



DPC: 25/30481279 DC

BSI Group Headquarters

389 Chiswick High Road London W4 4AL

Tel: +44 (0)20 8996 9000

Fax: +44 (0)20 8996 7400

www.bsigroup.com

Date: 12 June 2025

Origin: International

Latest date for receipt of comments: 05 August 2025

Project No. 2023/02358

Responsible committee: AUE/16 Data Communication (Road Vehicles)

Interested committees:

Title: Draft BS ISO 14229-1 Road vehicles. Unified diagnostic services (UDS). Part 1: Application layer.

Please notify admin.start@bsigroup.com if you are aware of any keywords that might assist in clarifying or identifying the standard or of the content of this standard:

- i) has any issues related to 3rd party IPR, patent or copyright
- ii) affects other national standard(s)
- iii) requires additional national guidance or information

WARNING: THIS IS A DRAFT AND MUST NOT BE REGARDED OR USED AS A BRITISH STANDARD. THIS DRAFT IS NOT CURRENT BEYOND 05 August 2025

This draft is issued to allow comments from interested parties; all comments will be given consideration prior to publication. No acknowledgement will normally be sent. **See overleaf for information on the submission of comments.**

No copying is allowed, in any form, without prior written permission from BSI except as permitted under the Copyright, Designs and Patent Act 1988 or for circulation within a nominating organization for briefing purposes. Electronic circulation is limited to dissemination by e-mail within such an organization by committee members.

Further copies of this draft may be purchased from BSI Shop <http://shop.bsigroup.com> or from BSI Customer Services, Tel: +44(0) 20 8996 9001 or email cservices@bsigroup.com. British, International and foreign standards are also available from BSI Customer Services.

Information on the co-operating organizations represented on the committees referenced above may be obtained from <https://standardsdevelopment.bsigroup.com/>.

E-mail: admin.start@bsigroup.com

Introduction

This draft standard is based on national and international discussions. Your comments on this draft are invited and will assist in the preparation of the consequent standard.

For international standards, comments will be reviewed by the relevant UK national committee before sending the consensus UK vote and comments to the international committee, which will then decide appropriate action. If the international standard is approved, it is usual for the text to be published as a British Standard.

For national standards, comments will be reviewed by the relevant UK national committee and the resulting standards published as a British Standard.

UK Vote

Please indicate whether you consider the UK should submit a negative (with supporting technical reasons) or positive vote on this draft. Please indicate if you are aware of any reason why this draft standard should not be published as a British Standard.

Submission of Comments

- **Annotated drafts are not acceptable and will be rejected.**
- All comments should be submitted online at <https://standardsdevelopment.bsigroup.com/>. You will need to register in order to comment.

Template for comments and secretariat observations

Date: xx/xx/20xx	Document: ISO/DIS xxxx
------------------	------------------------

1	2	(3)	4	5	(6)	7
MB	Clause No./ Subclause No./Annex (e.g. 3.1)	Paragraph/ Figure/Table/Note	Type of comment	Comment (justification for change) by the MB	Proposed change by the MB	Secretariat observations on each comment submitted
	3.1	Definition 1	ed	Definition is ambiguous and needs clarifying.	Amend to read '...so that the mains connector to which no connection...'	
	6.4	Paragraph 2	te	The use of the UV photometer as an alternative cannot be supported as serious problems have been encountered in its use in the UK.	Delete reference to UV photometer.	



DRAFT International Standard

ISO/DIS 14229-1

Road vehicles — Unified diagnostic services (UDS) —

Part 1: Application layer

*Véhicules routiers — Services de diagnostic unifiés (SDU) —
Partie 1: Couche application*

ICS: 43.180

ISO/TC 22/SC 31

Secretariat: **DIN**

Voting begins on:
2025-06-12

Voting terminates on:
2025-09-04

This document is circulated as received from the committee secretariat.

THIS DOCUMENT IS A DRAFT CIRCULATED FOR COMMENTS AND APPROVAL. IT IS THEREFORE SUBJECT TO CHANGE AND MAY NOT BE REFERRED TO AS AN INTERNATIONAL STANDARD UNTIL PUBLISHED AS SUCH.

IN ADDITION TO THEIR EVALUATION AS BEING ACCEPTABLE FOR INDUSTRIAL, TECHNOLOGICAL, COMMERCIAL AND USER PURPOSES, DRAFT INTERNATIONAL STANDARDS MAY ON OCCASION HAVE TO BE CONSIDERED IN THE LIGHT OF THEIR POTENTIAL TO BECOME STANDARDS TO WHICH REFERENCE MAY BE MADE IN NATIONAL REGULATIONS.

RECIPIENTS OF THIS DRAFT ARE INVITED TO SUBMIT, WITH THEIR COMMENTS, NOTIFICATION OF ANY RELEVANT PATENT RIGHTS OF WHICH THEY ARE AWARE AND TO PROVIDE SUPPORTING DOCUMENTATION.

Reference number
ISO/DIS 14229-1:2025(en)

© ISO 2025



Documents

Document Sharing Hub



COPYRIGHT PROTECTED DOCUMENT

© ISO 2025

All rights reserved. Unless otherwise specified, or required in the context of its implementation, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

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

Published in Switzerland

Contents

Page

Foreword	viii
Introduction	ix
1 Scope	1
2 Normative references	1
3 Terms and definitions	2
4 Symbols and abbreviated terms	4
4.1 Symbols	4
4.2 Abbreviated terms	4
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 service primitives	8
6.3.1 General definition	8
6.3.2 Service request and service indication primitives	9
6.3.3 Service response and service confirm primitives	9
6.3.4 Service request-confirm and service response-confirm primitives	10
6.4 Service data unit specification	11
6.4.1 Mandatory parameters	11
6.4.2 Vehicle system requirements	13
6.4.3 Optional parameters - A_AE, application layer remote address	13
7 Application layer protocol	14
7.1 General definition	14
7.2 A_PDU, application protocol data unit	14
7.3 A_PCI, application protocol control information	15
7.4 SI, service identifier	15
7.5 A_NR_SI, Negative response service identifier	16
7.6 Negative response/confirmation service primitive	16
7.7 Server response implementation rules	17
7.7.1 General definitions	17
7.7.2 General server response behaviour	17
7.7.3 Request message with SubFunction parameter and server response behaviour	19
7.7.4 Request message without SubFunction parameter and server response behaviour	22
7.7.5 Pseudo code example of server response behaviour	24
7.7.6 Multiple concurrent request messages with physical and functional addressing	25
8 Service description conventions	26
8.1 Service description	26
8.2 Request message	26
8.2.1 Request message definition	26
8.2.2 Request message SubFunction parameter \$Level (LEV_) definition	27
8.2.3 Request message data-parameter definition	29
8.3 Positive response message	29
8.3.1 Positive response message definition	29
8.3.2 Positive response message data-parameter definition	30
8.4 Supported negative response codes (NRC_)	30
8.5 Message flow examples	31
9 Diagnostic and communication management functional unit	32
9.1 Overview	32
9.2 DiagnosticSessionControl (10 ₁₆) service	32
9.2.1 Service description	32

9.2.2	Request message	35
9.2.3	Positive response message	37
9.2.4	Supported negative response codes (NRC_)	38
9.2.5	Message flow example(s) DiagnosticSessionControl – Start programmingSession	38
9.3	ECUReset (11 ₁₆) service	39
9.3.1	Service description	39
9.3.2	Request message	39
9.3.3	Positive response message	41
9.3.4	Supported negative response codes (NRC_)	41
9.3.5	Message flow example ECUReset	42
9.4	SecurityAccess (27 ₁₆) service	42
9.4.1	Service description	42
9.4.2	Request message	44
9.4.3	Positive response message	45
9.4.4	Supported negative response codes (NRC_)	46
9.4.5	Message flow example(s) SecurityAccess	47
9.5	CommunicationControl (28 ₁₆) service	49
9.5.1	Service description	49
9.5.2	Request message	49
9.5.3	Positive response message	50
9.5.4	Supported negative response codes (NRC_)	51
9.5.5	Message flow example CommunicationControl (disable transmission of network management messages)	51
9.5.6	Message flow example CommunicationControl (switch a remote network into the diagnostic-only scheduling mode where the node with address 000A ₁₆ is connected to)	52
9.5.7	Message flow example CommunicationControl (switch to application scheduling mode with enhanced address information, the node 000A ₁₆ , which is connected to a sub-network, is addressed)	52
9.6	Authentication (29 ₁₆) service	53
9.6.1	Service overview	53
9.6.2	Authentication with PKI Certificate Exchange (APCE)	54
9.6.3	Authentication with Challenge-Response (ACR)	59
9.6.4	Common requirements	63
9.6.5	Request message	64
9.6.6	Positive response message	70
9.6.7	Supported negative response codes (NRC_)	77
9.6.8	Message flow example(s) Authentication	78
9.7	TesterPresent (3E ₁₆) service	97
9.7.1	Service description	97
9.7.2	Request message	98
9.7.3	Positive response message	98
9.7.4	Supported negative response codes (NRC_)	99
9.7.5	Message flow example(s) TesterPresent	99
9.8	ControlDTCSetting (85 ₁₆) service	100
9.8.1	Service description	100
9.8.2	Request message	100
9.8.3	Positive response message	101
9.8.4	Supported negative response codes (NRC_)	102
9.8.5	Message flow example(s) ControlDTCSetting	102
9.9	ResponseOnEvent (86 ₁₆) service	103
9.9.1	Service description	103
9.9.2	Request message	111
9.9.3	Positive response message	118
9.9.4	Supported negative response codes (NRC_)	120
9.9.5	Message flow example(s) ResponseOnEvent	121
9.10	LinkControl (87 ₁₆) service	135
9.10.1	Service description	135
9.10.2	Request message	136

9.10.3	Positive response message.....	138
9.10.4	Supported negative response codes (NRC_).....	138
9.10.5	Message flow example(s) LinkControl.....	139
10	Data transmission functional unit.....	141
10.1	Overview.....	141
10.2	ReadDataByIdentifier (22 ₁₆) service.....	142
10.2.1	Service description.....	142
10.2.2	Request message.....	142
10.2.3	Positive response message.....	143
10.2.4	Supported negative response codes (NRC_).....	144
10.2.5	Message flow example ReadDataByIdentifier.....	146
10.3	ReadMemoryByAddress (23 ₁₆) service.....	148
10.3.1	Service description.....	148
10.3.2	Request message.....	148
10.3.3	Positive response message.....	149
10.3.4	Supported negative response codes (NRC_).....	150
10.3.5	Message flow example ReadMemoryByAddress.....	152
10.4	ReadScalingDataByIdentifier (24 ₁₆) service.....	154
10.4.1	Service description.....	154
10.4.2	Request message.....	154
10.4.3	Positive response message.....	155
10.4.4	Supported negative response codes (NRC_).....	156
10.4.5	Message flow example ReadScalingDataByIdentifier.....	157
10.5	ReadDataByPeriodicIdentifier (2A ₁₆) service.....	160
10.5.1	Service description.....	160
10.5.2	Request message.....	163
10.5.3	Positive response message.....	163
10.5.4	Supported negative response codes (NRC_).....	164
10.5.5	Message flow example ReadDataByPeriodicIdentifier.....	167
10.6	DynamicallyDefineDataIdentifier (2C ₁₆) service.....	177
10.6.1	Service description.....	177
10.6.2	Request message.....	178
10.6.3	Positive response message.....	182
10.6.4	Supported negative response codes (NRC_).....	182
10.6.5	Message flow examples DynamicallyDefineDataIdentifier.....	183
10.7	WriteDataByIdentifier (2E ₁₆) service.....	196
10.7.1	Service description.....	196
10.7.2	Request message.....	196
10.7.3	Positive response message.....	197
10.7.4	Supported negative response codes (NRC_).....	197
10.7.5	Message flow example WriteDataByIdentifier.....	199
10.8	WriteMemoryByAddress (3D ₁₆) service.....	200
10.8.1	Service description.....	200
10.8.2	Request message.....	201
10.8.3	Positive response message.....	202
10.8.4	Supported negative response codes (NRC_).....	203
10.8.5	Message flow example WriteMemoryByAddress.....	205
11	Stored data transmission functional unit.....	207
11.1	Overview.....	207
11.2	ClearDiagnosticInformation (14 ₁₆) service.....	207
11.2.1	Service description.....	207
11.2.2	Request message.....	208
11.2.3	Positive response message.....	208
11.2.4	Supported negative response codes (NRC_).....	209
11.2.5	Message flow example ClearDiagnosticInformation.....	210
11.3	ReadDTCInformation (19 ₁₆) service.....	211
11.3.1	Service description.....	211
11.3.2	Request message.....	221

11.3.3	Positive response message.....	231
11.3.4	Supported negative response codes (NRC_).....	247
11.3.5	Message flow examples – ReadDTCInformation.....	248
12	InputOutput control functional unit.....	276
12.1	Overview.....	276
12.2	InputOutputControlByIdentifier (2F ₁₆) service.....	277
12.2.1	Service description.....	277
12.2.2	Request message.....	277
12.2.3	Positive response message.....	279
12.2.4	Supported negative response codes (NRC_).....	280
12.2.5	Message flow example(s) InputOutputControlByIdentifier.....	282
13	Routine functional unit.....	289
13.1	Overview.....	289
13.2	RoutineControl (31 ₁₆) service.....	290
13.2.1	Service description.....	290
13.2.2	Request message.....	291
13.2.3	Positive response message.....	292
13.2.4	Supported negative response codes (NRC_).....	293
13.2.5	Message flow example(s) RoutineControl.....	296
14	Upload download functional unit.....	299
14.1	Overview.....	299
14.2	RequestDownload (34 ₁₆) service.....	299
14.2.1	Service description.....	299
14.2.2	Request message.....	299
14.2.3	Positive response message.....	301
14.2.4	Supported negative response codes (NRC_).....	301
14.2.5	Message flow example(s) RequestDownload.....	303
14.3	RequestUpload (35 ₁₆) service.....	304
14.3.1	Service description.....	304
14.3.2	Request message.....	304
14.3.3	Positive response message.....	305
14.3.4	Supported negative response codes (NRC_).....	306
14.3.5	Message flow example(s) RequestUpload.....	307
14.4	TransferData (36 ₁₆) service.....	308
14.4.1	Service description.....	308
14.4.2	Request message.....	308
14.4.3	Positive response message.....	309
14.4.4	Supported negative response codes (NRC_).....	310
14.4.5	Message flow example(s) TransferData.....	312
14.5	RequestTransferExit (37 ₁₆) service.....	312
14.5.1	Service description.....	312
14.5.2	Request message.....	313
14.5.3	Positive response message.....	313
14.5.4	Supported negative response codes (NRC_).....	314
14.5.5	Message flow example(s) for downloading/uploading data.....	315
14.6	RequestFileTransfer (38 ₁₆) service.....	321
14.6.1	Service description.....	321
14.6.2	Request message.....	322
14.6.3	Positive response message.....	323
14.6.4	Supported negative response codes (NRC_).....	325
14.6.5	Message flow example(s) RequestFileTransfer.....	327
15	Security sub-layer definition.....	330
15.1	General.....	330
15.1.1	Purpose.....	330
15.1.2	Security sub-layer description.....	330
15.1.3	Security sub-layer access.....	331
15.1.4	General server response behaviour.....	334

15.2	SecuredDataTransmission (84 ₁₆) service	335
15.2.1	Service description.....	335
15.2.2	Request message	335
15.2.3	Positive response message for successful internal message	337
15.2.4	Supported negative response codes (NRC_).....	339
15.2.5	Message flow example SecuredDataTransmission.....	339
16	Non-volatile server memory programming process	342
16.1	General information.....	342
16.2	Detailed programming sequence.....	345
16.2.1	Programming phase #1 — Download of application software and/or application data.....	345
16.3	Server reprogramming requirements.....	354
16.3.1	Requirements for servers to support programming.....	354
16.3.2	Software, data identification and fingerprints.....	357
16.3.3	Server routine access	358
16.4	Non-volatile server memory programming message flow examples.....	358
16.4.1	General information.....	358
16.4.2	Programming phase #1 — Pre-Programming step.....	358
16.4.3	Programming phase #1 — Programming step.....	359
16.4.4	Programming phase #1 — Post-Programming step.....	363
	Annex A (normative) Global parameter definitions	364
	Annex B (normative) Diagnostic and communication management functional unit data-parameter definitions	372
	Annex C (normative) Data transmission functional unit data-parameter definitions.....	376
	Annex D (normative) Stored data transmission functional unit data-parameter definitions.....	389
	Annex E (normative) Input output control functional unit data-parameter definitions	406
	Annex F (normative) Routine functional unit data-parameter definitions.....	407
	Annex G (normative) Upload and download functional unit data-parameter	408
	Annex H (informative) Examples for addressAndLengthFormatIdentifier parameter values	409
	Annex I (normative) Security access state chart.....	411
	Annex J (normative) Implementation for multiple client environments.....	421
	Bibliography	427

Foreword

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

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

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

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

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

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

This third edition cancels and replaces the second edition (ISO 14229-1:2013), which has been technically revised. The main changes compared to the previous edition are as follows:

- new diagnostic service for Authentication has been introduced to address cyber security topics;
- new clause "Security sub-layer definition";
- some unused SubFunction of ReadDTCInformation service are deleted, e.g. Mirror Memory;
- the ReadDataByPeriodicIdentifier is updated; and
- several clarifications and corrections are implemented.

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

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

Introduction

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

To achieve this, ISO 14229 is based on the Open Systems Interconnection (OSI) Basic Reference Model in accordance with ISO/IEC 7498-1^[1] and ISO/IEC 10731^[2], which structures communication systems into seven layers in accordance with [Figure 1](#).

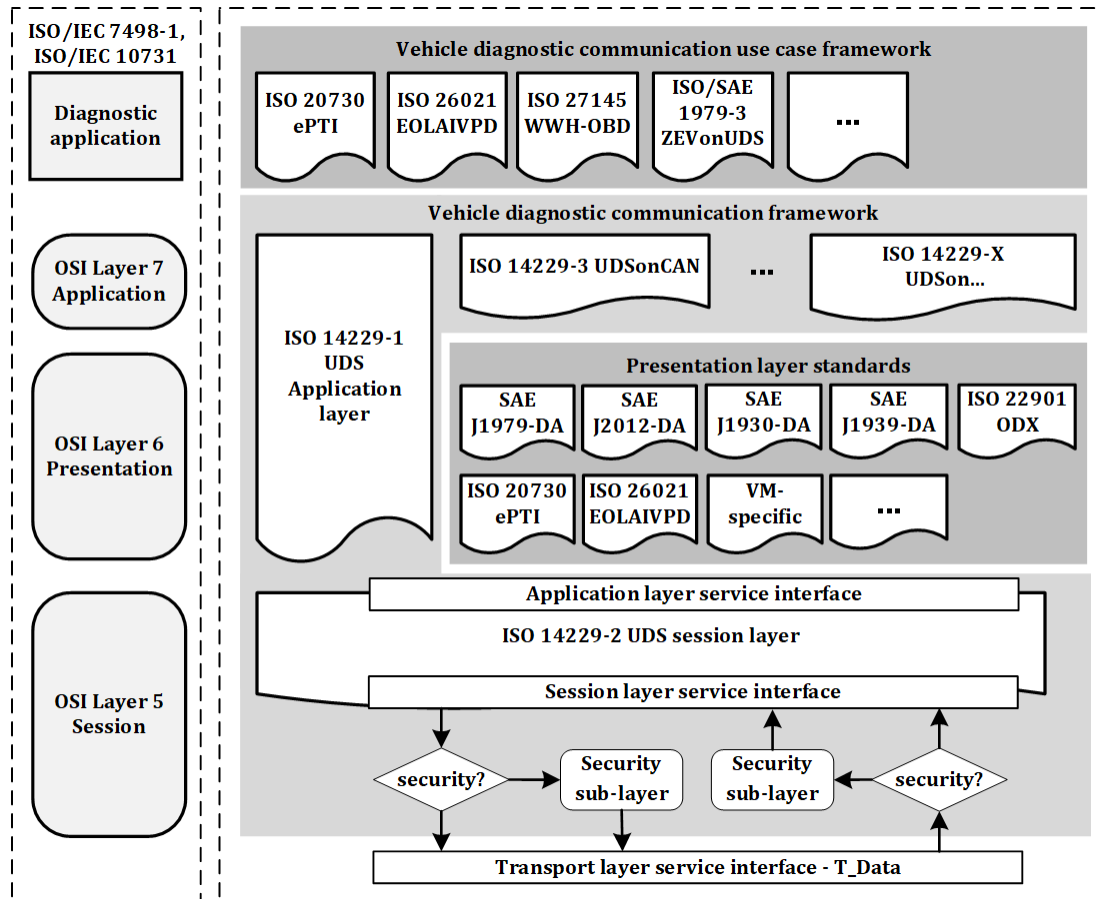


Figure 1 — UDS-based vehicle diagnostic communication framework according to OSI model

When mapped on the OSI model, the services used by a diagnostic tester (client) and an Electronic Control Unit (ECU, server) are broken into the following layers:

- Application layer (layer 7), unified diagnostic services specified in this document, ^[3] UDSONCAN to ISO 14229-X UDSON..., ^[4] ePTI, ^[5] EOLAIVPD, ^[6] WWH-OBd, ^[7] ZEVonUDS, and further standards.
- Presentation layer (layer 6), vehicle manufacturer specific, ^[8], ^[9], ^[10], ^[11], ^[12] ODX, ^[13] ePTI, and ^[14] EOLAIVPD.
- Session layer services (layer 5) specified in ISO 14229-2.

NOTE The diagnostic services in this document are implemented in various applications, e.g. road vehicles – tachograph systems, road vehicles – interchange of digital information on electrical connections between towing and towed vehicles, road vehicles – diagnostic systems, etc. Future modifications to this document provide long-term backward compatibility with the implementation standards as described above.

Road vehicles — Unified diagnostic services (UDS) —

Part 1: Application layer

1 Scope

This document specifies data link independent requirements of diagnostic communication services, which allow a diagnostic tester (client) to control diagnostic functions in an in-vehicle electronic control unit (ECU, server) such as an electronic fuel injection, automatic gearbox, anti-lock braking system, etc. connected to a serial data link embedded in a road vehicle.

It specifies diagnostic communication services, which allow the diagnostic tester (client) to stop or to resume non-diagnostic message transmission on the data link.

This document does not apply to non-diagnostic message transmission on the vehicle's communication data link between two electronic control units. This document does not restrict an in-vehicle on-board tester (client) implementation in an ECU/server in order to utilize the diagnostic communication services on the vehicle's communication data link to perform bidirectional diagnostic data exchange.

This document does not specify any implementation requirements.

2 Normative references

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

ISO 26021-3:2022, *Road vehicles — End-of-life activation of in-vehicle pyrotechnic devices — Part 3: Data definitions*

IEEE 754-2008, *IEEE Standard for Floating-Point Arithmetic*

IEEE 1609.2, *Standard for Wireless Access in Vehicular Environments — Security Services for Applications and Management Messages*

ISO 7816-8, *Identification cards — Integrated circuit cards — Part 8: Commands and mechanisms for security operations*

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

ISO/IEC 9594-8, *Information technology — Open Systems Interconnection — The Directory — Part 8: Public-key and attribute certificate frameworks*

ITU-T X.690, *Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*

RFC 5280, *Internet Engineering Task Force — Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*

RFC 5755, *Internet Engineering Task Force — An Internet Attribute Certificate Profile for Authorization*

X.509, *Information technology — Open Systems Interconnection — The Directory: Public-key and attribute certificate frameworks*

3 Terms and definitions

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

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

- ISO Online browsing platform: available at <https://www.iso.org/obp>
- IEC Electropedia: available at <http://www.electropedia.org/>

3.1

boot memory partition

area of the *server* (3.18) memory in which the *boot software* (3.2) is located

3.2

boot software

software which is executed in a special part of *server* (3.18) memory which is used primarily to boot the *ECU* (3.9) and perform server programming

Note 1 to entry: This area of memory is not erased during a normal programming sequence and executes when the server application is missing or otherwise deemed invalid to always ensure the capability to reprogram the server.

Note 2 to entry: See 9.2.2 and Clause 16.

3.3

client

function that is part of the *tester* (3.20) and that makes use of the *diagnostic services* (3.6)

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

3.4

diagnostic channel

dedicated transmission path from *client* (3.3) to *server* (3.18) for diagnostic communication

Note 1 to entry: Several simultaneously connected clients to one server can be differentiated by an individual *tester* (3.20) source address.

3.5

diagnostic data

data that is located in the memory of an *electronic control unit* (3.9) which may be inspected and/or possibly modified by the *tester* (3.20)

Note 1 to entry: Diagnostic data includes analogue inputs and outputs, digital inputs and outputs, intermediate values and various status information.

Note 2 to entry: Examples of diagnostic data are 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* (3.18) is not obliged to keep internal copies of its data for diagnostic purposes, in which case the tester may only request the current value.

Note 3 to entry: 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).

3.6

diagnostic service

information exchange initiated by a *client* (3.3) in order to require diagnostic information from a *server* (3.18) or/and to modify its behaviour for diagnostic purpose

3.7

diagnostic session

state within the *server* (3.18) in which a specific set of *diagnostic services* (3.6) and functionality is enabled

3.8

diagnostic trouble code

DTC

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

3.9

ECU

electronic control unit

unit providing information regarding the connected sensor and control network

Note 1 to entry: Systems considered as electronic control units include anti-lock braking system (ABS) and engine management system.

3.10

functional unit

set of functionally close or complementary *diagnostic services* (3.6)

3.11

local server

server (3.18) that is connected to the same local network as the *client* (3.3) and is part of the same address space as the client

3.12

permanent DTC

diagnostic trouble code (3.8) that remains in non-volatile memory, even after a clear DTC request, until other criteria (typically regulatory) are met (e.g. the appropriate monitors for each DTC have successfully passed)

Note 1 to entry: Refer to the relevant legislation for all necessary requirements.

3.13

record

one or more *diagnostic data* (3.5) elements that are referred to together by a single means of identification

Note 1 to entry: A snapshot including various input/output data and trouble codes is an example of a record.

3.14

remote server

server (3.18) that is not directly connected to the main diagnostic network

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

Note 2 to entry: A remote server is reached via a *local server* (3.11) 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 therefore always identifies a remote server: one local address that identifies the gate to the remote network and one remote address identifying the remote server itself.

3.15

remote client

client (3.3) that is not directly connected to the main diagnostic network

Note 1 to entry: A remote client is identified by means of a remote address.

Note 2 to entry: Remote addresses represent an own address space that is independent from the addresses on the main network.

3.16

reprogramming software

part of the *boot software* (3.2) that allows for reprogramming of the *electronic control unit* (3.9)

3.17

security

mechanism for protecting vehicle modules from "unauthorized" intrusion through a vehicle *diagnostic data* (3.5) link

3.18

server

function that is part of an *electronic control unit* (3.9) and that provides the *diagnostic services* (3.6)

Note 1 to entry: This document differentiates between the server (i.e. the function) and the electronic control unit to ensure independence from implementation.

3.19

supported DTC

diagnostic trouble code (3.8) which is currently configured/calibrated and enabled to execute under predefined vehicle conditions.

3.20

tester

system that controls functions such as test, inspection, monitoring, or diagnosis of an on-vehicle *electronic control unit* (3.9) and can be dedicated to a specific type of operator (e.g. an off-board scan tool dedicated to garage mechanics, an off-board test tool dedicated to assembly plants, or an on-board tester)

Note 1 to entry: The tester is also referenced as the *client* (3.3).

4 Symbols and abbreviated terms

4.1 Symbols

$t_{L5_S3_Client}$	OSI layer 5 client session timer
$t_{L5_S3_Server}$	OSI layer 5 server session timer
$t_{L7_P2_Client}$	OSI layer 7 client reception timer
$t_{L7_P2_Server}$	OSI layer 7 server response timer
$t_{L7_P2_Client_Max}$	OSI layer 7 client reception time enhanced maximum value
$t_{L7_P2_Server_Max}$	OSI layer 7 server response time enhanced maximum value

4.2 Abbreviated terms

.con	service primitive .confirmation
.ind	service primitive .indication
.req	service primitive .request
A_PCI	application layer protocol control information
ACR	Authentication with Challenge-Response

APCE	Authentication with PKI Certificate Exchange
BER	Basic Encoding Rules according to ITU-T X.690
CMAC	Cipher-based Message Authentication Code
CVC	Card Verifiable Certificate
ECU	electronic control unit
EDR	event data recorder
EOLAIVPD	end-of-life activation of in-vehicle pyrotechnic devices
ePTI	vehicle interface for electronic Periodic Technical Inspection
GMAC	Galois Message Authentication Code
HMAC	Hash-based Message Authentication Code
N/A	not applicable
NR_SI	negative response service identifier
NRC	negative response code
OID	Object Identifier according to ISO/IEC 9834-1
OSI	open systems interconnection
PKCS	Public-Key Cryptography Standards
PKI	Public-Key Infrastructure
POWN	Proof of Ownership
RA	remote address
SI	service identifier
TA	target address
SA	source address
SI	service identifier
TA	target address
TA_type	target address type
WWH-OBD	World-Wide Harmonized On-Board DiagnosticsISO/IEC 9594-8
ZEVonUDS	zero emission propulsion system on unified diagnostic services

5 Conventions

This document is based on the conventions discussed in the OSI service conventions (ISO/IEC 10731^[2]) as they apply for diagnostic services.

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

The distinction between service and protocol is summarised in [Figure 2](#).

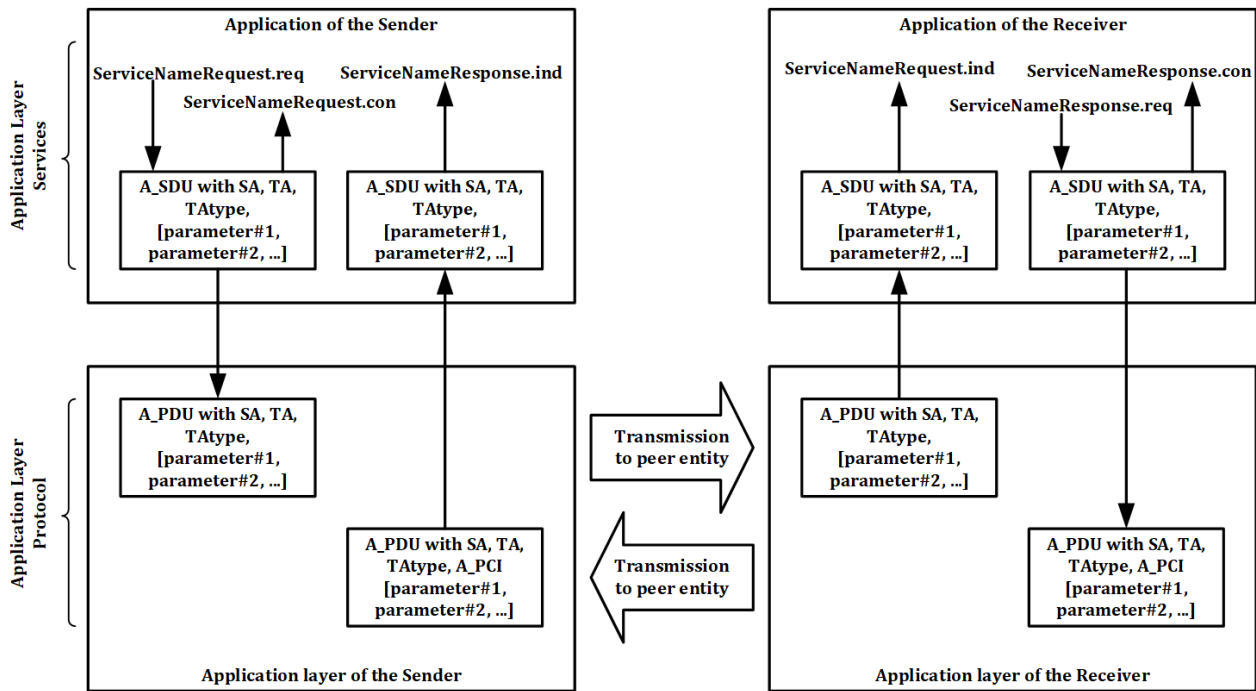


Figure 2 — The services and the protocol

This document defines both confirmed and unconfirmed services.

The confirmed services use the six service primitives request, req_confirm, indication, response, rsp_confirm and confirmation.

The unconfirmed services use only the request, req_confirm and indication service primitives.

For all services defined in this document, 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 document, only the request and response service primitives are listed.

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 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 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 successfully sent on the vehicle communication bus the diagnostic tester is connected to;
- 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 successfully sent on the vehicle communication bus the ECU received the diagnostic request on;
- a **service confirmation primitive**, used by the diagnostics application layer to pass data to the client function in the diagnostic tester application.

Figure 3 depicts the application layer service primitives - confirmed service.

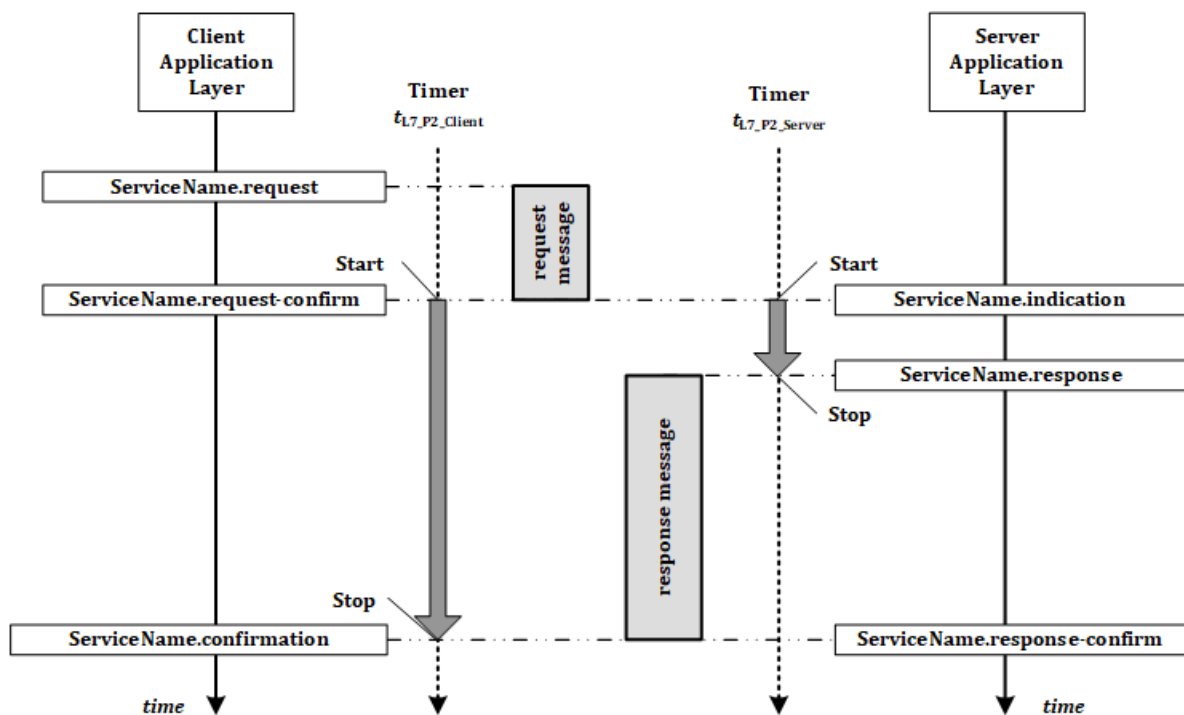


Figure 3 — Application layer service primitives – Confirmed service

Figure 4 depicts the application layer service primitives – Unconfirmed service.

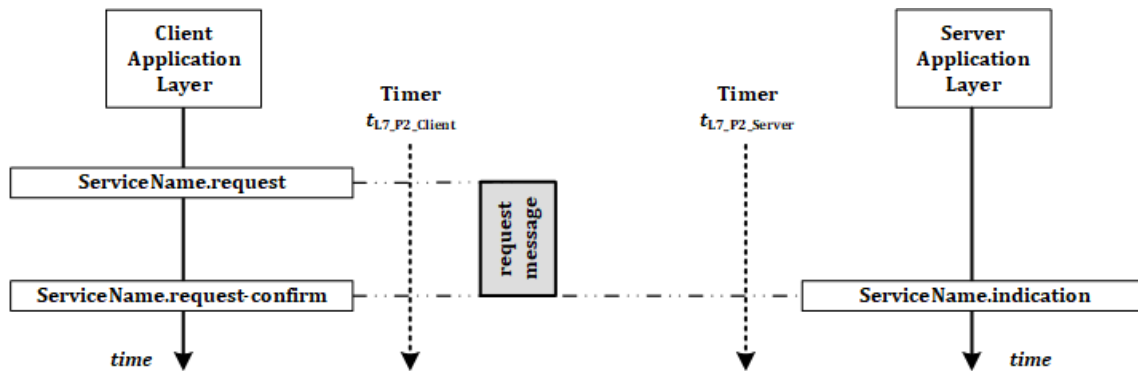


Figure 4 — Application layer service primitives – Unconfirmed service

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 this document does not make use of those service primitives, but the data link specific implementation documents might use those to define, for example service execution reference points (e.g. the ECUReset service would invoke the reset when the response is completely transmitted to the client which is indicated in the server via the service response-confirm primitive).

6.2 Format description of application layer services

Application layer services can have four different formats depending on how the vehicle diagnostic system is configured. The format of the application layer service is controlled by parameter A_Mtype.

If the vehicle system is configured so that the client can address all servers by using the A_SA and A_TA address parameters, the default format of application layer services shall be used. This implies A_Mtype = diagnostics.

If the vehicle system is configured so that the client needs address information in addition to the A_SA and A_TA address parameters allowing to address certain servers, the remote format of application layers services shall be used. This implies A_Mtype = remote diagnostics.

If the vehicle system is configured so that the client can address the server which treats secure data by using the A_SA and A_TA address parameters, the secure format of application layer services shall be used. This implies A_Mtype = secure diagnostics.

If the vehicle system is configured so that the client needs address information in addition to the A_SA and A_TA address parameters allowing to address the server which treats secure data, the secure remote format of application layers services shall be used. This implies A_Mtype = secure remote diagnostics.

The different formats for application layer services are specified in [6.3](#).

6.3 Format description of 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 (Application layer Service Data Unit) 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 (
    A_MType,
    A_SA,
    A_TA,
    A_TA_type,
    [A_AE],
    A_Length,
    A_Data[,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 (
    A_MType,
    A_SA,
    A_TA,
    A_TA_type,
    [A_AE],
    A_Length,
    A_Data[,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 pass data about the requested diagnostic service to the 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 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 (
    A_Mtype,
```

```

A_SA,
A_TA,
A_TA_type,
[A_AE],
A_Length
A_Data[,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 (
    A_Mtype,
    A_SA,
    A_TA,
    A_TA_type,
    [A_AE],
    A_Length
    A_Data[,parameter 1, ...],
)

```

The 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. 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 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) are 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 (
    A_Mtype,
    A_SA,
    A_TA,
    A_TA_type,
    [A_AE],
    A_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 communication results of an associated previous service request to the client function in the diagnostic tester application.

```

service_name.rsp_confirm (
    A_Mtype,
    A_SA,

```

```

A_TA,
A_TA_type,
[A_AE],
A_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 communication results of an associated previous service response to the server function in the ECU application.

6.4 Service data unit specification

6.4.1 Mandatory parameters

6.4.1.1 General definition

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

6.4.1.2 A_Mtype, application layer message type

Type: enumeration

Range: diagnostics, remote diagnostics, secure diagnostics, secure remote diagnostics

Description:

The parameter Mtype shall be used to identify the format of the vehicle diagnostic system as specified in [6.2](#). This document specifies a range of four values for this parameter:

- If A_Mtype = diagnostics, then the service_name primitive shall consist of the parameters A_SA, A_TA and A_TAtype.
- If A_Mtype = remote diagnostics, then the service_name primitive shall consist of the parameters A_SA, A_TA, A_TAtype and A_AE.
- If A_Mtype = secure diagnostics, then the service_name primitive shall consist of the parameters A_SA, A_TA and A_TAtype.
- If A_Mtype = secure remote diagnostics, then the service_name primitive shall consist of the parameters A_SA, A_TA, A_TAtype and A_AE.

6.4.1.3 A_SA, application layer source address

Type: 2 byte unsigned integer value

Range: 0000₁₆ to FFFF₁₆

Description:

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

For service requests (and service indications), A_SA represents the address of the client function that has requested the diagnostic service. Each client function that requests diagnostic services shall be represented with one A_SA value. If more than one client function is implemented in the same diagnostic tester, then each client function shall have its own client identifier and corresponding A_SA value.

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

several ECUs, then the respective server function addresses shall be encoded with one A_SA value for each individual server function.

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

NOTE The A_SA value in a response message is the same as the A_TA value in the corresponding request message if physical addressing was used for the request message.

6.4.1.4 A_TA, application layer target address

Type: 2 byte unsigned integer value

Range: 0000₁₆ to FFFF₁₆

Description:

The parameter A_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 function that shall respond to a diagnostic service request or if the server function is implemented as a distributed function 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), A_TA represents the server identifier for the server that shall perform the requested diagnostic service. If a remote server is being addressed, then A_TA represents the local server that is the gate from the main network to the remote network.

For service responses (and service confirmations), A_TA represents the address of the client function that originally requested the diagnostic service and shall receive the requested data (i.e. A_SA of the request). Service responses (and service confirmations) shall always use physical addressing. If a remote client is being addressed, then A_TA represents the local server that is the gate from the main network to the remote network.

NOTE The A_TA value of a response message is always be the same as the A_SA value of the corresponding request message.

6.4.1.5 A_TA_Type, application layer target address type

Type: enumeration

Range: physical, functional

Description:

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

6.4.1.6 A_Result

Type: enumeration

Range: ok, error

Description:

The parameter 'A_Result' is used by the req_confirm and rsp_confirm primitives to indicate if a message has been transmitted correctly (ok) or whether the message transmission was not successful (error).

6.4.1.7 A_Length

Type: 4 byte unsigned integer value

Range: 0_d to $(2^{32}-1)_d$

Description:

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

6.4.1.8 A_Data

Type: string of bytes

Range: not applicable

Description:

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

6.4.2 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 addresses of the 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 A_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.

6.4.3 Optional parameters - A_AE, application layer remote address

Type: 2 byte unsigned integer value

Range: 0000_{16} to $FFFF_{16}$

Description:

A_AE is used to extend the available address range to encode client and server identifiers. A_AE 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 A_AE shall be used to encode remote client and server identifiers. A_AE can represent either a remote target address or a remote source address depending on the direction of the message carrying the A_AE.

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

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

There is no special parameter that represents physical or functional remote addresses in the way A_TA_type specifies the addressing method for A_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, A_AE 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 A_AE value only. If a remote server is distributed and implemented in several ECUs, then the remote server identifier shall be encoded with one A_AE value for each physical location of the remote server.

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 from this rule shall be a few cases when 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 a negative response message shall not be sent even if the server failed to complete the requested diagnostic service. These exception cases are described at the relevant subclauses within this document (e.g. see [7.7](#)).

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 A_PDU, application protocol data unit

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

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

Where:

- "Mtype, SA, TA, TA_type, RA, Length" 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;
- "Length" determines the number of bytes of A_Data.

7.3 A_PCI, application protocol control information

The A_PCI consists of two formats. The format is identified by the value of the first byte of the A_PDU parameter. For all service requests and for service responses with the first byte unequal to $7F_{16}$, the following definition shall apply:

A_PCI (
 SI
)

Where:

— "SI" is the parameter service identifier;

For service responses with first byte equal to $7F_{16}$, 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 data response messages as defined in service ReadDataByPeriodicIdentifier ($2A_{16}$, see [10.5](#)) no A_PCI is present in the application layer protocol data unit (A_PDU).

7.4 SI, service identifier

Type: 1 byte unsigned integer value

Range: 00_{16} to FF_{16} according to definitions in [Table 1](#).

Table 1 — Service identifier values

A_SI	Service type (bit 6)	Defined in
00_{16}	Not applicable	Reserved by this document
01_{16} to $0F_{16}$	ISO 15031-5 [15] /SAE J1979 [16] specified services	ISO 15031-5 [15] /SAE J1979 [16]
10_{16} to $3E_{16}$	Service requests specified in this document	This document
$3F_{16}$	Not applicable	Reserved by this document
40_{16}	Not applicable	Reserved by this document
41_{16} to $4F_{16}$	ISO 15031-5 [15] / SAE J1979 [16] positive service response	ISO 15031-5 [15] / SAE J1979 [16]
50_{16} to $7E_{16}$	Positive service responses specified in this document	This document
$7F_{16}$	Negative response service identifier	This document
80_{16} to 83_{16}	Not applicable	Reserved by this document
84_{16} to 88_{16}	Service requests	This document
89_{16} to $B9_{16}$	Not applicable	Reserved by this document
BA_{16} to BE_{16}	Service requests	Defined by system supplier
BF_{16} to $C2_{16}$	Not applicable	Reserved by this document
$C3_{16}$ to $C8_{16}$	Positive service responses specified in this document	This document

Table 1 (continued)

A_SI	Service type (bit 6)	Defined in
C9 ₁₆ to F9 ₁₆	Not applicable	Reserved by this document
FA ₁₆ to FE ₁₆	Positive service responses	Defined by system supplier
FF ₁₆	Not applicable	Reserved by this 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 byte value indicating the service type. All request messages have SI bit 6 = 0. All positive response messages have SI bit 6 = 1, except periodic data response messages of the ReadDataByPeriodicIdentifier (2A₁₆, see 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).

7.5 A_NR_SI, Negative response service identifier

Type: 1 byte unsigned integer value

Fixed value: 7F₁₆

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 a SI value in order to make A_Data coding and decoding easier.

7.6 Negative response/confirmation service primitive

Each diagnostic service has a negative response/negative confirmation message specified with message A_Data bytes according to Table 2. 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 that the negative response message corresponds to.

Table 2 — Negative response A_PDU

A_PDU parameter	Parameter Name	Cvt	Byte value	Mnemonic
SA	Source Address	M ^a	XXXX ₁₆	SA
TA	Target Address	M ^a	XXXX ₁₆	TA
TAtype	Target Address type	M ^a	XX ₁₆	TAT
RA	Remote Address (optional)	C	XXXX ₁₆	RA
A_Data.A_PCI.NR_SI	Negative Response SID	M ^a	7F ₁₆	SIDNR
A_Data.A_PCI.SI	<Service Name> Request SID	M ^a	XX ₁₆	SIDRQ
A_Data.Parameter 1	responseCode	M ^a	XX ₁₆	NRC_

^a M (Mandatory): In case the negative response A_PDU is issued then those A_PDU parameters shall be present. 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 [Clause A.1](#).

7.7 Server response implementation rules

7.7.1 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

Abbreviation	Description
suppressPosRspMsgIndicationBit	TRUE = server shall NOT send a positive response message (exception see Clause A.1 in definition of NRC 78 ₁₆) 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 of the client request message are supported by the server
At least 1	At least 1 data-parameter of the client request message shall be supported by the server
None	None of the requested data-parameter of the client request message is supported by the server
DataParam	Data parameter
SI	Service identifier
SF	SubFunction

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

NOTE — As required by the tables in the following subclauses, negative response messages with negative response codes of SNS (serviceNotSupported), SNSIAS (serviceNotSupportedInActiveSession), SFNS (SubFunctionNotSupported), SFNSIAS (SubFunctionNotSupportedInActiveSession), and ROR (requestOutOfRange) shall not be transmitted when functional addressing was used for the request message (exception see [Clause A.1](#) in definition of NRC 78₁₆).

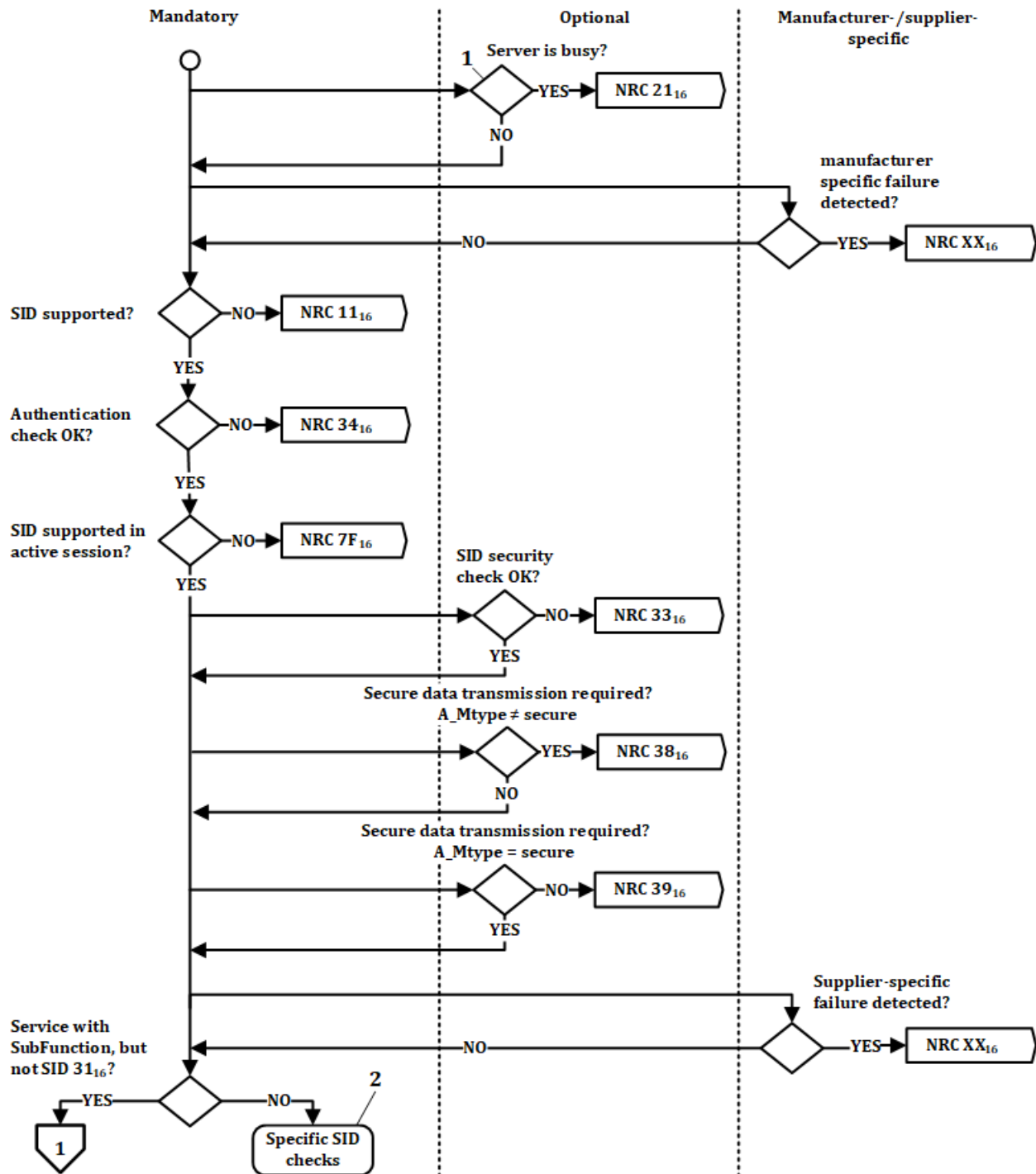
7.7.2 General server response behaviour

The general server response behaviour specified in this subclause is mandatory for all request messages. The validation steps start with the reception of the request message. The figure is divided into three subclauses:

- mandatory: to be evaluated by every request message;
- optional: could be optionally evaluated by every request message; and
- manufacturer/supplier specific: the procedure can be extended by additional manufacturer/supplier specific checks.

NOTE Depending on the choices implemented in all figures specifying NRC handling, a specific NRC is not guaranteed for all possible test pattern sequences.

Figure 5 depicts the general server response behaviour.



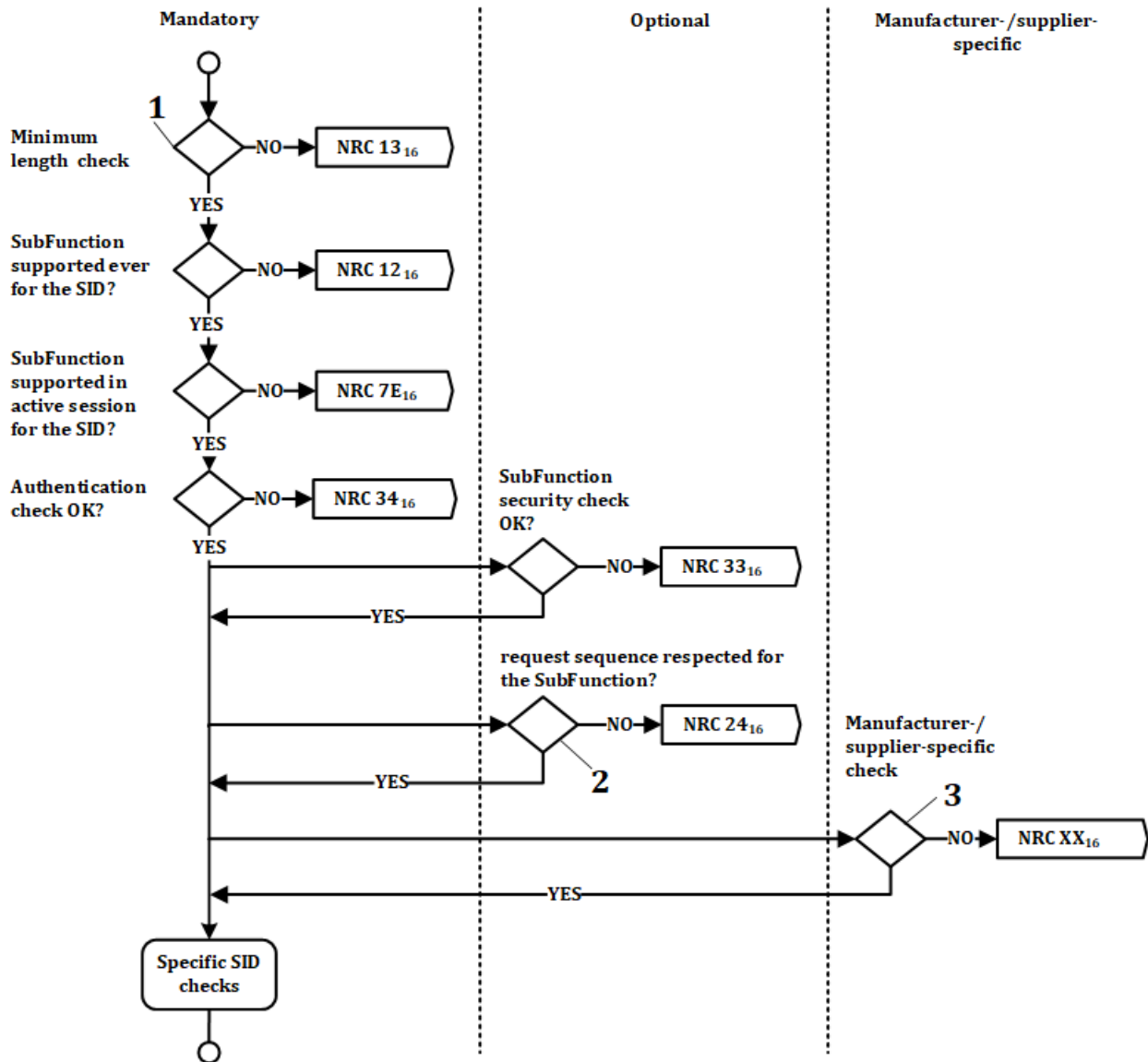
- 1 diagnostic request cannot be accepted because another diagnostic task is already requested and in progress by a different client
- 2 refer to the response behaviour (supported negative response codes) of each service

Figure 5 — General server response behaviour

7.7.3 Request message with SubFunction parameter and server response behaviour

7.7.3.1 General server response behaviour for request messages with SubFunction parameter

The general server response behaviour specified in this subclause is mandatory for all request messages with SubFunction parameter. A request message in the context of this subclause is defined as a service request message adhering to the formatting requirements defined in this document. [Figure 6](#) depicts the general server response behaviour for request messages with SubFunction parameter.



- 1 at least 2 (SID+SubFunction Parameter)
- 2 if SubFunction is subject to sequence check, e.g. LinkControl or SecurityAccess
- 3 refer to the response behaviour (supported negative response codes) of each service

Figure 6 — General server response behaviour for request messages with SubFunction parameter

7.7.3.2 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 SubFunction parameter in the physically addressed request message received from the client.

[Table 3](#) shows possible communication schemes with physical addressing.

Table 3 — Physically addressed request message with SubFunction parameter and server response behaviour

Server case #	Client request		Server capability			Server response		Comments to server response
	Addressing scheme	SF (sup-pressPosR-spMsgInd-Bit)	SI	SF	DataParam supported (only if applicable)	Message	NRC	
			supported					
a)	physical	FALSE (bit = 0)	YES	YES	At least 1	PosRsp	---	Server sends positive response
b)					At least 1	NegRsp	NRC= XX ₁₆	Server sends negative response because error occurred processing the data-parameters of the request message
c)								None
d)			NO	--	--		NRC= SNS or SNSIAS	Negative response with NRC 11 ₁₆ or NRC 7F ₁₆
e)			YES	NO	--		NRC= SFNS or SFNSIAS	Negative response with NRC 12 ₁₆ or NRC 7E ₁₆
f)		TRUE (bit = 1)			At least 1	NoRsp	---	Server does NOT send a response
g)			YES	YES	At least 1	NegRsp	NRC= XX ₁₆	Server sends negative response because error occurred reading the data-parameters of the request message
h)					None		NRC= ROOR	Negative response with NRC 31 ₁₆
i)			NO	--	--		NRC= SNS or SNSIAS	Negative response with NRC 11 ₁₆ or 7F ₁₆
j)			YES	NO	--		NRC=SFNS or SFNSIAS	Negative response with NRC 12 ₁₆ or 7E ₁₆

Description of server response cases on physically addressed client request messages with SubFunction:

- Server sends a positive response message because the service identifier and SubFunction parameter are supported of the client's request with indication for a response message.
- Server sends a negative response message (e.g. IML0IF: incorrectMessageLengthOrIncorrectFormat) because the service identifier and SubFunction parameter are supported of the client's request, but some other error appeared (e.g. wrong PDU length according to service identifier and SubFunction parameter in the request message) during processing of the SubFunction.
- Server sends a negative response message with the negative response code ROOR (requestOutOfRange) because the service identifier and SubFunction parameter are supported but none of the requested data-parameters are supported by the client's request message.
- Server sends a negative response message with the negative response code SNS (serviceNotSupported) or SNSIAS (serviceNotSupportedInActiveSession) because the service identifier is not supported of the client's request with indication for a response message.
- Server sends a negative response message with the negative response code SFNS (SubFunctionNotSupported) or SFNSIAS (SubFunctionNotSupportedInActiveSession) because the service identifier is supported and the SubFunction parameter is not supported for the data-parameters (if applicable) of the client's request with indication for a response message.

- f) Server sends no response message because the service identifier and SubFunction parameter are supported of 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.

- g) Same effect as in b) (i.e. a negative response message is sent) because the suppressPosRspMsgIndicationBit is ignored for any negative response that needs to be sent upon a physically addressed request messages.
- h) Same effect as in c) (i.e. 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 messages.
- i) Same effect as in d) (i.e. the negative response message is sent) because the suppressPosRspMsgIndicationBit is ignored for any negative response that needs to be sent upon a physically addressed request messages.
- j) Same effect as in e) (i.e. the negative response message is sent) because the suppressPosRspMsgIndicationBit is ignored for any negative response that needs to be sent upon a physically addressed request messages.

7.7.3.3 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 SubFunction parameter in the functionally addressed request message received from the client.

[Table 4](#) shows possible communication schemes with functional addressing.

Table 4 — Functionally addressed request message with SubFunction parameter and server response behaviour

Server case #	Client request		Server capability			Server response		Comments to server response
	Addressing scheme	SF (suppress-Pos-RspMsgInd-Bit)	SI	SF	DataParam supported (only if applicable)	Message	NRC	
a)	functional	FALSE (bit = 0)	YES	YES	At least 1	PosRsp	---	Server sends positive response
b)					At least 1	NegRsp	NRC= XX ₁₆	Server sends negative response because error occurred processing the data-parameters of the request message
c)					None		--	Server does NOT send a response
d)			NO	--	--	NoRsp	---	Server does NOT send a response
e)			YES	NO	--		---	Server does NOT send a response
f)		TRUE (bit = 1)	YES	YES	At least 1	NoRsp	---	Server does NOT send a response
g)					At least 1	NegRsp	NRC= XX ₁₆	Server sends negative response because error occurred reading the data-parameters of the request message
h)					None	NoRsp	--	Server does NOT send a response
i)			NO	--	--		---	Server does NOT send a response
j)			YES	NO	--		---	Server does NOT send a response

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

- a) Server sends a positive response message because the service identifier and SubFunction parameter are supported of the client's request with indication for a response message.
- b) Server sends a negative response message (e.g. IML0IF: incorrectMessageLengthOrIncorrectFormat) because the service identifier and SubFunction parameter are supported of the client's request, but some other error appeared (e.g. wrong PDU length according to service identifier and SubFunction parameter in the request message) during processing of the SubFunction.

- c) Server sends no response message because the negative response code ROOR (requestOutOfRange, which is identified by the server because the service identifier and SubFunction parameter are supported but a required data-parameter is not supported of the client's request) is always suppressed in case of a functionally addressed request message. The suppressPosRspMsgIndicationBit does not matter in such case.
- d) Server sends no response message because the negative response codes SNS (serviceNotSupported) and SNSIAS (serviceNotSupportedInActiveSession), which are identified by the server because the service identifier is not supported of the client's request, are always suppressed in case of a functionally addressed request message. The suppressPosRspMsgIndicationBit does not matter in such case.
- e) Server sends no response message because the negative response codes SFNS (SubFunctionNotSupported) and SFNSIAS (SubFunctionNotSupportedInActiveSession), which are identified by the server because the service identifier is supported and the SubFunction parameter is not supported for the data parameters (if applicable) of the client's request, are always suppressed in case of a functionally addressed request. The suppressPosRspMsgIndicationBit does not matter in such case.
- f) Server sends no response message because the service identifier and SubFunction parameter are supported of 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.

- g) Same effect as in b) (i.e. a negative response message is sent) because the suppressPosRspMsgIndicationBit is ignored for any negative response. This is also true in case the request message is functionally addressed.
- h) Same effect as in c) (i.e. no response message is sent) because the negative response code ROOR (requestOutOfRange, which is identified by the server because the service identifier and SubFunction parameter are supported but a required data-parameter is not supported of the client's request) is always suppressed in case of a functionally addressed request message. The suppressPosRspMsgIndicationBit does not matter in such case.
- i) Same effect as in d) (i.e. no response message is sent) because the negative response codes SNS (serviceNotSupported) and SNSIAS (serviceNotSupportedInActiveSession), which are identified by the server because the service identifier is not supported of the client's request, are always suppressed in case of a functionally addressed request message. The suppressPosRspMsgIndicationBit does not matter in such case.
- j) Same effect as in e) (i.e. no response message is sent) because the negative response codes SFNS (SubFunctionNotSupported) and SFNSIAS (SubFunctionNotSupportedInActiveSession), which are identified by the server because the service identifier is supported and the SubFunction parameter is not supported of the client's request, are always suppressed in case of a functionally addressed request message. The suppressPosRspMsgIndicationBit does not matter in such case.

7.7.4 Request message without SubFunction parameter and server response behaviour

7.7.4.1 General server response behaviour for request messages without SubFunction parameter

There is no general server response behaviour available for request messages without SubFunction parameter. A request message in the context of this subclause is defined as a service request message adhering to the formatting requirements defined in this document.

7.7.4.2 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 SubFunction parameter but a data-parameter in the physically addressed request message received from the client.

[Table 5](#) shows possible communication schemes with physical addressing.

Table 5 — Physically addressed request message without SubFunction parameter and server response behaviour

Server case #	Client request Addressing scheme	Server capability		Server response		Comments to server response
		SI	DataParam supported	Message	NRC	
a)	physical	YES	ALL	PosRsp	---	Server sends positive response
b)			At least 1		---	Server sends positive response
c)			At least 1	NegRsp	NRC=XX ₁₆	Server sends negative response because error occurred processing data-parameters of request message
d)			NONE		NRC=ROOR	Negative response with NRC 31 ₁₆
e)		NO	---		NRC=SNS or SNSIAS	Negative response with NRC 11 ₁₆ or NRC 7F ₁₆

Description of server response cases on physically addressed client request messages without SubFunction (data-parameter follows service identifier):

- Server sends a positive response message because the service identifier and all data-parameters are supported of the client's request message.
- Server sends a positive response message because the service identifier and at least one data-parameter is supported of the client's request message.
- Server sends a negative response message (e.g. IML0IF: incorrectMessageLengthOrIncorrectFormat) because the service identifier is supported and at least one data-parameter is supported of the client's request message, but some other error occurred (e.g. wrong length of the request message) during processing of the service.
- Server sends a negative response message with the negative response code ROOR (requestOutOfRange) because the service identifier is supported and none of the requested data-parameters are supported of the client's request message.
- Server sends a negative response message with the negative response code SNS (serviceNotSupported) or SNSIAS (serviceNotSupportedInActiveSession) because the service identifier is not supported of the client's request message.

7.7.4.3 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 SubFunction parameter but a data-parameter in the functionally addressed request message received from the client.

[Table 6](#) shows possible communication schemes with functional addressing.

Table 6 — Functionally addressed request message without SubFunction parameter and serve response behaviour

Server case #	Client request	Server capability		Server response		Comments to server response
		SI	DataParam	Message	NRC	
			supported			
a)	functional	YES	ALL	PosRsp	---	Server sends positive response
b)			At least 1		---	Server sends positive response
c)			At least 1	NegRsp	NRC= XX ₁₆	Server sends negative response because error occurred processing data-parameters of request message
d)			NONE	NoRsp	---	Server does NOT send a response
e)		NO	---		---	Server does NOT send a response

Description of server response cases on functionally addressed client request messages without SubFunction (data-parameter follows service identifier):

- Server sends a positive response message because the service identifier and all data-parameters are supported of the client's request message.
- Server sends a positive response message because the service identifier and at least one data-parameter is supported of the client's request message.
- 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 of the client's request message, but some other error occurred (e.g. wrong length of the request message) during processing of the service.
- Server sends no response message because the negative response code ROOR (requestOutOfRange; which would occur because the service identifier is supported, but none of the requested data-parameters is supported of the client's request) is always suppressed in case of a functionally addressed request.
- Server sends no response message because the negative response codes SNS (serviceNotSupported) and SNSIAS (serviceNotSupportedInActiveSession), which are identified by the server because the service identifier is not supported of the client's request, are always suppressed in case of a functionally addressed request.

7.7.5 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 */
    IF (message_length >= 2) THEN /* check minimum length of message with SubFunction */
      SWITCH (A_PDU.A_Data.A_Data.Parameter1 & 7F16)
        /* 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; /* pos. response message; set internal NRC
= 0016
            ELSE
              responseCode = IMLOIF; /* NRC 1316 : incorrectMessageLengthOrInvalidFormat
*/
            ENDIF
            BREAK;
          CASE SubFunction_01: /* test if SubFunction parameter value is supported */

```

```

:      /* prepare response message */
responseCode = positiveResponse; /* positive response message; set internal NRC
= 0016 */
:
CASE SubFunction_127: /* test if SubFunction parameter value is supported */
:      /* prepare response message */
responseCode = positiveResponse; /* positive response message; set internal NRC
= 0016 */
BREAK;
DEFAULT:
responseCode = SFNS; /* NRC 1216 : SubFunctionNotSupported */
}
ELSE
responseCode = IMLOIF; /* NRC 1316 : incorrectMessageLengthOrInvalidFormat */
ENDIF
suppressPosRspMsgIndicationBit = (A_PDU.A_Data.Parameter1 & 8016);
/* results in either 0016 or 8016 */
IF ( (suppressPosRspMsgIndicationBit) && (responseCode == positiveResponse) &&
("not yet a NRC 7816 response sent")) THEN /* test if pos. response is required and
if responseCode
is positive 0016 */
suppressResponse = TRUE; /* flag to NOT send a positive response message */
ELSE
suppressResponse = FALSE; /* flag to send the response message */
ENDIF
BREAK;
CASE Service_without_SubFunction: /* test if service without SubFunction is supported */
suppressResponse = FALSE; /* flag to send the response message */
IF (message_length == expected_message_length) THEN
IF (A_PDU.A_Data.Parameter1 == supported) THEN
/* test if data-parameter following the SID is supported*/
:      /* read data and prepare response message */
responseCode = positiveResponse; /* positive response message; set internal NRC =
0016 */
ELSE
responseCode = ROOR; /* NRC 3116 : requestOutOfRange */
ENDIF
ELSE
responseCode = IMLOIF; /* NRC 1316 : incorrectMessageLengthOrInvalidFormat */
ENDIF
BREAK;
DEFAULT:
responseCode = SNS; /* NRC 1116 : serviceNotSupported */
}
IF (A_PDU.TA_type == functional && ((responseCode == SNS) || (responseCode == SFNS) ||
(responseCode == SNSIAS) ||
(responseCode == SFNSIAS) || (responseCode == ROOR)) &&
("not yet a NRC 7816 response sent")) 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, and the negative response message with NRC=RCRRP (requestCorrectlyReceivedResponsePending) needs to be sent, then the final negative response message using NRC=SNS (serviceNotSupported), NRC=SNSIAS (serviceNotSupportedInActiveSession), NRC=SFNS (SubFunctionNotSupported), NRC=SFNSIAS (SubFunctionNotSupportedInActiveSession) or NRC=ROOR (requestOutOfRange) shall also be sent if it is the result of the PDU analysis of the received request message.

7.7.6 Multiple concurrent request messages with physical and functional addressing

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

The following exceptions shall be treated separately:

- a) 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 shall be processed by bypass logic. It is up to the server to make sure that this specific message cannot "block" the server's application layer and that an immediately following addressed message can be processed.
- b) If a server supports multiple clients and executes a service in non-Default session when a client with higher priority sends a request then any active service shall be aborted, the default session shall be started, and the diagnostic service with higher priority shall be processed. This requirement does not apply if the programming session is active.

See [Annex I](#) for further information of how multiple clients can be handled.

8 Service description conventions

8.1 Service description

This subclause defines how each diagnostic service is described in this document. It specifies the general service description format of each diagnostic service.

This subclause 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 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:

[Table 7](#) specifies the A_PDU parameter conventions.

Table 7 — 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. SubFunction/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.
The "<Service Name> Request SID" marked as 'M' (Mandatory), shall not imply that this service shall be supported by the server. The 'M' only indicates the mandatory presence of this parameter in the request A_PDU in case the server supports the service.		

8.2 Request message

8.2.1 Request message definition

This subclause includes one or multiple tables, which define the A_PDU (Application layer protocol data unit, see [Clause 7](#)) parameters for the service request/indication. There might be a separate table for each SubFunction parameter (\$Level) in case the request messages of the different SubFunction parameters (\$Level) differ in the structure of the A_Data parameters and cannot be specified clearly in single table.

[Table 8](#) specifies the request A_PDU definition with SubFunction

Table 8 — Request A_PDU definition with SubFunction

A_PDU parameter	Parameter Name	Cvt	Byte value	Mnemonic
MType	Message type	M	xx	MT
SA	Source Address	M	xxxx	SA
TA	Target Address	M	xxxx	TA
TAType	Target Address type	M	xx	TAT
RA	Remote Address	C ₁ ^a	xxxx	RA
A_Data.A_PCI.SI	<Service Name> Request SID	M	xx	SIDRQ
A_Data.Parameter 1	SubFunction = [parameter]	S	xx	LEV_PARAM
A_Data.Parameter 2	data-parameter#1	U	xx	DP_...#1
:	:	:	:	:
A_Data.Parameter k	data-parameter#k-1	U	xx	DP_...#k-1
Length	Length of A_Data	M	xxxxxxxx	LGT
^a C ₁ : The RA (Remote Address) PDU parameter is only present in case of remote addressing.				

[Table 9](#) specifies the request A_PDU definition without SubFunction.

Table 9 — Request A_PDU definition without SubFunction

A_PDU parameter	Parameter Name	Cvt	Byte value	Mnemonic
MType	Message type	M	xx	MT
SA	Source Address	M	xxxx	SA
TA	Target Address	M	xxxx	TA
TAType	Target Address type	M	xx	TAT
RA	Remote Address	C ₁ ^a	xxxx	RA
A_Data.A_PCI.SI	<Service Name> Request SID	M	xx	SIDRQ
A_Data.Parameter 1	data-parameter#1	U	xx	DP_...#1
:	:	:	:	:
A_Data.Parameter k	data-parameter#k	U	xx	DP_...#k
Length	Length of A_Data	M	xxxxxxxx	LGT
^a C ₁ : The RA (Remote Address) PDU parameter is only present in case of remote addressing.				

In all requests/indications the addressing information MType, TA, SA, TAType and Length is mandatory. The addressing information RA is optionally to 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 SubFunction parameter \$Level (LEV_) definition

This subclause specifies the SubFunction \$Levels (LEV_) parameter(s) defined for the request/indication of the service <Service Name>.

This subclause does not contain any definition in case the described service does not use a SubFunction parameter value and does not utilize the suppressPosRspMsgIndicationBit (this implicitly indicates that a response is required).

The SubFunction parameter byte is divided into two parts (on bit-level) as defined in [Table 10](#).

Table 10 — SubFunction 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 7.7. Even if a positive response is not required (i.e. SPRMIB = true), the execution of the service shall be completely passed to keep the implementation consistent regardless of SPRMIB value. suppressPosRspMsgIndicationBit values of both '0' and '1' shall be supported for all SubFunction parameter values (i.e. bits 6-0 of the SubFunction structure) supported by the server for any given service.
6 to 0	SubFunction parameter value The bits 0-6 of the SubFunction parameter contain the SubFunction parameter value of the service (00 ₁₆ to 7F ₁₆).

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

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

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

NOTE If SPRMIB is TRUE for responses with a big amount of data, where paged-buffer-handling needs to be used, this can result in a situation where the transmission of the first batch of data could be started still within the response timing window, but the termination of the service execution is beyond the limits of the response timing window. If the response is suppressed in this case, there is no way to inform the client about the delay, but the server is still busy and not yet ready to receive another request.

For the client it is recommended not to ask for a big amount of data and set SPRMIB in the same request (e.g. SID 19₁₆ SF 0A₁₆), as this would defeat the purpose of SPRMIB. For the server implementation it is recommended to send NRC 78₁₆ (RCRRP) and subsequently also send the positive response, in case paged-buffer-handling is used while SPRMIB is TRUE.

[Table 11](#) specifies the request message SubFunction parameter definition.

Table 11 — Request message SubFunction parameter definition

Bits 6 to 0	Description	Cvt	Mnemonic
xx	SubFunction#1 description of SubFunction parameter#1	M/U	SUBFUNC1
:	:	:	:
xx	SubFunction#m description of SubFunction parameter#m	M/U	SUBFUNCm

The convention (Cvt) column in the [Table 11](#) above shall be interpreted as defined in [Table 12](#).

Table 12 — SubFunction parameter conventions

Type	Name	Description
M	Mandatory	The SubFunction parameter shall be supported by the server in case the service is supported.
U	User option	The SubFunction parameter may or may not be supported by the server, depending on the usage of the service.

The complete SubFunction parameter byte value is calculated based on the value of the suppressPosRspMsgIndicationBit and the SubFunction parameter value chosen.

[Table 13](#) specifies the calculation of the SubFunction byte value.

Table 13 — Calculation of the SubFunction byte value

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SuppressPosRspMsg-Indication-Bit	SubFunction parameter value as specified in the SubFunction parameter value table of the service.						
resulting SubFunction parameter byte value (bit 7 to 0)							

8.2.3 Request message data-parameter definition

This subclause specifies the data-parameter(s) \$DataParam (DP_) for the request/indication of the service <Service Name>. This subclause does not contain any definition in case 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 specific annex is dependent upon the service. The annexes also specify whether a data-parameter shall be supported or is user optional to be supported in case the server supports the service.

[Table 14](#) specifies the request message data-parameters.

Table 14 — 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 subclause specifies the A_PDU parameters for the service response/confirmation (see [7.2](#) for a detailed description of the application layer protocol data unit A_PDU). There might be a separate table for each SubFunction parameter \$Level when the response messages of the different SubFunction 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. In case a diagnostic service requires a different handling (e.g. ECUReset service) then the appropriate description of when to send the positive response message can be found in the service description of the diagnostic service.

[Table 15](#) specifies the positive response A_PDU.

Table 15 — Positive response A_PDU

A_PDU parameter	Parameter Name	Cvt	Byte value	Mnemonic
SA	Source Address	M	xxxx	SA
TA	Target Address	M	xxxx	TA
TAtype	Target Address type	M	xx	TAT
RA	Remote Address	C ₁ ^a	xxxx	RA
A_Data.A_PCI.SI	<Service Name> Response SID	S	xx	SIDPR
A_Data.Parameter 1	data-parameter#1	U	xx	DP_...#1
:	:	:	:	:
A_Data.Parameter k	data-parameter#k	U	xx	DP_...#k

^a C₁: The RA (Remote Address) PDU parameter is only present in case of remote addressing.

In all responses/confirmations the addressing information TA, SA, and TAtype is mandatory. The addressing information RA is optionally to be present.

NOTE The addressing information is shown in the table above for definition purpose. Further service response/confirmation 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 specifies the data-parameter(s) for the response/confirmation of the service <Service Name>. This subclause does not contain any definition in case 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 specific annex is dependent upon the service. The annexes also specify whether a data-parameter shall be supported or is user optional to be supported in case the server supports the service.

[Table 16](#) specifies the response data-parameters.

Table 16 — Response data-parameter definition

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

8.4 Supported negative response codes (NRC_)

This subclause specifies the negative response codes that shall be implemented for this service. The circumstances under which each response code would occur are documented in a table as given below. The definition of the negative response message can be found in [7.6](#). The server shall use the negative response A_PDU for the indication of an identified error condition.

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

[Table 17](#) specifies the supported negative response codes.

Table 17 — Supported negative response codes

NRC	Description	Mnemonic
XX ₁₆	NegativeResponseCode#1 1. condition#1 : m. condition#m	NRC_
:	:	NRC_
XX ₁₆	NegativeResponseCode#n 1. condition#1 : k. condition#k	NRC_

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 18](#) specifies the request message flow example.

Table 18 — Request message flow example

Message direction	client → server		
Message type	Request		
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1 (A_PCI)	<Service Name> Request SID	XX ₁₆	SIDRQ
#2	SubFunction/data-parameter#1	XX ₁₆	LEV_/DP_
:	:	XX ₁₆	DP_
#n	data-parameter#m	XX ₁₆	DP_

[Table 19](#) specifies the positive response message flow example.

Table 19 — Positive response message flow example

Message direction	server → client		
Message type	Response		
A_Data	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1 (A_PCI)	<Service Name> Response SID	XX ₁₆	SIDPR
#2	data-parameter#1	XX ₁₆	DP_
:	:	:	:
#n	data-parameter#n-1	XX ₁₆	DP_

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

[Table 20](#) shows a message flow example for a negative response message.

Table 20 — Negative response message flow example

Message direction	server → client		
Message type	Response		
A_Data	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1 (A_PCI.NR_SI)	Negative Response SID	7F ₁₆	SIDRSIDNRQ
#2 (A_PCI.SI)	<Service Name> Request SID	XX ₁₆	SIDRQ
#3	responseCode	XX ₁₆	NRC_

9 Diagnostic and communication management functional unit

9.1 Overview

[Table 21](#) specifies the diagnostic and communication management functional unit.

Table 21 — 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 controls the setting of communication parameters in the server (e.g. communication baudrate).
TesterPresent	The client indicates to the server(s) that it is still present.
Authentication	The client requests to prove its identity, allowing it to access data and/or diagnostic services restricted access.
ControlDTCSetting	The client controls the setting of DTCs in the server.
ResponseOnEvent	The client requests to set up and/or control an event mechanism in the server.
LinkControl	The client requests control of the communication baudrate.

9.2 DiagnosticSessionControl (10₁₆) 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). 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 user of this document shall define the exact set of services and/or functionality enabled in each diagnostic session.

There shall always be exactly one diagnostic session active in a server. There can be more than one client communicating with the server, but the server has just one session that is used for all clients. 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 operation 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 7](#) that 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

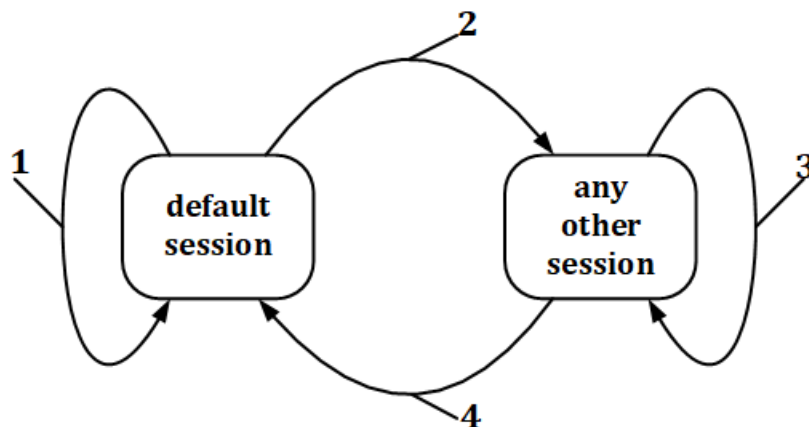
may require that the new session shall be entered before the positive response is sent, while maintaining the old protocol timings for sending the response. If the server is not able to start the requested new diagnostic session, then it shall respond with a DiagnosticSessionControl negative response message and the current session shall continue (see diagnosticSession parameter definitions for further information on how the server and client shall behave). 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 this document.

To start a new diagnostic session a server may request that certain conditions be fulfilled. All such conditions are user defined. Examples of such conditions are:

- 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 F4₁₆ may start the extendedDiagnosticSession).
- Certain safety conditions may need to be satisfied (e.g. vehicle shall not be moving or engine shall not be running). Transitions to, for example programmingSession may lead to loss of normal functionality and therefore some ECUs may require the vehicle to be in a safe state.

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 7](#) provides an overview about the diagnostic session transition and what the server shall do when it transitions to another session.



- 1 defaultSession: 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
- 2 other transition to any defaultSession: When the server transitions from the defaultSession to any other session than the defaultSession then the server shall only pause the events (similar to stopResponseOnEvent in the time period since the non-defaultSession is active) that have been configured in the server via the ResponseOnEvent (86₁₆) service during the defaultSession

- 3 same or other session: 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:
- a) a started ResponseOnEvent logic shall be stopped and all events that have been configured in the server shall be cleared
 - b) security shall be relocked. The locking of security access shall reset any active diagnostic functionality that was dependent on security access to be unlocked (e.g. active inputOutputControl of a DID)
 - c) all other active diagnostic functionality that is supported in the new session and is not dependent upon security access shall be maintained. For example, any configured periodic scheduler shall remain active when transitioning from one non-defaultSession to another or the same non-DefaultSession and the states of the CommunicationControl and ControlDTCSetting services shall not be affected, which means that normal communication shall remain disabled when it is disabled at the point in time the session is switched
- 4 transition to defaultSession: When the server transitions from any diagnostic session other than the default session to the defaultSession then the server shall resume each event that has been configured in the server via the ResponseOnEvent (86₁₆) service and event window is still valid. Further a locked security level shall be activated in the server. Any other active diagnostic functionality that is not supported in the defaultSession shall be terminated. For example, any configured periodic scheduler or output control shall be disabled and the states of the CommunicationControl and ControlDTCSetting services shall be reset, which means 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

Figure 7 — Server diagnostic session state diagram

[Table 22](#) specifies 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 shall be kept active by the client.

Table 22 — Services allowed during default and non-default diagnostic session

Service	defaultSession	non-defaultSession
DiagnosticSessionControl – 10 ₁₆	x	x
ECUReset – 11 ₁₆	x	x
SecurityAccess – 27 ₁₆	not applicable	x
CommunicationControl – 28 ₁₆	not applicable	x
TesterPresent – 3E ₁₆	x	x
Authentication – 29 ₁₆	x	x
SecuredDataTransmission – 84 ₁₆	not applicable	x
ControlDTCSetting – 85 ₁₆	not applicable	x
ResponseOnEvent – 86 ₁₆	x ^a	x
LinkControl – 87 ₁₆	not applicable	x
ReadDataByIdentifier – 22 ₁₆	x ^b	x
ReadMemoryByAddress – 23 ₁₆	x ^c	x
^a It is implementation specific whether the ResponseOnEvent service is also allowed during the defaultSession. ^b Secured dataIdentifiers 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 dataIdentifier 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 requires to be stopped actively by the client also requires a non-default session.		

Table 22 (continued)

Service	defaultSession	non-defaultSession
ReadScalingDataByIdentifier – 24 ₁₆	x ^b	x
ReadDataByPeriodicIdentifier – 2A ₁₆	not applicable	x
DynamicallyDefineDataIdentifier – 2C ₁₆	x ^d	x
WriteDataByIdentifier – 2E ₁₆	x ^b	x
WriteMemoryByAddress – 3D ₁₆	x ^c	x
ClearDiagnosticInformation – 14 ₁₆	x	x
ReadDTCInformation – 19 ₁₆	x	x
InputOutputControlByIdentifier – 2F ₁₆	not applicable	x
RoutineControl – 31 ₁₆	x ^e	x
RequestDownload – 34 ₁₆	not applicable	x
RequestUpload – 35 ₁₆	not applicable	x
TransferData – 36 ₁₆	not applicable	x
RequestTransferExit – 37 ₁₆	not applicable	X
RequestFileTransfer – 38 ₁₆	not applicable	X
^a It is implementation specific whether the ResponseOnEvent service is also allowed during the defaultSession. ^b Secured dataIdentifiers 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 dataIdentifier 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 requires to be stopped actively by the client also requires a non-default session.		

NOTE — The server and the client shall meet the request and response message behaviour as specified in [7.7](#).

9.2.2 Request message

9.2.2.1 Request message definition

[Table 23](#) specifies the request message.

Table 23 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	DiagnosticSessionControl Request SID	M	10 ₁₆	DSC
#2	SubFunction = [diagnosticSessionType]	M	00 ₁₆ to FF ₁₆	LEV_DS_

9.2.2.2 Request message SubFunction parameter \$Level (LEV_) definition

The SubFunction 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 in [Table 24](#).

The following SubFunction values are specified (suppressPosRspMsgIndicationBit (bit 7) not shown).

Table 24 — Request message SubFunction parameter definition

Bit 6-0	Description	Cvt	Mnemonic
00 ₁₆	Reserved by this document	M	ISOSAERESRVD
	This value is reserved by this document.		
01 ₁₆	defaultSession	M	DS
	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 the server diagnostic session state diagram given above):		
	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.		
	In case 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.		
02 ₁₆	ProgrammingSession	U	PRGS
	This diagnosticSession enables all diagnostic services required to support the memory programming of a server.		
	It is vehicle-manufacturer specific whether the positive response is sent prior or after the ECU has switched to/from programmingSession.		
	In case the server runs the programmingSession in the boot software, the programmingSession shall only be left via an ECUReset (11 ₁₆) service initiated by the client, a DiagnosticSessionControl (10 ₁₆) service with sessionType equal to defaultSession, or a session layer timeout in the server.		
	In case the server runs in the boot software when it receives the DiagnosticSessionControl (10 ₁₆) 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. This document 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 ECU reset, etc.).		
03 ₁₆	extendedDiagnosticSession	U	EXTDS
	This diagnosticSession can be used to enable all diagnostic services required to support the adjustment of functions like "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 (e.g. refer to timed services in Table 22).		
04 ₁₆	safetySystemDiagnosticSession	U	SSDS
	This diagnosticSession enables all diagnostic services required to support safety system related functions (e.g. airbag deployment).		
05 ₁₆ to 3F ₁₆	Reserved by this document	M	ISOSAERESRVD
	This value is reserved by this document for future definition.		
40 ₁₆ to 5F ₁₆	vehicleManufacturerSpecific	U	VMS
	This range of values is reserved for vehicle manufacturer specific use.		

Table 24 (continued)

Bit 6-0	Description	Cvt	Mnemonic
60 ₁₆ to 7E ₁₆	systemSupplierSpecific	U	SSS
	This range of values is reserved for system supplier specific use.		
7F ₁₆	Reserved by this document	M	ISOSAERESRVD
	This value is reserved by this document for future definition.		

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 25](#) specifies the positive response message definition.

Table 25 — Positive response message definition

A_Data byte	Parameter Name		Cvt	Byte value	Mnemonic
#1	DiagnosticSessionControl Response SID		M	50 ₁₆	DSCPR
#2	SubFunction = [diagnosticSessionType]		M	00 ₁₆ to FF ₁₆	LEV_DS_
	sessionParameterRecord[]#1 = [				SPREC_
#3		data#1	M	00 ₁₆ to FF ₁₆	DATA_1
:		:	:	:	:
#6		data#4]	M	00 ₁₆ to FF ₁₆	DATA_m

9.2.3.2 Positive response message data-parameter definition

[Table 26](#) specifies the response message data-parameter definition.

Table 26 — Response message data-parameter definition

Definition
diagnosticSessionType This parameter is an echo of bits 6 to 0 of the SubFunction parameter from the request message.
sessionParameterRecord This parameter record contains session specific parameter values reported by the server. The content of the sessionParameterRecord is defined in Table 27 and Table 28 .

[Table 27](#) and [Table 28](#) define the structure of the response message data-parameter sessionParameterRecord as applicable for the implementation of this service on supported data links.

Table 27 — sessionParameterRecord definition

Byte pos. in record	Description		Cvt	Byte value	Mnemonic
	sessionParameterRecord[] = [				SPREC_
#1		t _{L7_P2_Server_Max} (high byte)	M	00 ₁₆ to FF ₁₆	P2SMH
#2		t _{L7_P2_Server_Max} (low byte)	M	00 ₁₆ to FF ₁₆	P2SML
#3		t _{L7_P2*_Server_Max} (high byte)	M	00 ₁₆ to FF ₁₆	P2ESMH
#4		t _{L7_P2*_Server_Max} (low byte)]	M	00 ₁₆ to FF ₁₆	P2ESML

Table 28 — sessionParameterRecord content definition

Parameter	Description	# of bytes	Resolution	minimum value	maximum value
$t_{L7_P2_Server_Max}$	Default $t_{L7_P2_Server_Max}$ timing supported by the server for the activated diagnostic session.	2	1 ms	0 ms	65 535 ms
$t_{L7_P2*_Server_Max}$	Enhanced (NRC 78 ₁₆) $t_{L7_P2_Server_Max}$ supported by the server for the activated diagnostic session.	2	10 ms	0 ms	655 350 ms

Refer to ISO 14229-2 for further details on $t_{L7_P2_Server_Max}$ and $t_{L7_P2*_Server_Max}$.

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 would occur are documented in [Table 29](#). The listed negative responses shall be used if the error scenario applies to the server.

Table 29 — Supported negative response codes

NRC	Description	Mnemonic
12 ₁₆	SubFunctionNotSupported This NRC shall be sent if the SubFunction parameter is not supported.	SFNS
13 ₁₆	incorrectMessageLengthOrInvalidFormat This NRC shall be sent if the length of the message is wrong.	IMLOIF
22 ₁₆	conditionsNotCorrect This NRC shall be returned if the criteria for the request DiagnosticSessionControl are not met.	CNC

9.2.5 Message flow example(s) DiagnosticSessionControl – Start programmingSession

This message flow shows how to enable the diagnostic session "programmingSession" in a server. The client requests to have a response message by setting the suppressPosRspMsgIndicationBit (bit 7 of the SubFunction parameter) to "FALSE" ('0').

For the given example it is assumed that the $t_{L7_P2_Server_Max}$ is equal to 50 ms and the $t_{L7_P2*_Server_Max}$ is equal to 5 000 ms.

[Table 30](#) specifies the DiagnosticSessionControl request message flow example #1.

Table 30 — DiagnosticSessionControl request message flow example

Message direction	client → server		
Message type	Request		
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	DiagnosticSessionControl Request SID	10 ₁₆	DSC
#2	diagnosticSessionType = programmingSession, suppressPosRspMsgIndicationBit = FALSE	02 ₁₆	DS_ECUPRGS

[Table 31](#) specifies the DiagnosticSessionControl positive response message flow example.