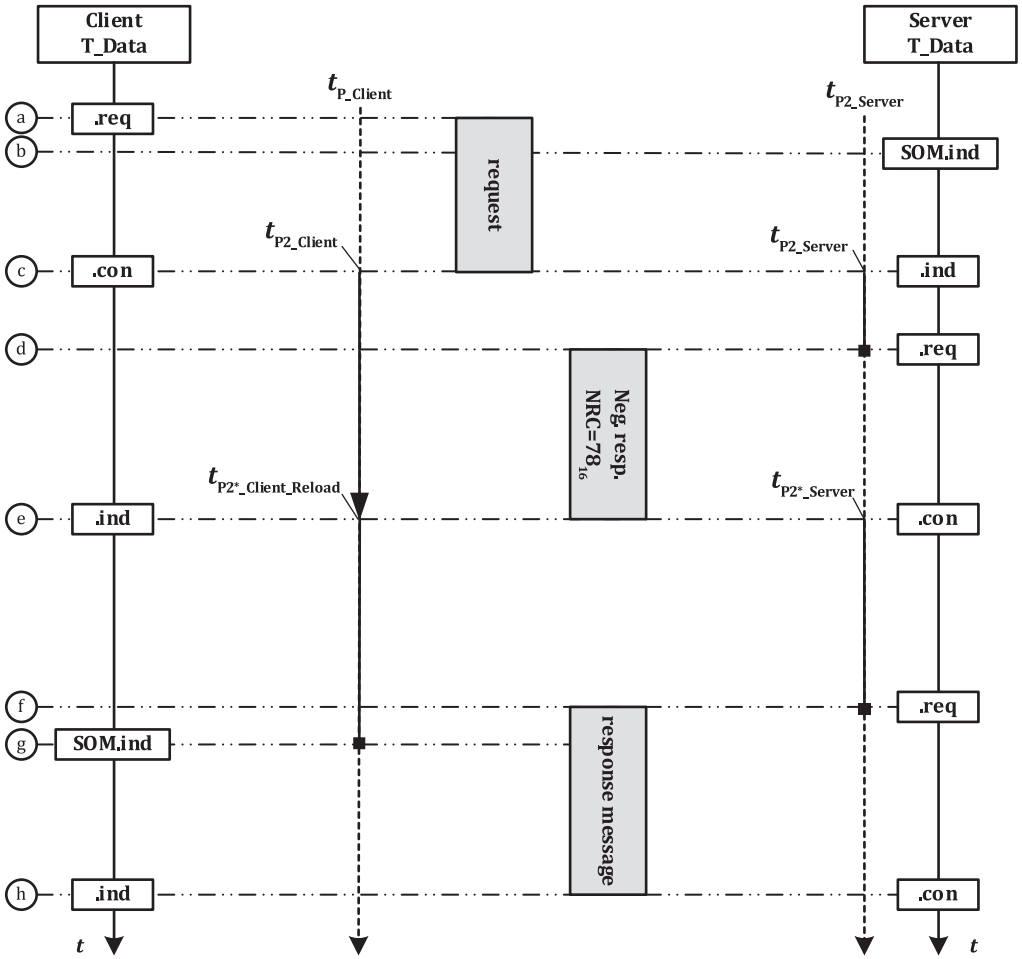
e Client `T_DataSOM.ind`: transport layer issues to the diagnostic application the reception of a StartOfMessage. Client stops the t_{P_Client} timer.

f Server `T_Data.conf`: transport layer issues to the diagnostic application the completion of the response message. Client `T_Data.ind`: transport layer issues to the diagnostic application the completion of the response message.

Figure 10 — Physical communication during default session – with SOM.ind

10.1.3 Physical communication during defaultSession with enhanced response timing

Figure 11 graphically depicts the timing handling in the client and the server for a physically-addressed request message during the default session and the request of the server for an enhanced response timing (negative response code 78_{16} handling).



Key

a Client `T_Data.req`: diagnostic application issues a request message to the transport layer.

b Server `T_DataSOM`.ind: transport layer issues to the diagnostic application the reception of a StartOfMessage of a request message, if the transport layer supports the `T_DataSOM.ind` interfaces.

c Server `T_Data.ind`: transport layer issues to the diagnostic application the completion of the request message. Server starts the t_{P2_Server} timer using the default value of $t_{P2\_Server} = t_{P2\_Server\_Max}$. Client `T_Data.conf`: transport layer issues to the diagnostic application the confirmation of the completion of the request message. Client starts its t_{P_Client} timer using the default reload value $t_{P\_Client} = t_{P\_Client\_Max}$.

d Server `T_Data.req`: diagnostic application does not have the positive response message ready and issues negative response message with NRC = 78_{16} by a `T_Data.req` to the transport layer within t_{P2_Server}. Server stops the t_{P2_Server} timer.

e Server `T_Data.conf`: transport layer issues to the diagnostic application the completion of the response message. Server starts the t_{P2_Server} timer using the value of $t_{P2\_Server} = t_{P2*\_Server\_Max}$ (default enhanced timing). In case the server can still not provide the requested information within the enhanced t_{P2*_Server}, then a further negative response message including negative response code 78_{16} can be sent by the server. This causes the client to restart its t_{P_Client} timer, using the enhanced reload value t_{P2*_Client}. For simplicity, the figure only shows a single negative response message with negative response code 78_{16}. Client `T_Data.ind`: transport layer issues to the diagnostic application the reception of a response message. Client stops the t_{P_Client} timer and reloads the t_{P_Client} timer with the value of $t_{P2*\_Client} = t_{P2*\_Client\_Max}$ (default enhanced timing).

f Server `T_Data.req`: diagnostic application issues a response message (positive or negative response other than negative response code 78_{16}) to the transport layer. Server stops the t_{P2_Server} timer.

g Client `T_DataSOM.ind`: transport layer issues to the diagnostic application reception of a StartOfMessage of a response message, if the transport layer supports the `T_DataSOM.ind` interfaces. Client stops the t_{P_Client} timer.

h Server T_Data.conf: transport layer issues to the diagnostic application the completion of the response message. Client T_Data.ind: transport layer issues to the diagnostic application the reception of a response message. When receiving this indication, the client stops its t_{P_Client} timer if the transport protocol does not support a T_DataSOM.ind interface.

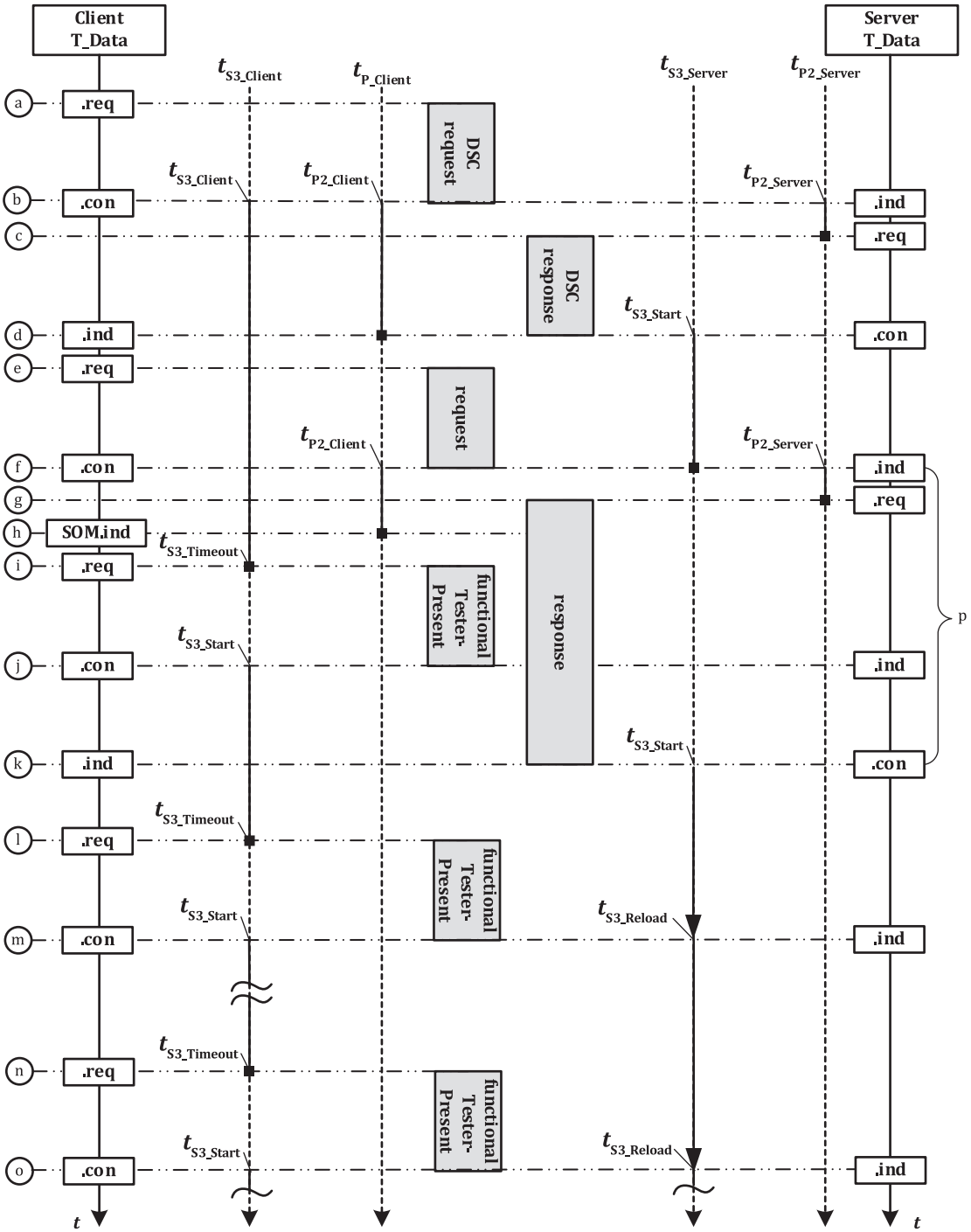
Figure 11 — Physical communication during default session – Enhanced response timing

10.1.4 Physical communication during a non-default session

10.1.4.1 Functionally-addressed TesterPresent ($3E_{16}$) message

Figure 12 graphically depicts the timing handling in the client and the server when performing physical communication during a non-default session (e.g. programmingSession) and using a functionally addressed, periodically-transmitted TesterPresent ($3E_{16}$) request message that does not require a response message from the server.

The handling of the t_{P_Client} and t_{P2_Server} timing is identical to the handling as described in 10.1.2. The only exception is that the reload values on the client side and the resulting time where the server shall send its final response time might differ. This is based on the transition into a session other than the default session where different t_{P_Client} timing parameters might apply (see DiagnosticSessionControl (10_{16}) service in ISO 14229-1 for details on how the timing parameters are reported to the client).



Key

a Client `T_Data.req`: diagnostic application issues the DiagnosticSessionControl (10_{16}) request message to the transport layer. The transport layer transmits the request message to the server.

b Client `T_Data.conf`: transport layer issues to the diagnostic application the reception of the DiagnosticSessionControl (10_{16}) request message. Now the response timing t_{P_Client} as described in 10.1.2 applies. The generated `T_Data.conf` in the client causes the start of the t_{S3_Client} timer (session timer). Client starts its t_{P_Client} timer using the default reload value t_{P2_Client} = $t_{P2\_Client\_Max}$. Server `T_Data.ind`: transport layer issues to the diagnostic application the completion of the DiagnosticSessionControl (10_{16}) request message. Now the response timing t_{P2_Server} as described in 10.1.2 applies.

c Server `T_Data.req`: diagnostic application issues the DiagnosticSessionControl (10_{16}) positive response message to the transport layer. For the figure given, it is assumed that the client requires a response from the server.

d Server T_Data.conf: the completion of the transmission of the response message is indicated in the server via T_Data.conf. Now the server starts its t_{S3_Server} timer, which keeps the activated non-default session active as long as it does not time out. It is the client's responsibility to ensure that the t_{S3_Server} timer is reset prior to its timeout to keep the server in the non-default session. Client T_Data.ind: transport layer issues to the diagnostic application the completion of the response message. Client stops the t_{P_Client} timer.

e Client T_Data.req: diagnostic application issues a new request message to the transport layer.

f Server T_Data.ind: transport layer issues to the diagnostic application the completion of the request message. Any time the server is in the process of handling any diagnostic service, it stops its t_{S3_Server} timer. Client starts its t_{P_Client} timer using the default reload value $t_{P2\_Client} = t_{P2\_Client\_Max}$. Client T_Data.conf: transport layer issues to the diagnostic application the completion of the request message.

g Server T_Data.req: diagnostic application issues the positive response message to the transport layer.

h Client T_DataSOM.ind: transport layer issues to the diagnostic application the reception of a StartOfMessage, if the T_DataSOM.ind interface is supported by the transport layer. Client stops the t_{P_Client} timer.

i Client T_Data.req: once the t_{S3_Client} timer is started in the client, this causes the transmission of a functionally-addressed TesterPresent ($3E_{16}$) request message, which does not require a response message, each time the t_{S3_Client} timer times out.

j Client T_Data.conf: upon the indication of the completed transmission of the TesterPresent ($3E_{16}$) request message via T_Data.conf of its transport layer, the client once again starts its t_{S3_Client} timer. This means that the functionally-addressed TesterPresent ($3E_{16}$) request message is sent on a periodic basis every time t_{S3_Client} times out. Server T_Data.ind: any TesterPresent ($3E_{16}$) request message that is received during processing another request message can be ignored by the server, because it has already stopped its t_{S3_Server} timer and restarts it once the service that is in progress is processed completely.

k Client T_Data.ind: transport layer issues to the diagnostic application the completion of the response message. Server T_Data.conf: when the diagnostic service is completely processed, then the server restarts its t_{S3_Server} timer. This means that any diagnostic service, including TesterPresent ($3E_{16}$), resets the t_{S3_Server} timer. A diagnostic service is considered to be in progress any time between the start of the reception of the request message (T_DataSOM.ind or T_Data.ind receive) and the completion of the transmission of the final response message, where a response message is required, or the completion of any action that is caused by the request, where no response message is required (point in time reached that would cause the start of the response message).

l Client T_Data.req: once the t_{S3_Client} timer is started in the client, this causes the transmission of a functionally-addressed TesterPresent ($3E_{16}$) request message, which does not require a response message, each time the t_{S3_Client} timer times out.

m Client T_Data.conf: upon the indication of the completed transmission of the TesterPresent ($3E_{16}$) request message via T_Data.conf of its transport layer, the client once again starts its t_{S3_Client} timer. This means that the functionally-addressed TesterPresent ($3E_{16}$) request message is sent on a periodic basis every time t_{S3_Client} times out. Server T_Data.ind: any TesterPresent ($3E_{16}$) request message that is received during an activated t_{S3_Server} timer reloads the t_{S3_Server} timer.

n Client T_Data.req: once the t_{S3_Client} timer is started in the client, this causes the transmission of a functionally-addressed TesterPresent ($3E_{16}$) request message, which does not require a response message, each time the t_{S3_Client} timer times out.

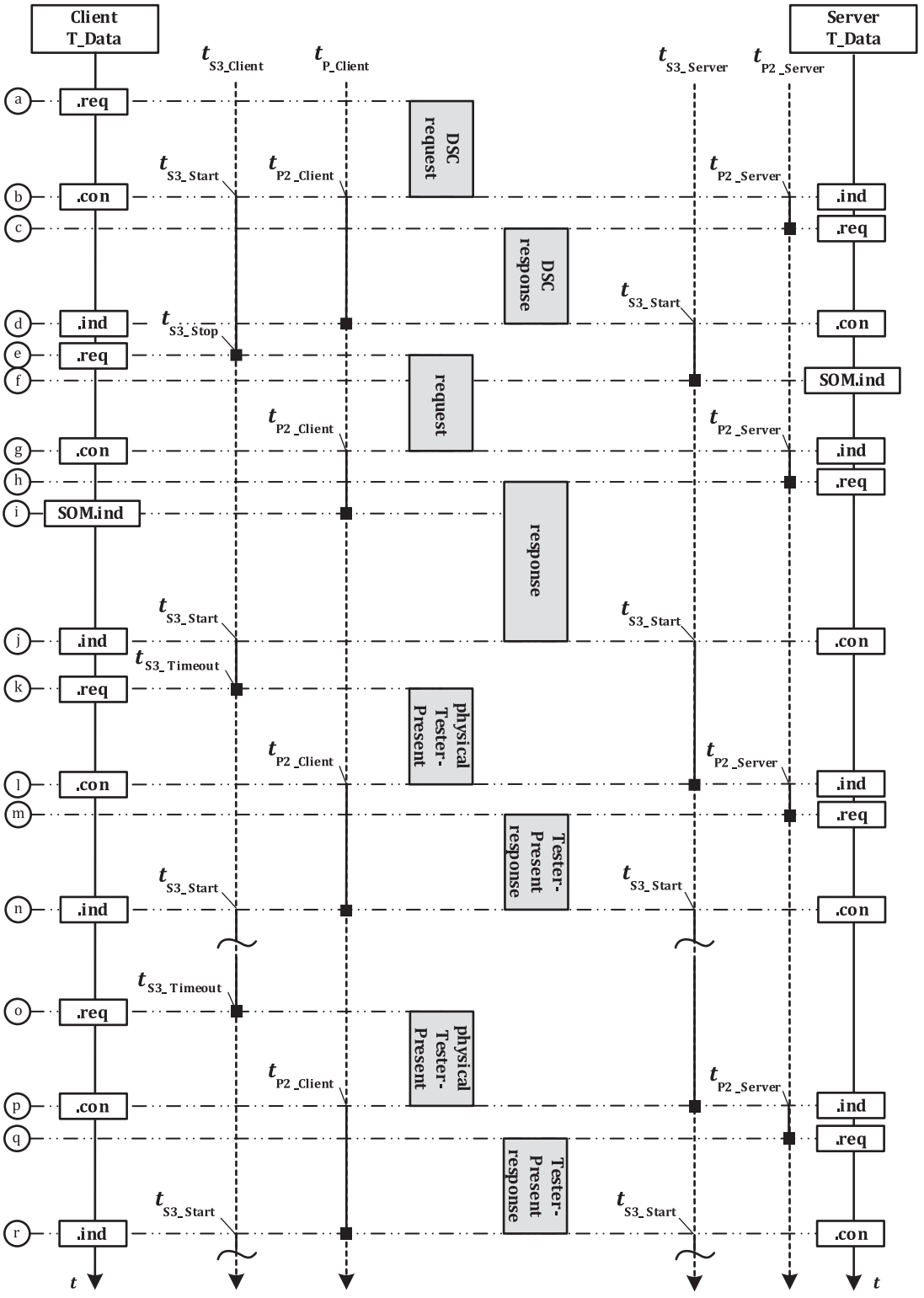
o Client T_Data.conf: upon the indication of the completed transmission of the TesterPresent ($3E_{16}$) request message via T_Data.conf of its transport layer, the client once again starts its t_{S3_Client} timer. This means that the functionally-addressed TesterPresent ($3E_{16}$) request message is sent on a periodic basis every time t_{S3_Client} times out. Server T_Data.ind: any TesterPresent ($3E_{16}$) request message that is received during an activated t_{S3_Server} timer reloads the t_{S3_Server} timer.

p Any TesterPresent that is received during a disabled t_{S3_Server} timer is ignored by the server.

Figure 12 — Physical communication during non-default session – functionally-addressed TesterPresent

10.1.4.2 Physically-addressed TesterPresent ($3E_{16}$) message

Figure 13 graphically depicts the timing handling in the client and the server when performing physical communication during a non-default session (e.g. programmingSession) and using a physically-addressed TesterPresent ($3E_{16}$) request message that requires a response message from the server to keep the diagnostic session active in case of the absence of any other diagnostic service.



Key

a Client `T_Data.req`: diagnostic application issues the DiagnosticSessionControl (10_{16}) request message to the transport layer. The transport layer transmits the request message to the server.

b Client `T_Data.conf`: transport layer issues to the diagnostic application the reception of the DiagnosticSessionControl (10_{16}) request message. Now the response timing t_{P_Client} as described in 10.1.2 applies. The generated `T_Data.conf` in the client causes the start of the t_{S3_Client} timer (session timer). Client starts its t_{P_Client} timer using the default reload value t_{P2_Client} = $t_{P2\_Client\_Max}$. Server `T_Data.ind`: transport layer issues to the diagnostic application the completion of the DiagnosticSessionControl (10_{16}) request message. Now the response timing t_{P2_Server} as described in 10.1.2 applies.

c Server T_Data.req: diagnostic application issues the DiagnosticSessionControl (10_{16}) positive response message to the transport layer. For the figure given, it is assumed that the client requires a response from the server.

d Server T_Data.conf: the completion of the transmission of the response message is indicated in the server via T_Data.conf. Now the server starts its t_{S3_Server} timer, which keeps the activated non-default session active as long as it does not time out. It is the client's responsibility to ensure that the t_{S3_Server} timer is reset prior to its timeout to keep the server in the non-default session. Client T_Data.ind: transport layer issues to the diagnostic application the completion of the response message. Client stops the t_{P_Client} timer.

e Client T_Data.req: diagnostic application issues a new request message to the transport layer. Whenever the client transmits a request message to the server (including the physically-addressed TesterPresent ($3E_{16}$) message), it stops its t_{S3_Client} timer.

f Server T_DataSOM.ind: transport layer issues to the diagnostic application the reception of a StartOfMessage, if the T_DataSOM.ind interface is supported by the transport layer. The reception of a StartOfMessage of the request message stops the t_{S3_Server} timer in the server.

g Server T_Data.ind: transport layer issues to the diagnostic application the completion of the request message. Client T_Data.conf: transport layer issues to the diagnostic application the completion of the request message. In the case where the client does not require a response message it shall start its t_{S3_Client} timer when it receives confirmation of the completion of the request message, which is indicated via T_Data.conf. In this case the server starts its t_{S3_Server} timer when it has completed the requested action. Client starts its t_{P_Client} timer using the default reload value $t_{P2\_Client} = t_{P2\_Client\_Max}$.

h Server T_Data.req: diagnostic application issues the positive response message to the transport layer.

i Client T_DataSOM.ind: transport layer issues to the diagnostic application the reception of a StartOfMessage, if the T_DataSOM.ind interface is supported by the transport layer. Client stops the t_{P_Client} timer.

j Client T_Data.ind: transport layer issues to the diagnostic application the completion of the response message. The generated T_Data.ind in the client causes the start of the t_{S3_Client} timer (session timer). Server T_Data.conf: transport layer issues to the diagnostic application the completion of the response message. Now the server starts its t_{S3_Server} timer.

k Client T_Data.req: in case the client would not send any diagnostic request message prior to the timeout of t_{S3_Client}, then the timeout of the t_{S3_Client} timer causes the client to transmit a physically-addressed TesterPresent ($3E_{16}$) request message. Client starts its t_{P_Client} timer using the default reload value $t_{P2\_Client} = t_{P2\_Client\_Max}$.

l Server T_Data.ind: the reception of the TesterPresent ($3E_{16}$) request message is indicated in the server via T_Data.ind. This causes the server to stop its t_{S3_Server} timer. Now the response timing as described in 10.1.2 applies. Client T_Data.conf: the completion of the TesterPresent ($3E_{16}$) request message is indicated in the client via T_Data.conf.

m Server T_Data.req: diagnostic application issues the TesterPresent ($3E_{16}$) response message to the transport layer.

n Client T_Data.ind: the completion of the TesterPresent ($3E_{16}$) response message is indicated in the client via T_Data.ind, which causes the client to start its t_{S3_Client} timer. Server T_Data.conf: the completion of the TesterPresent ($3E_{16}$) response message is indicated in the server via T_Data.conf, which causes the server to start its t_{S3_Server}. In the case where the client would not require a response message, then it shall start its t_{S3_Client} timer when it receives confirmation of the completion of the TesterPresent ($3E_{16}$) request message, which is indicated via T_Data.conf. The server would start its t_{S3_Server} timer when it has completed the requested action. For simplicity, the figure shows that a response is required. Client stops the t_{P_Client} timer.

o Client T_Data.req: in case the client would not send any diagnostic request message prior to the timeout of t_{S3_Client}, then the timeout of the t_{S3_Client} timer causes the client to transmit a physically-addressed TesterPresent ($3E_{16}$) request message.

p Server T_Data.ind: the reception of the TesterPresent ($3E_{16}$) request message is indicated in the server via T_Data.ind. This causes the server to stop its t_{S3_Server} timer. Now the response timing as described in 10.1.2 applies. Client T_Data.conf: the completion of the TesterPresent ($3E_{16}$) request message is indicated in the client via T_Data.conf. Client starts its t_{P_Client} timer using the default reload value $t_{P2\_Client} = t_{P2\_Client\_Max}$.

q Server T_Data.req: diagnostic application issues the TesterPresent ($3E_{16}$) response message to the transport layer.

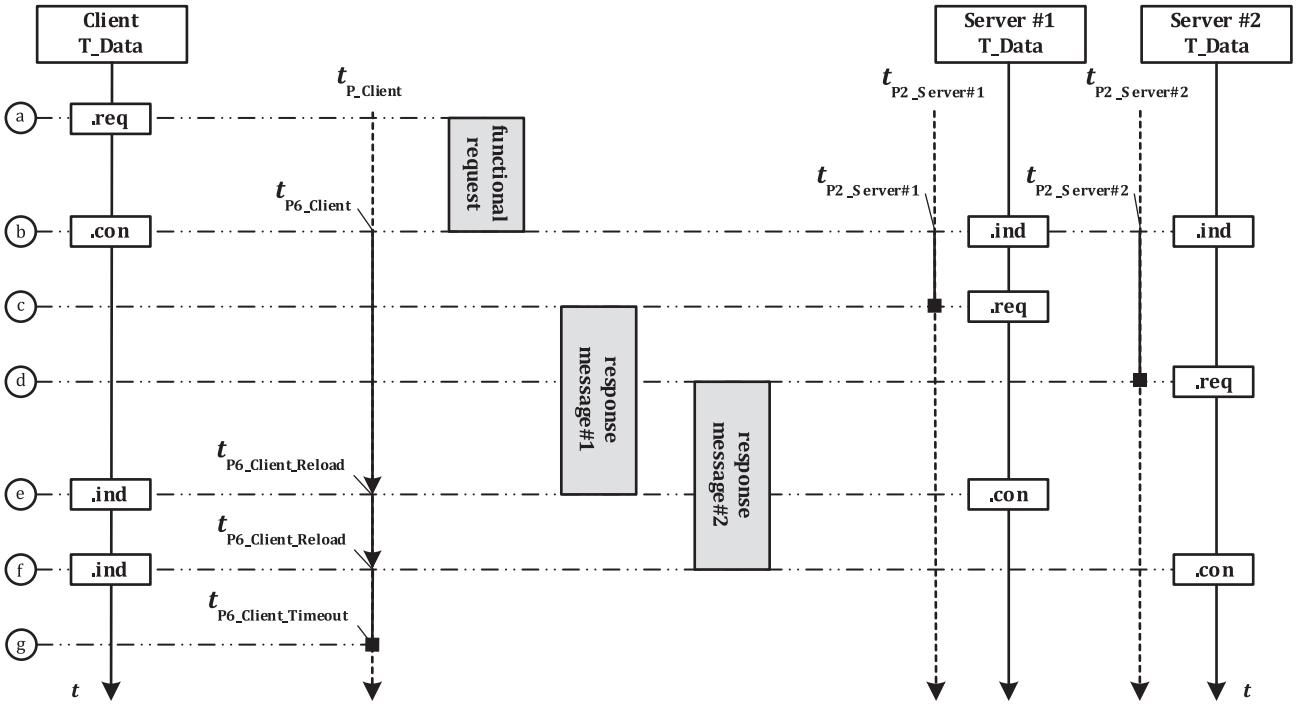
r Client `T_Data.ind`: the completion of the TesterPresent ($3E_{16}$) response message is indicated in the client via `T_Data.ind`, which causes the client to start its t_{S3_Client} timer. Server `T_Data.conf`: the completion of the TesterPresent ($3E_{16}$) response message is indicated in the server via `T_Data.conf`, which causes the server to start its t_{S3_Server}. In the case where the client would not require a response message, then it shall start its t_{S3_Client} timer when it receives confirmation of the completion of the TesterPresent ($3E_{16}$) request message, which is indicated via `T_Data.conf`. The server would start its t_{S3_Server} timer when it has completed the requested action. For simplicity, the figure shows that a response is required.

Figure 13 — Physical communication during non-default session – Physically-addressed TesterPresent

10.2 Functional communication

10.2.1 Functional communication during defaultSession – without SOM.ind

Figure 14 graphically depicts the timing handling in the client and two servers for a functionally-addressed request message during the default session. From a server point of view, there is no difference in the timing handling compared to a physically-addressed request message, but the client shall handle the timing different compared to physical communication.



Key

a Client `T_Data.req`: diagnostic application issues a functionally-addressed request message to the transport layer.

b All server `T_Data.ind`: transport layer issues to the diagnostic application the completion of a request message. All servers start the t_{P2_Server} timer using the value of $t_{P2\_Server} = t_{P2\_Server\_Max}$. Client `T_Data.conf`: transport layer issues to the diagnostic application the confirmation of the completion of the request message. Client starts its t_{P_Client} timer using the default reload value $t_{P6\_Client} = t_{P6\_Client\_Max}$. The value of the t_{P_Client} timer shall consider any latency that is involved based on the vehicle network design (e.g. communication over gateways, bus bandwidth, etc.). For simplicity, the figure assumes that the client and the server are located on the same network.

c Server #1 `T_Data.req`: diagnostic application has prepared the response message and issues a `T_Data.req` to the transport layer within t_{P2_Server}. Server stops the t_{P2_Server} timer.

d Server #2 `T_Data.req`: diagnostic application has prepared the response message and issues a `T_Data.req` to the transport layer within $t_{P2\_Server\_Max}$. Server stops the t_{P2_Server} timer.

e Server #1 `T_Data.conf`: transport layer issues to the diagnostic application the completion of the response message. Client `T_Data.ind`: transport layer issues to the diagnostic application the completion of the response message. This causes the client to restart its t_{P_Client} timer, using the default reload value $t_{P6\_Client\_Max}$.

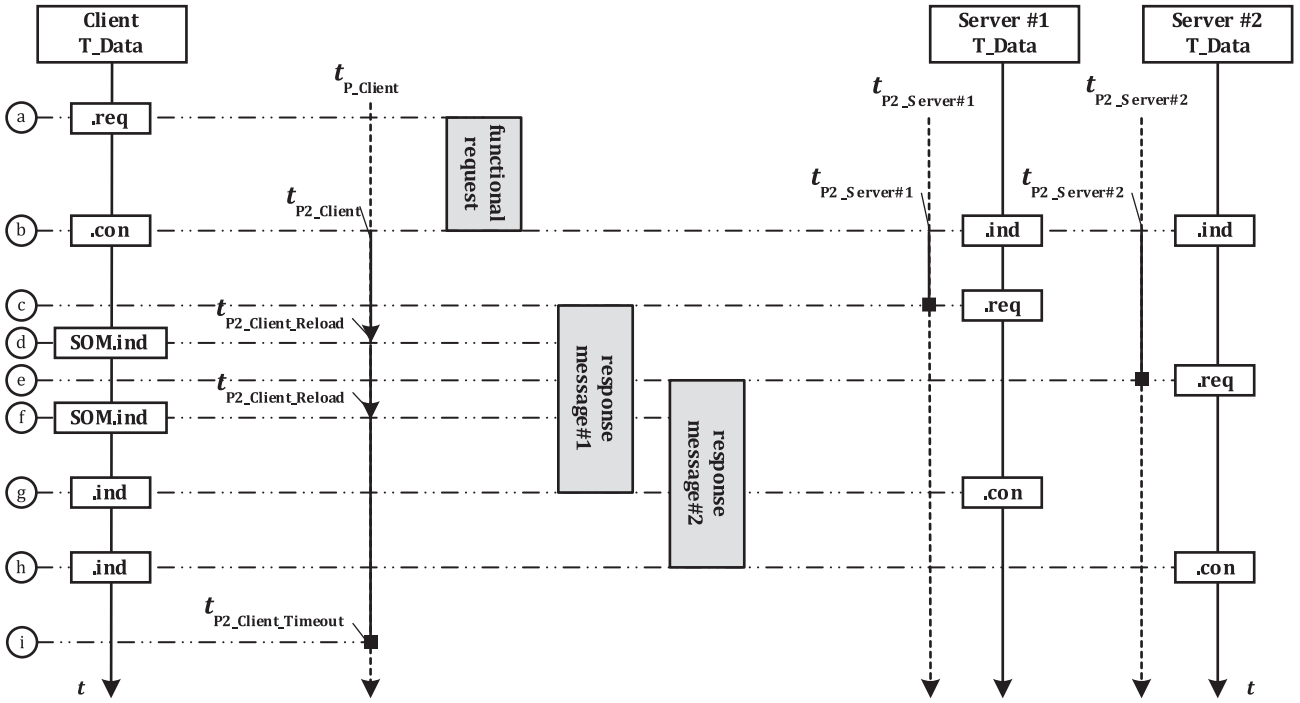
f Server #2 `T_Data.conf`: transport layer issues to the diagnostic application the completion of the response message. Client `T_Data.ind`: transport layer issues to the diagnostic application the completion of the response message. This causes the client to restart its t_{P_Client} timer, using the default reload value $t_{P6\_Client\_Max}$.

g Client: this is the indication for the client that no further response messages are expected and can continue with further requests.

Figure 14 — Functional communication during default session – without SOM.ind

10.2.2 Functional communication during defaultSession – with SOM.ind

Figure 15 graphically depicts the timing handling in the client and two servers for a functionally-addressed request message during the default session. From a server point of view, there is no difference in the timing handling compared to a physically-addressed request message, but the client shall handle the timing different compared to physical communication.



Key

a Client `T_Data.req`: diagnostic application issues a functionally-addressed request message to the transport layer.

b All server `T_Data.ind`: transport layer issues to the diagnostic application the completion of a request message. All servers start the t_{P2_Server} timer using the value of $t_{P2\_Server} = t_{P2\_Server\_Max}$. Client `T_Data.conf`: transport layer issues to the diagnostic application the confirmation of the completion of the request message. Client starts its t_{P_Client} timer using the default reload value $t_{P2\_Client} = t_{P2\_Client\_Max}$. The value of the t_{P_Client} timer shall consider any latency that is involved based on the vehicle network design (e.g. communication over gateways, bus bandwidth, etc.). For simplicity, the figure assumes that the client and the server are located on the same network.

c Server #1 `T_Data.req`: diagnostic application has prepared the response message and issues a `T_Data.req` to the transport layer within t_{P2_Server}. Server#1 stops the t_{P2_Server} timer.

d Client `T_DataSOM.ind`: transport layer issues to the diagnostic application the reception of a StartOfMessage. This causes the client to restart its t_{P_Client} timer, using the default reload value $t_{P2\_Client\_Max}$.

e Server #2 `T_Data.req`: diagnostic application has prepared the response message and issues a `T_Data.req` to the transport layer within $t_{P2\_Server\_Max}$. Server#2 stops the t_{P2_Server} timer.

f Client `T_DataSOM.ind`: transport layer issues to the diagnostic application the reception of a StartOfMessage. This causes the client to restart its t_{P_Client} timer, using the default reload value $t_{P2\_Client\_Max}$.

g Server #1 `T_Data.conf`: transport layer issues to the diagnostic application the completion of the response message.

h Server #2 `T_Data.conf`: transport layer issues to the diagnostic application the completion of the response message. Client `T_Data.ind`: transport layer issues to the diagnostic application the completion of the response message.

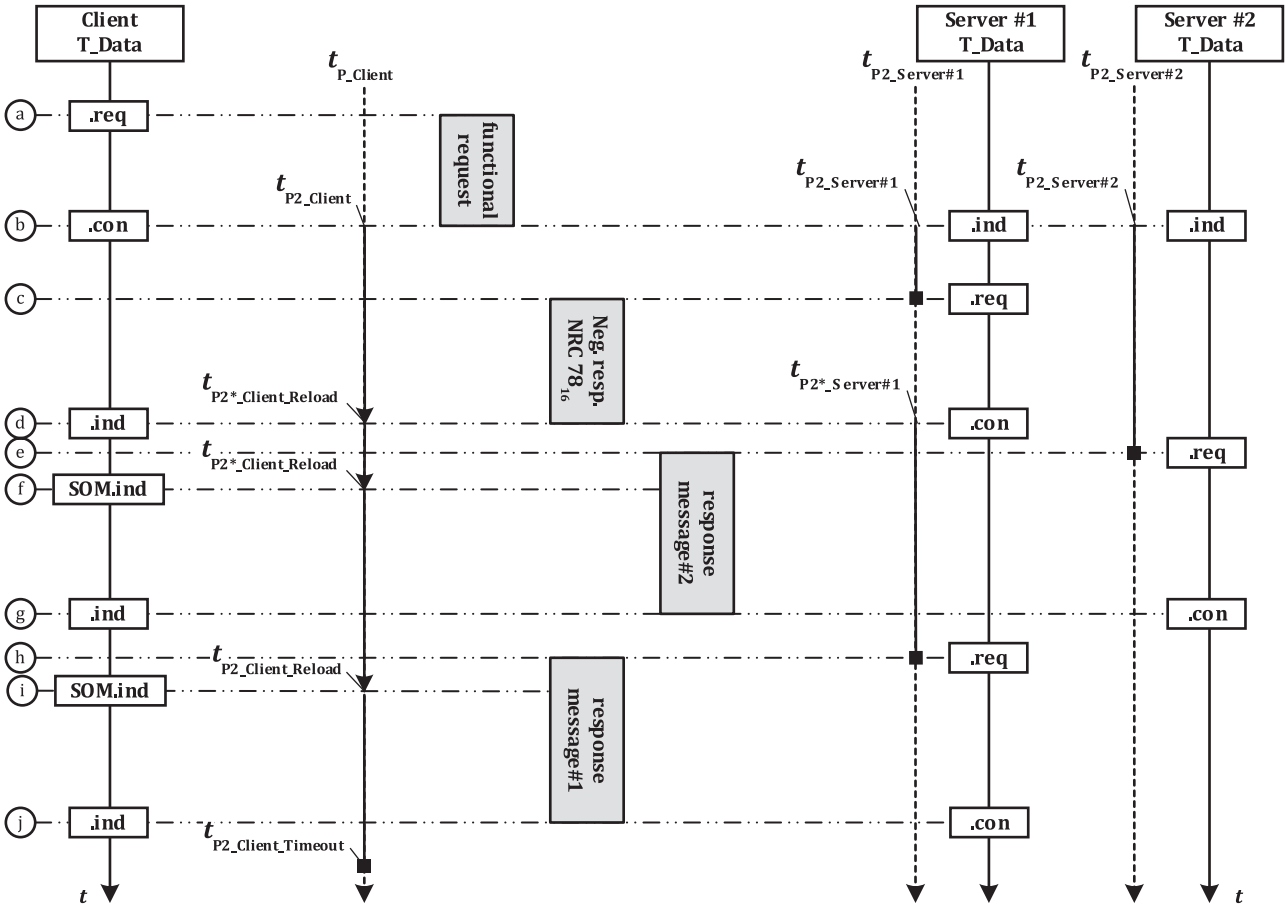
i Client: this is the indication for the client that no further response messages are expected and can continue with further requests.

Figure 15 — Functional communication during default session – with SOM.ind

10.2.3 Functional communication during defaultSession with enhanced response timing – with SOM.ind

Figure 16 graphically depicts the timing handling in the client and two servers for a functionally-addressed request message during the default session, where one server requests the enhanced response timing via a negative response message including negative response code 78_{16}.

From a server point of view there is no difference in the timing handling compared to a physically-addressed request message that requires enhanced response timing, but the client shall handle the timing differently compared to physical communication.



Key

a Client `T_Data.req`: diagnostic application issues a functionally-addressed request message to the transport layer.

b All server `T_Data.ind`: transport layer issues to the diagnostic application the completion of a request message. All servers start the t_{P2_Server} timer using the value of = $t_{P2\_Server\_Max}$. Client `T_Data.conf`: transport layer issues to the diagnostic application the confirmation of the completion of the request message. Client starts its t_{P_Client} timer using the default reload value t_{P2_Client} = $t_{P\_Client\_Max}$. The NRC 78_{16} pending list is empty. The value of the t_{P_Client} timer shall consider any latency that is involved based on the vehicle network design (e.g. communication over gateways, bus bandwidth, etc.). For simplicity, the figure assumes that the client and the server are located on the same network.

c Server #1 `T_Data.req`: diagnostic application does not have the positive response message ready and issues negative response message with NRC = 78_{16} by a `T_Data.req` to the transport layer within t_{P2_Server}. Server#1 stops the t_{P2_Server} timer. In case any of the addressed servers cannot provide the requested information within the t_{P2_Server} response timing, it can request an enhanced response timing window by sending a negative response message including negative response code 78_{16}.

d Server#1 `T_Data.conf`: transport layer issues to the diagnostic application the completion of the response message. Server#1 starts the t_{P2_Server} timer using the value of $t_{P2*\_Server\#1}$ = $t_{P2*\_Server\_Max}$ (default enhanced timing). Client `T_Data.ind`: transport layer issues to the diagnostic application the reception of a response message. Client reloads the t_{P_Client} timer with the value of t_{P2*_Client} = $t_{P2*\_Client\_Max}$ (default enhanced timing). An entry containing the response message address is added to the NRC 78_{16} pending list.

e Server#2 `T_Data.req`: diagnostic application has prepared the response message and issues a `T_Data.req` to the transport layer within t_{P2_Server}. Server#2 stops the t_{P2_Server} timer.

f Client `T_DataSOM.ind`: transport layer issues to the diagnostic application the reception of a StartOfMessage. This causes the client to restart its t_{P_Client} timer, using the default reload value t_{P2*_Client} (default enhanced timing).

g Server #2 `T_Data.conf`: transport layer issues to the diagnostic application the completion of the response message. Client `T_Data.ind`: transport layer issues to the diagnostic application the completion of the response message.

h Server#1 T_Data.req: diagnostic application has prepared the response message and issues a T_Data.req to the transport layer within t_{P2*_Server}. Server#1 stops the t_{P2_Server} timer.

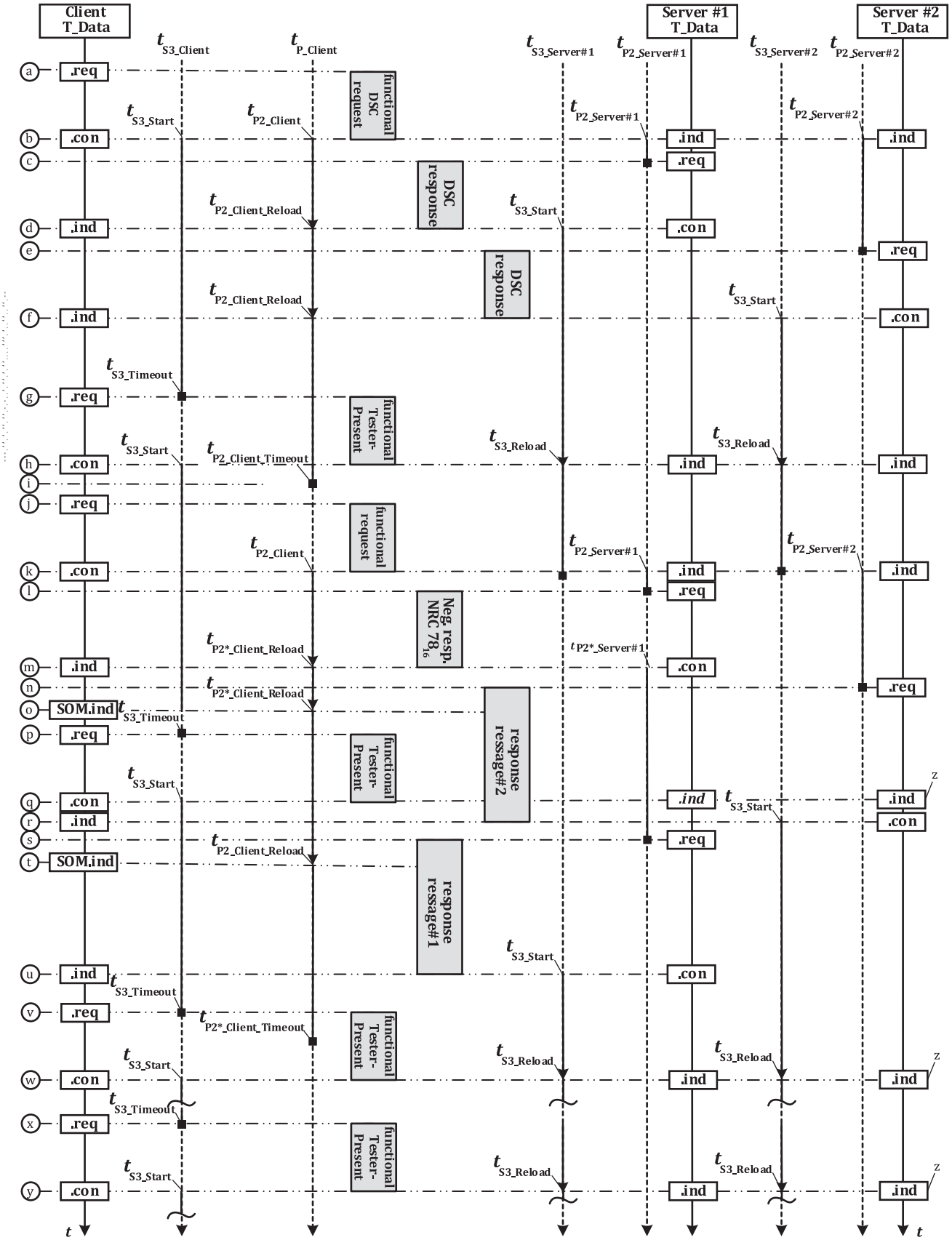
i Client T_DataSOM.ind: transport layer issues to the diagnostic application the reception of a StartOfMessage. The entry matching the response message address is removed from the pending list and the list is now empty. That means no further response messages are pending. This causes the client to restart its t_{P_Client} timer using the default reload value t_{P2_Client}.

j Server #1 T_Data.conf: transport layer issues to the diagnostic application the completion of the response message. Client T_Data.ind: transport layer issues to the diagnostic application the completion of the response message.

Figure 16 — Functional communication during default session – Enhanced response timing – with SOM.ind

10.2.4 Functional communication during non-default session – with SOM.ind

Figure 17 graphically depicts the timing handling in the client and two servers for a functionally-addressed request message during the non-default session (e.g. programmingSession), where one server requests an enhanced response timing via a negative response message including negative response code 78_{16}.



Key

a Client `T_Data.req`: diagnostic application issues the functional addressed DiagnosticSessionControl (10_{16}) request message to the transport layer. The transport layer transmits the request message to the servers.

b Client T_Data.conf: transport layer issues to the diagnostic application the reception of the DiagnosticSessionControl (10_{16}) request message. Now the response timing t_{P_Client} as described in 10.1.2 and 10.1.3 applies. The value of the t_{P_Client} timer shall consider any latency that is involved based on the vehicle network design (e.g. communication over gateways, bus bandwidth, etc.). For simplicity, the figure assumes that the client and the servers are located on the same network. The generated T_Data.conf in the client causes the start of the t_{S3_Client} timer (session timer). All Server T_Data.ind: transport layer issues to the diagnostic application the completion of the DiagnosticSessionControl (10_{16}) request message. All servers start the t_{P2_Server} timer using the value of $t_{P2\_Server} = t_{P2\_Server\_Max}$. Now the response timing t_{P2_Server} as described in 10.1.2 and 10.1.3 applies.

c Server#1 T_Data.req: diagnostic application issues the DiagnosticSessionControl (10_{16}) positive response message to the transport layer within $t_{P2\_Server\_Max}$. Server#1 stops the t_{P2_Server} timer. For the figure given, it is assumed that the client requires a response from the server.

d Server#1 T_Data.conf: the completion of the transmission of the DiagnosticSessionControl (10_{16}) positive response message is indicated in the server#1 via T_Data.conf. Now the server#1 starts its t_{S3_Server} timer, which keeps the activated non-default session active as long as it does not time out. It is the client's responsibility to ensure that the t_{S3_Server} timer is reset prior to its timeout to keep the server#1 in the non-default session. Client T_Data.ind: transport layer issues to the diagnostic application the completion of the response message. This causes the client to restart its t_{P_Client} timer, using the default reload value t_{P2_Client}.

e Server#2 T_Data.req: diagnostic application issues the DiagnosticSessionControl (10_{16}) positive response message to the transport layer within t_{P2_Server}. Server#2 stops the t_{P2_Server} timer. For the figure given, it is assumed that the client requires a response from the server.

f Server#2 T_Data.conf: the completion of the transmission of the DiagnosticSessionControl (10_{16}) positive response message is indicated in the server#2 via T_Data.conf. Now the server#2 starts its t_{S3_Server} timer, which keeps the activated non-default session active as long as it does not time out. It is the client's responsibility to ensure that the t_{S3_Server} timer is reset prior to its timeout to keep the server#2 in the non-default session. Client T_Data.ind: transport layer issues to the diagnostic application the completion of the response message. This causes the client to restart its t_{P_Client} timer, using the default reload value t_{P2_Client}.

g Client T_Data.req: once the t_{S3_Client} timer is started in the client, this causes the transmission of a functionally-addressed TesterPresent ($3E_{16}$) request message, which does not require a response message, each time the t_{S3_Client} timer times out.

h Client T_Data.conf: upon the indication of the completed transmission of the TesterPresent ($3E_{16}$) request message via T_Data.conf of its transport layer, the client once again starts its t_{S3_Client} timer. This means that the functionally-addressed TesterPresent ($3E_{16}$) request message is sent on a periodic basis every time t_{S3_Client} times out. All Server T_Data.ind: any TesterPresent ($3E_{16}$) request message that is received during an activated t_{S3_Server} timer reloads the t_{S3_Server} timer.

i Client: this is the indication for the client that no further response messages are expected and can continue with further requests.

j Client T_Data.req: diagnostic application issues a functionally-addressed request message to the transport layer.

k All server T_Data.ind: transport layer issues to the diagnostic application the completion of a request message. All servers start the t_{P2_Server} timer using the value of $t_{P2\_Server} = t_{P2\_Server\_Max}$. Any time the server is in the process of handling any diagnostic service, it stops its t_{S3_Server} timer. Client T_Data.conf: transport layer issues to the diagnostic application the confirmation of the completion of the request message. Client starts its t_{P_Client} timer using the default reload value $t_{P2\_Client} = t_{P2\_Client\_Max}$. The value of the t_{P_Client} timer shall consider any latency that is involved based on the vehicle network design (e.g. communication over gateways, bus bandwidth, etc.). For simplicity, the figure assumes that the client and the servers are located on the same network.

l Server #1 T_Data.req: diagnostic application does not have the positive response message ready and issues negative response message with NRC = 78_{16} by a T_Data.req to the transport layer within t_{P2_Server}. Server#1 stops the t_{P2_Server} timer. In case any of the addressed servers cannot provide the requested information within the t_{P2_Server} response timing, it can request an enhanced response timing window by sending a negative response message including NRC 78_{16}.

m Server#1 T_Data.conf: transport layer issues to the diagnostic application the completion of the response message. Server#1 starts the t_{P2_Server} timer using the value of $t_{P2*\_Server} = t_{P2*\_Server\_Max}$ (default enhanced timing). Client T_Data.ind: transport layer issues to the diagnostic application the reception of a response message. Client reloads the t_{P_Client} timer with the value of $t_{P2*\_Client} = t_{P2*\_Client\_Max}$ (default enhanced timing). An entry containing the response message address is added to the NRC 78_{16} pending list.

n Server#2 T_Data.req: diagnostic application has prepared the response message and issues a T_Data.req to the transport layer within t_{P2_Server}. Server#2 stops the t_{P2_Server} timer.

o Client T_DataSOM.ind: transport layer issues to the diagnostic application the reception of a StartOfMessage. This causes the client to restart its t_{P_Client} timer, using the default reload value t_{P2*_Client} (default enhanced timing).

p Client T_Data.req: once the t_{S3_Client} timer is started in the client, this causes the transmission of a functionally-addressed TesterPresent ($3E_{16}$) request message, which does not require a response message, each time the t_{S3_Client} timer times out.

q Client T_Data.conf: upon the indication of the completed transmission of the TesterPresent ($3E_{16}$) request message via T_Data.conf of its transport layer, the client once again starts its t_{S3_Client} timer. This means that the functionally-addressed TesterPresent ($3E_{16}$) request message is sent on a periodic basis every time t_{S3_Client} times out. All Server T_Data.ind: any TesterPresent ($3E_{16}$) request message that is received during an activated t_{S3_Server} timer reloads the t_{S3_Server} timer. Any TesterPresent that is received during a disabled t_{S3_Server} timer is ignored by the server(s).

r Client T_Data.ind: transport layer issues to the diagnostic application the completion of the response message. Server#2 T_Data.conf: when the diagnostic service is completely processed, then the server#2 restarts its t_{S3_Server} timer. This means that any diagnostic service, including TesterPresent ($3E_{16}$), resets the t_{S3_Server} timer. A diagnostic service is considered to be in progress any time between the start of the reception of the request message (T_DataSOM.ind or T_Data.ind receive) and the completion of the transmission of the final response message, where a response message is required, or the completion of any action that is caused by the request, where no response message is required (point in time reached that would cause the start of the response message).

s Server#1 T_Data.req: diagnostic application has prepared the response message and issues a T_Data.req to the transport layer within t_{P2*_Server}. Server#1 stops the t_{P2_Server} timer.

t Client T_DataSOM.ind: transport layer issues to the diagnostic application the reception of a StartOfMessage. The entry matching the response message address is removed from the pending list and the list is now empty. That means no further response messages are pending. This causes the client to restart its t_{P_Client} timer using the default reload value t_{P2_Client}.

u Client T_Data.ind: transport layer issues to the diagnostic application the completion of the response message. Server#1 T_Data.conf: transport layer issues to the diagnostic application the completion of the response message. Now the server#1 starts its t_{S3_Server} timer.

v Client T_Data.req: Each time the t_{S3_Client} timer times out causes the transmission of a functionally-addressed TesterPresent ($3E_{16}$) request message, which does not require a response message.

w Client T_Data.conf: upon the indication of the completed transmission of the TesterPresent ($3E_{16}$) request message via T_Data.conf of its transport layer, the client once again starts its t_{S3_Client} timer. This means that the functionally-addressed TesterPresent ($3E_{16}$) request message is sent on a periodic basis every time t_{S3_Client} times out. All Server T_Data.ind: any TesterPresent ($3E_{16}$) request message that is received during an activated t_{S3_Server} timer reloads the t_{S3_Server} timer.

x Client T_Data.req: once the t_{S3_Client} timer is started in the client, this causes the transmission of a functionally-addressed TesterPresent ($3E_{16}$) request message, which does not require a response message, each time the t_{S3_Client} timer times out.

y Client T_Data.conf: upon the indication of the completed transmission of the TesterPresent ($3E_{16}$) request message via T_Data.conf of its transport layer, the client once again starts its t_{S3_Client} timer. This means that the functionally-addressed TesterPresent ($3E_{16}$) request message is sent on a periodic basis every time t_{S3_Client} times out. All Server T_Data.ind: any TesterPresent ($3E_{16}$) request message that is received during an activated t_{S3_Server} timer reloads the t_{S3_Server} timer.

z Any TesterPresent that is received during a disabled t_{S3_Server} timer is ignored by the server.

Figure 17 — Functional communication during non-default session – with SOM.ind

The handling of the t_{P_Client} and t_{P2_Server} timing is identical to the handling as described in 10.1.2 and 10.1.3, the only exception being that the reload values on the client side and the resulting time the server shall send its final response time might differ. This is based on the transition into a session other than the default session where different t_{P_Client} timing parameters might apply (see DiagnosticSessionControl (10_{16}) service in ISO 14229-1 for details on how the timing parameters are reported to the client).

10.3 Minimum time between client request messages

The minimum time between request messages transmitted by the client is required in order to allow for a polling driven service data interpretation in the server. Based on normal functionality, a server might process diagnostic request messages with a design-specific scheduling rate (e.g. 10 ms). The time for the diagnostic service data interpretation scheduler shall be smaller than the performance requirement t_{P2_Server} in order to meet the server requirements as specified in 10.1.1, 10.1.2, and 10.1.3.

The timing parameter for the minimum time between request messages is divided into the following two-timing parameters.

— $t_{P3\_Client\_Func}$: this timing parameter applies to any functionally-addressed request message, because it can be the case that a server is not required to respond to a functionally-addressed request message if it does not support the requested data.

— $t_{P3\_Client\_Phys}$: this timing parameter applies to any physically-addressed request message where there is no response required to be transmitted by the server (suppressPosRspMsgIndicationBit = TRUE).

In the case of physical communication where a response is required by the server, the client can transmit the next request immediately after the complete reception of the last response message, because the server has responded completely to the request — which means that the request is completely handled by the server.

Figure 18 graphically depicts an example of a problem that can occur during functional communication, when the client transmits the next request immediately after it has determined that all expected servers responded to a previous request message.

This scenario does not only apply to functionally-addressed requests but also to physically-addressed requests where the client does not want to receive any response message (suppressPosRspMsgIndicationBit = TRUE).

In order to handle the described scenarios, the minimum times $t_{P3\_Client\_Phys}$ and $t_{P3\_Client\_Func}$, between the end of a physically- or functionally-addressed request message and the start of a new physically- or functionally-addressed request message, are defined for the client.

a) The value of $t_{P3\_Client\_Phys}$ is identical to $t_{P2\_Server\_Max}$ for the physically-addressed server. The timing applies to any physically-addressed request message in any diagnostic session (default and non-default session) and in case no response is required by the server.

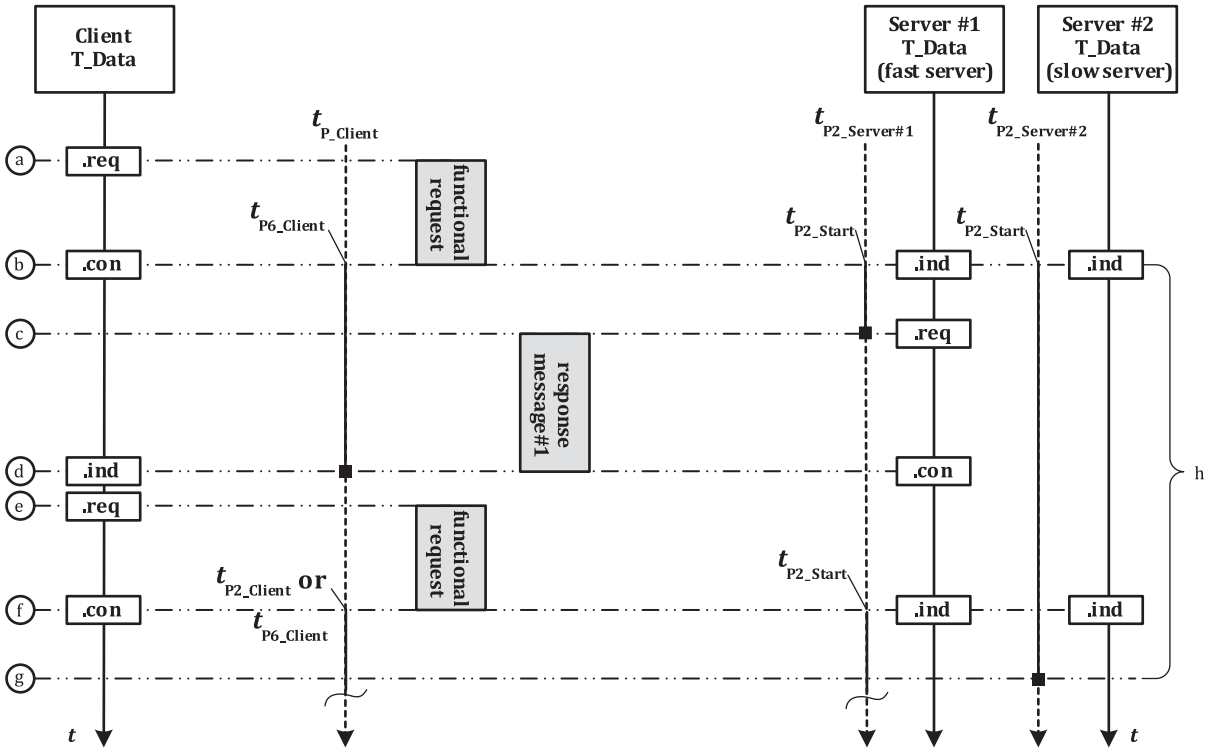
   The $t_{P3\_Client\_Phys}$ timer is started in the client each time a physically-addressed request message with no response required is successfully transmitted onto the bus, which is indicated via `T_Data.conf` in the client. When the client wants to transmit a new physically-addressed request message following a previous request that was completely handled, then this is only allowed in case the $t_{P3\_Client\_Phys}$ timer is no longer active at the time the client wants to transmit the physically-addressed request message. In case $t_{P3\_Client\_Phys}$ would still be active at the point in time the client would like to transmit a new physically-addressed request message, then the transmission shall be postponed until $t_{P3\_Client\_Phys}$ is timed out.

b) The value of $t_{P3\_Client\_Func}$ is the maximum (worst-case) value of all functionally-addressed server's $t_{P2\_Server\_Max}$ for any functionally-addressed request message in any diagnostic session (default and non-default session).

   The $t_{P3\_Client\_Func}$ timer is started in the client each time a functionally-addressed request message with response required or with no response required is successfully transmitted onto the bus, which is indicated via `T_Data.conf` in the client. When the client wants to transmit a new functionally-addressed request message following a previous request that was completely handled, then this is only allowed in case the $t_{P3\_Client\_Func}$ timer is no longer active at the time the client wants to transmit the functionally-addressed request message. In case $t_{P3\_Client\_Func}$ would still be active at the point in time the client would like to transmit a new functionally-addressed request message, then the transmission shall be postponed until $t_{P3\_Client\_Func}$ is timed out.

NOTE "Completely handled" means that no response is received in case no response is required. Additionally, it means that all expected responses to a functionally-addressed request are received in case the responding servers are known and responses are required or that a t_{P_Client} timeout occurred in case the responding servers are not known and responses are required.

The requirement for the server is that it shall start with its response message within t_{P2_Server}. This means that the diagnostic data interpretation rate of the server shall be less than $t_{P2\_Server\_Max}$.



Key

a Client `T_Data.req`: diagnostic application issues a request message to the transport layer.

b Client `T_Data.conf`: transport layer issues to the diagnostic application the confirmation of the completion of the request message. Client starts its t_{P_Client} timer using the default reload value $t_{P2\_Client} = t_{P2\_Client\_Max}$. All Server `T_Data.ind`: transport layer issues to the diagnostic application the completion of the request message. All servers start the t_{P2_Server} timer using the default value of $t_{P2\_Server} = t_{P2\_Server\_Max}$.

c Server#1 `T_Data.req`: diagnostic application has prepared the response message and issues a `T_Data.req` to the transport layer within t_{P2_Server}. Server#1 stops the t_{P2_Server} timer. For the request message, it is assumed that only server #1 supports the requested information, which means that there is no response from server#2. Server#1 is a fast server and can immediately process the received request message and transmits its response within t_{P2_Server}.

d Client `T_Data.ind`: transport layer issues to the diagnostic application the completion of the response message. The client only expected a response message from server #1, therefore it stops its timer t_{P_Client}. Server#1 `T_Data.conf`: transport layer issues to the diagnostic application the completion of the response message.

e Client `T_Data.req`: diagnostic application issues a request message to the transport layer. The client would send the next request right after the completion of all expected response messages.

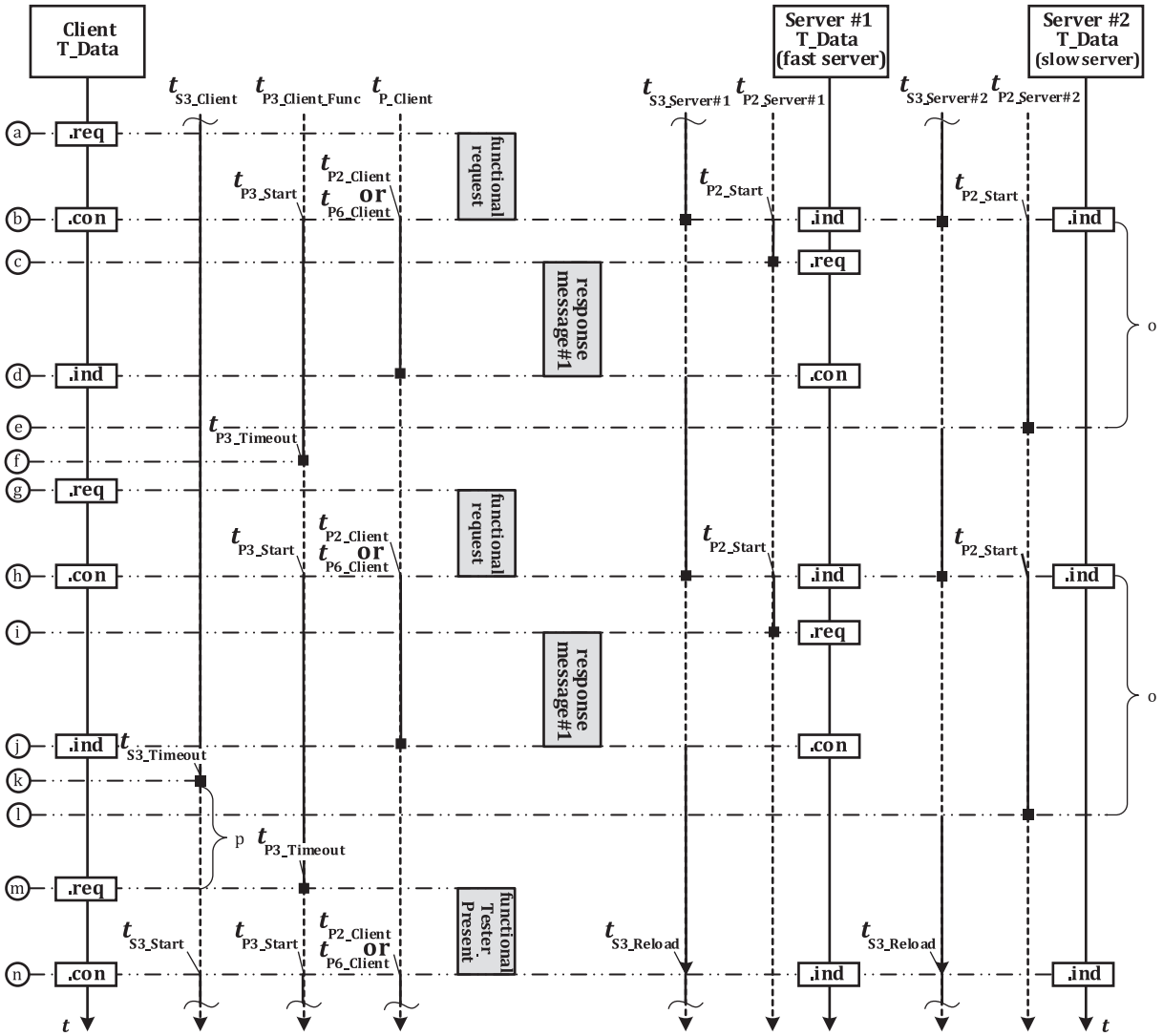
f Client `T_Data.conf`: transport layer issues to the diagnostic application the confirmation of the completion of the request message. Client starts its t_{P_Client} timer using the default reload value $t_{P2\_Client} = t_{P2\_Client\_Max}$. Server#1 `T_Data.ind`: transport layer issues to the diagnostic application the completion of the request message. Server#1 starts the t_{P2_Server} timer using the default value of $t_{P2\_Server} = t_{P2\_Server\_Max}$. Server#2 `T_Data.ind`: transport layer issues to the diagnostic application the completion of the request message. The request message is only processed by the fast server#1, because server#2 did not yet handle the previous request. Any request message that is received during the diagnostic service data interpretation rate of server#2 is ignored by the server#2.

g Server #2 is a slow server and interprets received requests on a periodic basis (diagnostic service data interpretation rate). In the worst-case, the last check for incoming request a message is prior to the transport layer reception of the functionally-addressed request message. This would mean that the request would be stored in a buffer and processed at the earliest the next time the scheduler checks for an incoming request. When server#2 processes the request, then it determines that it does not need to answer, because it does not support the requested information (e.g. server sends no response message because the negative response code requestOutOfRange is suppressed in case of a functionally-addressed request message). As shown in the figure, this would be after the completion of the response message of server#1, and even after the completion of the next request message transmitted by the client.

h Diagnostic service data interpretation rate of server#2.

Figure 18 — Example of critical issue when transmitting next request too early

Figure 19 graphically depicts the $t_{P3\_Client\_Func}$ timing handling for the client (based on the communication scenario illustrated in Figure 17). In addition, Figure 19 shows the handling of a functionally-addressed TesterPresent ($3E_{16}$) request message in the client in the case in which the $t_{P3\_Client\_Func}$ timer is still active when t_{S3_Client} times out (request is postponed until $t_{P3\_Client\_Func}$ times out).



Key

a Client `T_Data.req`: diagnostic application issues a request message to the transport layer.

b Client `T_Data.conf`: transport layer issues to the diagnostic application the confirmation of the completion of the request message. Client starts its t_{P_Client} timer using the default reload value $t_{P6\_Client} = t_{P6\_Client\_Max}$ and, furthermore, its $t_{P3\_Client\_Func}$ timer. All Server `T_Data.ind`: transport layer issues to the diagnostic application the completion of the request message. All servers start the t_{P2_Server} timer using the default value of $t_{P2\_Server} = t_{P2\_Server\_Max}$. All servers stop the t_{S3_Server} timer.

c Server#1 `T_Data.req`: diagnostic application has prepared the response message and issues a `T_Data.req` to the transport layer within t_{P2_Server}. Server#1 stops the t_{P2_Server} timer. For the request message, it is assumed that only server #1 supports the requested information, which means that there is no response from server#2. Server#1 is a fast server and can immediately process the received request message and transmits its response within t_{P2_Server}.

d Client `T_Data.ind`: transport layer issues to the diagnostic application the completion of the response message. The client only expected a response message from server #1, therefore it stops its t_{P_Client} timer. Server#1 `T_Data.conf`: transport layer issues to the diagnostic application the completion of the response message. Now the server#1 starts its t_{S3_Server} timer.

e Server #2 is a slow server and interprets received requests on a periodic basis (diagnostic service data interpretation rate). In the worst-case, the last check for incoming request a message is prior to the transport layer reception of the functionally-addressed request message. This would mean that the request would be stored in a buffer and processed at the earliest the next time the scheduler checks for an incoming request. When server#2 processes the request, then it determines that it does not need to answer, because it does not support the requested information (e.g. server sends no response message because the negative response code requestOutOfRange is suppressed in case of a functionally-addressed request message). Now the server#2 stops the t_{P2_Server} timer and starts its t_{S3_Server} timer.

f Client: Even if the client has received all expected response messages to a functionally-addressed request message, it shall wait until $t_{P3\_Client\_Func}$ times out before it is allowed to transmit the next request message. At the point in time $t_{P3\_Client\_Func}$ times out.

g Client `T_Data.req`: diagnostic application issues a request message to the transport layer. The client would send the next request right after the completion of all expected response messages.

h Client `T_Data.conf`: transport layer issues to the diagnostic application the confirmation of the completion of the request message. Client starts its t_{P_Client} timer using the default reload value $t_{P6\_Client} = t_{P6\_Client\_Max}$ and, furthermore, its timer $t_{P3\_Client\_Func}$. All Server `T_Data.ind`: transport layer issues to the diagnostic application the completion of the request message. All servers start the t_{P2_Server} timer using the default value of $t_{P2\_Server} = t_{P2\_Server\_Max}$. All servers stop the t_{S3_Server} timer.

i Server#1 `T_Data.req`: diagnostic application has prepared the response message and issues a `T_Data.req` to the transport layer within $t_{P2\_Server\_Max}$. Server#1 stops the t_{P2_Server} timer. For the request message, it is assumed that only server #1 supports the requested information, which means that there is no response from server#2. Server#1 is a fast server and can immediately process the received request message and transmits its response within t_{P2_Server}.

j Client `T_Data.ind`: transport layer issues to the diagnostic application the completion of the response message. The client only expected a response message from server #1, therefore it stops its t_{P_Client} timer. Server#1 `T_Data.conf`: transport layer issues to the diagnostic application the completion of the response message. Now the server#1 starts its t_{S3_Server} timer.

k Client: The t_{S3_Client} timer of the client times out, which forces the client to transmit a functionally-addressed TesterPresent ($3E_{16}$) request message, not requiring a response message from the addressed server(s). Based on the situation in which the $t_{P3\_Client\_Func}$ timer is still active at this point in time, the transmission of the TesterPresent ($3E_{16}$) shall be postponed until the expiration of the $t_{P3\_Client\_Func}$ timer.

l Server #2 is a slow server and interprets received requests on a periodic basis (diagnostic service data interpretation rate). In the worst-case, the last check for incoming request a message is prior to the transport layer reception of the functionally-addressed request message. This would mean that the request would be stored in a buffer and processed at the earliest the next time the scheduler checks for an incoming request. When server#2 processes the request, then it determines that it does not need to answer, because it does not support the requested information (e.g. server sends no response message because the negative response code requestOutOfRange is suppressed in case of a functionally-addressed request message). Now the server#2 stops the t_{P2_Server} timer and starts its t_{S3_Server} timer.

m Client `T_Data.req`: When the $t_{P3\_Client\_Func}$ timer times out, the functionally-addressed TesterPresent ($3E_{16}$) request can be transmitted by the client via `T_Data.req`.

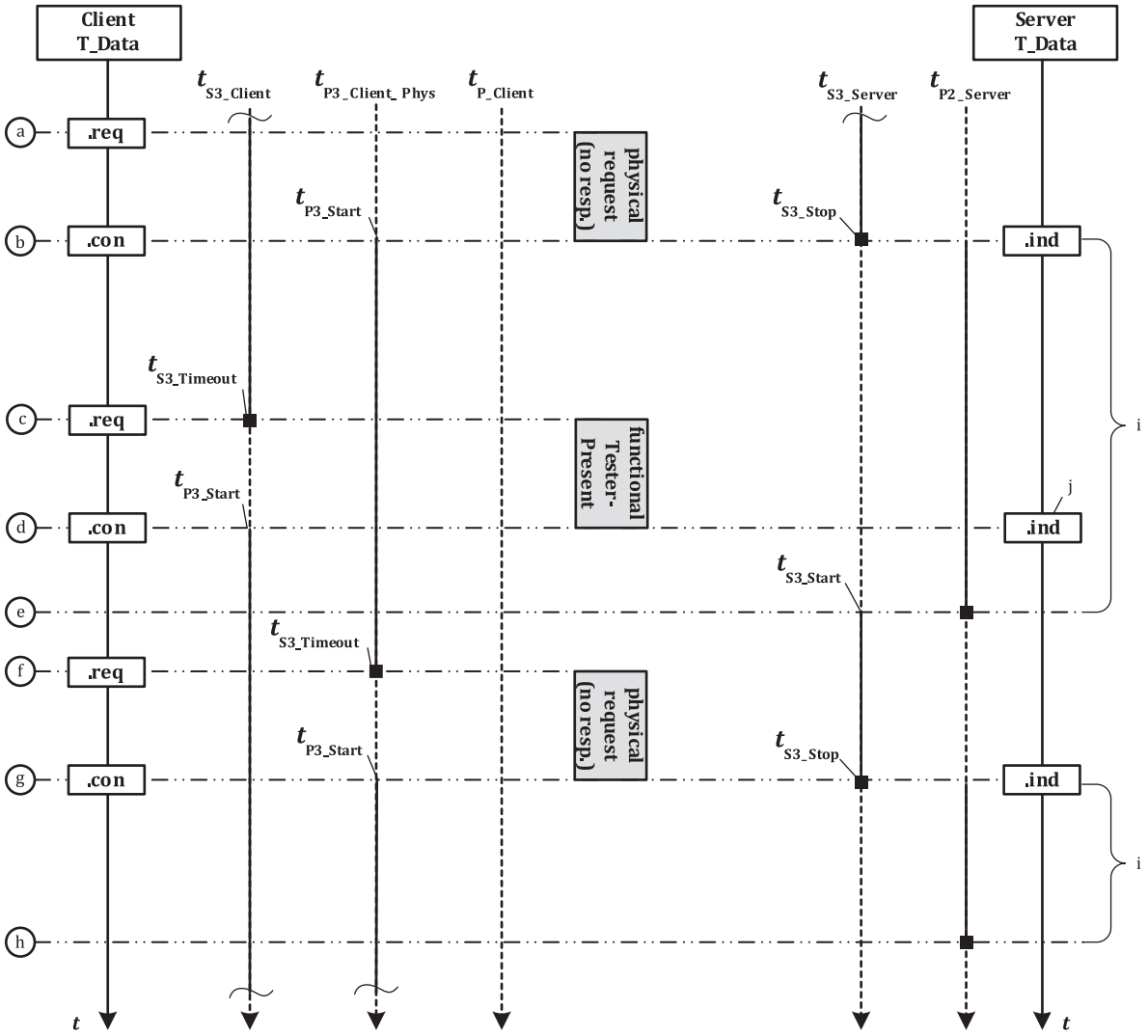
n Client `T_Data.conf`: upon the indication of the completed transmission of the TesterPresent ($3E_{16}$) request message via `T_Data.conf` of its transport layer, the client once again starts its t_{S3_Client} timer. This means that the functionally-addressed TesterPresent ($3E_{16}$) request message is sent on a periodic basis every time t_{S3_Client} times out. Furthermore, client starts its $t_{P3\_Client\_Func}$ timer. All Server `T_Data.ind`: transport layer issues to the diagnostic application the completion of the request message. Any TesterPresent ($3E_{16}$) request message that is received during an activated t_{S3_Server} timer reloads the t_{S3_Server} timer.

o Diagnostic service data interpretation rate of server#2.

p Functional TesterPresent delay.

Figure 19 — Minimum time between functionally-addressed request messages ($t_{P3\_Client\_Phys}$)

Figure 20 graphically depicts the $t_{P3\_Client\_Phys}$ timing handling for the client. The figure shows the handling of a physically-addressed request that does not require a response and of the functionally-addressed TesterPresent ($3E_{16}$) request message in the client when t_{S3_Client} times out.



Key

a Client `T_Data.req`: diagnostic application issues a request message to the transport layer that does not require a response.

b Client `T_Data.conf`: transport layer issues to the diagnostic application the confirmation of the completion of the request message. Client starts its $t_{P3\_Client\_Phys}$ timer using the default reload value $t_{P3\_Client\_Phys} = t_{P3\_Client\_Phys\_max}$. There is no response required to be transmitted, therefore the client does not need to start its t_{P_Client} timer. Server `T_Data.ind`: transport layer issues to the diagnostic application the completion of the request message. Server starts the t_{P2_Server} timer using the default value of $t_{P2\_Server} = t_{P2\_Server\_Max}$ and in any non-default session the t_{S3_Server} timer is now stopped.

c Client `T_Data.req`: The t_{S3_Client} timer of the client times out, which forces the client to transmit a functionally-addressed TesterPresent ($3E_{16}$) request message, not requiring a response message from the addressed server(s). It is assumed that the $t_{P3\_Client\_Func}$ timer is no active at this point in time, which means that the request is transmitted immediately.

d Client `T_Data.conf`: upon the indication of the completed transmission of the TesterPresent ($3E_{16}$) request message via `T_Data.conf` of its transport layer, the client once again starts its t_{S3_Client} timer. This means that the functionally-addressed TesterPresent ($3E_{16}$) request message is sent on a periodic basis every time t_{S3_Client} times out. Server `T_Data.ind`: any TesterPresent ($3E_{16}$) request message that is received during processing another request message can be ignored by the server, because it has already stopped its t_{S3_Server} timer and restarts it once the service that is in progress is processed completely.

e Server: Server interprets received requests on a periodic basis (diagnostic service data interpretation rate). The request is processed the next time the scheduler checks for incoming requests. The completed execution of the service would restart the t_{S3_Server} timer during any non-default session and stops the t_{P2_Server} timer.

f Client `T_Data.req`: when the $t_{P3\_Client\_Phys}$ timer times out in the client, the client can transmit the next physically-addressed request message by issuing `T_Data.req` to its transport layer.

g Client `T_Data.conf`: transport layer issues to the diagnostic application the confirmation of the completion of the request message. The client now starts its $t_{P3\_Client\_Phys}$ timer again. There is no response required to be transmitted, therefore the client does not need to start its t_{P_Client} timer. Server `T_Data.ind`: transport layer issues to the diagnostic application the completion of the request message. Server starts the t_{P2_Server} timer using the default value of $t_{P2\_Server} = t_{P2\_Server\_Max}$ and in any non-default session the t_{S3_Server} timer is now stopped.

h Server: Server interprets received requests on a periodic basis (diagnostic service data interpretation rate). The request is processed the next time the scheduler checks for incoming requests. The completed execution of the service would restart the t_{S3_Server} timer during any non-default session and stops the t_{P2_Server} timer.

i Diagnostic service data interpretation rate.

j Any TesterPresent that is received during a disabled t_{S3_Server} timer can be ignored by the server.

Figure 20 — Minimum time between physically-addressed request messages ($t_{P3\_Client\_Phys}$)

Annex A
(normative)

T_PDU interface

Figure A.1 shows the T_PDU (virtual PDU) as an interface between the unified diagnostic services PDU and any communication protocol.

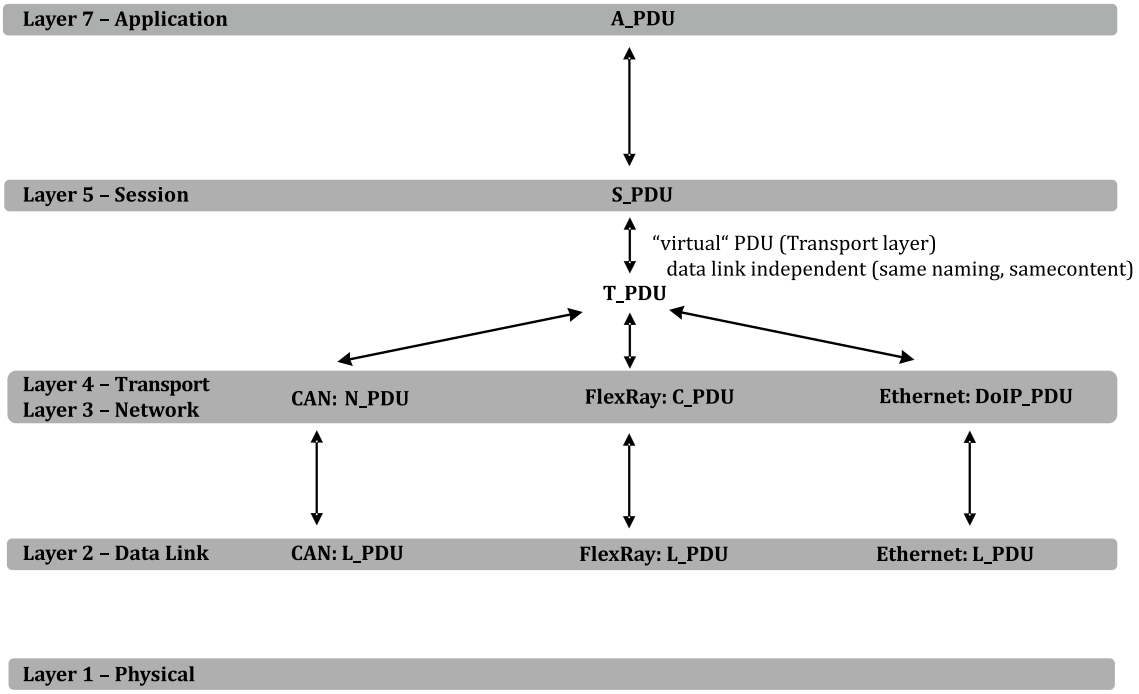


Figure A.1 — T_PDU virtual PDU interface to any communication protocol

Annex B
(informative)

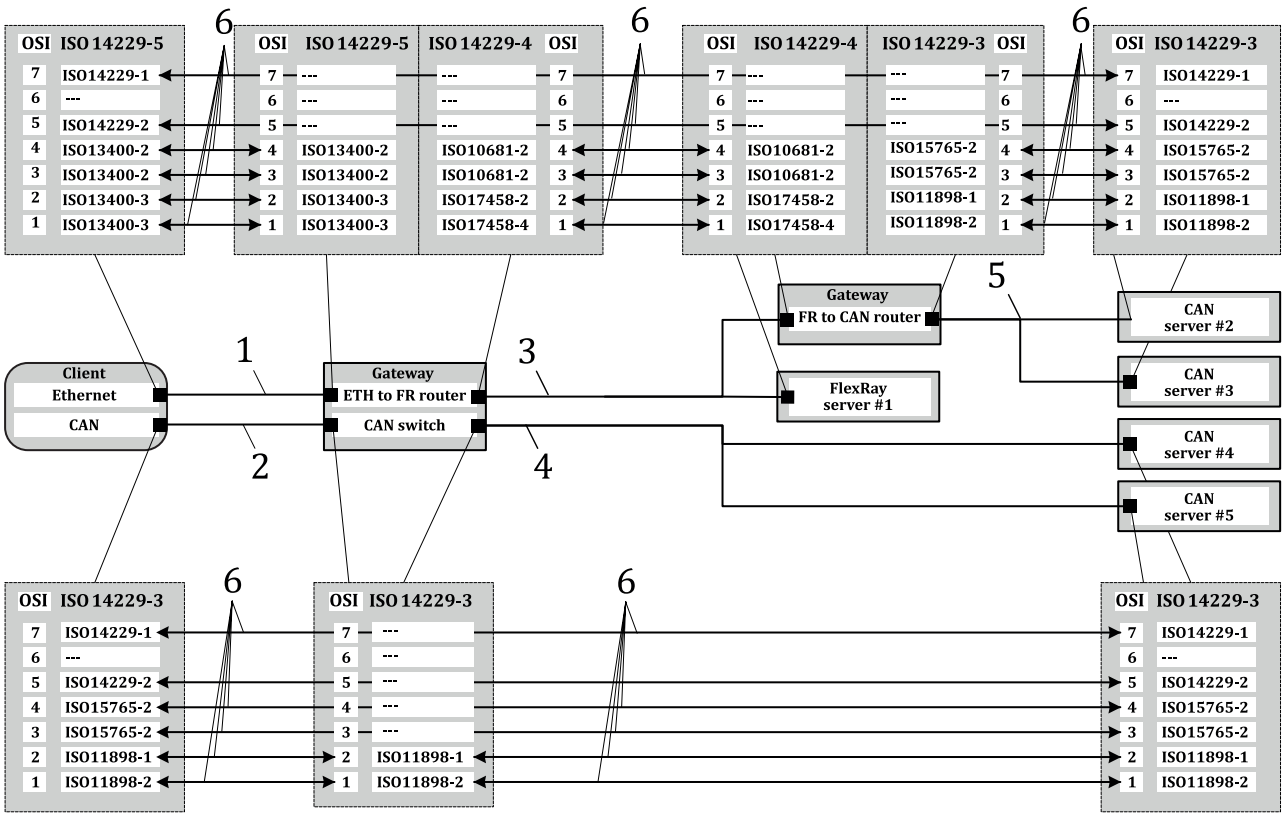
Vehicle diagnostic OSI layer architecture examples

B.1 Vehicle diagnostic OSI layer gateway example

Figure B.1 depicts an example of a vehicle diagnostic network architecture with different network technologies and two gateway instances. The “Ethernet to FlexRay router” implemented in the gateway device is a networking device that transfers the PDU on OSI layers 3 and 4. The “CAN switch” implemented in the gateway device is a networking device that transfers the PDU on OSI layer 2.

The framed boxes (dashed line) provide the ISO/OSI layer model about the different protocols which shall be supported by the corresponding interface.

Key 1 and key 2 show vehicle diagnostic data access from a client to a vehicle gateway. Key 3 to key 5 are examples for in-vehicle communication networks, while key 5 is located behind a second gateway (FlexRay to CAN router).



Key

1 diagnostic ethernet (ETH)
2 diagnostic CAN
3 FlexRay communications system (FR)
4 CAN
5 CAN
6 logical connection
— not applicable

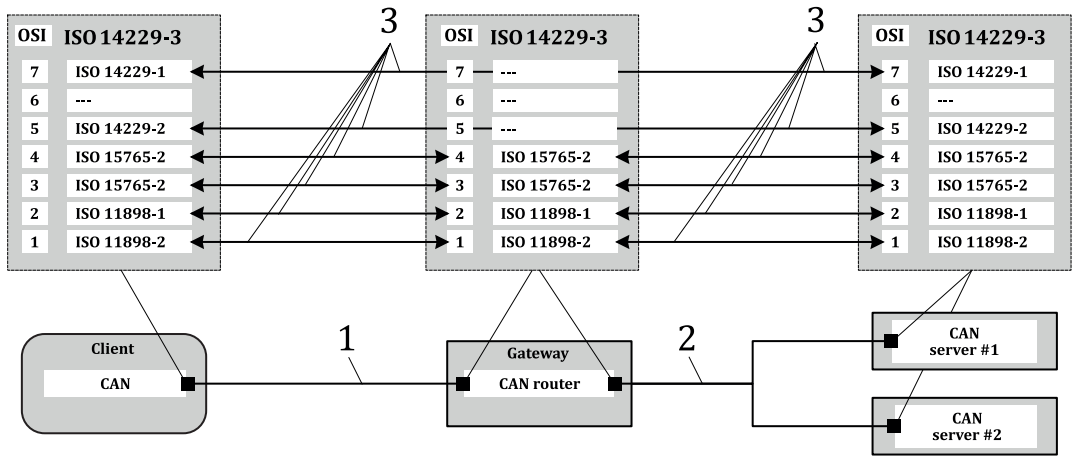
Figure B.1 — Vehicle diagnostic OSI layer gateway example

B.2 Vehicle diagnostic OSI layer CAN router example

Figure B.2 depicts an example of a vehicle diagnostic network architecture with a “CAN router” implemented in a gateway. The “CAN router” implemented in the gateway device is a networking device that transfers the PDU on OSI layers 3 and 4.

The framed boxes (dashed line) provide the ISO/OSI layer model about the different protocols which shall be supported by the corresponding interface.

Key 1 shows vehicle diagnostic data access from a client to a vehicle gateway. Key 2 is an example for an in-vehicle CAN network.



Key

1 diagnostic CAN

2 CAN

3 logical connection

— not applicable

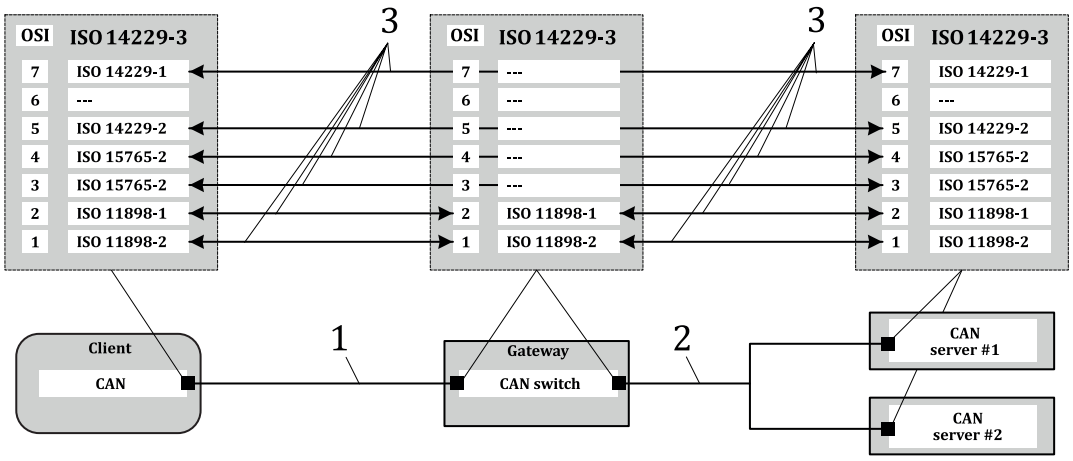
Figure B.2 — Vehicle diagnostic OSI layer CAN router example

B.3 Vehicle diagnostic OSI layer CAN switch example

Figure B.3 depicts an example of a vehicle diagnostic network architecture with a "CAN switch" implemented in a gateway. The "CAN switch" implemented in the gateway device is a networking device that transfers the PDU on OSI layer 2.

The framed boxes (dashed line) provide the ISO/OSI layer model about the different protocols which shall be supported by the corresponding interface.

Key 1 shows vehicle diagnostic data access from a client to a vehicle gateway. Key 2 is an example for an in-vehicle CAN network.



Key

1 diagnostic CAN
2 CAN
3 logical connection
— not applicable

Figure B.3 — Vehicle diagnostic OSI layer CAN switch example

Bibliography

[1] ISO/IEC 10731:1994, *Information technology — Open Systems Interconnection — Basic Reference Model — Conventions for the definition of OSI services*

[2] ISO 10681-2, *Road vehicles — Communication on FlexRay — Part 2: Communication layer services*

[3] ISO 13400-2, *Road vehicles — Diagnostic communication over Internet Protocol (DoIP) — Part 2: Transport protocol and network layer services*

[4] ISO 14230-2, *Road vehicles — Diagnostic communication over K-Line (DoK-Line) — Part 2: Data link layer*

[5] ISO 15765-2, *Road vehicles — Diagnostic communication over Controller Area Network (DoCAN) — Part 2: Transport protocol and network layer services*

[6] ISO 20794-3, *Road vehicles — Clock extension peripheral interface (CXPI) — Part 3: Transport and network layer*

ICS 43.180

Price based on 53 pages