

Road vehicles — Unified diagnostic services (UDS) — Specification and requirements

ICS 43.180

National foreword

This British Standard was published by BSI. It is the UK implementation of ISO 14229:2006.

The UK participation in its preparation was entrusted to Technical Committee AUE/16, Electrical and electronic equipment.

A list of organizations represented on AUE/16 can be obtained on request to its secretary.

This publication does not purport to include all the necessary provisions of a contract. Users are responsible for its correct application.

Compliance with a British Standard cannot confer immunity from legal obligations.

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 31 January 2007

© BSI 2007

ISBN 978-0-580-49925-8

Amendments issued since publication

Amd. No.	Date	Comments

INTERNATIONAL STANDARD

**ISO
14229**

Second edition
2006-12-01

Road vehicles — Unified diagnostic services (UDS) — Specification and requirements

*Véhicules routiers — Services de diagnostic unifiés (SDU) —
Spécification et exigences*



Reference number
ISO 14229:2006(E)



Documents

Document Sharing Hub

Contents

Page

Foreword.....	v
Introduction	vi
1 Scope	1
2 Normative references	2
3 Terms and definitions.....	3
4 Symbols and abbreviated terms	5
5 Conventions	5
6 Application layer services	6
6.1 General.....	6
6.2 Format description of application layer services	8
6.3 Format description of standard service primitives	8
6.4 Format description of remote service primitives	10
6.5 Service data unit specification	13
7 Application layer protocol	19
7.1 General definition	19
7.2 Protocol data unit specification	19
7.3 Application protocol control information.....	19
7.4 Negative response/confirmation service primitive	21
7.5 Server response implementation rules	22
8 Service description conventions	29
8.1 Service description.....	29
8.2 Request message	30
8.3 Positive response message.....	32
8.4 Supported negative response codes (NRC_)	34
8.5 Message flow examples	34
9 Diagnostic and communication management functional unit	36
9.1 Overview.....	36
9.2 DiagnosticSessionControl (10 hex) service.....	36
9.3 ECUReset (11 hex) service	42
9.4 SecurityAccess (27 hex) service	45
9.5 CommunicationControl (28 hex) service.....	52
9.6 TesterPresent (3E hex) service	55
9.7 AccessTimingParameter (83 hex) service.....	58
9.8 SecuredDataTransmission (84 hex) service	63
9.9 ControlDTCSetting (85 hex) service	69
9.10 ResponseOnEvent (86 hex) service.....	73
9.11 LinkControl (87 hex) service.....	91
10 Data transmission functional unit.....	97
10.1 Overview	97
10.2 ReadDataByIdentifier (22 hex) service	97
10.3 ReadMemoryByAddress (23 hex) service	102
10.4 ReadScalingDataByIdentifier (24 hex) service	106
10.5 ReadDataByPeriodicIdentifier (2A hex) service	112
10.6 DynamicallyDefinedDataIdentifier (2C hex) service	123
10.7 WriteDataByIdentifier (2E hex) service.....	143
10.8 WriteMemoryByAddress (3D hex) service	146

11	Stored data transmission functional unit.....	152
11.1	Overview	152
11.2	ClearDiagnosticInformation (14 hex) service.....	152
11.3	ReadDTCInformation (19 hex) service	154
12	InputOutput control functional unit.....	208
12.1	Overview	208
12.2	InputOutputControlByIdentifier (2F hex) service.....	209
13	Remote activation of routine functional unit.....	224
13.1	Overview	224
13.2	RoutineControl (31 hex) service	225
14	Upload download functional unit	231
14.1	Overview	231
14.2	RequestDownload (34 hex) service.....	231
14.3	RequestUpload (35 hex) service.....	234
14.4	TransferData (36 hex) service	237
14.5	RequestTransferExit (37 hex) service	242
Annex A (informative)	Global parameter definitions.....	250
Annex B (normative)	Diagnostic and communication management functional unit data parameter definitions	257
Annex C (normative)	Data transmission functional unit data parameter definitions	259
Annex D (normative)	Stored data transmission functional unit data parameter definitions.....	272
Annex E (normative)	Input output control functional unit data parameter definitions	289
Annex F (normative)	Remote activation of routine functional unit data parameter definitions.....	290
Annex G (informative)	Examples for addressAndLengthFormatIdentifier parameter values	291
Bibliography		293

Foreword

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

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

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

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights.

ISO 14229 was prepared by Technical Committee ISO/TC 22, *Road vehicles*, Subcommittee SC 3, *Electrical and electronic equipment*.

This second edition of ISO 14229 cancels and replaces the first edition (ISO 14229:1998), which has been technically revised.

Introduction

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

To achieve this, it is based on the Open Systems Interconnection (OSI) Basic Reference Model in accordance with ISO 7498-1 and ISO/IEC 10731, 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 broken into:

- unified diagnostic services (layer 7); and
- communication services (layers 1 to 6).

NOTE The diagnostic services in ISO 14229 are implemented in various applications, e.g. ISO 16844 (all parts), ISO 11992 (all parts), ISO 9141 (all parts), ISO 14230 (all parts), etc. Future modifications to this International Standard will provide long-term backward compatibility with the implementation standards as described above.

Table 1 — Example of diagnostic/programming specifications applicable to the OSI layers

Applicability	OSI layer	Enhanced diagnostics services (non-emissions-related)	
Seven layers according to ISO/IEC 7498-1 and ISO/IEC 10731	Application (layer 7)	ISO 14229/ISO 15765-3/ISO 11992-4	ISO 14229/further standards
	Presentation (layer 6)	—	—
	Session (layer 5)	ISO 15765-3/ISO 11992-4	further standards
	Transport (layer 4)	ISO 15765-2/ISO 11992-4	further standards
	Network (layer 3)	ISO 15765-2/ISO 11992-4	further standards
	Data link (layer 2)	ISO 11898/ISO 11992-1/SAE J1939-15	further standards
	Physical (layer 1)	ISO 11898/ISO 11992-1/SAE J1939-15	further standards

Figure 1 shows an example of the possible future implementation of ISO 14229 onto various data links.

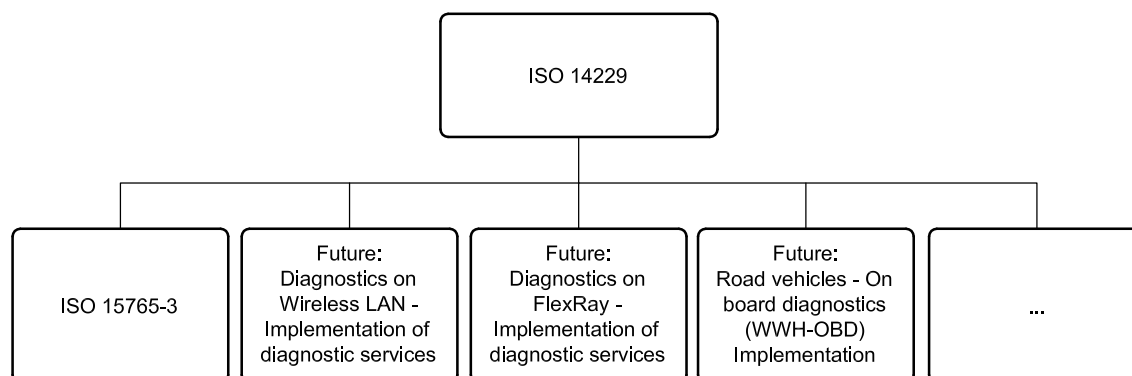


Figure 1 — Available International Standards and possible future implementations of ISO 14229

Road vehicles — Unified diagnostic services (UDS) — Specification and requirements

1 Scope

ISO 14229 specifies data link independent requirements of diagnostic services, which allow a diagnostic tester (client) to control diagnostic functions in an on-vehicle Electronic Control Unit (server) such as an electronic fuel injection, automatic gear box, anti-lock braking system, etc. connected on a serial data link embedded in a road vehicle. It specifies generic services which allow the diagnostic tester (client) to stop or to resume non-diagnostic message transmission on the data link. ISO 14229 does not apply to non-diagnostic message transmission or to use of the communication data link between two Electronic Control Units. It does not specify any implementation requirements.

The vehicle diagnostic architecture of ISO 14229 applies to:

- a single tester (client) that may be temporarily or permanently connected to the on-vehicle diagnostic data link; and
- several on-vehicle Electronic Control Units (servers) connected directly or indirectly.

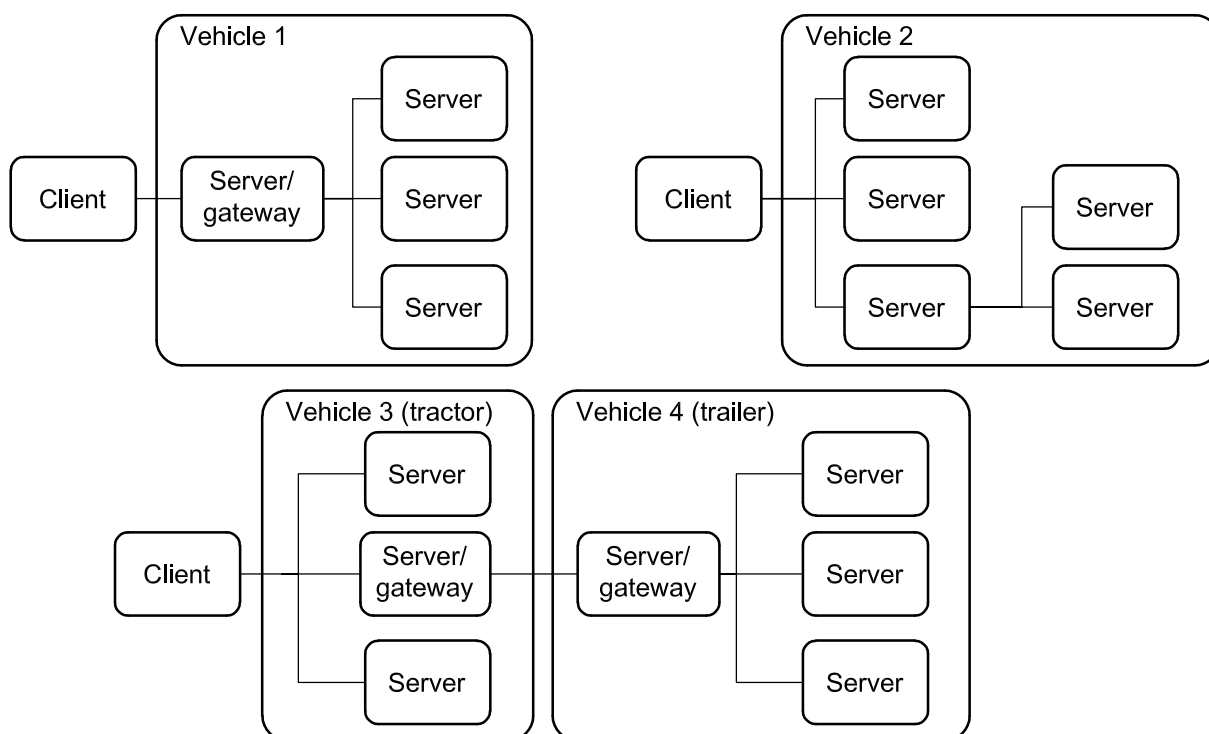


Figure 2 — Vehicle diagnostic architecture

In Figure 2:

- For vehicle 1, the servers are connected over an internal data link and indirectly connected to the diagnostic data link through a gateway. ISO 14229 applies to the diagnostic communications over the diagnostic data link; the diagnostic communications over the internal data link may conform to ISO 14229 or to another protocol.
- For vehicle 2, the servers are directly connected to the diagnostic data link.
- For vehicle 3, the servers are directly connected to the diagnostic data link through a gateway (same as vehicle 2) and vehicle 4 connects its server/gateway directly to the vehicle 3 server/gateway.

2 Normative references

The following referenced documents are indispensable for the application 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 7498-1, *Information technology — Open Systems Interconnection — Basic Reference Model: The Basic Model*

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

ISO 11898 (all parts), *Road vehicles — Controller area network (CAN)*

ISO 11992-1, *Road vehicles — Interchange of digital information on electrical connections between towing and towed vehicles — Part 1: Physical and data-link layers*

ISO 11992-4, *Road vehicles — Interchange of digital information on electrical connections between towing and towed vehicles — Part 4: Diagnostics*

ISO 14230 (all parts), *Road vehicles — Diagnostic systems — Keyword Protocol 2000*

ISO 15765-2, *Road vehicles — Diagnostics on Controller Area Networks (CAN) — Part 2: Network layer services*

ISO 15765-3, *Road vehicles — Diagnostics on Controller Area Networks (CAN) — Part 3: Implementation of unified diagnostic services (UDS on CAN)*

ISO/TR 15031-2, *Road vehicles — Communication between vehicle and external equipment for emissions-related diagnostics — Part 2: Terms, definitions, abbreviations and acronyms*

ISO 15031-5, *Road vehicles — Communication between vehicle and external equipment for emissions-related diagnostics — Part 5: Emissions-related diagnostic services*

ISO 15031-6, *Road vehicles — Communication between vehicle and external equipment for emissions-related diagnostics — Part 6: Diagnostic trouble code definitions*

ISO 15031-7, *Road vehicles — Communication between vehicle and external equipment for emissions-related diagnostics — Part 7: Data link security*

ISO 15764, *Road vehicles — Extended data link security*

3 Terms and definitions

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

3.1

integer type

simple type with distinguished values which are the positive and the negative whole numbers

NOTE The range of integer type is not specified within this document.

3.2

diagnostic trouble code

numerical common identifier for a fault condition identified by the on-board diagnostic system

3.3

diagnostic service

information exchange initiated by a client in order to require diagnostic information from a server and/or to modify its behaviour for diagnostic purposes

3.4

client

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

NOTE A tester normally makes use of other functions such as database management, specific interpretation, human-machine interface.

3.5

server

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

NOTE ISO 14229 differentiates between the server (i.e. the function) and the electronic control unit so that this International Standard remains independent from the implementation.

3.6

tester

system that controls functions such as test, inspection, monitoring or diagnosis of an on-vehicle electronic control unit and which may be dedicated to a specific type of operator (e.g. a scan tool dedicated to garage mechanics or a test tool dedicated to assembly plant agents)

NOTE The tester is also referenced as the client.

3.7

diagnostic data

data that is located in the memory of an electronic control unit which may be inspected and/or possibly modified by the tester (diagnostic data includes analogue inputs and outputs, digital inputs and outputs, intermediate values and various status information)

EXAMPLES Examples of diagnostic data include vehicle speed, throttle angle, mirror position, system status, etc. Three types of values are defined for diagnostic data:

- the current value: the value currently used by (or resulting from) the normal operation of the electronic control unit;
- a stored value: an internal copy of the current value made at specific moments, e.g. when a malfunction occurs or periodically (this copy is made under the control of the electronic control unit);
- a static value: e.g. VIN; the server is not obliged to keep internal copies of its data for diagnostic purposes, in which case the tester may only request the current value.

3.8 diagnostic session

current mode of the server, which affects the level of diagnostic functionality

NOTE Defining a repair shop or development testing session selects different server functionality (e.g. access to all memory locations may only be allowed in the development testing session).

3.9 diagnostic routine

routine that is embedded in an electronic control unit and that may be started by a server upon a request from the client

NOTE It could either run instead of a normal operating program or run concurrently to the normal operating program. In the first case, normal operation of the ECU is not possible. In the second case, multiple diagnostic routines may be enabled that run while all other parts of the electronic control unit are functioning normally.

3.10 record

one or more diagnostic data elements that are referred to together by a single means of identification

NOTE A snapshot including various input/output data and trouble codes is an example of a record.

3.11 security

as used in ISO 14229, security access method that satisfies the requirements for tamper protection as specified in ISO 15031-7

3.12 functional unit

set of functionally close or complementary diagnostic services

3.13 local server

server that is connected to the same local network as the client and is part of the same address space as the client

3.14 local client

client that is connected to the same local network as the server and is part of the same address space as the server

3.15 remote server

server that is not directly connected to the main diagnostic network

NOTE 1 A remote server is identified by means of a remote network address. Remote network addresses represent an own network address space that is independent from the addresses on the main network.

NOTE 2 A remote server is reached via a local server on the main network. Each local server on the main network can act as a gate to one independent set of remote servers. A pair of addresses will therefore always identify a remote server: a local address that identifies the gate to the remote network and a remote address identifying the remote server itself.

3.16 remote client

client that is not directly connected to the main diagnostic network

NOTE A remote client is identified by means of a remote network address. Remote network addresses represent an own address space that is independent from the addresses on the main network.

3.17**permanent DTC**

stored in NVRAM and not erasable by any test equipment command or by disconnecting power to the on-board computer

4 Symbols and abbreviated terms

A_PCI Application layer Protocol Control Information

A_PDU Application layer Protocol Data Unit

A_SDU Application layer Service Data Unit

ECU Electronic Control Unit

NOTE An ECU contains at least one server. Systems considered as Electronic Control Units include anti-lock braking system (ABS), engine management system, etc.

NR_SI Negative Response Service Identifier

OBD On-Board Diagnostic

OSI Open Systems Interconnection

RA Remote Address

SA Source Address

SI Service Identifier

TA Target Address

TA_type Target Address type

5 Conventions

ISO 14229 is guided by the conventions discussed in the OSI Service Conventions (ISO 10731) as they apply to diagnostic services. These conventions specify the interactions between the service user and the service provider. Information is passed between the service user and the service provider by service primitives, which may convey parameters.

The distinction between service and protocol is summarized in Figure 3.

ISO 14229 defines both, confirmed and unconfirmed services.

- **Confirmed services** use the six (6) service primitives, request, req_confirm, indication, response, rsp_confirm and confirmation.
- **Unconfirmed services** use only the request, req_confirm and indication service primitives.

For all services defined in ISO 14229, the request and indication service primitives always have the same format and parameters. Consequently, for all services the response and confirmation service primitives (except req_confirm and rsp_confirm) always have the same format and parameters. When the service primitives are defined in this International Standard, only the request and response service primitives are listed.

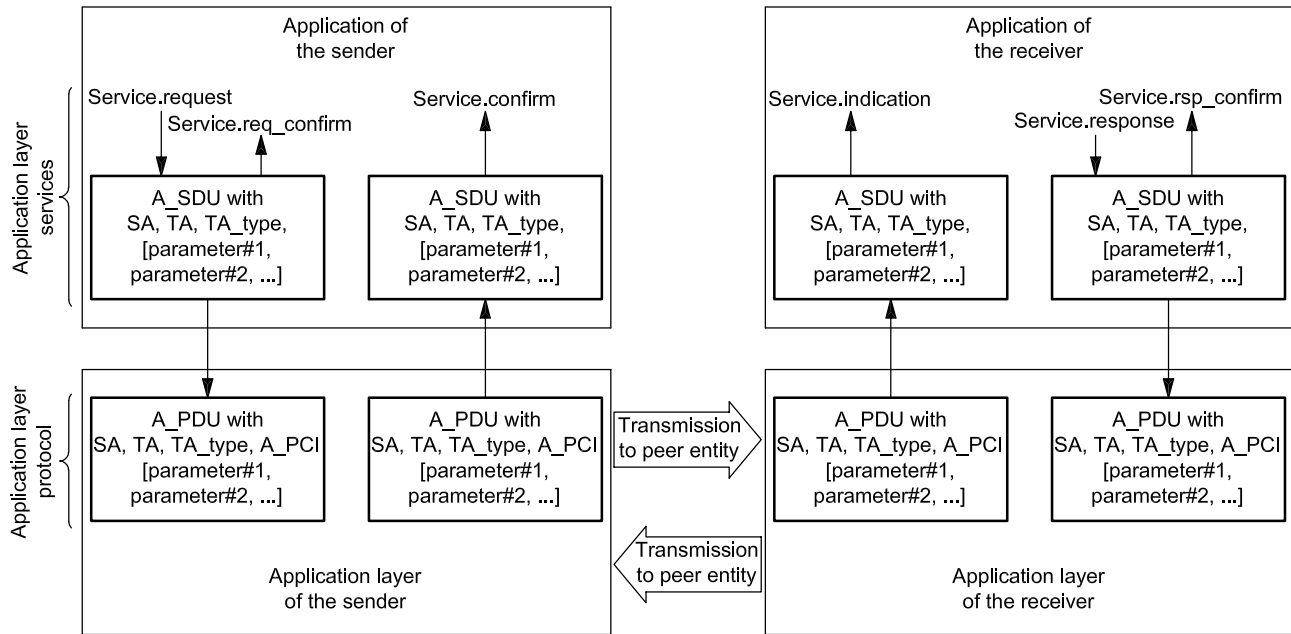


Figure 3 — The services and the protocol

6 Application layer services

6.1 General

Application layer services are usually referred to as diagnostic services. The application layer services are used in client-server-based systems to perform functions such as test, inspection, monitoring or diagnosis of on-board vehicle servers. The client, usually referred to as an External Test Equipment, uses the application layer services to request diagnostic functions to be performed in one or more servers. The server, usually a function that is part of an ECU, uses the application layer services to send response data, provided by the requested diagnostic service, back to the client. The client is usually an off-board tester but can, in some systems, also be an on-board tester. The usage of application layer services is independent from the client being an off-board or on-board tester. It is possible to have more than one client in the same vehicle system.

The service access point of the diagnostics application layer provides a number of services that all have the same general structure. For each service, six (6) service primitives are specified:

- a **service request primitive**, used by the client function in the diagnostic tester application to pass data about a requested diagnostic service to the diagnostics application layer;
- a **service request-confirmation primitive**, used by the client function in the diagnostic tester application to indicate that the data passed in the service request primitive is completely transferred to the server;
- a **service indication primitive**, used by the diagnostics application layer to pass data to the server function of the ECU diagnostic application;
- a **service response primitive**, used by the server function in the ECU diagnostic application to pass response data provided by the requested diagnostic service to the diagnostics application layer;

- a **service response-confirmation primitive**, used by the server function in the ECU diagnostic application to indicate that the data passed in the service response primitive is completely transferred to the client;
- a **service confirmation primitive**, used by the diagnostics application layer to pass data to the client function in the diagnostic tester application.

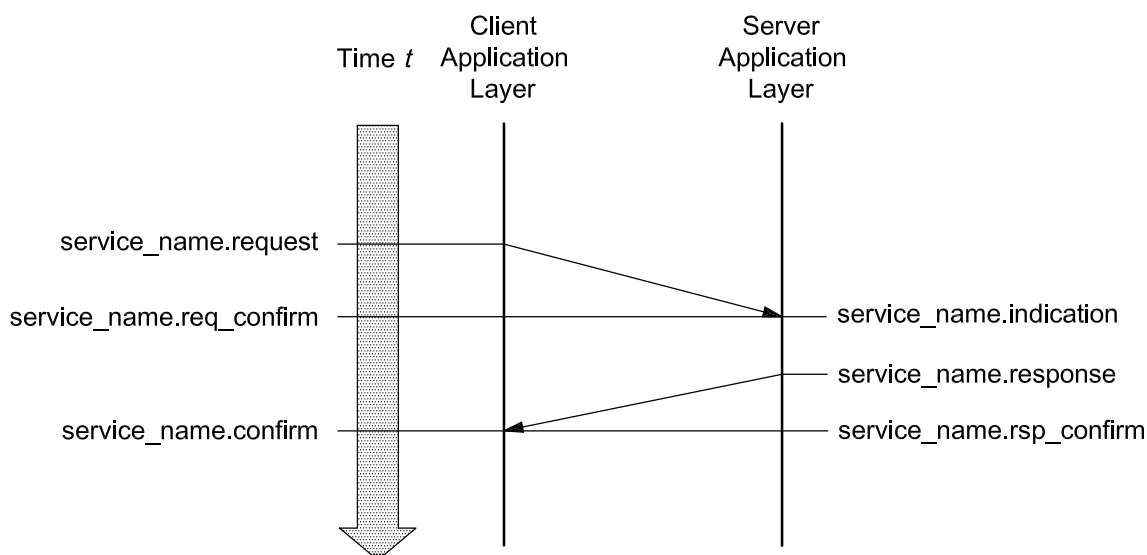


Figure 4 — Application layer service primitives — confirmed service

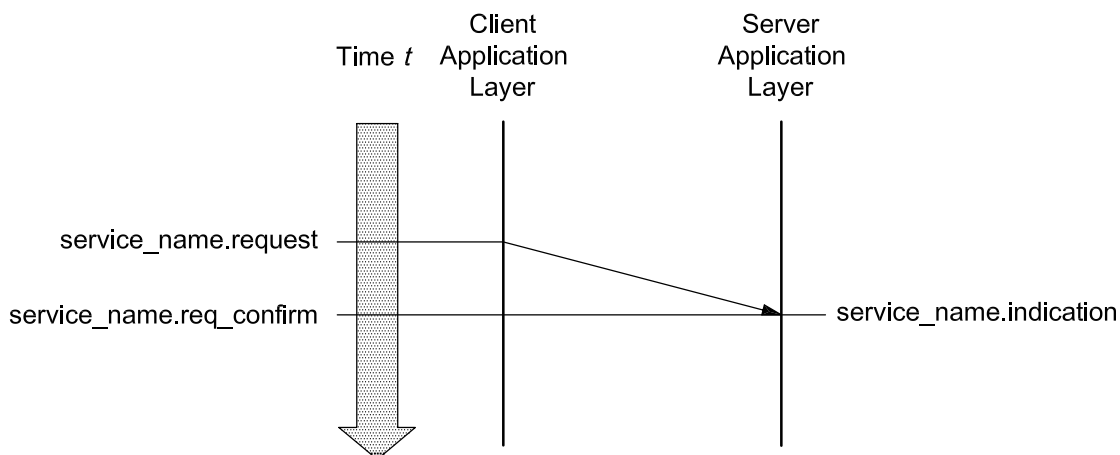


Figure 5 — Application layer service primitives — unconfirmed service

For a given service, the request primitive and the indication primitive always have the same service data unit. ISO 14229 will only list and specify the parameters of the service data unit belonging to each service request primitive. The user shall assume exactly the same parameters for each corresponding service indication primitive.

For a given service, the response primitive and the confirmation primitive always have the same service data unit. ISO 14229 only lists and specifies the parameters of the service data unit belonging to each service response primitive. The user shall assume exactly the same parameters for each corresponding service confirmation primitive.

For each service response primitive (and corresponding service confirmation primitive), two different service data units (two sets of parameters) will be specified. One set of parameters shall be used in a positive service response primitive if the requested diagnostic service can be successfully performed by the server function in the ECU diagnostic application. The other set of parameters (the negative response service data unit) shall be used if the requested diagnostic service fails or cannot be completed in time by the server function in the ECU diagnostic application.

For a given service, the request-confirmation primitive and the response-confirmation primitive always have the same service data unit. The purpose of these service primitives is to indicate the completion of an earlier request or response service primitive invocation. The service descriptions in ISO 14229 do not make use of those service primitives, but the data link specific implementation documents might use them to define e.g. service execution reference points (e.g. the ECUReset service would reset the ECU after the response has been completely transmitted to the client, which is indicated in the server by the service response-confirm primitive).

6.2 Format description of application layer services

Application layer services can have two different formats depending on how the vehicle diagnostic system is configured.

If the vehicle system is configured as a single (one logical) diagnostic network where all clients and servers are connected directly, then the default (also called normal or standard) format of application layer services shall be used. This format is compatible with the diagnostic system formats used on data links such as K- and L-lines. The default application layer services format is specified in 6.3.

The remote format of application layer services shall be used in vehicle systems implementing the concept of local servers and remote servers. The remote format has one additional address parameter called remote address. The remote format is used to access servers that are not directly connected to the main diagnostic network in the vehicle. The remote format for application layer services is specified in 6.4.

6.3 Format description of standard service primitives

6.3.1 General definition

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

```
service_name.type (
    parameter A, parameter B, parameter C
    [,parameter 1, ...]
)
```

where:

- “service_name” is the name of the diagnostic service (e.g. DiagnosticSessionControl);
- “type” indicates the type of the service primitive (e.g. request);
- “parameter A, ...” is the A_SDU as a list of values passed by the service primitive (addressing information);
- “parameter A, parameter B, parameter C” are mandatory parameters that shall be included in all service calls;
- “[,parameter 1, ...]” are parameters that depend on the specific service (e.g. parameter 1 can be the diagnosticSession for the DiagnosticSessionControl service). The brackets indicate that this part of the parameter list may be empty.

6.3.2 Service request and service indication primitives

For each application layer service, service request and service indication primitives are specified according to the following general format:

```
service_name.request (
    SA,
    TA,
    TA_type
    [,parameter 1, ...]
)
```

The request primitive is used by the client function in the diagnostic tester application to initiate the service and pass data about the requested diagnostic service to the application layer.

```
service_name.indication (
    SA,
    TA,
    TA_type
    [,parameter 1, ...]
)
```

The indication primitive is used by the application layer to indicate an internal event which is significant to the ECU diagnostic application and to pass data about the requested diagnostic service to the server function of the ECU diagnostic application.

The request and indication primitives of a specific application layer service always have the same parameters and parameter values. This means that the values of individual parameters shall not be changed by the communicating peer protocol entities of the application layer when the data is transmitted from the client to the server. The same values that are passed by the client function in the client application to the application layer in the service request call shall be received by the server function of the diagnostic application from the service indication of the peer application layer.

6.3.3 Service response and service confirm primitives

For each confirmed application layer service, service response and service confirm primitives are specified according to the following general format:

```
service_name.response (
    SA,
    TA,
    TA_type,
    Result
    [,parameter 1, ...]
)
```

The response primitive is used by the server function in the ECU diagnostic application, to initiate the service and pass response data provided by the requested diagnostic service to the application layer.

```
service_name.confirm (
    SA,
    TA,
    TA_type,
    Result
    [,parameter 1, ...]
)
```

The confirm primitive is used by the application layer to indicate an internal event which is significant to the client application and to pass results of an associated previous service request to the client function in the diagnostic tester application. It does not necessarily indicate any activity at the remote peer interface, e.g. if the requested service is not supported by the server or if the communication is broken.

The response and confirm primitives of a specific application layer service always have the same parameters and parameter values. This means that the values of individual parameters shall not be changed by the communicating peer protocol entities of the application layer when the data is transmitted from the server to the client. The same values that are passed by the server function of the ECU diagnostic application to the application layer in the service response call shall be received by the client function in the diagnostic tester application from the service confirmation of the peer application layer.

For each response and confirm primitive two different service data units (two sets of parameters) will be specified.

- A positive response and positive confirm primitive shall be used with the first service data unit if the requested diagnostic service could be successfully performed by the server function in the ECU.
- A negative response and confirm primitive shall be used with the second service data unit if the requested diagnostic service failed or could not be completed in time by the server function in the ECU.

6.3.4 Service request-confirm and service response-confirm primitives

For each application layer service, service request-confirm and service response-confirm primitives are specified according to the following general format:

```
service_name.req_confirm(  
    SA,  
    TA,  
    TA_type,  
    Result  
)
```

The request-confirm primitive is used by the application layer to indicate an internal event, which is significant to the client application, and pass results of an associated previous service request to the client function in the diagnostic tester application.

```
service_name.rsp_confirm(  
    SA,  
    TA,  
    TA_type,  
    Result  
)
```

The response-confirm primitive is used by the application layer to indicate an internal event, which is significant to the server application, and pass results of an associated previous service response to the server function in the ECU application.

6.4 Format description of remote service primitives

6.4.1 General definition

Diagnostic communication between a local client and a remote server can take place if the remote format of application layer services is used. All definitions made for the default format of application layer services shall be applicable also for the remote format of application layer services with the addition of one more addressing parameter.

Diagnostic communication can take place between a local client on the main network and one or more remote servers on a remote network. Communication can also take place between a remote client on a remote network and one or more local servers on the main network.

Diagnostic communication cannot take place between any combination of clients and servers on two different remote networks.

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

```
service_name.type (
    parameter A, parameter B, parameter C,
    parameter D
    [,parameter 1, ...]
)
```

where:

- “service_name” is the name of the diagnostic service (e.g. DiagnosticSessionControl);
- “type” indicates the type of the service primitive (e.g. request);
- “parameter A, ...” is the A_SDU as a list of values passed by the service primitive (addressing information);
- “parameter A, parameter B, parameter C” are mandatory parameters that shall be included in all service calls;
- “parameter D” is an additional parameter that is only used in vehicles implementing the concept of remote servers (remote address);
- “[,parameter 1, ...]” are parameters that depend on the specific service (e.g. parameter 1 can be the diagnosticSession for the DiagnosticSessionControl service). The brackets indicate that this part of the parameter list may be empty.

6.4.2 Remote service request and service indication primitives

For each remote format application layer service, service request and service indication primitives are specified according to the following general format:

```
service_name.request (
    SA,
    TA,
    TA_type
    [,RA]
    [,parameter 1, ...]
)
```

The request primitive is used by the local client function in the client application, to initiate the service and pass data about the requested diagnostic service to the application layer.

```
service_name.indication (
    SA,
    TA,
    TA_type
    [,RA]
    [,parameter 1, ...]
)
```

The indication primitive is used by the remote application layer to indicate an internal event which is significant to the ECU diagnostic application and to pass data about the requested diagnostic service to the remote server function of the ECU diagnostic application.

The request and indication primitive of a specific application layer service always have the same parameters and parameter values. This means that the values of individual parameters shall not be changed by the communicating peer protocol entities of the application layer when the data is transmitted from the client to the server. The same values that are passed by the client function in the diagnostic tester application to the application layer in the service request call shall be received by the server function of the ECU application from the service indication of the peer application layer.

NOTE For clarity, the text assumes communication between a local client and one or more remote server. The protocol also supports communication between a remote client and one or more local servers using the same remote format application layer services.

6.4.3 Remote service response and service confirm primitives

For each remote format application layer service, service response and service confirm primitives are specified according to the following general format:

```
service_name.response (  
    SA,  
    TA,  
    TA_type,  
    [RA,]  
    Result  
    [,parameter 1, ...]  
)
```

The response primitive is used by the remote server function in the ECU diagnostic application, to initiate the service and pass response data provided by the requested diagnostic service to the application layer.

```
service_name.confirm (  
    SA,  
    TA,  
    TA_type,  
    [RA,]  
    Result  
    [,parameter 1, ...]  
)
```

The confirm primitive is used by the local application layer to indicate an internal event which is significant to the client application and to pass results of an associated previous service request to the client function in the ECU application. It does not necessarily indicate any activity at the remote peer interface, e.g. if the requested service is not supported by the server or if the communication is broken.

The response and confirm primitive of a specific application layer service always has the same parameters and parameter values. This means that the values of individual parameters shall not be changed by the communicating peer protocol entities of the application layer when the data is transmitted from the server to the client. The same values that are passed by the server function of the ECU diagnostic application to the application layer in the service response call shall be received by the client function in the diagnostic tester application from the service confirmation of the peer application layer.

For each response and confirm primitive, two different service data units (two sets of parameters) will be specified.

- A positive response and positive confirm primitive shall be used with the first service data unit if the requested diagnostic service could be successfully performed by the server function in the ECU.

- A negative response and confirm primitive shall be used with the second service data unit if the requested diagnostic service failed or could not be completed in time by the server function in the ECU.

NOTE For clarity, the text assumes communication between a local client and one or more remote server. The protocol also supports communication between a remote client and one or more local servers using the same remote format application layer services.

6.4.4 Remote service request-confirm and service response-confirm primitives

For each application layer service, service request-confirm and service response-confirm primitives are specified according to the following general format:

```
service_name.req_confirm(
    SA,
    TA,
    TA_type,
    [RA,]
    Result
)
```

The request-confirm primitive is used by the client application layer to indicate an internal event which is significant to the client application and to pass results of an associated previous service request to the client function in the ECU application.

```
service_name.rsp_confirm(
    SA,
    TA,
    [RA,]
    TA_type,
    Result,
)
```

The response-confirm primitive is used by the server application layer to indicate an internal event which is significant to the server application and to pass results of an associated previous service response to the server function in the ECU application.

6.5 Service data unit specification

6.5.1 Mandatory parameters

6.5.1.1 General definition

The application layer services contain three (3) mandatory parameters. The following parameter definitions are applicable to all application layer services specified in this International Standard (standard and remote format).

6.5.1.2 Source address (SA)

Type: 1 byte unsigned integer value

Range: 00-FF hex

Description:

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

For service requests (and service indications), SA represents the client identifier for the client function that has requested the diagnostic service. The client shall always be located in one diagnostic tester only. There shall be a strict, one-to-one relation between client identifiers and source addresses. Each client identifier shall be encoded with one SA value. If more than one client is implemented in the same diagnostic tester, then each client shall have its own client identifier and corresponding SA value.

For service responses (and service confirmations), SA represents the physical location of the server that has performed the requested diagnostic service. A server may be implemented in one ECU only or be distributed and implemented in several ECUs. If a server is implemented in one ECU only, then it shall be encoded with one SA value only. If a server is distributed and implemented in several ECUs, then the server identifier shall be encoded with one SA value for each physical location of the server.

If a remote client or server is the original source for a message, then SA represents the local server that is the gate from the remote network to the main network.

NOTE The SA value in a response message will be the same as the TA value in the corresponding request message if physical addressing was used for the request message.

6.5.1.3 Target address (TA)

Type: 1 byte unsigned integer value

Range: 00-FF hex

Description:

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

Two different addressing methods, called physical addressing and functional addressing, are specified for diagnostics. Therefore, two independent sets of target addresses can be defined for a vehicle system (one for each addressing method).

Physical addressing shall always be a dedicated message to a server implemented in one ECU. When physical addressing is used, the communication is a point-to-point communication between the client and the server.

Functional addressing is used by the client if it does not know the physical address of the server that will respond to a service request or if the server is implemented as a distributed server in several ECUs. When functional addressing is used, the communication is a broadcast communication from the client to a server implemented in one or more ECUs.

For service requests (and service indications), TA represents the server identifier for the server that will perform the requested diagnostic service. If a remote server is being addressed, then TA represents the local server that is the gate from the main network to the remote network.

For service responses (and service confirmations), TA represents the client identifier for the client that originally requested the diagnostic service and will receive the requested data. Service responses (and service confirmations) shall always use physical addressing. If a remote client is being addressed, then TA represents the local server that is the gate from the main network to the remote network.

NOTE The TA value of a response message will always be the same as the SA value of the corresponding request message.

6.5.1.4 TA_Type, Target Address type

Type: enumeration

Range: physical, functional

Description:

The parameter TA_type is an extension to the TA parameter. It is used to represent the addressing method chosen for a message transmission.

6.5.1.5 Result

Type: enumeration

Range: positive, negative

Description:

The parameter "Result" is used by the response and confirm primitives to indicate if a message is a positive response/positive confirm message or a negative response/negative confirm message. The service-specific parameters in the message are different depending on the value of the Result parameter.

6.5.2 Vehicle system requirements

The vehicle manufacturer shall ensure that each server in the system has a unique server identifier. The vehicle manufacturer shall also ensure that each client in the system has a unique client identifier.

All client and server identifiers for the main diagnostic network in a vehicle system shall be encoded into the same range of source addresses. This means that a client and a server shall not be represented by the same SA value in a given vehicle system.

The physical target address for a server shall always be the same as the source address for the server.

Remote server identifiers can be assigned independently from client and server identifiers on the main network.

In general only the server(s) addressed shall respond to the client request message.

6.5.3 Optional parameters

6.5.3.1 Remote address (RA)

Type: 1 byte unsigned integer value

Range: 00-FF hex

Description:

RA is used to extend the available address range to encode client and server identifiers. RA shall only be used in vehicles that implement the concept of local servers and remote servers. Remote addresses represent their own address range and are independent from the addresses on the main network.

The parameter RA shall be used to encode remote client and server identifiers. RA can represent either a remote target address or a remote source address, depending on the direction of the message carrying the RA.

For service requests (and service indications) sent by a client on the main network, RA represents the remote server identifier (remote target address) for the server that will perform the requested diagnostic service.

RA can be used both as a physical and a functional address. For each value of RA, the system builder shall specify if that value represents a physical or functional address.

NOTE There is no special parameter that represents physical or functional remote addresses in the way TA_type specifies the addressing method for TA. Physical and functional remote addresses share the same 1 byte range of values and the meaning of each value shall be defined by the system builder.

For service responses (and service confirmations) sent by a remote server, RA represents the physical location (remote source address) of the remote server that has performed the requested diagnostic service.

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

For service requests (and service indications) sent by a remote client, RA represents the remote server identifier (remote source address) for the client function that has requested the diagnostic service.

For service responses (and service confirmations) sent by a local server, RA represents the remote client identifier (remote target address) for the client that originally requested the diagnostic service and shall receive the requested data.

6.5.3.2 Remote server example with remote network

In some systems, the remote server is connected to a remote network separated from the main diagnostic network by a gateway. The following is an example showing how the parameters SA, TA and RA shall be used for proper communication between a local client on the main network and a remote server via a gateway. In the example, it is assumed that the same type of addressing is used on the remote network as on the main network.

The external test equipment is connected to the main network and has client identifier 241 (F1 hex). The gateway is connected to both the main network and the remote network. On the main network the gateway has client identifier 200 (C8 hex). On the remote network, the gateway has client identifier 10 (0A hex). The remote server is connected to the remote network and has client identifier 62 (3E hex). The configuration is described in Figure 6.



Figure 6 — Remote server system example 1

The external test equipment sends a remote diagnostic request message with

- SA = 241 (F1 hex),
- TA = 200 (C8 hex), and
- RA = 62 (3E hex).

The gateway receives the message and sends it out on the remote network with

- SA = 10 (0A hex),
- TA = 62 (3E hex), and
- RA = 241 (F1 hex).

The remote server receives the message.

The remote server sends back a remote diagnostic response message with

- SA = 62 (3E hex),
- TA = 10 (0A hex), and
- RA = 241 (F1 hex).

The gateway receives the message and sends it out on the main network with

- SA = 200 (C8 hex),
- TA = 241 (F1 hex), and
- RA = 62 (3E hex).

The external test equipment receives the message.

6.5.3.3 Remote server example without remote network

In some systems, the remote server is a functional part of a server belonging to the main network. The server has been given a remote server identifier in order to extend the available address range to encode client and server identifiers. In such systems the remote server is logically separated from the main network even if the ECU, of which the remote server is a part, is connected to the main diagnostic network. To get a working system, the server must also have a gateway function that is part of the main diagnostic network and can serve as a gate to the remote server. The following is an example showing how the parameters SA, TA and RA are used for proper communication between a local client on the main network and a remote server via a gateway.

The external test equipment is connected to the main network and has client identifier 241 (F1 hex). The gateway is connected to the same main network. The gateway has client identifier 200 (C8 hex). The remote server has client identifier 62 (3E hex). The configuration is described in Figure 7.

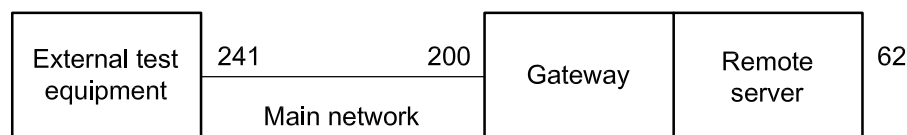


Figure 7 — Remote server system example 2

The external test equipment sends a remote diagnostic request message with

- SA = 241 (F1 hex),
- TA = 200 (C8 hex), and
- RA = 62 (3E hex).

The gateway receives the message and passes it over to the remote server function. The remote server receives the message.

The remote server sends back a remote diagnostic response message by passing it to the gateway function. The gateway receives the message and sends it out on the main network with

- SA = 200 (C8 hex),
- TA = 241 (F1 hex), and
- RA = 62 (3E hex).

The external test equipment receives the message.

6.5.3.4 Remote client example with remote network

In some systems, the client is connected to a remote network separated from the main diagnostic network by a gateway. The following is an example showing how the parameters SA, TA and RA are used for proper communication between a remote client on a remote network and a local server on the main network via a gateway. In the example, it is assumed that the same type of addressing is used on the remote network as on the main network.

The external test equipment is connected to the remote network and has client identifier 242 (F2 hex). The gateway is connected to both the main network and the remote network. On the main network, the gateway has client identifier 200 (C8 hex). On the remote network, the gateway has client identifier 10 (0A hex). The local server is connected to the main network and has client identifier 18 (12 hex). The configuration is described in Figure 8.

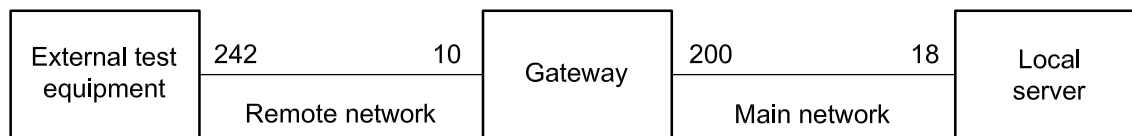


Figure 8 — Remote client example

The external test equipment sends a remote diagnostic request message with

- SA = 242 (F1 hex),
- TA = 10 (0A hex), and
- RA = 18 (12 hex).

The gateway receives the message and sends it out on the main network with

- SA = 200 dec,
- TA = 18 dec, and
- RA = 242 dec.

The local server receives the message.

The local server sends back a remote diagnostic response message with

- SA = 18 (12 hex),
- TA = 200 (C8 hex), and
- RA = 242 (F1 hex).

The gateway receives the message and sends it out on the remote network with

- SA = 10 (0A hex),
- TA = 242 (F1 hex), and
- RA = 18 (12 hex).

The external test equipment receives the message.

7 Application layer protocol

7.1 General definition

The application layer protocol shall always be a confirmed message transmission, meaning that for each service request sent from the client, there shall be one or more corresponding responses sent from the server.

The only exception to this rule shall be a few cases when e.g. functional addressing is used or the request/indication specifies that no response/confirmation shall be generated. In order not to burden the system with many unnecessary messages, there are a few cases when negative response messages shall not be sent even if the server failed to complete the requested diagnostic service.

The application layer protocol shall be handled in parallel with the session layer protocol. This means that, even if the client is waiting for a response to a previous request, it shall maintain proper session layer timing (e.g. sending a TesterPresent request if that is needed to keep a diagnostic session going in other servers; the implementation depends on the data link layer used).

7.2 Protocol data unit specification

The A_PDU is directly constructed from the A_SDU and the layer-specific control information A_PCI (Application layer Protocol Control Information). The A_PDU shall have the following general format:

```
A_PDU (
    SA,
    TA,
    TA_type,
    [RA,]
    A_Data = A_PCI + [parameter 1, ...]
)
```

where:

- “SA, TA, TA_type, RA” are the same parameters as used in the A_SDU;
- “A_Data” is a string of byte data defined for each individual application layer service. The A_Data string shall start with the A_PCI followed by all service-specific parameters from the A_SDU as specified for each service. The brackets indicate that this part of the parameter list may be empty.

7.3 Application protocol control information

The A_PCI shall have two alternative formats depending on which type of service primitive that has been called and the value of the Result parameter. For all service requests and for service responses/service confirmations with Result = positive, the following definition shall apply:

```
A_PCI (
    SI
)
```

where “SI” is the parameter service identifier.

For service responses/service confirmations with Result = negative, the following definition shall apply:

A_PCI (
 NR_SI,
 SI
)

where:

- “NR_SI” is the special parameter identifying negative service responses/confirmations;
- “SI” is the parameter service identifier.

NOTE For the transmission of periodic messages utilizing response message type #2 as defined in the service ReadDataByPeriodicIdentifier (2A hex, see 10.5) no A_PCI is present in the application layer protocol data unit (A_PDU).

7.3.1 Service identifier (SI)

Type: 1 byte unsigned integer value

Range: 00-FF hex according to definitions in Table 2

Table 2 — Service identifier (SI) values

Service identifier (hex value)	Service type (bit 6)	Where defined
00 – 0F	OBD service requests	ISO 15031-5
10 – 3E	ISO 14229 service requests	ISO 14229
3F	Not applicable	Reserved by document
40 – 4F	OBD service responses	ISO 15031-5
50 – 7E	ISO 14229 positive service responses	ISO 14229
7F	Negative response service identifier	ISO 14229
80	Not applicable	Reserved by ISO 14229
81 – 82	Not applicable	Reserved by ISO 14230
83 – 88	ISO 14229 service requests	ISO 14229
89 – 9F	Service requests	Reserved for future expansion as needed
A0 – B9	Service requests	Defined by vehicle manufacturer
BA – BE	Service requests	Defined by system supplier
BF	Not applicable	Reserved by document
C0	Not applicable	Reserved by ISO 14229
C1 – C2	Not applicable	Reserved by ISO 14230
C3 – C8	ISO 14229 positive service responses	ISO 14229
C9 – DF	Positive service responses	Reserved for future expansion as needed
E0 – F9	Positive service responses	Defined by vehicle manufacturer
FA – FE	Positive service responses	Defined by system supplier
FF	Not applicable	Reserved by document

NOTE There is a one-to-one correspondence between service identifiers for request messages and service identifiers for positive response messages, with bit 6 of the SI hex value indicating the service type. All request messages have SI bit 6 = 0. All positive response messages have SI bit 6 = 1, except response message type #2 of the ReadDataByPeriodicIdentifier (2A hex, see section 10.5) service.

Description:

The SI shall be used to encode the specific service that has been called in the service primitive. Each request service shall be assigned a unique SI value. Each positive response service shall be assigned a corresponding unique SI value.

The service identifier is used to represent the service in the A_Data data string that is passed from the application layer to lower layers (and returned from lower layers).

7.3.2 Negative response service identifier (NR_SI)

Type: 1 byte unsigned integer value

Fixed value: 7F hex

Description:

The parameter NR_SI is a special parameter identifying negative service responses/confirmations. It shall be part of the A_PCI for negative response/confirm messages.

NOTE The NR_SI value is coordinated with the SI values. The NR_SI value is not used as an SI value in order to make A_Data coding and decoding easier.

7.4 Negative response/confirmation service primitive

Each diagnostic service has a negative response/negative confirmation message specified with message A_Data bytes according to Table 3. The first A_Data byte (A_PCI.NR_SI) is always the specific negative response service identifier. The second A_Data byte (A_PCI.SI) shall be a copy of the service identifier value from the service request/indication message to which the negative response message corresponds.

Table 3 — Negative response A_PDU

A_PDU parameter	Parameter name	Cvt	Hex value	Mnemonic
SA	Source Address	M ^a	xx	SA
TA	Target Address	M	xx	TA
TA_type	Target Address type	M	xx	TA_type
RA	Remote Address (optional)	C ^b	xx	RA
A_Data.A_PCI.NR_SI	Negative Response Service Id	M	7F	SIDNR
A_Data.A_PCI.SI	<Service Name> Request Service Id	M	xx	SIDRQ
A_Data.Parameter 1	responseCode	M	xx	NRC_
^a M (Mandatory): In case the negative response A_PDU is issued then those A_PDU parameters shall be present. ^b C (Conditional): The RA (Remote Address) PDU parameter is only present in case of remote addressing.				

NOTE A_Data represents the message data bytes of the negative response message.

The parameter responseCode is used in the negative response message to indicate why the diagnostic service failed or could not be completed in time. Values are defined in A.1.

7.5 Server response implementation rules

7.5.1 General definitions

The following subclauses specify the behaviour of the server when executing a service. The server and the client shall follow these implementation rules.

Legend for subclauses 7.5.2, 7.5.3 and 7.5.4

Abbreviation	Description
suppressPosRspMsgIndicationBit	TRUE = server shall NOT send a positive response message FALSE = server shall send a positive or negative response message
PosRsp	Abbreviation for positive response message
NegRsp	Abbreviation for negative response message
NoRsp	Abbreviation for NOT sending a positive or negative response message
NRC	Abbreviation for negative response code
ALL	All of the requested data parameters (service without sub-function parameter) of the client request message are supported by the server
at least 1	At least 1 data parameter (service without sub-function parameter) of the client request message must be supported by the server
NONE	None of the requested data parameters (service without sub-function parameter) of the client request message is supported by the server

The server shall support its list of diagnostic services regardless of addressing mode (physical, functional addressing type).

IMPORTANT — As required by the tables in the following subclauses, negative response messages with negative response codes of SNS (serviceNotSupported), SFNS (subFunctionNotSupported) and ROOR (requestOutOfRange) shall never be transmitted when functional addressing was used for the request message.

7.5.2 Request message with sub-function parameter and server response behaviour

7.5.2.1 Physically addressed client request message

The server response behaviour specified in this subclause is referenced in the service description of each service, which supports a sub-function parameter in the physically addressed request message received from the client.

Table 4 shows possible communication schemes with physical addressing.

Table 4 — Physically addressed request message with sub-function parameter and server response behaviour

Server case #	Client request message		Server capability			Server response		Comments on server response
	Addressing scheme	subFunction (suppress-PosRspMsg-Indication-Bit)	Service ID supported	Sub-function supported	Data parameter supported (only if applicable)	Message	Negative: NRC/section	
1	physical	FALSE (bit = 0)	YES	YES	At least 1	PosRsp	—	Server sends positive response
2					—	NegRsp	NRC=xx	Server sends negative response because error occurred reading the data parameters of the request message
3			NO	—	—		NRC=SNS	Negative response with NRC 11 hex
4			YES	NO	—		NRC=SFNS	Negative response with NRC 12 hex
5		TRUE (bit = 1)	YES	YES	At least 1	NoRsp	—	Server does NOT send a response
6					—	NegRsp	NRC=xx	Server sends negative response because error occurred reading the data parameters of the request message
7			NO	—	—		NRC=SNS	Negative response with NRC 11 hex
8			YES	NO	—		NRC=SFNS	Negative response with NRC 12 hex

The following is a description of server response cases on physically addressed client request messages with subFunction.

- 1) Server sends a positive response message because the service identifier and sub-function parameter is supported by the client's request with indication for a response message.
- 2) Server sends a negative response message (e.g. IMLOIF: incorrectMessageLengthOrIncorrectFormat) because the service identifier and sub-function parameter of the client's request is supported but some other error appeared (e.g. wrong PDU length according to service identifier and sub-function parameter in the request message) during processing of the sub-function.
- 3) Server sends a negative response message with the negative response code SNS (service not supported) because the service identifier of the client's request is not supported with indication for a response message.
- 4) Server sends a negative response message with the negative response code SFNS (sub-function not supported) because the service identifier is supported and the sub-function parameter of the client's request is not supported with indication for a response message.
- 5) Server sends no response message because the service identifier and sub-function parameter is supported by the client's request with indication for no response message. If a negative response code RCRRP (requestCorrectlyReceivedResponsePending) is used, a final response shall be given independent of the suppressPosRspMsgIndicationBit value.

- 6) Same effect as in 2) (e.g. a negative response message is sent) because the `suppressPosRspMsgIndicationBit` is ignored for any negative response that needs to be sent upon receipt of a physically addressed request message.
- 7) Same effect as in 3) (e.g. the negative response message is sent) because the `suppressPosRspMsgIndicationBit` is ignored for any negative response that needs to be sent upon receipt of a physically addressed request message.
- 8) Same effect as in 4) (e.g. the negative response message is sent) because the `suppressPosRspMsgIndicationBit` is ignored for any negative response that needs to be sent upon receipt of a physically addressed request message.

7.5.2.2 Functionally addressed client request message

The server response behaviour specified in this subclause is referenced in the service description of each service which supports a sub-function parameter in the functionally addressed request message received from the client.

Table 5 shows possible communication schemes with functional addressing.

Table 5 — Functionally addressed request message with sub-function parameter and server response behaviour

Server case #	Client request message		Server capability			Server response		Comments on server response
	Addressing scheme	subFunction (suppress-PosRspMsgIndication-Bit)	Service ID supported	Sub-function supported	Data parameter supported (only if applicable)	Message	Negative: NRC/section	
1	functional	FALSE (bit = 0)	YES	YES	At least 1	PosRsp	—	Server sends positive response
2					At least 1	NegRsp	NRC=xx	Server sends negative response because error occurred reading the data parameters of the request message
3					None	NoRsp	—	Server does NOT send a response
4			NO	—	—		—	Server does NOT send a response
5			YES	NO	—		—	Server does NOT send a response
6		TRUE (bit = 1)	YES	YES	At least 1	NoRsp	—	Server does NOT send a response
7					At least 1	NegRsp	NRC=xx	Server sends negative response because error occurred reading the data parameters of the request message
8					None	NoRsp	—	Server does NOT send a response
9			NO	—	—		—	Server does NOT send a response
10			YES	NO	—		—	Server does NOT send a response

Description of server response cases on functionally addressed client request messages with subFunction:

- 1) Server sends a positive response message because the service identifier and sub-function parameter is supported by the client's request with indication for a response message.
- 2) Server sends a negative response message (e.g. IMLOIF: incorrectMessageLengthOrIncorrectFormat) because the service identifier and sub-function parameter is supported by the client's request, but some other error appeared (e.g. wrong PDU length according to service identifier and sub-function parameter in the request message) during processing of the sub-function.
- 3) Server sends no response message because the negative response code ROOR (requestOutOfRange, which is identified by the server because the service identifier and sub-function parameter are supported but a required data parameter is not supported by the client's request) is always suppressed in case of a functionally addressed request message. The suppressPosRspMsgIndicationBit does not matter in such cases.
- 4) Server sends no response message because the negative response code SNS (serviceNotSupported, which is identified by the server because the service identifier is not supported by the client's request) is always suppressed in case of a functionally addressed request message. The suppressPosRspMsgIndicationBit does not matter in such cases.
- 5) Server sends no response message because the negative response code SFNS (subFunctionNotSupported, which is identified by the server because the service identifier is supported and the sub-function parameter is not supported by the client's request) is always suppressed in case of a functionally addressed request. The suppressPosRspMsgIndicationBit does not matter in such cases.
- 6) Server sends no response message because the service identifier and sub-function parameter is supported by the client's request with indication for no response message.

NOTE If a negative response code RCRP (requestCorrectlyReceivedResponsePending) is used, a final response shall be given independent of the suppressPosRspMsgIndicationBit value.

- 7) Same effect as in 2) (e.g. a negative response message is sent) because the suppressPosRspMsgIndicationBit is ignored for any negative response. This is also true if the request message is functionally addressed.
- 8) Same effect as in 3) (e.g. no response message is sent) because the negative response code ROOR (requestOutOfRange, which is identified by the server because the service identifier and sub-function parameter are supported but a required data parameter is not supported by the client's request) is always suppressed in case of a functionally addressed request message. The suppressPosRspMsgIndicationBit does not matter in such a case.
- 9) Same effect as in 4) (e.g. no response message is sent) because the negative response code SNS (serviceNotSupported, which is identified by the server because the service identifier is not supported by the client's request) is always suppressed in case of a functionally addressed request message. The suppressPosRspMsgIndicationBit does not matter in such a case.
- 10) Same effect as in 5) (e.g. no response message is sent) because the negative response code SFNS (subFunctionNotSupported, which is identified by the server because the service identifier is supported and the sub-function parameter is not supported by the client's request) is always suppressed in case of a functionally addressed request message. The suppressPosRspMsgIndicationBit does not matter in such a case.

7.5.3 Request message without sub-function parameter and server response behaviour

7.5.3.1 Physically addressed client request message

The server response behaviour specified in this subclause is referenced in the service description of each service which does not support a sub-function parameter but a data parameter in the physically addressed request message received from the client.

Table 6 shows possible communication schemes with physical addressing.

Table 6 — Physically addressed request message without sub-function parameter and server response behaviour

Server case #	Client request message	Server capability		Server response		Comments on server response
	Addressing scheme	Service ID supported	Parameter supported	Message	Negative: NRC/section	
1	physical	YES	ALL	PosRsp	—	Server sends positive response
2			At least 1		—	Server sends positive response
3			At least 1, more than 1, or ALL	NegRsp	NRC=xx	Server sends negative response because error occurred reading data parameters of request message
4			NONE		NRC=ROOR	Negative response with NRC 31 hex
5		NO	—		NRC=SNS	Negative response with NRC 11 hex

The following is a description of server response cases on physically addressed client request messages without sub-function (data parameter follows service identifier).

- 1) Server sends a positive response message because the service identifier and all data parameters are supported by the client's request message.
- 2) Server sends a positive response message because the service identifier and a single data parameter is supported by the client's request message.
- 3) Server sends a negative response message (e.g. IMLOIF: incorrectMessageLengthOrIncorrectFormat) because the service identifier is supported and at least one, more than one or all data parameters are supported by the client's request message, but some other error occurred (e.g. wrong length of the request message) during processing of the service.
- 4) Server sends a negative response message with the negative response code ROOR (requestOutOfRange) because the service identifier is supported but none of the requested data parameters are supported by the client's request message.
- 5) Server sends a negative response message with the negative response code SNS (serviceNotSupported) because the service identifier is not supported by the client's request message.

7.5.3.2 Functionally addressed client request message

The server response behaviour specified in this subclause is referenced in the service description of each service which does not support a sub-function parameter but a data parameter in the functionally addressed request message received from the client.

Table 7 shows possible communication schemes with functional addressing.

Table 7 — Functionally addressed request message without sub-function parameter and server response behaviour

Server case #	Client request message	Server capability		Server response		Comments on server response
	Addressing scheme	Service ID supported	Parameter supported	Message	Negative: NRC/section	
1	functional	YES	YES	PosRsp	—	Server sends positive response
2			at least 1		—	Server sends positive response
3			At least 1, more than 1, or ALL	NegRsp	NRC=xx	Server sends negative response because error occurred reading data parameters of request message
4			NONE	NoRsp	—	Server does NOT send a response
5		NO	—		—	Server does NOT send a response

The following is a description of server response cases on functionally addressed client request messages without sub-function (data parameter follows service identifier).

- 1) Server sends a positive response message because the service identifier and single data parameter is supported by the client's request message.
- 2) Server sends a positive response message because the service identifier and at least one data parameter is supported by the client's request message.
- 3) Server sends a negative response message (e.g. IMLOIF: incorrectMessageLengthOrIncorrectFormat) because the service identifier is supported and at least one, more than one or all data parameters are supported by the client's request message, but some other error occurred (e.g. wrong length of the request message) during processing of the service.
- 4) Server sends no response message because the negative response code ROOR (request out of range, which would occur because the service identifier is supported, but none of the requested data parameters is supported by the client's request) is always suppressed in case of a functionally addressed request.
- 5) Server sends no response message because the negative response code SNS (serviceNotSupported, which is identified by the server because the service identifier is not supported by the client's request) is always suppressed in case of a functionally addressed request.

7.5.4 Pseudo code example of server response behaviour

The following is a server pseudo code example to describe the logical steps a server shall perform when receiving a request from the client.

```

SWITCH (A_PDU.A_Data.A_PCI.SI)
{
  CASE Service_with_subFunction:                                /* test if service with subFunction is supported */
    SWITCH (A_PDU.A_Data.A_Data.Parameter1 & 0x7F)              /* get subFunction parameter value without bit 7 */
    {
      CASE subFunction_00:                                        /* test if subFunction parameter value is supported */
        IF (message_length == expected_subFunction_message_length) THEN
          :                                                        /* prepare response message */
          responseCode = positiveResponse;                        /* positive response message; set internal NRC = 0x00 */
        ELSE
          responseCode = IMLOIF;                                  /* NRC 0x13: incorrectMessageLengthOrInvalidFormat */
        ENDIF
      BREAK;
    }
}

```

```

CASE subFunction_01:
    :
    responseCode = positiveResponse;
    :
    :
    :
CASE subFunction_127:
    :
    responseCode = positiveResponse;
    BREAK;
DEFAULT:
    responseCode = SFNS;
}
suppressPosRspMsgIndicationBit = (A_PDU.A_Data.Parameter1 & 0x80);
IF ( (suppressPosRspMsgIndicationBit) && (responseCode == positiveResponse) ) THEN
    /* test if positive response is required and if responseCode is positive 0x00 */
    suppressResponse = TRUE;
ELSE
    suppressResponse = FALSE;
ENDIF
BREAK;

CASE Service_without_subFunction:
    suppressResponse = FALSE;
    IF (message_length == expected_message_length) THEN
        IF (A_PDU.A_Data.Parameter1 == supported) THEN
            :
            responseCode = positiveResponse;
        ELSE
            responseCode = ROOR;
        ENDIF
    ELSE
        responseCode = IMLOIF;
    ENDIF
    BREAK;
DEFAULT:
    responseCode = SNS;
}
IF (A_PDU.TA_type == functional && ((responseCode == SNS) || (responseCode == SFNS) || (responseCode == ROOR))) THEN
    /* suppress negative response message */
ELSE
    IF (suppressResponse == TRUE) THEN
        /* suppress positive response message */
    ELSE
        /* send negative or positive response */
    ENDIF
ENDIF

```

When functional addressing is used for the request message, the negative response message with the negative response code (NRC) 78 hex, requestCorrectlyReceivedResponsePending (RCRRP), shall not be implemented if a negative response message with NRC=SNS (serviceNotSupported), NRC=SFNS (subFunctionNotSupported) or NRC=ROOR (requestOutOfRange) is the result of the PDU analysis of the received request message.

7.5.5 Multiple concurrent request messages with physical and functional addressing

A common server implementation has only one diagnostic protocol instance available in the server which can only handle one request at a time. The rule is that any received message (regardless of whether the addressing mode is physical or functional) occupies this resource until the request message is processed (with final response sent or application call without response).

There are only two (2) exceptions which have to be treated separately.

- 1) The keep-alive logic is used by a client to keep a previously enabled session active in one or multiple servers. Keep-Alive-Logic is defined as the functionally addressed valid TesterPresent message with SPRMIB=true and has to be processed by a bypass logic. It is up to the server to make sure that this specific message can not “block” the server’s application layer and that an immediately following addressed message can be processed.
- 2) If a server supports one or more legislated diagnostic requests and one of these requests is received while a non-legislated service (e.g. enhanced diagnostics) is active, then the active service shall be aborted, the default session shall be started and the legislated diagnostic service shall be processed. This requirement does not apply if the programming session is active.

7.5.6 Size of dataIdentifier (DID)

The dataIdentifier (DID) parameter has a size of two (2) bytes in all services throughout ISO 14229.

An implementation standard based on ISO 14229 shall specify the size of the dataIdentifier (DID) parameter if it does not match this International Standard.

8 Service description conventions

8.1 Service description

This clause defines how each diagnostic service is described in ISO 14229. It defines the general service description format of each diagnostic service.

This clause gives a brief outline of the functionality of the service. Each diagnostic service specification starts with a description of the actions performed by the client and the server(s) which are specific to each service. The description of each service includes a table which lists the parameters of its primitives: request/indication, response/confirmation for a positive or negative result. All have the same structure.

For a given request/indication and response/confirmation A_PDU definition, the presence of each parameter is described by one of the following convention (Cvt) values given in Table 8.

Table 8 — A_PDU parameter conventions

Type	Name	Description
M	Mandatory	The parameter shall be present in the A_PDU.
C	Conditional	The parameter can be present in the A_PDU, based on certain criteria (e.g. sub-function/parameters within the A_PDU).
S	Selection	Indicates that the parameter is mandatory (unless otherwise specified) and is a selection from a parameter list.
U	User option	The parameter may or may not be present, depending on dynamic usage by the user.
NOTE The “<Service Name> Request Service Id” marked as “M” (Mandatory) shall not imply that this service must be supported by the server. The “M” only indicates the mandatory presence of this parameter in the request A_PDU if the server supports the service.		

8.2 Request message

8.2.1 Request message definition

This subclause includes multiple tables which define the A_PDU (see Clause 7) parameters for the service request/indication. There might be a separate table for each sub-function parameter (\$Level) if the request messages of the different sub-function parameters (\$Level) differ in the structure of the A_Data parameters and cannot be specified clearly in one table.

Table 9 — Request A_PDU definition with sub-function

A_PDU parameter	Parameter name	Cvt	Hex value	Mnemonic
SA	Source Address	M	xx	SA
TA	Target Address	M	xx	TA
TA_type	Target Address type	M	xx	TAT
RA	Remote Address	C	xx	RA
A_Data.A_PCI.SI	<Service Name> Request Service Id	M	xx	SIDRQ
A_Data. Parameter 1	sub-function = [parameter]	S	xx	LEV_ PARAM
Parameter 2 : Parameter k	data-parameter#1 : data-parameter#k-1	U : U	xx : xx	DP_...#1 : DP_...#k-1
C: The RA (Remote Address) PDU parameter is only present in case of remote addressing.				

Table 10 — Request A_PDU definition without sub-function

A_PDU parameter	Parameter name	Cvt	Hex value	Mnemonic
SA	Source Address	M	xx	SA
TA	Target Address	M	xx	TA
TA_type	Target Address type	M	xx	TAT
RA	Remote Address	C	xx	RA
A_Data.A_PCI.SI	<Service Name> Request Service Id	M	xx	SIDRQ
A_Data. Parameter 1 : Parameter k	data-parameter#1 : data-parameter#k	U : U	xx : xx	DP_...#1 : DP_...#k
C: The RA (Remote Address) PDU parameter is only present in case of remote addressing.				

In all requests/indications, the addressing information TA, SA, and TA_type is mandatory. The addressing information RA may optionally be present.

NOTE The addressing information is shown in the table above for definition purposes. Further service request/indication definitions only specify the A_Data A_PDU parameter because the A_Data A_PDU parameter represents the message data bytes of the service request/indication.

8.2.2 Request message sub-function parameter \$Level (LEV_) definition

This subclause defines the sub-function \$levels (LEV_) parameter(s) defined for the request/indication of the service <Service Name>.

This subclause does not contain any definition for cases where the described service does not use a sub-function parameter value and does not utilize the suppressPosRspMsgIndicationBit (this implicitly indicates that a response is required).

The sub-function parameter byte is divided into two parts (on bit-level) as defined in Table 11.

Table 11 — Sub-function parameter structure

Bit position	Description
7	suppressPosRspMsgIndicationBit This bit indicates if a positive response message shall be suppressed by the server. '0' = FALSE, do not suppress a positive response message (a positive response message is required). '1' = TRUE, suppress response message (a positive response message shall not be sent; the server being addressed shall not send a positive response message). Independent of the suppressPosRspMsgIndicationBit, negative response messages are sent by the server(s) according to the restrictions specified in 7.5.
6-0	sub-function parameter value The bits 0-6 of the sub-function parameter contain the sub-function parameter value of the service (00 - 7F hex). Each service utilizing the sub-function parameter byte, but only supporting the suppressPosRspMsgIndicationBit has to support the zeroSubFunction sub-function parameter value (00 hex).

The sub-function parameter value is a 7-bit value (bits 6-0 of the sub-function parameter byte) that can have multiple values to further specify the service behaviour.

Each service only supporting the suppressPosRspMsgIndicationBit has to support the zeroSubFunction (00 hex).

Services supporting sub-function parameter values in addition to the suppressPosRspMsgIndicationBit shall support the sub-function parameter values as defined in the sub-function parameter value table.

Each service contains a table that defines values for the sub-function parameter values, taking into account only the bits 0-6.

Table 12 — Request message sub-function parameter definition

Hex (bit 6-0)	Description	Cvt	Mnemonic
xx	sub-function#1 description of sub-function parameter#1	M/U	SUBFUNC1
:	:	:	:
xx	sub-function#m description of sub-function parameter#m	M/U	SUBFUNCm

The convention (Cvt) column in the table above shall be interpreted as follows.

Table 13 — Sub-function parameter conventions

Type	Name	Description
M	Mandatory	The sub-function parameter has to be supported by the server if the service is supported.
U	User option	The sub-function parameter may or may not be supported by the server, depending on the usage of the service.

The complete sub-function parameter byte value is calculated based on the value of the suppressPosRspMsgIndicationBit and the sub-function parameter value chosen.

Table 14 — Calculation of the sub-function byte value

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SuppressPosRspMsg-IndicationBit	Sub-function parameter value as specified in the sub-function parameter value table of the service						
Resulting sub-function parameter byte value (bit 7 - 0)							

8.2.3 Request message data parameter definition

This subclause defines the data-parameter(s) \$DataParam (DP_) for the request/indication of the service <Service Name>. This subclause does not contain any definition if the described service does not use any data parameter. The data parameter portion can contain multiple bytes. This subclause provides a generic description of each data parameter; detailed definitions can be found in the annexes of this document. The annexes also specify whether a data parameter shall be supported or is user-optional to be supported if the server supports the service.

Table 15 — Request message data parameter definition

Definition
data-parameter#1 description of data-parameter#1
:
data-parameter#n description of data-parameter#n

8.3 Positive response message

8.3.1 Positive response message definition

This section includes multiple tables that define the A_PDU parameters for the service response/confirmation (see Clause 7 for a detailed description of the application layer protocol data unit A_PDU). There might be a separate table for each sub-function parameter \$Level when the response messages of the different sub-function parameters \$Level differ in the structure of the A_Data parameters.

The positive response message of a diagnostic service (if required) shall be sent after the execution of the diagnostic service. If a diagnostic service requires different handling (e.g. ECUReset service), the appropriate

description when to sent the positive response message can be found in the service description of the diagnostic service.

Table 16 — Positive response A_PDU

A_PDU parameter	Parameter name	Cvt	Hex value	Mnemonic
SA	Source Address	M	xx	SA
TA	Target Address	M	xx	TA
TA_type	Target Address type	M	xx	TAT
RA	Remote Address	C	xx	RA
A_Data.A_PCI.SI	<Service Name> Response Service Id	S	xx	SIDPR
A_Data.Parameter 1 : A_Data.Parameter n	data-parameter#1 : data-parameter#n	U	xx : xx	DP_...#1 : DP_...#n
C: The RA (Remote Address) PDU parameter is only present in case of remote addressing.				

In all responses/confirmations, the addressing information TA, SA, and TA_type is mandatory. The addressing information RA is used if and only if remote addressing is used.

NOTE The addressing information is shown in Table 16 for definition purposes. Further service request/indication definitions only specify the A_Data A_PDU parameter because the A_Data A_PDU parameter represents the message data bytes of the service response/confirmation.

8.3.2 Positive response message data parameter definition

This subclause defines the data parameter(s) for the response/confirmation of the service <Service Name>. It does not contain any definition if the described service does not use any data parameter. The data parameter portion can contain multiple bytes. This subclause provides a generic description of each data parameter. Detailed definitions can be found in the annexes of this document. The annexes also specify whether a data parameter will be supported or is user-optional to be supported if the server supports the service.

Table 17 — Response data parameter definition

Definition
data-parameter#1 description of data-parameter#1. If the request supports a sub-function parameter byte then this parameter is an echo of the 7-bit sub-function parameter value contained within the sub-function parameter byte from the request message with bit 7 set to zero. The suppressPosRspMsgIndicationBit from the sub-function parameter byte is not echoed.
data-parameter#m description of data-parameter#m

8.4 Supported negative response codes (NRC_)

This subclause defines the negative response codes that will be implemented for this service. The circumstances under which each response code would occur are documented in Tables 18 and 19. The definition of the negative response message can be found in section 7.4. The server shall use the negative response A_PDU for the indication of an identified error condition.

The negative response codes listed in A.1 shall be used in addition to the negative response codes specified in each service description if applicable. Details can be found in A.1.

Table 18 — Supported negative response codes

Hex	Description	Cvt	Mnemonic
xx	NegativeResponseCode#1 1. condition#1 : m. condition #m	M	NRC_
:	:	U	NRC_
xx	NegativeResponseCode#n 1. condition#1 : k. condition #k	U	NRC_

The convention (Cvt) column in Table 18 shall be interpreted as follows:

Table 19 — Sub-function parameter conventions

Type	Name	Description
M	Mandatory	The negative response code shall be supported by the server if the service is supported.
U	User option	The negative response code may or may not be supported by the server, depending on the usage of the service.

8.5 Message flow examples

This subclause contains message flow examples for the service <Service Name>. All examples are shown on a message level (without addressing information).

Table 20 — Request message flow example

Message direction:	client → server		
Message type:	Request		
A_Data byte	Description (all values are in hexadecimal)	Byte value (hex)	Mnemonic
#1 (A_PCI)	<Service Name> request Service Id	xx	SIDRQ
#2	sub-function/data-parameter#1	xx	LEV_/DP_
:	:	xx	DP_
#n	data-parameter#m	xx	DP_

Table 21 — Positive response message flow example

Message direction:		server → client	
Message type:		Response	
A_Data	Description (all values are in hexadecimal)	Byte value (hex)	Mnemonic
#1 (A_PCI)	<Service Name> response Service Id	xx	SIDPR
#2	data-parameter#1	xx	DP_
:	:	:	:
#n	data-parameter#n-1	xx	DP_

There might be multiple examples applicable to the service <Service Name> (e.g. one for each sub-function parameter \$Level).

Table 22 shows a message flow example for a negative response message.

Table 22 — Negative response message flow example

Message direction:		server → client	
Message type:		Response	
A_Data	Description (all values are in hexadecimal)	Byte value (hex)	Mnemonic
#1 (A_PCI.NR_SI)	Negative Response Service Id	7F	SIDRSIDNRQ
#2 (A_PCI.SI)	<Service Name> request Service Id	xx	SIDRQ
#3	responseCode	xx	NRC_

9 Diagnostic and communication management functional unit

9.1 Overview

Table 23 — Diagnostic and communication management functional unit

Service	Description
DiagnosticSessionControl	The client requests to control a diagnostic session with a server(s).
ECUReset	The client forces the server(s) to perform a reset.
SecurityAccess	The client requests to unlock a secured server(s).
CommunicationControl	The client requests the server to control its communication.
TesterPresent	The client indicates to the server(s) that it is still present.
AccessTimingParameter	The client uses this service to read/modify the timing parameters for an active communication.
SecuredDataTransmission	The client uses this service to perform data transmission with an extended data link security.
ControlDTCSetting	The client controls the setting of DTCs in the server.
ResponseOnEvent	The client requests to start an event mechanism in the server.
LinkControl	The client requests control of the communication baud rate.

9.2 DiagnosticSessionControl (10 hex) service

9.2.1 Service description

The DiagnosticSessionControl service is used to enable different diagnostic sessions in the server(s).

A diagnostic session enables a specific set of diagnostic services and/or functionality in the server(s). It can, furthermore, enable a data link layer dependent set of timing parameters applicable for the started session. This service provides the capability that the server(s) can report data link layer specific parameter values valid for the enabled diagnostic session (e.g. timing parameter values). The data link layer specific implementation document defines the structure and content of the optional parameter record contained in the response message of this service. The user of this International Standard shall define the exact set of services and/or functionality enabled in each diagnostic session (superset of functionality that is available in the defaultSession).

There shall always be exactly one diagnostic session active in a server. A server shall always start the default diagnostic session when powered up. If no other diagnostic session is started, then the default diagnostic session shall be running as long as the server is powered.

A server shall be capable of providing diagnostic functionality under normal operating conditions and in other operating conditions defined by the vehicle manufacturer, e.g. limp home operation condition.

If the client has requested a diagnostic session which is already running, then the server shall send a positive response message and behave as shown in Figure 9, which describes the server internal behaviour when transitioning between sessions.

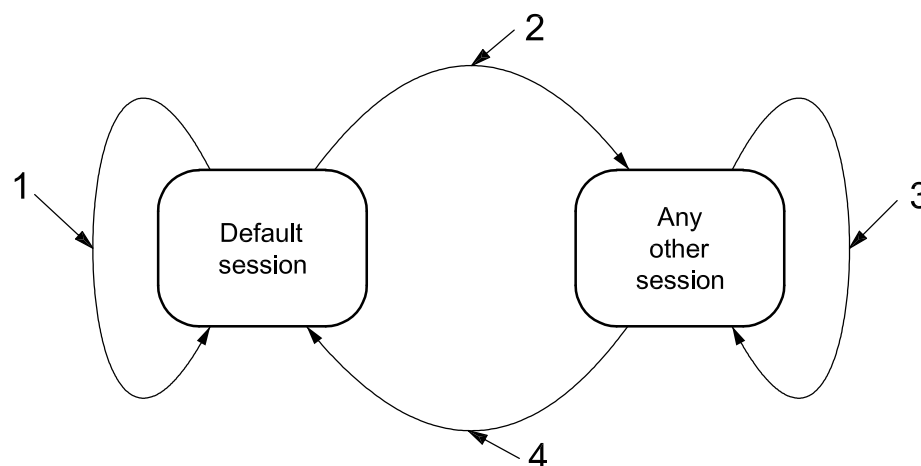
Whenever the client requests a new diagnostic session, the server shall send the DiagnosticSessionControl positive response message before the timings of the new session become active in the server. Some situations may require that the new session must be entered before the positive response is sent while maintaining the old protocol timings for sending the response. If the server is not able to start the requested new diagnostic session, then it shall respond with a DiagnosticSessionControl negative response message and the current session shall continue (see diagnosticSession parameter definitions for further information on

how the server and client shall behave). There shall be only one session active at a time. A diagnostic session enables a specific set of diagnostic services and functions, which shall be defined by the vehicle manufacturer. The set of diagnostic services and diagnostic functionality in a non-default diagnostic session (excluding the programmingSession) is a superset of the functionality provided in the defaultSession, which means that the diagnostic functionality of the defaultSession is also available when switching to any non-default diagnostic session. A session can enable vehicle-manufacturer-specific services and functions which are not part of ISO 14229.

To start a new diagnostic session, a server may request that certain conditions be fulfilled. All such conditions are user-defined. An example of such a condition is the following.

- The server may only allow a client with a certain client identifier (client diagnostic address) to start a specific new diagnostic session (e.g. a server may require that only a client having the client identifier F4 hex may start the extendedDiagnosticSession).
- In some systems, it is desirable to change communication-timing parameters when a new diagnostic session is started. The DiagnosticSessionControl service entity can use the appropriate service primitives to change the timing parameters as specified for the underlying layers to change communication timing in the local node and potentially in the nodes the client wants to communicate with.

Figure 9 provides an overview about the diagnostic session transition and what the server will do when it transitions to another session.



Key

- 1 default session
- 2 other session
- 3 same or other session
- 4 default session

Figure 9 — Server diagnostic session state diagram

The following is a description of diagnostic session transition:

- 1) When the server is in the defaultSession and the client requests to start the defaultSession, then the server shall re-initialize the defaultSession completely. The server shall reset all activated/initiated/changed settings/controls during the activated session. This does not include long-term changes programmed into non-volatile memory.
- 2) When the server transitions from the defaultSession to any other session than the defaultSession, then the server shall only reset the events that have been configured in the server via the ResponseOnEvent (86 hex) service during the defaultSession.
- 3) When the server transitions from any diagnostic session other than the defaultSession to another session other than the defaultSession (including the currently active diagnostic session), then the server shall (re-) initialize the diagnostic session, which means that each event that has been configured in the server via the ResponseOnEvent (86 hex) service shall be reset and that security shall be enabled. Any

configured periodic scheduler shall remain active when transitioning from one non-defaultSession to another or the same non-defaultSession. The states of the CommunicationControl and ControlDTCSetting services shall not be affected, which means, for example, that normal communication shall remain disabled when it is disabled at the point in time at which the session is switched.

- 4) When the server transitions from any diagnostic session other than the defaultSession to the defaultSession, then the server shall reset each event that has been configured in the server via the ResponseOnEvent (86 hex) service and security shall be enabled. Any configured periodic scheduler shall be disabled. Furthermore, the states of the CommunicationControl and ControlDTCSetting services shall be reset, which means, for example, that normal communication shall be re-enabled when it was disabled at the point in time the session is switched to the defaultSession. The server shall reset all activated/initiated/changed settings/controls during the activated session. This does not include long-term changes programmed into non-volatile memory.

Table 24 shows the services which are allowed during the defaultSession and the non-defaultSession (timed services). Any non-defaultSession is tied to a diagnostic session timer that has to be kept active by the client.

Table 24 — Services allowed during default and non-default diagnostic sessions

Service	defaultSession	non-defaultSession
DiagnosticSessionControl - 10 hex	x	x
ECUReset - 11 hex	x	x
SecurityAccess - 27 hex	N/A	x
CommunicationControl - 28 hex	N/A	x
TesterPresent - 3E hex	x	x
AccessTimingParameter - 83 hex	N/A	x
SecuredDataTransmission - 84 hex	N/A	
ControlDTCSetting - 85 hex	N/A	x
ResponseOnEvent - 86 hex	x ^a	x
LinkControl - 87 hex	N/A	x
ReadDataByIdentifier - 22 hex	x ^b	x
ReadMemoryByAddress - 23 hex	x ^c	x
ReadScalingDataByIdentifier - 24 hex	x ^b	x
ReadDataByPeriodicIdentifier - 2A hex	N/A	x
DynamicallyDefineDataIdentifier - 2C hex	x ^d	x
WriteDataByIdentifier - 2E hex	x ^b	x
WriteMemoryByAddress - 3D hex	x ^c	x
ClearDiagnosticInformation - 14 hex	x	x
ReadDTCInformation - 19 hex	x	x
InputOutputControlByIdentifier - 2F hex	N/A	x
RoutineControl - 31 hex	x ^e	x
RequestDownload - 34 hex	N/A	x
RequestUpload - 35 hex	N/A	x
TransferData - 36 hex	N/A	x
RequestTransferExit - 37 hex	N/A	x

^a It is implementation-specific whether the ResponseOnEvent service is also allowed during the defaultSession.

^b Secured data identifiers require a SecurityAccess service and therefore a non-default diagnostic session.

^c Secured memory areas require a SecurityAccess service and therefore a non-default diagnostic session.

^d A data identifier can be defined dynamically in the default and non-default diagnostic session.

^e Secured routines require a SecurityAccess service and therefore a non-default diagnostic session. A routine that needs to be stopped actively by the client also requires a non-default session.

IMPORTANT — The server and the client shall meet the request and response message behaviour as specified in 7.5.2 in the event that those addressing methods are implemented for this service.

9.2.2 Request message

9.2.2.1 Request message definition

Table 25 — Request message definition

A_Data byte	Parameter name	Cvt	Hex value	Mnemonic
#1	DiagnosticSessionControl Request Service Id	M	10	DSC
#2	sub-function = [diagnosticSessionType]	M	00-FF	LEV_ DS_

9.2.2.2 Request message sub-function parameter \$Level (LEV_) definition

The sub-function parameter diagnosticSessionType is used by the DiagnosticSessionControl service to select the specific behaviour of the server. Explanations and usage of the possible diagnostic sessions are detailed below. The following sub-function values are specified [suppressPosRspMsgIndicationBit (bit 7) not shown]:

Table 26 — Request message sub-function parameter definition

Hex (bit 6-0)	Description	Cvt	Mnemonic
00	ISOSAEReserved This value is reserved by ISO 14229.	M	ISOSAERESRVD
01	defaultSession This diagnostic session enables the default diagnostic session in the server(s) and does not support any diagnostic application timeout handling provisions (e.g. no TesterPresent service is necessary to keep the session active). If any other session than the defaultSession has been active in the server and the defaultSession is once again started, then the following implementation rules shall be followed (see also Figure 9). — The server shall stop the current diagnostic session when it has sent the DiagnosticSessionControl positive response message and shall start the newly requested diagnostic session afterwards. — If the server has sent a DiagnosticSessionControl positive response message, it shall have re-locked the server if the client unlocked it during the diagnostic session. — If the server sends a negative response message with the DiagnosticSessionControl request service identifier, the active session shall be continued. If the used data link requires an initialization step, then the initialized server(s) shall start the default diagnostic session by default. No DiagnosticSessionControl with diagnosticSession set to defaultSession shall be required after the initialization step.	M	DS

Table 26 (continued)

Hex (bit 6-0)	Description	Cvt	Mnemonic
02	programmingSession This diagnosticSession enables all diagnostic services required to support the memory programming of a server. If the server runs the programmingSession in the boot software, the programmingSession shall only be left via an ECUReset (11 hex) service initiated by the client, a DiagnosticSessionControl (10 hex) service with sessionType equal to defaultSession, or a session layer timeout in the server. If the server runs in the boot software when it receives the DiagnosticSessionControl (10 hex) service with sessionType equal to defaultSession, or a session layer timeout occurs and a valid application software is present for both cases, then the server shall restart the application software. ISO 14229 does not specify the various implementation methods of how to achieve the restart of the valid application software (e.g. a valid application software can be determined directly in the boot software, during the ECU startup phase when performing an ECUReset, etc.).	U	PRGS
03	extendedDiagnosticSession This diagnosticSession can e.g. be used to enable all diagnostic services required to support the adjustment of functions such as "Idle Speed", "CO Value", etc. in the server's memory. It can also be used to enable diagnostic services, which are not specifically tied to the adjustment of functions.	U	EXTDS
04	safetySystemDiagnosticSession This diagnosticSession enables all diagnostic services required to support safety-system-related functions e.g. airbag deployment.	U	SSDS
05 - 3F	ISOSAEReserved This value is reserved by ISO 14229 for future definition.	M	ISOSAERESRVD
40 - 5F	vehicleManufacturerSpecific This range of values is reserved for vehicle-manufacturer-specific use.	U	VMS
60 - 7E	systemSupplierSpecific This range of values is reserved for system-supplier-specific use.	U	SSS
7F	ISOSAEReserved This value is reserved by ISO 14229 for future definition.	M	ISOSAERESRVD

9.2.2.3 Request message data parameter definition

This service does not support data parameters in the request message.

9.2.3 Positive response message

9.2.3.1 Positive response message definition

Table 27 — Positive response message definition

A_Data byte	Parameter name	Cvt	Hex value	Mnemonic
#1	DiagnosticSessionControl Response Service Id	S	50	DSCPR
#2	diagnosticSessionType	M	00-7F	DS_
#3 : #n	sessionParameterRecord[] #1 = [data#1 : data#m]	C ^a : C	00-FF : 00-FF	SPREC_ DATA_1 : DATA_m

^a C is the presence, structure and content of the sessionParameterRecord and is data-link-layer-dependant and therefore defined in the implementation specification(s) of ISO 14229.

9.2.3.2 Positive response message data parameter definition

Table 28 — Response message data parameter definition

Definition
diagnosticSessionType This parameter is an echo of bits 6 - 0 of the sub-function parameter from the request message.
sessionParameterRecord This parameter record contains session-specific parameter values reported by the server. The content and structure of this parameter record is data-link-layer-specific and can be found in the implementation specification(s) of ISO 14229.

9.2.4 Supported negative response codes (NRC_)

The following negative response codes shall be implemented for this service. The circumstances under which each response code occurs are documented in Table 29.

Table 29 — Supported negative response codes

Hex	Description	Cvt	Mnemonic
12	subFunctionNotSupported Send if the sub-function parameter in the request message is not supported.	M	SFNS
13	incorrectMessageLengthOrInvalidFormat The length of the message is wrong.	M	IMLOIF
22	conditionsNotCorrect This code shall be returned if the criteria for the request DiagnosticSessionControl are not met.	M	CNC

9.2.5 Message flow example(s) DiagnosticSessionControl

9.2.5.1 Example #1 — Start programmingSession

This message flow shows how to enable the diagnostic session “programmingSession” in a server. The client requests a response message by setting the suppressPosRspMsgIndicationBit (bit 7 of the sub-function parameter) to “FALSE” (‘0’). For the given example, it is assumed that the sessionParameterRecord is supported for the data link layer for which the service is implemented.

Table 30 — DiagnosticSessionControl request message flow example #1

Message direction:		client → server	
Message type:		Request	
A_Data byte	Description (all values are in hexadecimal)	Byte value (hex)	Mnemonic
#1	DiagnosticSessionControl request SID	10	DSC
#2	diagnosticSessionType = programmingSession, suppressPosRspMsgIndicationBit = FALSE	02	DS_ECUPRGS

Table 31 — DiagnosticSessionControl positive response message flow example #1

Message direction:		server → client	
Message type:		Response	
A_Data byte	Description (all values are in hexadecimal)	Byte value (hex)	Mnemonic
#1	DiagnosticSessionControl response SID	50	DSCPR
#2	diagnosticSessionType = programmingSession	02	DS_ECUPRGS

9.3 ECUReset (11 hex) service

9.3.1 Service description

The ECUReset service is used by the client to request a server reset.

This service requests the server to effectively perform a server reset based on the content of the resetType parameter value embedded in the ECUReset request message. The ECUReset positive response message (if required) shall be sent before the reset is executed in the server(s). After a successful server reset, the server shall activate the defaultSession.

IMPORTANT — The server and the client shall meet the request and response message behaviour as specified in 7.5.2 in the event that those addressing methods are implemented for this service.

9.3.2 Request message

9.3.2.1 Request message definition

Table 32 — Request message definition

A_Data byte	Parameter name	Cvt	Hex value	Mnemonic
#1	ECUReset Request Service Id	M	11	ER
#2	sub-function = [resetType]	M	00-FF	LEV_ RT_