INTERNATIONAL STANDARD

ISO 14229-1

Third edition
2020-02

Road vehicles — Unified diagnostic services (UDS) —

Part 1: Application layer

Véhicules routiers — Services de diagnostic unifiés (SDU) —

Partie 1: Couches application

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

Reference number
ISO 14229-1:2020(E)

© ISO 2020

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

COPYRIGHT PROTECTED DOCUMENT

© ISO 2020

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
Fax: +41 22 749 09 47
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

© ISO 2020 – All rights reserved

Contents

Page

Foreword ix
Introduction x
1 Scope 1
2 Normative references 1
3 Terms and definitions 2
4 Symbols and abbreviated terms 5
5 Conventions 5
6 Document overview 6
7 Application layer services 7
7.1 General 7
7.2 Format description of application layer services 9
7.3 Format description of service primitives 9
7.3.1 General definition 9
7.3.2 Service request and service indication primitives 10
7.3.3 Service response and service confirm primitives 11
7.3.4 Service request-confirm and service response-confirm primitives 11
7.4 Service data unit specification 12
7.4.1 Mandatory parameters 12
7.4.2 Vehicle system requirements 14
7.4.3 Optional parameters - A_AE, application layer remote address 15
8 Application layer protocol 15
8.1 General definition 15
8.2 A_PDU, application protocol data unit 16
8.3 A_PCI, application protocol control information 16
8.4 SI, service identifier 17
8.5 A_NR_SI, Negative response service identifier 17
8.6 Negative response/confirmation service primitive 18
8.7 Server response implementation rules 18
8.7.1 General definitions 18
8.7.2 General server response behaviour 19
8.7.3 Request message with SubFunction parameter and server response behaviour 21
8.7.4 Request message without SubFunction parameter and server response behaviour 25
8.7.5 Pseudo code example of server response behaviour 27
8.7.6 Multiple concurrent request messages with physical and functional addressing 29
9 Service description conventions 29
9.1 Service description 29
9.2 Request message 30
9.2.1 Request message definition 30
9.2.2 Request message SubFunction parameter $Level (LEV_) definition 31
9.2.3 Request message data-parameter definition 33
9.3 Positive response message 33
9.3.1 Positive response message definition 33
9.3.2 Positive response message data-parameter definition 34

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

9.4 Supported negative response codes (NRC_) 34
9.5 Message flow examples 35

10 Diagnostic and communication management functional unit 36
10.1 Overview 36
10.2 DiagnosticSessionControl (10_{16}) service 36
10.2.1 Service description 36
10.2.2 Request message 40
10.2.3 Positive response message 41
10.2.4 Supported negative response codes (NRC_) 42
10.2.5 Message flow example(s) DiagnosticSessionControl – Start programmingSession 43
10.3 ECUReset (11_{16}) service 43
10.3.1 Service description 43
10.3.2 Request message 44
10.3.3 Positive response message 45
10.3.4 Supported negative response codes (NRC_) 46
10.3.5 Message flow example ECUReset 47
10.4 SecurityAccess (27_{16}) service 47
10.4.1 Service description 47
10.4.2 Request message 49
10.4.3 Positive response message 51
10.4.4 Supported negative response codes (NRC_) 51
10.4.5 Message flow example(s) SecurityAccess 52
10.5 CommunicationControl (28_{16}) service 54
10.5.1 Service description 54
10.5.2 Request message 54
10.5.3 Positive response message 56
10.5.4 Supported negative response codes (NRC_) 56
10.5.5 Message flow example CommunicationControl (disable transmission of network management messages) 57
10.5.6 Message flow example CommunicationControl (switch a remote network into the diagnostic-only scheduling mode where the node with address $000A_{16}$ is connected to) 57
10.5.7 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) 58
10.6 Authentication (29_{16}) service 59
10.6.1 Service overview 59
10.6.2 Authentication with PKI Certificate Exchange (APCE) 60
10.6.3 Authentication with Challenge-Response (ACR) 65
10.6.4 Common requirements 69
10.6.5 Request message 71
10.6.6 Positive response message 78
10.6.7 Supported negative response codes (NRC_) 85
10.6.8 Message flow example(s) Authentication 86
10.7 TesterPresent ($3E_{16}$) service 108
10.7.1 Service description 108
10.7.2 Request message 108
10.7.3 Positive response message 108
10.7.4 Supported negative response codes (NRC_) 109
10.7.5 Message flow example(s) TesterPresent 109
10.8 ControlDTCSetting (85_{16}) service 110
10.8.1 Service description 110
10.8.2 Request message 111

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

10.8.3 Positive response message ... 112
10.8.4 Supported negative response codes (NRC_) ... 112
10.8.5 Message flow example(s) ControlDTCSetting ... 113
10.9 ResponseOnEvent (86_{16}) service ... 114
10.9.1 Service description ... 114
10.9.2 Request message ... 121
10.9.3 Positive response message ... 127
10.9.4 Supported negative response codes (NRC_) ... 130
10.9.5 Message flow example(s) ResponseOnEvent ... 131
10.10 LinkControl (87_{16}) service ... 146
10.10.1 Service description ... 146
10.10.2 Request message ... 147
10.10.3 Positive response message ... 149
10.10.4 Supported negative response codes (NRC_) ... 149
10.10.5 Message flow example(s) LinkControl ... 150

11 Data transmission functional unit ... 152
11.1 Overview ... 152
11.2 ReadDataByIdentifier (22_{16}) service ... 153
11.2.1 Service description ... 153
11.2.2 Request message ... 153
11.2.3 Positive response message ... 154
11.2.4 Supported negative response codes (NRC_) ... 155
11.2.5 Message flow example ReadDataByIdentifier ... 157
11.3 ReadMemoryByAddress (23_{16}) service ... 159
11.3.1 Service description ... 159
11.3.2 Request message ... 159
11.3.3 Positive response message ... 161
11.3.4 Supported negative response codes (NRC_) ... 161
11.3.5 Message flow example ReadMemoryByAddress ... 163
11.4 ReadScalingDataByIdentifier (24_{16}) service ... 166
11.4.1 Service description ... 166
11.4.2 Request message ... 166
11.4.3 Positive response message ... 166
11.4.4 Supported negative response codes (NRC_) ... 167
11.4.5 Message flow example ReadScalingDataByIdentifier ... 169
11.5 ReadDataByPeriodicIdentifier ($2A_{16}$) service ... 172
11.5.1 Service description ... 172
11.5.2 Request message ... 176
11.5.3 Positive response message ... 176
11.5.4 Supported negative response codes (NRC_) ... 177
11.5.5 Message flow example ReadDataByPeriodicIdentifier ... 180
11.6 DynamicallyDefineDataIdentifier ($2C_{16}$) service ... 191
11.6.1 Service description ... 191
11.6.2 Request message ... 192
11.6.3 Positive response message ... 195
11.6.4 Supported negative response codes (NRC_) ... 196
11.6.5 Message flow examples DynamicallyDefineDataIdentifier ... 197
11.7 WriteDataByIdentifier ($2E_{16}$) service ... 212
11.7.1 Service description ... 212
11.7.2 Request message ... 212
11.7.3 Positive response message ... 213
11.7.4 Supported negative response codes (NRC_) ... 214
11.7.5 Message flow example WriteDataByIdentifier ... 215

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

11.8 WriteMemoryByAddress ($3D_{16}$) service 216
11.8.1 Service description 216
11.8.2 Request message 217
11.8.3 Positive response message 218
11.8.4 Supported negative response codes (NRC_) 219
11.8.5 Message flow example WriteMemoryByAddress 221

12 Stored data transmission functional unit 223
12.1 Overview 223
12.2 ClearDiagnosticInformation (14_{16}) service 223
12.2.1 Service description 223
12.2.2 Request message 224
12.2.3 Positive response message 225
12.2.4 Supported negative response codes (NRC_) 225
12.2.5 Message flow example ClearDiagnosticInformation 226
12.3 ReadDTCInformation (19_{16}) service 227
12.3.1 Service description 227
12.3.2 Request message 238
12.3.3 Positive response message 249
12.3.4 Supported negative response codes (NRC_) 263
12.3.5 Message flow examples – ReadDTCInformation 264

13 InputOutput control functional unit 297
13.1 Overview 297
13.2 InputOutputControlByIdentifier ($2F_{16}$) service 297
13.2.1 Service description 297
13.2.2 Request message 298
13.2.3 Positive response message 299
13.2.4 Supported negative response codes (NRC_) 300
13.2.5 Message flow example(s) InputOutputControlByIdentifier 301

14 Routine functional unit 310
14.1 Overview 310
14.2 RoutineControl (31_{16}) service 311
14.2.1 Service description 311
14.2.2 Request message 312
14.2.3 Positive response message 314
14.2.4 Supported negative response codes (NRC_) 315
14.2.5 Message flow example(s) RoutineControl 317

15 Upload download functional unit 321
15.1 Overview 321
15.2 RequestDownload (34_{16}) service 321
15.2.1 Service description 321
15.2.2 Request message 322
15.2.3 Positive response message 323
15.2.4 Supported negative response codes (NRC_) 324
15.2.5 Message flow example(s) RequestDownload 325
15.3 RequestUpload (35_{16}) service 325
15.3.1 Service description 325
15.3.2 Request message 326
15.3.3 Positive response message 327
15.3.4 Supported negative response codes (NRC_) 328
15.3.5 Message flow example(s) RequestUpload 329
15.4 TransferData (36_{16}) service 330
15.4.1 Service description 330

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

15.4.2 Request message ... 330
15.4.3 Positive response message ... 331
15.4.4 Supported negative response codes (NRC_) ... 332
15.4.5 Message flow example(s) TransferData ... 334
15.5 RequestTransferExit (37_{16}) service ... 334
15.5.1 Service description ... 334
15.5.2 Request message ... 335
15.5.3 Positive response message ... 335
15.5.4 Supported negative response codes (NRC_) ... 336
15.5.5 Message flow example(s) for downloading/uploading data ... 337
15.6 RequestFileTransfer (38_{16}) service ... 344
15.6.1 Service description ... 344
15.6.2 Request message ... 344
15.6.3 Positive response message ... 346
15.6.4 Supported negative response codes (NRC_) ... 348
15.6.5 Message flow example(s) RequestFileTransfer ... 350

16 Security sub-layer definition ... 353
16.1 General ... 353
16.1.1 Purpose ... 353
16.1.2 Security sub-layer description ... 353
16.1.3 Security sub-layer access ... 354
16.1.4 General server response behaviour ... 356
16.2 SecuredDataTransmission (84_{16}) service ... 358
16.2.1 Service description ... 358
16.2.2 Request message ... 358
16.2.3 Positive response message for successful internal message ... 360
16.2.4 Supported negative response codes (NRC_) ... 362
16.2.5 Message flow example SecuredDataTransmission ... 363

17 Non-volatile server memory programming process ... 366
17.1 General information ... 366
17.2 Detailed programming sequence ... 370
17.2.1 Programming phase #1 — Download of application software and/or application data ... 370
17.3 Server reprogramming requirements ... 379
17.3.1 Requirements for servers to support programming ... 379
17.3.2 Software, data identification and fingerprints ... 382
17.3.3 Server routine access ... 383
17.4 Non-volatile server memory programming message flow examples ... 383
17.4.1 General information ... 383
17.4.2 Programming phase #1 — Pre-Programming step ... 383
17.4.3 Programming phase #1 — Programming step ... 384
17.4.4 Programming phase #1 — Post-Programming step ... 389

Annex A (normative) Global parameter definitions ... 390

Annex B (normative) Diagnostic and communication management functional unit data-parameter definitions ... 400

Annex C (normative) Data transmission functional unit data-parameter definitions ... 405

Annex D (normative) Stored data transmission functional unit data-parameter definitions ... 422

Annex E (normative) Input output control functional unit data-parameter definitions ... 444

Annex F (normative) Routine functional unit data-parameter definitions ... 445

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Annex G (normative) Upload and download functional unit data-parameter 447
Annex H (informative) Examples for addressAndLengthFormatIdentifier parameter values ... 448
Annex I (normative) Security access state chart 450
Annex J (informative) Recommended implementation for multiple client environments 458
Bibliography 464

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

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.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

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 and ISO/IEC 10731, which structures communication systems into seven layers. When mapped on this model, the services used by a diagnostic tester (client) and an Electronic Control Unit (ECU, server) are broken into the following layers in accordance with Table 1:

— Application layer (layer 7), unified diagnostic services specified in this document, ISO 14229-3 UDSonCAN, ISO 14229-4 UDSonFR, ISO 14229-5 UDSonIP, ISO 14229-6 UDSonK-Line, ISO 14229-7 UDSonLIN, ISO 14229-8[1] UDSonCXPI, further standards and ISO 27145-3 VOBD.

— Presentation layer (layer 6), vehicle manufacturer specific, ISO°27145-2 VOBD.

— Session layer services (layer 5) specified in ISO 14229-2.

— Transport layer services (layer 4), specified in ISO 15765-2 DoCAN, ISO 10681-2 Communication on FlexRay, ISO 13400-2 DoIP, ISO 17987-2 LIN, ISO 20794-3[2] CXPI, ISO 27145-4 VOBD.

— Network layer services (layer 3), specified in ISO 15765-2 DoCAN, ISO 10681-2 Communication on FlexRay, ISO 13400-2 DoIP, ISO 17987-2 LIN, ISO 20794-3 CXPI, ISO 27145-4 VOBD.

— Data link layer (layer 2), specified in ISO 11898-1, ISO 11898-2, ISO 17458-2, ISO 13400-3, IEEE 802.3, ISO 14230-2, ISO 17987-3 LIN, ISO 20794-4[3] CXPI, and further standards, ISO 27145-4 VOBD.

— Physical layer (layer 1), specified in ISO 11898-1, ISO 11898-2, ISO 17458-4, ISO 13400-3, IEEE 802.3, ISO 14230-1, ISO 17987-4 LIN, ISO 20794-4 CXPI, and further standards, ISO 27145-4 VOBD.

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 will provide long-term backward compatibility with the implementation standards as described above.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

[1] Under preparation. Stage at the time of publication: ISO/FDIS 14229-8:2020.

[2] Under preparation. Stage at the time of publication: ISO/FDIS 20794-3:2020.

[3] Under preparation. Stage at the time of publication: ISO/FDIS 20794-4:2020.

© ISO 2020 – All rights reserved

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

<table>
<tr><th>OSI seven layer[a]</th><th colspan="7">Enhanced diagnostics services</th><th>VOBD</th></tr>
<tr><td>Application (layer 7)</td><td colspan="7">ISO 14229-1, ISO 14229-3 UDSonCAN, ISO 14229-4 UDSonFR, ISO 14229-5 UDSonIP, ISO 14229-6 UDSonK-Line, ISO 14229-7 UDSonLIN, ISO 14229-8 UDSonCXPI, further standards</td><td>ISO 27145-3</td></tr>
<tr><td>Presentation (layer 6)</td><td colspan="7">vehicle manufacturer specific</td><td>ISO 27145-2</td></tr>
<tr><td>Session (layer 5)</td><td colspan="8">ISO 14229-2</td></tr>
<tr><td>Transport (layer 4)</td><td rowspan="2">ISO 15765-2</td><td rowspan="2">ISO 10681-2</td><td rowspan="2">ISO 13400-2</td><td rowspan="2">Not applicable</td><td rowspan="2">ISO 17987-2</td><td rowspan="2">ISO 20794-3</td><td>further standards</td><td rowspan="4">ISO 27145-4</td></tr>
<tr><td>Network (layer 3)</td><td>further standards</td></tr>
<tr><td>Data link (layer 2)</td><td rowspan="2">ISO 11898-1, ISO 11898-2</td><td>ISO 17458-2</td><td rowspan="2">ISO 13400-3, IEEE 802.3</td><td>ISO 14230-2</td><td>ISO 17987-3</td><td rowspan="2">ISO 20794-4</td><td>further standards</td></tr>
<tr><td>Physical (layer 1)</td><td>ISO 17458-4</td><td>ISO 14230-1</td><td>ISO 17987-4</td><td>further standards</td></tr>
<tr><td colspan="9">[a] Seven layers according to ISO/IEC 7498-1 and ISO/IEC 10731.</td></tr>
</table>

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

Road vehicles — Unified diagnostic services (UDS) —

Part 1: Application layer

1 Scope

This document specifies data link independent requirements of diagnostic services, which allow a diagnostic tester (client) to control diagnostic functions in an on-vehicle electronic control unit (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 generic 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. However, this document does not restrict an in-vehicle on-board tester (client) implementation in an ECU in order to utilize the diagnostic 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 14229-2, *Road vehicles — Unified diagnostic services (UDS) — Part 2: Session layer services*

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

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

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*

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

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*

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

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 0 and 17.3.1.1.

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.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

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

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

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).

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

4 Symbols and 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
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
SA	source address
SI	service identifier
TA	target address
TA_type	target address type
X.509	PKI standard according to ISO/IEC 9594-8

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

5 Conventions

This document is based on the conventions discussed in the OSI service conventions (ISO/IEC 10731) 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 1.

© ISO 2020 – All rights reserved

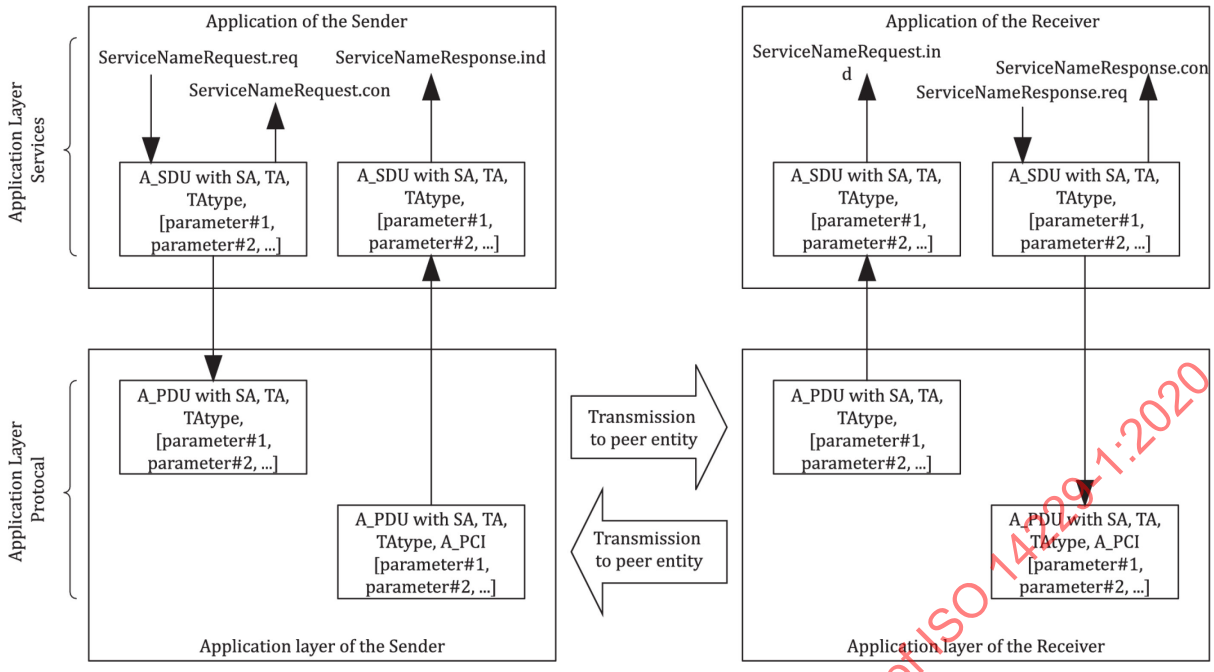


Figure 1 — The services and the protocol

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 Document overview

Figure 2 depicts the implementation of UDS document reference according to OSI model.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

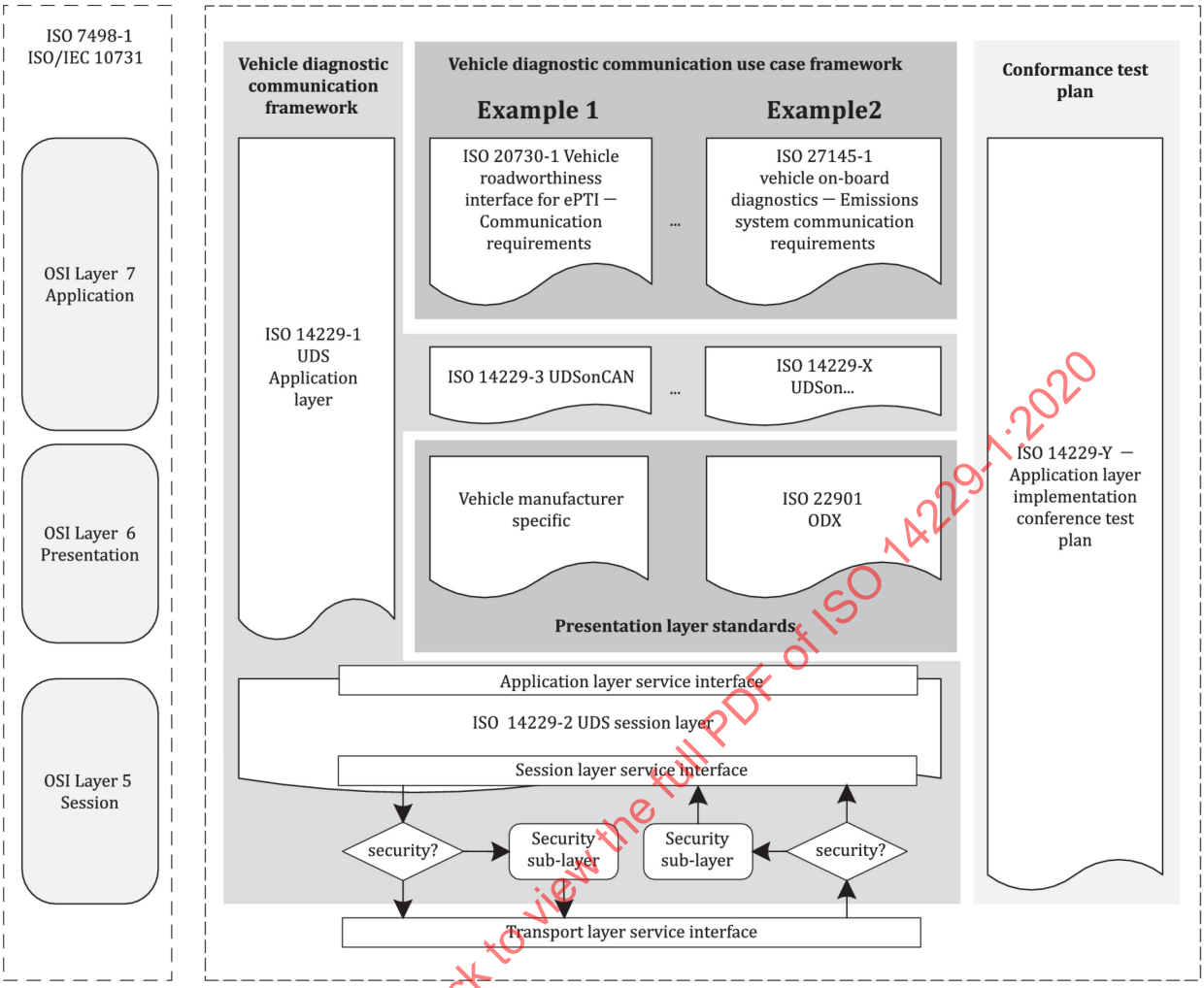


Figure 2 — Implementation of UDS document reference according to OSI model

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

7 Application layer services

7.1 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;

© ISO 2020 – All rights reserved

— 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.

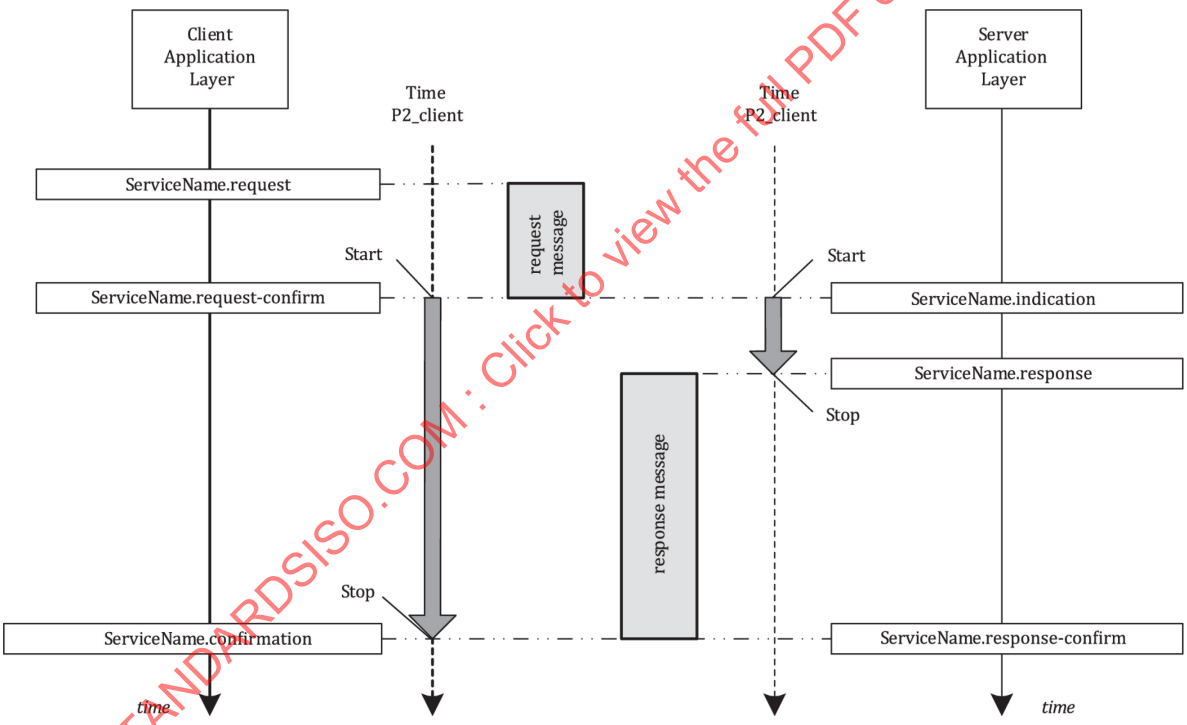


Figure 3 — Application layer service primitives – Confirmed service

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

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

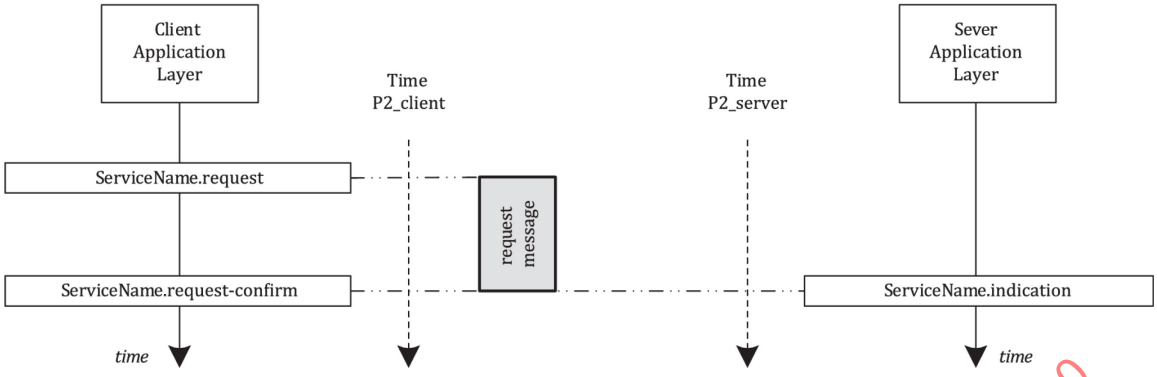


Figure 4 — Application layer service primitives – Unconfirmed service

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 will 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).

7.2 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 7.3.

7.3 Format description of service primitives

7.3.1 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, ...]
                    )
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

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.

7.3.2 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

7.3.3 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) will be specified.

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

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

7.3.4 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,
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

```
                              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.

7.4 Service data unit specification

7.4.1 Mandatory parameters

7.4.1.1 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).

7.4.1.2 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 7.2. 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

7.4.1.3 A_SA, application layer source address

Type: 2 byte unsigned integer value

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

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 will be the same as the A_TA value in the corresponding request message if physical addressing was used for the request message.

7.4.1.4 A_TA, application layer target address

Type: 2 byte unsigned integer value

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

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

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

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 will always be the same as the A_SA value of the corresponding request message.

7.4.1.5 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.

7.4.1.6 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).

7.4.1.7 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.

7.4.1.8 A_Data

Type: string of bytes

Range: not applicable

Description:

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

7.4.2 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

7.4.3 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.

8 Application layer protocol

8.1 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 8.7).

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).

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

8.2 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.

8.3 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 11.5) no A_PCI is present in the application layer protocol data unit (A_PDU).

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

8.4 SI, service identifier

Type: 1 byte unsigned integer value

Range: 00_{16} to FF_{16} according to definitions in Table 2.

Table 2 — 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/SAE J1979 specified services	ISO 15031-5/SAE J1979
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 / SAE J1979 positive service response	ISO 15031-5 / SAE J1979
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 82_{16}	Not applicable	Reserved by this document
83_{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
$C9_{16}$ 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 ($2A_{16}$, see 11.5) 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).

8.5 A_NR_SI, Negative response service identifier

Type: 1 byte unsigned integer value

Fixed value: $7F_{16}$

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.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

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.

8.6 Negative response/confirmation service primitive

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

Table 3 — Negative response A_PDU

A_PDU parameter	Parameter Name	Cvt	Byte value	Mnemonic
SA	Source Address	M	$XXXX_{16}$	SA
TA	Target Address	M	$XXXX_{16}$	TA
TAtype	Target Address type	M	XX_{16}	TAT
RA	Remote Address (optional)	C	$XXXX_{16}$	RA
A_Data.A_PCI.NR_SI	Negative Response SID	M	$7F_{16}$	SIDNR
A_Data.A_PCI. SI	<Service Name> Request SID	M	XX_{16}	SIDRQ
A_Data.Parameter 1	responseCode	M	XX_{16}	NRC_
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 A.1.

8.7 Server response implementation rules

8.7.1 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 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

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

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).

IMPORTANT — 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 ROOR (requestOutOfRange) shall not be transmitted when functional addressing was used for the request message (exception see A.1 in definition of NRC 78_{16}).

8.7.2 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

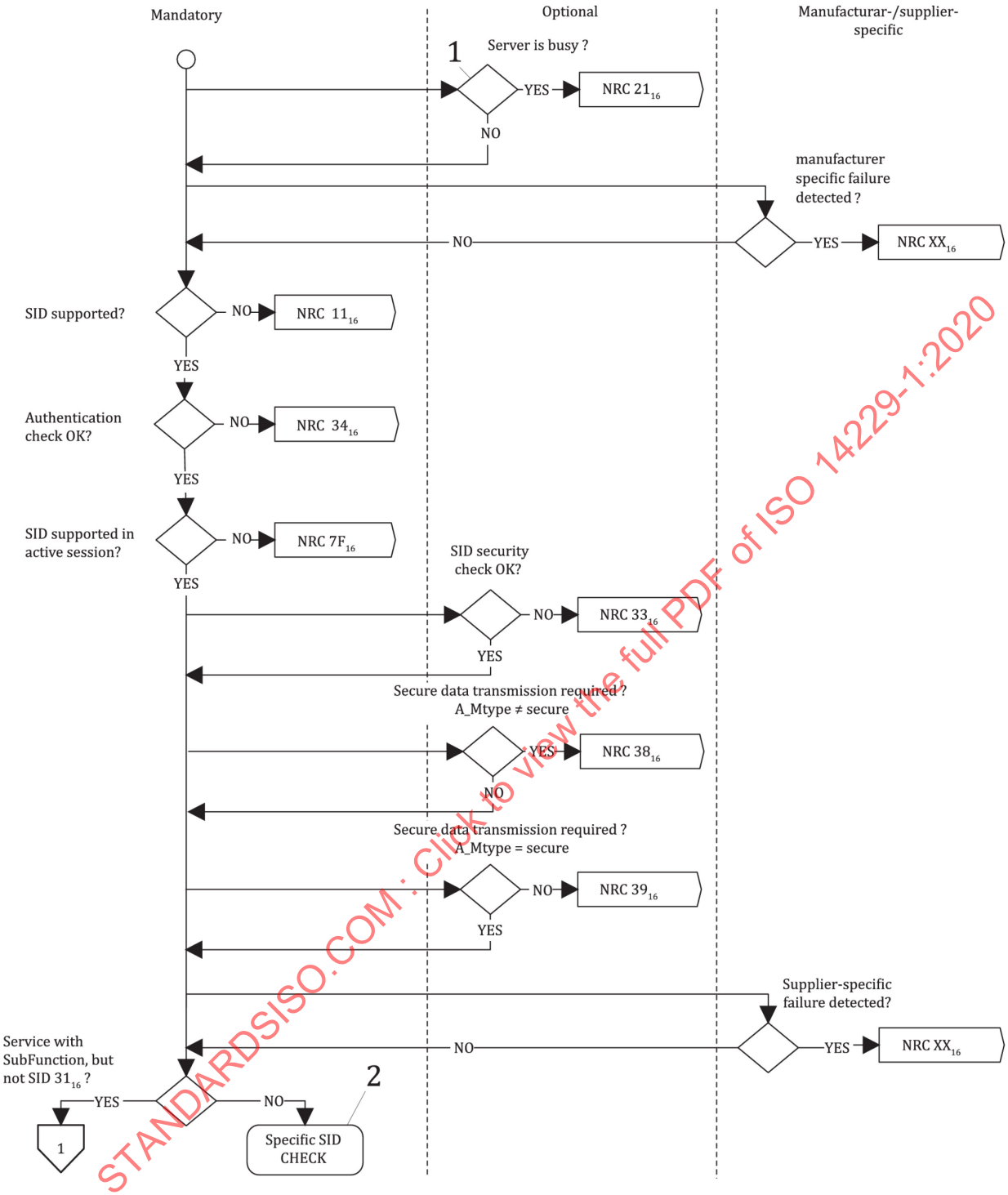
— manufacturer/supplier specific: the procedure can be extended by additional manufacturer/supplier specific checks.

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.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved



STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

Key

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

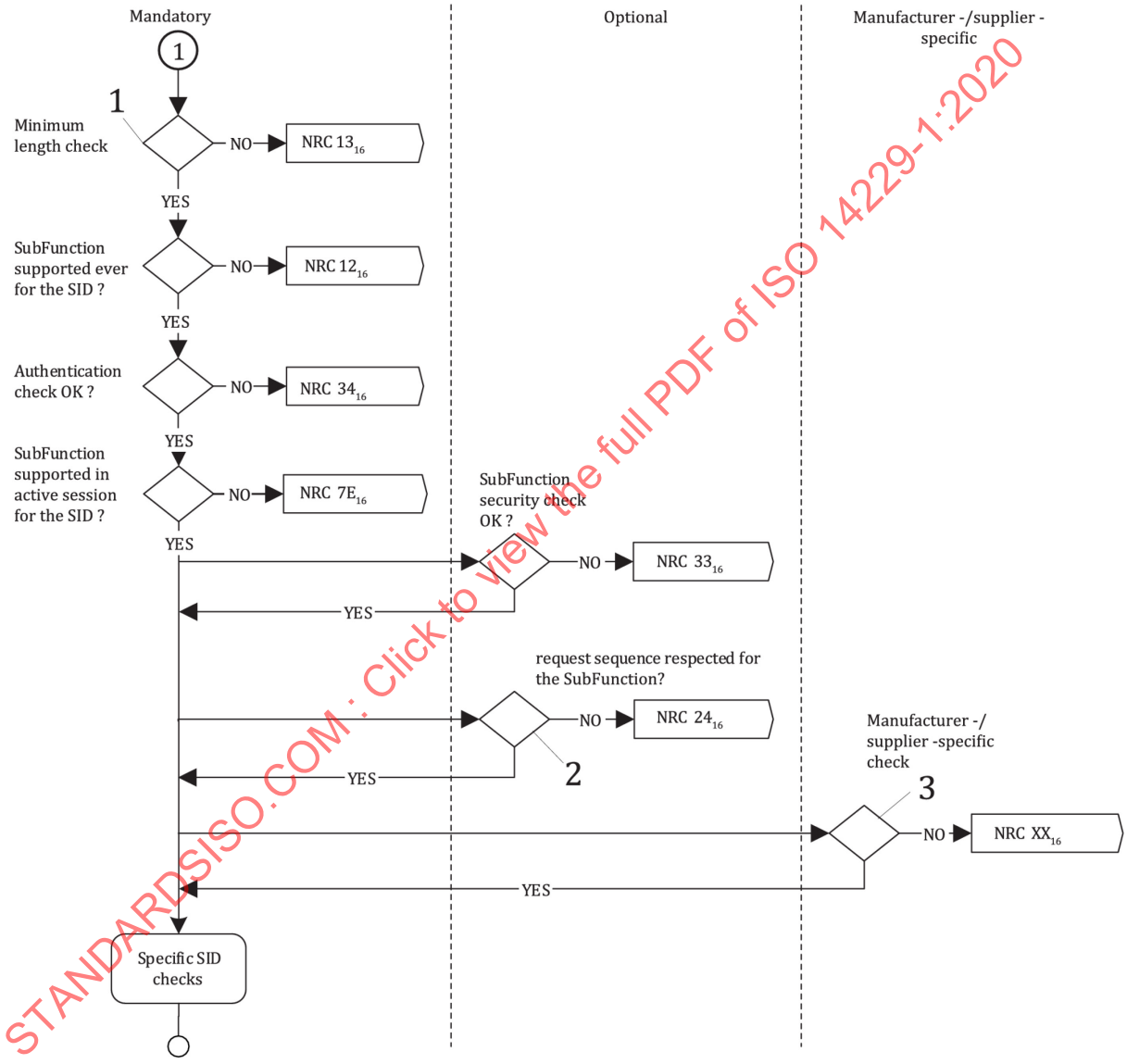
Figure 5 — General server response behaviour

© ISO 2020 – All rights reserved

8.7.3 Request message with SubFunction parameter and server response behaviour

8.7.3.1 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.



STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

Key

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

© ISO 2020 – All rights reserved

8.7.3.2 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 4 shows possible communication schemes with physical addressing.

Table 4 — 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 (suppressPosRspMsgIndBit)	SI supported	SF supported	DataParam supported (only if applicable)	Message	NRC	
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		NRC= ROOR	Negative response with NRC 31_{16}
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)	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		NRC= ROOR	Negative response with NRC 31_{16}
i)			NO	--	--		NRC= SNS	Negative response with NRC 11_{16} or $7F_{16}$
j)			YES	NO	--		NRC= SFNS	Negative response with NRC 12_{16} or $7E_{16}$

Description of server response cases on physically 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. IMLOIF: incorrectMessageLengthOrIncorrectFormat) because the service identifier and SubFunction

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

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 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.

d) 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.

e) 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.

8.7.3.3 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 5 shows possible communication schemes with functional addressing.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 5 — 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 (suppressPosRspMsgIndBit)	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	NoRsp	--	Server does NOT send a response
d)			NO	--	--		---	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. IMLOIF: 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

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.

8.7.4 Request message without SubFunction parameter and server response behaviour

8.7.4.1 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.

8.7.4.2 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 6 shows possible communication schemes with physical addressing.

Table 6 — Physically addressed request message without SubFunction parameter and server response behaviour

Server case #	Client request	Server capability		Server response		Comments to server response
	Addressing scheme	SI	DataParam	Message	NRC	
		supported				
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 occured 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):

a) Server sends a positive response message because the service identifier and all data-parameters are supported of the client's request message.

b) Server sends a positive response message because the service identifier and at least one data-parameter is supported of the client's request message.

c) Server sends a negative response message (e.g. IMLOIF: 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.

d) 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.

e) 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

8.7.4.3 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 7 shows possible communication schemes with functional addressing.

© ISO 2020 – All rights reserved

Table 7 — Functionally addressed request message without SubFunction parameter and server response behaviour

Server case #	Client request	Server capability		Server response		Comments to server response
	Addressing scheme	SI supported	DataParam supported	Message	NRC	
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 occured 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):

a) Server sends a positive response message because the service identifier and all data-parameters are supported of the client's request message.

b) Server sends a positive response message because the service identifier and at least one data-parameter is supported of the client's request message.

c) 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.

d) 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.

e) 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.

8.7.5 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 */
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

```
                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

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

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.

8.7.6 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 application call without response).

There are only two exceptions which shall be treated separately:

— 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.

— If a server supports services in the range of 00_{16} to $0F_{16}$ receives diagnostic requests in the range of 00_{16} to $0F_{16}$ then any active service outside the range of 00_{16} to $0F_{16}$ shall be aborted, the default session shall be started, and the diagnostic service in the range of 00_{16} to $0F_{16}$ shall be processed. This requirement does not apply if the programming session is active.

See Annex J for further information of how multiple clients can be handled.

9 Service description conventions

9.1 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 8 specifies the A_PDU parameter conventions.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 8 — A_PDU parameter conventions

Type	Name	Description
M	Mandatory	The parameter shall be present in the A_PDU.
C	Conditional	The parameter can be present in the A_PDU, based on certain criteria (e.g. 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.		

9.2 Request message

9.2.1 Request message definition

This subclause includes one or multiple tables, which define the A_PDU (Application layer protocol data unit, see Clause 8) 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 9 specifies the request A_PDU definition with SubFunction

Table 9 — 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	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
C: The RA (Remote Address) PDU parameter is only present in case of remote addressing.				

Table 10 specifies the request A_PDU definition without SubFunction.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

Table 10 — 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	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
C: 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.

9.2.2 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 11.

Table 11 — 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 8.7. 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Bit position	Description
6-0	**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 19_{16} SF $0A_{16}$), as this would defeat the purpose of SPRMIB. For the server implementation it is recommended to send NRC 78_{16} (RCRRP) and subsequently also send the positive response, in case paged-buffer-handling is used while SPRMIB is TRUE.

Table 12 specifies the request message SubFunction parameter definition.

Table 12 — 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 12 above shall be interpreted as defined in Table 13.

Table 13 — 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 14 specifies the calculation of the SubFunction byte value.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

Table 14 — Calculation of the SubFunction byte value

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

9.2.3 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 specifiy whether a data-parameter shall be supported or is user optional to be supported in case the server supports the service.

Table 15 specifies the request message data-parameters.

Table 15 — Request message data-parameter definition

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

9.3 Positive response message

9.3.1 Positive response message definition

This subclause specifies the A_PDU parameters for the service response/confirmation (see 8.2 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 16 specifies the positive response A_PDU.

Table 16 — 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	xxxx	RA
A_Data.A_PCI.SI	<Service Name> Response SID	S	xx	SIDPR

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

A_PDU parameter	Parameter Name	Cvt	Byte value	Mnemonic
A_Data.Parameter 1	data-parameter#1	U	xx	DP_...#1
:	:	:	:	:
A_Data.Parameter k	data-parameter#k	U	xx	DP_...#k
C: The RA (Remote Address) PDU parameter is only present in case of remote addressing.				

In all responses/confirmations the addressing information TA, SA, and 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.

9.3.2 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 specifiy whether a data-parameter shall be supported or is user optional to be supported in case the server supports the service.

Table 17 specifies the response data-parameters.

Table 17 — 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

9.4 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 8.6. The server shall use the negative response A_PDU for the indication of an identified error condition.

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

Table 18 specifies the supported negative response codes.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

Table 18 — 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_

9.5 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 19 specifies the request message flow example.

Table 19 — 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 20 specifies the positive response message flow example.

Table 20 — 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 21 shows a message flow example for a negative response message.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 21 — 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_

10 Diagnostic and communication management functional unit

10.1 Overview

Table 22 specifies the diagnostic and communication management functional unit.

Table 22 — 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.
SecuredDataTransmission	The client uses this service to perform data transmission with an extended data link security.
ControlDTCSetting	The client controls the setting of DTCs in the server.
ResponseOnEvent	The client requests to set up and/or control an event mechanism in the server.
LinkControl	The client requests control of the communication baudrate.

10.2 DiagnosticSessionControl (10_{16}) service

10.2.1 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. 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).

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

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 $F4_{16}$ may start the extendedDiagnosticSession).

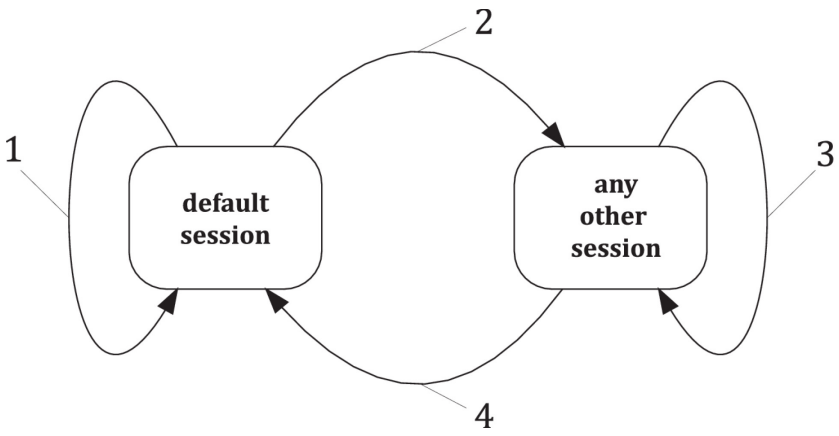
— Certain safety conditions may need to be satisfied (e.g. vehicle shall not be moving or engine shall not be running). Transitions to, for example programmingSession may lead to loss of normal functionality and therefore some ECUs may require the vehicle to be in a safe state.

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.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved



Key

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 memory

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 (86_{16}) 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:

- i) each event that has been configured in the server via the ResponseOnEvent (86_{16}) service shall be stopped
- ii) 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)
- iii) 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 (86_{16}) 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 23 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

Table 23 — 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
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.

IMPORTANT — The server and the client shall meet the request and response message behaviour as specified in 8.7.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

10.2.2 Request message

10.2.2.1 Request message definition

Table 24 specifies the request message.

Table 24 — 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

10.2.2.2 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 25.

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

Table 25 — Request message SubFunction parameter definition

Bit 6-0	Description	Cvt	Mnemonic
00_{16}	**ISOSAEReserved** This value is reserved by this document.	M	ISOSAERESRVD
01_{16}	**defaultSession** This diagnostic session enables the default diagnostic session in the server(s) and does not support any diagnostic application timeout handling provisions (e.g. no TesterPresent service is necessary to keep the session active). If any other session than the defaultSession has been active in the server and the defaultSession is once again started, then the following implementation rules shall be followed (see also 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.	M	DS
02_{16}	**ProgrammingSession**	U	PRGS

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Bit 6-0	Description	Cvt	Mnemonic
	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** 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 23).	U	EXTDS
04_{16}	**safetySystemDiagnosticSession** This diagnosticSession enables all diagnostic services required to support safety system related functions (e.g. airbag deployment).	U	SSDS
05_{16} to $3F_{16}$	**ISOSAEReserved** This value is reserved by this document for future definition.	M	ISOSAERESRVD
40_{16} to $5F_{16}$	**vehicleManufacturerSpecific** This range of values is reserved for vehicle manufacturer specific use.	U	VMS
60_{16} to $7E_{16}$	**systemSupplierSpecific** This range of values is reserved for system supplier specific use.	U	SSS
$7F_{16}$	**ISOSAEReserved** This value is reserved by this document for future definition.	M	ISOSAERESRVD

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

10.2.2.3 Request message data-parameter definition

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

10.2.3 Positive response message

10.2.3.1 Positive response message definition

Table 26 specifies the positive response message definition.

© ISO 2020 – All rights reserved

Table 26 — 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_
#3 : #6	sessionParameterRecord[]#1 = [data#1 : data#4]	M : M	00_{16} to FF_{16} : 00_{16} to FF_{16}	SPREC_ DATA_1 : DATA_m

10.2.3.2 Positive response message data-parameter definition

Table 27 specifies the response message data-parameter definition.

Table 27 — 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 28 and Table 29.

Table 28 and Table 29 define the structure of the response message data-parameter sessionParameterRecord as applicable for the implementation of this service on supported data links.

Table 28 — sessionParameterRecord definition

Byte pos. in record	Description	Cvt	Byte value	Mnemonic
	sessionParameterRecord[] = [			SPREC_
#1	$P2_{Server_max}$ (high byte)	M	00_{16} to FF_{16}	P2SMH
#2	$P2_{Server_max}$ (low byte)	M	00_{16} to FF_{16}	P2SML
#3	$P2^*_{Server_max}$ (high byte)	M	00_{16} to FF_{16}	P2ESMH
#4	$P2^*_{Server_max}$ (low byte)]	M	00_{16} to FF_{16}	P2ESML

Table 29 — sessionParameterRecord content definition

Parameter	Description	# of bytes	Resolution	minimum value	maximum value
$P2_{Server_max}$	Default $P2_{Server_max}$ timing supported by the server for the activated diagnostic session.	2	1 ms	0 ms	65 535 ms
$P2^*_{Server_max}$	Enhanced (NRC 78_{16}) $P2_{Server_max}$ 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 $P2_{Server}$ and $P2^*_{Server}$.

10.2.4 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 30. The listed negative responses shall be used if the error scenario applies to the server.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 30 — 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

10.2.5 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 $P2_{Server_max}$ is equal to 50 ms and the $P2^*_{Server_max}$ is equal to 5 000 ms.

Table 31 specifies the DiagnosticSessionControl request message flow example #1.

Table 31 — 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 32 specifies the DiagnosticSessionControl positive response message flow example.

Table 32 — DiagnosticSessionControl positive response message flow example

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	DiagnosticSessionControl Response SID	50_{16}	DSCPR
#2	diagnosticSessionType = programmingSession	02_{16}	DS_ECUPRGS
#3 #4 #5 #6	sessionParameterRecord [$P2_{Server_max}$ (high byte)] sessionParameterRecord [$P2_{Server_max}$ (low byte)] sessionParameterRecord [$P2^*_{Server_max}$ (high byte)] sessionParameterRecord [$P2^*_{Server_max}$ (low byte)]	00_{16} 32_{16} 01_{16} $F4_{16}$	SPREC_1 SPREC_2 SPREC_3 SPREC_4

10.3 ECUReset (11_{16}) service

10.3.1 Service description

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

This service requests the server to effectively perform a server reset based on the content of the resetType parameter value embedded in the ECUReset request message. The ECUReset positive

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

response message (if required) can be sent before or after the reset has been executed in the server(s). It is strongly recommended to send the ECUReset positive response message before the ECU reset is executed.

IMPORTANT — The server and the client shall meet the request and response message behaviour as specified in 8.7.

This document does not define the behaviour of the ECU from the time following the positive response message to the ECU reset request until the reset has successfully completed. It is recommended that during this time the ECU does not accept any request messages and send any response messages.

10.3.2 Request message

10.3.2.1 Request message definition

Table 33 specifies the request message definition.

Table 33 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	ECUReset Request SID	M	11_{16}	ER
#2	SubFunction = [resetType]	M	00_{16} to FF_{16}	LEV_RT_

10.3.2.2 Request message SubFunction Parameter $Level (LEV_) definition

The SubFunction parameter resetType is used by the ECUReset request message to describe how the server shall perform the reset (suppressPosRspMsgIndicationBit (bit 7) not shown).

Table 34 specifies the request message SubFunction parameter definition.

Table 34 — Request message SubFunction parameter definition

Bits 6 to 0	Description	Cvt	Mnemonic
00_{16}	**ISOSAEReserved** This value is reserved by this document.	M	ISOSAERESRVD
01_{16}	**hardReset** This SubFunction identifies a "hard reset" condition which simulates the power-on/start-up sequence typically performed after a server has been previously disconnected from its power supply (i.e. battery). The performed action is implementation specific and not defined by this document. It might result in the re-initialization of both volatile memory and non-volatile memory locations to predetermined values.	U	HR
02_{16}	**keyOffOnReset** This SubFunction identifies a condition similar to the driver turning the ignition key off and back on. This reset condition should simulate a key-off-on sequence (i.e. interrupting the switched power supply). The performed action is implementation specific and not defined by this document. Typically the values of non-volatile memory locations are preserved; volatile memory will be initialized.	U	KOFFONR
03_{16}	**softReset**	U	SR

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Bits 6 to 0	Description	Cvt	Mnemonic
	This SubFunction identifies a "soft reset" condition, which causes the server to immediately restart the application program if applicable. The performed action is implementation specific and not defined by this document. A typical action is to restart the application without reinitializing of previously learned configuration data, adaptive factors and other long-term adjustments.		
04_{16}	**enableRapidPowerShutDown** This SubFunction applies to ECUs which are not ignition powered but battery powered only. Therefore a shutdown forces the sleep mode rather than a power off. Sleep means power off but still ready for wake-up (battery powered). The intention of the SubFunction is to reduce the stand-by time of an ECU after ignition is turned into the off position. This value requests the server to enable and perform a "rapid power shut down" function. The server shall execute the function immediately once the "key/ignition" is switched off. While the server executes the power down function, it shall transition either directly or after a defined stand-by-time to sleep mode. If the client requires a response message and the server is already prepared to execute the "rapid power shut down" function, the server shall send the positive response message prior to the start of the "rapid power shut down" function. The next occurrence of a "key on" or "ignition on" signal terminates the "rapid power shut down" function. NOTE This SubFunction is only applicable to a server supporting a stand-by-mode.	U	ERPSD
05_{16}	**disableRapidPowerShutDown** This SubFunction requests the server to disable the previously enabled "rapid power shut down" function.	U	DRPSD
06_{16} to $3F_{16}$	**ISOSAEReserved** This range of values is reserved by this document for future definition.	M	ISOSAERESRVD
40_{16} to $5F_{16}$	**vehicleManufacturerSpecific** This range of values is reserved for vehicle manufacturer specific use.	U	VMS
60_{16} to $7E_{16}$	**systemSupplierSpecific** This range of values is reserved for system supplier specific use.	U	SSS
$7F_{16}$	**ISOSAEReserved** This value is reserved by this document for future definition.	M	ISOSAERESRVD

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

10.3.2.3 Request message data-parameter definition

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

10.3.3 Positive response message

10.3.3.1 Positive response message definition

Table 35 specifies the positive response message.

© ISO 2020 – All rights reserved

Table 35 — Positive response message definition

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	ECUReset Response SID	M	51_{16}	ERPR
#2	SubFunction = [resetType]	M	00_{16} to $7F_{16}$	LEV_RT_
#3	powerDownTime	C	00_{16} to FF_{16}	PDT
C: This parameter is present if the SubFunction parameter is set to the enableRapidPowerShutDown value (04_{16});				

10.3.3.2 Positive response message data-parameter definition

Table 36 specifies the data-parameters of the response message.

Table 36 — Response message data-parameter definition

Definition
resetType This parameter is an echo of bits 6 to 0 of the SubFunction parameter from the request message.
powerDownTime This parameter indicates to the client the minimum time of the stand-by-sequence the server will remain in the power down sequence. The resolution of this parameter is one (1) second per count. The follwing values are valid: — 00_{16} to FE_{16}: 0 to 254 seconds powerDownTime, — FF_{16}: indicates a failure or time not available.

10.3.4 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 37. The listed negative responses shall be used if the error scenario applies to the server.

Table 37 — 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 ECUReset request is not met.	CNC
33_{16}	**securityAccessDenied** This NRC shall be sent if the requested reset is secured and the server is not in an unlocked state.	SAD

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

10.3.5 Message flow example ECUReset

This subclause specifies the conditions for the example to be fulfilled to successfully perform an ECUReset service in the server.

Condition of server: ignition = on, system shall not be in an operational mode (e.g. if the system is an engine management, engine shall be off).

The client requests to have a response message by setting the suppressPosRspMsgIndicationBit (bit 7 of the SubFunction parameter) to 'FALSE'.

The server shall send an ECUReset positive response message before the server performs the resetType.

Table 38 specifies the ECUReset request message flow example #1.

Table 38 — ECUReset request message flow example #1

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ECUReset Request SID	11_{16}	ER
#2	ResetType = hardReset, suppressPosRspMsgIndicationBit = FALSE	01_{16}	RT_HR

Table 39 specifies the ECUReset positive response message flow example #1.

Table 39 — ECUReset positive response message flow example #1

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ECUReset Response SID	51_{16}	ERPR
#2	resetType = hardReset	01_{16}	RT_HR

10.4 SecurityAccess (27_{16}) service

10.4.1 Service description

The purpose of this service is to provide a means to access data and/or diagnostic services, which have restricted access for security, emissions, or safety reasons. Diagnostic services for downloading/uploading routines or data into a server and reading specific memory locations from a server are situations where security access may be required. Improper routines or data downloaded into a server could potentially damage the electronics or other vehicle components or risk the vehicle's compliance to emission, safety, or security standards. The security concept uses a seed and key relationship.

A typical example of the use of this service is as follows:

— client requests the "Seed",

— server sends the "Seed",

— client sends the "Key" (appropriate for the Seed received),

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

— server responds that the “Key” was valid and that it will unlock itself.

The 'requestSeed' SubFunction parameter value shall always be an odd number and the corresponding ‘sendKey' SubFunction parameter value for the same security level shall equal the 'requestSeed' SubFunction parameter value plus one.

Only one security level shall be active at any instant of time. For example, if the security level associated with requestSeed 03_{16} is active and a tester request is successful in unlocking the security level associated with requestSeed 01_{16}, then only the secured functionality supported by the security level associated with requestSeed 01_{16} shall be unlocked at that time. Any additional secured functionality that was previously unlocked by the security level associated with requestSeed 03_{16} shall no longer be active. The security levels numbering is arbitrary and does not imply any relationship between the levels.

The client shall request the server to "unlock" by sending the service SecurityAccess ‘requestSeed’ message. The server shall respond by sending a "seed" using the service SecurityAccess ‘requestSeed’ positive response message. The client shall then respond by returning a "key" number back to the server using the appropriate service SecurityAccess ‘sendKey’ request message. The server shall compare this "key" to one internally stored/calculated. If the two numbers match, then the server shall enable ("unlock") the client's access to specific services/data and indicate that with the service SecurityAccess ‘sendKey’ positive response message. If the two numbers do not match, this shall be considered a false access attempt. An invalid key shall require the client to start over from the beginning with a SecurityAccess 'requestSeed' message as specified in Annex I. Additional details regarding security access handling details are specified in Annex I.

If a server supports security, but the requested security level is already unlocked when a SecurityAccess ‘requestSeed’ message is received, that server shall respond with a SecurityAccess ‘requestSeed’ positive response message service with a seed value equal to zero (0). The server shall never send an all zero seed for a given security level that is currently locked. The client shall use this method to determine if a server is locked for a particular security level by checking for a non-zero seed.

A vehicle manufacturer specific time delay may be required before the server can positively respond to a service SecurityAccess ‘requestSeed’ message from the client after server power up/reset and after a certain number of false access attempts (see further description below). If this delay timer is supported then the delay shall be activated after a vehicle manufacturer specified number of false access attempts has been reached or when the server is powered up/reset and a previously performed SecurityAccess service has failed due to a single false access attempt. In case the server supports this delay timer then after a successful SecurityAccess service 'sendKey' execution the server internal indication information for a delay timer invocation on a power up/reset shall be cleared by the server. In case the server supports this delay timer and cannot determine if a previously performed SecurityAccess service prior to the power up/reset has failed then the delay timer shall always be active after power up/reset. The delay is only required if the server is locked when powered up/reset. The vehicle manufacturer shall select if the delay timer is supported.

Attempts to access security shall not prevent normal vehicle communications or other diagnostic communication.

Servers, which provide security shall support reject messages if a secure service is requested while the server is locked.

Some diagnostic functions/services requested during a specific diagnostic session may require a successful security access sequence. In such case the following sequence of services shall be required:

— DiagnosticSessionControl service,

— SecurityAccess service, and

— Secured diagnostic service.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

There are different accessModes allowed for an enabled diagnosticSession (session started) in the server.

IMPORTANT — The server and the client shall meet the request and response message behaviour as specified in 8.7.

10.4.2 Request message

10.4.2.1 Request message definition

Table 40 specifies the request message definition - SubFunction = requestSeed.

Table 40 — Request message definition - SubFunction = requestSeed

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	SecurityAcces Request SID	M	27_{16}	SA
#2	SubFunction = [securityAccessType = requestSeed]	M	01_{16}, 03_{16}, 05_{16}, 07_{16} to $7D_{16}$	LEV_ SAT_RSD
#3 : #n	securityAccessDataRecord[] = [parameter#1 : parameter#m]	U : U	00_{16} to FF_{16} : 00_{16} to FF_{16}	SECACCDR_ PARA1 : PARAm

Table 41 specifies the request message definition - SubFunction = sendKey.

Table 41 — Request message definition - SubFunction = sendKey

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	SecurityAcces Request SID	M	27_{16}	SA
#2	SubFunction = [securityAccessType = sendKey]	M	02_{16}, 04_{16}, 06_{16}, 08_{16} to $7E_{16}$	LEV_SAT_SK
#3 : #n	securityKey[] = [key#1 (high byte) : key#m (low byte)]	M : U	00_{16} to FF_{16} : 00_{16} to FF_{16}	SECKEY_ KEY1HB : KEYmLB

10.4.2.2 Request message SubFunction parameter $Level (LEV_) definition

The SubFunction parameter securityAccessType indicates to the server the step in progress for this service, the level of security the client wants to access and the format of seed and key. If a server supports different levels of security each level shall be identified by the requestSeed value, which has a fixed relationship to the sendKey value:

— "requestSeed = 01_{16}" identifies a fixed relationship between "requestSeed = 01_{16}" and "sendKey = 02_{16}".

— "requestSeed = 03_{16}" identifies a fixed relationship between "requestSeed = 03_{16}" and "sendKey = 04_{16}".

Values are defined in Table 42 for requestSeed and sendKey (suppressPosRspMsgIndicationBit (bit 7) not shown).

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 42 — Request message SubFunction parameter definition

Bits 6 to 0	Description	Cvt	Mnemonic
00_{16}	**ISOSAEReserved** This value is reserved by this document.	M	ISOSAERESRVD
01_{16}	**requestSeed** RequestSeed with the level of security defined by the vehicle manufacturer.	U	RSD
02_{16}	**sendKey** SendKey with the level of security defined by the vehicle manufacturer.	U	SK
03_{16}, 05_{16}, 07_{16} to 41_{16}	**requestSeed** RequestSeed with different levels of security defined by the vehicle manufacturer.	U	RSD
04_{16}, 06_{16}, 08_{16} to 42_{16}	**sendKey** SendKey with different levels of security defined by the vehicle manufacturer.	U	SK
43_{16} to $5E_{16}$	**ISOSAEReserved** This value is reserved by this document for future definition.	M	ISOSAERESRVD
$5F_{16}$	**ISO26021-2 values** RequestSeed with different levels of security defined for end of life activation of on-board pyrotechnic devices as defined in ISO 26021-2.	U	RSD
60_{16}	**ISO26021-2 sendKey values** SendKey with different levels of security defined for end of life activation of on-board pyrotechnic devices as defined in ISO 26021-2.	U	SK
61_{16} to $7E_{16}$	**systemSupplierSpecific** This range of values is reserved for system supplier specific use.	U	SSS
$7F_{16}$	**ISOSAEReserved** This value is reserved by this document for future definition.	M	ISOSAERESRVD

10.4.2.3 Request message data-parameter definition

Table 43 specifies the data-parameters of the request message.

Table 43 — Request message data-parameter definition

Definition
securityKey (high and low bytes) The “Key” parameter in the request message is the value generated by the security algorithm corresponding to a specific “Seed” value.
securityAccessDataRecord This parameter record is user optional to transmit data to a server when requesting the seed information. It can for example contain an identification of the client that is verified in the server.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

10.4.3 Positive response message

10.4.3.1 Positive response message definition

Table 44 specifies the positive response message.

Table 44 — Positive response message definition

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	SecurityAccess Response SID	M	67	SAPR
#2	SubFunction = [securityAccessType]	M	00-7F	LEV_SAT_SK
#3 : #n	securitySeed[] = [seed#1 (high byte) : seed#m (low byte)]	C : C	00_{16} to FF_{16} : 00_{16} to FF_{16}	SECSEED_ SEED1HB : SEEDmLB
C: The presence of this parameter depends on the securityAccessType parameter. It is mandatory to be present if the securityAccessType parameter indicates that the client wants to retrieve the seed from the server.				

10.4.3.2 Positive response message data-parameter definition

Table 45 specifies the data-parameters of the response message.

Table 45 — Response message data-parameter definition

Definition
securityAccessType This parameter is an echo of bits 6 to 0 of the SubFunction parameter from the request message.
securitySeed (high and low bytes) The seed parameter is a data value sent by the server and is used by the client when calculating the key needed to access security. The securitySeed data bytes are only present in the response message if the request message was sent with the SubFunction set to a value which requests the seed of the server.

10.4.4 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 46. The listed negative responses shall be used if the error scenario applies to the server.

Table 46 — 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 SecurityAccess are not met.	CNC
24_{16}	**requestSequenceError** Send if the 'sendKey' SubFunction is received without first receiving a 'requestSeed' request message.	RSE

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

NRC	Description	Mnemonic
31_{16}	**requestOutOfRange**	ROOR
	This NRC shall be sent if the user optional securityAccessDataRecord contains invalid data.	
35_{16}	**invalidKey** Send if an expected 'sendKey' SubFunction value is received, the value of the key does not match the server's internally stored/calculated key, and the delay timer is not activated by this request.	IK
36_{16}	**exceededNumberOfAttempts** Send if an expected 'sendKey'SubFunction is received, the value of the key does not match the server's internally stored/calculated key, and the delay timer is activated by this request (i.e. due to reaching the limit of false access attempts which activate the delay timer).	ENOA
37_{16}	**requiredTimeDelayNotExpired** Send if a 'requestSeed'SubFunction is received and the delay timer is active for the requested security level.	RTDNE

10.4.5 Message flow example(s) SecurityAccess

10.4.5.1 Assumptions

For the below given message flow examples the following conditions are assumed to successfully unlock the server if it is in a "locked" state:

— SubFunction to request the seed: 01_{16} (requestSeed)

— SubFunction to send the key: 02_{16} (sendKey)

— seed of the server (2 bytes): 3657_{16}

— key of the server (2 bytes): $C9A9_{16}$ (e.g. 2's complement of the seed value)

The client requests to have a response message by setting the suppressPosRspMsgIndicationBit (bit 7 of the SubFunction parameter) to "FALSE" ('0').

10.4.5.2 Example #1 - server is in a "locked" state

10.4.5.2.1 Step #1: Request the Seed

Table 47 specifies the SecurityAccess request message flow example #1 – step #1.

Table 47 — SecurityAccess request message flow example #1 – step #1

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	SecurityAccess Request SID	27_{16}	SA
#2	SecurityAccessType = requestSeed, suppressPosRspMsgIndicationBit = FALSE	01_{16}	SAT_RSD

Table 48 specifies the SecurityAccess positive response message flow example #1 – step #1.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 48 — SecurityAccess positive response message flow example #1 – step #1

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	SecurityAccess Response SID	67_{16}	SAPR
#2	securityAccessType = requestSeed	01_{16}	SAT_RSD
#3	securitySeed [byte#1] = seed#1 (high byte)	36_{16}	SECHB
#4	securitySeed [byte#2] = seed#2 (low byte)	57_{16}	SECLB

10.4.5.2.2 Step #2: Send the Key

Table 49 specifies the SecurityAccess request message flow example #1 – step #2.

Table 49 — SecurityAccess request message flow example #1 – step #2

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	SecurityAccess Request SID	27_{16}	SA
#2	securityAccessType = sendKey, suppressPosRspMsgIndicationBit = FALSE	02_{16}	SAT_SK
#3	securityKey [byte#1] = key#1 (high byte)	$C9_{16}$	SECKEY_HB
#4	securityKey [byte#2] = key#2 (low byte)	$A9_{16}$	SECKEY_LB

Table 50 specifies the SecurityAccess positive response message flow example #1 – step #2.

Table 50 — SecurityAccess positive response message flow example #1 – step #2

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	SecurityAccess Response SID	67_{16}	SAPR
#2	securityAccessType = sendKey	02_{16}	SAT_SK

10.4.5.3 Example #2 - server is in an “unlocked” state

10.4.5.3.1 Step #1: Request the Seed

Table 51 specifies the SecurityAccess request message flow example #2 – step #1.

Table 51 — SecurityAccess request message flow example #2 – step #1

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	SecurityAccess Request SID	27_{16}	SA

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Message direction	client → server		
Message type	Request		
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#2	securityAccessType = requestSeed, suppressPosRspMsgIndicationBit = FALSE	01_{16}	SAT_RSD

Table 52 specifies the SecurityAccess positive response message flow example #2 – step #2.

Table 52 — SecurityAccess positive response message flow example #2 – step #2

Message direction	server → client		
Message type	Response		
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	SecurityAccess Response SID	67_{16}	SAPR
#2	securityAccessType = requestSeed	01_{16}	SAT_RSD
#3	securitySeed [byte#1] = seed#1 (high byte)	00_{16}	SECHB
#4	securitySeed [byte#2] = seed#2 (low byte)	00_{16}	SECLB

10.5 CommunicationControl (28_{16}) service

10.5.1 Service description

The purpose of this service is to switch on/off the transmission and/or the reception of certain messages of (a) server(s) (e.g. application communication messages).

The server shall still send a positive response if the service is supported in the active session with a requested SubFunction even if the requested SubFunction state is already active.

IMPORTANT — The server and the client shall meet the request and response message behaviour as specified in 8.7.

10.5.2 Request message

10.5.2.1 Request message definition

Table 53 specifies the request message.

Table 53 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	CommunicationControl Request SID	M	28_{16}	CC
#2	SubFunction = [controlType]	M	00_{16} to FF_{16}	LEV_CTRLTP
#3	communicationType	M	00_{16} to FF_{16}	CTP
#4	nodeIdentificationNumber (high byte)	C[a]	00_{16} to FF_{16}	NIN
#5	nodeIdentificationNumber (low byte)	C[a]	00_{16} to FF_{16}	NIN

[a] The presence of the C parameter requires the controlType either being 04_{16} or 05_{16}.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

10.5.2.2 Request message SubFunction parameter $Level (LEV_) definition

The SubFunction parameter controlType contains information on how the server shall modify the communication type referenced in the communicationType parameter (suppressPosRspMsgIndicationBit (bit 7) not shown in Table 54).

Table 54 — Request message SubFunction parameter definition

Bits 6 to 0	Description	Cvt	Mnemonic
00_{16}	**enableRxAndTx** This value indicates that the reception and transmission of messages shall be enabled for the specified communicationType.	U	ERXTX
01_{16}	**enableRxAndDisableTx** This value indicates that the reception of messages shall be enabled and the transmission shall be disabled for the specified communicationType.	U	ERXDTX
02_{16}	**disableRxAndEnableTx** This value indicates that the reception of messages shall be disabled and the transmission shall be enabled for the specified communicationType.	U	DRXETX
03_{16}	**disableRxAndTx** This value indicates that the reception and transmission of messages shall be disabled for the specified communicationType.	U	DRXTX
04_{16}	**enableRxAndDisableTxWithEnhancedAddressInformation** This value indicates that the addressed bus master shall switch the related sub-bus segment to the diagnostic-only scheduling mode.	U	ERXDTXWEAI
05_{16}	**enableRxAndTxWithEnhancedAddressInformation** This value indicates that the addressed bus master shall switch the related sub-bus segment to the application scheduling mode.	U	ERXTXWEAI
06_{16} to $3F_{16}$	**ISOSAEReserved** This range of values is reserved by this document for future definition.	M	ISOSAERESRVD
40_{16} to $5F_{16}$	**vehicleManufacturerSpecific** This range of values is reserved for vehicle manufacturer specific use.	U	VMS
60_{16} to $7E_{16}$	**systemSupplierSpecific** This range of values is reserved for system supplier specific use.	U	SSS
$7F_{16}$	**ISOSAEReserved** This value is reserved by this document for future definition.	M	ISOSAERESRVD

10.5.2.3 Request message data-parameter definition

Table 55 specifies the data-parameters of the request message.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 55 — Request message data-parameter definition

Definition
communicationType This parameter is used to reference the kind of communication to be controlled. The communicationType parameter is a bit-code value, which allows controlling multiple communication types at the same time. (See B.1 for the coding of the communicationType data-parameter.)
nodeIdentificationNumber This 2 byte parameter is used to identify a node on a sub-network somewhere in the vehicle, which cannot be addressed using the addressing methods of the lower OSI layers 1 to 6. This parameter is only present, if the SubFunction parameter controlType is set to 04_{16} or 05_{16} (see B.4 for the coding of the nodeIdentificationNumber data-parameter).

10.5.3 Positive response message

10.5.3.1 Positive response message definition

Table 56 specifies the positive response message.

Table 56 — Positive response message definition

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	CommunicationControl Response SID	M	68_{16}	CCPR
#2	SubFunction = [controlType]	M	00_{16} to $7F_{16}$	LEV_CTRLTP

10.5.3.2 Positive response message data-parameter definition

Table 57 specifies the data-parameters of the positive response message.

Table 57 — Response message data-parameter definition

Definition
controlType This parameter is an echo of bits 6 to 0 of the SubFunction parameter from the request message.

10.5.4 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 58. The listed negative responses shall be used if the error scenario applies to the server.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

Table 58 — 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** Used when the server is in a critical normal mode activity and therefore cannot disable/enable the requested communication type.	CNC
31_{16}	**requestOutOfRange** The server shall use this response code, if it detects an error in the communicationType or nodeIdentificationNumber parameter.	ROOR

10.5.5 Message flow example CommunicationControl (disable transmission of network management messages)

The client requests to have a response message by setting the suppressPosRspMsgIndicationBit (bit 7 of the SubFunction parameter) to "FALSE" ('0').

Table 59 specifies the CommunicationControl request message flow example.

Table 59 — CommunicationControl request message flow example

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	CommunicationControl Request SID	28_{16}	CC
#2	controlType = enableRxAndDisableTx, suppressPosRspMsgIndicationBit = FALSE	01_{16}	ERXTX
#3	communicationType = network management	02_{16}	NWMCP

Table 60 specifies the CommunicationControl positive response message flow example.

Table 60 — CommunicationControl positive response message flow example

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	CommunicationControl Response SID	68_{16}	CCPR
#2	ControlType	01_{16}	CTRLTP

10.5.6 Message flow example CommunicationControl (switch a remote network into the diagnostic-only scheduling mode where the node with address $000A_{16}$ is connected to)

The client requests to have a response message by setting the suppressPosRspMsgIndicationBit (bit 7 of the SubFunction parameter) to "FALSE" ('0').

Table 61 specifies the CommunicationControl request message flow example.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 61 — CommunicationControl request message flow example

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	CommunicationControl Request SID	28_{16}	CC
#2	controlType = enableRxAndDisableTxWithEnhancedAddressInformation, suppressPosRspMsgIndicationBit = FALSE	04_{16}	ERXTX
#3	communicationType = normal messages	01_{16}	NMCP
#4	nodeIdentificationNumber (high byte)	00_{16}	NIN
#5	nodeIdentificationNumber (low byte)	$0A_{16}$	NIN

Table 62 specifies the CommunicationControl positive response message flow example.

Table 62 — CommunicationControl positive response message flow example

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	CommunicationControl Response SID	68_{16}	CCPR
#2	controlType = enableRxAndDisableTxWithEnhancedAddressInformation, suppressPosRspMsgIndicationBit = FALSE	04_{16}	ERXTX

10.5.7 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)

The client requests to have a response message by setting the suppressPosRspMsgIndicationBit (bit 7 of the SubFunction parameter) to "FALSE" ('0').

Table 63 specifies the CommunicationControl request message flow example.

Table 63 — CommunicationControl request message flow example

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	CommunicationControl Request SID	28_{16}	CC
#2	controlType = enableRxAndTxWithEnhancedAddressInformation, suppressPosRspMsgIndicationBit = FALSE	05_{16}	ERXTX
#3	communicationType = normal messages	01_{16}	NMCP
#4	nodeIdentificationNumber (high byte)	00_{16}	NIN
#5	nodeIdentificationNumber (low byte)	$0A_{16}$	NIN

Table 64 specifies the CommunicationControl positive response message flow example.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 64 — CommunicationControl positive response message flow example

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	CommunicationControl Response SID	68_{16}	CCPR
#2	controlType = enableRxAndTxWithEnhancedAddressInformation, suppressPosRspMsgIndicationBit = FALSE	05_{16}	ERXTX

10.6 Authentication (29_{16}) service

10.6.1 Service overview

The purpose of this service is to provide a means for the client to prove its identity, allowing it to access data and/or diagnostic services, which have restricted access for, for example security, emissions, or safety reasons. Diagnostic services for downloading/uploading routines or data into a server and reading specific memory locations from a server are situations where authentication may be required. Improper routines or data downloaded into a server could potentially damage the electronics or other vehicle components or risk the vehicle's compliance to emission, safety, or security standards. On the other hand, data security might be violated when retrieving data from a server.

This service supports two security concepts:

— Concept #1 is based on PKI certificate exchange procedures using asymmetric cryptography (see 10.6.2). As certificate format, CVC according to ISO 7816-8 and X.509 according to ISO/IEC 9594-8, RFC 5280 and RFC 5755 or IEEE 1609.2 shall be used.

— Concept #2 is based on challenge-response procedure without PKI certificates using either asymmetric cryptography with software authentication tokens (see 10.6.3) or symmetric cryptography (see 10.6.3).

NOTE 1 The generation, distribution and storage of cryptographic material is out of scope for this document.

Figure 8 gives an overview over the Authentication service security concepts.

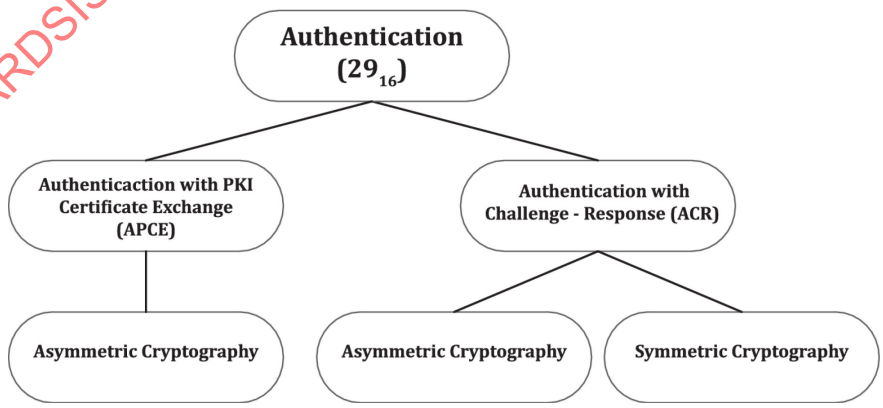


Figure 8 — Overview over the Authentication service security concepts

NOTE 2 Figure 8 shows only the major options of authentication.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

To identify the supported concept in the server, the client can send this service with the SubFunction parameter 'authenticationConfiguration'.

10.6.2 Authentication with PKI Certificate Exchange (APCE)

— The Authentication service is used for authentication, deAuthentication and explicit certificate transmission. The 'authenticationTask' SubFunction parameter identifies the corresponding task to process.

— "deAuthenticate", this SubFunction parameter actively ends the authenticated state.

— "verifyCertificateUnidirectional", this SubFunction parameter initiates the unidirectional Authentication procedure to only authenticate the client against the server.

— "verifyCertificateBidirectional", this SubFunction parameter initiates the bidirectional Authentication procedure to authenticate the client against the server and the server against the client.

— "proofOfOwnership", this SubFunction parameter is used to transmit the proof of ownership data to the client.

— "transmitCertificate", this SubFunction parameter transfers a certificate independently or subsequent to a previous authentication.

Prerequisites:

Different sets of certificates (and corresponding private keys) shall be available at both sides (client and server):

— In case of unidirectional authentication, the client needs a certificate client with its private key, which allows the client to identify itself as a legitimate client. Depending on the trust model of the public key infrastructure (PKI), the server may need a certificate of the certificate authority (CA) which issued and signed the certificate client.

— In case of bidirectional authentication, the client needs a certificate client with its private key, which allows the client to identify itself as a legitimate client. Additionally, the server also needs a certificate server with its private key, which allows the server to identify itself as a legitimate server. Depending on the trust model of the public key infrastructure (PKI), the client and the server may need a certificate of the certificate authority (CA) which issued and signed the certificate client and certificate server.

The authentication process triggered by "verifyCertificateUnidirectional" or "verifyCertificateBidirectional" takes place in two steps as depicted in Figure 9.

Variant 1: Unidirectional authentication:

— Client creates challenge client if used in proof of ownership client (1).

  It is recommended to create the challenge based on ISO/IEC 9798-3 (unilateral, two pass authentication) or as a security-related equivalent challenge to the one as per ISO/IEC 9798-3.

— Client sends its certificate client and, if generated in (1), its challenge client via SubFunction verifyCertificateUnidirectional (2).

— Server verifies the certificate client (3).

— Server creates a challenge server (4).

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

— If session key establishment based on the Ephemeral Diffie-Hellman key agreement is indicated by the client in (2), then the server generates an ephemeral private/public key pair to later derive a session key for further secure communication (5).

NOTE 1 Diffie-Hellman key agreement is necessary if algorithms are used in the certificates which are only usable for signature calculation, but are not usable for a key agreement protocol.

— Server sends the challenge server and, if generated in (5), its ephemeral public key (7).

— If session key establishment based on the Ephemeral Diffie-Hellman key agreement is indicated by the client in (2), then the client also generates an ephemeral private/public key pair to later derive a session key for further secure communication (9).

NOTE 2 Diffie-Hellman key agreement is necessary if algorithms are used in the certificates which are only usable for signature calculation, but are not usable for a key agreement protocol.

— Client calculates proof of ownership client by building an appropriate authentication token whose content to be signed include at least (parts of) the challenge server and, if generated in (9), its ephemeral public key (10).

It is recommended to build the authentication token based on ISO/IEC 9798-3 (unilateral, two pass authentication) or as a security-related equivalent authentication token to the one as per ISO/IEC 9798-3.

— Client sends the proof of ownership client and, if generated in (9), its ephemeral public key via SubFunction proofOfOwnership (11).

— Server verifies the proof of ownership client with the public key from the received certificate client (12).

— If session key establishment is indicated by the client in (2), then the server creates or derives and enables the session key(s) for further secure communication and sets up the session key info (13).

— The server grants access to diagnostic objects according to access rights (14).

— The server responds that the authentication was successful and, if present from (13), sends the session key info (15).

— If session key establishment is indicated by the client in (2), then the client extracts the session key(s) from the session key Info or derives the session key(s) for further secure communication (16).

— If session key establishment is indicated by the client in (2), then the client verifies the session key info using the session key(s) (17).

NOTE 3 Step (17) ensures that the session key establishment is complete and valid.

— If session key establishment is indicated by the client in (2), then the client enables the session key(s) for further secure diagnostic communication (18).

NOTE 4 When using unidirectional authentication, the server does not authenticate against the client. Thus, the client cannot be sure that it is communicating with the correct server.

Variant 2: Bidirectional authentication:

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

— Client creates challenge client (1) and sends it together with its certificate client via SubFunction verifyCertificateBidirectional (2).

— Server verifies the certificate client (3).

— Server creates a challenge server (4).

— If session key establishment based on the Ephemeral Diffie-Hellman key agreement is indicated by the client in (2), then the server generates an ephemeral private/public key pair to later derive a session key for further secure communication (5).

NOTE 5 Diffie-Hellman key agreement is necessary if algorithms are used in the certificates which are only usable for signature calculation, but are not usable for a key agreement protocol.

— Server calculates a proof of ownership server by building an appropriate authentication token whose content to be signed include at least (parts of) the challenge client and, if generated in (5), its ephemeral public key (6) and sends it together with the challenge server, its certificate server and, if generated in (5), its ephemeral public key (7).

It is recommended to build the authentication token based on ISO/IEC 9798-3 (mutual, three pass authentication) or as a security-related equivalent authentication token to the one as per ISO/IEC 9798-3.

— Client verifies the certificate server and the proof of ownership server with the public key from the received certificate server (8).

— If session key establishment based on the Ephemeral Diffie-Hellman key agreement is indicated by the client in (2), then the client also generates an ephemeral private/public key pair to later derive a session key for further secure communication (9).

NOTE 6 Diffie-Hellman key agreement is necessary if algorithms are used in the certificates which are only usable for signature calculation, but aren't usable for a key agreement protocol.

— Client calculates proof of ownership client by building an appropriate authentication token whose content to be signed include at least (parts of) the challenge server and, if generated in (9), its ephemeral public key (10).

It is recommended to build the authentication token based on ISO/IEC 9798-3 (mutual, three pass authentication) or as a security-related equivalent authentication token to the one as per ISO/IEC 9798-3.

— Client sends the proof of ownership client and, if generated in (9), its ephemeral public key via SubFunction proofOfOwnership (11).

— Server verifies the proof of ownership client with the public key from the received certificate client (12).

— If session key establishment is indicated by the client in (2), then the server creates or derives and enables the session key(s) for further secure communication and sets up the session key info (13).

— The server grants access to diagnostic objects according to access rights (14).

— The server responds that the authentication was successful and, if present from (13), sends the session key info (15).

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

— If session key establishment is indicated by the client in (2), then the client extracts the session key(s) from the session key info or derives the session key(s) for further secure communication (16).

— If session key establishment is indicated by the client in (2), then the client verifies the session key info using the session key(s) (17).

NOTE 7 Step (17) ensures that the session key establishment is complete and valid.

— If session key establishment is indicated by the client in (2), then the client enables the session key(s) for further secure diagnostic communication (18).

If each verification is successful, the server shall allow the client to access the diagnostic services the information in the certificate client refer to and respond with a positive response to the client. If the verification fails at any point in this process, the server or the client shall stop the authentication process and send the appropriate response. The client shall display an appropriate message (see external test equipment specification).

NOTE 8 The management of failed attempts (e.g. maximum number of attempts, delay, etc.) is left to the vehicle manufacturer's discretion.

NOTE 9 In case the authentication fails on the client side, especially after the server already accepted the client and set the access rights, the client can optionally send the SubFunction "deAuthenticate" to the server to be sure that the server leaves the authenticated state and refuses further unauthorized requests. Access control is in the responsibility of the vehicle manufacturer.

The session key that is established by any means in this subclause shall be valid at maximum for the time of the authenticated session.

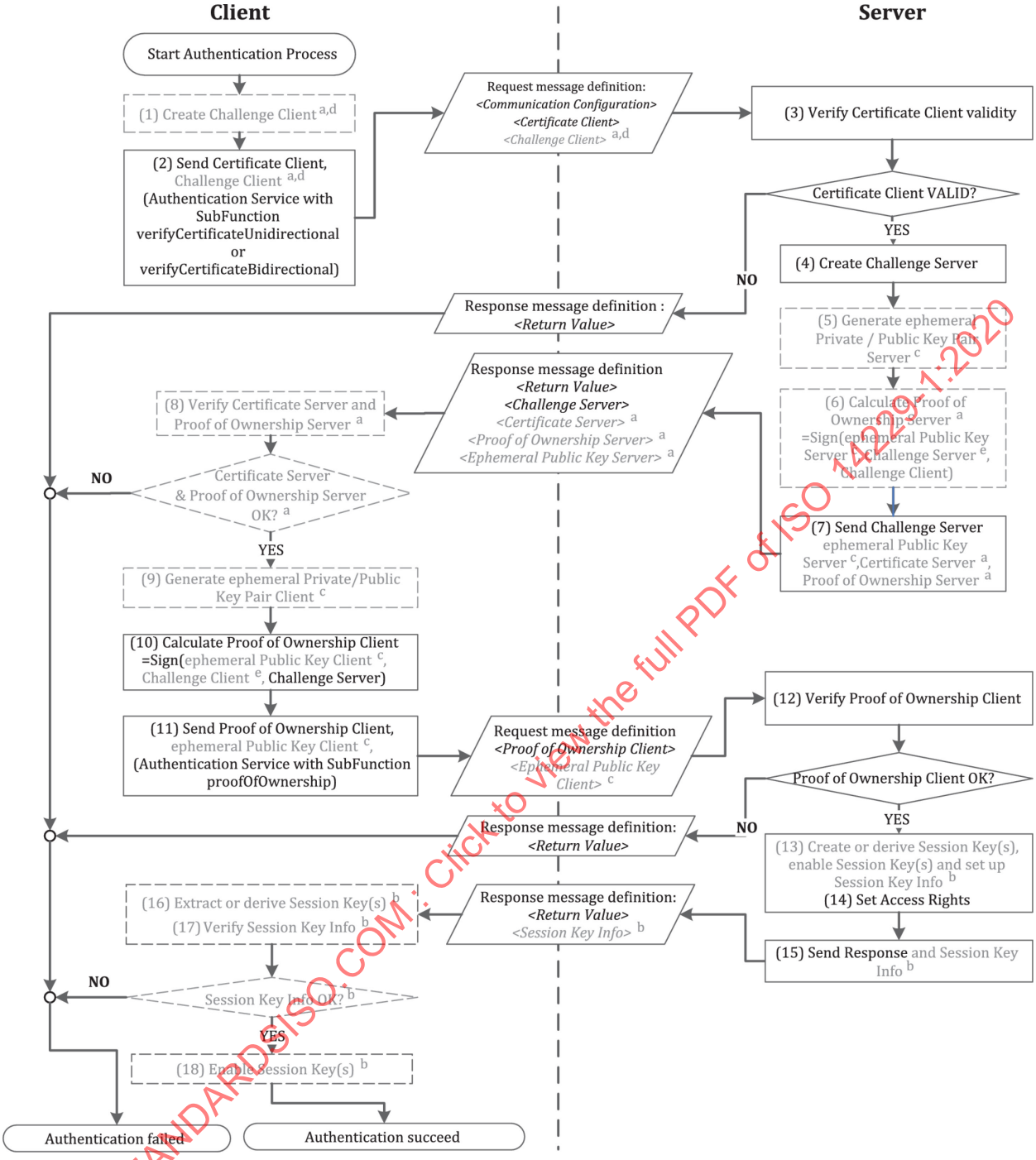
To transmit certificates independently or subsequent to a previous authentication the SubFunction "transmitCertificate" can be used. The intention of this SubFunction is to present the server a certificate for further processing without any challenge response sequence. The certificate can be used for additional rights activation or to proof signed data (with the embedded public key of the certificate). Therefore the data shall be signed with the corresponding private key (data and signature shall be sent independently to the server).

For each use case, such as additional rights activation, a different certificateEvaluationId shall be provided to allow the server to identify the certificate. This SubFunction can be used to support more than one kind of certificate.

NOTE 10 The mechanism to add rights through certificates is left to the vehicle manufacturer's discretion.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved



STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

Key

a When using bidirectional authentication.

b Only when using secure diagnostic communication.

c Only when using Diffie-Hellman key agreement for secure diagnostic communication.

d If challenge client is used in Proof of Ownership client.

e Optionally include challenge server and challenge client in each Proof of Ownership server and client.

Figure 9 — Authentication sequence with PKI Certificate Exchange (APCE)

© ISO 2020 – All rights reserved

10.6.3 Authentication with Challenge-Response (ACR)

Prerequisites:

— In case of using asymmetric cryptography, a client key pair shall exist: the client private key shall be present in the client and the client public key shall be present in the server. In case of the bidirectional authentication an additional server key pair shall exist: the server private key shall be present in the server and the server public key shall be present in the client.

— In case of using symmetric cryptography, a symmetric key shall exist and shall be pre-shared between the client and the server.

Variant 1: Unidirectional authentication:

— The client requests an authentication via SubFunction requestChallengeForAuthentication indicating the algorithm to be used and if session key(s) shall be established (1).

— The server creates the challenge server (2).

— The server sends the challenge server and the indication if additional parameters shall be provided (3).

— The client creates challenge client if used in proof of ownership client (4).

It is recommended to create the challenge based on ISO/IEC 9798-2 or ISO/IEC 9798-4 (unilateral, two pass authentication) or as a security-related equivalent challenge to the one as per ISO/IEC 9798.

— The client calculates the client-side proof of ownership (POWN) as follows (5):

  — In case of using asymmetric cryptography: build an appropriate (vehicle manufacturer specific) token (e.g. based on CVC) content containing token authority, authentication, rights/roles, server-side challenge information and as the case may be client-side challenge information and additional information, compute the token content signature using the client private key and build the client-side authentication token containing the token content and the signature. The resulting client-side authentication token is the client-side POWN.

  — It is recommended to build the authentication token based on ISO/IEC 9798-2 or ISO/IEC 9798-4 (unilateral, two pass authentication) or as a security-related equivalent authentication token to the one as per ISO/IEC 9798.

  — In case of using symmetric cryptography: compute the signature (for example one time signature or HMAC or CMAC or GMAC) over the server-side challenge and as the case may be over client-side challenge information and the additional parameters (such as rights/roles that are predefined by vehicle manufacturer) with a pre-shared symmetric key. The resulting signature is the client-side POWN.

  — It is recommended to build the authentication token based on ISO/IEC 9798-2 or ISO/IEC 9798-4 (unilateral, two pass authentication) or as a security-related equivalent authentication token to the one as per ISO/IEC 9798.

— If additional parameters are indicated by the server in (3), then the client provides appropriate additional parameters in needed additional parameter (6).

— The client sends the client-side POWN, if generated in (4), the challenge client and, if indicated, the needed additional parameter via SubFunction verifyProofOfOwnershipUnidirectional (7).

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

— The server verifies the client-side POWN (8).

— If session key establishment is indicated by the client in (1), then the server creates or derives and enables the session key(s) for further secure communication and sets up the session key info (10).

— The server grants access to diagnostic objects according to access rights (11).

— The server responds that the authentication was successful and, if present, sends the session key info (12).

— If session key establishment is indicated by the client in (1), then the client extracts session key(s) from the session key info or derives session key(s) for further secure communication (14).

— If session key establishment is indicated by the client in (1), then the client verifies the session key info using the session key(s) (15).

NOTE 1 Step (15) ensures that the session key establishment is complete and valid.

— If session key establishment is indicated by the client in (1), then the client enables the session key(s) for further secure diagnostic communication (16).

Variant 2: Bidirectional authentication:

— The client requests an authentication via SubFunction requestChallengeForAuthentication indicating the algorithm to be used and if session key(s) shall be established (1).

— The server creates the challenge server (2).

— The server sends the challenge server and the indication if additional parameters shall be provided (3).

— The client creates the challenge client (4).

— The client calculates the client-side proof of ownership (POWN) as follows (5):

  — In case of using asymmetric cryptography: build an appropriate (vehicle manufacturer specific) token (e.g. based on CVC) content containing token authority, authentication, rights/roles, server-side challenge information and as the case may be client-side challenge information and additional information, compute the token content signature using the client private key and build the client-side authentication token containing the token content and the signature. The resulting client-side authentication token is the client-side POWN.

  — It is recommended to build the authentication token based on ISO/IEC 9798-2 or ISO/IEC 9798-4 (mutual, three pass authentication) or as a security-related equivalent authentication token to the one as per ISO/IEC 9798.

  — In case of using symmetric cryptography: compute the signature (for example one-time signature or HMAC or CMAC or GMAC) over the server-side challenge and as the case may be over client-side challenge information and the additional parameters (such as rights/roles that are predefined by vehicle manufacturer) with a pre-shared symmetric key. The resulting signature is the client-side POWN.

  — It is recommended to build the authentication token based on ISO/IEC 9798-2 or ISO/IEC 9798-4 (mutual, three pass authentication) or as a security-related equivalent authentication token to the one as per ISO/IEC 9798.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

— If additional parameters (vehicle manufacturer specific) are indicated by the server in (3), then the client provides appropriate additional parameters in needed additional parameter (6).

— The client sends the client-side POWN, the challenge client and, if indicated, the needed additional parameter via SubFunction verifyProofOfOwnershipBidirectional (7).

— The server verifies the client-side POWN (8).

— The server calculates the server-side proof of ownership (POWN) as follows (9):

  — In case of using asymmetric cryptography: build an appropriate (vehicle manufacturer specific) token content containing token authority, authentication, client-side challenge information and as the case may be server-side challenge information, compute the token content signature using the server private key and build the server-side authentication token containing the token content and the signature. The resulting server-side authentication token is the server-side POWN.

  — It is recommended to build the authentication token based on ISO/IEC 9798-2 or ISO/IEC 9798-4 (mutual, three pass authentication) or as a security-related equivalent authentication token to the one as per ISO/IEC 9798.

  — In case of using symmetric cryptography: compute the signature (for example one-time signature or HMAC or CMAC or GMAC) over the client-side challenge and as the case may be server-side challenge with a pre-shared symmetric key. The resulting signature is the server-side POWN.

  — It is recommended to build the authentication token based on ISO/IEC 9798-2 or ISO/IEC 9798-4 (mutual, three pass authentication) or as a security-related equivalent authentication token to the one as per ISO/IEC 9798.

— If session key establishment is indicated by the client in (1), then the server creates or derives and enables the session key(s) for further secure communication and sets up the appropriate Session Key Info (10).

— The server grants access to diagnostic objects according to access rights (11).

— The server responds that the authentication was successful and sends the server-side POWN and, if present, the session key info (12).

— The client verifies the server-side POWN (13).

— If session key establishment is indicated by the client in (1), then the client extracts the session key(s) from the session key info or derives the session key(s) for further secure communication (14).

— If session key establishment is indicated by the client in (1), then the client verifies the session key info using the session key(s) (15).

NOTE 2 Step (15) ensures that the session key establishment is complete and valid.

— If session key establishment is indicated by the client in (1), then the client enables the session key(s) for further secure diagnostic communication (16).

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

If the verification fails at any point in this process, the server or the client shall stop the authentication process and send the appropriate response. The client shall display an appropriate message (see external test equipment specification).

NOTE 3 The management of failed attempts (e.g. maximum number of attempts, delay, etc.) is left to the vehicle manufacturer's discretion.

NOTE 4 In case the authentication fails on the client side, especially after the server already accepted the client and set the access rights, the client can optionally send the SubFunction "deAuthenticate" to the server to be sure that the server leaves the authenticated state and refuses further unauthorized requests.

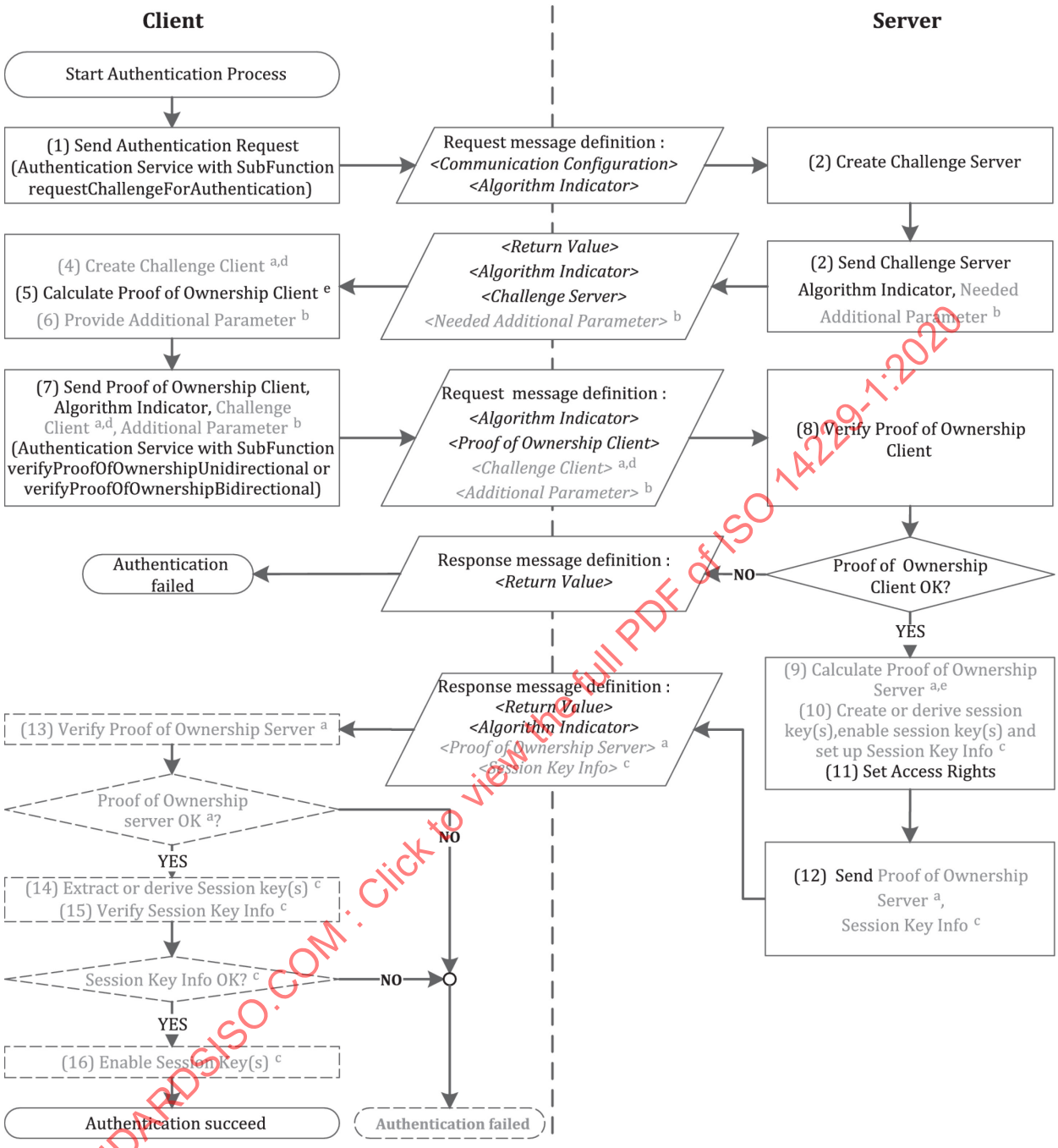
The session key that is established by any means in this subclause shall be valid at maximum for the time of the authenticated session.

NOTE 5 The current granted access to diagnostic objects in step (11) can be changed by performing the ACR procedure again with new rights/roles in step (4). By doing this, the new rights/roles replace the current ones and the granted access changes accordingly.

Figure 10 shows the authentication sequence with challenge-response (ACR).

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved



STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

Key

a When using bidirectional authentication.

b Only if the server needs additional parameter for authentication.

c Only when using secure diagnostic communication.

d If challenge client is used in Proof of Ownership client.

e Optionally include challenge server and challenge client in each Proof of Ownership server and client.

Figure 10 — Authentication sequence with Challenge-Response (ACR)

10.6.4 Common requirements

The authentication is designed to secure applicable diagnostic sessions/functions/services. Therefore the following sequence of services shall be required:

© ISO 2020 – All rights reserved

— Authentication service,

— Any diagnostic service that is secured or restricted by authentication.

For authentication there is no direct relationship to diagnostic session or security level. Once authenticated, the server shall be in an authenticated state which it only shall leave if a security timeout occurs, a mileage offset limit is reached or the authenticated state is intentionally left by the "deAuthenticate" request. Applicable diagnostic services assigned to the corresponding authentication settings shall be accessible as long as the client is authenticated.

There are several possibilities to leave the authenticated state:

— Explicitly: the authentication request shall be called with the SubFunction parameter "deAuthenticate".

— Implicitly using a timeout as fallback: a timer shall be installed. The timer shall start when transitioning to the authenticated state. If the timer times out, then the authenticated state shall be ended. The timeout can be set individually or can be bound to an existing timing parameter (e.g. session layer timing parameter definitions). At least every request message of the same diagnostic protocol received by the transport protocol layer shall keep the authenticated state active and reset the timeout period.

  NOTE 1 The timer value is vehicle manufacturer specific.

  It is recommended that the usage of the timeout is based on a reliable internal time.

— Implicitly using a mileage offset limit as fallback: a mileage monitoring shall be installed. The mileage monitoring shall start when transitioning to the authenticated state. If the mileage limit is reached, then the authenticated state shall be ended.

  NOTE 2 The mileage offset limit is vehicle manufacturer specific.

  It is recommended that the usage of the mileage monitoring is based on reliable mileage information.

The explicit exit condition shall be mandatory, the implementation of at least one of the implicit exit conditions shall be mandatory, both are optionally possible.

If a server supports authentication and is already in an authenticated state when an authentication request message is received from the same client, that server shall stay in the authenticated state until the authentication is again successfully completed. The server shall change its state to the new received authentication information.

Unsuccessful attempts to access an authenticated state shall not prevent other diagnostic communication. An authenticated state shall be linked to a certain diagnostic channel. Multiple clients can be handled on multiple channels with different authentication settings. The basis for this would be a multi-user server system.

Servers which provide security shall support the NRC 34_{16} "authenticationRequired", if a secured service is requested while the server is in a non-authenticated state.

As an option, session keys can be established and used to further secure communication between client and server.

This can be achieved, for example, using one of the following approaches:

— by creation of the session key on the server side and an encrypted transmission to the client, or

— by using an asymmetric key agreement protocol, or

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

— by derivation of existing pre-shared keys on both sides (in case of symmetric cryptography).

10.6.5 Request message

10.6.5.1 Request message definition

Table 65 specifies the request message definition - SubFunction = deAuthenticate.

Table 65 — Request message definition - SubFunction = deAuthenticate

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	Authentication Request SID	M	29_{16}	ARS
#2	SubFunction = [authenticationTask = deAuthenticate]	M	00_{16}	LEV_AT_DA

Table 66 specifies the request message definition - SubFunction = verifyCertificateUnidirectional.

Table 66 — Request message definition - SubFunction = verifyCertificateUnidirectional

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	Authentication Request SID	M	29_{16}	ARS
#2	SubFunction = [authenticationTask = verifyCertificateUnidirectional]	M	01_{16}	LEV_AT_VCU
#3	communicationConfiguration [] = [byte#1]	M	00_{16} to FF_{16}	COCO
	lengthOfCertificateClient [] = [			LOCECL
#4	byte#1 (MSB)	M	00_{16} to FF_{16}	
#5	byte#2 (LSB)]	M	00_{16} to FF_{16}	
	certificateClient [] = [			CECL
#6	byte#1	M	00_{16} to FF_{16}	
:	:	:	:	
#m+5	byte#m]	M	00_{16} to FF_{16}	
	lengthOfChallengeClient [] = [			LOCHCL
#m+6	byte#1 (MSB)	M	00_{16} to FF_{16}	
#m+7	byte#2 (LSB)]	M	00_{16} to FF_{16}	
	challengeClient [] = [			CHCL
#m+8	byte#1	C	00_{16} to FF_{16}	
:	:	:	:	
#n+m+7	byte#n]	C	00_{16} to FF_{16}	

C The presence of this parameter depends on the length information parameter of the lengthOfChallengeClient. If lengthOfChallengeClient is equal to 0000_{16} then the parameter challengeClient will not be transmitted.

Table 67 specifies the request message definition - SubFunction = verifyCertificateBidirectional.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 67 — Request message definition - SubFunction = verifyCertificateBidirectional

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	Authentication Request SID	M	29_{16}	ARS
#2	SubFunction = [authenticationTask = verifyCertificateBidirectional]	M	02_{16}	LEV_AT_VCB
#3	communicationConfiguration [] = [byte#1]	M	00_{16} to FF_{16}	COCO
	lengthOfCertificateClient [] = [			LOCECL
#4	byte#1 (MSB)	M	00_{16} to FF_{16}	
#5	byte#2 (LSB)]	M	00_{16} to FF_{16}	
	certificateClient [] = [			CECL
#6	byte#1	M	00_{16} to FF_{16}	
:	:	:	:	
#m+5	byte#m]	M	00_{16} to FF_{16}	
	lengthOfChallengeClient [] = [			LOCHCL
#m+6	byte#1 (MSB)	M	00_{16} to FF_{16}	
#m+7	byte#2 (LSB)]	M	00_{16} to FF_{16}	
	challengeClient [] = [			CHCL
#m+8	byte#1	M	00_{16} to FF_{16}	
:	:	:	:	
#n+m+7	byte#n]	M	00_{16} to FF_{16}	

Table 68 specifies the request message definition - SubFunction = proofOfOwnership.

Table 68 — Request message definition - SubFunction = proofOfOwnership

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	Authentication Request SID	M	29_{16}	ARS
#2	SubFunction = [authenticationTask = proofOfOwnership]	M	03_{16}	LEV_AT_POWN
	lengthOfProofOfOwnershipClient[] = [			LPOWNCL
#3	byte#1 (MSB)	M	00_{16} to FF_{16}	
#4	byte#2 (LSB)]	M	00_{16} to FF_{16}	
	proofOfOwnershipClient [] = [			POWNCL
#5	byte#1	M	00_{16} to FF_{16}	
:	:	:	:	
#m+4	byte#m]	M	00_{16} to FF_{16}	
	lengthOfEphemeralPublicKeyClient [] = [			LOEPKCL
#m+5	byte#1 (MSB)	M	00_{16} to FF_{16}	
#m+6	byte#2 (LSB)]	M	00_{16} to FF_{16}	

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
	ephemeralPublicKeyClient [] = [			EPKCL
#m+7	byte#1	C	00_{16} to FF_{16}	
:	:	:	:	
#n+m+6	byte#n]	C	00_{16} to FF_{16}	
C The presence of this parameter depends on the length information parameter of the lengthOfEphemeralPublicKeyClient. If lengthOfEphemeralPublicKeyClient is equal to 0000_{16} then the parameter ephemeralPublicKeyClient will not be transmitted.				

Table 69 specifies the request message definition - SubFunction = transmitCertificate.

Table 69 — Request message definition - SubFunction = transmitCertificate

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	Authentication Request SID	M	29_{16}	ARS
#2	SubFunction = [			LEV_AT_TC
	authenticationTask = transmitCertificate]	M	04_{16}	
#3	certificateEvaluationId	M	00_{16} to FF_{16}	CEID
#4			00_{16} to FF_{16}	
	lengthOfCertificateData [] = [			LOCEDA
#5	byte#1 (MSB)	M	00_{16} to FF_{16}	
#6	byte#2 (LSB)]	M	00_{16} to FF_{16}	
	certificateData [] = [			CEDA
#7	byte#1	M	00_{16} to FF_{16}	
:	:	:	:	
#m+6	byte#m]	M	00_{16} to FF_{16}	

Table 70 specifies the request message definition - SubFunction = requestChallengeForAuthentication.

Table 70 — Request message definition - SubFunction = requestChallengeForAuthentication

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	Authentication Request SID	M	29_{16}	ARS
#2	SubFunction = [authenticationTask = requestChallengeForAuthentication]	M	05_{16}	LEV_AT_RCFA
#3	communicationConfiguration [] = [byte#1]	M	00_{16} to FF_{16}	COCO
	algorithmIndicator [] = [			AI
#4	byte#1	M	00_{16} to FF_{16}	
:	:	:	:	
#19	byte#16]	M	00_{16} to FF_{16}	

Table 71 specifies the request message definition - SubFunction = verifyProofOfOwnershipUnidirectional.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 71 — Request message definition - SubFunction = verifyProofOfOwnershipUnidirectional

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	Authentication Request SID	M	29_{16}	ARS
#2	SubFunction = [authenticationTask = verifyProofOfOwnershipUnidirectional]	M	06_{16}	LEV_AT_VPOWNU
	algorithmIndicator [] = [			AI
#3	byte#1	M	00_{16} to FF_{16}	
:	:	:	:	
#18	byte#16]	M	00_{16} to FF_{16}	
	lengthOfProofOfOwnershipClient [] = [			LPOWNCL
#19	byte#1 (MSB)	M	00_{16} to FF_{16}	
#20	byte#2 (LSB)]	M	00_{16} to FF_{16}	
	proofOfOwnershipClient [] = [			POWNCL
#21	byte#1	M	00_{16} to FF_{16}	
:	:	:	:	
#m+20	byte#m]	M	00_{16} to FF_{16}	
	lengthOfChallengeClient [] = [			LOCHCL
#m+21	byte#1 (MSB)	M	00_{16} to FF_{16}	
#m+22	byte#2 (LSB)]	M	00_{16} to FF_{16}	
	challengeClient [] = [			CHCL
#m+23	byte#1	C1	00_{16} to FF_{16}	
:	:	:	:	
#n+m+22	byte#n]	C1	00_{16} to FF_{16}	
	lengthOfAdditionalParameter [] = [			LOAP
#n+m+23	byte#1(MSB)	M	00_{16} to FF_{16}	
#n+m+24	byte#2 (LSB)]	M	00_{16} to FF_{16}	
	additionalParameter [] = [			AP
#n+m+25	byte#1	C2	00_{16} to FF_{16}	
:	:	:	:	
#o+n+m+24	byte#o]	C2	00_{16} to FF_{16}	

C1 The presence of this parameter depends on the length information parameter of the lengthOfChallengeClient. If lengthOfChallengeClient is equal to 0000_{16} then the parameter challengeClient will not be transmitted

C2 The presence of this parameter depends on the length information parameter of the lengthOfAdditionalParameter. If lengthOfAdditionalParameter is equal to 0000_{16} then the parameter additionalParameter will not be transmitted.

The value of algorithmIndicator in the request message definition - SubFunction = verifyProofOfOwnershipUnidirectional (Table 71) shall be the same as the value of algorithmIndicator in the request message definition - SubFunction = requestChallengeForAuthentication (Table 70).

Table 72 specifies the request message definition - SubFunction = verifyProofOfOwnershipBidirectional.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 72 — Request message definition - SubFunction = verifyProofOfOwnershipBidirectional

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	Authentication Request SID	M	29_{16}	ARS
#2	SubFunction = [authenticationTask = verifyProofOfOwnershipBidirectional]	M	07_{16}	LEV_AT_VPOWNB
#3 : #18	algorithmIndicator [] = [byte#1 : byte#16]	M : M	00_{16} to FF_{16} : 00_{16} to FF_{16}	AI
#19 #20	lengthOfProofOfOwnershipClient [] = [byte#1 (MSB) byte#2 (LSB)]	M M	00_{16} to FF_{16} 00_{16} to FF_{16}	LPOWNCL
#21 : #m+20	proofOfOwnershipClient [] = [byte#1 : byte#m]	M : M	00_{16} to FF_{16} : 00_{16} to FF_{16}	POWNCL
#m+21 #m+22	lengthOfChallengeClient [] = [byte#1 (MSB) byte#2 (LSB)]	M M	00_{16} to FF_{16} 00_{16} to FF_{16}	LOCHCL
#m+23 : #n+m+22	challengeClient [] = [byte#1 : byte#n]	M : M	00_{16} to FF_{16} : 00_{16} to FF_{16}	CHCL
#n+m+23 #n+m+24	lengthOfAdditionalParameter [] = [byte#1(MSB) byte#2 (LSB)]	M M	00_{16} to FF_{16} 00_{16} to FF_{16}	LOAP
#n+m+25 : #o+n+m+24	additionalParameter [] = [byte#1 : byte#o]	C : C	00_{16} to FF_{16} : 00_{16} to FF_{16}	AP

C The presence of this parameter depends on the length information parameter of the lengthOfAdditionalParameter. If lengthOfAdditionalParameter is equal to 0000_{16} then the parameter additionalParameter will not be transmitted.

The value of algorithmIndicator in the request message definition - SubFunction = verifyProofOfOwnershipBidirectional (Table 72) shall be the same as the value of algorithmIndicator in the request message definition - SubFunction = requestChallengeForAuthentication (Table 70).

Table 73 specifies the request message definition - SubFunction = authenticationConfiguration.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 73 — Request message definition - SubFunction = authenticationConfiguration

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	Authentication Request SID	M	29_{16}	ARS
#2	SubFunction = [authenticationTask = authenticationConfiguration]	M	08_{16}	LEV_AT_AC

10.6.5.2 Request message SubFunction parameter $Level (LEV_) definition

The SubFunction parameter authenticationTask indicates to the server the explicit task which it should conduct (suppressPosRspMsgIndicationBit (bit 7) not shown). Table 74 specifies the request message SubFunction parameter definition.

Table 74 — Request message SubFunction parameter definition

Bit 6 to 0	Description	Cvt	Mnemonic
00_{16}	**deAuthenticate** Request to leave the authenticated state.	M	DA
01_{16}	**verifyCertificateUnidirectional** Initiate Authentication by verifying the Certificate.	C_1	VCU
02_{16}	**verifyCertificateBidirectional** Initiate Authentication by verifying the Certificate and generating a Proof of Ownership from the server.	C_1	VCB
03_{16}	**proofOfOwnership** Verify the Proof of Ownership from the client.	C_1	POWN
04_{16}	**transmitCertificate** Verify Certificate and extract information from Certificate to handle it according to its contents.	C_1	TC
05_{16}	**requestChallengeForAuthentication** Initiate the Authentication process by requesting server to output a challenge.	C_2	RCFA
06_{16}	**verifyProofOfOwnershipUnidirectional** Request server to verify the POWN for unidirectional authentication.	C_2	VPOWNU
07_{16}	**verifyProofOfOwnershipBidirectional** Request server to verify the client side POWN and provide server-side POWN for bidirectional authentication.	C_2	VPOWNB
08_{16}	**authenticationConfiguration** Indicates the provided authentication configuration of the server.	M	AC
09_{16} to $7F_{16}$	**ISOSAEReserved** This value is reserved by this document for future definition.	M	ISOSAERESRVD
C_1 Only if authentication with PKI Certificate Exchange (APCE) is used. C_2 Only if authentication with Challenge-Response (ACR) is used.			

10.6.5.3 Request message data-parameter definition

Table 75 specifies the data-parameters of the request message.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

Table 75 — Request message data-parameter definition

Definition
communicationConfiguration Configuration information about how to proceed with security in further diagnostic communication after the Authentication. NOTE 1 This information is linked to the presence and the contents of the response message data-parameter sessionKeyInfo. The format of this parameter as well as the creation of the session key(s) and the computation of the proof value(s) are up to the vehicle manufacturer choice.
lengthOfCertificateClient Length parameter for certificateClient.
certificateClient The Certificate to verify.
certificateEvaluationId A unique ID to identify the evaluation type of the transmitted certificate. The value of this parameter is vehicle manufactuer specific. Subsequent diagnostic requests with the same evaluationTypeId will overwrite the certificate data of the previous requests.
lengthOfChallengeClient Length parameter for challengeClient.
challengeClient The challenge contains vehicle manufacturer specific formatted client data (likely containing randomized information) or is a random number. This parameter record is to transmit the challenge to the server.
lengthOfProofOfOwnershipClient Length parameter for proofOfOwnershipClient.
proofOfOwnershipClient Proof of Ownership of the previous given challenge to be verified by the server.
lengthOfEphemeralPublicKeyClient Length parameter for ephemeralPublicKeyClient, The value of this field shall be 0000_{16} if no private/public key pair is present.
ephemeralPublicKeyClient Ephemeral public key generated by the client for Diffie-Hellman key agreement.
lengthOfCertificateData Length parameter for certificateData.
certificateData The Certificate to verify.
algorithmIndicator Indicates the algorithm used in the generating and verifying Proof of Ownership (POWN), which further determines the parameters used in the algorithm and possibly the session key creation mode. This field is a 16 byte value containing the BER encoded OID value of the algorithm used. The value is left aligned and right padded with zero up to 16 bytes. NOTE 2 Algorithm OIDs can be looked up in the OID repository http://oid-info.com.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Definition
lengthOfAdditionalParameter
Length parameter for additionalParameter.
additionalParameter The additional parameter is provided to the server if the server indicates as neededAdditionalParameter in Table 85.

10.6.6 Positive response message

10.6.6.1 Positive response message definition

Table 76 specifies the response message definition - SubFunction = deAuthenticate.

Table 76 — Response message definition - SubFunction = deAuthenticate

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	Authentication Response SID	M	69_{16}	ARS
#2	SubFunction = [authenticationTask = deAuthenticate]	M	00_{16}	LEV_AT_DA
#3	returnValue [] = [authenticationReturnParameter]	M[a]	00_{16} to FF_{16}	RV

[a] AuthenticationReturnParameter shall be implemented as defined in B.5.

Table 77 specifies the response message definition - SubFunction = verifyCertificateUnidirectional.

Table 77 — Response message definition - SubFunction = verifyCertificateUnidirectional

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	Authentication Response SID	M	69_{16}	ARS
#2	SubFunction = [authenticationTask = verifyCertificateUnidirectional]	 M	 01_{16}	LEV_AT_VCU
#3	returnValue [] = [authenticationReturnParameter]	M[a]	00_{16} to FF_{16}	RV
 #4 #5	lengthOfChallengeServer [] = [byte#1 (MSB) byte#2 (LSB)]	 M M	 00_{16} to FF_{16} 00_{16} to FF_{16}	LOCHSE
 #6 : #m+5	challengeServer [] = [byte#1 : byte#m]	 M : M	 00_{16} to FF_{16} : 00_{16} to FF_{16}	CHSE
 #m+6 #m+7	lengthOfEphemeralPublicKeyServer [] = [byte#1 (MSB) byte#2 (LSB)]	 M M	 00_{16} to FF_{16} 00_{16} to FF_{16}	LOEPKSE
 #m+8 : #n+m+7	ephemeralPublicKeyServer [] = [byte#1 : byte#n]	 C : C	 00_{16} to FF_{16} : 00_{16} to FF_{16}	EPKSE

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
[a] AuthenticationReturnParameter shall be implemented as defined in B.5. C The presence of this parameter depends on the length information parameter of the lengthOfEphemeralPublicKeyServer. If lengthOfEphemeralPublicKeyServer is equal to 0000_{16} then the parameter ephemeralPublicKeyServer shall not be transmitted.				

Table 78 specifies the response message definition - SubFunction = verifyCertificateBidirectional.

Table 78 — Response message definition - SubFunction = verifyCertificateBidirectional

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	Authentication Response SID	M	69_{16}	ARS
#2	SubFunction = [authenticationTask = verifyCertificateBidirectional]	M	02_{16}	LEV_AT_VCB
#3	returnValue [] = [authenticationReturnParameter]	M[a]	00_{16} to FF_{16}	RV
	lengthOfChallengeServer [] = [			LOCHSE
#4	byte#1 (MSB)	M	00_{16} to FF_{16}	
#5	byte#2 (LSB)]	M	00_{16} to FF_{16}	
	challengeServer [] = [			CHSE
#6	byte#1	M	00_{16} to FF_{16}	
:	:	:	:	
#m+5	byte#m]	M	00_{16} to FF_{16}	
	lengthOfCertificateServer [] = [			LOCESE
#m+6	byte#1 (MSB)	M	00_{16} to FF_{16}	
#m+7	byte#2 (LSB)]	M	00_{16} to FF_{16}	
	certificateServer [] = [			CESE
#m+8	byte#1	M	00_{16} to FF_{16}	
:	:	:	:	
#n+m+7	byte#n]	M	00_{16} to FF_{16}	
	lengthOfProofOfOwnershipServer [] = [			LPOWNSE
#n+m+8	byte#1 (MSB)	M	00_{16} to FF_{16}	
#n+m+9	byte#2 (LSB)]	M	00_{16} to FF_{16}	
	proofOfOwnershipServer [] = [			POWNSE
#n+m+10	byte#1	M	00_{16} to FF_{16}	
:	:	:	:	
#o+n+m+9	byte#o]	M	00_{16} to FF_{16}	
	lengthOfEphemeralPublicKeyServer [] = [			LOEPKSE
#o+n+m+10	byte#1 (MSB)	M	00_{16} to FF_{16}	
#o+n+m+11	byte#2 (LSB)]	M	00_{16} to FF_{16}	

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
	ephemeralPublicKeyServer [] = [			EPKSE
#o+n+m+12	byte#1	C	00_{16} to FF_{16}	
:	:	:	:	
#p+o+n+m+11	byte#p]	C	00_{16} to FF_{16}	

[a] AuthenticationReturnParameter shall be implemented as defined in B.5.

C The presence of this parameter depends on the length information parameter of the lengthOfEphemeralPublicKeyServer. If lengthOfEphemeralPublicKeyServer is equal to 0000_{16} then the parameter ephemeralPublicKeyServer shall not be transmitted.

Table 79 specifies the response message definition - SubFunction = proofOfOwnership.

Table 79 — Response message definition - SubFunction = proofOfOwnership

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	Authentication Response SID	M	69_{16}	ARS
#2	SubFunction = [authenticationTask = proofOfOwnership]	M	03_{16}	LEV_AT_POWN
#3	returnValue [] = [authenticationReturnParameter]	M[a]	00_{16} to FF_{16}	RV
	lengthOfSessionKeyInfo [] = [			LOSKI
#4	byte#1 (MSB)	M	00_{16} to FF_{16}	
#5	byte#2 (LSB)]	M	00_{16} to FF_{16}	
	sessionKeyInfo [] = [			SKI
#6	byte#1	C	00_{16} to FF_{16}	
:	:	:	:	
#m+5	byte#m]	C	00_{16} to FF_{16}	

[a] AuthenticationReturnParameter shall be implemented as defined in B.5.

C The presence of this parameter depends on the length information parameter of the lengthOfSessionKeyInfo. If lengthOfSessionKeyInfo is equal to 0000_{16} then the parameter sessionKeyInfo shall not be transmitted.

Table 80 specifies the response message definition - SubFunction = transmitCertificate.

Table 80 — Response message definition - SubFunction = transmitCertificate

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	Authentication Response SID	M	69_{16}	ARS
#2	SubFunction = [authenticationTask = transmitCertificate]	M	04_{16}	LEV_AT_TC
#3	returnValue [] = [authenticationReturnParameter]	M[a]	00_{16} to FF_{16}	RV

[a] AuthenticationReturnParameter shall be implemented as defined in B.5.

Table 81 specifies the response message definition - SubFunction = requestChallengeForAuthentication.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 81 — Response message definition - SubFunction = requestChallengeForAuthentication

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	Authentication Response SID	M	69_{16}	ARS
#2	SubFunction = [authenticationTask = requestChallengeForAuthentication]	M	05_{16}	LEV_AT_RCFA
#3	returnValue [] = [authenticationReturnParameter]	M[a]	00_{16} to FF_{16}	RV
#4 : #19	algorithmIndicator [] = [byte#1 : byte#16]	M : M	00_{16} to FF_{16} : 00_{16} to FF_{16}	AI
#20 #21	lengthOfChallengeServer [] = [byte#1 (MSB) byte#2 (LSB)]	M M	00_{16} to FF_{16} 00_{16} to FF_{16}	LOCHSE
#22 : #m+21	challengeServer [] = [byte#1 : byte#m]	M : M	00_{16} to FF_{16} : 00_{16} to FF_{16}	CHSE
#m+22 #m+23	lengthOfNeededAdditionalParameter [] = [byte#1(MSB) byte#2 (LSB)]	M M	00_{16} to FF_{16} 00_{16} to FF_{16}	LONAP
#m+24 : #n+m+23	neededAdditionalParameter [] = [byte#1 : byt #n]	C : C	00_{16} to FF_{16} : 00_{16} to FF_{16}	NAP

[a] AuthenticationReturnParameter shall be implemented as defined in B.5.

C The presence of this parameter depends on the length information parameter of the lengthOfNeededAdditionalParameter. If lengthOfNeededAdditionalParameter is equal to 0000_{16} then the parameter neededAdditionalParameter shall not be transmitted.

The value of algorithmIndicator in the response message definition - SubFunction = requestChallengeForAuthentication (Table 81) shall be the same as the value of algorithmIndicator in the request message definition - SubFunction = requestChallengeForAuthentication (Table 70).

Table 82 specifies the response message definition - SubFunction = verifyProofOfOwnershipUnidirectional.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 82 — Response message definition - SubFunction = verifyProofOfOwnershipUnidirectional

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	Authentication Response SID	M	69_{16}	ARS
#2	SubFunction = [authenticationTask = verifyProofOfOwnershipUnidirectional]	M	06_{16}	LEV_AT_VPOWNU
#3	returnValue [] = [authenticationReturnParameter]	M_1	00_{16} to FF_{16}	RV
#4 : #19	algorithmIndicator [] = [byte#1 : byte#16]	M : M	00_{16} to FF_{16} : 00_{16} to FF_{16}	AI
#20 #21	lengthOfSessionKeyInfo [] = [byte#1 (MSB) byte#2 (LSB)]	M M	00_{16} to FF_{16} 00_{16} to FF_{16}	LOSKI
#22 : #m+21	sessionKeyInfo [] = [byte#1 : byte#m]	C : C	00_{16} to FF_{16} : 00_{16} to FF_{16}	SKI

M_1: AuthenticationReturnParameter shall be implemented as defined in B.5.

C: The presence of this parameter depends on the length information parameter of the lengthOfSessionKeyInfo. If lengthOfSessionKeyInfo is equal to 0000_{16} then the parameter sessionKeyInfo shall not be transmitted.

The value of algorithmIndicator in the response message definition - SubFunction = verifyProofOfOwnershipBidirectional (Table 82) shall be the same as the value of algorithmIndicator in the request message definition - SubFunction = requestChallengeForAuthentication (Table 70).

Table 83 specifies the response message definition - SubFunction = verifyProofOfOwnershipBidirectional.

Table 83 — Response message definition - SubFunction = verifyProofOfOwnershipBidirectional

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	Authentication Response SID	M	69_{16}	ARS
#2	SubFunction = [authenticationTask = verifyProofOfOwnershipBidirectional]	M	07_{16}	LEV_AT_VPOWNB
#3	returnValue [] = [authenticationReturnParameter]	M_1	00_{16} to FF_{16}	RV
#4 : #19	algorithmIndicator [] = [byte#1 : byte#16]	M : M	00_{16} to FF_{16} : 00_{16} to FF_{16}	AI

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
	lengthOfProofOfOwnershipServer [] = [			LPOWNSE
#20	byte#1 (MSB)	M	00_{16} to FF_{16}	
#21	byte#2 (LSB)]	M	00_{16} to FF_{16}	
	proofOfOwnershipServer [] = [			POWNSE
#22	byte#1	M	00_{16} to FF_{16}	
:	:	:	:	
#m+21	byte#m]	M	00_{16} to FF_{16}	
	lengthOfSessionKeyInfo [] = [			LOSKI
#m+22	byte#1 (MSB)	M	00_{16} to FF_{16}	
#m+23	byte#2 (LSB)]	M	00_{16} to FF_{16}	
	sessionKeyInfo [] = [			SKI
#m+24	byte#1	C	00_{16} to FF_{16}	
:	:	:	:	
#n+m+23	byte#n]	C	00_{16} to FF_{16}	
M_1: AuthenticationReturnParameter shall be implemented as defined in B.5.				
C: The presence of this parameter depends on the length information parameter of the lengthOfSessionKeyInfo. If lengthOfSessionKeyInfo is equal to 0000_{16} then the parameter sessionKeyInfo shall not be transmitted.				

The value of algorithmIndicator in the response message definition - SubFunction = verifyProofOfOwnershipBidirectional (Table 83) shall be the same as the value of algorithmIndicator in the request message definition - SubFunction = requestChallengeForAuthentication (Table 70).

Table 84 specifies the response message definition - SubFunction = authenticationConfiguration.

Table 84 — Response message definition - SubFunction = authenticationConfiguration

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	Authentication Response SID	M	69_{16}	ARS
#2	SubFunction = [authenticationTask = authenticationConfiguration]	M	08_{16}	LEV_AT_AC
#3	returnValue [] = [authenticationReturnParameter]	M_1	00_{16} to FF_{16}	RV
M_1: AuthenticationReturnParameter shall be implemented as defined in B.5.				

10.6.6.2 Positive response message data-parameter definition

Table 85 specifies the data-parameters of the response message.

Table 85 — Response message data-parameter definition

Definition
authenticationTask This parameter is an echo of the authenticationTask from the request message.
returnValue This parameter returns the result of the procedure on the server. The parameter details shall be implemented as defined in B.5.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Definition
lengthOfChallengeServer Length parameter for the following challenge.
challengeServer The challenge contains vehicle manufacturer specific formatted server data (eventually containing randomized information) or is a random number. This parameter record is to transmit the challenge to the client.
lengthOfEphemeralPublicKeyServer Length parameter for ephemeralPublicKeyServer. The value of this field shall be 0000_{16} if no private/public key pair is present.
ephemeralPublicKeyServer Ephemeral public key generated by the server for Diffie-Hellman key agreement.
lengthOfCertificateServer Length parameter for the following Certificate.
certificateServer The Certificate to verify.
lengthOfProofOfOwnershipServer Length parameter for the following Proof of Ownership.
proofOfOwnershipServer Proof of Ownership to be verified by the client.
lengthOfSessionKeyInfo Length parameter for the following session key information, if it is present. The value of this field shall be 0000_{16} if no session key information is present.
sessionKeyInfo If present, this value shall contain session key information, e.g. the encrypted session key(s) for securing further communication in the actual session and/or proof value(s) (e.g. a hash value) for the validation of the session key(s) on the client side. NOTE 1 This information is linked to the contents of the request message data-parameter communicationConfiguration. The format of this parameter as well as the creation of the session key(s) and the computation of the proof value(s) are up to the vehicle manufacturer choice.
algorithmIndicator
Indicates the algorithm used in the generating and verifying Proof of Ownership (POWN), which further determines the parameters used in the algorithm and possibly the session key creation mode. This field is a 16 byte value containing the BER encoded OID value of the algorithm used. The value is left aligned and right padded with zero up to 16 bytes. NOTE 2 Algorithm OIDs can be looked up in the OID repository http://oid-info.com.
lengthOfNeededAdditionalParameter If the server needs an additional parameter from the client for refining authentication, ensuring rights, or even derivation from symmetric key hierarchy, the server can use this field to “request” the client to send this additional parameter. The value of this field shall be 0000_{16} if no additional parameter is needed.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Definition
neededAdditionalParameter Indicate what additional parameters, if needed, are expected by the server.

10.6.7 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 86. The listed negative responses shall be used if the error scenario applies to the server.

Table 86 — 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 Authentication are not met.	CNC
24_{16}	**requestSequenceError** This NRC shall be returned in case of the following scenarios: — If the 'proofOfOwnership' SubFunction is received without first successfully processing either a 'verifyCertificateUnidirectional' or 'verifyCertificateBidirectional' request message, or — If either the 'verifyProofOfOwnershipUnidirectional' or 'verifyProofOfOwnershipBidirectional' SubFunction is received without first successfully processing a 'requestChallengeForAuthentication' request message.	RSE

The evaluation sequence is documented in Figure 11.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

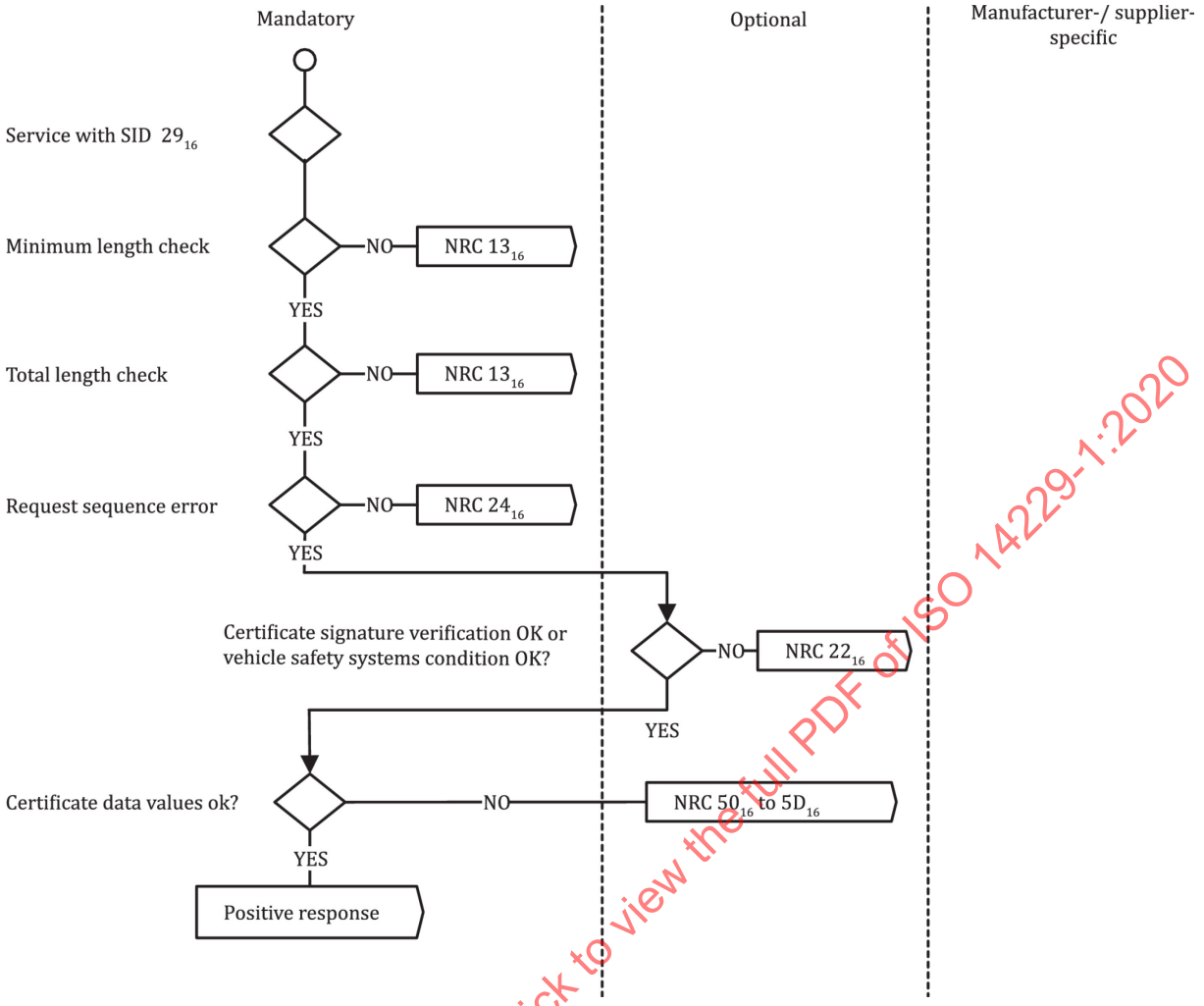


Figure 11 — Evaluation sequence for Authentication service NRCs

NOTE The detailed NRCs 50_{16} to $5D_{16}$ can be used. Alternatively, the general negative response code 10_{16} can be used.

10.6.8 Message flow example(s) Authentication

10.6.8.1 Example #1- Unidirectional Authentication with PKI Certificate Exchange without session key establishment (happy path)

10.6.8.1.1 Assumptions

For the below given message flow the following conditions apply:

— Step #1: This step is optional: client requests the authentication configuration of the server

— Step #2: No secure communication for further communication - communicationConfiguration: 00_{16}

— Step #2: Length of Certificate Client (2 bytes): $01F4_{16}$

— Step #2: Certificate Client (500 bytes)

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

— Step #2: Parameter returnValue in the positive response indicating Certificate verified, Ownership verification necessary: 11_{16}

— Step #2: Length of the Challenge Client (2 bytes): 0020_{16}

— Step #2: Challenge Client: 32 bytes random number

— Step #2: Length of the Challenge Server (2 bytes): 0040_{16}

— Step #2: Challenge Server: 32 bytes ECU identification concatenated with 32 bytes random number

— Step #3: Length of Proof of Ownership (2 bytes): 0150_{16}

— Step #3: Proof of Ownership (336 bytes)

— Step #3: Parameter returnValue in the positive response indicating Ownership verified, Authentication complete: 12_{16}

— Step #3: Length of session key information (2 bytes): 0000_{16}

— Step #4: Diagnostic service ECUReset (11_{16}) is a secured service

10.6.8.1.2 Step #1: Request Authentication Configuration

Table 87 specifies the Authentication Configuration request message flow example #1 - step #1.

Table 87 — Unidirectional Authentication with PKI Certificate Exchange without session key establishment request message flow example #1 – step #1

Message direction	**client -> server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	Authentication Request SID	29_{16}	ARS
#2	authenticationTask = authenticationConfiguration	08_{16}	LEV_AT_AC

Table 88 specifies the Authentication Configuration positive response message flow example #1 - step #1.

Table 88 — Unidirectional Authentication with PKI Certificate Exchange without session key establishment positive response message flow example #1 - step #1

Message direction	**server -> client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	Authentication Response SID	69_{16}	ARS
#2	authenticationTask = authenticationConfiguration	08_{16}	LEV_AT_AC
#3	returnValue = AuthenticationConfiguration APCE	02_{16}	RV_ ACAPCE

10.6.8.1.3 Step #2: Send Certificate Client

Table 89 specifies the Authentication request message flow example #1 - step #2.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 89 — Unidirectional Authentication with PKI Certificate Exchange without session key establishment request message flow example #1 – step #2

Message direction	**client -> server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	Authentication Request SID	29_{16}	ARS
#2	authenticationTask = verifyCertificateUnidirectional	01_{16}	LEV_AT_VCU
#3	communicationConfiguration = no secure communication	00_{16}	COCO
#4	lengthOfCertificateClient [byte#1]	01_{16}	LOCECL
#5	lengthOfCertificateClient [byte#2]	$F4_{16}$	
#6	certificateClient [byte#1]	30_{16}	CECL
:	:	:	
#505	certificateClient [byte#500]	AD_{16}	
#506	lengthOfChallengeClient [byte#1]	00_{16}	LOCHCL
#507	lengthOfChallengeClient [byte#2]	20_{16}	
#508	challengeClient [byte#1]	AA_{16}	CHCL
:	:	:	
#539	challengeClient [byte#32]	44_{16}	

Table 90 specifies the Authentication positive response message flow example #1 - step #2.

Table 90 — Unidirectional Authentication with PKI Certificate Exchange without session key establishment positive response message flow example #1 - step #2

Message direction	**server -> client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	Authentication Response SID	69_{16}	ARS
#2	authenticationTask = verifyCertificateUnidirectional	01_{16}	LEV_AT_VCU
#3	returnValue = Certificate verified, Ownership verification necessary	11_{16}	RV_CVOVN
#4	lengthOfChallengeServer [byte#1]	00_{16}	LOCHSE
#5	lengthOfChallengeServer [byte#2]	40_{16}	
#6	challengeServer [byte#1]	AA_{16}	CHSE
:	:	:	
#69	challengeServer [byte#64]	44_{16}	
#70	lengthOfEphemeralPublicKeyServer [byte#1]	00_{16}	LOEPKSE
#71	lengthOfEphemeralPublicKeyServer [byte#2]	00_{16}	

10.6.8.1.4 Step #3: Validate the Proof of Ownership

Table 91 specifies the Authentication request message flow example #1 - step #3.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 91 — Unidirectional Authentication with PKI Certificate Exchange without session key establishment request message flow example #1 - step #3

Message direction	**client -> server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	Authentication Request SID	29_{16}	ARS
#2	authenticationTask = proofOfOwnership	03_{16}	LEV_AT_POWN
#3	lengthOfProofOfOwnershipClient [byte#1]	01_{16}	LPOWNCL
#4	lengthOfProofOfOwnershipClient [byte#2]	50_{16}	
#5	proofOfOwnershipClient [byte#1]	$7F_{16}$	POWNCL
:	:	:	
#340	proofOfOwnershipClient [byte#336]	$B7_{16}$	
#341	lengthOfEphemeralPublicKeyClient [byte#1]	00_{16}	LOEPKCL
#342	lengthOfEphemeralPublicKeyClient [byte#2]	00_{16}	

Table 92 specifies the Authentication positive response message flow example #1 - step #3.

Table 92 — Unidirectional Authentication with PKI Certificate Exchange without session key establishment positive response message flow example #1 - step #3

Message direction	**server -> client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	Authentication Response SID	69_{16}	ARS
#2	authenticationTask = proofOfOwnership	03_{16}	LEV_AT_POWN
#3	returnValue = Ownership verified, Authentication complete	12_{16}	RV_OVAC
#4	lengthOfSessionKeyInfo [byte#1]	00_{16}	LOSKI
#5	lengthOfSessionKeyInfo [byte#2]	00_{16}	

10.6.8.1.5 Step #4: Attempt to send a random secured service

Table 93 specifies the ECUReset request message after successful Authentication request message attempt - example #1 - step #4.

Table 93 — ECUReset request message after successful Authentication request message attempt – example #1 - step #4

Message direction	**client -> server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ECUReset Request SID	11_{16}	ER
#2	ResetType = hardReset	01_{16}	RT_HR

Table 94 specifies the ECUReset response message after successful Authentication request message attempt example #1 - step #4.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 94 — ECUReset response message after successful Authentication request message attempt – example #1 - step #4

Message direction		server -> client	
Message type		Response	
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	ECUReset Response SID	51_{16}	ERPR
#2	resetType = hardReset	01_{16}	RT_HR

10.6.8.2 Example #2 - Unidirectional Authentication with PKI Certificate Exchange without session key establishment (failure path)

10.6.8.2.1 Assumptions

For the below given message flow the following conditions apply:

— Step #1: No secure communication for further communication - communicationConfiguration: 00_{16}

— Step #1: Length of Certificate Client (2 bytes): $01F4_{16}$

— Step #1: Certificate Client (500 bytes)

— Step #1: NRC 50_{16} Certificate verification failed - Invalid Time Period (CVFITP)

— Step #1: Length of the Challenge Client (2 bytes): 0020_{16}

— Step #1: Challenge Client: 32 bytes random number

— Step #2: Diagnostic service ECUReset (11_{16}) is a secured service

10.6.8.2.2 Step #1: Send Certificate Client

Table 95 specifies the Authentication request message flow example #2 - step #1.

Table 95 — Unidirectional Authentication with PKI Certificate Exchange without session key establishment request message flow example #2 – step #1

Message direction		client -> server	
Message type		Request	
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	Authentication Request SID	29_{16}	ARS
#2	authenticationTask = verifyCertificateUnidirectional	01_{16}	LEV_AT_VCU
#3	communicationConfiguration = no secure communication	00_{16}	COCO
#4	lengthOfCertificateClient [byte#1]	01_{16}	LOCECL
#5	lengthOfCertificateClient [byte#2]	$F4_{16}$	
#6	certificateClient [byte#1]	30_{16}	CECL
:	:	:	
#505	certificateClient [byte#500]	AD_{16}	

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Message direction	client -> server		
Message type	Request		
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#506	lengthOfChallengeClient [byte#1]	00_{16}	LOCHCL
#507	lengthOfChallengeClient [byte#2]	20_{16}	
#508	challengeClient [byte#1]	AA_{16}	CHCL
:	:	:	
#539	challengeClient [byte#32]	44_{16}	

Table 96 specifies the Authentication positive response message flow example #2 - step #1.

Table 96 — Unidirectional Authentication with PKI Certificate Exchange without session key establishment negative response message flow example #2 - step #1

Message direction	server -> client		
Message type	Response		
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	Negative Response SID	$7F_{16}$	SIDRSIDNRQ
#2	Authentication Request SID	29_{16}	ARS
#3	responseCode = invalidTimePeriod	50_{16}	NRC_CVFITP

10.6.8.2.3 Step #2: Attempt to send a random secured service

Table 97 specifies the ECUReset request message after failed Authentication request message attempt - example #2 - step #2.

Table 97 — ECUReset request message after failed Authentication request message attempt – example #2 - step #2

Message direction	client -> server		
Message type	Request		
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	ECUReset Request SID	11_{16}	ER
#2	ResetType = hardReset	01_{16}	RT_HR

Table 98 specifies the ECUReset response message after failed Authentication request message attempt example #2 - step #2.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 98 — ECUReset response message after failed Authentication request message attempt – example #2 - step #2

Message direction		server -> client	
Message type		Response	
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	Negative Response SID	$7F_{16}$	SIDRSIDNRQ
#2	ECUReset Request SID	11_{16}	ER
#3	responseCode = authenticationRequired	34_{16}	NRC_AR

10.6.8.3 Example #3 – Transmit Certificate after Authentication with PKI Certificate Exchange (happy path)

10.6.8.3.1 Assumptions

For the below given message flow the following conditions apply:

— Server is already in an authenticated state,

— Length of Certificate Data (2 bytes): $01F4_{16}$,

— Certificate Data (500 bytes),

— Parameter returnValue in the positive response indicating Certificate verified: 13_{16},

— Public key in Certificate is used for signature verification, e.g. after signed data is transferred to the server.

The signature may be sent within the data transferred to the server or an additional individual vehicle manufacturer routine. A suitable signature for the data may be sent to be able to verify the transferred data with the public key from the Certificate.

10.6.8.3.2 Step #1: Send Certificate

Table 99 specifies the transmitCertificate request message flow example #3 - step #1.

Table 99 — Transmit Certificate request message flow example #3 - step #1

Message direction		client -> server	
Message type		Request	
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	Authentication Request SID	29_{16}	ARS
#2	authenticationTask = transmitCertificate	04_{16}	LEV_AT_TC
#3	lengthOfCertificateData [byte#1]	01_{16}	LOCEDA
#4	lengthOfCertificateData [byte#2]	$F4_{16}$	
#5	certificateData [byte#1]	31_{16}	CEDA
:	:	:	
#504	certificateData [byte#500]	AC_{16}	

Table 100 specifies the transmitCertificate positive response message flow example #3 - step #1.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 100 — Transmit Certificate positive response message flow example #3 - step #1

Message direction	**server -> client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	Authentication Response SID	69_{16}	ARS
#2	authenticationTask = transmitCertificate	04_{16}	LEV_AT_TC
#3	returnValue = Certificate verified	13_{16}	RV_CV

10.6.8.4 Example #4 - Unidirectional Authentication using Challenge-Response with asymmetric cryptography without session key establishment (happy path)

10.6.8.4.1 Assumptions

For the below given message flow the following conditions apply:

— Step #1: This step is optional: client requests the authentication configuration of the server

— Parameter communicationConfiguration (1 byte): 00_{16}

— Parameter algorithmIndicator (16 bytes): $06092A864886F70D01010A0000000000_{16}$

NOTE 1 In this example the value $06092A864886F70D01010A0000000000_{16}$ is the BER encoding (right padded with zeros up to 16 bytes) of the OID 1.2.840.113549.1.1.10 identifying the RSA digital signature scheme RSASSA-PSS according to PKCS #1 v2.2 RSA Cryptography Standard.

— Length of the Challenge Client (2 bytes): 0020_{16}

— Challenge Client: 32 bytes random number

— No additional parameter data needed

— Length of the Challenge Server (2 bytes): 0040_{16}

— Server challenge: 32 bytes ECU identification concatenated with 32 bytes random number

— Length of the client Proof of Ownership (2 bytes): 0150_{16}

— Proof of Ownership: 336 bytes software authentication token with structure based on the card-verifiable certificate definition as per ISO/IEC 7618-8:

  — The authentication token is encoded in the TLV template $7F21_{16}$; this template contains the token content and the token signature.

  — The content is encoded in the TLV object $7F4E_{16}$ within the TLV template $7F21_{16}$.

  — The signature is encoded in the TLV object $5F37_{16}$ within the TLV template $7F21_{16}$.

NOTE 2 The authentication token is valid, i.e. the format, content and the signature are correct.

10.6.8.4.2 Step #1: Request Authentication Configuration

Table 101 specifies the Authentication Configuration request message flow example #4 - step #1.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 101 — Unidirectional Authentication using Challenge-Response with asymmetric cryptography without session key establishment request message flow example #4 – step #1

Message direction	**client -> server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	Authentication Request SID	29_{16}	ARS
#2	authenticationTask = authenticationConfiguration	08_{16}	LEV_AT_AC

Table 102 specifies the Authentication Configuration positive response message flow example #4 – step #1.

Table 102 — Unidirectional Authentication using Challenge-Response with asymmetric cryptography without session key establishment positive response message flow example #4 – step #1

Message direction	**server -> client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	Authentication Response SID	69_{16}	ARS
#2	authenticationTask = authenticationConfiguration	08_{16}	LEV_AT_AC
#3	returnValue = AuthenticationConfiguration ACR with asymmetric cryptography	03_{16}	RV_ACACR

10.6.8.4.3 Step #2: Request the Challenge

Table 103 specifies the Authentication request message flow example #4 - step #2.

Table 103 — Unidirectional Authentication using Challenge-Response with asymmetric cryptography without session key establishment request message flow example #4 - step #2

Message direction	**client -> server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	Authentication Request SID	29_{16}	ARS
#2	authenticationTask = requestChallengeForAuthentication	05_{16}	LEV_AT_RCFA
#3	communicationConfiguration = no secure communication	00_{16}	COCO
#4	algorithmIndicator [byte#1]	06_{16}	AI
:	:	:	
#14	algorithmIndicator [byte#11]	$0A_{16}$	
:	:	:	
#19	algorithmIndicator [byte#16]	00_{16}	

Table 104 specifies the Authentication positive response message flow example #4 - step #2.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 104 — Unidirectional Authentication using Challenge-Response with asymmetric cryptography without session key establishment positive response message flow example #4 - step #2

Message direction	server -> client		
Message type	Response		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	Authentication Response SID	69_{16}	ARS
#2	authenticationTask = requestChallengeForAuthentication	05_{16}	LEV_AT_RCFA
#3	returnValue = Request accepted	00_{16}	RV_RA
#4	algorithmIndicator [byte#1]	06_{16}	AI
:	:	:	
#14	algorithmIndicator [byte#11]	$0A_{16}$	
:	:	:	
#19	algorithmIndicator [byte#16]	00_{16}	
#20	lengthOfChallengeServer [byte#1]	00_{16}	LOCHSE
#21	lengthOfChallengeServer [byte#2]	40_{16}	
#22	challengeServer [byte#1]	AA_{16}	CHSE
:	:	:	
#85	challengeServer [byte#64]	44_{16}	
#86	lengthOfNeededAdditionalParameter [byte#1]	00_{16}	LONAP
#87	lengthOfNeededAdditionalParameter [byte#2]	00_{16}	

10.6.8.4.4 Step #3: Validate the Proof of Ownership

Table 105 specifies the Authentication request message flow example #4 - step #3.

Table 105 — Unidirectional Authentication using Challenge-Response with asymmetric cryptography without session key establishment request message flow example #4 - step #3

Message direction	client -> server		
Message type	Request		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	Authentication Request SID	29_{16}	ARS
#2	authenticationTask = verifyProofOfOwnershipUnidirectional	06_{16}	LEV_AT_VPOWNU
#3	algorithmIndicator [byte#1]	06_{16}	AI
:	:	:	
#13	algorithmIndicator [byte#11]	$0A_{16}$	
:	:	:	
#18	algorithmIndicator [byte#16]	00_{16}	
#19	lengthOfProofOfOwnershipClient [byte#1]	01_{16}	LPOWNCL
#20	lengthOfProofOfOwnershipClient [byte#2]	50_{16}	

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Message direction	client -> server		
Message type	Request		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#21	proofOfOwnershipClient [byte#1]	$7F_{16}$	POWNCL
#22	proofOfOwnershipClient [byte#2]	21_{16}	
#23	proofOfOwnershipClient [byte#3]	82_{16}	
#24	proofOfOwnershipClient [byte#4]	01_{16}	
#25	proofOfOwnershipClient [byte#5]	$4B_{16}$	
#26	proofOfOwnershipClient [byte#6]	$7F_{16}$	
#27	proofOfOwnershipClient [byte#7]	$4E_{16}$	
#28	proofOfOwnershipClient [byte#8]	44_{16}	
#29	proofOfOwnershipClient [byte#9]	00_{16} - FF_{16}	
:	:	:	
#95	proofOfOwnershipClient [byte#75]	00_{16} - FF_{16}	
#96	proofOfOwnershipClient [byte#76]	$5F_{16}$	
#97	proofOfOwnershipClient [byte#77]	37_{16}	
#98	proofOfOwnershipClient [byte#78]	82_{16}	
#99	proofOfOwnershipClient [byte#79]	01_{16}	
#100	proofOfOwnershipClient [byte#80]	00_{16}	
#101	proofOfOwnershipClient [byte#81]	00_{16} - FF_{16}	
:	:	:	
#356	proofOfOwnershipClient [byte#336]	00_{16} - FF_{16}	
#357	lengthOfChallengeClient [byte#1]	00_{16}	LOCHCL
#358	lengthOfChallengeClient [byte#2]	20_{16}	
#359	challengeClient [byte#1]	AA_{16}	CHCL
:	:	:	
#390	challengeClient [byte#32]	44_{16}	
#391	lengthOfAdditionalParameter [byte#1]	00_{16}	LOAP
#392	lengthOfAdditionalParameter [byte#2]	00_{16}	

Table 106 specifies the Authentication positive response message flow example #4 - step #3.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 106 — Unidirectional Authentication using Challenge-Response with asymmetric cryptography without session key establishment positive response message flow example #4 - step #3

Message direction	**server -> client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	Authentication Response SID	69_{16}	ARS
#2	authenticationTask = verifyProofOfOwnershipUnidirectional	06_{16}	LEV_AT_VPOWNU
#3	returnValue = Ownership verified, Authentication complete	12_{16}	RV_OVAC
#4 : #14 : #19	algorithmIndicator [byte#1] : algorithmIndicator [byte#11] : algorithmIndicator [byte#16]	06_{16} : $0A_{16}$: 00_{16}	AI
#20	lengthOfSessionKeyInfo [byte#1]	00_{16}	LOSKI
#21	lengthOfSessionKeyInfo [byte#2]	00_{16}	

10.6.8.5 Example #5 - Unidirectional Authentication using Challenge-Response (ACR) with asymmetric cryptography without session key establishment (failure path)

10.6.8.5.1 Assumptions

For the below given message flow the following conditions apply:

— Parameter communicationConfiguration (1 byte): 00_{16}

— Parameter algorithmIndicator (16 bytes): $06092A864886F70D01010A0000000000_{16}$

NOTE 1 In this example the value $06092A864886F70D01010A0000000000_{16}$ is the BER encoding (right padded with zeros up to 16 bytes) of the OID 1.2.840.113549.1.1.10 identifying the RSA digital signature scheme RSASSA-PSS according to PKCS #1 v2.2 RSA Cryptography Standard.

— Length of the Challenge Client (2 bytes): 0020_{16}

— Challenge Client: 32 bytes random number

— No additional parameter data needed

— Length of the Challenge Server (2 bytes): 0040_{16}

— Server challenge: 32 bytes ECU identification concatenated with 32 bytes random number

— Length of the client Proof of Ownership (2 bytes): 0150_{16}

— Proof of Ownership: 336 bytes software authentication token with structure based on the card-verifiable certificate definition as per ISO/IEC 7618-8:

  — The authentication token is encoded in the TLV template $7F21_{16}$; this template contains the token content and the token signature.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

— The content is encoded in the TLV object $7F4E_{16}$ within the TLV template $7F21_{16}$.

— The signature is encoded in the TLV object $5F37_{16}$ within the TLV template $7F21_{16}$.

NOTE 1 The authentication token is not valid due to an incorrect signature.

— Parameter returnValue in the positive response indicating invalid signature: 21_{16}

NOTE 2 The value of this parameter is in the vehicle manufacturer specific range. The Mnemonic is also manufacturer specific.

10.6.8.5.2 Step #1: Request the Challenge

Table 107 specifies the Authentication request message flow example #5 - step #1.

Table 107 — Unidirectional Authentication using Challenge-Response with asymmetric cryptography without session key establishment request message flow example #5 - step #1

Message direction	**client -> server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	Authentication Request SID	29_{16}	ARS
#2	authenticationTask = requestChallengeForAuthentication	05_{16}	LEV_AT_RCFA
#3	communicationConfiguration = no secure communication	00_{16}	COCO
#4	algorithmIndicator [byte#1]	06_{16}	AI
:	:	:	
#14	algorithmIndicator [byte#11]	$0A_{16}$	
:	:	:	
#19	algorithmIndicator [byte#16]	00_{16}	

Table 108 specifies the Authentication positive response message flow example #5 - step #1.

Table 108 — Unidirectional Authentication using Challenge-Response with asymmetric cryptography without session key establishment positive response message flow example #5 - step #1

Message direction	**server -> client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	Authentication Response SID	69_{16}	ARS
#2	authenticationTask = requestChallengeForAuthentication	05_{16}	LEV_AT_RCFA
#3	returnValue = Request accepted	00_{16}	RV_RA
#4	algorithmIndicator [byte#1]	06_{16}	AI
:	:	:	
#14	algorithmIndicator [byte#11]	$0A_{16}$	
:	:	:	
#19	algorithmIndicator [byte#16]	00_{16}	

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Message direction	server -> client		
Message type	Response		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#20	lengthOfChallengeServer [byte#1]	00_{16}	LOCHSE
#21	lengthOfChallengeServer [byte#2]	40_{16}	
#22	challengeServer [byte#1]	AA_{16}	CHSE
:	:	:	
#85	challengeServer [byte#64]	44_{16}	
#86	lengthOfNeededAdditionalParameter [byte#1]	00_{16}	LONAP
#87	lengthOfNeededAdditionalParameter [byte#2]	00_{16}	

10.6.8.5.3 Step #2: Validate the Proof of Ownership

Table 109 specifies the Authentication request message flow example #5 - step #2.

Table 109 — Unidirectional Authentication using Challenge-Response with asymmetric cryptography without session key establishment request message flow example #5 - step #2

Message direction	client -> server		
Message type	Request		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	Authentication Request SID	29_{16}	ARS
#2	authenticationTask = verifyProofOfOwnershipUnidirectional	06_{16}	LEV_AT_VPOWNU
#3	algorithmIndicator [byte#1]	06_{16}	AI
:	:	:	
#13	algorithmIndicator [byte#11]	$0A_{16}$	
:	:	:	
#18	algorithmIndicator [byte#16]	00_{16}	
#19	lengthOfProofOfOwnershipClient [byte#1]	01_{16}	LPOWNCL
#20	lengthOfProofOfOwnershipClient [byte#2]	50_{16}	
#21	proofOfOwnershipClient [byte#1]	$7F_{16}$	POWNCL
#22	proofOfOwnershipClient [byte#2]	21_{16}	
#23	proofOfOwnershipClient [byte#3]	82_{16}	
#24	proofOfOwnershipClient [byte#4]	01_{16}	
#25	proofOfOwnershipClient [byte#5]	$4B_{16}$	
#26	proofOfOwnershipClient [byte#6]	$7F_{16}$	
#27	proofOfOwnershipClient [byte#7]	$4E_{16}$	
#28	proofOfOwnershipClient [byte#8]	44_{16}	
#29	proofOfOwnershipClient [byte#9]	00_{16} - FF_{16}	
:	:	:	

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Message direction	client -> server		
Message type	Request		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#95	proofOfOwnershipClient [byte#75]	00_{16} - FF_{16}	POWNCL
#96	proofOfOwnershipClient [byte#76]	$5F_{16}$	
#97	proofOfOwnershipClient [byte#77]	37_{16}	
#98	proofOfOwnershipClient [byte#78]	82_{16}	
#99	proofOfOwnershipClient [byte#79]	01_{16}	
#100	proofOfOwnershipClient [byte#80]	00_{16}	
#101	proofOfOwnershipClient [byte#81]	00_{16} - FF_{16}	
:	:	:	
#356	proofOfOwnershipClient [byte#336]	00_{16} - FF_{16}	
#357	lengthOfChallengeClient [byte#1]	00_{16}	LOCHCL
#358	lengthOfChallengeClient [byte#2]	20_{16}	
#359	challengeClient [byte#1]	AA_{16}	CHCL
:	:	:	
#390	challengeClient [byte#32]	44_{16}	
#390	lengthOfAdditionalParameter [byte#1]	00_{16}	LOAP
#391	lengthOfAdditionalParameter [byte#2]	00_{16}	

Table 110 specifies the Authentication negative response message flow example #5 - step #2.

Table 110 — Unidirectional Authentication using Challenge-Response with asymmetric cryptography without session key establishment negative response message flow example #5 - step #2

Message direction	server -> client		
Message type	Response		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	Negative Response SID	$7F_{16}$	SIDRSIDNRQ
#2	Authentication Request SID	29_{16}	ARS
#3	responseCode = invalidSignature	51_{16}	NRC_CVFIS

10.6.8.6 Example #6 Unidirectional Authentication using Challenge-Response (ACR) with symmetric cryptography without session key establishment (happy path)

10.6.8.6.1 Assumptions

For the below given message flow the following conditions apply:

— Parameter communicationConfiguration (1 byte): 00_{16}

— Parameter algorithmIndicator (16 bytes): $06096086480165030401020000000000_{16}$

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

NOTE In this example the value $0609608648016503040102000000000000_{16}$ is the BER encoding (right padded with zeros up to 16 bytes) of the OID 2.16.840.1.101.3.4.1.2 identifying the AES-128 CBC mode cryptographic algorithm according to FIPS PUB 197. In this example the AES keys is $2B7E151628AED2A6ABF7158809CF4F3C_{16}$

— No Challenge Client needed

— No additional parameter data needed

— Length of the Challenge Server (2 bytes): 0010_{16}

— Server challenge: 16 bytes random number

— Length of the client Proof of Ownership (2 bytes): 0010_{16}

— Client Proof of Ownership: 16 bytes value calculated from the Challenge Server and the key. In this example, the calculation is AES encryption of the 16 bytes of Challenge Server using the key $2B7E151628AED2A6ABF7158809CF4F3C_{16}$

10.6.8.6.2 Step #1: Request challenge for Authentication

Table 111 specifies the Authentication request message flow example #6 – step #1.

Table 111 — Unidirectional Authentication using Challenge-Response (ACR) with symmetric cryptography without session key establishment request message flow example #6 - step #1

Message direction	**client -> server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	Authentication Request SID	29_{16}	ARS
#2	authenticationTask = requestChallengeForAuthentication	05_{16}	LEV_AT_RCFA
#3	communicationConfiguration = no secure communication	00_{16}	COCO
#4	algorithmIndicator [byte#1]	06_{16}	AI
:	:	:	
#14	algorithmIndicator [byte#11]	02_{16}	
:	:	:	
#19	algorithmIndicator [byte#16]	00_{16}	

Table 112 specifies the Authentication positive response message flow example #6 - step #1. The server returns 16 bytes of the challengeServer value.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 112 — Unidirectional Authentication using Challenge-Response (ACR) with symmetric cryptography without session key establishment positive response message flow example #6 - step #1

Message direction		server -> client	
Message type		Response	
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	Authentication Response SID	69_{16}	ARS
#2	authenticationTask = requestChallengeForAuthentication	05_{16}	LEV_AT_RCFA
#3	returnValue = Request accepted	00_{16}	RV_RA
#4	algorithmIndicator [byte#1]	06_{16}	AI
:	:	:	
#14	algorithmIndicator [byte#11]	02_{16}	
:	:	:	
#19	algorithmIndicator [byte#16]	00_{16}	
#20	lengthOfChallengeServer [byte#1] (high byte)	00_{16}	LOCHSE_HB
#21	lengthOfChallengeServer [byte#2] (low byte)	10_{16}	LOCHSE_LB
#22	challengeServer [byte#1]	32_{16}	CHSE
#23	challengeServer [byte#2]	43_{16}	
#24	challengeServer [byte#3]	$F6_{16}$	
#25	challengeServer [byte#4]	$A8_{16}$	
#26	challengeServer [byte#5]	88_{16}	
#27	challengeServer [byte#6]	$5A_{16}$	
#28	challengeServer [byte#7]	30_{16}	
#29	challengeServer [byte#8]	$8D_{16}$	
#30	challengeServer [byte#9]	31_{16}	
#31	challengeServer [byte#10]	31_{16}	
#32	challengeServer [byte#11]	98_{16}	
#33	challengeServer [byte#12]	$A2_{16}$	
#34	challengeServer [byte#13]	$E0_{16}$	
#35	challengeServer [byte#14]	37_{16}	
#36	challengeServer [byte#15]	07_{16}	
#37	challengeServer [byte#16]	34_{16}	
#38	lengthOfNeededAdditionalParameter [byte#1] (high byte)	00_{16}	LONAP_HB
#39	lengthOfNeededAdditionalParameter [byte#2] (low byte)	00_{16}	LONAP_LB

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

10.6.8.6.3 Step #2: Verify Proof of Ownership Unidirectional

Table 113 defines Authentication request message flow example #6 - step #2. The client provided the correct ProofOfOwnership value.

© ISO 2020 – All rights reserved

Table 113 — Unidirectional Authentication using Challenge-Response (ACR) with symmetric cryptography without session key establishment request message flow example #6 - step #2

Message direction	**client -> server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	Authentication Request SID	29_{16}	ARS
#2	authenticationTask = verifyProofOfOwnershipUnidirectional	06_{16}	LEV_AT_VPOWNU
#3	algorithmIndicator [byte#1]	06_{16}	AI
:	:	:	
#13	algorithmIndicator [byte#11]	02_{16}	
:	:	:	
#18	algorithmIndicator [byte#16]	00_{16}	
#19	lengthOfProofOfOwnershipClient [byte#1] (high byte)	00_{16}	LPOWNCL_HB
#20	lengthOfProofOfOwnershipClient [byte#2] (low byte)	10_{16}	LPOWNCL_LB
#21	proofOfOwnershipClient [byte#1]	39_{16}	POWNCL
#22	proofOfOwnershipClient [byte#2]	25_{16}	
#23	proofOfOwnershipClient [byte#3]	84_{16}	
#24	proofOfOwnershipClient [byte#4]	$1D_{16}$	
#25	proofOfOwnershipClient [byte#5]	02_{16}	
#26	proofOfOwnershipClient [byte#6]	DC_{16}	
#27	proofOfOwnershipClient [byte#7]	09_{16}	
#28	proofOfOwnershipClient [byte#8]	FB_{16}	
#29	proofOfOwnershipClient [byte#9]	DC_{16}	
#30	proofOfOwnershipClient [byte#10]	11_{16}	
#31	proofOfOwnershipClient [byte#11]	85_{16}	
#32	proofOfOwnershipClient [byte#12]	97_{16}	
#33	proofOfOwnershipClient [byte#13]	19_{16}	
#34	proofOfOwnershipClient [byte#14]	$6A_{16}$	
#35	proofOfOwnershipClient [byte#15]	$0B_{16}$	
#36	proofOfOwnershipClient [byte#16]	32_{16}	
#37	lengthOfChallengeClient [byte#1]	00_{16}	LOCHCL
#38	lengthOfChallengeClient [byte#2]	00_{16}	
#39	lengthOfAdditionalParameter [byte#1] (high byte)	00_{16}	LOAP_HB
#40	lengthOfAdditionalParameter [byte#2] (low byte)	00_{16}	LOAP_LB

Table 114 defines Authentication positive response message flow example #6 - step #2.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 114 — Unidirectional Authentication using Challenge-Response (ACR) with symmetric cryptography without session key establishment positive response message flow example #6 - step #2

Message direction	**server -> client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	Authentication Response SID	69_{16}	ARS
#2	authenticationTask = verifyProofOfOwnershipUnidirectional	06_{16}	LEV_AT_VPOWNU
#3	returnValue = Ownership verified, Authentication complete	12_{16}	RV_OVAC
#4 : #14 : #19	algorithmIndicator [byte#1] : algorithmIndicator [byte#11] : algorithmIndicator [byte#16]	06_{16} : 02_{16} : 00_{16}	AI
#20	lengthOfSessionKeyInfo [byte#1]	00_{16}	LOSKI
#21	lengthOfSessionKeyInfo [byte#2]	00_{16}	

10.6.8.7 Example #7 Unidirectional Authentication using Challenge-Response (ACR) with symmetric cryptography without session key establishment (failure path)

10.6.8.7.1 Assumptions

For the below given message flow the following conditions apply:

— Parameter communicationConfiguration (1 byte): 00_{16}

— Parameter algorithmIndicator (16 bytes): $0609608648016503040102000000000_{16}$

NOTE In this example the value $0609608648016503040102000000000_{16}$ is the BER encoding (right padded with zeros up to 16 bytes) of the OID 2.16.840.1.101.3.4.1.2 identifying the AES-128 CBC mode cryptographic algorithm according to FIPS PUB 197.

— No Challenge Client needed

— No additional parameter data needed

— Length of the Challenge Server (2 bytes): 0010_{16}

— Server challenge: 16 bytes random number

— Length of the client Proof of Ownership (2 bytes): 0010_{16}

— Client Proof of Ownership: 16 bytes value calculated from the Challenge Server and the key. In this example, the AES keys of the client do not match with the keys of the server.

10.6.8.7.2 Step #1: Request challenge for Authentication

Table 115 specifies the Authentication request message flow example #7 – step #1.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

Table 115 — Unidirectional Authentication using Challenge-Response (ACR) with symmetric cryptography without session key establishment request message flow example #7 - step #1

Message direction	**client -> server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	Authentication Request SID	29_{16}	ARS
#2	authenticationTask = requestChallengeForAuthentication	05_{16}	LEV_AT_RCFA
#3	communicationConfiguration = no secure communication	00_{16}	COCO
#4	algorithmIndicator [byte#1]	06_{16}	AI
:	:	:	
#14	algorithmIndicator [byte#11]	02_{16}	
:	:	:	
#19	algorithmIndicator [byte#16]	00_{16}	

Table 116 specifies the Authentication positive response message flow example #7 - step #1. The server returns 16 bytes of the challengeServer value.

Table 116 — Unidirectional Authentication using Challenge-Response (ACR) with symmetric cryptography without session key establishment positive response message flow example #7 - step #1

Message direction	**server -> client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	Authentication Response SID	69_{16}	ARS
#2	authenticationTask = requestChallengeForAuthentication	05_{16}	LEV_AT_RCFA
#3	returnValue = Request accepted	00_{16}	RV_RA
#4	algorithmIndicator [byte#1]	06_{16}	AI
:	:	:	
#14	algorithmIndicator [byte#11]	02_{16}	
:	:	:	
#19	algorithmIndicator [byte#16]	00_{16}	
#20	lengthOfChallengeServer [byte#1] (high byte)	00_{16}	LOCHSE_HB
#21	lengthOfChallengeServer [byte#2] (low byte)	10_{16}	LOCHSE_LB

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Message direction	server -> client		
Message type	Response		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#22	challengeServer [byte#1]	32_{16}	CHSE
#23	challengeServer [byte#2]	43_{16}	
#24	challengeServer [byte#3]	$F6_{16}$	
#25	challengeServer [byte#4]	$A8_{16}$	
#26	challengeServer [byte#5]	88_{16}	
#27	challengeServer [byte#6]	$5A_{16}$	
#28	challengeServer [byte#7]	30_{16}	
#29	challengeServer [byte#8]	$8D_{16}$	
#30	challengeServer [byte#9]	31_{16}	
#31	challengeServer [byte#10]	31_{16}	
#32	challengeServer [byte#11]	98_{16}	
#33	challengeServer [byte#12]	$A2_{16}$	
#34	challengeServer [byte#13]	$E0_{16}$	
#35	challengeServer [byte#14]	37_{16}	
#36	challengeServer [byte#15]	07_{16}	
#37	challengeServer [byte#16]	34_{16}	
#38	lengthOfNeededAdditionalParameter [byte#1] (high byte)	00_{16}	LONAP_HB
#39	lengthOfNeededAdditionalParameter [byte#2] (low byte)	00_{16}	LONAP_LB

10.6.8.7.3 Step #2: Verify Proof of Ownership Unidirectional

Table 117 defines Authentication request message flow example #7 - step #2. The client provides an invalid Proof of Ownership.

Table 117 — Unidirectional Authentication using Challenge-Response (ACR) with symmetric cryptography without session key establishment request message flow example #7 - step #2

Message direction	client -> server		
Message type	Request		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	Authentication Request SID	29_{16}	ARS
#2	authenticationTask = verifyProofOfOwnershipUnidirectional	06_{16}	LEV_AT_VPOWNU
#3	algorithmIndicator [byte#1]	06_{16}	AI
:	:	:	
#13	algorithmIndicator [byte#11]	02_{16}	
:	:	:	
#18	algorithmIndicator [byte#16]	00_{16}	

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Message direction	client -> server		
Message type	Request		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#19	lengthOfProofOfOwnershipClient [byte#1] (high byte)	00_{16}	LPOWNCL_HB
#20	lengthOfProofOfOwnershipClient [byte#2] (low byte)	10_{16}	LPOWNCL_LB
#21	proofOfOwnershipClient [byte#1]	01_{16}	POWNCL
#22	proofOfOwnershipClient [byte#2]	02_{16}	
#23	proofOfOwnershipClient [byte#3]	03_{16}	
#24	proofOfOwnershipClient [byte#4]	04_{16}	
#25	proofOfOwnershipClient [byte#5]	05_{16}	
#26	proofOfOwnershipClient [byte#6]	06_{16}	
#27	proofOfOwnershipClient [byte#7]	07_{16}	
#28	proofOfOwnershipClient [byte#8]	08_{16}	
#29	proofOfOwnershipClient [byte#9]	09_{16}	
#30	proofOfOwnershipClient [byte#10]	$0A_{16}$	
#31	proofOfOwnershipClient [byte#11]	$0B_{16}$	
#32	proofOfOwnershipClient [byte#12]	$0C_{16}$	
#33	proofOfOwnershipClient [byte#13]	$0D_{16}$	
#34	proofOfOwnershipClient [byte#14]	$0E_{16}$	
#35	proofOfOwnershipClient [byte#15]	$0F_{16}$	
#36	proofOfOwnershipClient [byte#16]	11_{16}	
#37	lengthOfChallengeClient [byte#1]	00_{16}	LOCHCL
#38	lengthOfChallengeClient [byte#2]	00_{16}	
#39	lengthOfAdditionalParameter [byte#1] (high byte)	00_{16}	LOAP_HB
#40	lengthOfAdditionalParameter [byte#2] (low byte)	00_{16}	LOAP_LB

Table 118 defines Authentication negative response message flow example #7 - step #2.

Table 118 — Unidirectional Authentication using Challenge-Response (ACR) with symmetric cryptography without session key establishment negative response message flow example #7 - step #2

Message direction	server -> client		
Message type	Response		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	Negative Response SID	$7F_{16}$	SIDRSIDNRQ
#2	Authentication Request SID	29_{16}	ARS
#3	responseCode = Ownership verification failed	58_{16}	NRC_OVF

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

10.7 TesterPresent ($3E_{16}$) service

10.7.1 Service description

This service is used to indicate to a server (or servers) that a client is still connected to the vehicle and that certain diagnostic services and/or communication that have been previously activated are to remain active.

This service is used to keep one or multiple servers in a diagnostic session other than the defaultSession. This can either be done by transmitting the TesterPresent request message periodically or in case of the absence of other diagnostic services to prevent the server(s) from automatically returning to the defaultSession. The detailed session requirements that apply to the use of this service when keeping a single server or multiple servers in a diagnostic session other than the defaultSession can be found in the implementation specifications of ISO 14229.

IMPORTANT — The server and the client shall meet the request and response message behaviour as specified in 8.7.

10.7.2 Request message

10.7.2.1 Request message definition

Table 119 specifies the request message.

Table 119 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	TesterPresent Request SID	M	$3E_{16}$	TP
#2	SubFunction = [zeroSubFunction]	M	$00_{16}/80_{16}$	LEV_ZSUBF

10.7.2.2 Request message SubFunction parameter $Level (LEV_) definition

Table 120 specifies the SubFunction parameter values defined for this service (suppressPosRspMsgIndicationBit (bit 7) not shown).

Table 120 — Request message SubFunction parameter definition

Bits 6 to 0	Description	Cvt	Mnemonic
00_{16}	**zeroSubFunction** This parameter value is used to indicate that no SubFunction value beside the suppressPosRspMsgIndicationBit is supported by this service.	M	ZSUBF
01_{16} to $7F_{16}$	**ISOSAEReserved** This range of values is reserved by this document.	M	ISOSAERESRVD

10.7.2.3 Request message data-parameter definition

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

10.7.3 Positive response message

10.7.3.1 Positive response message definition

Table 121 specifies the positive response message.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

Table 121 — Positive response message definition

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	TesterPresent Response SID	M	$7E_{16}$	TPPR
#2	SubFunction = [zeroSubFunction]	M	00_{16}	LEV_ZSUBF

10.7.3.2 Positive response message data-parameter definition

Table 122 specifies the data-parameter of the positive response message.

Table 122 — Response message data-parameter definition

Definition
zeroSubFunction This parameter is an echo of bits 6 - 0 of the SubFunction parameter from the request message.

10.7.4 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 123. The listed negative responses shall be used if the error scenario applies to the server.

Table 123 — 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

10.7.5 Message flow example(s) TesterPresent

10.7.5.1 Example #1 - TesterPresent (suppressPosRspMsgIndicationBit = FALSE)

Table 124 specifies the TesterPresent request message flow example #1.

Table 124 — TesterPresent request message flow example #1

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	TesterPresent Request SID	$3E_{16}$	TP
#2	zeroSubFunction, suppressPosRspMsgIndicationBit = FALSE	00_{16}	ZSUBF

Table 125 specifies the TesterPresent positive response message flow example #1.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 125 — TesterPresent positive response message flow example #1

Message direction		server → client	
Message type		Response	
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	TesterPresent Response SID	$7E_{16}$	TPPR
#2	zeroSubFunction, suppressPosRspMsgIndicationBit = FALSE	00_{16}	ZSUBF

10.7.5.2 Example #2 - TesterPresent (suppressPosRspMsgIndicationBit = TRUE)

Table 126 specifies the TesterPresent request message flow example #2.

Table 126 — TesterPresent request message flow example #2

Message direction		client → server	
Message type		Request	
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	TesterPresent Request SID	$3E_{16}$	TP
#2	zeroSubFunction, suppressPosRspMsgIndicationBit = TRUE	80_{16}	ZSUBF

There is no response sent by the server(s).

10.8 ControlDTCSetting (85_{16}) service

10.8.1 Service description

The ControlDTCSetting service shall be used by a client to stop or resume the updating of DTC status bits in the server(s). DTC status bits are reported in the statusOfDTC parameter of the positive response to certain SubFunctions of ReadDTCInformation (see D.2 for definition of the bits).

The ControlDTCSetting request message can be used to stop the updating of DTC status bits in an individual server or a group of servers. If the server being addressed is not able to stop the updating of DTC status bits, it shall respond with a ControlDTCSetting negative response message indicating the reason for the reject.

When the server accepts a ControlDTCSetting request with a SubFunction value of DTCSettingType = off, the server shall suspend any updates to the DTC status bits (i.e. freeze current values) until the functionality is enabled again. The update of the DTC status bit information shall continue once a ControlDTCSetting request is performed with SubFunction set to "on" or a transition to a session where ControlDTCSetting is not supported occurs (e.g. session layer timeout to defaultSession, ECU reset, etc.). The server shall still send a positive response if the service is supported in the active session with a requested SubFunction set to either "on" or "off" even if the requested DTC setting state is already active.

If a ClearDiagnosticInformation (14_{16}) service is sent by the client, the ControlDTCSetting shall not prohibit resetting the server's DTC status bits. The behaviour of the individual DTC status bits shall be implemented according to the definitions in D.2, Figure D.1 to Figure D.8.

DTC status bits document certain information relative to a numerical identifier (DTC) that represents a specific fault condition(s). ControlDTCSetting only switches on/off the DTC status bit updating. ControlDTCSetting service is not intended to cause fault monitoring to be switched off nor is it intended to cause failsoft strategies to be disabled. It is not recommended that failsoft or failsafe strategies be directly linked or coupled with DTC status bits (e.g. an accepted ClearDiagnosticInformation request does not directly remove any active failsoft).

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

IMPORTANT — The server and the client shall meet the request and response message behaviour as specified in 8.7.

10.8.2 Request message

10.8.2.1 Request message definition

Table 127 specifies the request message.

Table 127 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	ControlDTCSetting Request SID	M	85_{16}	CDTCS
#2	SubFunction = [DTCSettingType]	M	00_{16} to FF_{16}	LEV_DTCSTP_
#3 : #n	DTCSettingControlOptionRecord [] = [parameter#1 : parameter#m	U : U	00_{16} to FF_{16} : 00_{16} to FF_{16}	DTCSCOR_ PARA1 : PARAm

10.8.2.2 Request message SubFunction parameter $Level (LEV_) definition

The SubFunction parameter DTCSettingType is used by the ControlDTCSetting request message to indicate to the server(s) whether diagnostic trouble code status bit updating shall stop or start again (suppressPosRspMsgIndicationBit (bit 7) not shown in Table 128).

Table 128 — Request message SubFunction parameter definition

Bits 6 to 0	Description	Cvt	Mnemonic
00_{16}	**ISOSAEReserved** This value is reserved by this document.	M	ISOSAERESRVD
01_{16}	**on** The server(s) shall resume the updating of diagnostic trouble code status bits according to normal operating conditions.	M	ON
02_{16}	**off** The server(s) shall stop the updating of diagnostic trouble code status bits.	M	OFF
03_{16} to $3F_{16}$	**ISOSAEReserved** This range of values is reserved by this document for future definition.	M	ISOSAERESRVD
40_{16} to $5F_{16}$	**vehicleManufacturerSpecific** This range of values is reserved for vehicle manufacturer specific use.	U	VMS
60_{16} to $7E_{16}$	**systemSupplierSpecific** This range of values is reserved for system supplier specific use.	U	SSS
$7F_{16}$	**ISOSAEReserved** This value is reserved by this document for future definition.	M	ISOSAERESRVD

10.8.2.3 Request message data-parameter definition

Table 129 specifies the data-parameter of the request message.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 129 — Request message data-parameter definition

Definition
DTCSettingControlOptionRecord This parameter record is user optional to transmit data to a server when controlling the updating of DTC status bits (e.g. it can contain a list of DTCs to be turned on or off).

10.8.3 Positive response message

10.8.3.1 Positive response message definition

Table 130 specifies the positive response message.

Table 130 — Positive response message definition

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	ControlDTCSetting Response SID	M	$C5_{16}$	CDTCSPR
#2	DTCSettingType	M	00-7F	DTCSTP

10.8.3.2 Positive response message data-parameter definition

Table 131 specifies the data-parameter of the positive response message.

Table 131 — Response message data-parameter definition

Definition
DTCSettingType This parameter is an echo of bits 6 - 0 of the SubFunction parameter from the request message.

10.8.4 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 132. The listed negative responses shall be used if the error scenario applies to the server.

Table 132 — 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** Used when the server is in a critical normal mode activity and therefore cannot perform the requested DTC control functionality.	CNC
31_{16}	**requestOutOfRange** The server shall use this response code, if it detects an error in the DTCSettingControlOptionRecord.	ROOR

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

10.8.5 Message flow example(s) ControlDTCSetting

10.8.5.1 Example #1 - ControlDTCSetting (DTCSettingType = off)

Note that this example does not use the capability of the service to transfer additional data to the server. The client requests to have a response message by setting the suppressPosRspMsgIndicationBit (bit 7 of the SubFunction parameter) to "FALSE" ('0').

Table 133 specifies the ControlDTCSetting request message flow example #1.

Table 133 — ControlDTCSetting request message flow example #1

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ControlDTCSetting Request SID	85_{16}	RDTCS
#2	DTCSettingType = off, suppressPosRspMsgIndicationBit = FALSE	02_{16}	DTCSTP_OFF

Table 134 specifies the ControlDTCSetting positive response message flow example #1.

Table 134 — ControlDTCSetting positive response message flow example #1

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ControlDTCSetting Response SID	$C5_{16}$	RDTCSPR
#2	DTCSettingType = off	02_{16}	DTCSTP_OFF

10.8.5.2 Example #2 - ControlDTCSetting (DTCSettingType = on)

This example does not use the capability of the service to transfer additional data to the server. The client requests to have a response message by setting the suppressPosRspMsgIndicationBit (bit 7 of the SubFunction parameter) to "FALSE" ('0').

Table 135 specifies the ControlDTCSetting request message flow example #2.

Table 135 — ControlDTCSetting request message flow example #2

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ControlDTCSetting Request SID	85_{16}	ENC
#2	DTCSettingType = on, suppressPosRspMsgIndicationBit = FALSE	01_{16}	DTCSTP_ON

Table 136 specifies the ControlDTCSetting positive response message flow example #2.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 136 — ControlDTCSetting positive response message flow example #2

Message direction		server → client	
Message type		Response	
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	ControlDTCSetting Response SID	$C5_{16}$	RDTCSPR
#2	DTCSettingType = on	01_{16}	DTCSTP_ON

10.9 ResponseOnEvent (86_{16}) service

10.9.1 Service description

The ResponseOnEvent service requests a server to start or stop transmission of responses on a specified event.

This service provides the possibility to automatically execute a diagnostic service in case a specified event occurs in the server. The client specifies the event (including optional event parameters) and the service (including service parameters) to be executed in case the event occurs. See Figure 12 for a brief overview about the client and server behaviour.

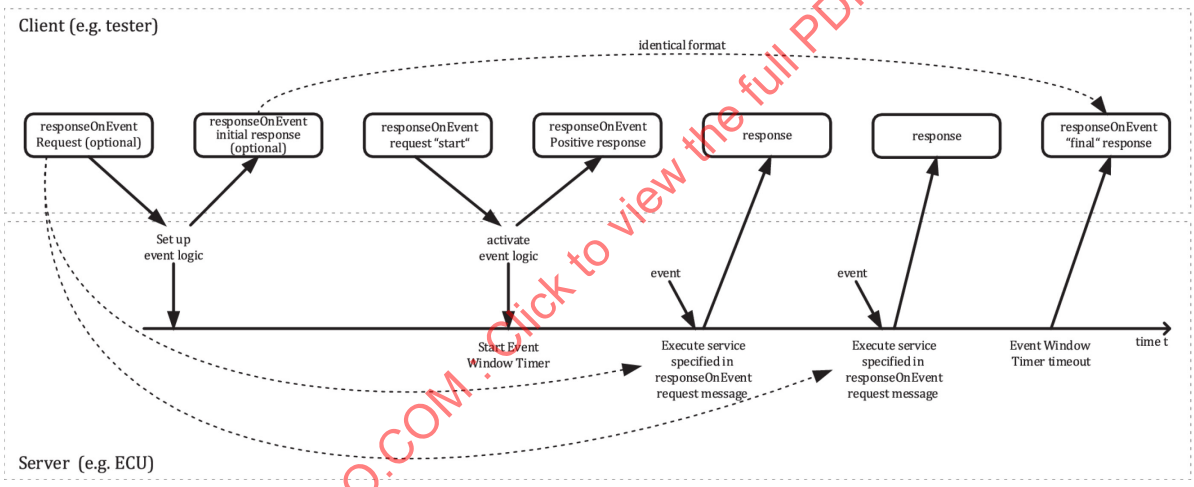


Figure 12 — ResponseOnEvent service – Client and server behaviour

NOTE 1 Figure 12 assumes, that the event window timer is configured to timeout prior to the power down of the server, therefore the final ResponseOnEvent positive response message is shown at the end of the event timing window.

The server shall evaluate the SubFunction and data content of the ResponseOnEvent request message at the time of the reception. This includes the following SubFunction and parameters:

— eventType,

— eventWindowTime,

— eventTypeRecord (eventTypeParameter #1-#m).

In case of invalid data in the ResponseOnEvent request message a negative response with the negative response code 31_{16} shall be sent. The serviceToRespondToRecord is not part of this evaluation. The serviceToRespondToRecord parameter will be evaluated when the specified event occurs, which

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

triggers the execution of the service contained in the serviceToRespondToRecord. At the time the event occurs the serviceToRespondToRecord (diagnostic service request message) shall be executed. The serviceToRespondToRecord execution follows the same rules as any other service. In particular all the general server response behaviour according to Figures 5 and 6 is followed. In case conditions are not correct a negative response message with the appropriate negative response code shall be sent. Multiple DTC events shall be signalled in the order of their occurrence. Multiple DID events shall be signalled in the order of their setup sequence.

Configured events with SubFunction equal to onChangeOfDataIdentifier or onComparisonOfValues require the server to sample and compare the value of the configured DataIdentifiers (DID). Therefore the server shall define the following parameters:

— ResponseOnEventSchedulerRate: the call rate of the periodic scheduler to compare the values of the DataIdentifier (DID) or to detect DTC status changes.

— MaxNumChangeOfDataIdentfierEvents: the maximum number of events that can be simultaneously configured with SubFunction onChangeOfDataIdentifier.

— MaxNumComparisionOfValueEvents: the maximum number of events that can be simultaneously configured with SubFunction onComparisonOfValues.

— MaxSupportedDIDLength: the maximum number of measureable data bytes allowed for each DID that is used for comparision or data change.

It is vehicle manufacturer-specific if multiple DataIdentifier change events are handled via First-In First-Out (FIFO) or if the event logic is temporarily stopped, while the event reporting is handled.

It is strongly recommended to use a value of MaxSupportedDIDLength with a size of of maximum 4 bytes (e.g. avoid definition of event logic reading constant “calibration” labels).

It is recommended to set ResponseOnEventSchedulerRate to an integer multiple of the server’s internal scheduler rate. The value of ResponseOnEventSchedulerRate has direct impact on the server performance. Every time the ResponseOnEventSchedulerRate ellapses, the server samples (reads) all configured DataIdentifiers into a local buffer and compares the values and triggers the event if the conditions are met. A very small value increases the server CPU load (oversampling) and could alter the overall server behaviour, which especially for debugging purposes is not desired.

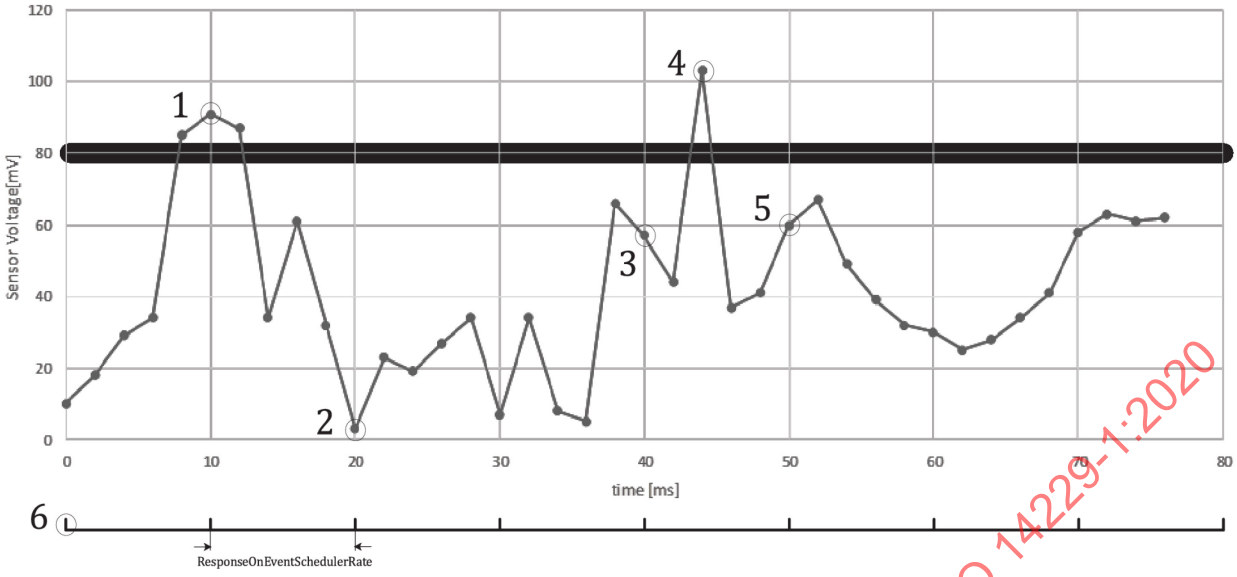
The server detects changes of DataIdentifier values only during the sampling phase. Changes of measurement values between two sampling operations will not trigger an event (undersampling).

It is a system design decision to choose a suiteable value for ResponseOnEventSchedulerRate to avoid the effects of over- and undersampling of DataIdentifiers.

In Figure 13 an example of the over- and undersampling is given. The server is configured with ResponseOnEventSchedulerRate = 10 ms, a configured event onComparisionOfValues with compareLogic ‘greater than 80 mV’ on the DID value. In the measured time, the value of the DIDs exceeds the configured threshold of ‘>80 mV’ twice, once after 10ms, another time after 44 ms. The server samples the DID every 10 ms and only the did value on the sampler after 10 ms will trigger a response on event message. The second time the value is above the threshold after 44 ms is not recognized as the value at the sample points before and after are both below the threshold.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved



Key

1 sample point: the measured value is above the threshold value – ROE event issue

2 sample point: the measured value is below the threshold value – no ROE event

3 sample point: the measured value is below the threshold value – no ROE event

4 measurement value is temporarily between two sample points above the threshold

5 sample point: the measured value is below the threshold value, which ends in a non-recognition of the threshold being exceeded – no ROE event

6 sample points

Figure 13 — Sampling of data identifiers

The SubFunctions onChangeOfDataIdentifier or onComparisonOfValues allow more than one event to be configured. The maximum number of allowed events that can get configured in the server is limited by parameters MaxNumChangeOfDataIdentfierEvents and MaxNumComparisionOfValueEvents. The values of these parameters have a direct impact on the server's RAM consumption.

A request with SubFunction onChangeOfDataIdentifier or onComparisonOfValues will add one more event to the server. The server shall add this event, independently from the fact that RoE is already started or stopped. Adding a new event will require that the server to sample the new DID data the next time the RoE scheduler is executed.

Configured events with SubFunction equal to onDTCStatusChange, reportMostRecentDtcOnStatusChange and reportDTCRecordInformationOnDtcStatusChange require the server to send an event if a certain DTC status change occurred. The server checks every ResponseOnEventSchedulerRate for changed DTC status bytes according to the provided DTCStatusMask and sends an event for each detected change. Therefore the server shall define the following parameters:

— MaxNumberOfStoredDTCStatusChangedEvents: the maximum number of DTCs that can be stored as DTCs with change status within one ResponseOnEventSchedulerRate interval.

The following implementation rules shall apply:

a) The ResponseOnEvent service can be set up and activated in any session, including the defaultSession. TesterPresent service is not necessarily required to keep the ResponseOnEvent service active.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

b) If the specified event occurs when a diagnostic service is in progress, which means that either a request message is in progress to be received, or a request is executed, or a response message is in progress (this includes the negative response message handling with response code 78_{16}) to be transmitted (if suppressPosRspMsgIndicationBit = FALSE) then the execution of the request message contained in the serviceToRespondToRecord shall be postponed until the completion of the diagnostic service in progress.

NOTE 2 In some circumstances due to the postponing of the ServiceToRespondTo the data contained in the ServiceToRespondTo Records cannot correspond with the data values which caused the event.

c) The server shall execute the service contained in the serviceToRespondToRecord when the event logic is satisfied and the event is generated within the event time window.

d) Once the ResponseOnEvent service is initiated by the ResponseOnEvent request "start", the server shall respond to the client which has set up the event logic and has started the ROE events till event window time expires.

e) A DiagnosticSessionControl request moving to any non-default session shall stop the ResponseOnEvent service regardless whether a different non-default session than the current session or the same non-default session is activated. On returning to DefaultSession all ResponseOnEvent services which had previously been active in DefaultSession shall be re-activated.

f) Multiple ResponseOnEvent services may run concurrently with different requirements (different EventTypes, serviceToRespondToRecords, ...) to start and stop diagnostic services. Start and stop SubFunctions shall always control all initialized ResponseOnEvent services.

g) If a ResponseOnEvent service has been set up and is started by startResponseOnEvent then the following shall apply:

   i) If bit 6 of the eventTypeSubFunction parameter is set to 0 (do not store event), then the event shall terminate when the server powers down. The server shall not continue a ResponseOnEvent diagnostic service after a reset or power on (i.e. the ResponseOnEvent service is terminated).

   ii) If bit 6 of the eventTypeSubFunction parameter is set to 1 (store event), it shall resume sending serviceToRespondTo-responses according to the ResponseOnEvent-setup after a power cycle of the server. StoreEvent is therefore only allowed in combination with infinite eventWindowTime.

h) The "suppressPosResponseMessageIndicationBit" = "yes" should only be used by the client for the eventType = stopResponseOnEvent, startResponseOnEvent or clearResponseOnEvent. The server shall always return a response to the event-triggered response when the specified event is detected. For the serviceToRespondToRecord the "suppressPosResponseMessageIndicationBit" shall always be set to 0.

i) The server shall return a ResponseOnEvent "final" response (see Figure 12) to indicate the ResponseOnEvent (86_{16}) service only if a finite window time was set and the finite window time has elapsed. No final response shall be sent if the ROE has been stopped by any means (e.g. "stopResponseOnEvent" SubFunction or change of session) before the finite window time has elapsed.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

j) The server shall return a ResponseOnEvent “intermediate” response to indicate that between two scheduler calls, more than MaxNumberOfStoredDTCStatusChangedEvents DTC status changes have occurred.

k) In order to avoid interference with normal diagnostic operation, it is recommended to use ResponseOnEvent service only to be applied to transient events and conditions. The server shall return a response once per event occurrence.

l) Only one response per ResponseOnEvent scheduler shall be transmitted to avoid high busload. In case multiple events are detected within a ResponseOnEvent scheduler, the first event shall be transmitted immediately. If the server uses a queue the following events/messages shall be distributed to the next cycles of ResponseOnEvent scheduler, otherwise the server will discard the events. New occurences of DID events shall be postponed after the current message transmissions. While the ResponseOnEvent service is active, the server shall be able to process concurrent diagnostic request and response messages accordingly, see Figure 14. This should be accomplished with a different serviceToRespondTo ECU Source Address. If the same diagnostic request/response ECU Address is used for diagnostic communication and the serviceToRespondTo-responses, the following restrictions shall apply:

   i) The server may ignore an incoming diagnostic request issued by the client that initiated the event, after an event has occurred and the serviceToRespondTo-response is in progress, until the serviceToRespondTo-response is completed. If another client issued the request, the server may or may not ignore the request, depending on the server’s capabilities.

   ii) When the client receives any response after sending a diagnostic request, the response shall be classified according to the possible serviceToRespondTo-responses and the expected diagnostic responses to the previously sent request.

      1) If the response is a serviceToRespondTo-response (one of the possible responses set up with ResponseOnEvent-service), the client shall repeat the request after the serviceToRespondTo-response has been received completely.

      2) If the response is ambiguous (i.e. the response could originate from the serviceToRespondTo initiated by an event or from the response to a diagnostic request), the client shall present the response both as a serviceToRespondTo-response and as the response to the diagnostic request. The client shall not repeat the request with the exception of NegativeResponseCode busyRepeatRequest (21_{16}) (see the negative response code definitions in this document).

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

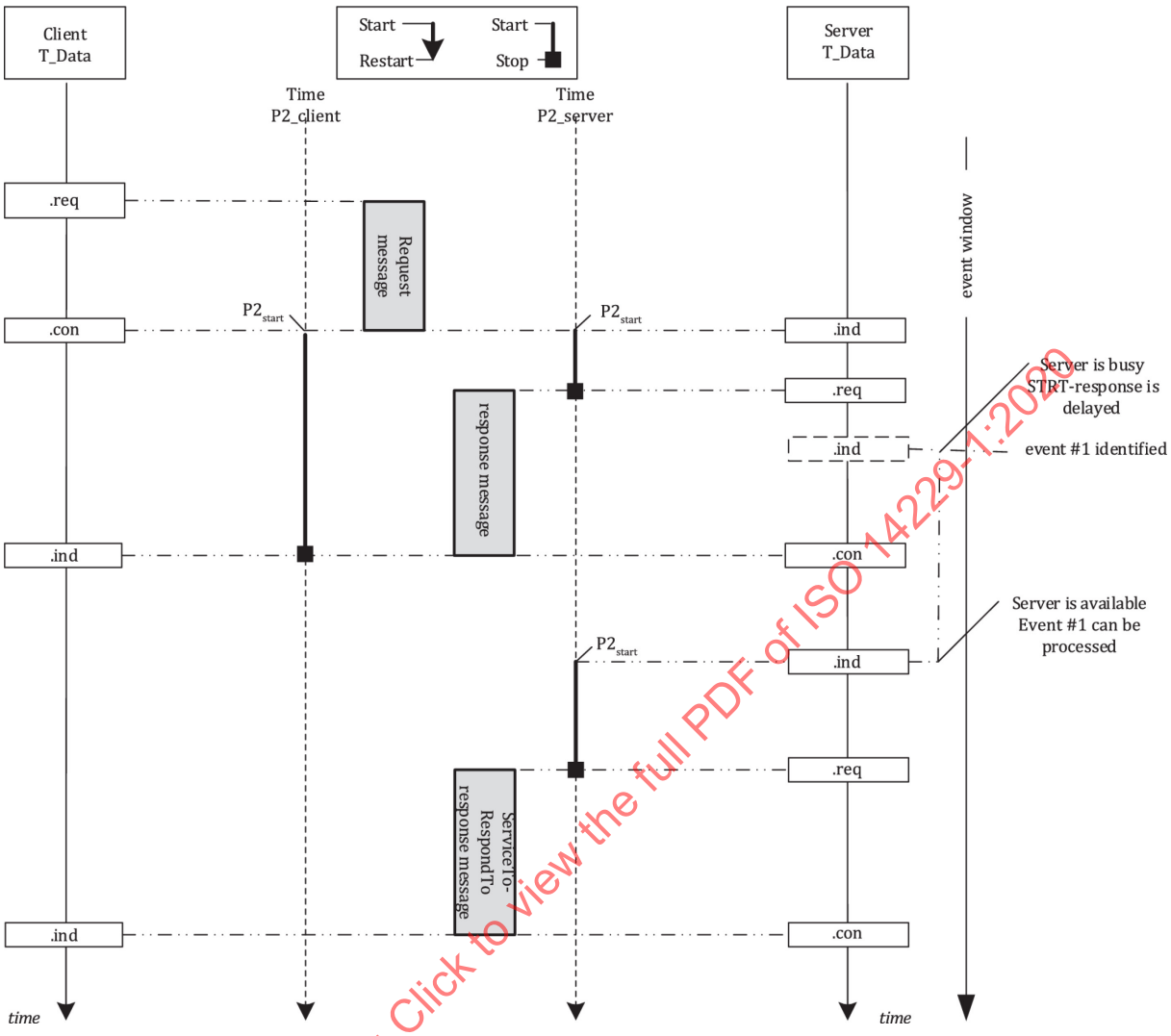


Figure 14 — Concurrent requests when the event occurs

iii) When the specified event is detected, the server shall respond immediately with the appropriate serviceToRespondTo-response message. The immediate serviceToRespondTo-response message shall not disrupt any other diagnostic request or response transmission already in progress (i.e. the serviceToRespondTo-response shall be delayed until the current message transmission has been completed, see Figure 15).

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

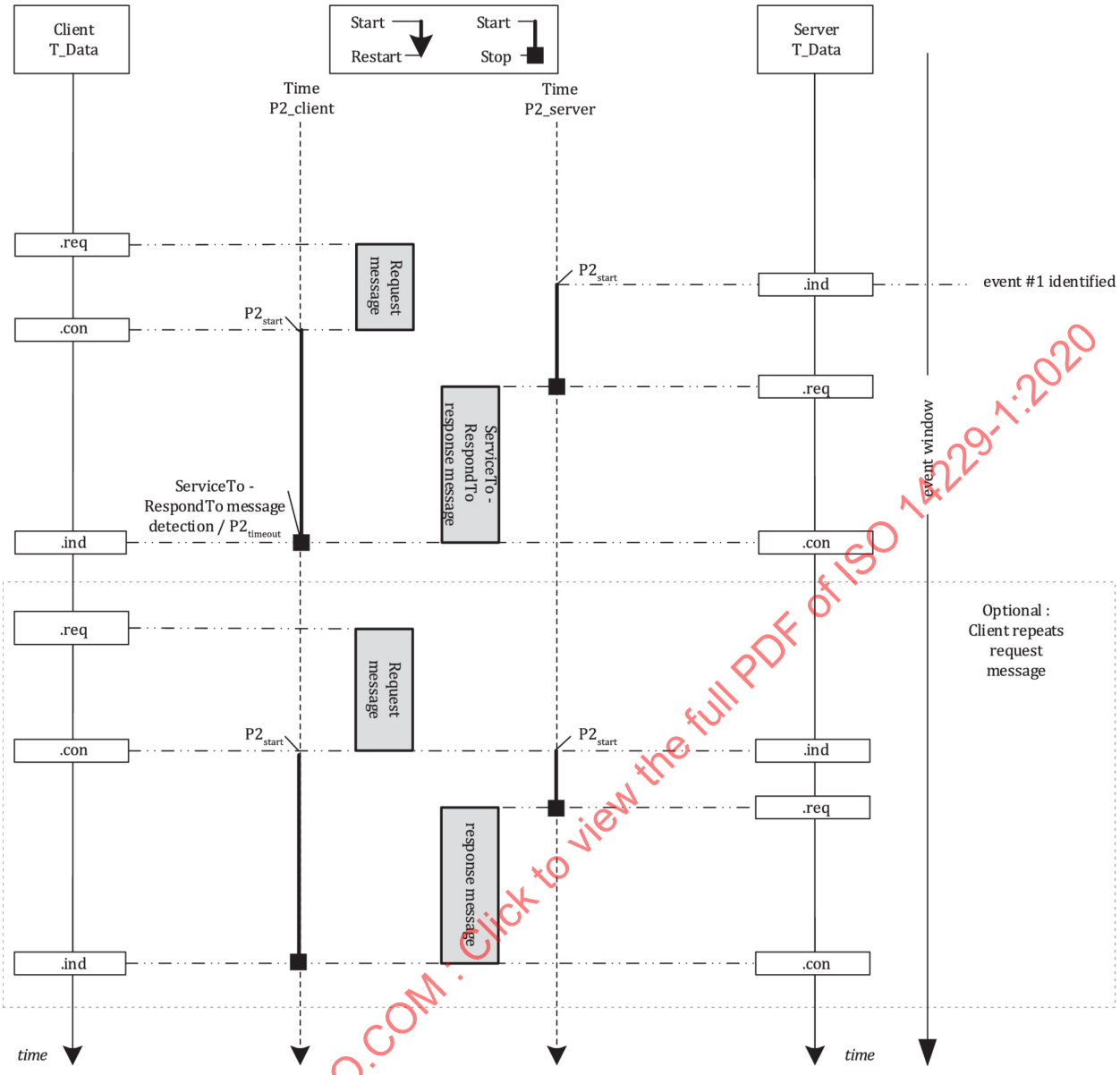


Figure 15 — Event occurrence during a message in progress

It is allowed to use only the services listed in Table 137 for the service to be performed in case the specified event occurs (serviceToRespondToRecord request service identifier).

Table 137 — Recommended services to be used with the ResponseOnEvent service

Recommended services (ServiceToRespondTo)	Request SID	Response SID
ReadDataByIdentifier	22_{16}	62_{16}
ReadDTCInformation	19_{16}	59_{16}

For performance reasons (e.g. avoid missed execution of serviceToRespondToRecord request service identifier) it is recommended to respect the following suggestions:

— The DID datalength should be as small as possible. A recommendation is to use maximum 32 bit data length. Longer data length will take more time to compare the values of the DID.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

— DID may contain measurable data (e.g. avoid definition of event logic reading constant “calibration” labels).

— serviceToRespondToRecord positive response may be limited in size up to a vehicle manufacturer specific value.

IMPORTANT — The server and the client shall meet the request and response message behaviour as specified in 8.7.

Each client has an own event logic, i.e. they can run independently of each other.

NOTE A shared event logic between different clients is not supported. If one client is changing the session to non-default session, the server will pause all response on event activities.

It is vehicle manufacturer specific which client supports ResponseOnEvent. In case ResponseOnEvent service is not supported for a specific client, the NRC 11_{16} ServiceNotSupported shall be returned for this client.

10.9.2 Request message

10.9.2.1 Request message definition

Table 138 specifies the request message.

Table 138 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	ResponseOnEvent Request SID	M	86_{16}	ROE
#2	SubFunction = [eventType]	M	00_{16} to FF_{16}	LEV_ETP
#3	eventWindowTime	M	00_{16} to FF_{16}	EWT
#4 : #(m-1)+4	eventTypeRecord[] = [eventTypeParameter 1 : eventTypeParameter m]	 C1 : C1	 00_{16} to FF_{16} : 00_{16} to FF_{16}	ETR_ ETP1 : ETPm
#n-(r-1)-1 #n-(r-1) : #n	serviceToRespondToRecord[] = [serviceId serviceParameter 1 : serviceParameter r]	 C2 C3 : C3	 00_{16} to FF_{16} 00_{16} to FF_{16} : 00_{16} to FF_{16}	STRTR_ SI SP1 : SPr

C1: Present if the eventType requires additional parameters to be specified for the event to respond to.

C2: Mandatory to be present if the SubFunction parameter is not equal to reportActivatedEvents, stopResponseOnEvent, startResponseOnEvent, ClearResponseOnEvent.

C3: Present if the service request of the service to respond to requires additional service parameters.

10.9.2.2 Request message SubFunction Parameter $Level (LEV_) Definition

10.9.2.2.1 ResponseOnEvent request message SubFunction Parameter definition

The SubFunction parameter eventType is used by the ResponseOnEvent request message to specify the event to be configured in the server and to control the ResponseOnEvent setup. Each SubFunction parameter value given in Table 139 also specifies the length of the applicable eventTypeRecord (suppressPosRspMsgIndicationBit (bit 7) not shown in table below).

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Bit 6 of the eventType SubFunction parameter is used to indicate whether the event shall be stored in non-volatile memory in the server and re-activated upon the next power-up of the server or if it shall terminate once the server powers down (storageState parameter).

Table 139 — eventType SubFunction bit 6 definition - storageState

Bit 6 value	Description	Cvt	Mnemonic
00_{16}	**doNotStoreEvent** This value indicates that the event shall terminate when the server powers down and the server shall not continue a ResponseOnEvent diagnostic service after a reset or power on (i.e. the ResponseOnEvent service is terminated). For SubFunctions different to startResponseOnEvent and stopResponseOnEvent only doNotStoreEvent is allowed.	M	DNSE
01_{16}	**storeEvent** This value indicates that on startResponseOnEvent request in the default session, that the event shall resume sending serviceToRespondTo-responses according to the ResponseOnEvent-setup after a reset or power on. The state is persisted within the server until any ResponseOnEvent request with a SubFunction different to startResponseOnEvent is received. Setting up a new event, clearing or stopping the response on event service, will also remove the persisted start for response of event after reset or power on of the server.	U	SE

Table 140 specifies the request message SubFunction parameter.

Table 140 — Request message SubFunction parameter definition

Bit 5 to 0 value	Description	Cvt	Type of ROE SubFunction	Mnemonic
00_{16}	**stopResponseOnEvent** This value is used to stop the server sending responses on event. The event logic that has been set up is not cleared but can be restarted with the startResponseOnEvent SubFunction parameter. Length of eventTypeRecord: 0 byte	U	control	STPROE
01_{16}	**onDTCStatusChange** This value identifies the event as a DTC status has changed while the DTC status matches the DTCStatusMask specified for this event. This is calculated by a logical AND operation on the DTC status and the provided DTCStatusMask. If the old and new DTC status is different it will trigger an event. DTC status changes that are caused by service ClearDiagnosticInformation (14_{16}) or aging will not trigger events. Length of eventTypeRecord: 1 byte This eventType requires the specification of the DTCStatusMask in the request message (eventTypeParameter#1).	U	setup	ONDTCS
02_{16}	**ISOSAEReserved** This range of values is reserved by this document for future definition.	M	-	ISOSAERESRVD

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Bit 5 to 0 value	Description	Cvt	Type of ROE SubFunction	Mnemonic
03_{16}	**onChangeOfDataIdentifier** This value identifies the event as a new internal data record identified by dataIdentifier. The data values are vehicle manufacturer specific. The event is added to the set of active events of the server. This eventType requires the specification of more details in the request message (eventTypeRecord). Length of eventTypeRecord: 2 bytes	U	setup	OCODID
04_{16}	**reportActivatedEvents** This value is used to indicate that in the positive response all events are reported that have been activated in the server with the ResponseOnEvent service (and are currently active). Length of eventTypeRecord: 0 bytes	U	control	RAE
05_{16}	**startResponseOnEvent** This value is used to indicate to the server to activate the event logic (including event window timer) that has been set up and to start sending responses on event. In case storeEvent in combination with an infinite time window is requested in the default session, the current setup is persistent to be resumed after power-down. Length of eventTypeRecord: 0 byte.	M	control	STRTROE
06_{16}	**clearResponseOnEvent** This value is used to clear the event logic that has been set up in the server. (This also stops the server sending responses on event.) Length of eventTypeRecord: 0 byte.	U	control	CLRROE
07_{16}	**onComparisonOfValues** A defined alteration of a data value out of a specific record identified by a dataIdentifier which identifies a data value event. With this SubFunction the user shall have the possibility to define an event at the occurrence of a specific result gathered from a defined measurement value comparison. A specific measurement value included in a data record assigned to a defined dataIdentifier is compared with a given comparison value. The specified operator specifies the kind of comparison. The event occurs if the comparison result is positive. Length of eventTypeRecord: 10 bytes	U	setup	OCOV
08_{16}	**reportMostRecentDtcOnStatusChange** The value identifies the event as a new DTC detected having a changed testFailed (Bit0) or confirmedDTC (Bit3) DTC status bit transition from 0 to 1. There is no serviceToResponseToRecord for this SubFunction. Length of eventTypeRecord: 1 Byte	U	control	RMRDOSC

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Bit 5 to 0 value	Description	Cvt	Type of ROE SubFunction	Mnemonic
09_{16}	**reportDTCRecordInformationOnDtcStatusChange** This value identifies the event as a DTC status has changed while the DTC status matches the DTCStatusMask specified for this event. This is calculated by a logical AND operation on the DTC status and the provided DTCStatusMask. If the old and new DTC status is different it will trigger an event. DTC status changes that are caused by service ClearDiagnosticInformation (14_{16}) or aging will not trigger events. Length of eventTypeRecord: 3-4 byte This eventType requires the specification of the DTCStatusMask in the request message (eventTypeParameter#1). The eventTypeParameter#2 define the SubFunction of service ReadDTCInformation that is used in the response on event. Allowed values are all SubFunctions of service ReadDTCInformation having the DTCMaskRecord in the request of the service. The eventTypeParameter#3 to #n are the ReadDTCInformation request parameters of that SubFunction that are present after the DTCMaskRecord in the request message.	U	control	RDRIODSC
$0A_{16}$ to $3F_{16}$	**ISOSAEReserved** This range of values is reserved by this document for future definition.	M	-	ISOSAERESRVD

10.9.2.2.2 Detailed request message SubFunction onDTCStatusChange parameters specification

With subfunction onDTCStatusChange the server monitors changes of DTC status and sends events if a DTC status has changed that matches the DTC status mask.

For SubFunctions reportMostRecentTestFailedDTC ($0D_{16}$) with a DTCStatusMask of 01_{16} or reportMostRecentConfirmedDTC ($0E_{16}$) with a DTCStatusMask of 08_{16}, the event shall contain the DTCAndStatusRecord of the DTC with the changed status.

10.9.2.2.3 Detailed request message SubFunction reportMostRecentDtcOnStatusChange parameters specification

This SubFunction guarantees with a FIFO queue a reporting of all changed DTCs in comparison of onDTCStatusChange SubFunction. A queue overrun is signalled by a responseOnEvent "intermediate" response message with numberOfIdentifiedEvents set to FF_{16}. This intermediate response message allows the tester to re-synchronize its internal DTC states with e.g. 19_{16} 02_{16}.

With SubFunction reportMostRecentDtcOnStatusChange the server monitors changes of testFailed (Bit 0) or confirmedDTC (Bit 3) DTC status bits and sends a response on changes.

The eventTypeRecord of the request message has a length one byte and allows the values $0D_{16}$ and $0E_{16}$. All other values are reserved and shall not be used. A provided eventTypeRecord of $0D_{16}$ specifies the response on event trigger on DTC status changes of the testFailed (Bit 0) from 0 to 1. In this case the response on event message is defined by the positive response of the ReadDTCInformation service with SubFunction reportMostRecentTestFailedDTC having the DTCAndStatusRecord with the DTC number and DTC status of the DTC that triggered the event. A provided eventTypeRecord of $0E_{16}$ specifies the response on event trigger on DTC status changes of the confirmedDTC (Bit 3) from 0 to 1. In this case

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

the response on event message is defined by the positive response of the ReadDTCInformation service with SubFunction reportMostRecentConfirmedDTC having the DTCAndStatusRecord with the DTC number and DTC status of the DTC that triggered the event.

The server is directly sending the response without general server response behaviour according to Figure 5.

This SubFunction does not use a serviceToRespondToRecord, as it is implicitly defined by the first byte in the eventTypeRecord. The value of this byte is aligned with the SubFunction of the service ReadDTCInformation.

Back to back responses are allowed for this SubFunction to allow a better dimensioning of the MaxNumberOfStoredDTCStatusChangedEvents queue (the maximum burst on the network is equal to the size of the MaxNumberOfStoredDTCStatusChangedEvents queue).

10.9.2.2.4 Detailed request message SubFunction reportDTCRecordInformationOnDtcStatusChange parameters specification

With subfunction reportDTCRecordInformationOnDtcStatusChange the server monitors changes of DTC status and sends a response on event message if a DTC status has changed that matches the DTC status mask. The send response on event message is fixed to SubFunctions of service ReadDTCInformation having the DTCMaskRecord as parameter of the request message. The DTCMaskRecord of that request message is replaced with the 3 byte DTC number of the DTC whose status change was detected.

This allows the server to report further information such as extended data records or snapshot records based on the DTC that changed the status. A use of a serviceToRespondToRecord with definition of diagnostic requests for ReadDTCInformation would not allow to read information for the DTC with the changed status as the DTCMaskRecord would be fixed within the serviceToRespondToRecord.

This allows to get within one response message the changed DTC, its status and, e.g. the detailed timestamp of the occurrence stored within a Snapshot- or ExtendedDataRecord.

10.9.2.2.5 Detailed request message SubFunction onChangeOfDataIdentifier parameters specification

WithSubFunction onChangeOfDataIdentifier the server is allowed to poll the measurements in a fixed period of time and compare the content, therefore it is acceptable that the server might lose some changes and trigger less events than expected.

The eventTypeRecord sets the two byte DID value that shall be monitored for any change. Dynamically Defined DIDs in the range between $F200_{16}$ – $F3FF_{16}$ are excluded.

NOTE: Statically defined DIDs are allowed.

10.9.2.2.6 Detailed request message SubFunction onComparisonOfValues parameters specification

Tables 141 to 143 specify the parameters for the request message with SubFunction onComparisonOfValues parameters in case of serviceToRespondToRecord specifying a comparison between read DIDs.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 141 — SubFunction onComparisonOfValues parameters definition

Data byte	Parmeter name	Byte value	Comment	Detailed requirement
1	ServiceID	86_{16}	Request SID	---
2	eventType	07_{16}	SubFunction onComparisonOfValues	---
3	eventWindowTime	02_{16}	InfiniteTimeWindow specification	---
4	eventTypeRecord byte1	01_{16}	DataIdentifier (DID) high byte	Can be a different DID than the one used in serviceToRespondToRecord. Can be a dynamically defined DID.
5	eventTypeRecord byte 2	04_{16}	DataIdentifier (DID) low byte	---
6	eventTypeRecord byte 3	01_{16}	Comparison logic, see Table 142	The eventTypeRecord byte 3 sets the logics of the comparison method.
7	eventTypeRecord byte 4	00_{16}	Raw reference comparison value MSB	The eventTypeRecord byte 4, 5, 6, 7 sets the reference comparison value.
8	eventTypeRecord byte 5	00_{16}	Raw reference comparison value	---
9	eventTypeRecord byte 6	01_{16}	Raw reference comparison value	---
10	eventTypeRecord byte 7	00_{16}	Raw reference comparison value LSB	---
11	eventTypeRecord byte 8	$0A_{16}$	hysteresis value	The eventTypeRecord byte 8 defines an histeresys value in percentage from 0 % (00_{16}) to 100 % (64_{16}).
12	eventTypeRecord byte 9	BC_{16}	Localization byte 1 MSB, see Table 143	The eventTypeRecord byte 9, 10 defines localization of value within the data identifier, see Table 143.
13	eventTypeRecord byte 10	58_{16}	Localization byte 2 LSB, see Table 143	---
14	serviceToRespondTo byte 1	22_{16}	SID	The serviceToRespondToRecord sets the service and the DID to be read and compared.
15	serviceToRespondTo byte 2	$A1_{16}$	DID1	---
16	serviceToRespondTo byte 3	00_{16}	DID2	---

Table 142 specifies the comparison logic parameter definition.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 142 — Comparison logic parameter definition

Comparison logic ID	Event will be generated when
01_{16}	Comparison Parameter < Measured Value
02_{16}	Comparison Parameter > Measured Value
03_{16}	Comparison Parameter = Measured Value
04_{16}	Comparison Parameter <> Measured Value

Table 143 specifies the localization of value 16 bit bitfield parameter definition.

Table 143 — Localization of value 16 bit bitfield parameter definition

Bitfield bit position	Description
15	(MSB), bit = 0 means comparison without sign, bit = 1 comparison with sign
14 - 10	Bit#10 (LSB) - Bit#14 (MSB) contain the length of data identifier value to be compared. The value 00_{16} shall be used to compare all 4 bytes. All other values shall set the size in bits. With 5 bits, the maximal size of a length is 31 bits.
9 - 0	Offset on the positve response message from where to extract the data identifier value. Bit#0 (LSB) - Bit#9 (MSB) contain the start bit number offset. With 10 bits, the maximal size of an offset is 1 023 bits.

10.9.2.3 Request message data-parameter definition

Table 144 specifies the data-parameters of the request message.

Table 144 — Request message data-parameter definition

Definition
eventWindowTime The parameter eventWindowTime is used to specify a window for the event logic to be active in the server. If the parameter value of eventWindowTime is set to 02_{16} then the response time is infinite. In case of an infinite event window and storageState equal to doNotStoreEvent it is recommended to close the event window by a certain signal (e.g. power off). See B.2 for specified eventWindowTimes. A combination of finite event window and storageState equal to storeEvent shall not be used. NOTE This parameter is not applicable to be evaluated by the server in case the eventType is equal to a ROE control SubFunction.
eventTypeRecord This parameter record contains additional parameters for the specified eventType.
serviceToRespondToRecord This parameter record contains the service parameters (service Id and service parameters) of the service to be executed in the server each time the specified event defined in the eventTypeRecord occurs.

10.9.3 Positive response message

10.9.3.1 Positive response message definition

Table 145 specifies the positive response message for all SubFunctions but reportActivatedEvents.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 145 — Positive response message definition for all SubFunctions but reportActivatedEvents

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	ResponseOnEvent Response SID	M	$C6_{16}$	ROEPR
#2	eventType	M	00_{16} to $7F_{16}$	ETP
#3	numberOfIdentifiedEvents	M	00_{16} to FF_{16}	NOIE
#4	eventWindowTime	M	00_{16} to FF_{16}	EWT
#5 : #(m-1)+5	eventTypeRecord[] = [eventTypeParameter 1 : eventTypeParameter m]	C1 : C1	00_{16} to FF_{16} : 00_{16} to FF_{16}	ETR_ ETP1 : ETPm
#n-(r-1)-1 #n-(r-1) : #n	serviceToRespondToRecord[] = [serviceId serviceParameter 1 : serviceParameter r]	M C2 : C2	00_{16} to FF_{16} 00_{16} to FF_{16} : 00_{16} to FF_{16}	STRTR_ SI SP1 : SPr
C1: Present if the eventType required additional parameters to be specified for the event to respond to. C2: Present if the service request of the service to respond to required additional service parameters.				

Table 146 specifies the positive response message for SubFunction = reportActivatedEvents.

Table 146 — Positive response message definition for SubFunction = reportActivatedEvents

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	ResponseOnEvent Response SID	M	$C6_{16}$	ROEPR
#2	eventType = reportActivatedEvents	M	04_{16}	ETP_RAE
#3	numberOfActivatedEvents	M	00_{16} to FF_{16}	NOIE
#4	eventTypeOfActiveEvent#1	C1	00_{16} to FF_{16}	EVOAE
#5	eventWindowTime#1	C1	00_{16} to FF_{16}	EWT
#6 : #(m-1)+6	eventTypeRecord#1[] = [eventTypeParameter 1 : eventTypeParameter m]	C2 : C2	00_{16} to FF_{16} : 00_{16} to FF_{16}	ETR_ ETP1 : ETPm
#p-(o-1)-1 #p-(o-1) : #p	serviceToRespondToRecord#1[] = [serviceId serviceParameter 1 : serviceParameter o]	C3 C4 : C4	00_{16} to FF_{16} 00_{16} to FF_{16} : 00_{16} to FF_{16}	STRTR_ SI SP1 : SPo
:	:	:	:	:
#n-(r-1)-4-(q-1)	eventTypeOfActiveEvent#k	C1	00_{16} to FF_{16}	EVOAE
#n-(r-1)-3-(q-1)	eventWindowTime#k	C1	00_{16} to FF_{16}	EWT

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#n-(r-1)-2-(q-1) : #n-(r-1)-2	eventTypeRecord#k[] = [eventTypeParameter 1 : eventTypeParameter q]	 C2 : C2	 00_{16} to FF_{16} : 00_{16} to FF_{16}	ETR_ ETP1 : ETPm
#n-(r-1)-1 #n-(r-1) : #n	serviceToRespondToRecord#k[] = [serviceId serviceParameter 1 : serviceParameter r]	 C3 C4 : C4	 00_{16} to FF_{16} 00_{16} to FF_{16} : 00_{16} to FF_{16}	STRTR_ SI SP1 : SPr
C1: Present if an active event is reported. C2: Present if the reported eventType of the active event (eventTypeOfActiveEvent) requires additional parameters to be specified for the event to respond to. C3: Mandatory to be present when reporting an active event. C4: Present if the reported service request of the service to respond to requires additional service parameters.				

10.9.3.2 Positive response message data-parameter definition

Table 147 specifies the data-parameters of the positive response message.

Table 147 — Response message data-parameter definition

Definition
eventType This parameter is an echo of bits 6 - 0 of the SubFunction parameter of the request message.
eventTypeOfActiveEvent This parameter is an echo of the SubFunction parameter of the request message that was issued to set up the active event. The applicable values are the ones specified for the eventType SubFunction parameter.
numberOfActivatedEvents This parameter contains the number of active events when the client requests to report the number of active events. This number reflects the number of events reported in the response message.
numberOfIdentifiedEvents This parameter contains the number of identified events during an active event window and is only applicable for the response message sent at the end of the event window (in case of a finite event window). The initial response to the request message shall contain a zero (0) in this parameter. A value of FF_{16} identifies an overflow of the server internal queue to store occurred events. A value of FE_{16} is the upper limit for the number of identified events (i.e. even if more than 254 events happened, the value within the final response message shall not exceed FE_{16}).
eventWindowTime This parameter is an echo of the eventWindowTime parameter from the request message. When reporting an active event then this parameter contains the time remaining for the event to be active.
eventTypeRecord This parameter is an echo of the eventTypeRecord parameter from the request message. When reporting an active event then this parameter is an echo of the eventTypeRecord of the request that was issued to set up the active event.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Definition
serviceToRespondToRecord This parameter is an echo of the serviceToRespondToRecord parameter from the request message. When reporting an active event then this parameter is an echo of the serviceToRespondToRecord of the request that was issued to set up the active event.

10.9.4 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 148. The listed negative responses shall be used if the error scenario applies to the server.

Table 148 — 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 request message is wrong.	IMLOIF
22_{16}	**conditionsNotCorrect** Used when the server is in a critical normal mode activity and therefore cannot perform the requested functionality.	CNC
31_{16}	**requestOutOfRange** The server shall use this response code: — if it detects an error in the eventTypeRecord parameter; — if the specified eventWindowTime is invalid; — if the requested DID is not supported or the DID length exceeds the supported length of the server; — if a combination of finite event window and storageState equal to storeEvent is requested; — if the serviceToRespondToRecord has a “suppressPosResponseMessageIndicationBit” set to ‘1’; — if the serviceToRespondToRecord is larger than the maximum length supported in the server; — if the serviceToRespondToRecord defines a not supported service — if the SubFunction is equal to onChangeOfDataIdentifier and the number of active configured onChangeOfDataIdentifier events of this kind is already equal to MaxNumChangeOfDataIdentfierEvents; — if the SubFunction is equal to onComparisonOfValues and the number of active configured onComparisonOfValues events of this kind is already equal to	ROOR

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

NRC	Description	Mnemonic
	MaxNumComparisionOfValueEvents; — if the SubFunction is equal to onChangeOfDataIdentifier or onComparisonOfValues and the dataIdentifier number is dynamically defined and within the range of $F200_{16}$ – $F3FF_{16}$; — the storeEvent bit is set and — a SubFunction different to startResponseOnEvent was requested; — the eventWindowTime is different to infinite; — a non-default session is active.	

10.9.5 Message flow example(s) ResponseOnEvent

10.9.5.1 Assumptions

For the message flow examples it is assumed, that the eventWindowTime equal to 08_{16} defines an event window of 80 s (eventWindowTime × 10 s). The client requests to have a response message by setting the suppressPosRspMsgIndicationBit (bit 7 of the SubFunction parameter) to "FALSE" ('0').

NOTE The definition of the eventWindowTime is vehicle manufacturer specific, except for certain values as specified in B.2.

The following conditions apply to the shown message flow examples and flowcharts:

— **Trigger signal:** It is up to the vehicle manufacturer to define a specific trigger signal, which causes the client (external test equipment, OBD-Unit, diagnostic master, etc.) to start the ResponseOnEvent request message. This trigger signal could be enabled by an event as well as by a fixed timing schedule like a heartbeat-time (which should be greater than the eventWindowTime). Furthermore there could be a synchronous message (e.g. SYNCH-signal) on the data link used as trigger signal.

— **Open event window:** Receiving the ResponseOnEvent request message, the server shall evalaute the request. If the evaluation was positive, the server shall set up the event logic and shall send the initial positive response message of the ResponseOnEvent service. To activate the event logic the client shall request ResponseOnEvent SubFunction startResponseOnEvent. After the positive response the event logic is activated and the event window timer is running. It is up to the vehicle manufacturer to define the event window in detail, using the parameter eventWindowTime (e.g. timing window, ignition on/off window). In case of detecting the specified eventType (EART_) the server shall respond immediately with the response message corresponding to the serviceToRespondToRecord in the ResponseOnEvent request message.

— **Close event window:** It is recommended to close the event window of the server according to the parameter eventWindowTime. After this action, the server shall stop sending event driven diagnostic response messages. The same could either be reached by sending the ResponseOnEvent (ROE_) request message including the parameter stopResponseOnEvent or by power off.

10.9.5.2 Example #1 - ResponseOnEvent (finite event window)

Table 149 specifies the setup of ResponseOnEvent request message flow example #1.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 149 — Setup of ResponseOnEvent request message flow example #1

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ResponseOnEvent Request SID	86_{16}	ROE
#2	eventTypeRecord [eventType] = onDTCStatusChange, storageState = doNotStoreEvent suppressPosRspMsgIndicationBit = FALSE	01_{16}	ET_ODTCSC
#3	eventWindowTime = 80 seconds	08_{16}	EWT
#4	eventTypeRecord [eventTypeParameter] = testFailed status	01_{16}	ETP1
#5	serviceToRespondToRecord [serviceId] = ReadDTCInformation	19_{16}	RDTCI
#6	serviceToRespondToRecord [SubFunction] = reportNumberOfDTCByStatusMask	01_{16}	RNDTC
#7	serviceToRespondToRecord [DTCStatusMask] = testFailed status	01_{16}	DTCSM

Table 150 specifies the ResponseOnEvent initial positive response message flow example #1.

Table 150 — ResponseOnEvent initial positive response message flow example #1

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ResponseOnEvent Response SID	$C6_{16}$	ROEPR
#2	eventType = onDTCStatusChange	01_{16}	ET_ODTCSC
#3	numberOfIdentifiedEvents = 0	00_{16}	NOIE
#4	eventWindowTime = 80 seconds	08_{16}	EWT
#5	eventTypeRecord [eventTypeParameter] = testFailed status	01_{16}	ETP1
#6	serviceToRespondToRecord [serviceId] = ReadDTCInformation	19_{16}	RDTCI
#7	serviceToRespondToRecord [SubFunction] = reportNumberOfDTCByStatusMask	01_{16}	RNDTC
#8	serviceToRespondToRecord [DTCStatusMask] = testFailed status	01_{16}	DTCSM

The event logic is set up; now it shall be activated.

Table 151 specifies the start of the ResponseOnEvent request message flow example #1.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

Table 151 — Start of ResponseOnEvent request message flow example #1

Message direction	client → server		
Message type	Request		
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	ResponseOnEvent Request SID	86_{16}	ROE
#2	eventTypeRecord [eventType] = startResponseOnEvent, storageState = doNotStoreEvent, suppressPosRspMsgIndicationBit = FALSE	05_{16}	ET_STRTROE
#3	eventWindowTime (will not be evaluated)	08_{16}	EWT

Table 152 specifies the ResponseOnEvent positive response message flow example #1.

Table 152 — ResponseOnEvent positive response message flow example #1

Message direction	server → client		
Message type	Response		
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	ResponseOnEvent Response SID	$C6_{16}$	ROEPR
#2	eventType = onDTCStatusChange	01_{16}	ET_ODTCSC
#3	numberOfIdentifiedEvents = 0	00_{16}	NOIE
#4	eventWindowTime	08_{16}	EWT

In case the specified event occurs the server sends the response message according to the specified serviceToRespondToRecord.

Table 153 specifies the ReadDTCInformation positive response message flow example #1.

Table 153 — ReadDTCInformation positive response message flow example #1

Message direction	server → client		
Message type	Response		
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	ReadDTCInformation Response SID	59_{16}	RDTCI
#2	reportType = reportNumberOfDTCByStatusMask	01_{16}	RNODTCBSM
#3	DTCStatusAvailibilityMask	FF_{16}	DTCSAM
#4	DTCFormatIdentifier = SAE_J2012-DA_DTCFormat_00	00_{16}	J2012-DADTCF00
#5	DTCCount [DTCCountHighByte] = 0	00_{16}	DTCCNT_HB
#6	DTCCount [DTCCountLowByte] = 4	04_{16}	DTCCNT_LB

The message flow for the case where the client would request to report the currently active events in the server during the active event window will look as follows.

Table 154 specifies the ResponseOnEvent request number of active events message flow example #1.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 154 — ResponseOnEvent request number of active events message flow example #1

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ResponseOnEvent Request SID	86_{16}	ROE
#2	eventTypeRecord [eventType] = reportActivatedEvents, storageState = doNotStoreEvent, suppressPosRspMsgIndicationBit = FALSE	04_{16}	ET_RAE
#3	eventWindowTime (will not be evaluated)	08_{16}	EWT

Table 155 specifies the ResponseOnEvent reportActivatedEvents positive response message flow example #1.

Table 155 — ResponseOnEvent reportActivatedEvents positive response message flow example #1

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ResponseOnEvent Response SID	$C6_{16}$	ROEPR
#2	eventType = reportActivatedEvents	04_{16}	ET_RAE
#3	numberOfActivatedEvents = 1	01_{16}	NOAE
#4	eventTypeOfActiveEvent = onDTCStatusChange	01_{16}	ET_ODTCSC
#5	eventWindowTime = 80 seconds	08_{16}	EWT
#6	eventTypeRecord [eventTypeParameter] = testFailed status	01_{16}	ETP1
#7	serviceToRespondToRecord [serviceId] = ReadDTCInformation	19_{16}	RDTCI
#8	serviceToRespondToRecord [SubFunction] = reportNumberOfDTCByStatusMask	01_{16}	RNDTC
#9	serviceToRespondToRecord [DTCStatusMask] = testFailed status	01_{16}	DTCSM

If the specified event window time has expired the server shall send a final positive response.

Table 156 specifies the ResponseOnEvent final positive response message flow example #1.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 156 — ResponseOnEvent final positive response message flow example #1

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ResponseOnEvent Response SID	$C6_{16}$	ROEPR
#2	eventType = onDTCStatusChange	01_{16}	ET_ODTCSC
#3	numberOfIdentifiedEvents = 1	01_{16}	NOIE
#4	eventWindowTime = 80 seconds	08_{16}	EWT
#5	eventTypeRecord [eventTypeParameter] = testFailed status	01_{16}	ETP1
#6	serviceToRespondToRecord [serviceId] = ReadDTCInformation	19_{16}	RDTCI
#7	serviceToRespondToRecord [SubFunction] = reportNumberOfDTCByStatusMask	01_{16}	RNDTC
#8	serviceToRespondToRecord [DTCStatusMask] = testFailed status	01_{16}	DTCSM

The following flowcharts show two different kind of server behaviour:

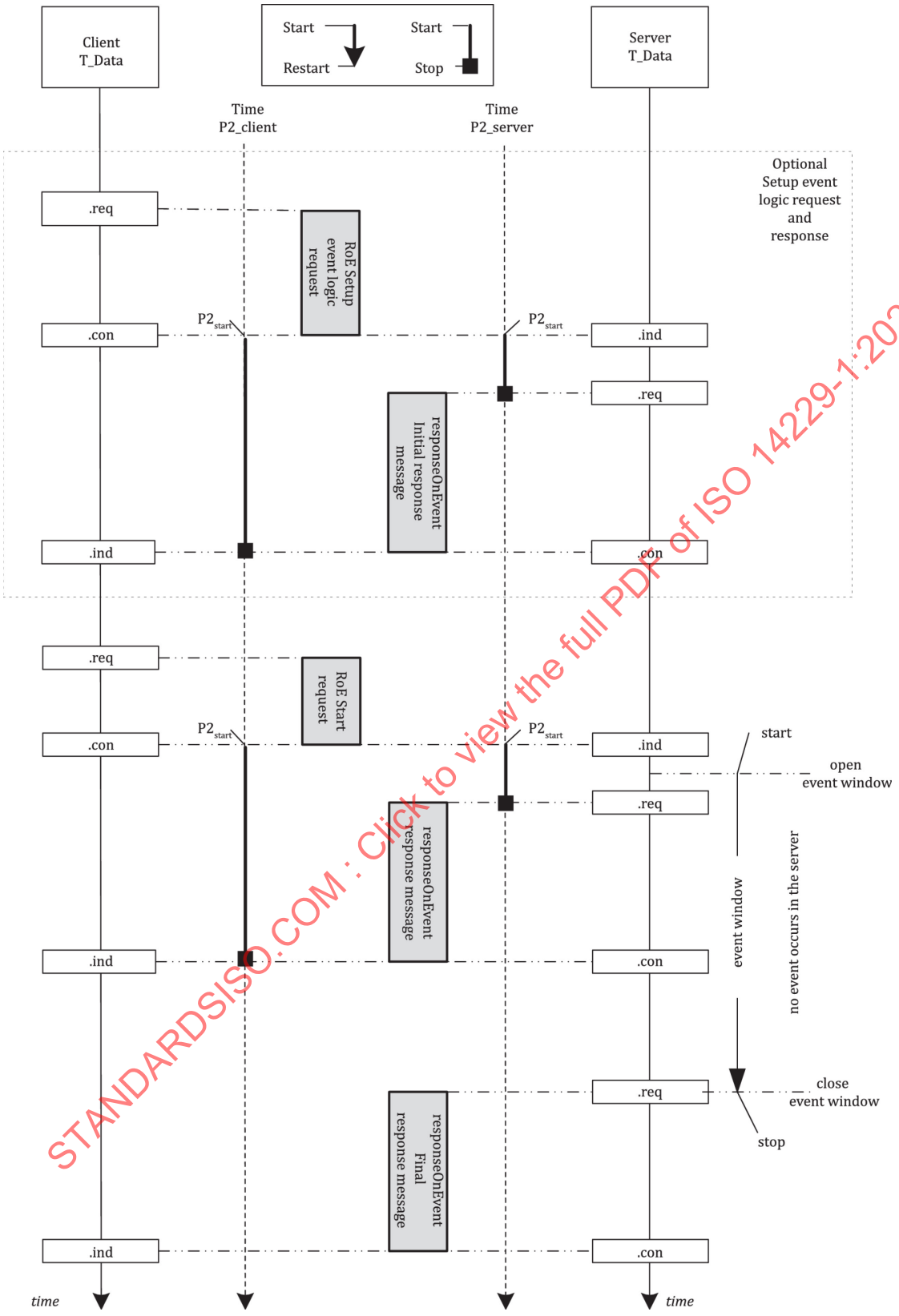
— No event occurs within the finite event window. In this case the server shall send the response of the ResponseOnEvent at the end of the event window.

— Multiple events (#1 to #n) within a finite event window. Each positive response of the serviceToRespondTo is related to an identified event (#1 to #n) and shall have the same service identifier (SId) but might have different content. At the end of the event_Window the server shall transmit a positive response message of the responseOnEvent service, which indicates the numberOfIdentifiedEvents.

Figure 16 depicts the finite event window – no event during active event window.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

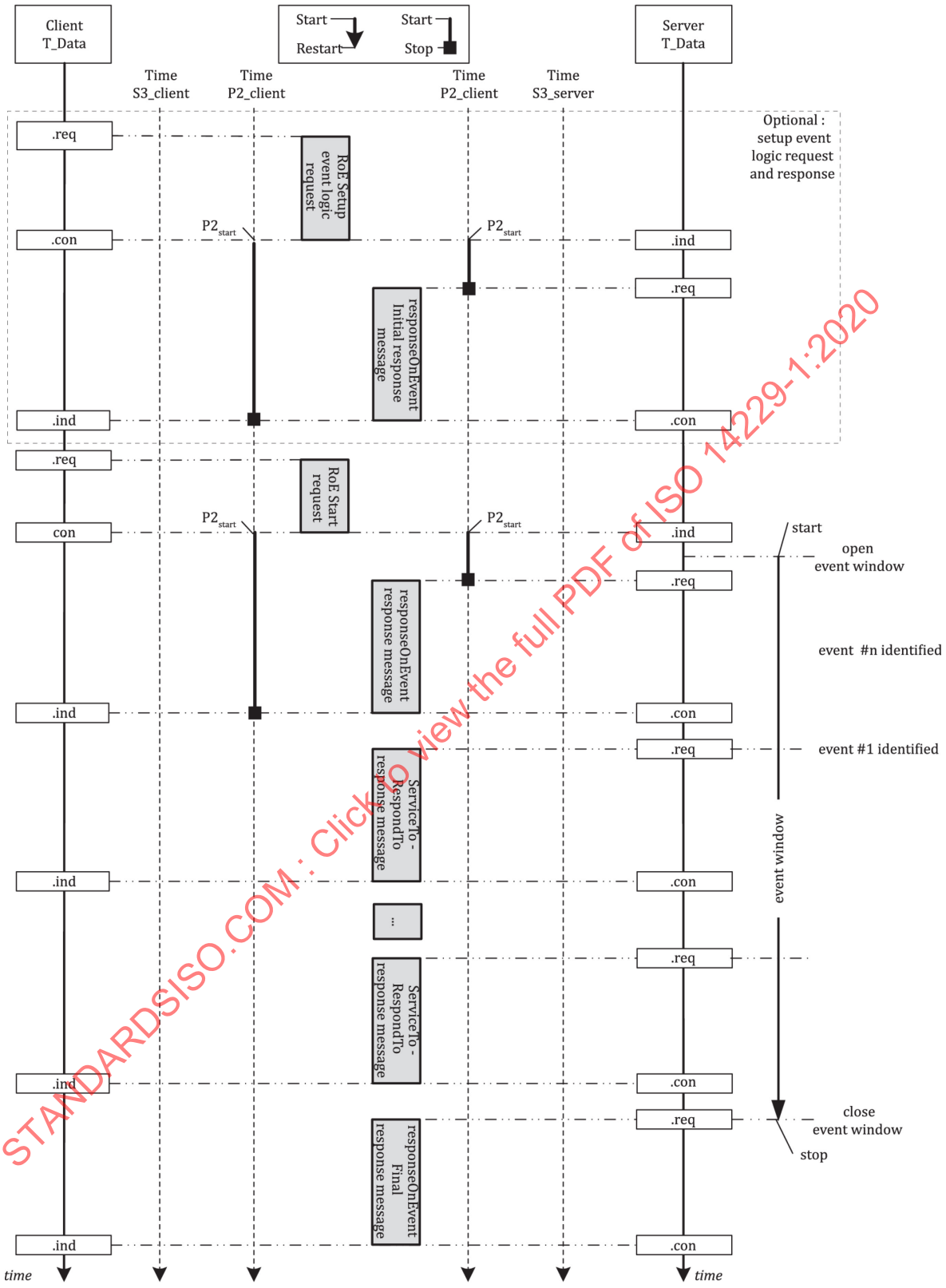


STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

Figure 16 — Finite event window – No event during active event window

Figure 17 depicts the finite event window – multiple events during active event window.

© ISO 2020 – All rights reserved



STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

Figure 17 — Finite event window - multiple events during active event window

10.9.5.3 Example #2 - ResponseOnEvent (infinite event window)

Table 157 specifies the ResponseOnEvent request message flow example #2.

© ISO 2020 – All rights reserved

Table 157 — ResponseOnEvent request message flow example #2

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ResponseOnEvent Request SID	86_{16}	ROE
#2	eventTypeRecord [eventType] = onDTCStatusChange, storageState = doNotStoreEvent, suppressPosRspMsgIndicationBit = FALSE	01_{16}	ET_ODTCSC
#3	eventWindowTime = infinite	02_{16}	EWT
#4	eventTypeRecord [eventTypeParameter] = testFailed status	01_{16}	ETP1
#5	serviceToRespondToRecord [serviceId] = ReadDTCInformation	19_{16}	RDTCI
#6	serviceToRespondToRecord [SubFunction] = reportNumberOfDTCByStatusMask	01_{16}	RNDTC
#7	serviceToRespondToRecord [DTCStatusMask] = testFailed status	01_{16}	DTCSM

Table 158 specifies the ResponseOnEvent initial positive response message flow example #2.

Table 158 — ResponseOnEvent initial positive response message flow example #2

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ResponseOnEvent Response SID	$C6_{16}$	ROEPR
#2	eventType = onDTCStatusChange	01_{16}	ET_ODTCSC
#3	numberOfIdentifiedEvents = 0	00_{16}	NOIE
#4	eventWindowTime = infinite	02_{16}	EWT
#5	eventTypeRecord [eventTypeParameter] = testFailed status	01_{16}	ETP1
#6	serviceToRespondToRecord [serviceId] = ReadDTCInformation	19_{16}	RDTCI
#7	serviceToRespondToRecord [SubFunction] = reportNumberOfDTCByStatusMask	01_{16}	RNDTC
#8	serviceToRespondToRecord [DTCStatusMask] = testFailed status	01_{16}	DTCSM

The event logic is set up, now it shall be activated.

Table 159 specifies the start of ResponseOnEvent request message flow example #2.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 159 — Start of ResponseOnEvent request message flow example #2

Message direction	client → server		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ResponseOnEvent Request SID	86_{16}	ROE
#2	eventTypeRecord [eventType] = startResponseOnEvent, storageState = doNotStoreEvent, suppressPosRspMsgIndicationBit = FALSE	05_{16}	ET_STRTROE
#3	eventWindowTime (will not be evaluated)	02_{16}	EWT

Table 160 specifies the ResponseOnEvent positive response message flow example #2.

Table 160 — ResponseOnEvent positive response message flow example #2

Message direction	server → client		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ResponseOnEvent Response SID	$C6_{16}$	ROEPR
#2	eventType = onDTCStatusChange	05_{16}	ET_ODTCSC
#3	numberOfIdentifiedEvents = 0	00_{16}	NOIE
#4	eventWindowTime	02_{16}	EWT

In case the specified event occurs the server sends the response message according to the specified serviceToRespondToRecord.

Table 161 specifies the ReadDTCInformation positive response message flow example #2.

Table 161 — ReadDTCInformation positive response message flow example #2

Message direction	server → client		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadDTCInformation Response SID	59_{16}	RDTCI
#2	DTCStatusAvailibilityMask	XX_{16}	DTCSAM
#3	DTCCount [DTCCountHighByte] = 0	00_{16}	DTCCNT_HB
#4	DTCCount [DTCCountLowByte] = 4	04_{16}	DTCCNT_LB

The following flowcharts show two different kind of server behaviour:

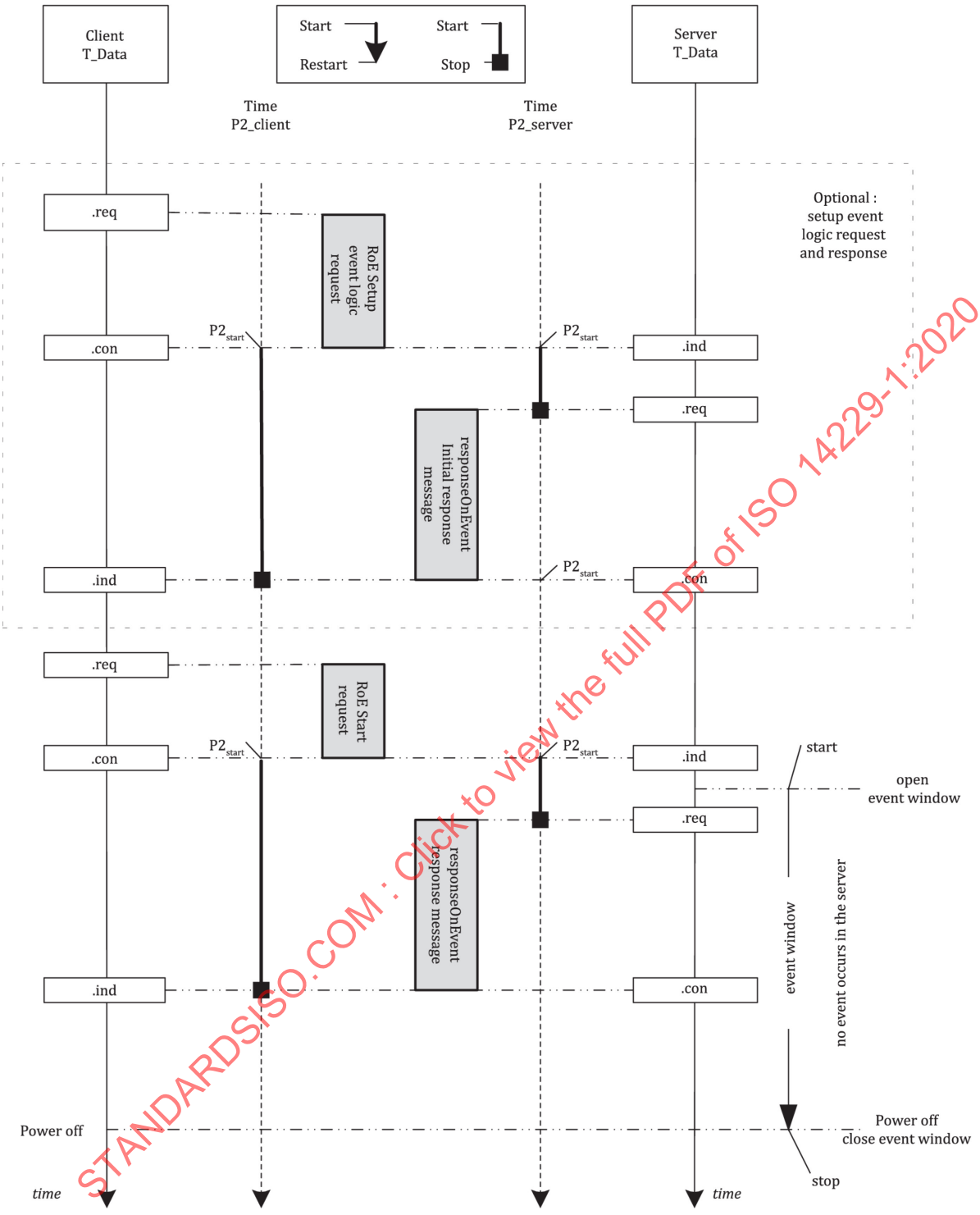
— No event occurs within the infinite event window.

— Multiple events (#1 to #n) within a infinite event window. Each positive response of the serviceToRespondTo is related to an identified event (#1 to #n) and shall have the same service identifier (SI) but might have different content.

Figure 18 depicts the infinite event window – Nn event during active event window.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved



STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

Figure 18 — Infinite event window – No event during active event window

Figure 19 depicts the infinite event window – multiple events during active event window.

© ISO 2020 – All rights reserved

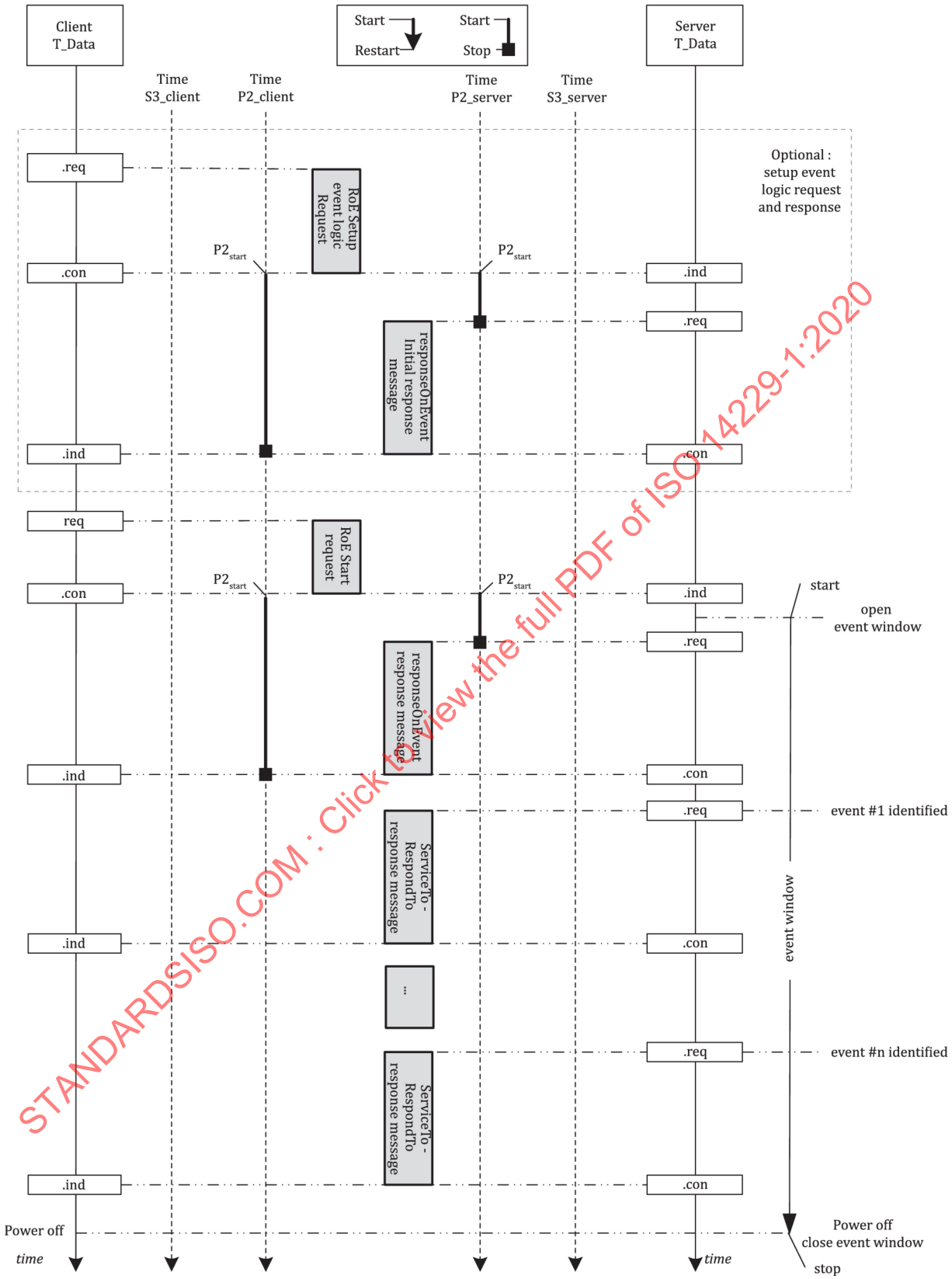


Figure 19 — Infinite event window – Multiple events during active event window

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

10.9.5.4 Example #3 - ResponseOnEvent (infinite event window) – SubFunction parameter "onComparisonOfValues"

This example only explains the utilisation of SubFunction parameter "onComparisonOfValues" assuming that the communication behaviour of the ROE service described in Example #1 and Example #2 has not changed. Therefore this example does not describe the complete message flow. Instead, only the event window setup request message and the positive response message to the occurring event is shown and explained. Start and Stop request messages as well as the different response messages are already described in the examples above. The following conditions apply:

— service 22_{16} – ReadDataByIdentifier is chosen as the serviceToRespondTo,

— the dataIdentifier 0104_{16} includes the measurement value which is to be compared at data byte#11 and #12 (this measurement value may also be read by utilising service 22_{16}),

— an event occurs if the measurement value (MV) is higher than the so called comparison parameter (CP) therefore the operator value (see description below) is chosen as 01_{16} – "MV > CP",

— as hysteresis value $0A_{16}$ – 10 % is chosen,

— as eventWindowTime the value 02_{16} – "infinite" is chosen,

— as doNotStoreEvent (eventType SubFunction bit 6) the value 0_2 is chosen,

— in any case a response is requested.

Definition for examples:

— Byte#4 to 5: dataIdentifier 0104_{16}

— Byte#6 to 7: Localisation of reading and definition of reading type.

EXAMPLE 1 If the reading is in the 11th byte of the data record, the following applies:

— 11x8 = 88 dec = 000101 1000_2 Bit#10 - Bit#14: length in bits - 1.

— With 5 bits, there is a maximum size of 32 bits = "long".

EXAMPLE 2 For a "word", the length is therefore 15 dec = 0 1111_2 Bit#15: Sign entry: 1=signed, 0=unsigned.

EXAMPLE 3 Total assignment would be:

— 1011 1100 0101 1000_2= $BC58_{16}$ thus byte#6 contains BC_{16}, byte#7 contains 58_{16}

— Byte#8: Comparison operation (operator)

EXAMPLE 4 operator MV > CP = 01_{16}

— Byte#9-12: Comparison parameters due to the 4 byte length, all data formats from 'Bit' through 'Long' type can be transmitted.

EXAMPLE 5 If comparison value is 5 242 = 0000 $147A_{16}$

— byte#9 = 00_{16}, byte#10 = 00_{16}, byte#11 = 14_{16} and byte#12 = $7A_{16}$

— Byte#13: Hysteresis value (specified as percentage of comparison parameter). The value is specified directly. It only applies to the operators "<" and ">". In case of zero as comparison value, the hysteresis value shall be defined as an absolute value.

EXAMPLE 6 Hysteresis value 10 % = $0A_{16}$.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

Table 162 specifies the ResponseOnEvent request message example #3.

Table 162 — ResponseOnEvent request message example #3

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ResponseOnEvent Request SID	86_{16}	ROE
#2	eventTypeRecord [eventType] = onComparisonOfValues, storageState = doNotStoreEvent suppressPosRspMsgIndicationBit=FALSE	07_{16}	ET_OCOV
#3	eventWindowTime = infinite	02_{16}	EWT
#4	eventTypeRecord [eventTypeParameter#1] = recordDataIdentifier (High Byte)	01_{16}	ETR_ETP1
#5	eventTypeRecord [eventTypeParameter#2] = recordDataIdentifier (Low Byte)	04_{16}	ETR_ETP2
#6	eventTypeRecord [eventTypeParameter#3] = Valueinfo#1	BC_{16}	ETR_ETP3
#7	eventTypeRecord [eventTypeParameter#4] = Valueinfo#2	58_{16}	ETR_ETP4
#8	eventTypeRecord [eventTypeParameter#5] = Operator	01_{16}	ETR_ETP5
#9	eventTypeRecord [eventTypeParameter#6] = Comparison Parameter (Byte#4)	00_{16}	ETR_ETP6
#10	eventTypeRecord [eventTypeParameter#7] = Comparison Parameter (Byte#3)	00_{16}	ETR_ETP7
#11	eventTypeRecord [eventTypeParameter#8] = Comparison Parameter (Byte#2)	14_{16}	ETR_ETP8
#12	eventTypeRecord [eventTypeParameter#9] = Comparison Parameter (Byte#1)	$7A_{16}$	ETR_ETP9
#13	eventTypeRecord [eventTypeParameter#10] = Hysteresis [%]	$0A_{16}$	ETR_ETP10
#14	serviceToRespondToRecord [serviceID] = ReadDataByIdentifier	22_{16}	RDBI
#15	serviceToRespondToRecord [serviceParameter#1] = dataIdentifier (MSB)	01_{16}	DID_B1
#16	serviceToRespondToRecord [serviceParameter#2] = dataIdentifier (LSB)	04_{16}	DID_B2

NOTE Response message and subsequent initialisation sequence is not shown.

The server reacts if the measurement value is higher than 5 242_d after a successful event window set up and activation of the ROE mechanism. The specified event occurs and the server sends the following message.

Table 163 specifies the ReadDataByIdentifier positive response message example #3.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 163 — ReadDataByIdentifier positive response message example #3

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadDataByIdentifier Response SID	62_{16}	RDBIPR
#2	dataIdentifier [byte#1] (MSB)	01_{16}	DID_B1
#3	dataIdentifier [byte#2] (LSB)	04_{16}	DID_B2
#4	dataRecord [data#1]	XX_{16}	DREC_DATA1
#5	dataRecord [data#2]	XX_{16}	DREC_DATA2
#6	dataRecord [data#3]	XX_{16}	DREC_DATA3
#7	dataRecord [data#4]	XX_{16}	DREC_DATA4
#8	dataRecord [data#5]	XX_{16}	DREC_DATA5
#9	dataRecord [data#6]	XX_{16}	DREC_DATA6
#10	dataRecord [data#7]	XX_{16}	DREC_DATA7
#11	dataRecord [data#8]	XX_{16}	DREC_DATA8
#12	dataRecord [data#9]	XX_{16}	DREC_DATA9
#13	dataRecord [data#10]	XX_{16}	DREC_DATA10
#14	dataRecord [data#11]data content of byte#11: 14_{16}	14_{16}	DREC_DATA11
#15	dataRecord [data#12]data content of byte#12: $7B_{16}$	$7B_{16}$	DREC_DATA12
:	:	:	:

A further event occurs not before the measurement value is at least once below 90 % of the comparison parameter value. This behaviour is specified by the hysteresis value. If this condition is fulfilled and the measurement value is again higher than the comparison value, a new event occurs and a new ReadDataByIdentifier response message is sent by the server.

10.9.5.5 Example #4 - ResponseOnEvent request message (reportMostRecentDtcOnStatusChange)

This example explains the request message for a ResponseOnEvent setup requesting the most recent test failed DTC. Table 164 specifies the ResponseOnEvent request message flow example #4.

Table 164 — ResponseOnEvent request message flow example #4

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte Value**	**Mnemonic**
#1	ResponseOnEvent Request SID	86_{16}	ROE
#2	eventTypeRecord [eventType] = reportMostRecentDtcOnStatusChange, storageState = doNotStoreEvent, suppressPosRspMsgIndicationBit = FALSE]	08_{16}	RMRDOSC

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

#3	eventWindowTime = infinite	02_{16}	EWT
#4	eventTypeParameter #1 [reportType] = reportMostRecentTestFailedDTC	$0D_{16}$	ETP1

After setting up the response on event in this example, the server starts to monitor test failed (Bit0) DTC status changes. If the DTC $12A104_{16}$ gets test failed, the server sends the response on event, by sending the result of the service 19_{16} $0D_{16}$ and the DTCAndStatusRecord for the test failed DTC within the positive response message. Table 165 specifies the ReadDTCInformation positive response message flow example #4.

Table 165 — ReadDTCInformation positive response message flow example #4

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte Value**	**Mnemonic**
#1	ReadDTCInformation Response SID	59_{16}	RDTCI
#2	reportMostRecentTestFailedDTC	$0D_{16}$	RMRTFDTC
#3	DTCStatusAvailabilityMask	FF_{16}	DTCSAM
	DTCAndStatusRecord[] = [		DTCASR_
#4	DTCHighByte	12_{16}	DTCHB
#5	DTCMiddleByte	$A1_{16}$	DTCMB
#6	DTCLowByte	04_{16}	DTCLB
#7	statusOfDTC = testFailed]	01_{16}	SODTC

10.9.5.6 Example #5 - ResponseOnEvent request message (reportDTCRecordInformationOnDtcStatusChange)

This example explains the request message for a ResponseOnEvent setup requesting an extended data record 12_{16} for all DTC that have the confirmedDTC (Bit 3) set. Table 166 specifies the ResponseOnEvent request message flow example #4.

Table 166 — ResponseOnEvent request message flow example #4

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte Value**	**Mnemonic**
#1	ResponseOnEvent Request SID	86_{16}	ROE
#2	eventTypeRecord [eventType] = reportDTCRecordInformationOnDtcStatusChange, storageState = doNotStoreEvent, suppressPosRspMsgIndicationBit = FALSE	09_{16}	RDRIODSC
#3	eventWindowTime = infinite	02_{16}	EWT
#4	eventTypeParameter #1 [DTCStatusMask] = confirmed DTCs (Bit3)	08_{16}	ETP1
#5	eventTypeParameter #2 [ReadDTCInformation.SubFunction] = reportDTCExtDataRecordByDTCNumber	06_{16}	ETP2
#6	eventTypeParameter #3 [DTCExtDataRecordNumber]	12_{16}	ETP3

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

After setting up the response on event in this example, the server starts to monitor Bit 3 DTC status changes. If the DTC $12A104_{16}$ is confirmed, the server sends the response on event, by sending the result of the service 19_{16} 06_{16} 12_{16} $A1_{16}$ 04_{16} 12_{16} and sends a positive response message. Table 167 specifies the ReadDTCInformation positive response message flow example #1.

Table 167 — ReadDTCInformation positive response message flow example #1

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte Value**	**Mnemonic**
#1	ReadDTCInformation Response SID	59_{16}	RDTCI
#2	reportDTCExtDataRecordByDTCNumber	06_{16}	RDTCEDRBD
	DTCAndStatusRecord[] = [		DTCASR_
#3	DTCHighByte	12_{16}	DTCHB
#4	DTCMiddleByte	$A1_{16}$	DTCMB
#5	DTCLowByte	04_{16}	DTCLB
#6	statusOfDTC = testFailed, Pending, Confirmed]	$0B_{16}$	SODTC
#7	DTCExtDataRecordNumber#1	12_{16}	DTCEDRN
	DTCExtDataRecord[]#1 = [		DTCSSR_
#8	extendedData#1 byte#1	03_{16}	EDD11
#9	extendedData#1 byte#2]	02_{16}	EDD12

10.10 LinkControl (87_{16}) service

10.10.1 Service description

The LinkControl service is used to control the communication between the client and the server(s) in order to gain bus bandwidth for diagnostic purposes (e.g. programming). This service optionally applies to those data link layers, which provides the capability to reconfigure its communication parameter (e.g. change the baudrate on CAN or reconfigure a FlexRay cycle design) during a non-default diagnostic session.

NOTE Further details on the application and usage of this service on a certain data link layer can be found in the individual data link layer specific diagnostic services implementation UDSonXYZ 'data link' specification.

This service is used to transition the data link layer into a certain state which allows utilizing higher diagnostic bandwidth most likely for programming purposes. To overcome functional communication constraints (e.g. the baudrate shall be transitioned in multiple servers at the same time) the transition itself is split into two steps:

— **Step #1**: The client verifies if the transition can be performed and informs the server(s) about the mode transition mechanism to be used. Each server shall respond positively (suppressPosRspMsgIndicationBit = FALSE) before the client performs step #2. This step actually does not perform the mode transition.

— **Step #2**: The client actually requests the mode transition (e.g. higher baudrate). This step shall only be requested if step #1 has been performed successfully. In case of functional communication it is recommended that there shall not be any response from a server when the mode transition is performed (suppressPosRspMsgIndicationBit = TRUE), because one server might already have been transitioned to the new mode while others are still in progress.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

The linkControlType parameter in the request message in conjunction with the conditional linkControlModeIdentifier/linkRecord parameter provides a mechanism to transition with either a predefined mode transition parameter or a specifically defined mode transition parameter.

This service is tied to a non-defaultSession. A session layer timer timeout will transition the server(s) back to its (their) normal mode of operation. The same applies in case an ECUReset service (11_{16}) is performed. Once a data link mode transition has taken place, any additional non-defaultSession request(s) shall not cause a retransition into the default mode of operation (e.g. during a programming session).

IMPORTANT — The server and the client shall meet the request and response message behaviour as specified in 8.7.

10.10.2 Request message

10.10.2.1Request message definition

Table 168 specifies the request message (linkControlType = verifyModeTransitionWithFixedParameter).

Table 168 — Request message definition (linkControlType = verifyModeTransitionWithFixedParameter)

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	LinkControl Request SID	M	87_{16}	LC
#2	SubFunction = [linkControlType]	M	01_{16}	LEV_LCTP_
#3	linkControlModeIdentifier	M	00_{16} to FF_{16}	LCMI_

Table 169 specifies the request message (linkControlType = verifyModeTransitionWithSpecificParameter).

Table 169 — Request message definition (linkControlType = verifyModeTransitionWithSpecificParameter)

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	LinkControl Request SID	M	87_{16}	LC
#2	SubFunction = [linkControlType]	M	02_{16}	LEV_LCTP_
	linkRecord[] = [			LBR_
#3	modeParameterHighByte	M	00_{16} to FF_{16}	MPHB
#4	modeParameterMiddleByte	M	00_{16} to FF_{16}	MPMB
#5	modeParameterLowByte]	M	00_{16} to FF_{16}	MPLB

Table 170 specifies the request message (linkControlType = transitionMode).

Table 170 — Request message definition (linkControlType = transitionMode)

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	LinkControl Request SID	M	87_{16}	LC
#2	SubFunction = [linkControlType]	M	03_{16}	LEV_LCTP_

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

10.10.2.2 Request message SubFunction parameter $Level (LEV_) definition

The SubFunction parameter linkControlType is used by the LinkControl request message to describe the action to be performed in the server (suppressPosRspMsgIndicationBit (bit 7) not shown in table below).

Table 171 specifies the request message SubFunction parameters.

Table 171 — Request message SubFunction parameter definition

Bits 6 to 0	Description	Cvt	Mnemonic
00_{16}	**ISOSAEReserved** This value is reserved by this document.	M	ISOSAERESRVD
01_{16}	**verifyModeTransitionWithFixedParameter** This parameter is used to verify if a transition with a predefined parameter, which is specified by the linkControlModeIdentifier data-parameter can be performed.	U	VMTWFP
02_{16}	**verifyModeTransitionWithSpecificParameter** This parameter is used to verify if a transition to a specifically defined parameter (e.g. specific baudrate), which is specified by the linkRecord data-parameter can be performed.	U	VMTWSP
03_{16}	**transitionMode** This SubFunction parameter requests the server(s) to transition the data link into the mode which was requested in the preceding verification message.	U	TM
04_{16} to $3F_{16}$	**ISOSAEReserved** This range of values is reserved by this document for future definition.	M	ISOSAERESRVD
40_{16} to $5F_{16}$	**vehicleManufacturerSpecific** This range of values is reserved for vehicle manufacturer specific use.	U	VMS
60_{16} to $7E_{16}$	**systemSupplierSpecific** This range of values is reserved for system supplier specific use.	U	SSS
$7F_{16}$	**ISOSAEReserved** This value is reserved by this document for future definition.	M	ISOSAERESRVD

10.10.2.3 Request message data-parameter definition

Table 172 specifies the data-parameters of the request message.

Table 172 — Request message data-parameter definition

Definition
linkControlModeIdentifier This conditional parameter references a fixed defined mode parameter to transition to (see B.3).
linkRecord This conditional parameter record contains a specific mode parameter in case the SubFunction parameter indicates that a specific parameter is used. The format of the linkRecord is specified in the individual data links specific diagnostic specification (UDSonXYZ).

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

10.10.3 Positive response message

10.10.3.1 Positive response message definition

Table 173 specifies the positive response message.

Table 173 — Positive response message definition

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	LinkControl Response SID	M	$C7_{16}$	LCPR
#2	linkControlType	M	00_{16} to $7F_{16}$	LCTP

10.10.3.2 Positive response message data-parameter definition

Table 174 specifies the data-parameter of the positive response message.

Table 174 — Response message data-parameter definition

Definition
linkControlType This parameter is an echo of bits 6 - 0 of the linkControlType SubFunction parameter from the request message.

10.10.4 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 175. The listed negative responses shall be used if the error scenario applies to the server.

Table 175 — 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 requested LinkControl are not met.	CNC
24_{16}	**requestSequenceError** This NRC shall be returned if the client requests the transition of the mode of operation without a preceding verification step, which specifies the mode to transition to.	RSE
31_{16}	**requestOutOfRange** This NRC shall be returned if: the requested linkControlModeIdentifier is invalid; the specific modeParameter (linkRecord) is invalid.	ROOR

STANDARDSISO.COM: Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

10.10.5 Message flow example(s) LinkControl

10.10.5.1 Example #1 - Transition baudrate to fixed baudrate (PC baudrate 115,2 kBit/s)

10.10.5.1.1 Step#1: Verify if all criteria are met for a baudrate switch

Table 176 specifies the LinkControl request message flow example #1 - step #1.

Table 176 — LinkControl request message flow example #1 - step #1

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	LinkControl Request SID	87_{16}	LC
#2	linkControlType = verifyModeTransitionWithFixedParameter, suppressPosRspMsgIndicationBit = FALSE	01_{16}	VMTWFP
#3	linkControlModeIdentifier = PC115200Baud	05_{16}	BI_PC115200

Table 177 specifies the LinkControl positive response message flow example #1 - step #1.

Table 177 — LinkControl positive response message flow example #1 - step #1

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	LinkControl Response SID	$C7_{16}$	LCPR
#2	linkControlType = verifyModeTransitionWithFixedParameter	01_{16}	VMTWFP

10.10.5.1.2 Step#2: Transition the baudrate

Table 178 specifies the LinkControl request message flow example #1 - step #2.

Table 178 — LinkControl request message flow example #1 - step #2

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	LinkControl Request SID	87_{16}	LC
#2	linkControlType = transitionMode, suppressPosRspMsgIndicationBit = TRUE	83_{16}	TM

There is no response from the server(s). The client and the server(s) shall transition the baudrate of their communication link.

10.10.5.2 Example #2 - Transition baudrate to specific baudrate (150 kBit/s)

10.10.5.2.1 Step#1: Verify if all criteria are met for a baudrate switch

Table 179 specifies the LinkControl request message flow example #2 - step #1.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

Table 179 — LinkControl request message flow example #2 - step #1

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	LinkControl Request SID	87_{16}	LC
#2	linkControlType = verifyModeTransitionWithSpecificParameter, suppressPosRspMsgIndicationBit = FALSE	02_{16}	VMTWSP
#3	linkRecord [modeParameterHighByte] (150kBit/s)	02_{16}	MPHB
#4	linkRecord [modeParameterMiddleByte]	49_{16}	MPMB
#5	linkRecord [modeParameterLowByte]	$F0_{16}$	MPLB

Table 180 specifies the LinkControl positive response message flow example #2 - step #1.

Table 180 — LinkControl positive response message flow example #2 - step #1

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	LinkControl Response SID	$C7_{16}$	LCPR
#2	linkControlType = verifyModeTransitionWithSpecificParameter	02_{16}	VMTWSP

10.10.5.2.2 Step#2: Transition the baudrate

Table 181 specifies the LinkControl request message flow example #2 - step #2.

Table 181 — LinkControl request message flow example #2 - step #2

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	LinkControl Request SID	87_{16}	LC
#2	linkControlType = transitionMode, suppressPosRspMsgIndicationBit = TRUE	83_{16}	TM

There is no response from the server(s). The client and the server(s) shall transition the baudrate of their communication link.

10.10.5.3 Example #3 - Transition FlexRay cycle design to 'programming'

The following example reflects a scenario, where a FlexRay network cycle design is transitioned into an optimized 'programming' mode (e.g. utilizing an enhanced dynamic segment for programming).

10.10.5.3.1 Step#1: Verify if all criteria are met for a scheduler switch

Table 182 specifies the LinkControl request message flow example #3 - step #1.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 182 — LinkControl request message flow example #3 - step #1

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	LinkControl Request SID	87_{16}	LC
#2	linkControlType = verifyModeTransitionWithFixedParameter, suppressPosRspMsgIndicationBit = FALSE	01_{16}	VMTWFP
#3	linkControlModeIdentifier = ProgrammingSetup	20_{16}	PROGSU

Table 183 specifies the LinkControl positive response message flow example #3 - step #1.

Table 183 — LinkControl positive response message flow example #3 - step #1

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	LinkControl Response SID	$C7_{16}$	LCPR
#2	linkControlType = verifyModeTransitionWithFixedParameter	01_{16}	VMTWFP

10.10.5.3.2 Step#2: Transition to programming scheduler

Table 184 specifies the LinkControl request message flow example #3 - step #2.

Table 184 — LinkControl request message flow example #3 - step #2

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	LinkControl Request SID	87_{16}	LC
#2	linkControlType = transitionMode, suppressPosRspMsgIndicationBit = TRUE	83_{16}	TM

There is no response from the server(s). The client and the server(s) shall transition the cycle design of the FlexRay communication link.

11 Data transmission functional unit

11.1 Overview

Table 185 specifies the Data Transmission functional unit.

Table 185 — Data Transmission functional unit

Service	Description
ReadDataByIdentifier	The client requests to read the current value of a record identified by a provided dataIdentifier.
ReadMemoryByAddress	The client requests to read the current value of the provided memory range.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

ReadScalingDataByIdentifier	The client requests to read the scaling information of a record identified by a provided dataIdentifier.
ReadDataByPeriodicIdentifier	The client requests to schedule data in the server for periodic transmission.
DynamicallyDefineDataIdentifier	The client requests to dynamically define data Identifiers that may subsequently be read by the readDataByIdentifier service.
WriteDataByIdentifier	The client requests to write a record specified by a provided dataIdentifier.
WriteMemoryByAddress	The client requests to overwrite a provided memory range.

11.2 ReadDataByIdentifier (22_{16}) service

11.2.1 Service description

The ReadDataByIdentifier service allows the client to request data record values from the server identified by one or more dataIdentifiers.

The client request message contains one or more two byte dataIdentifier values that identify data record(s) maintained by the server (see C.1 for allowed dataIdentifier values). The format and definition of the dataRecord shall be vehicle manufacturer or system supplier specific, and may include analog input and output signals, digital input and output signals, internal data, and system status information if supported by the server.

The server may limit the number of dataIdentifiers that can be simultaneously requested as agreed upon by the vehicle manufacturer and system supplier.

Upon receiving a ReadDataByIdentifier request, the server shall access the data elements of the records specified by the dataIdentifier parameter(s) and transmit their value in one single ReadDataByIdentifier positive response containing the associated dataRecord parameter(s). The request message may contain the same dataIdentifier multiple times. The server shall treat each dataIdentifier as a separate parameter and respond with data for each dataIdentifier as often as requested.

IMPORTANT — The server and the client shall meet the request and response message behaviour as specified in 8.7.

11.2.2 Request message

11.2.2.1 Request message definition

Table 186 specifies the request message.

Table 186 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	ReadDataByIdentifier Request SID	M	22_{16}	RDBI
	dataIdentifier[]#1 = [			DID_
#2	byte#1 (MSB)	M	00_{16} to FF_{16}	HB
#3	byte#2]	M	00_{16} to FF_{16}	LB
:	:	:	:	:
	dataIdentifier[]#m = [			DID_
#n-1	byte#1 (MSB)	U	00_{16} to FF_{16}	HB
#n	byte#2]	U	00_{16} to FF_{16}	LB

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

11.2.2.2 Request message SubFunction parameter $Level (LEV_) Definition

This service does not use a SubFunction parameter.

11.2.2.3 Request message data-parameter definition

Table 187 specifies the data-parameter for the request message.

Table 187 — Request message data-parameter definition

Definition
dataIdentifier (#1 to #m) This parameter identifies the server data record(s) that are being requested by the client (see C.1 for detailed parameter definition).

11.2.3 Positive response message

11.2.3.1 Positive response message definition

Table 188 specifies the positive response message.

Table 188 — Positive response message definition

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	ReadDataByIdentifier Response SID	M	62_{16}	RDBIPR
#2 #3	dataIdentifier[]#1 = [byte#1 (MSB) byte#2]	M M	00_{16} to FF_{16} 00_{16} to FF_{16}	DID_ HB LB
#4 : #(k-1)+4	dataRecord[]#1 = [data#1 : data#k]	M : U	00_{16} to FF_{16} : 00_{16} to FF_{16}	DREC_ DATA_1 : DATA_k
:	:	:	:	:
#n-(o-1)-2 #n-(o-1)-1	dataIdentifier[]#m = [byte#1 (MSB) byte#2]	U U	00_{16} to FF_{16} 00_{16} to FF_{16}	DID_ HB LB
#n-(o-1) : #n	dataRecord[]#m = [data#1 : data#o]	U : U	00_{16} to FF_{16} : 00_{16} to FF_{16}	DREC_ DATA_1 : DATA_o

11.2.3.2 Positive response message data-parameter definition

Table 189 specifies the data-parameters of the positive response message.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 189 — Response message data-parameter definition

Definition
dataIdentifier (#1 to #m) This parameter is an echo of the data-parameter dataIdentifier from the request message.
dataRecord (#1 to #k/o) This parameter is used by the ReadDataByIdentifier positive response message to provide the requested data record values to the client. The content of the dataRecord is not defined in this document and is vehicle manufacturer specific.

11.2.4 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 190. The listed negative responses shall be used if the error scenario applies to the server.

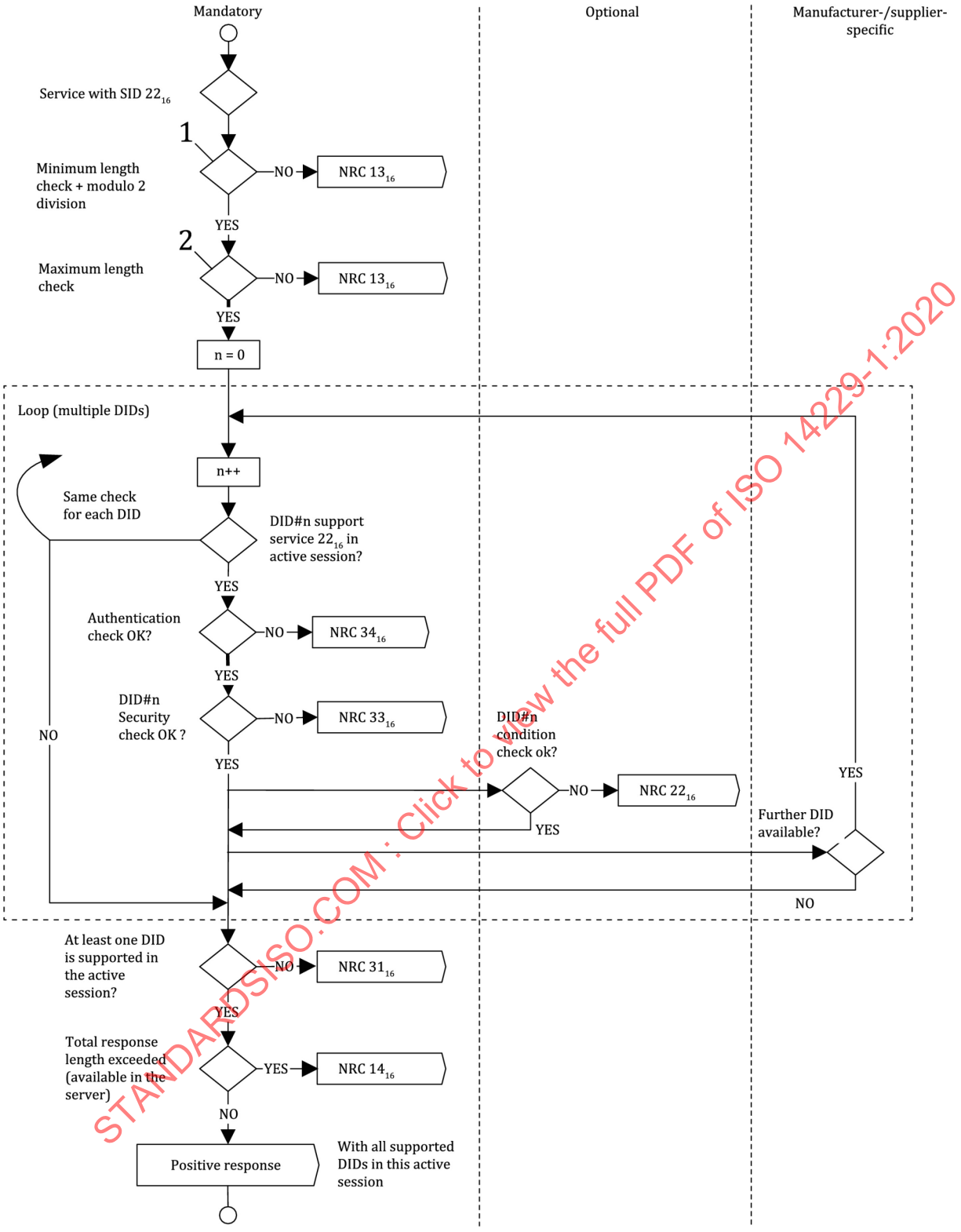
Table 190 — Supported negative response codes

NRC	Description	Mnemonic
13_{16}	**incorrectMessageLengthOrInvalidFormat** This NRC shall be sent if the length of the request message is invalid or the client exceeded the maximum number of dataIdentifiers allowed to be requested at a time.	IMLOIF
14_{16}	**responseTooLong** This NRC shall be sent if the total length of the response message exceeds the limit of the underlying transport protocol (e.g. when multiple DIDs are requested in a single request).	RTL
22_{16}	**conditionsNotCorrect** This NRC shall be sent if the operating conditions of the server are not met to perform the required action.	CNC
31_{16}	**requestOutOfRange** This NRC shall be sent if: none of the requested dataIdentifier values are supported by the device; none of the requested dataIdentifiers are supported in the current session; the requested dynamicDefinedDataIdentifier has not been assigned yet.	ROOR
33_{16}	**securityAccessDenied** This NRC shall be sent if at least one of the dataIdentifiers is secured and the server is not in an unlocked state.	SAD

The evaluation sequence is documented in Figure 20.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved



STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

Key

1 minimum length is 3 byte (SI + DID)

2 maximum length is 1 byte (SI) + 2*n bytes (DID(s))

Figure 20 — NRC handling for ReadDataByIdentifier service

© ISO 2020 – All rights reserved

11.2.5 Message flow example ReadDataByIdentifier

11.2.5.1 Assumptions

This subclause specifies the conditions to be fulfilled for the example to perform a ReadDataByIdentifier service. The client may request a dataIdentifier data at any time independent of the status of the server.

The dataIdentifier examples below are specific to a powertrain device (e.g. engine control module). Refer to ISO°15031-2[16] for further details regarding accepted terms/definitions/acronyms for emission related systems.

The first example reads a single two byte dataIdentifier containing a single piece of information (where dataIdentifier $F190_{16}$ contains the VIN number).

The second example demonstrates requesting of multiple dataIdentifiers with a single request (where dataIdentifier $010A_{16}$ contains engine coolant temperature, throttle position, engine speed, manifold absolute pressure, mass air flow, vehicle speed sensor, barometric pressure, calculated load value, idle air control, and accelerator pedal position, and dataIdentifier 0110_{16} contains battery positive voltage).

11.2.5.2 Example #1: read single dataIdentifier $F190_{16}$ (VIN number)

Table 191 specifies the ReadDataByIdentifier request message flow example #1.

Table 191 — ReadDataByIdentifier request message flow example #1

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadDataByIdentifier Request SID	22_{16}	RDBI
#2	dataIdentifier [byte#1] (MSB)	$F1_{16}$	DID_B1
#3	dataIdentifier [byte#2]	90_{16}	DID_B2

Table 192 specifies the ReadDataByIdentifier positive response message flow example #1.

Table 192 — ReadDataByIdentifier positive response message flow example #1

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadDataByIdentifier Response SID	62_{16}	RDBIPR
#2	dataIdentifier [byte#1] (MSB)	$F1_{16}$	DID_B1
#3	dataIdentifier [byte#2]	90_{16}	DID_B2
#4	dataRecord [data#1] = VIN Digit 1 = “W”	57_{16}	DREC_DATA1
#5	dataRecord [data#2] = VIN Digit 2 = “0”	30_{16}	DREC_DATA2
#6	dataRecord [data#3] = VIN Digit 3 = “L”	$4C_{16}$	DREC_DATA3
#7	dataRecord [data#4] = VIN Digit 4 = “0”	30_{16}	DREC_DATA4
#8	dataRecord [data#5] = VIN Digit 5 = “0”	30_{16}	DREC_DATA5
#9	dataRecord [data#6] = VIN Digit 6 = “0”	30_{16}	DREC_DATA6
#10	dataRecord [data#7] = VIN Digit 7 = “0”	30_{16}	DREC_DATA7

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Message direction	server → client		
Message type	**Response**		
#11	dataRecord [data#8] = VIN Digit 8 = “4”	34_{16}	DREC_DATA8
#12	dataRecord [data#9] = VIN Digit 9 = “3”	33_{16}	DREC_DATA9
#13	dataRecord [data#10] = VIN Digit 10 = “M”	$4D_{16}$	DREC_DATA10
#14	dataRecord [data#11] = VIN Digit 11 = “B”	42_{16}	DREC_DATA11
#15	dataRecord [data#12] = VIN Digit 12 = “5”	35_{16}	DREC_DATA12
#16	dataRecord [data#13] = VIN Digit 13 = “4”	34_{16}	DREC_DATA13
#17	dataRecord [data#14] = VIN Digit 14 = “1”	31_{16}	DREC_DATA14
#18	dataRecord [data#15] = VIN Digit 15 = “3”	33_{16}	DREC_DATA15
#19	dataRecord [data#16] = VIN Digit 16 = “2”	32_{16}	DREC_DATA16
#20	dataRecord [data#17] = VIN Digit 17 = “6”	36_{16}	DREC_DATA17

11.2.5.3 Example #2: Read multiple dataIdentifiers $010A_{16}$ and 0110_{16}

Table 193 specifies the ReadDataByIdentifier request message flow example #2.

Table 193 — ReadDataByIdentifier request message flow example #2

Message direction	client → server		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadDataByIdentifier Request SID	22_{16}	RDBI
#2	dataIdentifier#1 [byte#1] (MSB)	01_{16}	DID_B1
#3	dataIdentifier#1 [byte#2]	$0A_{16}$	DID_B2
#4	dataIdentifier#2 [byte#1] (MSB)	01_{16}	DID_B1
#5	dataIdentifier#2 [byte#2]	10_{16}	DID_B2

Table 194 specifies the ReadDataByIdentifier positive response message flow example #2.

Table 194 — ReadDataByIdentifier positive response message flow example #2

Message direction	server → client		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadDataByIdentifier Response SID	62_{16}	RDBIPR
#2	dataIdentifier [byte#1] (MSB)	01_{16}	DID_B1
#3	dataIdentifier [byte#2] (LSB)	$0A_{16}$	DID_B2
#4	dataRecord [data#1] = ECT	$A6_{16}$	DREC_DATA1
#5	dataRecord [data#2] = TP	66_{16}	DREC_DATA2
#6	dataRecord [data#3] = RPM	07_{16}	DREC_DATA3
#7	dataRecord [data#4] = RPM	50_{16}	DREC_DATA4
#8	dataRecord [data#5] = MAP	20_{16}	DREC_DATA5

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Message direction	server → client		
Message type	**Response**		
#9	dataRecord [data#6] = MAF	$1A_{16}$	DREC_DATA6
#10	dataRecord [data#7] = VSS	00_{16}	DREC_DATA7
#11	dataRecord [data#8] = BARO	63_{16}	DREC_DATA8
#12	dataRecord [data#9] = LOAD	$4A_{16}$	DREC_DATA9
#13	dataRecord [data#10] = IAC	82_{16}	DREC_DATA10
#14	dataRecord [data#11] = APP	$7E_{16}$	DREC_DATA11
#15	dataIdentifier [byte#1] (MSB)	01_{16}	DID_B1
#16	dataIdentifier [byte#2] (LSB)	10_{16}	DID_B2
#17	dataRecord [data#1] = B+	$8C_{16}$	DREC_DATA1

11.3 ReadMemoryByAddress (23_{16}) service

11.3.1 Service description

The ReadMemoryByAddress service allows the client to request memory data from the server via provided starting address and size of memory to be read.

The ReadMemoryByAddress request message is used to request memory data from the server identified by the parameter memoryAddress and memorySize. The number of bytes used for the memoryAddress and memorySize parameter is defined by addressAndLengthFormatIdentifier (low and high nibble).

It is also possible to use a fixed addressAndLengthFormatIdentifier and unused bytes within the memoryAddress or memorySize parameter are padded with the value 00_{16} in the higher range address locations.

In case of overlapping memory areas it is possible to use an additional memoryAddress byte as a memory identifier (e.g. use of internal and external flash).

The server sends data record values via the ReadMemoryByAddress positive response message. The format and definition of the dataRecord parameter shall be vehicle manufacturer specific. The dataRecord parameter may include analog input and output signals, digital input and output signals, internal data and system status information if supported by the server.

IMPORTANT — The server and the client shall meet the request and response message behaviour as specified in 8.7.

11.3.2 Request message

11.3.2.1 Request message definition

Table 195 specifies the request message.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 195 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	ReadMemoryByAddress Request SID	M	23_{16}	RMBA
#2	addressAndLengthFormatIdentifier	M	00_{16} to FF_{16}	ALFID
#3 : #(m-1)+3	memoryAddress[] = [byte#1 (MSB) : byte#m]	M : C1	00_{16} to FF_{16} : 00_{16} to FF_{16}	MA_ B1 : Bm
#n-(k-1) : #n	memorySize[] = [byte#1 (MSB) : byte#k]	M : C2	00_{16} to FF_{16} : 00_{16} to FF_{16}	MS_ B1 : Bk
C1: The presence of this parameter depends on address length information parameter of the addressAndLengthFormatIdentifier. C2: The presence of this parameter depends on the memory size length information of the addressAndLengthFormatIdentifier.				

11.3.2.2 Request message SubFunction parameter $Level (LEV_) definition

This service does not use a SubFunction parameter.

11.3.2.3 Request message data-parameter definition

Table 196 specifies the data-parameters of the request message.

Table 196 — Request message data-parameter definition

Definition
addressAndLengthFormatIdentifier This parameter is a one Byte value with each nibble encoded separately (see Table H.1 for example values): bit 7 - 4: Length (number of bytes) of the memorySize parameter bit 3 - 0: Length (number of bytes) of the memoryAddress parameter
memoryAddress The parameter memoryAddress is the starting address of server memory from which data is to be retrieved. The number of bytes used for this address is defined by the low nibble (bit 3 - 0) of the addressAndLengthFormatIdentifier. Byte#m in the memoryAddress parameter is always the least significant byte of the address being referenced in the server. The most significant byte(s) of the address can be used as a memory identifier. An example of the use of a memory identifier would be a dual processor server with 16 bit addressing and memory address overlap (when a given address is valid for either processor but yields a different physical memory device or internal and external flash is used). In this case, an otherwise unused byte within the memoryAddress parameter can be specified as a memory identifier used to select the desired memory device. Usage of this functionality shall be as defined by vehicle manufacturer/system supplier.
memorySize The parameter memorySize in the ReadMemoryByAddress request message specifies the number of bytes to be read starting at the address specified by memoryAddress in the server's memory. The number of bytes used for this size is defined by the high nibble (bit 7 - 4) of the addressAndLengthFormatIdentifier.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

11.3.3 Positive response message

11.3.3.1 Positive response message definition

Table 197 specifies the positive response message.

Table 197 — Positive response message definition

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	ReadMemoryByAddress Response SID	M	63_{16}	RMBAPR
#2 : #n	dataRecord[] = [data#1 : data#m]	 M : U	 00_{16} to FF_{16} : 00_{16} to FF_{16}	DREC_ DATA_1 : DATA_m

11.3.3.2 Positive response message data-parameter definition

Table 198 specifies the data parameter of the positive response message.

Table 198 — Response message data-parameter definition

Definition
dataRecord This parameter is used by the ReadMemoryByAddress positive response message to provide the requested data record values to the client. The content of the dataRecord is not defined in this document and shall reflect the requested memory contents. Data formatting shall be as defined by vehicle manufacturer/system supplier.

11.3.4 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 199. The listed negative responses shall be used if the error scenario applies to the server.

Table 199 — Supported negative response codes

NRC	Description	Mnemonic
13_{16}	**incorrectMessageLengthOrInvalidFormat** This NRC shall be sent if the length of the message is wrong.	IMLOIF
14_{16}	**responseTooLong** This NRC shall be sent if the total length of the response message exceeds the limit of the underlying transport protocol or response buffer size.	RTL
22_{16}	**conditionsNotCorrect** This NRC shall be sent if the operating conditions of the server are not met to perform the required action.	CNC

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

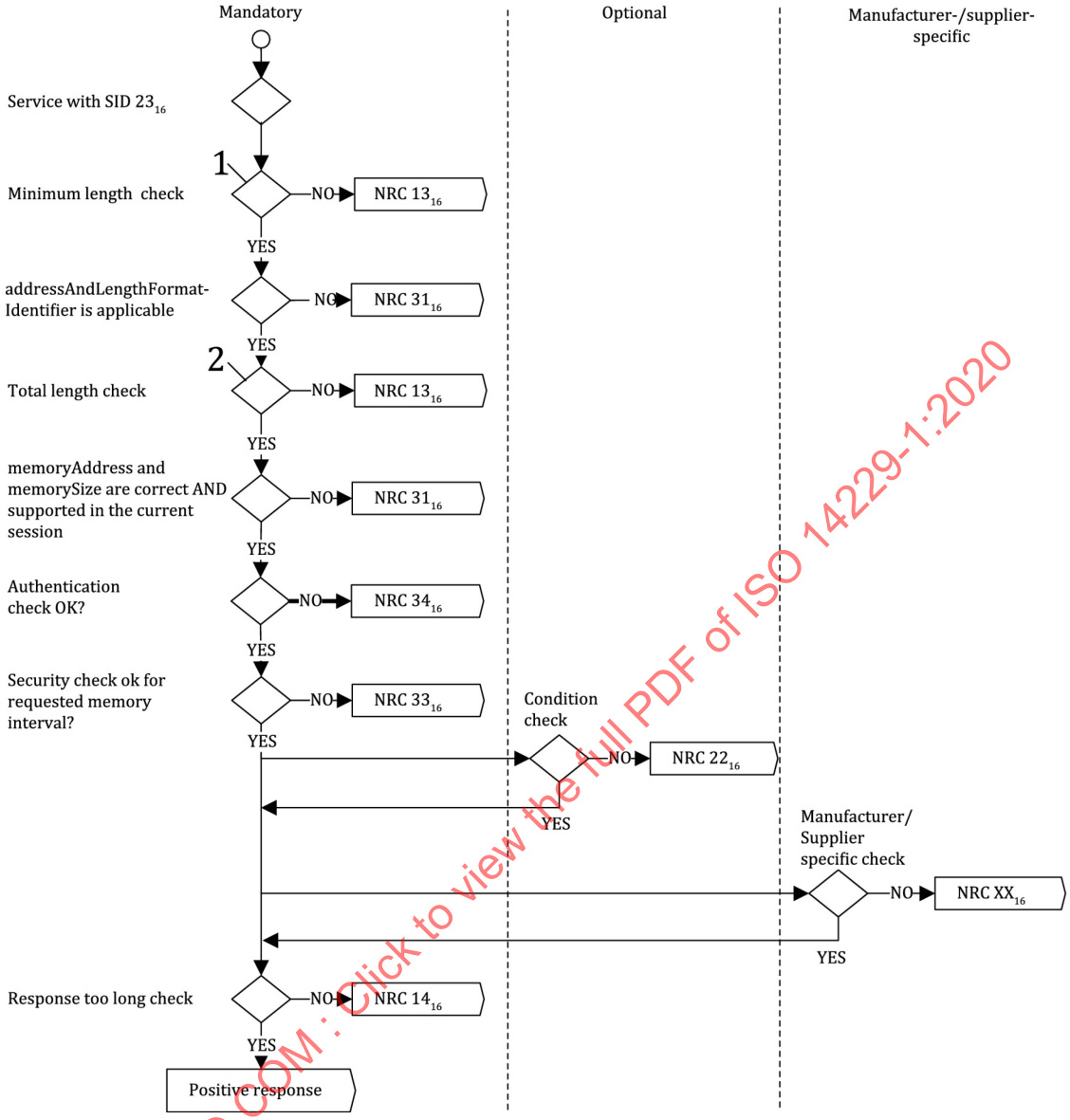
© ISO 2020 – All rights reserved

NRC	Description	Mnemonic
31_{16}	**requestOutOfRange** This NRC shall be sent if: — Any memory address within the interval [MA_{16}, (MA_{16} + MS_{16} -1_{16})] is invalid; — Any memory address within the interval [MA_{16}, (MA_{16} + MS_{16} -1_{16})] is restricted; — The memorySize parameter value in the request message is not supported by the server; — The specified addressAndLengthFormatIdentifier is not valid; — The memorySize parameter value in the request message is zero.	ROOR
33_{16}	**SecurityAccessDenied**	SAD
	This NRC shall be sent if any memory address within the interval [MA_{16}, (MA_{16} + MS_{16} -1_{16})] is secure and the server is locked.	

The evaluation sequence is documented in Figure 21.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved



Key

1 at least 4 (SI + addressAndLengthFormatIdentifier+min memoryAddress+min memorySize)

2 at 1 byte SI + 1 byte addressAndLengthFormatIdentifier + n byte memoryAddress parameter length + n byte memorySize parameter length)

Figure 21 — NRC handling for ReadMemoryByAddress service

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

11.3.5 Message flow example ReadMemoryByAddress

11.3.5.1 Assumptions

This subclause specifies the conditions to be fulfilled for the example to perform a ReadMemoryByAddress service. The service in this example is not limited by any restriction of the server.

11.3.5.2 Example #1: ReadMemoryByAddress - 4-byte (32-bit) addressing

The client reads 259 data bytes from the server's memory starting at memory address 2048 $_{16}$1392.

Table 200 specifies the ReadMemoryByAddress request message flow example #1.

© ISO 2020 – All rights reserved

Table 200 — ReadMemoryByAddress request message flow example #1

Message direction	client → server		
Message type	Request		
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	ReadMemoryByAddress Request SID	23_{16}	RMBA
#2	addressAndLengthFormatIdentifier	24_{16}	ALFID
#3	memoryAddress [byte#1] (MSB)	20_{16}	MA_B1
#4	memoryAddress [byte#2]	48_{16}	MA_B2
#5	memoryAddress [byte#3]	13_{16}	MA_B3
#6	memoryAddress [byte#4]	92_{16}	MA_B4
#7	memorySize [byte#1] (MSB)	01_{16}	MS_B1
#8	memorySize [byte#2]	03_{16}	MS_B2

Table 201 specifies the ReadMemoryByAddress positive response message flow example #1.

Table 201 — ReadMemoryByAddress positive response message flow example #1

Message direction	server → client		
Message type	Response		
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	ReadMemoryByAddress Response SID	63_{16}	RMBAPR
#2	dataRecord [data#1] (memory cell#1)	00_{16}	DREC_DATA_1
:	:	:	:
#259+1	dataRecord [data#259] (memory cell#259)	$8C_{16}$	DREC_DATA_259

11.3.5.3 Example #2: ReadMemoryByAddress - 2-byte (16-bit) addressing.

The client reads five data bytes from the server's memory starting at memory address 4813_{16}.

Table 202 specifies the ReadMemoryByAddress request message flow example #2.

Table 202 — ReadMemoryByAddress request message flow example #2

Message direction	client → server		
Message type	Request		
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	ReadMemoryByAddress Request SID	23_{16}	RMBA
#2	addressAndLengthFormatIdentifier	12_{16}	ALFID
#3	memoryAddress [byte#1 (MSB)]	48_{16}	MA_B1
#4	memoryAddress [byte#2 (LSB)]	13_{16}	MA_B2
#5	memorySize [byte#1]	05_{16}	MS_B1

Table 203 specifies the ReadMemoryByAddress positive response message flow example #2.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

Table 203 — ReadMemoryByAddress positive response message flow example #2

Message direction		server → client	
Message type		Response	
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	ReadMemoryByAddress Response SID	63_{16}	RMBAPR
#2	dataRecord [data#1] (memory cell#1)	43_{16}	DREC_DATA_1
#3	dataRecord [data#2] (memory cell#2)	$2A_{16}$	DREC_DATA_2
#4	dataRecord [data#3] (memory cell#3)	07_{16}	DREC_DATA_3
#5	dataRecord [data#4] (memory cell#4)	$2A_{16}$	DREC_DATA_4
#6	dataRecord [data#5] (memory cell#5)	55_{16}	DREC_DATA_5

11.3.5.4 Example #3: ReadMemoryByAddress, 3-byte (24-bit) addressing

The client reads three data bytes from the server's external RAM cells starting at memory address 204813_{16}.

Table 204 specifies the ReadMemoryByAddress request message flow example #3.

Table 204 — ReadMemoryByAddress request message flow example #3

Message direction		client → server	
Message type		Request	
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	ReadMemoryByAddress Request SID	23_{16}	RMBA
#2	addressAndLengthFormatIdentifier	23_{16}	ALFID
#3	memoryAddress [byte#1 (MSB)]	20_{16}	MA_B1
#4	memoryAddress [byte#2]	48_{16}	MA_B2
#5	memoryAddress [byte#3 (LSB)]	13_{16}	MA_B3
#6	memorySize [byte#1 (MSB)]	00_{16}	MS_B1
#7	memorySize [byte#2 (LSB)]	03_{16}	MS_B2

Table 205 specifies the ReadMemoryByAddress first positive response message, example #3.

Table 205 — ReadMemoryByAddress first positive response message, example #3

Message direction		server → client	
Message type		Response	
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	ReadMemoryByAddress Response SID	63_{16}	RMBAPR
#2	dataRecord [data#1] (memory cell#1)	00_{16}	DREC_DATA_1
#3	dataRecord [data#2] (memory cell#2)	01_{16}	DREC_DATA_2
#4	dataRecord [data#3] (memory cell#3)	$8C_{16}$	DREC_DATA_3

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

11.4 ReadScalingDataByIdentifier (24_{16}) service

11.4.1 Service description

The ReadScalingDataByIdentifier service allows the client to request scaling data record information from the server identified by a dataIdentifier.

The client request message contains one dataIdentifier value that identifies data record(s) maintained by the server (see C.1 for allowed dataIdentifier values). The format and definition of the dataRecord shall be vehicle manufacturer specific, and may include analog input and output signals, digital input and output signals, internal data, and system status information if supported by the server.

Upon receiving a ReadScalingDataByIdentifier request, the server shall access the scaling information associated with the specified dataIdentifier parameter and transmit the scaling information values in one ReadScalingDataByIdentifier positive response.

IMPORTANT — The server and the client shall meet the request and response message behaviour as specified in 8.7.

11.4.2 Request message

11.4.2.1 Request message definition

Table 206 specifies the request message.

Table 206 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	ReadScalingDataByIdentifier Request SID	M	24_{16}	RSDBI
	dataIdentifier[] = [			DID_
#2	byte#1 (MSB)	M	00_{16} to FF_{16}	HB
#3	byte#2]	M	00_{16} to FF_{16}	LB

11.4.2.2 Request message SubFunction parameter $Level (LEV_) definition

This service does not use a SubFunction parameter.

11.4.2.3 Request message data-parameter definition

Table 207 specifies the data-parameter of the request message.

Table 207 — Request message data-parameter definition

Definition
DataIdentifier This parameter identifies the server data record that is being requested by the client (see C.1 for detailed parameter definition).

11.4.3 Positive response message

11.4.3.1 Positive response message definition

Table 208 specifies the positive response message.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 208 — Positive response message definition

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	ReadScalingDataByIdentifier Response SID	M	64_{16}	RSDBIPR
#2 #3	dataIdentifier[] = [byte#1 (MSB) byte#2 (LSB)]	M M	00_{16} to FF_{16} 00_{16} to FF_{16}	DID_ HB LB
#4	scalingByte#1	M	00_{16} to FF_{16}	SB_1
#5 : #(p-1)+5	scalingByteExtension[]#1 = [scalingByteExtensionParameter#1 : scalingByteExtensionParameter#p]	C1 : C1	00_{16} to FF_{16} : 00_{16} to FF_{16}	SBE_ PAR1 / PARp
:	:	:	:	:
#n-r	scalingByte#k	C2	00_{16} to FF_{16}	SB_k
#n-(r-1) : #n	scalingByteExtension[]#k = [scalingByteExtensionParameter#1 : scalingByteExtensionParameter#r]	C1 : C1	00_{16} to FF_{16} : 00_{16} to FF_{16}	SBE_ PAR1 : PARr
C1: The presence of this parameter depends on the scalingByte high nibble. It is mandatory to be present if the scalingByte high nibble is encoded as formula, unit/format, or bitMappedReportedWithOutMask. C2: The presence of this parameter depends on whether the encoding of the scaling information requires more than one byte.				

11.4.3.2 Positive response message data-parameter definition

Table 209 specifies the data-parameters of the positive response message.

Table 209 — Response message data-parameter definition

Definition
dataIdentifier This parameter is an echo of the data-parameter dataIdentifier from the request message.
scalingByte (#1 to #k) This parameter is used by the ReadScalingDataByIdentifier positive response message to provide the requested scaling data record values to the client (see C.2 for detailed parameter definition).
scalingByteExtension (#1 to #p/#1 to #r)
This parameter is used to provide additional information for scalingBytes with a high nibble encoded as formula, unit/format, or bitmappedReportedWithOutMask (see C.3 for detailed parameter definition).

11.4.4 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 210. The listed negative responses shall be used if the error scenario applies to the server.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 210 — Supported negative response codes

NRC	Description	Mnemonic
13_{16}	**incorrectMessageLengthOrInvalidFormat** This NRC shall be sent if the length of the request message is invalid.	IMLOIF
22_{16}	**conditionsNotCorrect** This NRC shall be sent if the operating conditions of the server are not met to perform the required action.	CNC
31_{16}	**requestOutOfRange** This NRC shall be returned if: the requested dataIdentifier value is not supported by the device, the requested dataIdentifier value is supported by the device, but no scaling information is available for the specified dataIdentifier.	ROOR
33_{16}	**securityAccessDenied** This NRC shall be sent if the dataIdentifier is secured and the server is not in an unlocked state.	SAD
34_{16}	**authenticationRequired** This NRC shall be sent if the dataIdentifier is secured and the client has insufficient rights based on its Authentication state.	AR

The evaluation sequence is documented in Figure 22.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

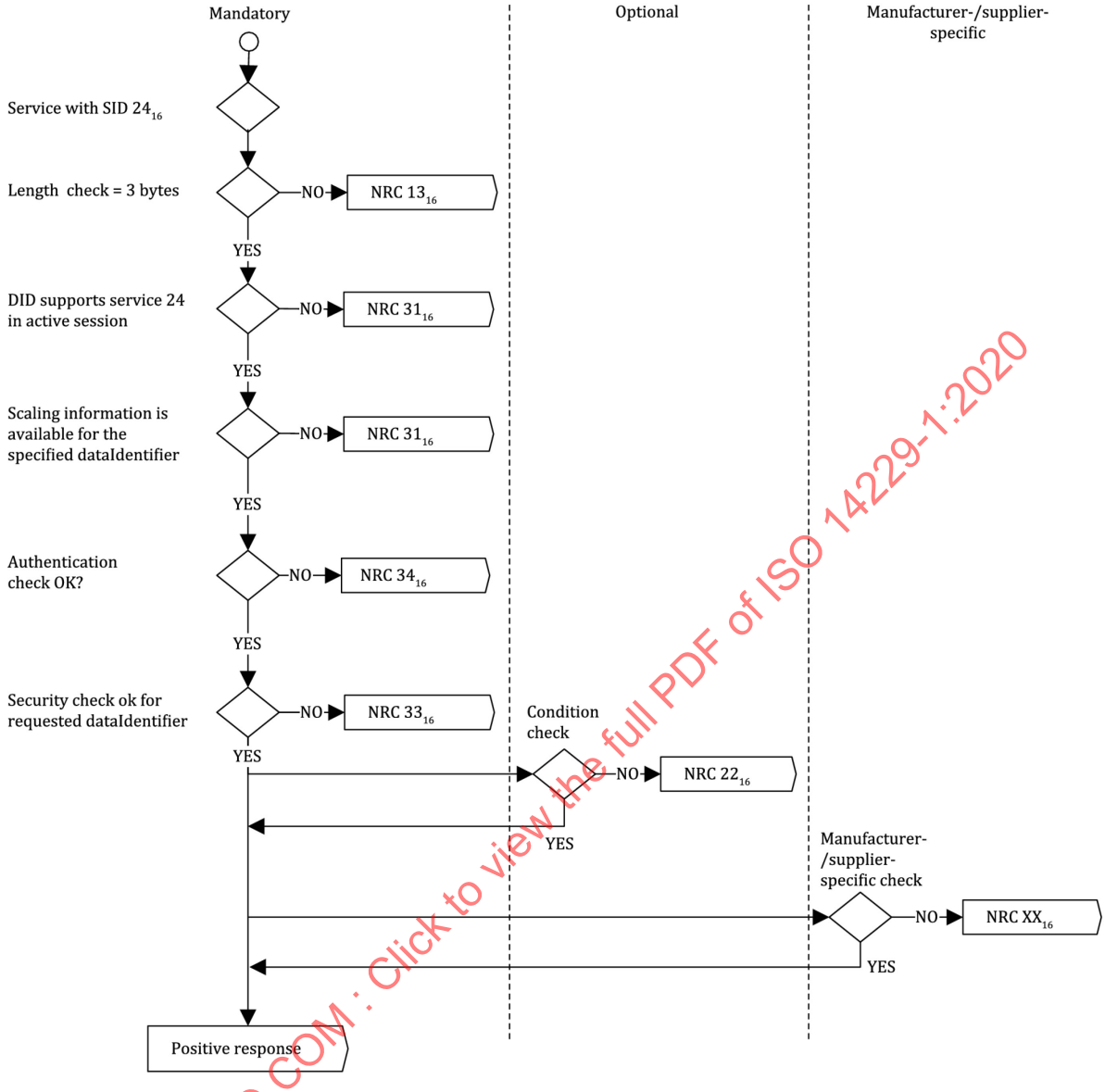


Figure 22 — NRC handling for ReadScalingDataByIdentifier service

11.4.5 Message flow example ReadScalingDataByIdentifier

11.4.5.1 Assumptions

This subclause specifies the conditions to be fulfilled for the example to perform a ReadScalingDataByIdentifier service. The client may request dataIdentifier scaling data at any time independent of the status of the server.

The first example reads the scaling information associated with the two byte dataIdentifier $F190_{16}$, which contains a single piece of information (17 character VIN number).

The second example demonstrates the use of a formula and unit identifier for specifiying a data variable in a server.

The third example illustrates the use of readScalingDataByIdentifier to return the supported bits (validity mask) for a bit mapped dataIdentifier that is reported without the mask through the use of readDataByIdenditfier.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

11.4.5.2 Example #1: readScalingDataByIdentifier wth dataIdentifier $F190_{16}$ (VIN number)

Table 211 specifies the ReadScalingDataByIdentifier request message flow example #1.

Table 211 — ReadScalingDataByIdentifier request message flow example #1

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadScalingDataByIdentifier Request SID	24_{16}	RSDBI
#2	dataIdentifier [byte#1] (MSB)	$F1_{16}$	DID_B1
#3	dataIdentifier [byte#2] (LSB)	90_{16}	DID_B2

Table 212 specifies the ReadScalingDataByIdentifier positive response message flow example #1.

Table 212 — ReadScalingDataByIdentifier positive response message flow example #1

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadScalingDataByIdentifier Response SID	64_{16}	RSDBIPR
#2	dataIdentifier [byte#1] (MSB)	$F1_{16}$	DID_B1
#3	dataIdentifier [byte#2] (LSB)	90_{16}	DID_B2
#4	scalingByte#1 {ASCII, 15 data bytes}	$6F_{16}$	SB_1
#5	scalingByte#2 {ASCII, 2 data bytes}	62_{16}	SB_2

11.4.5.3 Example #2: readScalingDataByIdentifier wth dataIdentifier 0105_{16} (Vehicle Speed)

Table 213 specifies the ReadScalingDataByIdentifier request message flow example #2.

Table 213 — ReadScalingDataByIdentifier request message flow example #2

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadScalingDataByIdentifier Request SID	24_{16}	RSDBI
#2	dataIdentifier [byte#1] (MSB)	01_{16}	DID_B1
#3	dataIdentifier [byte#2] (LSB)	05_{16}	DID_B2

Table 214 specifies the ReadScalingDataByIdentifier positive response message flow example #2.

STANDARDSISO.COM: Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

Table 214 — ReadScalingDataByIdentifier positive response message flow example #2

Message direction	server → client		
Message type	Response		
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	ReadScalingDataByIdentifier Response SID	64_{16}	RSDBIPR
#2	dataIdentifier [byte#1] (MSB)	01_{16}	DID_B1
#3	dataIdentifier [byte#2] (LSB)	05_{16}	DID_B2
#4	scalingByte#1 {unsigned numeric, 1 data byte}	01_{16}	SBYT_1
#5	scalingByte#2 {formula, 5 data bytes}	95_{16}	SB_2
#6	scalingByteExtension#2 [byte#1] {formulaIdentifier = C0 * x + C1}	00_{16}	SBE_21
#7	scalingByteExtension#2 [byte#2] {C0 high byte}	$E0_{16}$	SBE_22
#8	scalingByteExtension#2 [byte#3] {C0 low byte} [C0 = 75 * 10P-2P]	$4B_{16}$	SBE_23
#9	scalingByteExtension#2 [byte#4] {C1 high byte}	00_{16}	SBE_24
#10	scalingByteExtension#2 [byte#5] {C1 low byte} [C1 = 30 * 10P0P]	$1E_{16}$	SBE_25
#11	scalingByte#3 {unit/format, 1 data byte}	$A1_{16}$	SB_3
#12	scalingByteExtension#3 [byte#1] {unit ID, km/h}	30_{16}	SBE_31

Using the information contained in C.2 for decoding the scalingBytes, constants (C0, C1) and units, the data variable of vehicle speed is calculated using the following formula:

Vehicle Speed = (0,75 × x + 30) km/h

where 'x' is the actual data stored in the server and is identified by dataIdentifier 0105_{16}.

11.4.5.4 Example #3: readScalingDataByIdentifier wth dataIdentifier 0967_{16}

This example shows how a client could determine which bits are supported for a dataIdentifier in a server that is formatted as a bit mapped record reported without a validity mask.

The example dataIdentifier (0967_{16}) is defined in Table 215.

Table 215 — Example data definition

Data byte	Bit(s)	Description
#1	7 to 4	Unusued
	3	Medium speed fan is commanded on
	2	Medium speed fan output fault detected
	1	Purge monitor soak time status flag
	0	Purge monitor idle test is prevented due to refuel event
#2	7	Check fuel cap light is commanded on
	6	Check fuel cap light output fault detected
	5	Fan control A output fault detected
	4	Fan control B output fault detected

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

	3	High speed fan output fault detected
	2	High speed fan output is commanded on
	1	Purge monitor idle test (small leak) ready to run
	0	Purge monitor small leak has been monitored

Table 216 specifies the ReadScalingDataByIdentifier request message flow example #3.

Table 216 — ReadScalingDataByIdentifier request message flow example #3

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadScalingDataByIdentifier Request SID	24_{16}	RSDBI
#2	dataIdentifier [byte#1] (MSB)	09_{16}	DID_B1
#3	dataIdentifier [byte#2] (LSB)	67_{16}	DID_B2

Table 217 specifies the ReadScalingDataByIdentifier positive response message flow example #3.

Table 217 — ReadScalingDataByIdentifier positive response message flow example #3

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadScalingDataByIdentifier Response SID	64_{16}	RSDBIPR
#2	dataIdentifier [byte#1] (MSB)	09_{16}	DID_HB
#3	dataIdentifier [byte#2] (LSB)	67_{16}	DID_LB
#4	scalingByte#1 {bitMappedReportedWithOutMask, 2 data bytes}	22_{16}	SBYT_1
#5	scalingByteExtension#1 [byte#1] {dataRecord#1 Validity Mask}	03_{16}	SBYE_11
#6	scalingByteExtension#1 [byte#2] {dataRecord#2 Validity Mask}	43_{16}	SBYE_12

The above example makes the assumption that the only bits supported (i.e. that contain information) for this dataIdentifier in the server are byte#1, bits 1 and 0, and byte#2, bits 6, 1, and 0.

11.5 ReadDataByPeriodicIdentifier ($2A_{16}$) service

11.5.1 Service description

The ReadDataByPeriodicIdentifier service allows the client to request the periodic transmission of data record values from the server identified by one or more periodicDataIdentifiers.

The client request message contains one or more 1-byte periodicDataIdentifier values that identify data record(s) maintained by the server. The periodicDataIdentifier represents the low byte of a dataIdentifier out of the dataIdentifier range reserved for this service ($F2XX_{16}$, see C.1 for allowed periodicDataIdentifier values), e.g. the periodicDataIdentifier $E3_{16}$ used in this service is the dataIdentifier $F2E3_{16}$.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

The format and definition of the dataRecord shall be vehicle manufacturer specific, and may include analog input and output signals, digital input and output signals, internal data, and system status information if supported by the server.

Upon receiving a ReadDataByPeriodicIdentifier request other than stopSending the server shall check whether the conditions are correct to execute the service.

A periodicDataIdentifier shall only be supported with a single transmissionMode at a given time. A change to the schedule of a periodicDataIdentifier shall be performed on reception of a request message with the transmissionMode parameter set to a new schedule for the same periodicDataIdentifier. Multiple schedules for different periodicDataIdentifiers shall be supported upon vehicle manufacturer's request.

IMPORTANT — If the conditions are correct then the server shall transmit a positive response message, including only the service identifier. The server shall never transmit a negative response message once it has accepted the initial request message by responding positively.

Following the initial positive response message, the server shall access the data elements of the records specified by the periodicDataIdentifier parameter(s) and transmit their value in separate periodic data response messages for each periodicDataIdentifier containing the associated dataRecord parameters.

The separate periodic data response messages defined to transmit the periodicDataIdentifier data to the client following the initial positive response message shall include the periodicDataIdentifier and the data of the periodicDataIdentifier, but not the positive response service identifier. The mapping of the periodic response message onto certain data link layers is described in the appropriate implementation specifications of ISO 14229.

The documented periodic rate for a specific transmissionMode is defined as the time between any two consecutive response messages with the same periodicDataIdentifier, when only a single periodicDataIdentifier is scheduled. If multiple periodicDataIdentifiers are scheduled concurrently, the effective period between the same periodicDataIdentifier will vary based upon the following design parameters:

— SchedulerRate: the call rate of the periodic scheduler.

— NumPeriodicAddr: the number of available protocol specific periodic data response message address information IDs allocated per scheduler call (e.g. CAN identifier on CAN).

— MaxNumPDID: the number of periodicDataIdentifiers that can be defined in parallel to be transmitted concurrently.

These parameter values will impact the extent of how much the effective period between the same periodicDataIdentifier will increase if multiple periodicDataIdentifers are transmitted concurrently. Therefore, all of the previously mentioned design parameters shall be specified by the vehicle manufacturer. Each time the periodic scheduler is called it shall determine if any periodicDataIdentifiers are ready to transmit.

It is useful to define the following parameters:

— NumPDID: the current number of periodicDataIdentifiers scheduled to be transmitted concurrently.

— PeriodicRateCount: the number of SchedulerRate intervals within the periodic rate.

NOTE The periodic rate is an integer multiple of the periodic scheduler call rate.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Two scheduling implementation types are defined that specify how periodicDataIdentifiers are transmitted for a transmissionMode based upon the above parameters. It is vehicle manufacturer specific which implementation is used.

Both scheduler types use a PeriodicRateCounter, which is initially loaded with the PeriodicRateCount and decrements every time the scheduler is called. When the PeriodicRateCounter reaches zero the periodic rate has expired and is reloaded.

— **Scheduler type #1:** When the PeriodicRateCounter reaches zero then transmission of all scheduled PDIDs begins, starting with the first in the list of scheduled PDIDs. NumPeriodicAddr PDIDs will be transmitted each scheduler call, using the addresses that are available. This continues until all NumPDID scheduled PDIDs have been transmitted, and begins again the next time PeriodicRateCounter reaches zero.

— **Scheduler type #2:** When the PeriodicRateCounter reaches zero then the next scheduled NumPeriodicAddr PDIDs are transmitted. The PDIDs are only transmitted at the scheduler call when PeriodicRateCounter reaches zero, and not at other scheduler calls.

Scheduler type #1 can result in "aliasing" when there are too many PDIDs to transmit within the periodic rate. This occurs when NumPDID > PeriodicRateCount × NumPeriodicAddr. If aliasing occurs, then the PDIDs are continually transmitted at the scheduler rate in their scheduled order. The ECU shall not drop any PDIDs.

The following is a pseudo-code specification of a simple scheduler for a single transmission rate that supports both scheduler types. This is just for guidance, actual implementations may vary:

```
// Initialization …
PeriodicRateCount = PeriodicRate/SchedulerRate;
PeriodicRateCounter = PeriodicRateCount;
PDIDnext = 0;    // Index into list if active PDIDs to be transmitted.

// Determine, if scheduler type #1 has an alias situation
Alias = 0;
IF (SchedulerType == 1)
      IF ( Ceiling(NumPDID/NumPeriodicAddr) > PeriodicRateCount )
         Alias = 1;
      ENDIF;
ENDIF;

START periodic scheduler;


// Periodic scheduler called at the ScheduleRate …
      SchedulerCall()
PeriodicRateCounter = PeriodicRateCounter - 1;

// If scheduler type 1 and not all PDIDs in list have been transmitted, OR
// PeriodicRateCounter indicates a new Periodic Rate is to begin, OR
// scheduler type 1 aliasing …
IF ( ( (SchedulerType == 1) && (PDIDnext != 0) ) ||
       (PeriodicRateCounter == 0)                  ||
       (Alias)
      )
      // Transmit the next set of NumPeriodicAddr PDIDs from the list of active PDIDs
      i = 0;
      Periodic_Address = Initial Periodic_Address;

      // Loop and send the next set of PDIDs based on the Number of addresses available.
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

```
      WHILE(i < NumPeriodicAddr)
         // Fill in data and Periodic_Address of next PDID to transmit
         PDID_MSG.data[] = PDID[PDIDnext] data;
         PDID_MSG.UUDTaddr = Periodic_Address;

         SEND PDID_MSG;

            PDIDnext = (PDIDnext + 1) % NumPDID;

         // Don't send duplicate PDID(s) for scheduler type 1, unless an alias situation
         IF (SchedulerType == 1)
            IF ( (PDIDnext == 0) && (!Alias) )
               EXITWHILE;     // Don't send duplicate PDID(s) for scheduler type 1
            ENDIF;
         ENDIF;

         // Use next periodic address
         Periodic_Address = Periodic_Address + 1;
         i = i + 1;
      ENDWHILE;
ENDIF;

// Has the Periodic time expired?
IF (PeriodicRateCounter == 0)
      // Yes, reload the periodic rate count for the next transmission rate period
      PeriodicRateCounter = PeriodicRateCount;
ENDIF;
```

Example, two distinct ECU implementations may both support a fast transmissionMode with a periodic rate of 10 ms and a single unique periodic data response message address information ID. If the first implementation calls the periodic scheduler every 10 ms, the time between the same periodicDataIdentifier would increase to 20 ms when two periodicDataIdentifiers are scheduled and would increase to 40 ms when four periodicDataIdentifiers are scheduled. If the second implementation calls the periodic scheduler every 5 ms, the time between the same periodicDataIdentifier would remain at 10 ms when two periodicDataIdentifiers are scheduled and would increase to 20 ms when four periodicDataIdentifiers are scheduled. See more examples in 11.6.5.

Upon receiving a ReadDataByPeriodicIdentifier request including the transmissionMode stopSending the server shall either stop the periodic transmission of the periodicDataIdentifier(s) contained in the request message or stop the transmission of all periodicDataIdentifier if no specific one is specified in the request message. The response message to this transmissionMode only contains the service identifier.

The server may limit the number of periodicDataIdentifiers that can be simultaneously supported as agreed upon by the vehicle manufacturer and system supplier. Exceeding the maximum number of periodicDataIdentifier that can be simultaneously supported shall result in a single negative response and none of the periodicDataIdentifiers in that request shall be scheduled. Repetition of the same periodicDataIdentifier in a single request message is not allowed and the server shall ignore them all except one periodicDataIdentifer if the client breaks this rule.

IMPORTANT — The server and the client shall meet the request and response message behaviour as specified in 8.7.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

11.5.2 Request message

11.5.2.1 Request message definition

Table 218 specifies the request message.

Table 218 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	ReadDataByPeriodicIdentifier Request SID	M	$2A_{16}$	RDBPI
#2	transmissionMode	M	00_{16} to FF_{16}	TM
#3	periodicDataIdentifier[]#1	C	00_{16} to FF_{16}	PDID1
:	:	:	:	:
#m+2	periodicDataIdentifier[]#m	U	00_{16} to FF_{16}	PDIDm
C: The first periodicDataIdentifier is mandatory to be present in the request message if the transmissionMode is equal to sendAtSlowRate, sendAtMediumRate, or sendAtFastRate. In case the transmissionMode is equal to stopSending there can either be no periodicDataIdentifier present in order to stop all scheduled periodicDataIdentifier or the client can explicitly specify one or more periodicDataIdentifier(s) to be stopped.				

11.5.2.2 Request message SubFunction parameter $Level (LEV_) definition

This service does not use a SubFunction parameter.

11.5.2.3 Request message data-parameter definition

Table 219 specifies the data-parameters of the request message.

Table 219 — Request message data-parameter definition

Definition
transmissionMode This parameter identifies the transmission rate of the requested periodicDataIdentifiers to be used by the server (see C.4).
periodicDataIdentifier (#1 to #m) This parameter identifies the server data record(s) that are being requested by the client (see C.1 and service description above for detailed parameter definition). It shall be possible to request multiple periodicDataIdentifiers with a single request.

11.5.3 Positive response message

11.5.3.1 Positive response message definition

There shall be a distinction between the initial positive response message, which indicates that the server accepts the service and subsequent periodic data response messages, which include periodicDataIdentifier data.

Table 220 specifies the initial positive response message to be transmitted by the server when it accepts the request.

Table 220 — Positive response message definition

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	ReadDataByPeriodicIdentifier Response SID	M	$6A_{16}$	RDBPIPR

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

The data of a periodicDataIdentifier is transmitted periodically (with updated data) at a rate determined by the transmissionMode parameter of the request.

After the initial positive response, for each supported periodicDataIdentifier in the request the server shall start sending a single periodic data response message as defined below.

Table 221 specifies the periodic data response message data definition.

Table 221 — Periodic data response message data definition

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	periodicDataIdentifier	M	00_{16} to FF_{16}	PDID
#2 : #k+2	dataRecord[] = [data#1 : data#k]	M : U	00_{16} to FF_{16} : 00_{16} to FF_{16}	DREC_ DATA_1 : DATA_k

11.5.3.2 Positive response message data-parameter definition

This service does not support response message data-parameters in the positive response message.

Table 222 specifies the periodic message data-parameters of the defined periodic data response message.

Table 222 — Periodic message data-parameter definition

Definition
periodicDataIdentifier This parameter references a periodicDataIdentifier from the request message.
dataRecord This parameter is used by the ReadDataByPeriodicIdentifier positive response message to provide the requested data record values to the client. The content of the dataRecord is not defined in this document and is vehicle manufacturer specific.

11.5.4 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 223. The listed negative responses shall be used if the error scenario applies to the server.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

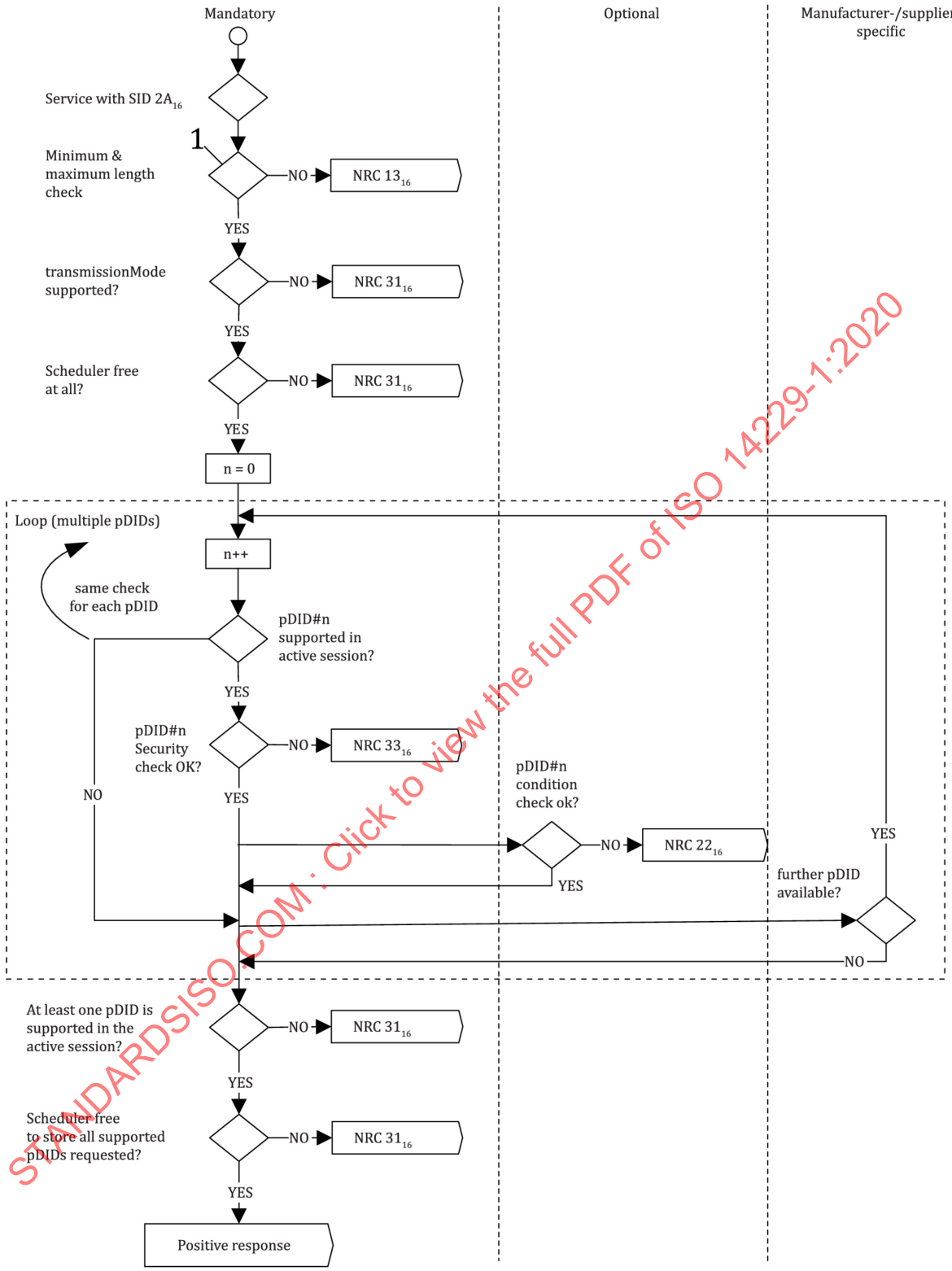
Table 223 — Supported negative response codes

NRC	Description	Mnemonic
13_{16}	**incorrectMessageLengthOrInvalidFormat** This NRC shall be sent if the length of the request message is invalid or the client exceeded the maximum number of periodicDataIdentifiers allowed to be requested at a time.	IMLOIF
22_{16}	**conditionsNotCorrect** This NRC shall be sent if the operating conditions of the server are not met to perform the required action. For example, this could occur if the client requests periodicDataIdentifiers with different transmissionModes and the server does not support multiple transmissionModes simultaneously.	CNC
31_{16}	**requestOutOfRange** This NRC shall be sent if: none of the requested periodicDataIdentifier values are supported by the device; none of the requested periodicDataIdentifiers are supported in the current session; the specified transmissionMode is not supported by the device; the requested dynamicDefinedDataIdentifier has not been assigned yet; the client exceeded the maximum number of periodicDataIdentifiers allowed to be scheduled concurrently.	ROOR
33_{16}	**securityAccessDenied** This NRC shall be sent if at least one of the periodicDataIdentifier is secured and the server is not in an unlocked state.	SAD

The evaluation sequence is documented in Figure 23.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved



STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

Key

1 minimum length is 2 byte if TM = stopSending (SI + TM), minimum length is 3 bytes (SI + TM + pDID) if TM <> stopSending, maximum length is 1 byte (SI) + 1 byte (TM) + n bytes (pDID(s))

Figure 23 — NRC handling for ReadDataByPeriodicIdentifier service

© ISO 2020 – All rights reserved

11.5.5 Message flow example ReadDataByPeriodicIdentifier

11.5.5.1 General assumptions

The examples below show how the ReadDataByPeriodicIdentifer behaves. The client may request a periodicDataIdentifier data at any time independent of the status of the server.

The periodicDataIdentifier examples below are specific to a powertrain device (e.g. engine control module). Refer to ISO°15031-2[16] for further details regarding accepted terms/definitions/acronyms for emission related systems.

11.5.5.2 Example #1 - Read multiple periodicDataIdentifiers $E3_{16}$ and 24_{16} at medium rate

11.5.5.2.1 Assumptions

The example demonstrates requesting of multiple dataIdentifiers with a single request (where periodicDataIdentifier $E3_{16}$ (= dataIdentifier $F2E3_{16}$) contains engine coolant temperature, throttle position, engine speed, vehicle speed sensor, and periodicDataIdentifier 24_{16} (= dataIdentifier $F224_{16}$) contains battery positive voltage, manifold absolute pressure, mass air flow, vehicle barometric pressure, and calculated load value).

The client requests the transmission at medium rate and after a certain amount of time retrieving the periodic data the client stops the transmission of the periodicDataIdentifier $E3_{16}$ only.

11.5.5.2.2 Step #1: Request periodic transmission of the periodicDataIdentifiers

Table 224 specifies the ReadDataByPeriodicIdentifier request message flow example – step #1.

Table 224 — ReadDataByPeriodicIdentifier request message flow example – step #1

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadDataByPeriodicIdentifier Request SID	$2A_{16}$	RDBPI
#2	transmissionMode = sendAtMediumRate	02_{16}	TM_SAMR
#3	periodicDataIdentifier#1	$E3_{16}$	PDID1
#4	periodicDataIdentifier#2	24_{16}	PDID2

Table 225 specifies the ReadDataByPeriodicIdentifier initial positive response message flow example – step #1.

Table 225 — ReadDataByPeriodicIdentifier initial positive response message flow example – step #1

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadDataByPeriodicIdentifier Response SID	$6A_{16}$	RDBPIPR

Table 226 specifies the ReadDataByPeriodicIdentifier subsequent positive response message #1 flows – step #1.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

Table 226 — ReadDataByPeriodicIdentifier subsequent positive response message #1 flows – step #1

Message direction		server → client	
Message type		Response	
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	periodicDataIdentifier#1	$E3_{16}$	PDID1
#2	dataRecord [data#1] = ECT	$A6_{16}$	DREC_DATA_1
#3	dataRecord [data#2] = TP	66_{16}	DREC_DATA_2
#4	dataRecord [data#3] = RPM	07_{16}	DREC_DATA_3
#5	dataRecord [data#4] = RPM	50_{16}	DREC_DATA_4
#6	dataRecord [data#5] = VSS	00_{16}	DREC_DATA_5

Table 227 specifies the ReadDataByPeriodicIdentifier subsequent positive response message #2 flows – step #1.

Table 227 — ReadDataByPeriodicIdentifier subsequent positive response message #2 flows – step #1

Message direction		Server → client	
Message type		Response	
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	periodicDataIdentifier#1	24_{16}	PDID2
#2	dataRecord [data#1] = B+	$8C_{16}$	DREC_DATA_1
#3	dataRecord [data#2] = MAP	20_{16}	DREC_DATA_2
#4	dataRecord [data#3] = MAF	$1A_{16}$	DREC_DATA_3
#5	dataRecord [data#4] = BARO	63_{16}	DREC_DATA_4
#6	dataRecord [data#5] = LOAD	$4A_{16}$	DREC_DATA_5

The server transmits the above shown subsequent response messages at the medium rate as applicable to the server.

11.5.5.2.3 Step #2: Stop the transmission of the periodicDataIdentifiers

Table 228 specifies the ReadDataByIdentifier request message flow example – step #2.

Table 228 — ReadDataByIdentifier request message flow example – step #2

Message direction		client → server	
Message type		Request	
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	ReadDataByPeriodicIdentifier Request SID	$2A_{16}$	RDBPI
#2	transmissionMode = stopSending	04_{16}	TM_SS
#3	periodicDataIdentifier#1	$E3_{16}$	PDID

Table 229 specifies the ReadDataByIdentifier positive response message flow example – step #2.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 229 — ReadDataByIdentifier positive response message flow example – step #2

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadDataByPeriodicIdentifier Response SID	$6A_{16}$	RDBPIPR

The server stops the transmission of the periodicDataIdentifier $E3_{16}$ only. The periodicDataIdentifier 24_{16} is still transmitted at the server medium rate.

11.5.5.3 Example #2 - Graphical and tabular example of ReadDataByPeriodicIdentifier service periodic schedule rates

11.5.5.3.1 ReadDataByPeriodicIdentifier example overview

This subclause contains an example of scheduled periodic data, with both a graphical and tabular example of the ReadDataByPeriodicIdentifier ($2A_{16}$) service.

The example contains a graphical depiction of what messages (request/response) are transmitted between the client and the server application, followed by a table which shows a possible implementation of a server periodic scheduler, its variables and how they change each time the background function that checks the periodic scheduler is executed.

In the examples below, the following implementation is defined:

— The fast periodic rate is 25 ms and the medium periodic rate is 300 ms.

— The periodic scheduler is checked every 12,5 ms, which means that the periodic scheduler background function is called (polled) with this period. Each time the background periodic scheduler is called it will traverse the scheduler entries until a single periodic identifier is sent, or until all identifiers in the scheduler have been checked and none are ready to transmit. In the example implementation the “periodic scheduler transmit index” variable in the tables is the first index checked when traversing the scheduler to see if an identifier is ready for transmit.

— The maximum number of periodicDataIdentifiers which may be scheduled concurrently is 4.

— One unique periodic data response message address information ID is allocated.

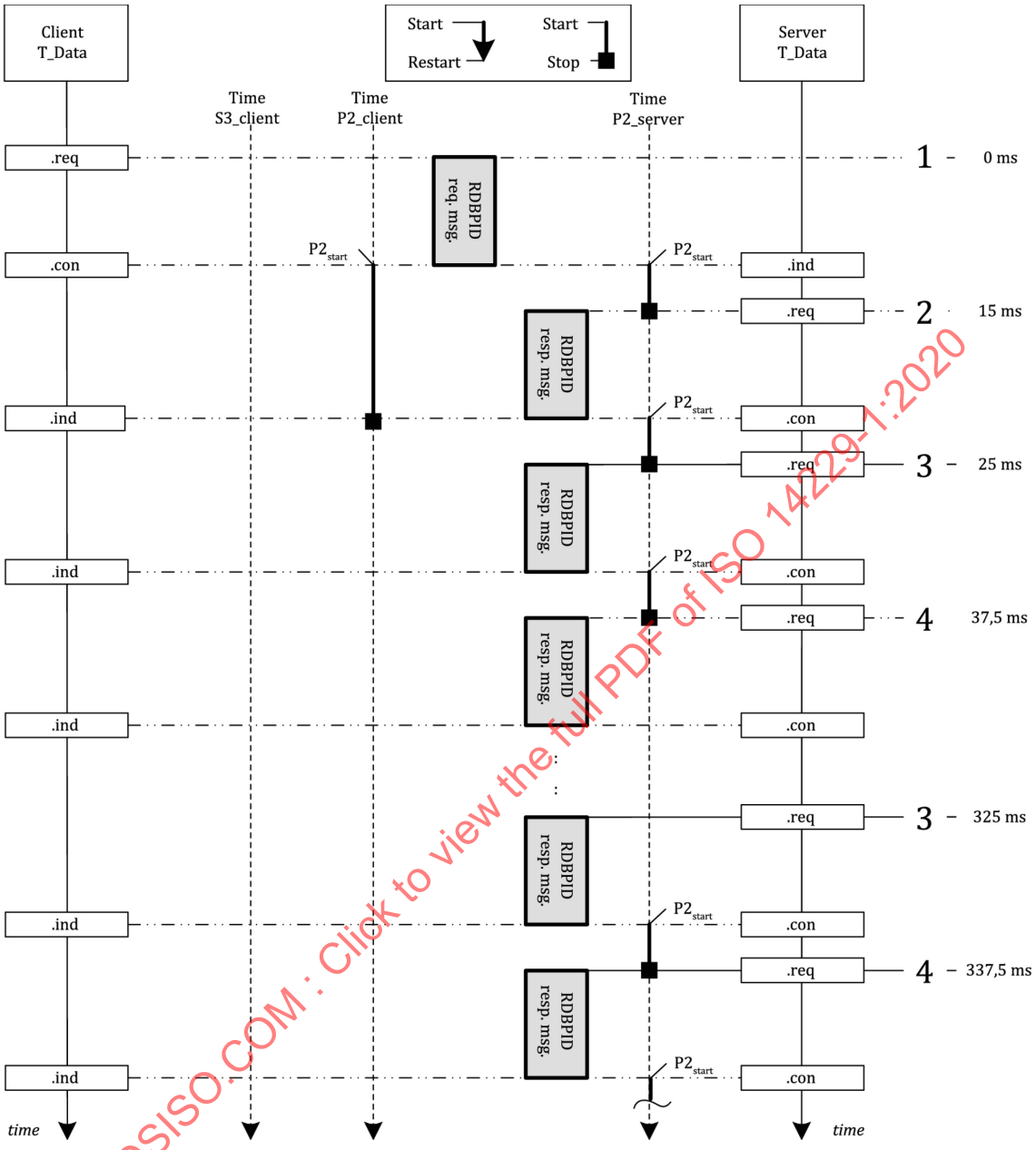
Since the periodic scheduler poll-rate is 12,5 ms, the fast rate loop counter would be set to 2 (this value is based on the scheduled rate (25 ms) divided by the periodic scheduler poll-rate (12,5 ms) or 25/12,5) each time a fast rate periodicDataIdentifier is sent and the medium rate loop counter would be reset to 24 (scheduled rate divided by the periodic scheduler poll-rate or 300/12,5) each time a medium rate periodicDataIdentifier is sent.

11.5.5.3.2 Example #2 – Read multiple periodicDataIdentifiers $E3_{16}$ and 24_{16} at medium rate

At t = 0,0 ms, the client begins sending the request to schedule 2 periodicDataIdentifiers ($F2E3_{16}$ and $F224_{16}$) at the medium rate (300 ms). For the purposes of this example, the server receives the request and executes the periodic scheduler background function the first time t = 25,0 ms, see Figure 24.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved



Key

1 ReadDataByPeriodicIdentifier ($2A_{16}$, 02_{16}, $F2E3_{16}$, $F224_{16}$) request message (sendAtMediumRate)

2 ReadDataByPeriodicIdentifier positive response message ($6A_{16}$, no data included)

3 ReadDataByPeriodicIdentifier periodic data response message ($E3_{16}$, XX_{16}, …, XX_{16})

4 ReadDataByPeriodicIdentifier periodic data response message (24_{16}, XX_{16}, …, XX_{16})

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

Figure 24 — Example #2 – periodicDataIdentifiers scheduled at medium rate

Table 230 and Table 231 show a possible implementation of the periodic scheduler in the server. The table contains the periodic scheduler variables and how they change each time the background function that checks the periodic scheduler is executed.

© ISO 2020 – All rights reserved

Table 230 — Example #2: Periodic scheduler table for scheduler type 1

Time *t* (ms)	Periodic scheduler transmit index	Periodic identifier sent	Periodic scheduler loop #	Scheduler[0] transmit count	Scheduler[1] transmit count
25,0	0	$E3_{16}$	1	0->24	0
37,5	1	24_{16}	2	23	0->24
50,0	0	None	3	22	23
62,5	0	None	4	21	22
75,0	0	None	5	20	21
87,5	0	None	6	19	20
100,0	0	None	7	18	19
112,5	0	None	8	17	18
125,0	0	None	9	16	17
137,5	0	None	10	15	16
150,0	0	None	11	14	15
162,5	0	None	12	13	14
175,0	0	None	13	12	13
187,5	0	None	14	11	12
200,0	0	None	15	10	11
212,5	0	None	16	9	10
225,0	0	None	17	8	9
237,5	0	None	18	7	8
250,0	0	None	19	6	7
262,5	0	None	20	5	6
275,0	0	None	21	4	5
287,5	0	None	22	3	4
300,0	0	None	23	2	3
312,5	0	None	24	1	2
325,0	0	$E3_{16}$	25	0->24	1
337,5	1	24_{16}	26	23	0->24
350,0	0	None	27	22	23
362,5	0	None	28	21	22

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 231 — Example #2: Periodic scheduler table for scheduler type 2

Time *t* (ms)	Periodic scheduler transmit index	Periodic identifier sent	Periodic scheduler loop #	Scheduler[0] transmit count	Scheduler[1] transmit count
25,0	0	$E3_{16}$	1	0->24	0->24
37,5	1	None	2	23	23
50,0	1	None	3	22	22
:	:	:	:	:	:
300,0	1	None	23	2	2
312,5	1	None	24	1	1
325,0	1	24_{16}	25	0->24	0->24
337,5	0	None	26	23	23
:	:	:	:	:	:
612,5	0	None	48	1	1
625,0	0	$E3_{16}$	49	0->24	0->24
637,5	1	None	50	23	23
:	:	:	:	:	:
925,0	1	24_{16}	73	0->24	0->24
937,0	0	None	74	23	23

11.5.5.4 Example #3 - Graphical and tabular example of ReadDataByPeriodicIdentifier service periodic schedule rates

11.5.5.4.1 ReadDataByPeriodicIdentifier example overview

This subclause contains an example of scheduled periodic data with both a graphical and tabular example of the ReadDataByPeriodicIdentifier ($2A_{16}$) service.

The example is based on the example given in 11.5.5.3. The example contains a graphical depiction of what messages (request/response) are transmitted between the client and the server application, followed by a table which shows a possible implementation of a server periodic scheduler, its variables and how they change each time the background function that checks the periodic scheduler is executed.

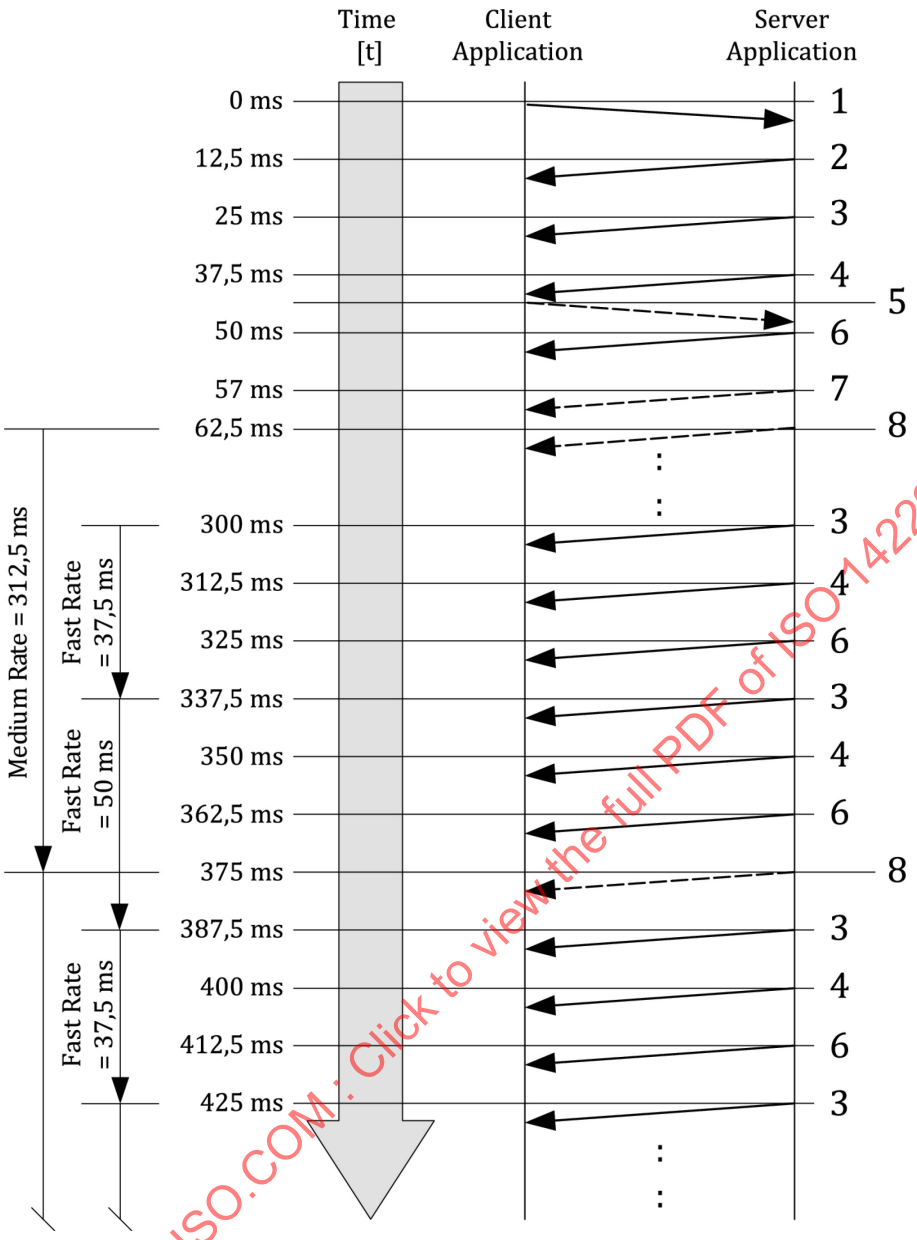
11.5.5.4.2 Read multiple periodicDataIdentifiers at different periodic rates

In this example, three periodicDataIdentifiers (for simplicity 01_{16}, 02_{16}, 03_{16}) are scheduled at the fast periodic rate (25 ms) and then another request is sent for a single periodicDataIdentifier (04_{16}) to be scheduled at the medium periodic rate (300 ms). For the purposes of this example, the server receives the first ReadDataByPeriodicIdentifier request (1), sends a positive response (2) without any periodic data and executes the periodic scheduler background function for the first time *t* = 25,0 ms (3). When the second ReadDataByPeriodicIdentifier request (5) is received, the server sends a positive response (7) without any periodic data and starts executing the periodic scheduler background function at t = 62,5 ms (8) at a scheduled medium rate of 300 ms.

Figure 25 depicts the example #3 – periodicDataIdentifiers scheduled at fast and medium rate.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved



STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

Key

1 ReadDataByPeriodicIdentifier ($2A_{16}$, 03_{16}, $F201_{16}$, $F202_{16}$, $F203_{16}$) request message (sendAtFastRate)
2 ReadDataByPeriodicIdentifier positive response message ($6A_{16}$, no data included)
3 ReadDataByPeriodicIdentifier periodic data response message (01_{16}, XX_{16}, …, XX_{16})
4 ReadDataByPeriodicIdentifier periodic data response message (02_{16}, XX_{16}, …, XX_{16})
5 ReadDataByPeriodicIdentifier ($2A_{16}$, 02_{16}, $F204_{16}$) request message (sendAtMediumRate)
6 ReadDataByPeriodicIdentifier periodic data response message (03_{16}, XX_{16}, …, XX_{16})
7 ReadDataByPeriodicIdentifier positive response message ($6A_{16}$, no data included)
8 ReadDataByPeriodicIdentifier periodic data response message (04_{16}, XX_{16}, …, XX_{16})

Figure 25 — Example #3 – periodicDataIdentifiers scheduled at fast and medium rate

Table 232 shows a possible implementation of the periodic scheduler in the server. The table contains the periodic scheduler variables and how they change each time the background function that checks the periodic scheduler is executed. Table 232 also shows aliasing for the three fast rate periodicDataIdentifiers.

© ISO 2020 – All rights reserved

Table 232 — Example #3: Periodic scheduler table for scheduler type 1

Time *t* (ms)	Periodic scheduler transmit index	Periodic identifier sent	Periodic scheduler loop #	Scheduler transmit count			
				[0]	[1]	[2]	[3]
25,0	0	01_{16}	1	0->2	0	0	N/A
37,5	1	02_{16}	2	1	0->2	0	N/A
50,0	2	03_{16}	3	0	1	0->2	0
62,5	3	04_{16}	4	0	0	1	0->24
75,0	0	01_{16}	5	0->2	0	0	23
87,5	1	02_{16}	6	1	0->2	0	22
100,0	2	03_{16}	7	0	1	0->2	21
112,5	3	01_{16}	8	0->2	0	1	20
125,0	1	02_{16}	9	1	0->2	0	19
137,5	2	03_{16}	10	0	1	0->2	18
150,0	3	01_{16}	11	0->2	0	1	17
162,5	1	02_{16}	12	1	0->2	0	16
175,0	2	03_{16}	13	0	1	0->2	15
187,5	3	01_{16}	14	0->2	0	1	14
200,0	1	02_{16}	15	1	0->2	0	13
212,5	2	03_{16}	16	0	1	0->2	12
225,0	3	01_{16}	17	0->2	0	1	11
237,5	1	02_{16}	18	1	0->2	0	10
250,0	2	03_{16}	19	0	1	0->2	9
262,5	3	01_{16}	20	0->2	0	1	8
275,0	1	02_{16}	21	1	0->2	0	7
287,5	2	03_{16}	22	0	1	0->2	6
300,0	3	01_{16}	23	0->2	0	1	5
312,5	1	02_{16}	24	1	0->2	0	4
325,0	2	03_{16}	25	0	1	0->2	3
337,5	3	01_{16}	26	0->2	0	1	2
350,0	1	02_{16}	27	1	0->2	0	1
362,5	2	03_{16}	28	0	1	0->2	0
375,0	3	04_{16}	29	0	0	1	0->24
387,5	0	01_{16}	30	0->2	0	0	23

Table 233 modifies example 3 to add a second medium rate periodicDataIdentifiers (05_{16}) for the example of a scheduler type 2. The example implementation is simplified by using separate fast and medium rate transmit indexes.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 233 — Example #3: Periodic scheduler table for scheduler type 2

Time *t* (ms)	Periodic scheduler transmit		Periodic identifier sent	Periodic scheduler loop #	Scheduler				
	index fast	index medium			[0]	[1]	[2]	[3]	[4]
					transmit count				
25,0	0	3	01_{16}	1	0->2	0->2	0->2	N/A	N/A
37,5	1	3	none	2	1	1	1	N/A	N/A
50,0	1	3	02_{16}	3	0->2	0->2	0->2	N/A	N/A
62,5	2	3	04_{16}	4	1	1	1	0->24	0->24
75,0	2	4	03_{16}	5	0->2	0->2	0->2	23	23
87,5	0	4	none	6	1	1	1	22	22
100,0	1	4	01_{16}	7	0->2	0->2	0->2	21	21
112,5	1	4	none	8	1	1	1	20	20
125,0	1	4	02_{16}	9	0->2	0->2	0->2	19	19
137,5	2	4	none	10	1	1	1	18	18
150,0	2	4	03_{16}	11	0->2	0->2	0->2	17	17
162,5	0	4	none	12	1	1	1	16	16
175,0	0	4	01_{16}	13	0->2	0->2	0->2	15	15
187,5	1	4	none	14	1	1	1	14	14
200,0	1	4	02_{16}	15	0->2	0->2	0->2	13	13
212,5	2	4	none	16	1	1	1	12	12
225,0	2	4	03_{16}	17	0->2	0->2	0->2	11	11
237,5	0	4	none	18	1	1	1	10	10
250,0	0	4	01_{16}	19	0->2	0->2	0->2	9	9
262,5	1	4	none	20	1	1	1	8	8
275,0	1	4	02_{16}	21	0->2	0->2	0->2	7	7
287,5	2	4	none	22	1	1	1	6	6
300,0	2	4	03_{16}	23	0->2	0->2	0->2	5	5
312,5	0	4	none	24	1	1	1	4	4
325,0	0	4	01_{16}	25	0->2	0->2	0->2	3	3
337,5	1	4	none	26	1	1	1	2	2
350,0	1	4	02_{16}	27	0->2	0->2	0->2	1	1
362,5	2	4	05_{16}	28	1	1	1	0->24	0->24
375,0	2	3	03_{16}	29	0->2	0->2	0->2	23	23
387,5	3	3	none	30	1	1	1	22	22

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

11.5.5.5 Example #4 - Tabular example of ReadDataByPeriodicIdentifier service periodic schedule rates

This subclause contains an example of scheduled periodic data with a tabular example of the ReadDataByPeriodicIdentifier ($2A_{16}$) service. The example contains a table which shows a possible implementation of a server periodic scheduler, its variables and how they change each time the background function that checks the periodic scheduler is executed.

In the examples below, the following information is defined:

— The fast periodic rate is 10 ms.

— The periodic scheduler is checked every 10 ms, which means that the periodic scheduler background function is called (polled) with this period.

— The maximum number of periodicDataIdentifiers, which may be scheduled concurrently, is 16.

— Two unique periodic data response message address information IDs are allocated.

Since the periodic scheduler poll-rate is 10 ms, the fast rate loop counter would be set to 1 (this value is based on the scheduled rate (10 ms) divided by the periodic scheduler poll-rate (10 ms)) each time a fast rate periodicDataIdentifier is sent.

At t = 0,0 ms, the client begins sending the request to schedule 2 periodicDataIdentifier (for simplicity 01_{16}, 02_{16}) at the fast periodic rate (10 ms). For the purposes of this example, the server receives the request and executes the periodic scheduler background function the first time t = 10 ms.

Table 234 — Example #4: Periodic scheduler table

Time t (ms)	Response message identifier#	Periodic identifier sent	Periodic scheduler loop #
10	1	01_{16}	1
10	2	02_{16}	1
20	1	01_{16}	2
20	2	02_{16}	2
30	1	01_{16}	3
30	2	02_{16}	3
40	1	01_{16}	4
40	2	02_{16}	4
50	1	01_{16}	5
50	2	02_{16}	5
60	1	01_{16}	6
60	2	02_{16}	6
70	1	01_{16}	7
70	2	02_{16}	7
80	1	01_{16}	8
80	2	02_{16}	8
90	1	01_{16}	9

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Time *t* (ms)	Response message identifier#	Periodic identifier sent	Periodic scheduler loop #
90	2	02_{16}	9
100	1	01_{16}	10
100	2	02_{16}	10

11.5.5.6 Example #5 - Tabular example of ReadDataByPeriodicIdentifier service periodic schedule rates

This subclause uses the same assumptions as in 11.5.5.5. In this example, more periodicDataIdentifiers than unique periodic data response message address information IDs in the response message set are requested.

At t = 0,0 ms, the client begins sending the request to schedule 3 periodicDataIdentifier (for simplicity 01_{16}, 02_{16}, 03_{16}) at the fast periodic rate (10 ms). For the purposes of this example, the server receives the request and executes the periodic scheduler background function the first time t = 10 ms.

Table 235 — Example #5: Periodic scheduler table

Time *t* (ms)	Response message ID#	Periodic identifier sent	Periodic scheduler loop #
10	1	01_{16}	1
10	2	02_{16}	1
20	1	03_{16}	2
20	2	01_{16}	2
30	1	02_{16}	3
30	2	03_{16}	3
40	1	01_{16}	4
40	2	02_{16}	4
50	1	03_{16}	5
50	2	01_{16}	5
60	1	02_{16}	6
60	2	03_{16}	6
70	1	01_{16}	7
70	2	02_{16}	7
80	1	03_{16}	8
80	2	01_{16}	8
90	1	02_{16}	9
90	2	03_{16}	9
100	1	01_{16}	10
100	2	02_{16}	10

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

11.6 DynamicallyDefineDataIdentifier ($2C_{16}$) service

11.6.1 Service description

The DynamicallyDefineDataIdentifier service allows the client to dynamically define in a server a data identifier that can be read via the ReadDataByIdentifier service at a later time.

The intention of this service is to provide the client with the ability to group one or more data elements into a data superset that can be requested en masse via the ReadDataByIdentifier or ReadDataByPeriodicIdentifier service. The data elements to be grouped together can either be referenced by:

— a source data identifier, a position and size or

— a memory address and a memory length, or

— a combination of the two methods listed above using multiple requests to define the single data element. The dynamically defined dataIdentifier will then contain a concatenation of the data-parameter definitions.

This service allows greater flexibility in handling ad hoc data needs of the diagnostic application that extend beyond the information that can be read via statically defined data identifiers, and can also be used to reduce bandwidth utilization by avoiding overhead penalty associated with frequent request/response transactions.

The definition of the dynamically defined data identifier can either be done via a single request message or via multiple request messages. This allows for the definition of a single data element referencing source identifier(s) and memory addresses. The server shall concatenate the definitions for the single data element. A redefinition of a dynamically defined data identifier can be achieved by clearing the current definition and start over with the new definition. When multiple DynamicallyDefineDataIdentifier request messages are used to configure a single DataIdentifier and the server detects the overrun of the maximum number of bytes during a subsequent request for this DataIdentifier (e.g. definition of a periodicDataIdentifier), then the server shall leave the definition of the DataIdentifier as it was prior to the request that would have led to the overrun.

Although this service does not prohibit such functionality, it is not recommended for the client to reference one dynamically defined data record from another, because deletion of the referenced record could create data consistency problems within the referencing record.

This service also provides the ability to clear an existing dynamically defined data record. Requests to clear a data record shall be positively responded to if the specified data record identifier is within the range of valid dynamic data identifiers supported by the server (see C.1 for more details).

The server shall maintain the dynamically defined data record until it is cleared or as specified by the vehicle manufacturer (e.g. deletion of dynamically defined data records upon session transition or upon power down of the server).

The server can implement data records in two different ways:

— Composite data records containing multiple elemental data records which are not individually referenced.

— Unique 2-byte identification “tag” or dataIdentifier (DID) value for individual, elemental data records supported within the server (an example elemental data record, or DID, is engine speed or intake air temperature). This implementation of data records is a subset of a composite data record implementation, because it only references a single elemental data record instead of a data record including multiple elemental data records.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Both types of implementing data records are supported by the DynamicallyDefineDataIdentifier service to define a dynamic data identifier.

— Composite block of data: The position parameter shall reference the starting point in the composite block of data and the size parameter shall reflect the length of data to be placed in the dynamically defined data identifier. The tester is responsible to not include only a portion of an elemental data record of the composite block of data in the dynamic data record.

— 2-byte DID: The position parameter shall be set to one and the size parameter shall reflect the length of the DID (length of the elemental data record). The tester is responsible to not include only a portion of the 2-byte DID value in the dynamic data record.

The ordering of the data within the dynamically defined data record shall be of the same order as it was specified in the client request message(s). Also, first position of the data specified in the client's request shall be oriented such that it occurs closest to the beginning of the dynamic data record, in accordance with the ordering requirement mentioned in the preceding sentence.

In addition to the definition of a dynamic data identifier via a logical reference (a record data identifier) this service provides the capability to define a dynamically defined data identifier via an absolute memory address and a memory length information. This mechanism of defining a dynamic data identifier is recommended to be used only during the development phase of a server.

IMPORTANT — The server and the client shall meet the request and response message behaviour as specified in 8.7.

11.6.2 Request message

11.6.2.1 Request message definition

Table 236 specifies the request message – SubFunction = defineByIdentifier.

Table 236 — Request message definition - SubFunction = defineByIdentifier

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	DynamicallyDefineDataIdentifier Request SID	M	$2C_{16}$	DDDI
#2	SubFunction = [definitionType = defineByIdentifier]	M	01_{16}	LEV_ DBID
#3 #4	dynamicallyDefinedDataIdentifier[] = [byte#1 (MSB) byte#2 (LSB)]	M M	$F2_{16}$/$F3_{16}$ 00_{16} to FF_{16}	DDDDI_ HB LB
#5 #6	sourceDataIdentifier[]#1 = [byte#1 (MSB) byte#2 (LSB)]	M M	00_{16} to FF_{16} 00_{16} to FF_{16}	SDI_ HB LB
#7	positionInSourceDataRecord#1	M	01_{16} to FF_{16}	PISDR#1
#8	memorySize#1	M	01_{16} to FF_{16}	MS#1
:	:	:	:	:
#n-3 #n-2	sourceDataIdentifier[]#m = [byte#1 (MSB) byte#2 (LSB)]	U U	00_{16} to FF_{16} 00_{16} to FF_{16}	SDI_ HB LB
#n-1	positionInSourceDataRecord#m	U	01_{16} to FF_{16}	PISDR#m
#n	memorySize#m	U	01_{16} to FF_{16}	MS#m

Table 237 specifies the request message – SubFunction = defineByMemoryAddress.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

Table 237 — Request message definition - SubFunction = defineByMemoryAddress

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	DynamicallyDefineDataIdentifier Request SID	M	$2C_{16}$	DDDI
#2	SubFunction = [definitionType = defineByMemoryAddress]	M	02_{16}	LEV_ DBMA
#3 #4	dynamicallyDefinedDataIdentifier[] = [byte#1 (MSB) byte#2 (LSB)]	M M	$F2_{16}$/$F3_{16}$ 00_{16} to FF_{16}	DDDDI_ HB LB
#5	addressAndLengthFormatIdentifier	M1	00_{16} to FF_{16}	ALFID
#6 : #(m-1)+6	memoryAddress[] = [byte#1 (MSB) : byte#m]	M : C1	00_{16} to FF_{16} : 00_{16} to FF_{16}	MA_ B1 : Bm
#m+6 : #m+6+(k-1)	memorySize[] = [byte#1 (MSB) : byte#k]	M : C2	00_{16} to FF_{16} : 00_{16} to FF_{16}	MS_ B1 : Bk
:	:	:	:	:
#n-k-(m-1) : #n-k	memoryAddress[] = [byte#1 (MSB) : byte#m]	U : U/C1	00_{16} to FF_{16} : 00_{16} to FF_{16}	MA_ B1 : Bm
#n-(k-1) : #n	memorySize[] = [byte#1 (MSB) : byte#k]	U : U/C2	00_{16} to FF_{16} : 00_{16} to FF_{16}	MS_ B1 : Bk

M1: The addressAndLengthFormatIdentifier parameter is only present once at the very beginning of the request message and specifies the length of the address and length information for each memory location reference throughout the whole request message.

C1: The presence of this parameter depends on address length information parameter of the addressAndLengthFormatIdentifier.

C2: The presence of this parameter depends on the memory size length information of the addressAndLengthFormatIdentifier.

Table 238 specifies the request message – SubFunction = clearDynamicallyDefinedDataIdentifier.

Table 238 — Request message definition - SubFunction = clearDynamicallyDefinedDataIdentifier

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	DynamicallyDefineDataIdentifier Request SID	M	$2C_{16}$	DDDI
#2	SubFunction = [definitionType = clearDynamicallyDefinedDataIdentifier]	M	03_{16}	LEV_ CDDDID
#3 #4	dynamicallyDefinedDataIdentifier[] = [byte#1 (MSB) byte#2 (LSB)]	C C	$F2_{16}$/$F3_{16}$ 00_{16} to FF_{16}	DDDDI_ HB LB

C: The presence of this parameter requires the server to clear the dynamicallyDefinedDataIdentifier included in byte#1 and byte#2. If the parameter is not present all dynamicallyDefinedDataIdentifier in the server shall be cleared.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

11.6.2.2 Request message SubFunction parameter $Level (LEV_) definition

The sub-parameters defined as valid for the request message of this service are indicated in Table 239 (suppressPosRspMsgIndicationBit (bit 7) not shown).

Table 239 — Request message SubFunction parameter definition

Bits 6 – 0	Description	Cvt	Mnemonic
00	**ISOSAEReserved** This value is reserved by this document for future definition.	M	ISOSAERESRVD
01	**defineByIdentifier** This value shall be used to specify to the server that definition of the dynamic data identifier shall occur via a data identifier reference.	U	DBID
02	**defineByMemoryAddress** This value shall be used to specify to the server that definition of the dynamic data identifier shall occur via an address reference.	U	DBMA
03	**clearDynamicallyDefinedDataIdentifier** This value shall be used to clear the specified dynamic data identifier. Note that the server shall positively respond to a clear request from the client, even if the specified dynamic data identifier does not exist at the time of the request. The specified dynamic data identifier is required to be within a valid range (see C.1 for allowable ranges). If the specified dynamic data identifier is being reported periodically at the time of the request, the dynamic identifier shall first be stopped and then cleared.	U	CDDDI
04-7F	**ISOSAEReserved** This range of values is reserved by this document for future definition.	M	ISOSAERESRVD

11.6.2.3 Request message data-parameter definition

Table 240 specifies the data-parameters of the request message.

Table 240 — Request message data-parameter definition

Definition
dynamicallyDefinedDataIdentifier This parameter specifies how the dynamic data record, which is being defined by the client, will be referenced in future calls to the service ReadDataByIdentifier or ReadDataByPeriodicDataIdentifier. The dynamicallyDefinedDataIdentifier shall be handled as a dataIdentifier in the ReadDataByIdentifier service (see C.1 for further details). It shall be handled as a periodicRecordIdentifier in the ReadDataByPeriodicDataIdentifier service (see the ReadDataByPeriodicDataIdentifier service for requirements on the value of this parameter in order to be able to request the dynamically defined data identifier periodically).
sourceDataIdentifier This parameter is only present for SubFunction = defineByIdentifier. This parameter logically specifies the source of information to be included into the dynamic data record. For example, this could be a 2-byte DID used to reference engine speed, or a 2-byte DID used to reference a composite block of information containing engine speed, vehicle speed, intake air temperature, etc. (see C.1 for further details).
positionInSourceDataRecord This parameter is only present for SubFunction = defineByIdentifier. This 1-byte parameter is used to specify the starting byte position of the excerpt of the source data record to be included in the dynamic data record. A position of one shall reference the first byte of the data record referenced by the sourceDataIdentifier.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Definition
addressAndLengthFormatIdentifier This parameter is a one Byte value with each nibble encoded separately (see Table H.1 for example values): bit 7 to 4: Length (number of bytes) of the memorySize parameter(s); bit 3 to 0: Length (number of bytes) of the memoryAddress parameter(s).
memoryAddress This parameter is only present for SubFunction = defineByMemoryAddress. This parameter specifies the memory source address of information to be included into the dynamic data record. The number of bytes used for this address is defined by the low nibble (bit 3 to 0) of the addressAndLengthFormatIdentifier. Byte#m in the memoryAddress parameter is always the least significant byte of the address being referenced in the server. The most significant byte(s) of the address can be used as a memory identifier.
memorySize This parameter is used to specify the total number of bytes from the source data record/memory address that are to be included in the dynamic data record. The memorySize shall be greater than 1. In case of SubFunction = defineByIdentifier then the positionInSourceDataRecord parameter is used in addition to specify the starting position in the source data identifier from where the memorySize applies. The number of bytes used for this size is one byte. In case of SubFunction = defineByMemoryAddress then this parameter reflects the number of bytes to be included in the dynamically defined data identifier starting at the specified memoryAddress. The number of bytes used for this size is defined by the high nibble (bit 7 to 4) of the addressAndLengthFormatIdentifier.

11.6.3 Positive response message

11.6.3.1 Positive response message definition

Table 241 specifies the positive response message.

Table 241 — Positive response message definition

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	DynamicallyDefineDataIdentifier Response SID	M	$6C_{16}$	DDDIPR
#2	SubFunction = [definitionType]	M	00_{16} to $7F_{16}$	DM
#3 #4	dynamicallyDefinedDataIdentifier [] = [byte#1 (MSB) byte#2 (LSB)]	 C C	 $F2_{16}$/$F3_{16}$ 00_{16} to FF_{16}	DDDDI_ HB LB
C: The presence of this parameter is required if the dynamicallyDefinedDataIdentifier parameter is present in the request message, otherwise the parameter shall not be included.				

11.6.3.2 Positive response message data-parameter definition

Table 242 specifies the data-parameters of the positive response message.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 242 — Response message data-parameter definition

Definition
definitionType This parameter is an echo of bits 6 to 0 of the SubFunction parameter from the request message.
dynamicallyDefinedDataIdentifier This parameter is an echo of the data-parameter dynamicallyDefinedDataIdentifier from the request message.

11.6.4 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 243. The listed negative responses shall be used if the error scenario applies to the server.

Table 243 — 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 sent if the operating conditions of the server are not met to perform the required action.	CNC
31_{16}	**requestOutOfRange**	ROOR
	This NRC shall be sent if: — any data identifier (dynamicallyDefinedDataIdentifier or any sourceDataIdentifier) in the request message is not supported/invalid; — the positionInSourceDataRecord was incorrect (less than 1, or greater than maximum allowed by server); — any memory address in the request message is not supported in the server; — the specified memorySize was invalid; — the amount of data to be packed into the dynamic data identifier exceeds the maximum allowed by the server; — the specified addressAndLengthFormatIdentifier is not valid; — the total length of a dynamically defined periodicDataIdentifier exceeds the maximum length that fits into a single frame of the data link used for transmission of the periodic response message.	

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

NRC	Description	Mnemonic
33_{16}	**securityAccessDenied** This NRC shall be sent if: — any data identifier (dynamicallyDefinedDataIdentifier or any sourceDataIdentifier) in the request message is secured and the server is not in an unlocked state; — any memory address in the request message is secured and the server is not in an unlocked state.	SAD

11.6.5 Message flow examples DynamicallyDefineDataIdentifier

11.6.5.1 Assumptions

This subclause specifies the conditions to be fulfilled for the example to perform a DynamicallyDefineDataIdentifier service.

The service in this example is not limited by any restriction of the server.

In the first example, the server supports 2-byte identifiers (DIDs), which reference a single data information. The example builds a dynamic data identifier using the defineByIdentifier method and then sends a ReadDataByIdentifier request to read the just configured dynamic data identifier.

In the second example, the server supports data identifiers, which reference a composite block of data containing multiple data information. The example builds a dynamic identifier also using the defineByIdentifier method, and sends a ReadDataByIdentifier request to read the just defined data identifier.

The third example builds a dynamic data identifier using the defineByMemoryAddress method, and sends a ReadDataByIdentifier request to read the just defined data identifier.

In the fourth example, the server supports data identifiers, which reference a composite block of data containing multiple data information. The example builds a dynamic data identifier using the defineByIdentifier method and then uses the ReadDataByPeriodicIdentifier service to requests the dynamically defined data identifier to be sent periodically by the server.

The fifth example demonstrates the deletion of a dynamically defined data identifier.

Table 244 shall be used for the examples below. The values being reported may change over time on a real vehicle, but are shown to be constants for the sake of clarity.

Refer to ISO 15031-2[16] for further details regarding accepted terms/definitions/acronyms for emissions-related systems.

For all examples the client requests to have a response message by setting the suppressPosRspMsgIndicationBit (bit 7 of the SubFunction parameter) to "FALSE" ('0').

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 244 — Composite data blocks - DataIdentifier definitions

Data Identifier (block)	Data byte	Data Record Contents	Byte value
$010A_{16}$	#1	dataRecord [data#1] = B+	$8C_{16}$
	#2	dataRecord [data#2] = ECT	$A6_{16}$
	#3	dataRecord [data#3] = TP	66_{16}
	#4	dataRecord [data#4] = RPM	07_{16}
	#5	dataRecord [data#5] = RPM	50_{16}
	#6	dataRecord [data#6] = MAP	20_{16}
	#7	dataRecord [data#7] = MAF	$1A_{16}$
	#8	dataRecord [data#8] = VSS	00_{16}
	#9	dataRecord [data#9] = BARO	63_{16}
	#10	dataRecord [data#10] = LOAD	$4A_{16}$
	#11	dataRecord [data#11] = IAC	82_{16}
	#12	dataRecord [data#12] = APP	$7E_{16}$
$050B_{16}$	#1	dataRecord [data#1] = SPARKADV	00_{16}
	#2	dataRecord [data#2] = KS	91_{16}

Table 245 specifies the elemental data records – DID definitions.

Table 245 — Elemental data records - DID definitions

Data Identifier (DID)	Data byte	Data Record Contents	Byte value
1234_{16}	#1	EOT (MSB)	$4C_{16}$
	#2	EOT (LSB)	36_{16}
5678_{16}	#1	AAT	$4D_{16}$
$9ABC_{16}$	#1	EOL (MSB)	49_{16}
	#2	EOL	21_{16}
	#3	EOL	00_{16}
	#4	EOL (LSB)	17_{16}

Table 246 specifies the memory data records – memory address definitions.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 246 — Memory data records - Memory address definitions

Memory Address	Data byte	Data Record Contents	Byte value
21091968_{16}	#1	dataRecord [data#1] = B+	$8C_{16}$
	#2	dataRecord [data#2] = ECT	$A6_{16}$
	#3	dataRecord [data#3] = TP	66_{16}
	#4	dataRecord [data#4] = RPM	07_{16}
	#5	dataRecord [data#5] = RPM	50_{16}
	#6	dataRecord [data#6] = MAP	20_{16}
	#7	dataRecord [data#7] = MAF	$1A_{16}$
	#8	dataRecord [data#8] = VSS	00_{16}
	#9	dataRecord [data#9] = BARO	63_{16}
	#10	dataRecord [data#10] = LOAD	$4A_{16}$
	#11	dataRecord [data#11] = IAC	82_{16}
	#12	dataRecord [data#12] = APP	$7E_{16}$
13101994_{16}	#1	dataRecord [data#1] = SPARKADV	00_{16}
	#2	dataRecord [data#2] = KS	91_{16}

11.6.5.2 Example #1: DynamicallyDefineDataIdentifier, SubFunction = defineByIdentifier

The example in Table 247 will build up a dynamic data identifier (DDDI $F301_{16}$) containing engine oil temperature, ambient air temperature, and engine oil level using the 2-byte DIDs as the reference for the required data.

Table 247 — DynamicallyDefineDataIdentifier request DDDDI $F301_{16}$ message flow example #1

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	DynamicallyDefineDataIdentifier Request SID	$2C_{16}$	DDDI
#2	SubFunction = defineByIdentifier, suppressPosRspMsgIndicationBit = FALSE	01_{16}	DBID
#3	dynamicallyDefinedDataIdentifier [byte#1] (MSB)	$F3_{16}$	DDDDI_B1
#4	dynamicallyDefinedDataIdentifier [byte#2] (LSB)	01_{16}	DDDDI_B2
#5	sourceDataIdentifier#1 [byte#1] (MSB) - Engine Oil Temperature	12_{16}	SDI_B1
#6	sourceDataIdentifier#1 [byte#2]	34_{16}	SDI_B2
#7	positionInSourceDataRecord#1	01_{16}	PISDR#1
#8	memorySize#1	02_{16}	MS#1
#9	sourceDataIdentifier#2 [byte#1] (MSB) - Ambient Air Temperature	56_{16}	SDI_B1
#10	sourceDataIdentifier#2 [byte#2]	78_{16}	SDI_B2
#11	positionInSourceDataRecord#2	01_{16}	PISDR#2

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Message direction	client → server		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#12	memorySize#2	01_{16}	MS#2
#13	sourceDataIdentifier#3 [byte#1] (MSB) - Engine Oil Level	$9A_{16}$	SDI_B1
#14	sourceDataIdentifier#3 [byte#2]	BC_{16}	SDI_B2
#15	positionInSourceDataRecord#3	01_{16}	PISDR#3
#16	memorySize#3	04_{16}	MS#3

Table 248 specifies the DynamicallyDefineDataIdentifier positive response DDDDI $F301_{16}$ message flow example #1.

Table 248 — DynamicallyDefineDataIdentifier positive response DDDDI $F301_{16}$ message flow example #1

Message direction	server → client		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	DynamicallyDefineDataIdentifier Response SID	$6C_{16}$	DDDIPR
#2	definitionMode = defineByIdentifier	01_{16}	DBID
#3	dynamicallyDefinedDataIdentifier [byte#1] (MSB)	$F3_{16}$	DDDDI_B1
#4	dynamicallyDefinedDataIdentifier [byte#2] (LSB)	01_{16}	DDDDI_B2

Table 249 specifies the ReadDataByIdentifier request DDDDI $F301_{16}$ message flow example #1.

Table 249 — ReadDataByIdentifier request DDDDI $F301_{16}$ message flow example #1

Message direction	client → server		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadDataByIdentifier Request SID	22_{16}	RDBI
#2	dataIdentifier [byte#1] (MSB)	$F3_{16}$	DID_B1
#3	dataIdentifier [byte#2] (LSB)	01_{16}	DID_B2

Table 250 specifies the ReadDataByIdentifier positive response DDDDI $F301_{16}$ message flow example #1.

Table 250 — ReadDataByIdentifier positive response DDDDI $F301_{16}$ message flow example #1

Message direction	server → client		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadDataByIdentifier Response SID	62_{16}	RDBIPR
#2	dataIdentifier [byte#1] (MSB)	$F3_{16}$	DID_B1
#3	dataIdentifier [byte#2] (LSB)	01_{16}	DID_B2
#4	dataRecord [data#1] = EOT	$4C_{16}$	DREC_DATA_1

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Message direction	server → client		
Message type	Response		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#5	dataRecord [data#2] = EOT	36_{16}	DREC_DATA_2
#6	dataRecord [data#3] = AAT	$4D_{16}$	DREC_DATA_3
#7	dataRecord [data#4] = EOL	49_{16}	DREC_DATA_4
#8	dataRecord [data#5] = EOL	21_{16}	DREC_DATA_5
#9	dataRecord [data#6] = EOL	00_{16}	DREC_DATA_6
#10	dataRecord [data#7] = EOL	17_{16}	DREC_DATA_7

11.6.5.3 Example #2: DynamicallyDefineDataIdentifier, SubFunction = defineByIdentifier

The example in Table 251 will build up a dynamic data identifier (DDDDI $F302_{16}$) containing engine coolant temperature (from data record $010A_{16}$), engine speed (from data record $010A_{16}$), IAC Pintle Position (from data record $010A_{16}$) and knock sensor (from data record $050B_{16}$).

Table 251 — DynamicallyDefineDataIdentifier request DDDDI $F302_{16}$ message flow example #2

Message direction	client → server		
Message type	Request		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	DynamicallyDefineDataIdentifier Request SID	$2C_{16}$	DDDI
#2	SubFunction = defineByIdentifier, suppressPosRspMsgIndicationBit = FALSE	01_{16}	DBID
#3	dynamicallyDefinedDataIdentifier [byte#1] (MSB)	$F3_{16}$	DDDDI_B1
#4	dynamicallyDefinedDataIdentifier [byte#2] (LSB)	02_{16}	DDDDI_B2
#5	sourceDataIdentifier#1 [byte#1] (MSB)	01_{16}	SDI_B1
#6	sourceDataIdentifier#1 [byte#2] (LSB)	$0A_{16}$	SDI_B2
#7	positionInSourceDataRecord#1 - Engine Coolant Temperature	02_{16}	PISDR#1
#8	memorySize#1	01_{16}	MS#1
#9	sourceDataIdentifier#2 [byte#1] (MSB)	01_{16}	SDI_B1
#10	sourceDataIdentifier#2 [byte#2] (LSB)	$0A_{16}$	SDI_B2
#11	positionInSourceDataRecord#2 - Engine Speed	04_{16}	PISDR#2
#12	memorySize#2	02_{16}	MS#2
#13	sourceDataIdentifier#3 [byte#1] (MSB)	01_{16}	SDI_B1
#14	sourceDataIdentifier#3 [byte#2] (LSB)	$0A_{16}$	SDI_B2
#15	positionInSourceDataRecord#3 – Idle Air Control	$0B_{16}$	PISDR#3
#16	memorySize#3	01_{16}	MS#3
#17	sourceDataIdentifier#4 [byte#1] (MSB)	05_{16}	SDI_B1
#18	sourceDataIdentifier#4 [byte#2] (LSB)	$0B_{16}$	SDI_B2
#19	positionInSourceDataRecord#4 - Knock Sensor	02_{16}	PISDR#4
#20	memorySize#4	01_{16}	MS#4

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 252 specifies the DynamicallyDefineDataIdentifier positive response DDDDI $F302_{16}$ message flow example #2.

Table 252 — DynamicallyDefineDataIdentifier positive response DDDDI $F302_{16}$ message flow example #2

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	DynamicallyDefineDataIdentifier Response SID	$6C_{16}$	DDDIPR
#2	definitionMode = defineByIdentifier	01_{16}	DBID
#3	dynamicallyDefinedDataIdentifier [byte#1] (MSB)	$F3_{16}$	DDDDI_B1
#4	dynamicallyDefinedDataIdentifier [byte#2] (LSB)	02_{16}	DDDDI_B2

Table 253 specifies the ReadDataByIdentifier request DDDDI $F302_{16}$ message flow example #2.

Table 253 — ReadDataByIdentifier request DDDDI $F302_{16}$ message flow example #2

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadDataByIdentifier Request SID	22_{16}	RDBI
#2	dataIdentifier [byte#1] (MSB)	$F3_{16}$	DID_B1
#3	dataIdentifier [byte#2] (LSB)	02_{16}	DID_B2

Table 254 specifies the ReadDataByIdentifier positive response DDDDI $F302_{16}$ message flow example #2.

Table 254 — ReadDataByIdentifier positive response DDDDI $F302_{16}$ message flow example #2

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadDataByIdentifier Response SID	62_{16}	RDBIPR
#2	dataIdentifier [byte#1] (MSB)	$F3_{16}$	DID_B1
#3	dataIdentifier [byte#2] (LSB)	02_{16}	DID_B2
#4	dataRecord [data#1] = ECT	$A6_{16}$	DREC_DATA_1
#5	dataRecord [data#2] = RPM	07_{16}	DREC_DATA_2
#6	dataRecord [data#3] = RPM	50_{16}	DREC_DATA_3
#7	dataRecord [data#4] = IAC	82_{16}	DREC_DATA_4
#8	dataRecord [data#5] = KS	91_{16}	DREC_DATA_5

11.6.5.4 Example #3: DynamicallyDefineDataIdentifier, SubFunction = defineByMemoryAddress

The example in Table 255 will build up a dynamic data identifier (DDDDI $F302_{16}$) containing engine coolant temperature (from memory block starting at memory address 21091969_{16}), engine speed (from memory block starting at memory address $2109196B_{16}$), and knock sensor (from memory block starting at memory address 13101995_{16}).

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

Table 255 — DynamicallyDefineDataIdentifier request DDDDI $F302_{16}$ message flow example #3

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	DynamicallyDefineDataIdentifier Request SID	$2C_{16}$	DDDI
#2	SubFunction = defineByMemoryAddress, suppressPosRspMsgIndicationBit = FALSE	02_{16}	DBMA
#3	dynamicallyDefinedDataIdentifier [byte#1] (MSB)	$F3_{16}$	DDDDI_B1
#4	dynamicallyDefinedDataIdentifier [byte#2] (LSB)	02_{16}	DDDDI_B2
#5	addressAndLengthFormatIdentifier	14_{16}	ALFID
#6	memoryAddress#1 [byte#1] (MSB) - Engine Coolant Temperature	21_{16}	MA_1_B1
#7	memoryAddress#1 [byte#2]	09_{16}	MA_1_B2
#8	memoryAddress#1 [byte#3]	19_{16}	MA_1_B3
#9	memoryAddress#1 [byte#4]	69_{16}	MA_1_B4
#10	memorySize#1	01_{16}	MS#1
#11	memoryAddress#2 [byte#1] (MSB) - Engine Speed	21_{16}	MA_2_B1
#12	memoryAddress#2 [byte#2]	09_{16}	MA_2_B2
#13	memoryAddress#2 [byte#3]	19_{16}	MA_2_B3
#14	memoryAddress#2 [byte#4]	$6B_{16}$	MA_2_B4
#15	memorySize#2	02_{16}	MS#2
#16	memoryAddress#3 [byte#1] (MSB) - Knock Sensor	13_{16}	MA_3_B1
#17	memoryAddress#3 [byte#2]	10_{16}	MA_3_B2
#18	memoryAddress#3 [byte#3]	19_{16}	MA_3_B3
#19	memoryAddress#3 [byte#4]	95_{16}	MA_3_B4
#20	memorySize#3	01_{16}	MS#3

Table 256 specifies the DynamicallyDefineDataIdentifier positive response DDDDI $F302_{16}$ message flow example #3.

Table 256 — DynamicallyDefineDataIdentifier positive response DDDDI $F302_{16}$ message flow example #3

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	DynamicallyDefineDataIdentifier Response SID	$6C_{16}$	DDDIPR
#2	definitionMode = defineByMemoryAddress	02_{16}	DBMA
#3	dynamicallyDefinedDataIdentifier [byte#1] (MSB)	$F3_{16}$	DDDDI_B1
#4	dynamicallyDefinedDataIdentifier [byte#2] (LSB)	02_{16}	DDDDI_B2

Table 257 specifies the ReadDataByIdentifier request DDDDI $F302_{16}$ message flow example #3.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 257 — ReadDataByIdentifier request DDDDI $F302_{16}$ message flow example #3

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadDataByIdentifier Request SID	22_{16}	RDBI
#2	dataIdentifier [byte#1] (MSB)	$F3_{16}$	DID_B1
#3	dataIdentifier [byte#2] (LSB)	02_{16}	DID_B2

Table 258 specifies the ReadDataByIdentifier positive response DDDDI $F302_{16}$ message flow example #3.

Table 258 — ReadDataByIdentifier positive response DDDDI $F302_{16}$ message flow example #3

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadDataByIdentifier Response SID	62_{16}	RDBIPR
#2	dataIdentifier [byte#1] (MSB)	$F3_{16}$	DID_B1
#3	dataIdentifier [byte#2] (LSB)	02_{16}	DID_B2
#4	dataRecord [data#1] = ECT	$A6_{16}$	DREC_DATA_1
#5	dataRecord [data#2] = RPM	07_{16}	DREC_DATA_2
#6	dataRecord [data#3] = RPM	50_{16}	DREC_DATA_3
#7	dataRecord [data#4] = KS	91_{16}	DREC_DATA_4

11.6.5.5 Example #4: DynamicallyDefineDataIdentifier, SubFunction = defineByIdentifier

The example in Table 259 will build up a dynamic data identifier (DDDDI $F2E7_{16}$) containing engine coolant temperature (from data record $010A_{16}$), engine speed (from data record $010A_{16}$), and knock sensor (from data record $050B_{16}$).

The value for the dynamic data identifier is chosen out of the range that can be used to request data periodically. Following the definition of the dynamic data identifier, the client requests the data identifier to be sent periodically (fast rate).

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

Table 259 — DynamicallyDefineDataIdentifier request DDDDI F2E7$_{16}$ message flow example #4

Message direction	client → server		
Message type	Request		
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	DynamicallyDefineDataIdentifier Request SID	2C$_{16}$	DDDI
#2	SubFunction = defineByIdentifier, suppressPosRspMsgIndicationBit = FALSE	01$_{16}$	DBID
#3	dynamicallyDefinedDataIdentifier [byte#1] (MSB)	F2$_{16}$	DDDDI_B1
#4	dynamicallyDefinedDataIdentifier [byte#2] (LSB)	E7$_{16}$	DDDDI_B2
#5	sourceDataIdentifier#1 [byte#1] (MSB)	01$_{16}$	SDI_B1
#6	sourceDataIdentifier#1 [byte#2] (LSB)	0A$_{16}$	SDI_B2
#7	positionInSourceDataRecord#1 - Engine Coolant Temperature	02$_{16}$	PISDR
#8	memorySize#1	01$_{16}$	MS
#9	sourceDataIdentifier#2 [byte#1] (MSB)	01$_{16}$	SDI_B1
#10	sourceDataIdentifier#2 [byte#2] (LSB)	0A$_{16}$	SDI_B2
#11	positionInSourceDataRecord#2 - Engine Speed	04$_{16}$	PISDR
#12	memorySize#2	02$_{16}$	MS
#13	sourceDataIdentifier#3 [byte#1] (MSB)	05$_{16}$	SDI_B1
#14	sourceDataIdentifier#3 [byte#2] (LSB)	0B$_{16}$	SDI_B2
#15	positionInSourceDataRecord#3 - Knock Sensor	02$_{16}$	PISDR
#16	memorySize#3	01$_{16}$	MS

Table 260 specifies the DynamicallyDefineDataIdentifier positive response DDDDI F2E7$_{16}$ message flow example #4.

Table 260 — DynamicallyDefineDataIdentifier positive response DDDDI F2E7$_{16}$ message flow example #4

Message direction	server → client		
Message type	Response		
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	DynamicallyDefineDataIdentifier Response SID	6C$_{16}$	DDDIPR
#2	definitionMode = defineByIdentifier	01$_{16}$	DBID
#3	dynamicallyDefinedDataIdentifier [byte#1] (MSB)	F2$_{16}$	DDDDI_B1
#4	dynamicallyDefinedDataIdentifier [byte#2] (LSB)	E7$_{16}$	DDDDI_B2

Table 261 specifies the ReadDataByPeriodicIdentifier request DDDDI F2E7$_{16}$ message flow example #4.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 261 — ReadDataByPeriodicIdentifier request DDDDI F2E7$_{16}$ message flow example #4

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadDataByPeriodicIdentifier Request SID	2A$_{16}$	RDBPI
#2	transmissionMode = sendAtFastRate	04$_{16}$	TM
#3	PeriodicDataIdentifier	E7$_{16}$	PDID

Table 262 specifies the ReadDataByPeriodicIdentifier initial positive message flow example #4.

Table 262 — ReadDataByPeriodicIdentifier initial positive message flow example #4

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadDataByPeriodicIdentifier Response SID	6A$_{16}$	RDBPIPR

Table 263 and Table 264 define the ReadDataByPeriodicIdentifier periodic data response #1 DDDDI F2E7$_{16}$ message flow example #4.

Table 263 — ReadDataByPeriodicIdentifier periodic data response #1 DDDDI F2E7$_{16}$ message flow example #4

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	PeriodicDataIdentifier	E7$_{16}$	PDID
#2	dataRecord [data#1] = ECT	A6$_{16}$	DREC_DATA_1
#3	dataRecord [data#2] = RPM	07$_{16}$	DREC_DATA_2
#4	dataRecord [data#3] = RPM	50$_{16}$	DREC_DATA_3
#5	dataRecord [data#4] = KS	91$_{16}$	DREC_DATA_4

NOTE Multiple response messages with different Byte values are not shown in this example.

Table 264 — ReadDataByPeriodicIdentifier periodic data response #n DDDDI F2E7$_{16}$ message flow example #4

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	periodicDataIdentifier	E7$_{16}$	PDID
#2	dataRecord [data#1] = ECT	A6$_{16}$	DREC_DATA_1
#3	dataRecord [data#2] = RPM	07$_{16}$	DREC_DATA_2
#4	dataRecord [data#3] = RPM	55$_{16}$	DREC_DATA_3
#5	dataRecord [data#4] = KS	98$_{16}$	DREC_DATA_4

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

11.6.5.6 Example #5: DynamicallyDefineDataIdentifier, SubFunction = clearDynamicallyDefined-DataIdentifier

The example in Table 265 demonstrates the clearing of a dynamicallyDefinedDataIdentifier, and assumes that DDDDI $F303_{16}$ exists at the time of the request.

Table 265 — DynamicallyDefineDataIdentifier request clear DDDDI $F303_{16}$ message flow example #5

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	DynamicallyDefineDataIdentifier Request SID	$2C_{16}$	DDDI
#2	SubFunction = clearDynamicallyDefinedDataIdentifier, suppressPosRspMsgIndicationBit = FALSE	03_{16}	CDDDI
#3	dynamicallyDefinedDataIdentifier [byte#1] (MSB)	$F3_{16}$	DDDDI_B1
#4	dynamicallyDefinedDataIdentifier [byte#2] (LSB)	03_{16}	DDDDI_B2

Table 266 specifies the DynamicallyDefineDataIdentifier positive response clear DDDDI $F303_{16}$ message flow example #5.

Table 266 — DynamicallyDefineDataIdentifier positive response clear DDDDI $F303_{16}$ message flow example #5

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	DynamicallyDefineDataIdentifier Response SID	$6C_{16}$	DDDIPR
#2	definitionMode = clearDynamicallyDefinedDataIdentifier	03_{16}	CDDDI
#3	dynamicallyDefinedDataIdentifier [byte#1] (MSB)	$F3_{16}$	DDDDI_B1
#4	dynamicallyDefinedDataIdentifier [byte#2] (LSB)	03_{16}	DDDDI_B2

11.6.5.7 Example #6: DynamicallyDefineDataIdentifier, concatenation of definitions (defineByIdentifier/defineByAddress)

This example will build up a dynamic data identifier (DDDI $F301_{16}$) using the two definition types. The following list shows the order of the data in the dynamically defined data identifier (implicit order of request messages to define the dynamic data identifier):

— 1st portion: engine oil temperature and ambient air temperature referecend by 2-byte DIDs (defineByIdentifier),

— 2nd portion: engine coolant temperature and engine speed referenced by memory addresses,

— 3rd portion: engine oil level referenced by 2-byte DID.

11.6.5.7.1 Step #1: DynamicallyDefineDataIdentifier, SubFunction = defineByIdentifier (1 st portion)

Table 267 specifies the DynamicallyDefineDataIdentifier request DDDI $F301_{16}$ message flow example #6 definition of 1st portion (defineByIdentifier).

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 267 — DynamicallyDefineDataIdentifier request DDDI F301$_{16}$ message flow example #6 definition of 1st portion (defineByIdentifier)

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	DynamicallyDefineDataIdentifier Request SID	2C$_{16}$	DDDI
#2	SubFunction = defineByIdentifier, suppressPosRspMsgIndicationBit = FALSE	01$_{16}$	DBID
#3	dynamicallyDefinedDataIdentifier [byte#1] (MSB)	F3$_{16}$	DDDDI_B1
#4	dynamicallyDefinedDataIdentifier [byte#2] (LSB)	01$_{16}$	DDDDI_B2
#5	sourceDataIdentifier#1 [byte#1] (MSB) - Engine Oil Temperature	12$_{16}$	SDI_B1
#6	sourceDataIdentifier#1 [byte#2]	34$_{16}$	SDI_B2
#7	positionInSourceDataRecord#1	01$_{16}$	PISDR#1
#8	memorySize#1	02$_{16}$	MS#1
#9	sourceDataIdentifier#2 [byte#1] (MSB) - Ambient Air Temperature	56$_{16}$	SDI_B1
#10	sourceDataIdentifier#2 [byte#2] (LSB)	78$_{16}$	SDI_B2
#11	positionInSourceDataRecord#2	01$_{16}$	PISDR#2
#12	memorySize#2	01$_{16}$	MS#2

Table 268 specifies the DynamicallyDefineDataIdentifier positive response DDDI F301$_{16}$ message flow example #6 definition of first portion (defineByIdentifier).

Table 268 — DynamicallyDefineDataIdentifier positive response DDDI F301$_{16}$ message flow example #6 definition of first portion (defineByIdentifier)

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	DynamicallyDefineDataIdentifier Response SID	6C$_{16}$	DDDIPR
#2	definitionMode = defineByIdentifier	01$_{16}$	DBID
#3	dynamicallyDefinedDataIdentifier [byte#1] (MSB)	F3$_{16}$	DDDDI_B1
#4	dynamicallyDefinedDataIdentifier [byte#2] (LSB)	01$_{16}$	DDDDI_B2

11.6.5.7.2 Step #2: DynamicallyDefineDataIdentifier, SubFunction = defineByMemoryAddress (2nd portion)

Table 269 specifies the DynamicallyDefineDataIdentifier request DDDDI F301$_{16}$ message flow example #6 definition of 2nd portion (defineByMemoryAddress).

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 269 — DynamicallyDefineDataIdentifier request DDDDI $F301_{16}$ message flow example #6 definition of 2nd portion (defineByMemoryAddress)

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	DynamicallyDefineDataIdentifier Request SID	$2C_{16}$	DDDI
#2	SubFunction = defineByMemoryAddress, suppressPosRspMsgIndicationBit = FALSE	02_{16}	DBMA
#3	dynamicallyDefinedDataIdentifier [byte#1] (MSB)	$F3_{16}$	DDDDI_B1
#4	dynamicallyDefinedDataIdentifier [byte#2] (LSB)	01_{16}	DDDDI_B2
#5	addressAndLengthFormatIdentifier	14_{16}	ALFID
#6	memoryAddress#1 [byte#1] (MSB) - Engine Coolant Temperature	21_{16}	MA_B1#1
#7	memoryAddress#1 [byte#2]	09_{16}	MA_B2#1
#8	memoryAddress#1 [byte#3]	19_{16}	MA_B3#1
#9	memoryAddress#1 [byte#4]	69_{16}	MA_B4#1
#10	memorySize#1	01_{16}	MS#1
#11	memoryAddress#2 [byte#1] (MSB) - Engine Speed	21_{16}	MA_B1#2
#12	memoryAddress#2 [byte#2]	09_{16}	MA_B2#2
#13	memoryAddress#2 [byte#3]	19_{16}	MA_B3#2
#14	memoryAddress#2 [byte#4]	$6B_{16}$	MA_B4#2
#15	memorySize#2	02_{16}	MS#2

Table 270 specifies the DynamicallyDefineDataIdentifier positive response DDDI $F301_{16}$ message flow example #6.

Table 270 — DynamicallyDefineDataIdentifier positive response DDDI $F301_{16}$ message flow example #6

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	DynamicallyDefineDataIdentifier Response SID	$6C_{16}$	DDDIPR
#2	definitionMode = defineByMemoryAddress	02_{16}	DBMA
#3	dynamicallyDefinedDataIdentifier [byte#1] (MSB)	$F3_{16}$	DDDDI_B1
#4	dynamicallyDefinedDataIdentifier [byte#2] (LSB)	01_{16}	DDDDI_B2

11.6.5.7.3 Step #3: DynamicallyDefineDataIdentifier, SubFunction = defineByIdentifier (3 rd portion)

Table 271 specifies the DynamicallyDefineDataIdentifier request DDDI $F301_{16}$ message flow example #6 definition of 3rd portion (defineByIdentifier).

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 271 — DynamicallyDefineDataIdentifier request DDDI F301$_{16}$ message flow example #6 definition of 3rd portion (defineByIdentifier)

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	DynamicallyDefineDataIdentifier Request SID	2C$_{16}$	DDDI
#2	SubFunction = defineByIdentifier, suppressPosRspMsgIndicationBit = FALSE	01$_{16}$	DBID
#3	dynamicallyDefinedDataIdentifier [byte#1] (MSB)	F3$_{16}$	DDDDI_B1
#4	dynamicallyDefinedDataIdentifier [byte#2] (LSB)	01$_{16}$	DDDDI_B2
#5	sourceDataIdentifier#1 [byte#1] (MSB) - Engine Oil Level	9A$_{16}$	SDI_B1
#6	sourceDataIdentifier#1 [byte#2]	BC$_{16}$	SDI_B2
#7	positionInSourceDataRecord#1	01$_{16}$	PISDR#3
#8	memorySize#1	04$_{16}$	MS#3

Table 272 specifies the DynamicallyDefineDataIdentifier positive response DDDI F301$_{16}$ message flow example #6.

Table 272 — DynamicallyDefineDataIdentifier positive response DDDI F301$_{16}$ message flow example #6

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	DynamicallyDefineDataIdentifier Response SID	6C$_{16}$	DDDIPR
#2	definitionMode = defineByIdentifier	01$_{16}$	DBID
#3	dynamicallyDefinedDataIdentifier [byte#1] (MSB)	F3$_{16}$	DDDDI_B1
#4	dynamicallyDefinedDataIdentifier [byte#2] (LSB)	01$_{16}$	DDDDI_B2

11.6.5.7.4 Step #4: ReadDataByIdentifier - dataIdentifier = DDDDI F301$_{16}$

Table 273 specifies the ReadDataByIdentifier request DDDDI F301$_{16}$ message flow example #6.

Table 273 — ReadDataByIdentifier request DDDDI F301$_{16}$ message flow example #6

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadDataByIdentifier Request SID	22$_{16}$	RDBI
#2	dataIdentifier [byte#1] (MSB)	F3$_{16}$	DID_B1
#3	dataIdentifier [byte#2] (LSB)	01$_{16}$	DID_B2

Table 274 specifies the ReadDataByIdentifier positive response DDDDI F301$_{16}$ message flow example #6.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

Table 274 — ReadDataByIdentifier positive response DDDDI $F301_{16}$ message flow example #6

Message direction		server → client	
Message type		Response	
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	ReadDataByIdentifier Response SID	62_{16}	RDBIPR
#2	dataIdentifier [byte#1] (MSB)	$F3_{16}$	DID_B1
#3	dataIdentifier [byte#2] (LSB)	01_{16}	DID_B2
#4	dataRecord [data#1] = EOT (MSB)	$4C_{16}$	DREC_DATA_1
#5	dataRecord [data#2] = EOT	36_{16}	DREC_DATA_2
#6	dataRecord [data#3] = AAT	$4D_{16}$	DREC_DATA_3
#7	dataRecord [data#4] = ECT	$A6_{16}$	DREC_DATA_4
#8	dataRecord [data#5] = RPM	07_{16}	DREC_DATA_5
#9	dataRecord [data#6] = RPM	50_{16}	DREC_DATA_6
#10	dataRecord [data#7] = EOL (MSB)	49_{16}	DREC_DATA_7
#11	dataRecord [data#8] = EOL	21_{16}	DREC_DATA_8
#12	dataRecord [data#9] = EOL	00_{16}	DREC_DATA_9
#13	dataRecord [data#10] = EOL	17_{16}	DREC_DATA_10

11.6.5.7.5 Step #5: DynamicallyDefineDataIdentifier - clear definition of DDDDI $F301_{16}$

Table 275 specifies the DynamicallyDefineDataIdentifier request clear DDDDI $F301_{16}$ message flow example #6.

Table 275 — DynamicallyDefineDataIdentifier request clear DDDDI $F301_{16}$ message flow example #6

Message direction		client → server	
Message type		Request	
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	DynamicallyDefineDataIdentifier Request SID	$2C_{16}$	DDDI
#2	SubFunction = clearDynamicallyDefinedDataIdentifier, suppressPosRspMsgIndicationBit = FALSE	03_{16}	CDDDI
#3	dynamicallyDefinedDataIdentifier [byte#1] (MSB)	$F3_{16}$	DDDDI_B1
#4	dynamicallyDefinedDataIdentifier [byte#2] (LSB)	01_{16}	DDDDI_B2

Table 276 specifies the DynamicallyDefineDataIdentifier positive response clear DDDDI $F301_{16}$ message flow example #6.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 276 — DynamicallyDefineDataIdentifier positive response clear DDDDI $F301_{16}$ message flow example #6

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	DynamicallyDefineDataIdentifier Response SID	$6C_{16}$	DDDIPR
#2	definitionMode = clearDynamicallyDefinedDataIdentifier	03_{16}	CDDDI
#3	dynamicallyDefinedDataIdentifier [byte#1] (MSB)	$F3_{16}$	DDDDI_B1
#4	dynamicallyDefinedDataIdentifier [byte#2] (LSB)	01_{16}	DDDDI_B2

11.7 WriteDataByIdentifier ($2E_{16}$) service

11.7.1 Service description

The WriteDataByIdentifier service allows the client to write information into the server at an internal location specified by the provided data identifier.

The WriteDataByIdentifier service is used by the client to write a dataRecord to a server. The data is identified by a dataIdentifier and may or may not be secured.

Dynamically defined dataIdentifer(s) shall not be used with this service. It is the vehicle manufacturer's responsibility that the server conditions are met when performing this service. Possible uses for this service are:

— programming configuration information into server (e.g. VIN number);

— clearing non-volatile memory;

— resetting learned values; and

— setting option content.

NOTE The server can restrict or prohibit write access to certain dataIdentifier values (as defined by the system supplier/vehicle manufacturer for read-only identifiers, etc.).

IMPORTANT — The server and the client shall meet the request and response message behaviour as specified in 8.7.

11.7.2 Request message

11.7.2.1 Request message definition

Table 277 specifies the request message.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

Table 277 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	WriteDataByIdentifier Request SID	M	$2E_{16}$	WDBI
	dataIdentifier[] = [			DID_
#2	byte#1 (MSB)	M	00_{16} to FF_{16}	HB
#3	byte#2]	M	00_{16} to FF_{16}	LB
	dataRecord[] = [			DREC_
#4	data#1	M	00_{16} to FF_{16}	DATA_1
:	:	:	:	:
#m+3	data#m]	U	00_{16} to FF_{16}	DATA_m

11.7.2.2 Request message SubFunction parameter $Level (LEV_) definition

This service does not use a SubFunction parameter.

11.7.2.3 Request message data-parameter definition

Table 278 specifies the data-parameters of the request message.

Table 278 — Request message data-parameter definition

Definition
dataIdentifier This parameter identifies the server data record that the client is requesting to write to (see C.1 for detailed parameter definition).
dataRecord This parameter provides the data record associated with the dataIdentifier that the client is requesting to write to.

11.7.3 Positive response message

11.7.3.1 Positive response message definition

Table 279 specifies the positive response message.

Table 279 — Positive response message definition

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	WriteDataByIdentifier Response SID	M	$6E_{16}$	WDBIPR
	dataIdentifier[] = [			DID_
#2	byte#1 (MSB)	M	00_{16} to FF_{16}	HB
#3	byte#2]	M	00_{16} to FF_{16}	LB

11.7.3.2 Positive response message data-parameter definition

Table 280 specifies the data-parameters of the positive response message.

Table 280 — Response message data-parameter definition

Definition
dataIdentifier This parameter is an echo of the data-parameter dataIdentifier from the request message.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

11.7.4 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 281. The listed negative responses shall be used if the error scenario applies to the server.

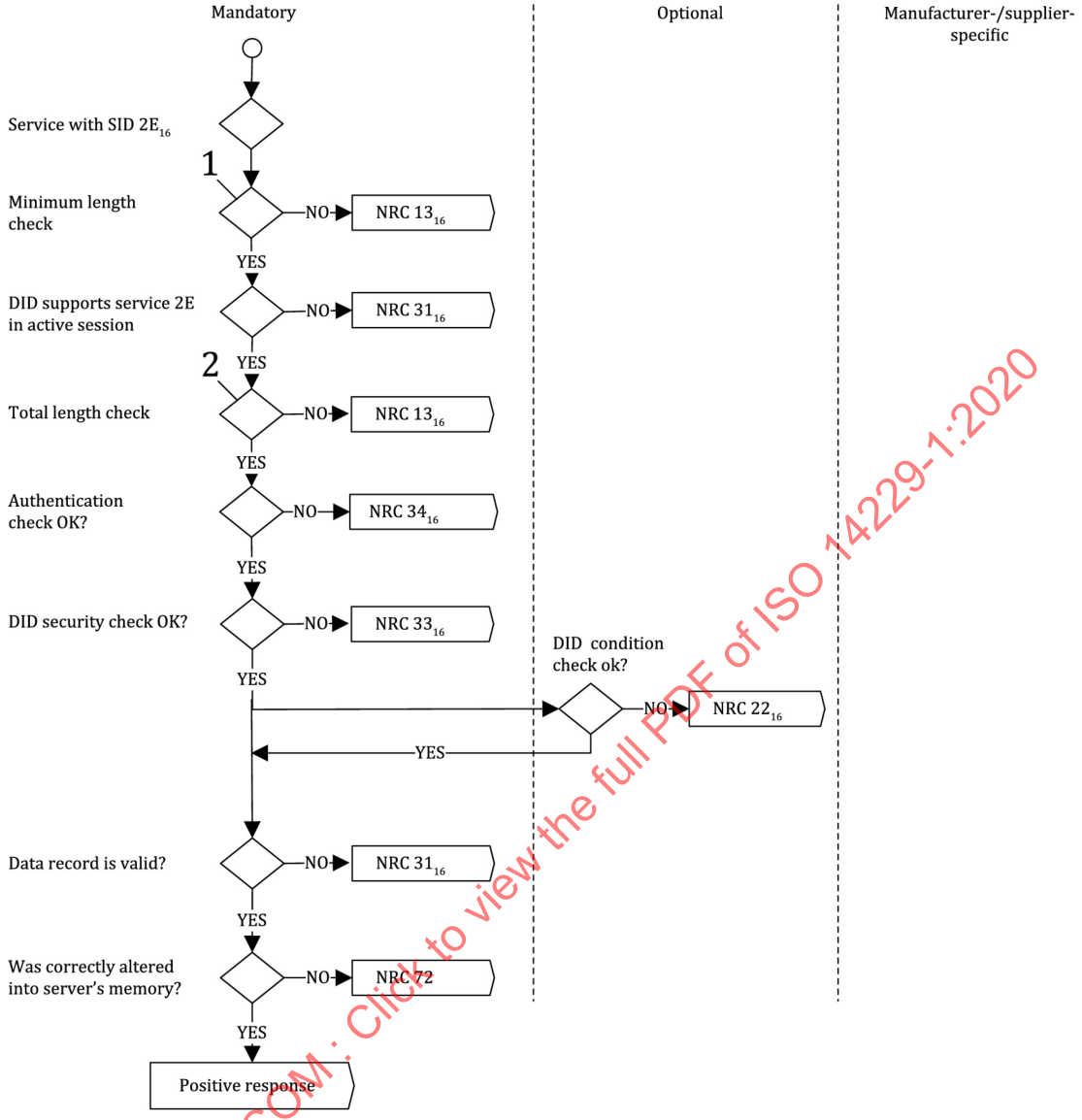
Table 281 — Supported negative response codes

NRC	Description	Mnemonic
13_{16}	**incorrectMessageLengthOrInvalidFormat** This NRC shall be sent if the length of the message is wrong.	IMLOIF
22_{16}	**conditionsNotCorrect** This NRC shall be sent if the operating conditions of the server are not met to perform the required action.	CNC
31_{16}	**requestOutOfRange** This NRC shall be sent if: — the dataIdentifier in the request message is not supported in the server or the dataIdentifier is supported for read only purpose (via ReadDataByIdentifier service); — any data transmitted in the request message after the dataIdentifier is invalid (if applicable to the node).	ROOR
33_{16}	**securityAccessDenied** This NRC shall be sent if the dataIdentifier, which reference a specific address, is secured and the server is not in an unlocked state.	SAD
72_{16}	**generalProgrammingFailure** This NRC shall be returned if the server detects an error when writing to a memory location.	GPF

The evaluation sequence is documented in Figure 26.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved



Key

1 minimum length is 4 byte (SI + DID + DREC)

2 total length is 3 byte (SI + DID) + nth byte DREC

Figure 26 — NRC handling for WriteDataByIdentifier service

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

11.7.5 Message flow example WriteDataByIdentifier

11.7.5.1 Assumptions

This subclause specifies the conditions to be fulfilled for the example to perform a WriteDataByIdentifier service.

The service in this example is not limited by any restriction of the server. This example demonstrates VIN programming via a two byte dataIdentifier $F190_{16}$.

11.7.5.2 Example #1: write dataIdentifier $F190_{16}$ (VIN)

Table 282 specifies the WriteDataByIdentifier request message flow example #1.

© ISO 2020 – All rights reserved

Table 282 — WriteDataByIdentifier request message flow example #1

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	WriteDataByIdentifier Request SID	$2E_{16}$	WDBI
#2	dataIdentifier [byte#1] (MSB)	$F1_{16}$	DID_B1
#3	dataIdentifier [byte#2]	90_{16}	DID_B2
#4	dataRecord [data#1] = VIN Digit 1 = “W”	57_{16}	DREC_DATA1
#5	dataRecord [data#2] = VIN Digit 2 = “0”	30_{16}	DREC_DATA2
#6	dataRecord [data#3] = VIN Digit 3 = “L”	$4C_{16}$	DREC_DATA3
#7	dataRecord [data#4] = VIN Digit 4 = “0”	30_{16}	DREC_DATA4
#8	dataRecord [data#5] = VIN Digit 5 = “0”	30_{16}	DREC_DATA5
#9	dataRecord [data#6] = VIN Digit 6 = “0”	30_{16}	DREC_DATA6
#10	dataRecord [data#7] = VIN Digit 7 = “0”	30_{16}	DREC_DATA7
#11	dataRecord [data#8] = VIN Digit 8 = “4”	34_{16}	DREC_DATA8
#12	dataRecord [data#9] = VIN Digit 9 = “3”	33_{16}	DREC_DATA9
#13	dataRecord [data#10] = VIN Digit 10 = “M”	$4D_{16}$	DREC_DATA10
#14	dataRecord [data#11] = VIN Digit 11 = “B”	42_{16}	DREC_DATA11
#15	dataRecord [data#12] = VIN Digit 12 = “5”	35_{16}	DREC_DATA12
#16	dataRecord [data#13] = VIN Digit 13 = “4”	34_{16}	DREC_DATA13
#17	dataRecord [data#14] = VIN Digit 14 = “1”	31_{16}	DREC_DATA14
#18	dataRecord [data#15] = VIN Digit 15 = “3”	33_{16}	DREC_DATA15
#19	dataRecord [data#16] = VIN Digit 16 = “2”	32_{16}	DREC_DATA16
#20	dataRecord [data#17] = VIN Digit 17 = “6”	36_{16}	DREC_DATA17

Table 283 specifies the WriteDataByIdentifier positive response message flow example #1.

Table 283 — WriteDataByIdentifier positive response message flow example #1

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	WriteDataByIdentifier Response SID	$6E_{16}$	WDBIPR
#2	dataIdentifier [byte#1] (MSB)	$F1_{16}$	DID_B1
#3	dataIdentifier [byte#2] (LSB)	90_{16}	DID_B2

11.8 WriteMemoryByAddress ($3D_{16}$) service

11.8.1 Service description

The WriteMemoryByAddress service allows the client to write information into the server at one or more contiguous memory locations.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

The WriteMemoryByAddress request message writes information specified by the parameter dataRecord[] into the server at memory locations specified by parameters memoryAddress and memorySize. The number of bytes used for the memoryAddress and memorySize parameter is defined by addressAndLengthFormatIdentifier (low and high nibble). It is also possible to use a fixed addressAndLengthFormatIdentifier and unused bytes within the memoryAddress or memorySize parameter are padded with the value 00_{16} in the higher range address locations.

The format and definition of the dataRecord shall be vehicle manufacturer specific and may or may not be secured. It is the vehicle manufacturer's responsibility to assure that the server conditions are met when performing this service. Possible uses for this service are:

— clear non-volatile memory; and

— change calibration values.

IMPORTANT — The server and the client shall meet the request and response message behaviour as specified in 8.7.

11.8.2 Request message

11.8.2.1 Request message definition

Table 284 specifies the request message definition.

Table 284 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	WriteMemoryByAddress Request SID	M	$3D_{16}$	WMBA
#2	addressAndLengthFormatIdentifier	M	00_{16} to FF_{16}	ALFID
#3 : #m+2	memoryAddress[] = [byte#1 (MSB) : byte#m]	M : C1	00_{16} to FF_{16} : 00_{16} to FF_{16}	MA_ B1 : Bm
#n-r-2-(k-1) : #n-r-2	memorySize[] = [byte#1 (MSB) : byte#k]	M : C2	00_{16} to FF_{16} : 00_{16} to FF_{16}	MS_ B1 : Bk
#n-(r-1) : #n	dataRecord[] = [data#1 : data#r]	M : U	00_{16} to FF_{16} : 00_{16} to FF_{16}	DREC_ DATA_1 : DATA_r
C1: The presence of this parameter depends on address length information parameter of the addressAndLengthFormatIdentifier. C2: The presence of this parameter depends on the memory size length information of the addressAndLengthFormatIdentifier.				

11.8.2.2 Request message SubFunction parameter $Level (LEV_) definition

This service does not use a SubFunction parameter.

11.8.2.3 Request message data-parameter definition

Table 285 specifies the data-parameters of the request message.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 285 — Request message data-parameter definition

Definition
addressAndLengthFormatIdentifier This parameter is a one Byte value with each nibble encoded separately (see Table H.1 or example values): bit 7 - 4: Length (number of bytes) of the memorySize parameter bit 3 - 0: Length (number of bytes) of the memoryAddress parameter
memoryAddress The parameter memoryAddress is the starting address of server memory to which data is to be written. The number of bytes used for this address is defined by the low nibble (bit 3 - 0) of the addressAndLengthFormatIdentifier. Byte#m in the memoryAddress parameter is always the least significant byte of the address being referenced in the server. The most significant byte(s) of the address can be used as a memory identifier. An example of the use of a memory identifier would be a dual processor server with 16 bit addressing and memory address overlap (when a given address is valid for either processor but yields a different physical memory device or internal and external flash is used). In this case, an otherwise unused byte within the memoryAddress parameter can be specified as a memory identifier used to select the desired memory device. Usage of this functionality shall be as defined by vehicle manufacturer/system supplier.
memorySize The parameter memorySize in the WriteMemoryByAddress request message specifies the number of bytes to be written starting at the address specified by memoryAddress in the server's memory. The number of bytes used for this size is defined by the high nibble (bit 7 - 4) of the addressAndLengthFormatIdentifier.
dataRecord This parameter provides the data that the client is actually attempting to write into the server memory addresses within the interval {MA_{16}, (MA_{16} + MS_{16} - 01_{16})}.

11.8.3 Positive response message

11.8.3.1 Positive response message definition

Table 286 specifies the positive response message.

Table 286 — Positive response message definition

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	WriteMemoryByAddress Response SID	M	$7D_{16}$	WMBAPR
#2	addressAndLengthFormatIdentifier	M	00_{16} to FF_{16}	ALFID
#3 : #(m-1)+3	memoryAddress[] = [byte#1 (MSB) : byte#m]	M : C1	00_{16} to FF_{16} : 00_{16} to FF_{16}	MA_ B1 : Bm
#n-(k-1) : #n	memorySize[] = [byte#1 (MSB) : byte#k]	M : C2	00_{16} to FF_{16} : 00_{16} to FF_{16}	MS_ B1 : Bk
C1: The presence of this parameter depends on address length information parameter of the addressAndLengthFormatIdentifier. C2: The presence of this parameter depends on the memory size length information of the addressAndLengthFormatIdentifier.				

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

11.8.3.2 Positive response message data-parameter definition

Table 287 specifies the data-parameters of the positive response message.

Table 287 — Response message data-parameter definition

Definition
addressAndLengthFormatIdentifier This parameter is an echo of the addressAndLengthFormatIdentifier from the request message.
memoryAddress This parameter is an echo of the memoryAddress from the request message.
memorySize This parameter is an echo of the memorySize from the request message.

11.8.4 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 288. The listed negative responses shall be used if the error scenario applies to the server.

Table 288 — Supported negative response codes

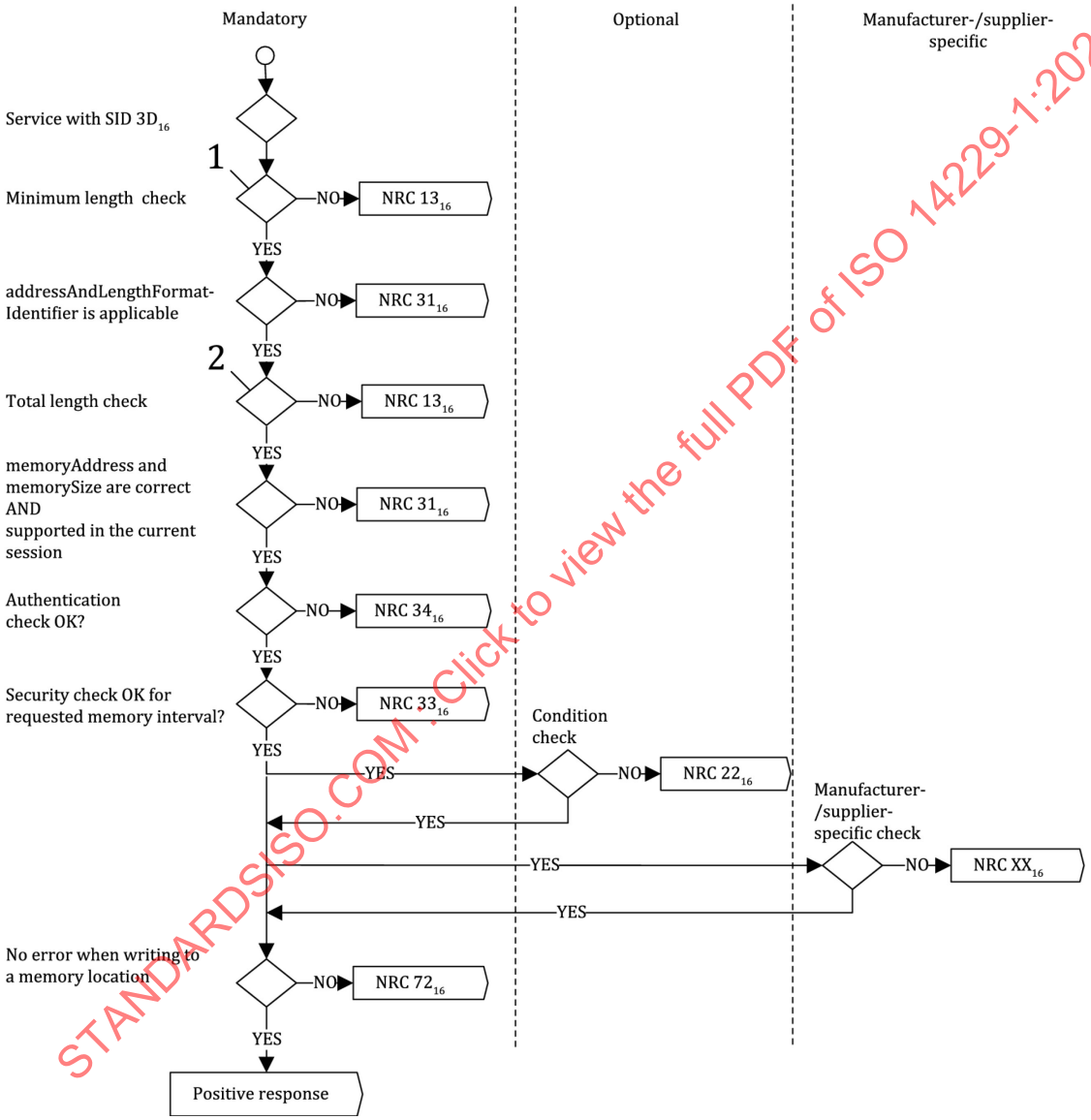
NRC	Description	Mnemonic
13_{16}	**incorrectMessageLengthOrInvalidFormat** This NRC shall be sent if the length of the message is wrong.	IMLOIF
22_{16}	**conditionsNotCorrect** This NRC shall be sent if the operating conditions of the server are not met to perform the required action.	CNC
31_{16}	**requestOutOfRange** This NRC shall be sent if: — any memory address within the interval [MA_{16}, (MA_{16} + MS_{16} -1_{16})] is invalid; — any memory address within the interval [MA_{16}, (MA_{16} + MS_{16} -1_{16})] is restricted; — the memorySize parameter value in the request message is not supported by the server; — the specified addressAndLengthFormatIdentifier is not valid; — the memorySize parameter value in the request message is zero.	ROOR
33_{16}	**securityAccessDenied** This NRC shall be sent if any memory address within the interval [MA_{16}, (MA_{16} + MS_{16} - 1_{16})] is secure and the server is locked.	SAD

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

34_{16}	**authenticationRequired** This NRC shall be sent if the dataIdentifier is secured and the client has insufficient rights based on its Authentication state.	AR
72_{16}	**generalProgrammingFailure** This NRC shall be returned if the server detects an error when writing to a memory location.	GPF

The evaluation sequence is documented in Figure 27.



STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

Key

1 at least 5 (SI+addressAndLengthFormatIdentifier + min memoryAddress+min memorySize + min dataRecord)

2 1 byte SI + 1 byte addressAndLengthFormatIdentifier + n byte memoryAddress parameter length + n byte memorySize parameter length + n byte dataRecord length

Figure 27 — NRC handling for WriteMemoryByAddress service

© ISO 2020 – All rights reserved

11.8.5 Message flow example WriteMemoryByAddress

11.8.5.1 Assumptions

This subclause specifies the conditions to be fulfilled for the example to perform a WriteMemoryByAddress service. The service in this example is not limited by any restriction of the server.

The following examples demonstrate writing data bytes into server memory for 2-byte, 3-byte, and 4-byte addressing formats, respectively.

11.8.5.2 Example #1: WriteMemoryByAddress, 2-byte (16-bit) addressing

Table 289 specifies the WriteMemoryByAddress request message flow example #1.

Table 289 — WriteMemoryByAddress request message flow example #1

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	WriteMemoryByAddress Request SID	$3D_{16}$	WMBA
#2	addressAndLengthFormatIdentifier	12_{16}	ALFID
#3	memoryAddress [byte#1] (MSB)	20_{16}	MA_B1
#4	memoryAddress [byte#2] (LSB)	48_{16}	MA_B2
#5	memorySize [byte#1]	02_{16}	MS_B1
#6	dataRecord [data#1]	00_{16}	DREC_DATA_1
#7	dataRecord [data#2]	$8C_{16}$	DREC_DATA_2

Table 290 specifies the WriteMemoryByAddress positive response message flow example #1.

Table 290 — WriteMemoryByAddress positive response message flow example #1

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	WriteMemoryByAddress Response SID	$7D_{16}$	WMBAPR
#2	addressAndLengthFormatIdentifier	12_{16}	ALFID
#3	memoryAddress [byte#1] (MSB)	20_{16}	MA_B1
#4	memoryAddress [byte#2] (LSB)	48_{16}	MA_B2
#5	memorySize [byte#1]	02_{16}	MS_B1

11.8.5.3 Example #2: WriteMemoryByAddress, 3-byte (24-bit) addressing

Table 291 specifies the WriteMemoryByAddress request message flow example #2.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 291 — WriteMemoryByAddress request message flow example #2

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	WriteMemoryByAddress Request SID	$3D_{16}$	WMBA
#2	addressAndLengthFormatIdentifier	13_{16}	ALFID
#3	memoryAddress [byte#1]	20_{16}	MA_B1
#4	memoryAddress [byte#2]	48_{16}	MA_B2
#5	memoryAddress [byte#3]	13_{16}	MA_B3
#6	memorySize [byte#1]	03_{16}	MS_B1
#7	dataRecord [data#1]	00_{16}	DREC_DATA_1
#8	dataRecord [data#2]	01_{16}	DREC_DATA_2
#9	dataRecord [data#3]	$8C_{16}$	DREC_DATA_3

Table 292 specifies the WriteMemoryByAddress positive response message flow example #2.

Table 292 — WriteMemoryByAddress positive response message flow example #2

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	WriteMemoryByAddress Response SID	$7D_{16}$	WMBAPR
#2	addressAndLengthFormatIdentifier	13_{16}	ALFID
#3	memoryAddress [byte#1]	20_{16}	MA_B1
#4	memoryAddress [byte#2]	48_{16}	MA_B2
#5	memoryAddress [byte#3]	13_{16}	MA_B3
#6	memorySize [byte#1]	03_{16}	MS_B1

11.8.5.4 Example #3: WriteMemoryByAddress, 4-byte (32-bit) addressing

Table 293 specifies the WriteMemoryByAddress request message flow example #3.

Table 293 — WriteMemoryByAddress request message flow example #3

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	WriteMemoryByAddress Request SID	$3D_{16}$	WMBA
#2	addressAndLengthFormatIdentifier	14_{16}	ALFID
#3	memoryAddress [byte#1] (MSB)	20_{16}	MA_B1
#4	memoryAddress [byte#2]	48_{16}	MA_B2

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

#5	memoryAddress [byte#3]	13_{16}	MA_B3
#6	memoryAddress [byte#4] (LSB)	09_{16}	MA_B4
#7	memorySize [byte#1]	05_{16}	MS_B1
#8	dataRecord [data#1]	00_{16}	DREC_DATA_1
#9	dataRecord [data#2]	01_{16}	DREC_DATA_2
#10	dataRecord [data#3]	$8C_{16}$	DREC_DATA_3
#11	dataRecord [data#4]	09_{16}	DREC_DATA_4
#12	dataRecord [data#5]	AF_{16}	DREC_DATA_5

Table 294 specifies the WriteMemoryByAddress positive response message flow example #3.

Table 294 — WriteMemoryByAddress positive response message flow example #3

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	WriteMemoryByAddress Response SID	$7D_{16}$	WMBAPR
#2	addressAndLengthFormatIdentifier	14_{16}	ALFID
#3	memoryAddress [byte#1] (MSB)	20_{16}	MA_B1
#4	memoryAddress [byte#2]	48_{16}	MA_B2
#5	memoryAddress [byte#3]	13_{16}	MA_B3
#6	memoryAddress [byte#4] (LSB)	09_{16}	MA_B4
#7	memorySize [byte#1]	05_{16}	MS_B1

12 Stored data transmission functional unit

12.1 Overview

Table 295 specifies the stored data transmission functional unit.

Table 295 — Stored data transmission functional unit

Service	Description
ClearDiagnosticInformation	Allows the client to clear diagnostic information from the server (including DTCs, captured data, etc.).
ReadDTCInformation	Allows the client to request diagnostic information from the server (including DTCs, captured data, etc.).

12.2 ClearDiagnosticInformation (14_{16}) service

12.2.1 Service description

The ClearDiagnosticInformation service is used by the client to clear diagnostic information in one or multiple servers' memory.

The server shall send a positive response when the ClearDiagnosticInformation service is completely processed. The server shall send a positive response even if no DTCs are stored. If a server supports multiple copies of DTC status information in memory (e.g. one copy in RAM and one copy in EEPROM)

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

the server shall clear the copy used by the ReadDTCInformation status reporting service. Additional copies, e.g. backup copy in long-term memory, are updated according to the appropriate backup strategy (e.g. in the power-latch phase).

NOTE In case the power-latch phase is disturbed (e.g. a battery disconnection during the power-latch phase) this can cause data inconsistency.

The behaviour of the individual DTC status bits shall be implemented according to the definitions in D.2, Figure D.1 to Figure D.8.

The request message of the client contains one parameter. The parameter groupOfDTC allows the client to clear a group of DTCs (e.g. Powertrain, Body, Chassis, etc.), or a specific DTC. Refer to D.1 for further details. Unless otherwise stated, the server shall clear both emissions-related and non emissions-related DTC information from memory for the requested group.

DTC information reset/cleared via this service includes but is not limited to the following:

— DTC status byte (see ReadDTCInformation service in 12.3),

— captured DTC snapshot data (DTCSnapshotData, see ReadDTCInformation service in 12.3),

— captured DTC extended data (DTCExtendedData, see ReadDTCInformation service in 12.3),

— other DTC related data such as first/most recent DTC, flags, counters, timers, etc. specific to DTCs.

IMPORTANT — The server and the client shall meet the request and response message behaviour as specified in 8.7.

12.2.2 Request message

12.2.2.1 Request message definition

Table 296 specifies the request message.

Table 296 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	ClearDiagnosticInformation Request SID	M	14_{16}	CDTCI
	groupOfDTC[] = [			GODTC_
#2	groupOfDTCHighByte	M	00_{16} to FF_{16}	HB
#3	groupOfDTCMiddleByte	M	00_{16} to FF_{16}	MB
#4	groupOfDTCLowByte]	M	00_{16} to FF_{16}	LB
#5	Memory Selection	U	00_{16} to FF_{16}	MEMYS

12.2.2.2 Request message SubFunction parameter $Level (LEV_) definition

There are no SubFunction parameters used by this service.

12.2.2.3 Request message data-parameter definition

Table 297 specifies the data-parameter of the request message.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

Table 297 — Request message data-parameter definition

Definition
groupOfDTC This parameter contains a 3-Byte value indicating the group of DTCs (e.g. Powertrain, Body, Chassis) or the particular DTC to be cleared. The definition of values for each value/range of values is included in D.1.
MemorySelection This parameter shall be used to address the respective user defined DTC memory when retrieving DTCs.

12.2.3 Positive response message

12.2.3.1 Positive response message definition

Table 298 specifies the positive response message.

Table 298 — Positive response message definition

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	ClearDiagnosticInformation Positive Response SID	M	54_{16}	CDTCIPR

12.2.3.2 Positive response message data-parameter definition

There are no data-parameters used by this service in the positive response message.

12.2.4 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 299. The listed negative responses shall be used if the error scenario applies to the server.

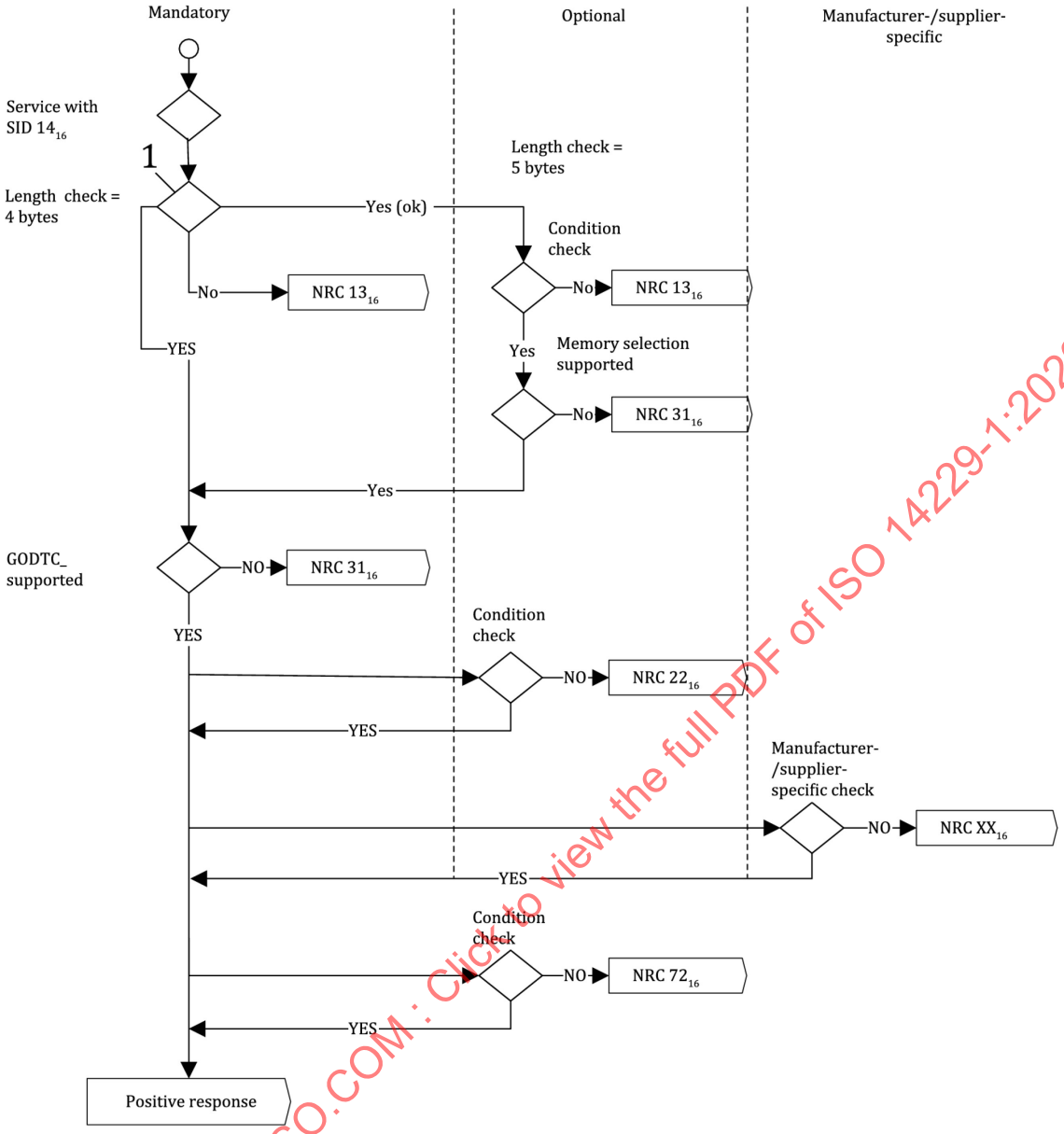
Table 299 — Supported negative response codes

NRC	Description	Mnemonic
13_{16}	**incorrectMessageLengthOrInvalidFormat** This NRC shall be sent if the length of the message is wrong.	IMLOIF
22_{16}	**conditionsNotCorrect** This NRC shall be used if internal conditions within the server prevent the clearing of DTC-related information stored in the server.	CNC
31_{16}	**requestOutOfRange** This NRC shall be returned if the specified groupOfDTC parameter is not supported.	ROOR
72_{16}	**generalProgrammingFailure** This NRC shall be returned if the server detects an error when writing to a memory location.	GPF

The evaluation sequence is documented in Figure 28.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved



Key

1 CDTCI + GODTC_

Figure 28 — NRC handling for ClearDiagnosticInformation service

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

12.2.5 Message flow example ClearDiagnosticInformation

The client sends a ClearDiagnosticInformation request message to a single server. Table 300 specifies the ClearDiagnosticInformation request message flow example #1. The client sends a ClearDiagnosticInformation request message to a single server.

© ISO 2020 – All rights reserved

Table 300 — ClearDiagnosticInformation request message flow example #1

Message direction		client → server	
Message type		Request	
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	ClearDiagnosticInformation Request SID	14_{16}	CDTCI
#2	groupOfDTC [DTCHighByte] (“Emissions-related systems”)	FF_{16}	DTCHB
#3	groupOfDTC [DTCMiddleByte]	FF_{16}	DTCMB
#4	groupOfDTC [DTCLowByte]	33_{16}	DTCLB

Table 301 specifies the ClearDiagnosticInformation positive response message flow example #1.

Table 301 — ClearDiagnosticInformation positive response message flow example #1

Message direction		server → client	
Message type		Response	
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	ClearDiagnosticInformation Response SID	54_{16}	CDTCIPR

12.3 ReadDTCInformation (19_{16}) service

12.3.1 Service description

12.3.1.1 General description

This service allows a client to read the status of server resident Diagnostic Trouble Code (DTC) information from any server, or group of servers within a vehicle. Unless otherwise required by the particularSubFunction, the server shall return all DTC information (e.g. emissions-related and non emissions-related). This service allows the client to do the following:

— retrieve the number of DTCs matching a client defined DTC status mask;

— retrieve the list of all DTCs matching a client defined DTC status mask;

— retrieve the list of DTCs within a particular functional group matching a client defined DTC status mask;

— retrieve all DTCs with "permanent DTC" status;

— retrieve DTCSnapshot data (sometimes referred to as freeze frames) associated with a client defined DTC: DTC Snapshots are specific data records associated with a DTC, that are stored in the server's memory. The typical usage of DTC Snapshots is to store data upon detection of a system malfunction. The DTC Snapshots will act as a snapshot of data values from the time of the system malfunction occurrence. The data-parameters stored in the DTC Snapshot shall be associated to the DTC. The DTC specific data-parameters are intended to ease the fault isolation process by the technician;

— retrieve a list of all DTCs which support a specific DTCExtendedDataRecord for a client defined DTCExtDataRecordNumber;

— retrieve DTCExtendedData associated with a client defined DTC and status mask combination out of the DTC memory. DTCExtendedData consists of extended status information associated with a DTC.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

DTCExtendedData contains DTC parameter values, which have been identified at the time of the request. A typical use of DTCExtendedData is to store dynamic data associated with the DTC, for example:

- DTC B1 Malfunction Indicator counter which conveys the amount of time (number of engine operating hours) during which the OBD system has operated while a malfunction is active;
- DTC occurrence counter, counts number of driving cycles in which "testFailed" has been reported;
- DTC aging counter, counts number of driving cycles since the fault was latest failed excluding the driving cycles in which the test has not reported "testPassed" or "testFailed";
- specific counters for OBD (e.g. number of remaining driving cycles until the "check engine" lamp is switched off if driving cycle can be performed in a fault free mode);
- time of last occurrence (etc.);
- test failed counter, counts number of reported "testFailed" and possible other counters if the validation is performed in several steps;
- uncompleted test counters, counts numbers of driving cycles since the test was latest completed (i.e. since the test reported "testPassed" or "testFailed");

— retrieve the number of DTCs matching a client defined severity mask;

— retrieve the list of DTCs matching a client defined severity mask record;

— retrieve severity information for a client defined DTC;

— retrieve the status of all DTCs supported by the server;

— retrieve the first DTC failed by the server;

— retrieve the most recently failed DTC within the server;

— retrieve the first DTC confirmed by the server;

— retrieve the most recently confirmed DTC within the server;

— retrieve all current "prefailed" DTCs which have or have not yet been detected as "pending" or "confirmed";

— retrieve DTCExtendedData associated with a client defined DTCExtendedData record status out of the DTC memory;

— retrieve the list of DTCs out of a user defined DTC memory matching a client defined DTC status mask;

— retrieve user defined DTC memory DTCExtendedData record data for a client defined DTC mask;

— retrieve user defined DTC memory DTCSnapshotRecord data for a client defined DTC mask out of the user defined DTC memory;

— retrieve DTC information for a client defined DTCReadinessGroupIdentifier.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

This service uses a SubFunction to determine which type of diagnostic information the client is requesting. Further details regarding each SubFunction parameter are provided in the following subclauses.

This service makes use of the following terms:

— **Enable criteria:** Server/vehicle manufacturer/system supplier specific criteria used to control when the server actually performs a particular internal diagnostic.

— **Test pass criteria:** Server/vehicle manufacturer/system supplier specific conditions, that define, whether a system being diagnosed is functioning properly within normal, acceptable operating ranges (e.g. no failures exist and the diagnosed system is classified as “OK”).

— **Test failure criteria:** Server/vehicle manufacturer/system supplier specific failure conditions that define, whether a system being diagnosed has failed the test.

— **Confirmed failure criteria:** Server/vehicle manufacturer/system supplier specific failure conditions that define whether the system being diagnosed is definitively problematic (confirmed), warranting storage of the DTC record in long term memory.

— **Occurrence counter:** A counter maintained by certain servers that records the number of instances in which a given DTC test reported a unique occurrence of a test failure.

— **Aging:** A process whereby certain servers evaluate past results of each internal diagnostic to determine if a confirmed DTC can be cleared from long-term memory, for example in the event of a calibrated number of failure free cycles.

A given DTC value (e.g. 080511_{16}) shall never be reported more than once in a positive response to readDTCInformation with the exception of reading DTCSnapshotRecords, where the response may contain multiple DTCSnapshotRecords for the same DTC.

When using paged-buffer-handling to read DTCs (especially for SubFunction = reportDTCByStatusMask), it is possible that the number of DTCs can decrease while creating the response. In such a case the response shall be filled up with DTC 000000_{16} and DTC status 00_{16}. The client shall treat these DTCs as not present in the response message.

IMPORTANT — The server and the client shall meet the request and response message behaviour as specified in 8.7.

12.3.1.2 Retrieving the number of DTCs that match a client defined status mask (SubFunction = 01_{16} reportNumberOfDTCByStatusMask)

A client can retrieve a count of the number of DTCs matching a client defined status mask by sending a request for this service with the SubFunction set to reportNumberOfDTCByStatusMask. The response to this request contains the DTCStatusAvailabilityMask, which provides an indication of DTC status bits that are supported by the server for masking purposes. Following the DTCStatusAvailabilityMask the response contains the DTCFormatIdentifier which reports information about the DTC formatting and encoding. The DTCFormatIdentifier is followed by the DTCCount parameter which is a 2-byte unsigned numeric number containing the number of DTCs available in the server's memory based on the status mask provided by the client.

12.3.1.3 Retrieving the list of DTCs that match a client defined status mask (SubFunction = 02_{16} reportDTCByStatusMask)

The client can retrieve a list of DTCs, which satisfy a client defined status mask by sending a request with the SubFunction byte set to reportDTCByStatusMask. This SubFunction allows the client to request the server to report all DTCs that are “testFailed” OR “confirmed” OR “etc.”.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

The evaluation shall be done as follows: The server shall perform a bit-wise logical AND-ing operation between the mask specified in the client's request and the actual status associated with each DTC supported by the server. In addition to the DTCStatusAvailabilityMask, the server shall return all DTCs for which the result of the AND-ing operation is non-zero (i.e. (statusOfDTC & DTCStatusMask) != 0). If the client specifies a status mask that contains bits that the server does not support, then the server shall process the DTC information using only the bits that it does support. If no DTCs within the server match the masking criteria specified in the client's request, no DTC or status information shall be provided following the DTCStatusAvailabilityMask byte in the positive response message.

DTC status information shall be cleared upon a successful ClearDiagnosticInformation request from the client (see DTC status bit definitions in D.2 for further descriptions on the DTC status bit handling in case of a ClearDiagnosticInformation service request reception in the server).

12.3.1.4 Retrieving DTCSnapshot record identification (SubFunction = 03_{16} reportDTCSnapshotIdentification)

A client can retrieve DTCSnapshot record identification information for all captured DTCSnapshot records by sending a request for this service with the SubFunction set to reportDTCSnapshotIdentification. The server shall return the list of DTCSnapshot record identification information for all stored DTCSnapshot records. Each item the server places in the response message for a single DTCSnapshot record shall contain a DTCRecord [containing the DTC number (high, middle, and low byte)] and the DTCSnapshot record number. In case multiple DTCSnapshot records are stored for a single DTC then the server shall place one item in the response for each occurrence, using a different DTCSnapshot record number for each occurrence (used for the later retrieval of the record data).

A server can support the storage of multiple DTCSnapshot records for a single DTC to track conditions present at each occurrence of the DTC. Support of this functionality, definition of "occurrence" criteria, and the number of DTCSnapshot records to be supported shall be defined by the system supplier/vehicle manufacturer.

DTCSnapshot record identification information shall be cleared upon a successful ClearDiagnosticInformation request from the client. It is in the responsibility of the vehicle manufacturer to specify the rules for the deletion of stored DTCs and DTCSnapshot data in case of a memory overflow (memory space for stored DTCs and DTCSnapshot data completely occupied in the server).

12.3.1.5 Retrieving DTCSnapshot record data for a client defined DTC mask (SubFunction = 04_{16} reportDTCSnapshotRecordByDTCNumber)

A client can retrieve captured DTCSnapshot record data for a client defined DTCMaskRecord in conjunction with a DTCSnapshot record number by sending a request for this service with the SubFunction set to reportDTCSnapshotRecordByDTCNumber. The server shall search through its supported DTCs for an exact match with the DTCMaskRecord specified by the client (containing the DTC number (high, middle, and low byte)). The UserDefDTCSnapshotRecordNumber parameter provided in the client's request shall specify a particular occurrence of the specified DTC for which DTCSnapshot record data is being requested.

NOTE 1 The UserDefDTCSnapshotRecordNumber does not share the same address space as the DTCStoredDataRecordNumber.

If the server supports the ability to store multiple DTCSnapshot records for a single DTC (support of this functionality to be defined by system supplier/vehicle manufacturer), then it is recommended that the server also implements the reportDTCSnapshotIdentification SubFunction parameter. It is recommended that the client first requests the identification of DTCSnapshot records stored using the SubFunction parameter reportDTCSnapshotIdentification before requesting a specific DTCSnapshotRecordNumber via the reportDTCSnapshotRecordByDTCNumber request.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

It is also recommended to support the SubFunction parameter reportDTCSnapshotRecordIdentification in order to give the client the opportunity to identify the stored DTCSnapshot records directly instead of parsing through all stored DTCs of the server to determine if a DTCSnapshot record is stored.

It shall be the responsibility of the system supplier/vehicle manufacturer to define whether DTCSnapshot records captured within such servers store data associated with occurrence information of a failure as part of the ECU documentation.

Along with the DTC number and statusOfDTC, the server shall return a single predefined DTCSnapshotRecord in response to the client's request, if a failure has been identified for the client defined DTCMaskRecord and DTCSnapshotRecordNumber parameters (DTCSnapshotRecordNumber unequal FF_{16}).

NOTE 2 The exact failure criteria is defined by the system supplier/vehicle manufacturer.

The DTCSnapshot record may contain multiple data-parameters that can be used to reconstruct the vehicle conditions (e.g. B+, RPM, time-stamp) at the time of the failure occurrence.

The vehicle manufacturer shall define format and content of the DTCSnapshotRecord. The data reported in the DTCSnapshotRecord first of all contains a dataIdentifier to identify the data that follows. This dataIdentifier/data combination can be repeated within the DTCSnapshotRecord.The usage of one or multiple dataIdentifiers in the DTCSnapshotRecord allows for the storage of different types of DTCSnapshotRecords for a single DTC for different occurrences of the failure. A parameter which indicates the number of record DataIdentifiers contained within each DTCSnapshotRecord shall be provided with each DTCSnapshotRecord to assist data retrieval.

The server shall report one DTCSnapshot record in a single response message, except the client has set the UserDefDTCSnapshotRecordNumber to FF_{16}, because this shall cause the server to respond with all DTCSnapshot records stored for the client defined DTCMaskRecord in a single response message. The DTCAndStatusRecord is only included one time in the response message. If the client has used FF_{16} for the parameter DTCSnapshotRecordNumber in its request, the server shall report all DTCSnapshot records captured for the particular DTC in numeric ascending order.

The server shall negatively respond if the DTCMaskRecord or DTCSnapshotRecordNumber parameters specified by the client are invalid or not supported by the server. This is to be differentiated from the case in which the DTCMaskRecord and/or DTCSnapshotRecordNumber parameters specified by the client are indeed valid and supported by the server, but have no DTCSnapshot data associated with it (e.g. because a failure event never occurred for the specified DTC or record number). The server shall send the positive response containing only the DTCAndStatusRecord [echo of the requested DTC number (high, middle, and low byte) plus the statusOfDTC].

DTCSnapshot information shall be cleared upon a successful ClearDiagnosticInformation request from the client. It is in the responsibility of the vehicle manufacturer to specify the rules for the deletion of stored DTCs and DTCSnapshot data in case of a memory overflow (memory space for stored DTCs and DTCsnapshot data completely occupied in the server).

12.3.1.6 Retrieving DTCStoredData record data for a client defined record number (SubFunction = 05_{16} reportDTCStoredDataByRecordNumber)

A client can retrieve captured DTCStoredData record data for a DTCStoredDataRecordNumber by sending a request for this service with the SubFunction set to reportDTCStoredDataByRecordNumber. The server shall search through its stored DTCStoredData records for the match of the client provided record number.

The DTCStoredDataRecordNumber does not share the same address space as the DTCSnapshotRecordNumber.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

It shall be the responsibility of the system supplier/vehicle manufacturer to define whether DTCStoredData records captured within such servers store data associated with the first or most recent occurrence of a failure.

NOTE The exact failure criteria is defined by the system supplier/vehicle manufacturer.

The DTCStoredData record may contain multiple data-parameters that can be used to reconstruct the vehicle conditions (e.g. B+, RPM, time-stamp) at the time of the failure occurrence.

The vehicle manufacturer shall define format and content of the DTCStoredDataRecordNumber. The data reported in the DTCStoredDataRecord first of all contains a dataIdentifier to identify the data that follows. This dataIdentifier/data combination can be repeated within the DTCStoredDataRecord. The usage of one or multiple dataIdentifiers in the DTCStoredDataRecord allows for the storage of different types of DTCStoredDataRecords for a single DTC for different occurrences of the failure. A parameter which indicates the number of record DataIdentifiers contained within each DTCStoredDataRecord shall be provided with each DTCStoredDataRecord to assist data retrieval.

The server shall report one DTCStoredDataRecord in a single response message, except the client has set the DTCStoredDataRecordNumber to FF_{16}, because this shall cause the server to respond with all DTCStoredDataRecords stored in a single response message.

In case the client requested to report all DTCStoredDataRecords by record number, then the DTCAndStatusRecord shall be repeated in the response message for each stored DTCStoredDataRecord.

The server shall negatively respond if the DTCStoredDataRecordNumber parameters specified by the client are invalid or not supported by the server. This shall be differentiated from the case in which the DTCStoredDataRecordNumber parameters specified by the client are indeed valid and supported by the server, but have no DTCStoredDataRecord data associated with it (e.g. because a failure event never occurred for the specified record number). In this case, the server shall send the positive response containing only the DTCStoredDataRecordNumber (echo of the requested record number).

DTCStoredDataRecord information shall be cleared upon a successful ClearDiagnosticInformation request from the client. It is in the responsibility of the vehicle manufacturer to specify the rules for the deletion of stored DTCs and DTCStoredDataRecord data in case of a memory overflow (memory space for stored DTCs and DTCStoredDataRecord data completely occupied in the server).

12.3.1.7 Retrieving DTCExtendedData record data for a client defined DTC mask and a client defined DTCExtendedData record number (SubFunction = 06_{16} reportDTCExtDataRecordByDTCNumber)

A client can retrieve DTCExtendedData for a client defined DTCMaskRecord in conjunction with a DTCExtendedData record number by sending a request for this service with the SubFunction set to reportDTCExtDataRecordByDTCNumber. The server shall search through its supported DTCs for an exact match with the DTCMaskRecord specified by the client [containing the DTC number (high, middle, and low byte)]. In this case, the DTCExtDataRecordNumber parameter provided in the client's request shall specify a particular DTCExtendedData record of the specified DTC for which DTCExtendedData is being requested.

Along with the DTC number and statusOfDTC, the server shall return a single predefined DTCExtendedData record in response to the client's request (DTCExtDataRecordNumber unequal to FE_{16} or FF_{16}).

The vehicle manufacturer shall define format and content of the DTCExtDataRecord. The structure of the data reported in the DTCExtDataRecord is defined by the DTCExtDataRecordNumber in a similar way to the definition of data within a record DataIdentifier. Multiple DTCExtDataRecordNumbers and associated DTCExtDataRecords may be included in the response. The usage of one or multiple DTCExtDataRecordNumbers allows for the storage of different types of DTCExtDataRecords for a single DTC.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

The server shall report one DTCExtendedData record in a single response message, except the client has set the DTCExtDataRecordNumber to FE_{16} or FF_{16}, because this shall cause the server to respond with all DTCExtendedData records stored for the client defined DTCMaskRecord in a single response message.

The server shall negatively respond if the DTCMaskRecord or DTCExtDataRecordNumber parameters specified by the client are invalid or not supported by the server. This includes the case where a DTCExtDataRecordNumber of FE_{16} is sent by the client, but no OBD extended data records (90_{16} to EF_{16}) are supported by the server. This shall be differentiated from the case in which the DTCMaskRecord and/or DTCExtDataRecordNumber parameters specified by the client are indeed valid and supported by the server, but have no DTC extended data associated with it (e.g. because of memory overflow of the extended data). In case of reportDTCExtDataRecordByDTCNumber the server shall send the positive response containing only the DTCAndStatusRecord [echo of the requested DTC number (high, middle, and low byte) plus the statusOfDTC].

Clearance of DTCExtendedData information upon the reception of a ClearDiagnosticInformation service is specified in 11.2.1. It is in the responsibility of the vehicle manufacturer to specify the rules for the deletion of stored DTCs and DTC extended data in case of a memory overflow (memory space for stored DTCs and DTC extended data completely occupied in the server).

12.3.1.8 Retrieving the number of DTCs that match a client defined severity mask record (SubFunction = 07_{16} reportNumberOfDTCBySeverityMaskRecord)

A client can retrieve a count of the number of DTCs matching a client defined severity status mask record by sending a request for this service with the SubFunction set to reportNumberOfDTCBySeverityMaskRecord. The server shall scan through all supported DTCs, performing a bit-wise logical AND-ing operation between the mask record specified by the client with the actual information of each stored DTC.

(((statusOfDTC & DTCStatusMask) != 0) && ((severity & DTCSeverityMask) != 0)) == TRUE

For each AND-ing operation yielding a TRUE result, the server shall increment a counter by 1. If the client specifies a status mask within the mask record that contains bits that the server does not support, then the server shall process the DTC information using only the bits that it does support. Once all supported DTCs have been checked once, the server shall return the DTCStatusAvailabilityMask and resulting 2-byte count to the client.

If no DTCs within the server match the masking criteria specified in the client's request, the count returned by the server to the client shall be 0. The reported number of DTCs matching the DTC status mask is valid for the point in time when the request was made. There is no relationship between the reported number of DTCs and the actual list of DTCs read via the SubFunction reportDTCByStatusMask, because the request to read the DTCs is done at a different point in time.

12.3.1.9 Retrieving severity and functional unit information that match a client defined severity mask record (SubFunction = 08_{16} reportDTCBySeverityMaskRecord)

The client can retrieve a list of DTC severity and functional unit information, which satisfy a client defined severity mask record by sending a request with the SubFunction byte set to reportDTCBySeverityMaskRecord. This SubFunction allows the client to request the server to report all DTCs with a certain severity and status that are "testFailed" OR "confirmed" OR "etc.". The evaluation shall be done as follows.

The server shall perform a bit-wise logical AND-ing operation between the DTCSeverityMask and the DTCStatusMask specified in the client's request and the actual DTCSeverity and statusOfDTC associated with each DTC supported by the server.

In addition to the DTCStatusAvailabilityMask, server shall return all DTCs for which the result of the AND-ing operation is TRUE.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

(((statusOfDTC & DTCStatusMask) !=0) && ((severity & DTCSeverityMask) != 0)) == TRUE

If the client specifies a status mask within the mask record that contains bits that the server does not support, then the server shall process the DTC information using only the bits that it does support. If no DTCs within the server match the masking criteria specified in the client's request, no DTC or status information shall be provided following the DTCStatusAvailabilityMask byte in the positive response message.

12.3.1.10 Retrieving severity and functional unit information for a client defined DTC (SubFunction = 09_{16} reportSeverityInformationOfDTC)

A client can retrieve severity and functional unit information for a client defined DTCMaskRecord by sending a request for this service with the SubFunction set to reportSeverityInformationOfDTC. The server shall search through its supported DTCs for an exact match with the DTCMaskRecord specified by the client [containing the DTC number (high, middle, and low byte)].

12.3.1.11 Retrieving the status of all DTCs supported by the server (SubFunction = $0A_{16}$ reportSupportedDTC)

A client can retrieve the status of all DTCs supported by the server by sending a request for this service with the SubFunction set to reportSupportedDTCs. The response to this request contains the DTCStatusAvailabilityMask, which provides an indication of DTC status bits that are supported by the server for masking purposes. Following the DTCStatusAvailabilityMask, the response also contains the listOfDTCAndStatusRecord, which contains the DTC number and associated status for every diagnostic trouble code supported by the server.

12.3.1.12 Retrieving the first/most recent failed DTC (SubFunction = $0B_{16}$ reportFirstTestFailedDTC, SubFunction = $0D_{16}$ reportMostRecentTestFailedDTC)

The client can retrieve the first/most recent failed DTC from the server by sending a request with the SubFunction byte set to "reportFirstTestFailedDTC" or "reportMostRecentTestFailedDTC", respectively. Along with the DTCStatusAvailabilityMask, the server shall return the first or most recent failed DTC number and associated status to the client.

No DTC/status information shall be provided following the DTCStatusAvailabilityMask byte in the positive response message if there were no failed DTCs logged since the last time the client requested the server to clear diagnostic information. Also, if the status of only one DTC is failed since the last time the client requested the server to clear diagnostic information, the one failed DTC shall be returned to both reportFirstTestFailedDTC and reportMostRecentTestFailedDTC requests from the client.

Record of the first/most recent failed DTC shall be independent of the ageing process of confirmed DTCs.

As mentioned above, first/most recent failed DTC information shall be cleared upon a successful ClearDiagnosticInformation request from the client (see DTC status bit definitions in D.2 for further descriptions on the DTC status bit handling in case of a ClearDiagnosticInformation service request reception in the server).

12.3.1.13 Retrieving the first/most recently detected confirmed DTC (SubFunction = $0C_{16}$ reportFirstConfirmedDTC,SubFunction = $0E_{16}$ reportMostRecentConfirmedDTC)

The client can retrieve the first/most recent confirmed DTC from the server by sending a request with the SubFunction byte set to "reportFirstConfirmedDTC" or "reportMostRecentConfirmedDTC", respectively. Along with the DTCStatusAvailabilityMask, the server shall return the first or most recent confirmed DTC number and associated status to the client.

No DTC/status information shall be provided following the DTCStatusAvailabilityMask byte in the positive response message if there were no confirmed DTCs logged since the last time the client requested the server to clear diagnostic information. Also, if only 1 DTC is confirmed since the last time

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

the client requested the server to clear diagnostic information the one confirmed DTC shall be returned to both reportFirstConfirmedDTC and reportMostRecentConfirmedDTC requests from the client.

The record of the first confirmed DTC shall be preserved in the event that the DTC failed at one point in the past, but then satisfied aging criteria prior to the time of the request from the client (regardless of any other DTCs that become confirmed after the aforementioned DTC is confirmed). Similarly, record of the most recently confirmed DTC shall be preserved in the event that the DTC was confirmed at one point in the past, but then satisfied aging criteria prior to the time of the request from the client (assuming no other DTCs is confirmed after the aforementioned DTC failed).

As mentioned above, first/most recent confirmed DTC information shall be cleared upon a successful ClearDiagnosticInformation request from the client.

12.3.1.14 Retrieving a list of "prefailed" DTC status (SubFunction = 14_{16} reportDTCFaultDetection-Counter)

The client can retrieve a list of all current "prefailed" DTCs which have or have not yet been detected as "pending" or "confirmed" at the time of the client's request. The intention of the DTCFaultDetectionCounter is a simple method to identify a growing or intermittent problem which cannot be identified/read by the statusOfDTC byte of a particular DTC. The internal implementation of the DTCFaultDetectionCounter shall be vehicle manufacturer specific. The use case of "prefailed" DTCs is to speed up failure detection during testing in the manufacturing plants for DTCs that require a maturation time unacceptable to manufacturing testing. Service has a similar use case after repairing or installing new components.

12.3.1.15 Retrieving a list of DTCs with "permanent DTC" status (SubFunction = 15_{16} reportDTCWithPermanentStatus)

The client can retrieve a list of DTCs with "permanent DTC" status as described in 3.12.

The SubFunction 15_{16} will be replaced by SubFunction 55_{16} in the future. In case there is a need for PermanentDTC implementation it is recommended to use the 55_{16} SubFunction.

12.3.1.16 Retrieving DTCExtendedData record data for a client defined DTCExtendedData record number (SubFunction = 16_{16} reportDTCExtDataRecordByRecordNumber)

A client can retrieve DTCExtendedData for a client defined DTCExtendedData record number by sending a request for this service with the SubFunction set to reportDTCExtDataRecordByRecordNumber. The server shall search through all supported DTCs for exact matches with the DTCExtDataRecordNumber specified by the client. In this case, the DTCExtDataRecordNumber parameter provided in the client's request shall specify a particular DTCExtendedData record for all supported DTCs for which DTCExtendedData is being requested.

The server shall return a DTCExtendedData record along with the DTC number and statusOfDTC for each supported DTC that contains data for the requested DTCExtDataRecordNumber.

The vehicle manufacturer shall define the format and content of the DTCExtDataRecord. The structure of the data reported in the DTCExtDataRecord is defined by the DTCExtDataRecordNumber in a similar way to the definition of data within a record DataIdentifier.

The server shall negatively respond if the DTCExtDataRecordNumber parameter specified by the client is invalid or not supported by the server.

Clearance of DTCExtendedData information upon the reception of a ClearDiagnosticInformation service is specified in 11.2.1. It is in the responsibility of the vehicle manufacturer to specify the rules for the deletion of stored DTCs and DTC extended data in case of a memory overflow (memory space for stored DTCs and DTC extended data completely occupied in the server).

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

12.3.1.17 Retrieving the list of DTCs out of the server's user defined DTC memory that match a client defined DTC status mask (SubFunction = 17_{16} reportUserDefMemoryDTCByStatusMask)

The client can retrieve a list of DTCs from a user defined memory, which satisfy a client defined status mask by sending a request with the SubFunction byte set to reportUserDefMemoryDTCByStatusMask. This SubFunction allows the client to request the server to report all DTCs that are “testFailed” or “confirmed” or “etc.” from the user defined memory.

The evaluation shall be done as follows: the server shall perform a bit-wise logical AND-ing operation between the mask specified in the client’s request and the actual status associated with each DTC supported by the server in that user defined memory. In addition to the DTCStatusAvailabilityMask, the server shall return all DTCs for which the result of the AND-ing operation is non-zero (i.e. (statusOfDTC & DTCStatusMask) != 0) in that specific memory. If the client specifies a status mask that contains bits that the server does not support, then the server shall process the DTC information using only the bits that it does support. If no DTCs within the server match the masking criteria specified in the client’s request in that specific memory, no DTC or status information shall be provided following the DTCStatusAvailabilityMask byte in the positive response message.

DTC status information shall be cleared either by a service 14_{16} ClearDTC service with the memorySelection parameter set to the applicable memory or by manufacturer specific conditions (e.g. a routine control request from the client).

12.3.1.18 Retrieving user defined memory DTCSnapshot record data for a client defined DTC mask and a client defined DTCSnapshotNumber out of the DTC user defined memory (SubFunction = 18_{16} reportUserDefMemoryDTCSnapshotRecordByDTCNumber)

A client can retrieve captured DTCSnapshot record data for a client defined DTCMaskRecord in conjunction with a DTCSnapshot record number and a user defined memory identifier by sending a request for this service with the SubFunction set to reportUserDefMemoryDTCSnapshotRecordByDTCNumber. The server shall search through its supported DTCs for an exact match with the DTCMaskRecord specified by the client [containing the DTC number (high, middle, and low byte)]. The DTCSnapshotRecordNumber parameter provided in the client’s request shall specify a particular occurrence of the specified DTC and the defined memory for which DTCSnapshot record data is being requested.

NOTE 1 The DTCSnapshotRecordNumber does not share the same address space as the DTCStoredDataRecordNumber.

It shall be the responsibility of the system supplier/vehicle manufacturer to define whether DTCSnapshot records captured within such servers store data associated with the first or most recent occurrence of a failure.

Along with the DTC number and statusOfDTC, the server shall return a single predefined UserDefDTCSnapshotRecordNumber from the specific user memory in response to the client’s request, if a failure has been identified for the client defined DTCMaskRecord and UserDefDTCSnapshotRecordNumber parameters (UserDefDTCSnapshotRecordNumber unequal FF_{16}) and that specific memory.

NOTE 2 The exact failure criteria is defined by the system supplier/vehicle manufacturer.

The DTCSnapshot record may contain multiple data-parameters that can be used to reconstruct the vehicle conditions (e.g. B+, RPM, time-stamp) at the time of the failure occurrence.

The vehicle manufacturer shall define format and content of the DTCSnapshotRecord in the user defined memory (i.e. the content of the DTCSnapshotRecords can differ between different memories) records. The data reported in the DTCSnapshotRecord first of all contains a dataIdentifier to identify the data that follows. This dataIdentifier/data combination can be repeated within the

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

DTCSnapshotRecord. The usage of one or multiple dataIdentifiers in the DTCSnapshotRecord in the user defined memory allows for the storage of different types of DTCSnapshotRecords for a single DTC for different occurrences of the failure. A parameter which indicates the number of record DataIdentifiers contained within each DTCSnapshotRecord shall be provided with each DTCSnapshotRecord to assist data retrieval.

The server shall report one DTCSnapshot record in a single response message, except if the client has set the UserDefDTCSnapshotRecordNumber to FF_{16}, because this shall cause the server to respond with all DTCSnapshot records stored for the client defined DTCMaskRecord and the user defined memory in a single response message. The DTCAndStatusRecord is only included one time in the response message.

The server shall negatively respond if the DTCMaskRecord, UserDefDTCSnapshotRecordNumber, UserDefMemory parameters specified by the client are invalid or not supported by the server. This is to be differentiated from the case in which the DTCMaskRecord and/or UserDefDTCSnapshotRecordNumber parameters specified by the client are indeed valid and supported by the server for that specific memory, but have no DTCSnapshot data associated with it (e.g. because a failure event never occurred for the specified DTC or record number). The server shall send the positive response containing only the DTCAndStatusRecord [echo of the requested DTC number (high, middle, and low byte) plus the statusOfDTC].

DTCSnapshot information shall be cleared upon a manufacturer specific conditions (e.g a routine control) request from the client. It is in the responsibility of the vehicle manufacturer to specify the rules for the deletion of stored DTCs and DTCSnapshot data in case of a memory overflow (memory space for stored DTCs and DTCsnapshot data completely occupied in the server for that specific memory).

12.3.1.19 Retrieving user defined memory DTCExtendedData record data for a client defined DTC mask and a client defined DTCExtendedData record number out of the DTC memory (SubFunction = 19_{16} reportUserDefMemoryDTCExtDataRecordByDTCNumber)

A client can retrieve DTCExtendedData for a client defined DTCMaskRecord in conjunction with a DTCExtendedData record number and a UserDefMemoryIdenitfier by sending a request for this service with the SubFunction set to reportUserDefMemoryDTCExtDataRecordByDTCNumber. The server shall search through its supported DTCs for an exact match with the DTCMaskRecord specified by the client [containing the DTC number (high, middle, and low byte)] and the UserDefMemoryIdentifier. In this case the DTCExtDataRecordNumber parameter provided in the client's request shall specify a particular DTCExtendedData record of the specified DTC for which DTCExtendedData is being requested.

Along with the DTC number and statusOfDTC, the server shall return a single predefined DTCExtendedData record in response to the client's request (DTCExtDataRecordNumber unequal to FE_{16} or FF_{16}).

The vehicle manufacturer shall define format and content of the UserDefDTCExtDataRecord. The structure of the data reported in the DTCExtDataRecord is defined by the DTCExtDataRecordNumber for that specific user defined memory in a similar way to the definition of data within a record DataIdentifier. Multiple DTCExtDataRecordNumbers and associated DTCExtDataRecords may be included in the response. The usage of one or multiple DTCExtDataRecordNumbers allows for the storage of different types of DTCExtDataRecords for a single DTC.

The server shall report one DTCExtendedData record in a single response message, except the client has set the DTCExtDataRecordNumber to FE_{16} or FF_{16}, because this shall cause the server to response with all DTCExtendedData records stored for the client defined DTCMaskRecord out of the user defined memory in a single response message.

The server shall negatively respond if the DTCMaskRecord or DTCExtDataRecordNumber parameters specified by the client are invalid or not supported by the server or not in that specific memory. This shall be differentiated from the case in which the DTCMaskRecord and/or DTCExtDataRecordNumber

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

parameters specified by the client are indeed valid and supported by the server, but have no DTC extended data associated with it (e.g. because of memory overflow of the extended data). In case of reportDTCExtDataRecordByDTCNumber the server shall send the positive response containing only the DTCAndStatusRecord [echo of the requested DTC number (high, middle, and low byte) plus the statusOfDTC].

DTCExtendedDataRecord information shall be cleared upon a manufacturer specific conditions by either a service 14_{16} ClearDiagnosticInformation with the memorySelection parameter set to the applicable memory or by a routine control request from the client.

12.3.1.20 Retrieving the list of all DTCs that supports an specific DTCExtendedDataRecord (SubFunction = $1A_{16}$ reportSupportedDTCExtDataRecord)

A client can retrieve the list of all DTCs that supports a specific DTCExtendedDataRecord by sending a request for this service with the SubFunction set to reportDTCExtendedDatatIdentification. The server shall search through its supported DTCs. If a DTC support the requested DTCExtendedDataRecordNumber it should be included in the response. Along with the DTC number(s) and the statusOfDTC, the server shall return the DTCExtendedDataRecordNumber for that DTC(s). The server shall negatively respond if the DTCExtendedDataRecord parameter specified by the client is invalid or not supported by the server.

12.3.1.21 Retrieving the list of VOBD DTCs from a functional group that match a client defined status mask (SubFunction = 42_{16} reportWWHOBDDTCByMaskRecord)

The implementation and usage of DTCSeverityMask (with severity and class) is defined in ISO 27145-3[22].

12.3.1.22 Retrieving a list of VOBD DTCs with "permanent DTC" status (SubFunction = 55_{16} reportWWHOBDDTCWithPermanentStatus)

The client can retrieve a list of VOBD DTCs with the "permanent DTC" status as described in 3.12.

12.3.1.23 Retrieve DTC information for a client defined DTCReadinessGroupIdentifier (SubFunction = 56_{16} reportDTCInformationByDTCReadinessGroupIdentifier)

A client can retrieve DTC information for a client defined DTC Readiness Group Identifier by sending a request of this service with the SubFunction set to reportDTCInformationByDTCReadinessGroupIdentifer. The server shall search through its supported DTCs for an exact match with the DTCReadinessGroupIdentifier specified by the client. Along with the DTC number(s) and the statusOfDTC, the server shall return the DTCReadinessGroupIdentifier for those DTCs.

The server shall negatively respond if the DTCReadinessGroupIdentifier parameter specified by the client is invalid or not supported by the server.

12.3.2 Request message

12.3.2.1 Request message definition

Table 302 specifies the structure of the ReadDTCInformation request message based on the used SubFunction parameter.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

Table 302 — Request message definition - SubFunction = reportNumberOfDTCByStatusMask, reportDTCByStatusMask

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	ReadDTCInformation Request SID	M	19_{16}	RDTCI
#2	SubFunction = [reportType = reportNumberOfDTCByStatusMask reportDTCByStatusMask	M	 01_{16} 02_{16}	LEV_ RNODTCBSM RDTCBSM
#3	DTCStatusMask	M	00_{16} to FF_{16}	DTCSM

Table 303 specifies the structure of the ReadDTCInformation request message based on the used SubFunction parameter.

Table 303 — Request message definition - SubFunction = reportDTCSnapshotIdentification, reportDTCSnapshotRecordByDTCNumber

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	ReadDTCInformation Request SID	M	19_{16}	RDTCI
#2	SubFunction = [reportType = reportDTCSnapshotIdentification reportDTCSnapshotRecordByDTCNumber]	M	 03_{16} 04_{16}	LEV_ RDTCSSI RDTCSSBDTC
 #3 #4 #5	DTCMaskRecord[] = [DTCHighByte DTCMiddleByte DTCLowByte]	 C C C	 00_{16} to FF_{16} 00_{16} to FF_{16} 00_{16} to FF_{16}	DTCMREC_ DTCHB DTCMB DTCLB
#6	DTCSnapshotRecordNumber	C	00_{16} to FF_{16}	DTCSSRN
C: The DTCMaskRecord and DTCSnapshotRecordNumber parameters are only present in case the SubFunction parameter is equal to reportDTCSnapshotRecordByDTCNumber.				

Table 304 specifies the structure of the ReadDTCInformation request message based on the used SubFunction parameter.

Table 304 — Request message definition - SubFunction = reportDTCStoredDataByRecordNumber

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	ReadDTCInformation Request SID	M	19_{16}	RDTCI
#2	SubFunction = [reportType = reportDTCStoredDataByRecordNumber]	M	 05_{16}	LEV_ RDTCSDBRN
#3	DTCStoredDataRecordNumber	M	00_{16} to FF_{16}	DTCSDRN

Table 305 specifies the structure of the ReadDTCInformation request message based on the used SubFunction parameter.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 305 — Request message definition - SubFunction = reportDTCExtDataRecordByDTCNumber

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	ReadDTCInformation Request SID	M	19_{16}	RDTCI
#2	SubFunction = [reportType = reportDTCExtDataRecordByDTCNumber]	M	06_{16}	LEV_ RDTCEDRBDN
	DTCMaskRecord[] = [			DTCMREC_
#3	DTCHighByte	M	00_{16} to FF_{16}	DTCHB
#4	DTCMiddleByte	M	00_{16} to FF_{16}	DTCMB
#5	DTCLowByte]	M	00_{16} to FF_{16}	DTCLB
#6	DTCExtDataRecordNumber	M	00_{16} to FF_{16}	DTCEDRN

Table 306 specifies the structure of the ReadDTCInformation request message based on the used SubFunction parameter.

Table 306 — Request message definition - SubFunction = reportNumberOfDTCBySeverityMaskRecord, reportDTCSeverityInformation

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	ReadDTCInformation Request SID	M	19_{16}	RDTCI
#2	SubFunction = [reportType = reportNumberOfDTCBySeverityMaskRecord	M	07_{16}	LEV_ RNODTCBSMR
	reportDTCBySeverityMaskRecord]		08_{16}	RDTCBSMR
	DTCSeverityMaskRecord[] = [			DTCSVMREC_
#3	DTCSeverityMask	M	00_{16} to FF_{16}	DTCSVM
#4	DTCStatusMask]	M	00_{16} to FF_{16}	DTCSM

Table 307 specifies the structure of the ReadDTCInformation request message based on the used SubFunction parameter.

Table 307 — Request message definition - SubFunction = reportSeverityInformationOfDTC

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	ReadDTCInformation Request SID	M	19_{16}	RDTCI
#2	SubFunction = [reportType = reportSeverityInformationOfDTC]	M	09_{16}	LEV_ RSIODTC
	DTCMaskRecord[] = [			DTCMREC_
#3	DTCHighByte	M	00_{16} to FF_{16}	DTCHB
#4	DTCMiddleByte	M	00_{16} to FF_{16}	DTCMB
#5	DTCLowByte]	M	00_{16} to FF_{16}	DTCLB

Table 308 specifies the structure of the ReadDTCInformation request message based on the used SubFunction parameter.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 308 — Request message definition - SubFunction = reportSupportedDTC, reportFirstTestFailedDTC, reportFirstConfirmedDTC, reportMostRecentTestFailedDTC, reportMostRecentConfirmedDTC, reportDTCFaultDetectionCounter, reportDTCWithPermanentStatus

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	ReadDTCInformation Request SID	M	19_{16}	RDTCI
#2	SubFunction = [reportType = reportSupportedDTC reportFirstTestFailedDTC reportFirstConfirmedDTC reportMostRecentTestFailedDTC reportMostRecentConfirmedDTC reportDTCFaultDetectionCounter reportDTCWithPermanentStatus]	M	 $0A_{16}$ $0B_{16}$ $0C_{16}$ $0D_{16}$ $0E_{16}$ 14_{16} 15_{16}	LEV_ RSUPDTC RFTFDTC RFCDTC RMRTFDTC RMRCDTC RDTCFDC RDTCWPS

Table 309 specifies the structure of the ReadDTCInformation request message based on the used SubFunction parameter.

Table 309 — Request message definition - SubFunction = reportDTCExtDataRecordByRecordNumber

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	ReadDTCInformation Request SID	M	19_{16}	RDTCI
#2	SubFunction = [reportType = reportDTCExtDataRecordByRecordNumber]	M	 16_{16}	LEV_ RDTCEDRBR
#3	DTCExtDataRecordNumber	M	00_{16} to EF_{16}	DTCEDRN

Table 310 specifies the structure of the ReadDTCInformation request message based on the used SubFunction parameter.

Table 310 — Request message definition - SubFunction = reportUserDefMemoryDTCByStatusMask

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	ReadDTCInformation Request SID	M	19_{16}	RDTCI
#2	SubFunction = [reportType = reportUserDefMemoryDTCByStatusMask	M	 17_{16}	LEV_ RUDMDTCBSM
#3	DTCStatusMask	M	00_{16} – FF_{16}	DTCSM
#4	MemorySelection	M	00_{16} to FF_{16}	MEMYS

Table 311 specifies the structure of the ReadDTCInformation request message based on the used SubFunction parameter.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 311 — Request message definition - SubFunction = reportUserDefMemoryDTCSnapshotRecordByDTCNumber

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	ReadDTCInformation Request SID	M	19_{16}	RDTCI
#2	SubFunction = [reportType = reportUserDefMemoryDTCSnapshotRecordByDTCNumber]	M	18_{16}	LEV_ RUDMDTCSSBDTC
	DTCMaskRecord[] = [			DTCMREC_
#3	DTCHighByte	M	00_{16} to FF_{16}	DTCHB
#4	DTCMiddleByte	M	00_{16} to FF_{16}	DTCMB
#5	DTCLowByte]	M	00_{16} to FF_{16}	DTCLB
#6	UserDefDTCSnapshotRecordNumber	M	00_{16} to FF_{16}	UDDTCSSRN
#7	MemorySelection	M	00_{16} to FF_{16}	MEMYS

Table 312 specifies the structure of the ReadDTCInformation request message based on the used SubFunction parameter.

Table 312 — Request message definition - SubFunction = reportUserDefMemoryDTCExtDataRecordByDTCNumber

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	ReadDTCInformation Request SID	M	19_{16}	RDTCI
#2	SubFunction = [reportType = reportUserDefMemoryDTCExtDataRecordByDTCNumber]	M	19_{16}	LEV_ RUDMDTCEDRBDN
	DTCMaskRecord[] = [			DTCMREC_
#3	DTCHighByte	M	00_{16} to FF_{16}	DTCHB
#4	DTCMiddleByte	M	00_{16} to FF_{16}	DTCMB
#5	DTCLowByte]	M	00_{16} to FF_{16}	DTCLB
#6	DTCExtDataRecordNumber	M	00_{16} to FF_{16}	DTCEDRN
#7	MemorySelection	M	00_{16} to FF_{16}	MEMYS

Table 313 specifies the structure of the ReadDTCInformation request message based on the used SubFunction parameter.

Table 313 — Request message definition - SubFunction = reportSupportedDTCExtDataRecord

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ReadDTCInformation Request SID	M	19_{16}	RDTCI
#2	SubFunction = [reportType = reportSupportedDTCExtDataRecord]	M	$1A_{16}$	LEV_ RDTCEDI
#3	DTCExtDataRecordNumber	M	01_{16} to FD_{16}	DTCEDRN

Table 314 specifies the structure of the ReadDTCInformation request message based on the used SubFunction parameter.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 314 — Request message definition - SubFunction = reportWWHOBDDTCByMaskRecord

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	ReadDTCInformation Request SID	M	19_{16}	RDTCI
#2	SubFunction = [reportType = reportWWHOBDDTCByMaskRecord]	M	42_{16}	LEV_ ROBDDTCBMR
#3	FunctionalGroupIdentifier	M	00_{16} to FE_{16}	FGID
	DTCSeverityMaskRecord[] = [			DTCSVMREC_
#4	DTCStatusMask	M	00_{16} to FF_{16}	DTCSM
#5	DTCSeverityMask]	M	00_{16} to FF_{16}	DTCSVM

Table 315 specifies the structure of the ReadDTCInformation request message based on the used SubFunction parameter.

Table 315 — Request message definition - SubFunction = reportWWHOBDDTCWithPermanentStatus

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	ReadDTCInformation Request SID	M	19_{16}	RDTCI
#2	SubFunction = [reportType = reportWWHOBDDTCWithPermanentStatus]	M	55_{16}	LEV_ RWWHOBDDTCWPS
#3	FunctionalGroupIdentifier	M	00_{16} to FE_{16}	FGID

Table 316 specifies the structure of the ReadDTCInformation request message based on the used SubFunction parameter.

Table 316 — Request message definition - SubFunction = reportDTCInformationByDTCReadinessGroupIdentifier

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ReadDTCInformation Request SID	M	19_{16}	RDTCI
#2	SubFunction = [reportType = reportDTCInformationByDTCReadinessGroupIdentifier]	M	56_{16}	LEV_ RDTCBRGI
#3	FunctionalGroupIdentifier	M	00_{16} to FE_{16}	FGID
#4	DTCReadinessGroupIdentifier	M	00_{16} to FE_{16}	DTCRGI

12.3.2.2 Request message SubFunction parameter $Level (LEV_) definition

The SubFunction parameters are used by this service to select one of the DTC report types specified in Table 317. Explanations and usage of the possible levels are detailed below (suppressPosRspMsgIndicationBit (bit 7) not shown).

Table 317 — Request message SubFunction definition

Bits 6 to 0	Description	Cvt	Mnemonic
00_{16}	**ISOSAEReserved** This value is reserved by this document for future definition.	M	ISOSAERESRVD

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Bits 6 to 0	Description	Cvt	Mnemonic
01_{16}	**reportNumberOfDTCByStatusMask** This parameter specifies that the server shall transmit to the client the number of DTCs matching a client defined status mask.	U	RNODTCBSM
02_{16}	**reportDTCByStatusMask** This parameter specifies that the server shall transmit to the client a list of DTCs and corresponding statuses matching a client defined status mask.	U	RDTCBSM
03_{16}	**reportDTCSnapshotIdentification** This parameter specifies that the server shall transmit to the client all DTCSnapshot data record identifications [DTC number(s) and DTCSnapshot record number(s)].	U	RDTCSSI
04_{16}	**reportDTCSnapshotRecordByDTCNumber** This parameter specifies that the server shall transmit to the client the DTCSnapshot record(s) associated with a client defined DTC number and DTCSnapshot record number (FF_{16} for all records).	U	RDTCSSBDTC
05_{16}	**reportDTCStoredDataByRecordNumber** This parameter specifies that the server shall transmit to the client the DTCStoredDatarecord(s) associated with a client defined DTCStoredData record number (FF_{16} for all records).	U	RDTCSDBRN
06_{16}	**reportDTCExtDataRecordByDTCNumber** This parameter specifies that the server shall transmit to the client the DTCExtendedData record(s) associated with a client defined DTC number and DTCExtendedData record number (FF_{16} for all records, FE_{16} for all OBD records).	U	RDTCEDRBDN
07_{16}	**reportNumberOfDTCBySeverityMaskRecord** This parameter specifies that the server shall transmit to the client the number of DTCs matching a client defined severity mask record.	U	RNODTCBSMR
08_{16}	**reportDTCBySeverityMaskRecord** This parameter specifies that the server shall transmit to the client a list of DTCs and corresponding statuses matching a client defined severity mask record.	U	RDTCBSMR
09_{16}	**reportSeverityInformationOfDTC** This parameter specifies that the server shall transmit to the client the severity information of a specific DTC specified in the client request message.	U	RSIODTC
$0A_{16}$	**reportSupportedDTC** This parameter specifies that the server shall transmit to the client a list of all DTCs and corresponding statuses supported within the server.	U	RSUPDTC
$0B_{16}$	**reportFirstTestFailedDTC** This parameter specifies that the server shall transmit to the client the first failed DTC to be detected by the server since the last clear of diagnostic information. Note that the information reported via this SubFunction parameter shall be independent of whether or not the DTC was confirmed or aged.	U	RFTFDTC

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Bits 6 to 0	Description	Cvt	Mnemonic
$0C_{16}$	**reportFirstConfirmedDTC** This parameter specifies that the server shall transmit to the client the first confirmed DTC to be detected by the server since the last clear of diagnostic information. The information reported via this SubFunction parameter shall be independent of the aging process of confirmed DTCs (e.g. if a DTC ages such that its status is allowed to be reset, the first confirmed DTC record shall continue to be preserved by the server, regardless of any other DTCs that become confirmed afterwards).	U	RFCDTC
$0D_{16}$	**reportMostRecentTestFailedDTC** This parameter specifies that the server shall transmit to the client the most recent failed DTC to be detected by the server since the last clear of diagnostic information. Note that the information reported via this SubFunction parameter shall be independent of whether or not the DTC was confirmed or aged.	U	RMRTFDTC
$0E_{16}$	**reportMostRecentConfirmedDTC** This parameter specifies that the server shall transmit to the client the most recent confirmed DTC to be detected by the server since the last clear of diagnostic information. Note that the information reported via this SubFunction parameter shall be independent of the aging process of confirmed DTCs (e.g. if a DTC ages such that its status is allowed to be reset, the first confirmed DTC record shall continue to be preserved by the server assuming no other DTCs become confirmed afterwards).	U	RMRCDTC
14_{16}	**reportDTCFaultDetectionCounter** This parameter specifies that the server shall transmit to the client a list of current "prefailed" DTCs which have or have not yet been detected as "pending" or "confirmed". The intention of the DTCFaultDetectionCounter is a simple method to identify a growing or intermittent problem which can not be identified/read by the statusOfDTC byte of a particular DTC. The internal implementation of the DTCFaultDetectionCounter shall be vehicle manufacturer specific (e.g. number of bytes, signed versus unsigned, etc.) but the reported value shall be a scaled 1 byte signed value so that +127 ($7F_{16}$) represents a test result of "failed" and any other non-zero positive value represents a test result of "prefailed". However DTCs with DTCFaultDetectionCounter with the value +127 shall not be reported according to below stated rule. The DTCFaultDetectionCounter shall be incremented by a vehicle manufacturer specific amount each time the test logic runs and indicates a fail for that test run. A reported DTCFaultDetectionCounter value greater than zero and less than +127 (i.e. 01_{16} to $7E_{16}$) indicates that the DTC enable criteria was met and that a non completed test result prefailed at least in one condition or threshold. Only DTCs with DTCFaultDetectionCounters with a non-zero positive	U	RDTCFDC

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Bits 6 to 0	Description	Cvt	Mnemonic
	value less than +127 ($7F_{16}$) shall be reported. The DTCFaultDetectionCounter shall be decremented by a vehicle manufacturer specific amount each time the test logic runs and indicates a pass for that test run. If the DTCFaultDetectionCounter is decremented to zero or below the DTC shall no longer be reported in the positive response message. The value of the DTCFaultDetectionCounter shall not be maintained between operation cycles. If a ClearDiagnosticInformation service request is received the DTCFaultDetectionCounter value shall be reset to zero for all DTCs. Additional reset conditions shall be defined by the vehicle manufacturer. Refer to D.5 for example implementation details.		
15_{16}	**reportDTCWithPermanentStatus** This parameter specifies that the server shall transmit to the client a list of DTCs with "permanent DTC" status as described in 3.12.	U	RDTCWPS
16_{16}	**reportDTCExtDataRecordByRecordNumber** This parameter specifies that the server shall transmit to the client the DTCExtendedData records associated with a client defined DTCExtendedData record number less than $F0_{16}$.	U	RDTCEDBR
17_{16}	**reportUserDefMemoryDTCByStatusMask** This parameter specifies that the server shall transmit to the client a list of DTCs out of the user defined DTC memory and corresponding statuses matching a client defined status mask.	U	RUDMDTCBSM
18_{16}	**reportUserDefMemoryDTCSnapshotRecordByDTCNumber** This parameter specifies that the server shall transmit to the client the DTCSnapshot record(s) – out of the user defined DTC memory - associated with a client defined DTC number and DTCSnapshot record number (FF_{16} for all records).	U	RUDMDTCSSBDTC
19_{16}	**reportUserDefMemoryDTCExtDataRecordByDTCNumber** This parameter specifies that the server shall transmit to the client the DTCExtendedData record(s) – out of the user defined DTC memory - associated with a client defined DTC number and DTCExtendedData record number (FF_{16} for all records).	U	RUDMDTCEDRBDN
$1A_{16}$	**reportDTCExtendedDataRecordIdentification** This parameter specifies that the server shall transmit to the client the DTCs which supports a DTCExtendedDataRecord.	U	RDTCEDI
$1B_{16}$ to 41_{16}	**ISOSAEReserved** This value is reserved by this document for future definition.	M	ISOSAERESRVD
42_{16}	**reportWWHOBDDTCByMaskRecord** This parameter specifies that the server shall transmit to the client a list of WWH OBD DTCs and corresponding status and severity information matching a client defined status mask and severity mask record.	U	RWWHOBDDTCBMR
43_{16} to 54_{16}	**ISOSAEReserved** This value is reserved by this document for future definition.	M	ISOSAERESRVD

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Bits 6 to 0	Description	Cvt	Mnemonic
55_{16}	**reportWWHOBDDTCWithPermanentStatus** This parameter specifies that the server shall transmit to the client a list of WWH OBD DTCs with "permanent DTC" status as described in 3.12.	U	RWWHOBDDTCWPS
56_{16}	**reportDTCInformationByDTCReadinessGroupIdentifier** This parameter specifies that the server shall transmit to the client a list of OBD DTCs which matches the DTCReadiness Group Identifier.	U	RDTCBRGI
57_{16} to $7F_{16}$	**ISOSAEReserved** This value is reserved by this document for future definition.	M	ISOSAERESRVD

12.3.2.3 Request message data-parameter definition

Table 318 specifies the data-parameters of the request message.

Table 318 — Request data-parameter definition

Definition
DTCStatusMask The DTCStatusMask contains eight (8) DTC status bits. The definitions for each of the eight bits can be found in D.2. This byte is used in the request message to allow a client to request DTC information for the DTCs whose status matches the DTCStatusMask. A DTCs status matches the DTCStatusMask if any one of the DTCs actual status bits is set to '1' and the corresponding status bit in the DTCStatusMask is also set to '1' (i.e. if the DTCStatusMask is bit-wise logically ANDed with the DTCs actual status and the result is non-zero, then a match has occurred). If the client specifies a status mask that contains bits that the server does not support, then the server shall process the DTC information using only the bits that it does support.
DTCMaskRecord [DTCHighByte, DTCMiddleByte, DTCLowByte] DTCMaskRecord is a 3-Byte value containing DTCHighByte, DTCMiddleByte and DTCLowByte, which together represent a unique identification number for a specific diagnostic trouble code supported by a server. The definition of the 3-byte DTC number allows for several ways of coding DTC information. It can either be done: — by using the decoding of the DTCHighByte, DTCMiddleByte and DTCLowByte according to the ISO 15031-6[17] specification. This format is identified by the DTCFormatIdentifier = SAE_J2012-DA_DTCFormat_00, or — by using the decoding of the DTCHighByte, DTCMiddleByte and DTCLowByte according to this document which does not specify any decoding method and therefore allows a vehicle manufacturer defined decoding method. This format is identified by the DTCFormatIdentifier = ISO_14229-1_DTCFormat, or — by using the decoding of the DTCHighByte, DTCMiddleByte and DTCLowByte according to the SAE J1939-73[24] specification. This format is identified by the DTCFormatIdentifier = SAE_J1939-73_DTCFormat, or — by using the decoding of the DTCHighByte, DTCMiddleByte and DTCLowByte according to the ISO 11992-4[9] specification. This format is identified by the DTCFormatIdentifier = ISO_11992-4_DTCFormat, — by using the decoding of the DTCHighByte, DTCMiddleByte and DTCLowByte according to the ISO 27145-2[21] specification. This format is identified by the DTCFormatIdentifier = SAE_J2012-DA_VOBD_DTCFormat. The DTCMaskRecord shall contain only single DTC values. Group of DTC values are prohibited.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Definition
UserDefDTCSnapshotRecordNumber UserDefDTCSnapshotRecordNumber is a 1-Byte value indicating the number of the specific DTCSnapshot data record requested for a client defined DTCMaskRecord via the reportDTCSnapshotByDTCNumber SubFunction. DTCSnapshot data record number 00_{16} and $F0_{16}$ shall be reserved for legislated purposes (e.g. VOBD). DTCSnapshot records in the range of 01_{16} through EF_{16} and $F1_{16}$ through FE_{16} shall be available for vehicle manufacturer specific usage. A value of FF_{16} requests the server to report all stored DTCSnapshot data records at once.
DTCStoredDataRecordNumber DTCStoredDataRecordNumber is a 1-Byte value indicating the number of the specific DTCStoredDataRecord requested via the reportDTCStoredDataByRecordNumber SubFunction. DTCStoredDataRecordNumber 00_{16} shall be reserved for legislated purposes. DTCStoredData records in range of 01_{16} through FE_{16} shall be available for vehicle manufacturer specific usage. A value of FF_{16} requests the server to report all stored DTCStoredData data records at once.
DTCExtDataRecordNumber
DTCExtDataRecordNumber is a 1-Byte value indicating the number of the specific DTCExtendedData record requested for a client defined DTCMaskRecord via the reportDTCExtDataRecordByDTCNumber and reportDTCExtDataRecordByRecordNumber SubFunction. This parameter is also used by reportDTCExtendedDataRecordIdentification to identify which DTC supports a specific DTCExtendedDataRecordNumber. The DTCExtendedDataRecordNumber ranges are defined in D.8.
DTCSeverityMaskRecord [DTCSeverityMask, DTCStatusMask] DTCSeverityMaskRecord is a 2-Byte value containing the DTCSeverityMask and the DTCStatusMask (see D.3 and D.2).
DTCSeverityMask The DTCSeverityMask contains three DTC severity bits. The definitions for each of the three bits can be found in D.3. This byte is used in the request message to allow a client to request DTC information for the DTCs whose severity definition matches the DTCSeverityMask. A DTCs severity definition matches the DTCSeverityMask if any one of the DTCs actual severity bits is set to '1' and the corresponding severity bit in the DTCSeverityMask is also set to '1' (i.e. if the DTCSeverityMask is bit-wise logically ANDed with the DTCs actual severity and the result is non-zero, then a match has occurred).
FunctionalGroupIdentifier The FunctionalGroupIdentifier has been introduced to distinguish commands sent by the test equipment between different functional system groups within an electrical architecture which consists of many different ECUs. If an ECU has implemented software of the emissions system as well as other systems which may be inspected during an I/M test, it is important that only the DTC information of the requested functional system group is reported. An I/M test should not be failed because another functional system group has DTC information stored. The FunctionalGroupIdentifiers are specified in D.5.
MemorySelection This parameter shall be used to address the respective user defined DTC memory when retrieving DTCs.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Definition
DTCReadinessGroupIdentifier The DTCReadinessGroupIdentifier specifies the reference to the the DTC readiness group and associated DTC(s). The use-case specific documents (e.g. SAE J1979-DA) define the corresponding DTC readiness groups and the FunctionalGroupIdentifier. Each use-case specific document shall specify a FunctionalGroupIdentifier value, e.g. SAE J1979-2 FunctionalGroupIdentifier 33_{16}. The RGID Table specified in SAE J1979-DA depends on the value of the functional group identifier. This means that e. g. for emissions related systems, another table is used than for safety related systems. The table is defined in the related use case specific documents (e. g. SAE J1979-DA for emissions related systems).

12.3.3 Positive response message

12.3.3.1 Positive response message definition

Positive response(s) to the service ReadDTCInformation requests depend on the SubFunction in the service request.

Table 319 specifies the positive response message format of the SubFunction parameter.

Table 319 — Response message definition - SubFunction = reportNumberOfDTCByStatusMask, reportNumberOfDTCBySeverityMaskRecord

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	ReadDTCInformation Response SID	M	59_{16}	RDTCIPR
#2	reportType = [reportNumberOfDTCByStatusMask reportNumberOfDTCBySeverityMaskRecord]	M	 01_{16} 07_{16}	LEV_ RNODTCBSM RNODTCBSMR
#3	DTCStatusAvailabilityMask	M	00_{16} to FF_{16}	DTCSAM
#4	DTCFormatIdentifier = [SAE_J2012-DA_DTCFormat_00 ISO_14229-1_DTCFormat SAE_J1939-73_DTCFormat ISO_11992-4_DTCFormat SAE_J2012-DA_DTCFormat_04]	M	 00_{16} 01_{16} 02_{16} 03_{16} 04_{16}	DTCFID_ J2012-DADTCF00 14229-1DTCF J1939-73DTCF 11992-4DTCF J2012-DADTCF04
 #5 #6	DTCCount[] = [DTCCountHighByte DTCCountLowByte]	 M M	 00_{16} to FF_{16} 00_{16} to FF_{16}	DTCC_ DTCCHB DTCCLB

Table 320 specifies the positive response message format of the SubFunction parameter.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 320 — Response message definition - SubFunction = reportDTCByStatusMask, reportSupportedDTCs, reportFirstTestFailedDTC, reportFirstConfirmedDTC, reportMostRecentTestFailedDTC, reportMostRecentConfirmedDTC, reportDTCWithPermanentStatus

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	ReadDTCInformation Response SID	M	59_{16}	RDTCIPR
#2	reportType = [reportDTCByStatusMask reportSupportedDTCs reportFirstTestFailedDTC reportFirstConfirmedDTC reportMostRecentTestFailedDTC reportMostRecentConfirmedDTC reportDTCWithPermanentStatus]	M	 02_{16} $0A_{16}$ $0B_{16}$ $0C_{16}$ $0D_{16}$ $0E_{16}$ 15_{16}	LEV_ RDTCBSM RSUPDTC RFTFDTC RFCDTC RMRTFDTC RMRCDTC RDTCWPS
#3	DTCStatusAvailabilityMask	M	00_{16} to FF_{16}	DTCSAM
 #4 #5 #6 #7 #8 #9 #10 #11 : #n-3 #n-2 #n-1 #n	DTCAndStatusRecord[] = [DTCHighByte#1 DTCMiddleByte#1 DTCLowByte#1 statusOfDTC#1 DTCHighByte#2 DTCMiddleByte#2 DTCLowByte#2 statusOfDTC#2 : DTCHighByte#m DTCMiddleByte#m DTCLowByte#m statusOfDTC#m]	 C_1 C_1 C_1 C_1 C_2 C_2 C_2 C_2 : C_2 C_2 C_2 C_2	 00_{16} to FF_{16} 00_{16} to FF_{16} 00_{16} to FF_{16} 00_{16} to FF_{16} 00_{16} to FF_{16} 00_{16} to FF_{16} 00_{16} to FF_{16} 00_{16} to FF_{16} : 00_{16} to FF_{16} 00_{16} to FF_{16} 00_{16} to FF_{16} 00_{16} to FF_{16}	DTCASR_ DTCHB DTCMB DTCLB SODTC DTCHB DTCMB DTCLB SODTC : DTCHB DTCMB DTCLB SODTC

C_1: This parameter is only present if DTC information is available to be reported.

C_2: This parameter is only present if reportType = reportSupportedDTCs, reportDTCByStatusMask, reportDTCWithPermanentStatus and more than one DTC information is available to be reported.

Table 321 specifies the positive response message format of the SubFunction parameter.

Table 321 — Response message definition - SubFunction = reportSnapshotIdentification

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	ReadDTCInformation Response SID	M	59_{16}	RDTCIPR
#2	reportType = [reportDTCSnapshotIdentification]	M	 03_{16}	LEV_ RDTCSSI
 #3 #4 #5	DTCRecord[]#1 = [DTCHighByte#1 DTCMiddleByte#1 DTCLowByte#1]	 C_1 C_1 C_1	 00_{16} to FF_{16} 00_{16} to FF_{16} 00_{16} to FF_{16}	DTCASR_ DTCHB DTCMB DTCLB
#6	DTCSnapshotRecordNumber#1	C_1	00_{16} to FF_{16}	DTCSSRN
:	:	:	:	:
 #n-3 #n-2 #n-1	DTCRecord[]#m = [DTCHighByte#m DTCMiddleByte#m DTCLowByte#m]	 C_2 C_2 C_2	 00_{16} to FF_{16} 00_{16} to FF_{16} 00_{16} to FF_{16}	DTCASR_ DTCHB DTCMB DTCLB

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#n	DTCSnapshotRecordNumber#m	C_2	00_{16} to FF_{16}	DTCSSRN
C_1: The DTCRecord and DTCSnapshotRecordNumber parameter is only present if at least one DTCSnapshot record is available to be reported. C_2: The DTCRecord and DTCSnapshotRecordNumber parameter is only present if more than one DTCSnapshot record is available to be reported.				

Table 322 specifies the positive response message format of the SubFunction parameter.

Table 322 — Response message definition - SubFunction = reportDTCSnapshotRecordByDTCNumber

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	ReadDTCInformation Response SID	M	59_{16}	RDTCIPR
#2	reportType = [reportDTCSnapshotRecordByDTCNumber]	M	04_{16}	LEV_ RDTCSSBDTC
	DTCAndStatusRecord[] = [			DTCASR_
#3	DTCHighByte	M	00_{16} to FF_{16}	DTCHB
#4	DTCMiddleByte	M	00_{16} to FF_{16}	DTCMB
#5	DTCLowByte	M	00_{16} to FF_{16}	DTCLB
#6	statusOfDTC]	M	00_{16} to FF_{16}	SODTC
#7	DTCSnapshotRecordNumber#1	C_1	00_{16} to FF_{16}	DTCSSRN
#8	DTCSnapshotRecordNumberOfIdentifiers#1	C_1	00_{16} to FF_{16}	DTCSSRNI
	DTCSnapshotRecord[]#1 = [			DTCSSR_
#9	dataIdentifier#1 byte#1 (MSB)	C_1	00_{16} to FF_{16}	DIDB11
#10	dataIdentifier#1 byte#2 (LSB)	C_1	00_{16} to FF_{16}	DIDB12
#11	snapshotData#1 byte#1	C_1	00_{16} to FF_{16}	SSD11
:	:	C_1	:	:
# 11+(p-1)	snapshotData#1 byte#p	C_1	00_{16} to FF_{16}	SSD1p
:	:	:	:	:
#r-(m-1)-2	dataIdentifier#w byte#1 (MSB)	C_2	00_{16} to FF_{16}	DIDB21
#r-(m-1)-1	dataIdentifier#w byte#2 (LSB)	C_2	00_{16} to FF_{16}	DIDB22
#r-(m-1)	snapshotData#w byte#1	C_2	00_{16} to FF_{16}	SSD21
:	:	C_2	:	:
#r	snapshotData#w byte#m]	C_2	00_{16} to FF_{16}	SSD2m
:	:	:	:	:
#t	DTCSnapshotRecordNumber#x	C_3	00_{16} to FF_{16}	DTCSSRN
#t+1	DTCSnapshotRecordNumberOfIdentifiers#x	C_3	00_{16} to FF_{16}	DTCSSRNI
	DTCSnapshotRecord[]#x = [			DTCSSR_
#t+2	dataIdentifier#1 byte#1 (MSB)	C_3	00_{16} to FF_{16}	DIDB11
#t+3	dataIdentifier#1 byte#2 (LSB)	C_3	00_{16} to FF_{16}	DIDB12
#t+5	snapshotData#1 byte#1	C_3	00_{16} to FF_{16}	SSD11
:	:	C_3	:	:
#t+5+(p-1)	snapshotData#1 byte#p	C_3	00_{16} to FF_{16}	SSD1p
:	:	:	:	:
#n-(u-1)-2	dataIdentifier#w byte#1 (MSB)	C_4	00_{16} to FF_{16}	DIDB21
#n-(u-1)-1	dataIdentifier#w byte#2 (LSB)	C_4	00_{16} to FF_{16}	DIDB22
#n-(u-1)	snapshotData#w byte#1	C_4	00_{16} to FF_{16}	SSD21
:	:	C_4	:	:
#n	snapshotData#w byte#u]	C_4	00_{16} to FF_{16}	SSD2u

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
C_1: The DTCSnapshotRecordNumber and the first dataIdentifier/snapshotData combination in the DTCSnapshotRecord parameter is only present if at least one DTCSnapshot record is available to be reported. C_2/C_4: There are multiple dataIdentifier/snapshotData combinations allowed to be present in a single DTCSnapshotRecord. This can, for example be the case for the situation where a single dataIdentifier only references an integral part of data. When the dataIdentifier references a block of data then a single dataIdentifier/snapshotData combination can be used. C_3: The DTCSnapshotRecordNumber and the first dataIdentifier/snapshotData combination in the DTCSnapshotRecord parameter is only present if all records are requested to be reported (DTCSnapshotRecordNumber set to FF_{16} in the request) and more than one record is available to be reported.				

Table 323 specifies the positive response message format of the SubFunction parameter.

Table 323 — Response message definition - SubFunction = reportDTCStoredDataByRecordNumber

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	ReadDTCInformation Response SID	M	59_{16}	RDTCIPR
#2	reportType = [reportDTCStoredDataByRecordNumber]	M	05_{16}	LEV_ RDTCSDBRN
#3	DTCStoredDataRecordNumber#1	M	00_{16} to FF_{16}	DTCSDRN
	DTCAndStatusRecord[]#1 = [			DTCASR_
#4	DTCHighByte	C_1	00_{16} to FF_{16}	DTCHB
#5	DTCMiddleByte	C_1	00_{16} to FF_{16}	DTCMB
#6	DTCLowByte	C_1	00_{16} to FF_{16}	DTCLB
#7	statusOfDTC]	C_1	00_{16} to FF_{16}	SODTC
#8	DTCStoredDataRecordNumberOfIdentifiers#1	C_1	00_{16} to FF_{16}	DTCSDRNI
	DTCStoredDataRecord[]#1 = [			DTCSDR_
#9	dataIdentifier#1 byte#1 (MSB)	C_1	00_{16} to FF_{16}	DIDB11
#10	dataIdentifier#1 byte#2 (LSB)	C_1	00_{16} to FF_{16}	DIDB12
#11	DTCstoredData#1 byte#1	C_1	00_{16} to FF_{16}	DSD11
:	:	:	:	:
#11+(p-1)	DTCstoredData#1 byte#p	C_1	00_{16} to FF_{16}	DSD1p
:	:	:	:	:
#r-(m-1)-2	dataIdentifier#w byte#1 (MSB)	C_2	00_{16} to FF_{16}	DIDB21
#r-(m-1)-1	dataIdentifier#w byte#2 (LSB)	C_2	00_{16} to FF_{16}	DIDB22
#r-(m-1)	DTCstoredData#w byte#1	C_2	00_{16} to FF_{16}	DSD21
:	:	:	:	:
#r	DTCstoredData#w byte#m]	C_2	00_{16} to FF_{16}	DSD2m
:	:	:	:	:
#t	DTCStoredDataRecordNumber#x	C_3	00_{16} to FF_{16}	DTCSDRN
	DTCAndStatusRecord[]#x = [			DTCASR_
#t+1	DTCHighByte	C_3	00_{16} to FF_{16}	DTCHB
#t+2	DTCMiddleByte	C_3	00_{16} to FF_{16}	DTCMB
#t+3	DTCLowByte	C_3	00_{16} to FF_{16}	DTCLB
#t+4	statusOfDTC]	C_3	00_{16} to FF_{16}	SODTC
#t+5	DTCStoredDataRecordNumberOfIdentifiers#x	C_3	00_{16} to FF_{16}	DTCSDRNI

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
	DTCStoredDataRecord[]#x = [			DTCSDR_
#t+6	dataIdentifier#1 byte#1 (MSB)	C_3	00_{16} to FF_{16}	DIDB11
#t+7	dataIdentifier#1 byte#2 (LSB)	C_3	00_{16} to FF_{16}	DIDB12
#t+8	DTCstoredData#1 byte#1	C_3	00_{16} to FF_{16}	DSD11
:	:	:	:	:
#t+8+(p-1)	DTCstoredDataa#1 byte#p	C_3	00_{16} to FF_{16}	DSD1p
:	:	:	:	:
#n-(u-1)-2	dataIdentifier#w byte#1 (MSB)	C_4	00_{16} to FF_{16}	DIDB21
#n-(u-1)-1	dataIdentifier#w byte#2 (LSB)	C_4	00_{16} to FF_{16}	DIDB22
#n-(u-1)	DTCstoredData#w byte#1	C_4	00_{16} to FF_{16}	DSD21
:	:	:	:	:
#n	DTCstoredData#w byte#u]	C_4	00_{16} to FF_{16}	DSD2u

C_1: The DTCAndStatusRecord and the first dataIdentifier/DTCStoredData combination in the DTCStoredDataRecord parameter is only present if at least one DTCStoredData record is available to be reported.

C_2/C_4: There are multiple dataIdentifier/DTCStoredData combinations allowed to be present in a single DTCStoredDataRecord. This can for example be the case for the situation where a single dataIdentifier only references an integral part of data. When the dataIdentifier references a block of data then a single dataIdentifier/DTCStoredData combination can be used.

C_3: The DTCStoredDataRecordNumber, DTCAndStatusRecord, and the first dataIdentifier/DTCStoredData combination in the DTCStoredDataRecord parameter is only present if all records are requested to be reported (DTCStoredDataRecordNumber set to FF_{16} in the request) and more than one record is available to be reported.

Table 324 specifies the positive response message format of the SubFunction parameter.

Table 324 — Response message definition - SubFunction = reportDTCExtDataRecordByDTCNumber

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	ReadDTCInformation Response SID	M	59_{16}	RDTCIPR
#2	reportType = [reportDTCExtDataRecordByDTCNumber]	M	06_{16}	LEV_ RDTCEDRBD
	DTCAndStatusRecord[] = [			DTCASR_
#3	DTCHighByte	M	00_{16} to FF_{16}	DTCHB
#4	DTCMiddleByte	M	00_{16} to FF_{16}	DTCMB
#5	DTCLowByte	M	00_{16} to FF_{16}	DTCLB
#6	statusOfDTC]	M	00_{16} to FF_{16}	SODTC
#7	DTCExtDataRecordNumber#1	C_1	00_{16} to FD_{16}	DTCEDRN
	DTCExtDataRecord[]#1 = [			DTCSSR_
#8	extendedData#1 byte#1	C_1	00_{16} to FF_{16}	EDD11
:	:	C_1	:	:
#8+(p-1)	extendedData#1 byte#p]	C_1	00_{16} to FF_{16}	EDD1p
:	:	:	:	:
#t	DTCExtDataRecordNumber#x	C_2	00_{16} to FD_{16}	DTCEDRN
	DTCExtDataRecord[]#x = [			DTCSSR_
#t+1	extendedData#x byte#1	C_2	00_{16} to FF_{16}	EDDx1
:	:	C_2	:	:
#t+1+(q-1)	extendedData#x byte#q]	C_2	00_{16} to FF_{16}	EDDxq

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
C_1: The DTCExtDataRecordNumber and the extendedData in the DTCExtDataRecord parameter are only present if at least one DTCExtDataRecord is available to be reported. C_2: The DTCExtDataRecordNumber and the extendedData in the DTCExtDataRecord parameter are only present if all records are requested to be reported (DTCExtDataRecordNumber set to FE_{16} or FF_{16} in the request) and more than one record is available to be reported.				

Table 325 specifies the positive response message format of the SubFunction parameter.

Table 325 — Response message definition - SubFunction = reportDTCBySeverityMaskRecord, reportSeverityInformationOfDTC

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	ReadDTCInformation Response SID	M	59_{16}	RDTCIPR
#2	reportType = [reportDTCBySeverityMaskRecord reportSeverityInformationOfDTC]	M	 08_{16} 09_{16}	LEV_ RDTCBSMR RSIODTC
#3	DTCStatusAvailabilityMask	M	00_{16} to FF_{16}	DTCSAM
	DTCAndSeverityRecord[] = [			DTCASR_
#4	DTCSeverity#1	C_1	00_{16} to FF_{16}	DTCS
#5	DTCFunctionalUnit#1	C_1	00_{16} to FF_{16}	DTCFU
#6	DTCHighByte#1	C_1	00_{16} to FF_{16}	DTCHB
#7	DTCMiddleByte#1	C_1	00_{16} to FF_{16}	DTCMB
#8	DTCLowByte#1	C_1	00_{16} to FF_{16}	DTCLB
#9	statusOfDTC#1	C_1	00_{16} to FF_{16}	SODTC
:	:	:	:	:
#n-5	DTCSeverity#m	C_2	00_{16} to FF_{16}	DTCS
#n-4	DTCFunctionalUnit#m	C_2	00_{16} to FF_{16}	DTCFU
#n-3	DTCHighByte#m	C_2	00_{16} to FF_{16}	DTCHB
#n-2	DTCMiddleByte#m	C_2	00_{16} to FF_{16}	DTCMB
#n-1	DTCLowByte#m	C_2	00_{16} to FF_{16}	DTCLB
#n	statusOfDTC#m]	C_2	00_{16} to FF_{16}	SODTC
C_1: In case of reportDTCBySeverityMaskRecord this parameter shall be present if at least one DTC matches the client defined DTC severity mask. In case of reportSeverityInformationOfDTC this parameter shall be present if the server supports the DTC specified in the request message. C_2: This parameter record is only present if reportType = reportDTCBySeverityMaskRecord. It shall be present if more than one DTC matches the client defined DTC severity mask.				

Table 326 specifies the positive response message format of the SubFunction parameter.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 326 — Response message definition - SubFunction = reportDTCFaultDetectionCounter

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	ReadDTCInformation Response SID	M	59_{16}	RDTCIPR
#2	reportType = [reportDTCFaultDetectionCounter]	M	14_{16}	LEV_ RDTCFDC
	DTCFaultDetectionCounterRecord[] = [			DTCFDCR_
#3	DTCHighByte#1	C_1	00_{16} to FF_{16}	DTCHB
#4	DTCMiddleByte#1	C_1	00_{16} to FF_{16}	DTCMB
#5	DTCLowByte#1	C_1	00_{16} to FF_{16}	DTCLB
#6	DTCFaultDetectionCounter#1	C_1	01_{16} to FF_{16}	DTCFDC
#7	DTCHighByte#2	C_2	00_{16} to FF_{16}	DTCHB
#8	DTCMiddleByte#2	C_2	00_{16} to FF_{16}	DTCLB
#9	DTCLowByte#2	C_2	00_{16} to FF_{16}	DTCFT
#10	DTCFaultDetectionCounter#2	C_2	01_{16} to FF_{16}	DTCFDC
#n-3	:	:	:	:
#n-2	DTCHighByte#m	C_2	00_{16} to FF_{16}	DTCHB
#n-1	DTCMiddleByte#m	C_2	00_{16} to FF_{16}	DTCMB
#n	DTCLowByte#m	C_2	00_{16} to FF_{16}	DTCLB
	DTCFaultDetectionCounter#m]	C_2	01_{16} to FF_{16}	DTCFDC

C_1: This parameter is only present if at least one DTC has a DTCFaultDetectionCounter with a positive value less than $7F_{16}$.

C_2: This parameter record is only present if more than one DTC has a DTCFaultDetectionCounter with a positive value less than $7F_{16}$.

Table 327 specifies the positive response message format of the SubFunction parameter.

Table 327 — Response message definition - SubFunction = reportDTCExtDataRecordByRecordNumber

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	ReadDTCInformation Response SID	M	59_{16}	RDTCIPR
#2	reportType = [reportDTCExtDataRecordByRecordNumber]	M	16_{16}	LEV_ RDTCEDRBR
#3	DTCExtDataRecordNumber	M	00_{16} to EF_{16}	DTCEDRN
	DTCAndStatusRecord[]#1 = [			DTCASR_
#4	DTCHighByte	C_1	00_{16} to FF_{16}	DTCHB
#5	DTCMiddleByte	C_1	00_{16} to FF_{16}	DTCMB
#6	DTCLowByte	C_1	00_{16} to FF_{16}	DTCLB
#7	statusOfDTC]	C_1	00_{16} to FF_{16}	SODTC
	DTCExtDataRecord[]#1= [			DTCEDR_
#8	extendedData#1 byte#1	C_1	00_{16} to FF_{16}	EDD11
:	:	:	:	:
#8+(p-1)	extendedData#1 byte#p]	C_1	00_{16} to FF_{16}	EDD1p
:	:	:	:	:
	DTCAndStatusRecord[]#x = [			DTCSSR
#t	DTCHighByte	C_2	00_{16} to FF_{16}	
#t+1	DTCMiddleByte	C_2	00_{16} to FF_{16}	
#t+2	DTCLowByte	C_2	00_{16} to FF_{16}	
#t+3	statusOfDTC]	C_2	00_{16} to FF_{16}	

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#t+4 : #t+4+(p-1)	DTCExtDataRecord[]#x = [extendedData#x byte#1 : extendedData#x byte#p]	 C2 : C2	 00_{16} to FF_{16} : 00_{16} to FF_{16}	DTCEDR_ EDDx1 : EDDxp
C1: The DTCAndStatusRecord and the DTCExtDataRecord parameters are only present if at least one DTCExtDataRecord is available to be reported. C2: The DTCAndStatusRecord and the DTCExtDataRecord parameters are only present if more than one DTCExtDataRecord is available to be reported. NOTE It is up to the implementer to specify that a response will not exceed a length that it is possible by the used diagnostic communication.				

Table 328 specifies the positive response message format of the SubFunction parameter.

Table 328 — Response message definition - SubFunction = reportUserDefMemoryDTCByStatusMask

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	ReadDTCInformation Response SID	M	59	RDTCIPR
#2	reportType = [reportUserDefMemoryDTCByStatusMask]	M	17	LEV_ RUDMDTCBSM
#3	MemorySelection	M	00_{16} to FF_{16}	MEMYS
#4	DTCStatusAvailabilityMask	M	00_{16} to FF_{16}	DTCSAM
	DTCAndStatusRecord[] = [			DTCASR_
#5	DTCHighByte#1	C1	00_{16} to FF_{16}	DTCHB
#6	DTCMiddleByte#1	C1	00_{16} to FF_{16}	DTCMB
#7	DTCLowByte#1	C1	00_{16} to FF_{16}	DTCLB
#8	statusOfDTC#1	C1	00_{16} to FF_{16}	SODTC
#9	DTCHighByte#2	C2	00_{16} to FF_{16}	DTCHB
#10	DTCMiddleByte#2	C2	00_{16} to FF_{16}	DTCMB
#11	DTCLowByte#2	C2	00_{16} to FF_{16}	DTCLB
#12	statusOfDTC#2	C2	00_{16} to FF_{16}	SODTC
:	:	:	:	:
#n-3	DTCHighByte#m	C2	00_{16} to FF_{16}	DTCHB
#n-2	DTCMiddleByte#m	C2	00_{16} to FF_{16}	DTCMB
#n-1	DTCLowByte#m	C2	00_{16} to FF_{16}	DTCLB
#n	statusOfDTC#m]	C2	00_{16} to FF_{16}	SODTC
C1: This parameter is only present if DTC information is available to be reported. C2: This parameter is only present if more than one DTC information is available to be reported.				

Table 329 specifies the positive response message format of the SubFunction parameter.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 329 — Response message definition - SubFunction = reportUserDefMemoryDTCSnapshotRecordByDTCNumber

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	ReadDTCInformation Response SID	M	59_{16}	RDTCIPR
#2	reportType = [reportUserDefMemoryDTCSnapshotRecordByDTCNumber]	M	18_{16}	LEV_ RUDMDTCSSBDTC
#3	MemorySelection	M	00_{16}-FF_{16}	MEMYS
	DTCAndStatusRecord[] = [			DTCASR_
#4	DTCHighByte	M	00_{16} to FF_{16}	DTCHB
#5	DTCMiddleByte	M	00_{16} to FF_{16}	DTCMB
#6	DTCLowByte	M	00_{16} to FF_{16}	DTCLB
#7	statusOfDTC]	M	00_{16} to FF_{16}	SODTC
#8	UserDefDTCSnapshotRecordNumber#1	C_1	00_{16} to FF_{16}	UDDTCSSRN
#9	DTCSnapshotRecordNumberOfIdentifiers#1	C_1	00_{16} to FF_{16}	DTCSSRNI
	DTCSnapshotRecord[]#1 = [			DTCSSR_
#10	dataIdentifier#1 byte#1 (MSB)	C_1	00_{16} to FF_{16}	DIDB11
#11	dataIdentifier#1 byte#2 (LSB)	C_1	00_{16} to FF_{16}	DIDB12
#12	snapshotData#1 byte#1	C_1	00_{16} to FF_{16}	SSD11
:	:	C_1	:	:
# 12+(p-1)	snapshotData#1 byte#p	C_1	00_{16} to FF_{16}	SSD1p
:	:	:	:	:
#r-(m-1)-2	dataIdentifier#w byte#1 (MSB)	C_2	00_{16} to FF_{16}	DIDB21
#r-(m-1)-1	dataIdentifier#w byte#2 (LSB)	C_2	00_{16} to FF_{16}	DIDB22
#r-(m-1)	snapshotData#w byte#1	C_2	00_{16} to FF_{16}	SSD21
:	:	C_2	:	:
#r	snapshotData#w byte#m]	C_2	00_{16} to FF_{16}	SSD2m
:	:	:	:	:
#t	UserDefDTCSnapshotRecordNumber#x	C_3	00_{16} to FF_{16}	UDDTCSSRN
#t+1	DTCSnapshotRecordNumberOfIdentifiers#x	C_3	00_{16} to FF_{16}	DTCSSRNI
	DTCSnapshotRecord[]#x = [			DTCSSR_
#t+2	dataIdentifier#1 byte#1 (MSB)	C_3	00_{16} to FF_{16}	DIDB11
#t+3	dataIdentifier#1 byte#2 (LSB)	C_3	00_{16} to FF_{16}	DIDB12
#t+5	snapshotData#1 byte#1	C_3	00_{16} to FF_{16}	SSD11
:	:	C_3	:	:
#t+5+(p-1)	snapshotData#1 byte#p	C_3	00_{16} to FF_{16}	SSD1p
:	:	:	:	:
#n-(u-1)-2	dataIdentifier#w byte#1 (MSB)	C_4	00_{16} to FF_{16}	DIDB21
#n-(u-1)-1	dataIdentifier#w byte#2 (LSB)	C_4	00_{16} to FF_{16}	DIDB22
#n-(u-1)	snapshotData#w byte#1	C_4	00_{16} to FF_{16}	SSD21
:	:	C_4	:	:
#n	snapshotData#w byte#u]	C_4	00_{16} to FF_{16}	SSD2u

C_1: The UserDefDTCSnapshotRecordNumber and the first dataIdentifier/snapshotData combination in the DTCSnapshotRecord parameter is only present if at least one DTCSnapshot record is available to be reported.

C_2/C_4: There are multiple dataIdentifier/snapshotData combinations allowed to be present in a single DTCSnapshotRecord. This can for example be the case for the situation where a single dataIdentifier only references an integral part of data. When the dataIdentifier references a block of data then a single dataIdentifier/snapshotData combination can be used.

C_3: The UserDefDTCSnapshotRecordNumber and the first dataIdentifier/snapshotData combination in the DTCSnapshotRecord parameter is only present if all records are requested to be reported (UserDefDTCSnapshotRecordNumber set to FF_{16} in the request) and more than one record is available to be reported.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 330 specifies the positive response message format of the SubFunction parameter.

Table 330 — Response message definition - SubFunction = reportUserDefMemoryDTCExtDataRecordByDTCNumber

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	ReadDTCInformation Response SID	M	59_{16}	RDTCIPR
#2	reportType = [reportUserDefMemoryDTCExtDataRecordByDTCNumber]	M	19_{16}	LEV_ RUDMDTCEDRBDN
#3	MemorySelection	M	00_{16}-FF_{16}	MEMYS
	DTCAndStatusRecord[] = [			DTCASR_
#4	DTCHighByte	M	00_{16} to FF_{16}	DTCHB
#5	DTCMiddleByte	M	00_{16} to FF_{16}	DTCMB
#6	DTCLowByte	M	00_{16} to FF_{16}	DTCLB
#7	statusOfDTC]	M	00_{16} to FF_{16}	SODTC
#8	DTCExtDataRecordNumber#1	C_1	00_{16}-FF_{16}	DTCEDRN
	DTCExtDataRecord[]#1 = [			DTCSSR_
#9	extendedData#1 byte#1	C_1	00_{16} to FF_{16}	EDD11
:	:	C_1	:	:
#9+(p-1)	extendedData#1 byte#p]	C_1	00_{16} to FF_{16}	EDD1p
:	:	:	:	:
	DTCExtDataRecord[]#x = [			DTCSSR_
#t+1	extendedData#x byte#1	C_2	00_{16} to FF_{16}	EDDx1
:	:	C_2	:	:
#t+1+(q-1)	extendedData#x byte#q]	C_2	00_{16} to FF_{16}	EDDxq

C_1: The DTCExtDataRecordNumber and the extendedData in the DTCExtDataRecord parameter are only present if at least one DTCExtDataRecord is available to be reported.

C_2: The DTCExtDataRecordNumber and the extendedData in the DTCExtDataRecord parameter are only present if all records are requested to be reported (DTCExtDataRecordNumber set to FE_{16} or FF_{16} in the request) and more than one record is available to be reported.

Table 331 specifies the positive response message format of the SubFunction parameter.

Table 331 — Response message definition - SubFunction = reportSupportedDTCExtDataRecord

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ReadDTCInformation Response SID	M	59_{16}	RDTCIPR
#2	reportType = [reportSupportedDTCExtDataRecord]	M	$1A_{16}$	LEV_ RDTCEDI
#3	DTCStatusAvailabilityMask	M	00_{16} to FF_{16}	DTCSAM
#4	DTCExtDataRecordNumber	C_1	01_{16} to FD_{16}	DTCEDRN
	DTCAndStatusRecord[]#1 = [			DTCASR_
#5	DTCHighByte#1	C1	00_{16} to FF_{16}	DTCHB
#6	DTCMiddleByte#1	C1	00_{16} to FF_{16}	DTCMB
#7	DTCLowByte#1	C1	00_{16} to FF_{16}	DTCLB
#8	statusOfDTC#1]	C_1	00_{16} to FF_{16}	SODTC
:	:	:	:	:

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
	DTCAndStatusRecord[]#m = [			DTCASR_
#n-3	DTCHighByte#m	C2	00_{16} to FF_{16}	DTCHB
#n-2	DTCMiddleByte#m	C2	00_{16} to FF_{16}	DTCMB
#n-1	DTCLowByte#m	C2	00_{16} to FF_{16}	DTCLB
#n	statusOfDTC#m]	C_2	00_{16} to FF_{16}	SODTC
C_1: This parameter shall be present if at least one DTC supports the DTCExtendedDataRecord				
C_2: This parameter record is only present if more than one DTC support the DTCExtendedDataRecord				

Table 332 specifies the positive response message format of the SubFunction parameter.

Table 332 — Response message definition - SubFunction = reportWWHOBDDTCByMaskRecord

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	ReadDTCInformation Response SID	M	59_{16}	RDTCIPR
#2	reportType = [reportWWHOBDDTCByMaskRecord]	M	42_{16}	LEV_ RWWHOBDDTCBSMR
#3	FunctionalGroupIdentifier	M	00_{16} to FE_{16}	FGID
#4	DTCStatusAvailabilityMask	M	00_{16} to FF_{16}	DTCSAM
#5	DTCSeverityAvailabilityMask	M	00_{16} to FF_{16}	DTCSVAM
#6	DTCFormatIdentifier = [SAE_J2012-DA_DTCFormat_04 SAE_J1939-73_DTCFormat]	M	04_{16} 02_{16}	DTCFID_ J2012-DADTCF04 J1939-73DTCF
	DTCAndSeverityRecord[] = [			DTCASR_
#7	DTCSeverity#1	C_1	00_{16} to FF_{16}	DTCS
#8	DTCHighByte#1 (MSB)	C_1	00_{16} to FF_{16}	DTCHB
#9	DTCMiddleByte#1	C_1	00_{16} to FF_{16}	DTCMB
#10	DTCLowByte#1	C_1	00_{16} to FF_{16}	DTCLB
#11	statusOfDTC#1	C_1	00_{16} to FF_{16}	SODTC
:	:	:	:	:
#n-4	DTCSeverity#m	C_2	00_{16} to FF_{16}	DTCS
#n-3	DTCHighByte#m (MSB)	C_2	00_{16} to FF_{16}	DTCHB
#n-2	DTCMiddleByte#m	C_2	00_{16} to FF_{16}	DTCMB
#n-1	DTCLowByte#m	C_2	00_{16} to FF_{16}	DTCLB
#n	statusOfDTC#m]	C_2	00_{16} to FF_{16}	SODTC
C_1: This parameter is only present if DTC information is available to be reported.				
C_2: This parameter is only present if more than one DTC information is available to be reported.				

Table 333 specifies the positive response message format of the SubFunction parameter.

Table 333 — Response message definition - SubFunction = reportWWHOBDDTCWithPermanentStatus

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	ReadDTCInformation Response SID	M	59_{16}	RDTCIPR
#2	reportType = [reportWWHOBDDTCWithPermanentStatus]	M	55_{16}	LEV_ RWWHOBDDTCWPS
#3	FunctionalGroupIdentifier	M	00_{16} to FE_{16}	FGID
#4	DTCStatusAvailabilityMask	M	00_{16} to FF_{16}	DTCSAM

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#5	DTCFormatIdentifier = [SAE_J2012-DA_DTCFormat_04 SAE_J1939-73_DTCFormat]	M	 04_{16} 02_{16}	DTCFID_ J2012-DADTCF04 J1939-73DTCF
 #6 #7 #8 #9 : #n-3 #n-2 #n-1 #n	DTCAndStatusRecord[] = [DTCHighByte#1 (MSB) DTCMiddleByte#1 DTCLowByte#1 statusOfDTC#1 : DTCHighByte#m (MSB) DTCMiddleByte#m DTCLowByte#m statusOfDTC#m]	 C_1 C_1 C_1 C_1 : C_2 C_2 C_2 C_2	 00_{16} to FF_{16} 00_{16} to FF_{16} 00_{16} to FF_{16} 00_{16} to FF_{16} : 00_{16} to FF_{16} 00_{16} to FF_{16} 00_{16} to FF_{16} 00_{16} to FF_{16}	DTCASR_ DTCHB DTCMB DTCLB SODTC : DTCHB DTCMB DTCLB SODTC
C_1: This parameter is only present if DTC information is available to be reported. C_2: This parameter is only present if more than one DTC information is available to be reported.				

Table 334 specifies the positive response message format of the SubFunction parameter.

Table 334 — Response message definition - SubFunction = reportDTCByReadinessGroupIdentifier

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ReadDTCInformation Response SID	M	59_{16}	RDTCIPR
#2	reportType = [reportDTCByReadinessGroupIdentifier]	M	 56_{16}	LEV_ RDTCBRGI
#3	FunctionalGroupIdentifier	M	00_{16} to FE_{16}	FGID
#4	DTCStatusAvailabilityMask	M	00_{16} to FF_{16}	DTCSAM
#5	DTCFormatIdentifier	M	00_{16} to FF_{16}	DTCFID_
#6	DTCReadinessGroupIdentifier	M	00_{16} to FE_{16}	RGI
 #7 #8 #9 #10 : #n #n-2 #n-1 #n	DTCAndStatusRecord[] = [DTCHighByte#1 DTCMiddleByte#1 DTCLowByte#1 statusOfDTC#1 : DTCHighByte#m DTCMiddleByte#m DTCLowByte#m statusOfDTC#m]	 C_1 C_1 C_1 C_1 : C_2 C_2 C_2 C_2	 00_{16} to FF_{16} 00_{16} to FF_{16} 00_{16} to FF_{16} 00_{16} to FF_{16} : 00_{16} to FF_{16} 00_{16} to FF_{16} 00_{16} to FF_{16} 00_{16} to FF_{16}	DTCASR_ DTCHB DTCMB DTCLB SODTC : DTCHB DTCMB DTCLB SODTC
C_1: This parameter shall be present if at least one DTC matches the client defined Readiness Group Identifier. C_2: This parameter record is only present if more than one DTC matches the client defined Readines Group Identifier				

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

12.3.3.2 Positive response message data-parameter definition

Table 335 specifies the data-parameters of the positive response message.

© ISO 2020 – All rights reserved

Table 335 — Response data-parameter definition

Definition
reportType This parameter is an echo of bits 6 - 0 of the SubFunction parameter provided in the request message from the client.
DTCAndSeverityRecord This parameter record contains one or more groupings of DTCSeverity, DTCFunctionalUnit, DTCHighByte, DTCMiddleByte, DTCLowByte, and statusOfDTC if of SAE_J2012-DA_DTCFormat_00, ISO_14229-1_DTCFormat, SAE_J1939-73_DTCFormat (see below for further details), ISO11992-4DTCFormat or SAE_J2012-DA_DTCFormat_04. The DTCSeverity identifies the importance of the failure for the vehicle operation and/or system function and allows to display recommended actions to the driver. The definitions of DTCSeverity can be found in D.3. The DTCFuncitonalUnit is a 1-Byte value which identifies the corresponding basic vehicle/system function which reports the DTC. The definitions of DTCFunctionalUnit are implementation specific and shall be specified in the respective implementation standard. DTCHighByte, DTCMiddleByte and DTCLowByte together represent a unique identification number for a specific diagnostic trouble code supported by a server. The DTCHighByte and DTCMiddleByte represent a circuit or system that is being diagnosed. The DTCLowByte represents the type of fault in the circuit or system (e.g. sensor open circuit, sensor shorted to ground, algorithm-based failure, etc). The definition can be found in ISO°15031-6[17] specification. This parameter record contains one or more groupings of DTCSeverity, DTCFunctionalUnit, SPN (Suspect Parameter Number), FMI (Failure Mode Identifier), and OC (Occurrence Counter) if of SAE_J1939-73_DTCFormat. The SPN, FMI, and OC are defined in SAE J1939[23].
DTCAndStatusRecord This parameter record contains one or more groupings of DTCHighByte, DTCMiddleByte, DTCLowByte and statusOfDTC if of ISO_14229-1_DTCFormat, SAE_J2012-DA_DTCFormat_00, SAE_J1939-73_DTCFormat, SAE_J2012-DA_DTCFormat_04 or ISO_11992-4_DTCFormat. The SAE_J1939-73_DTCFormat supports the SPN (Suspect Parameter Number), FMI (Failure Mode Identifier), and OC (Occurrence Counter) parameters. The SPN, FMI, and OC are defined in SAE J1939. DTCHighByte, DTCMiddleByte and DTCLowByte together represent a unique identification number for a specific diagnostic trouble code supported by a server. The coding of the 3-byte DTC number can either be done: — by using the decoding of the DTCHighByte, DTCMiddleByte and DTCLowByte according to the ISO 15031-6[17] specification. This format is identified by the DTCFormatIdentifier = SAE_J2012-DA_DTCFormat_00, or — by using the decoding of the DTCHighByte, DTCMiddleByte and DTCLowByte according to the ISO 14229-1 specification which does not specify any decoding method and therefore allows a vehicle manufacturer defined decoding method. This format is identified by the DTCFormatIdentifier = ISO_14229-1_DTCFormat, or — by using the decoding of the DTCHighByte, DTCMiddleByte and DTCLowByte according to the SAE J1939-73[24] specification. This format is identified by the DTCFormatIdentifier = SAE_J1939-73_DTCFormat, or — by using the decoding of the DTCHighByte, DTCMiddleByte and DTCLowByte according to the ISO 11992-4[9] specification. This format is identified by the DTCFormatIdentifier = ISO_11992-4_DTCFormat, — by using the decoding of the DTCHighByte, DTCMiddleByte and DTCLowByte according to the ISO 27145-2[21] specification. This format is identified by the DTCFormatIdentifier = SAE_J2012-DA_DTCFormat_04.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Definition
DTCRecord This parameter record contains one or more groupings of DTCHighByte, DTCMiddleByte, and DTCLowByte. The interpretation of the DTCRecord depends on the value included in the DTCFormatIdentifier parameter as defined in this table.
StatusOfDTC The status of a particular DTC (e.g. test failed this operation cycle, etc). The definition of the bits contained in the statusOfDTC byte can be found in D.2 of this specification. Bits that are not supported by the server shall be reported as '0'.
DTCStatusAvailabilityMask A byte whose bits are defined the same as statusOfDTC and represents the status bits that are supported by the server. Bits that are not supported by the server shall be set to '0'. Each supported bit (indicated by a value of '1') shall be implemented for every DTC supported by the server.
DTCFormatIdentifier This 1-byte parameter value specifies the format of a DTC reported by the server. — SAE_J2012-DA_DTCFormat_00: This parameter value identifies the DTC format reported by the server as defined in ISO 15031-6[17] specification. — ISO_14229-1_DTCFormat: This parameter value identifies the DTC format reported by the server as defined in this table by the parameter DTCAndStatusRecord. — SAE_J1939-73_DTCFormat: This parameter value identifies the DTC format reported by the server as defined in SAE J1939-73[24]. — ISO_11992-4_DTCFormat: This parameter value identifies the DTC format reported by the server as defined in ISO 11992-4[9] specification. — SAE_J2012-DA_DTCFormat_04: This parameter value identifies the DTC format reported by the server as defined in ISO 27145-2[21] specification. The definition of the Byte values contained in the DTCFormatIdentifier byte can be found in D.4. A given server shall support only one DTCFormatIdentifier.
DTCCount This 2-byte parameter refers collectively to the DTCCountHighByte and DTCCountLowByte parameters that are sent in response to a reportNumberOfDTCByStatusMask request. DTCCount provides a count of the number of DTCs that match the DTCStatusMask defined in the client's request.
UserDefDTCSnapshotRecordNumber Either the echo of the UserDefDTCSnapshotRecordNumber parameter specified by the client in the reportUserDefMemoryDTCSnapshotRecordByDTCNumber request, or the actual UserDefDTCSnapshotRecordNumber of a stored DTCSnapshot record.
DTCSnapshotRecordNumberOfIdentifiers This single byte parameter shows the number of dataIdentifiers immediately following the DTCSnapshotRecord. A value of 00_{16} shall be used to indicate that an undefined number of dataIdentifiers are included in the corresponding DTCSnapshotRecord (e.g. primary use case is when the DTCSnapshotRecord contains more than 255 dataIdentifiers).
DTCSnapshotRecord The DTCSnapshotRecord contains a snapshot of data values from the time of the system malfunction occurrence.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Definition
DTCStoredDataRecord The DTCStoredDataRecord contains a freeze frame of data values from the time of the system malfunction occurrence.
DTCStoredDataRecordNumber Either the echo of the DTCStoredDataRecordNumber parameter specified by the client in the reportDTCStoredDataByRecordNumber request, or the actual DTCStoredDataRecordNumber of a stored DTCStoredDataRecord.
DTCStoredDataRecordNumberOfIdentifiers This single byte parameter shows the number of dataIdentifiers in the immediately following DTCStoredDataRecord.
DTCExtDataRecordNumber Either the echo of the DTCExtDataRecordNumber parameter specified by the client in the reportDTCExtDataRecordByDTCNumber, reportDTCExtendedDataRecordIdentification or reportDTCExtDataRecordByRecordNumber request, or the actual DTCExtDataRecordNumber of a stored DTCExtendedData record.
DTCExtDataRecord The DTCExtDataRecord is a server specific block of information that may contain extended status information associated with a DTC. DTCExtendedData contains DTC parameter values, which have been identified at the time of the request.
DTCFaultDetectionCounterRecord The DTCFaultDetectionCounterRecord is a record including one or multiple DTC numbers and the DTC specific DTCFaultDetectionCounter parameter value.
DTCFaultDetectionCounter The DTCFaultDetectionCounter reports the number of fault detection counts of a DTC.
DTCReadinessGroupIdentifier The DTCReadinessGroupIdentifier is used to request and report one or multiple DTCAndStatusRecord(s) for a DTC readiness group.
FunctionalGroupIdentifier A one byte identifier which contains the functional system group the DTC(s) are related to, for example Brakes, Emissions, Occupant Restraints, Tire Inflation, Forward/External lighting, etc. The values are defined in D.5.
DTCSeverityAvailabilityMask A byte whose bits are defined the same as the DTCSeverity and represents the severity bits that are supported by the server. Bits that are not supported by the server shall be set to '0'.
MemorySelection This parameter is an echo of the MemorySelection parameter provided in the request message from the client.

12.3.4 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 336. The listed negative responses shall be used if the error scenario applies to the server.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 336 — 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
31_{16}	**requestOutOfRange**	ROOR
	This NRC shall be sent if: — the client specified a DTCMaskRecord that was not recognized by the server; — the client specified an invalid DTCSnapshotRecordNumber/DTCExtDataRecordNumber/ UserDefDTCSnapshotRecordNumber. This is to be differentiated from the case where the DTCSnapshotRecordNumber and DTCMaskRecord or UserDefDTCSnapshotRecordNumber and DTCMaskRecord combination or the DTCExtDataRecordNumber and DTCMaskRecord combination is supported by the server, but no data is currently associated with it (i.e. positive response required with no data); — the client specified a FunctionalGroupIdentifier that was not recognized by the server; — the client specified a DTCReadinessGroupIdentifier that was not recognized by the server; — the client specified the DTCMaskRecord with a groupOfDTC parameter value; — the MemorySelection identifier was not recognized by the server.	

12.3.5 Message flow examples – ReadDTCInformation

12.3.5.1 General asumption

For all examples the client requests to have a response message by setting the suppressPosRspMsgIndicationBit (bit 7 of the SubFunction parameter) to "FALSE" ('0').

12.3.5.2 Example #1 - ReadDTCInformation, SubFunction = reportNumberOfDTCByStatusMask

12.3.5.2.1 Example #1 overview

This example demonstrates the usage of the reportNumberOfDTCByStatusMask SubFunction parameter for confirmed DTCs (DTC status mask 08_{16}), as well as various masking principles. The DTCStatusAvailabilityMask for this sever = $2F_{16}$.

12.3.5.2.2 Example #1 assumptions

The server supports a total of three DTCs (for the sake of simplicity), which have the following states at the time of the client request.

The following assumptions apply to DTC P0805-11 Clutch Position Sensor - circuit short to ground (080511_{16}), statusOfDTC 24_{16} ($0010\ 0100_2$).

Table 337 specifies the statusOfDTC = 24_{16} of DTC P0805-11.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

Table 337 — statusOfDTC = 24_{16} of DTC P0805-11

statusOfDTC: bit field name	Bit #	Bit state	Description
testFailed	0	0	DTC is no longer failed at the time of the request
testFailedThisOperationCycle	1	0	DTC failed during the current operation cycle
pendingDTC	2	1	DTC failed on the current or previous operation cycle
confirmedDTC	3	0	DTC is not confirmed at the time of the request
testNotCompletedSinceLastClear	4	0	DTC test was completed since the last code clear
testFailedSinceLastClear	5	1	DTC test failed at least once since last code clear
testNotCompletedThisOperationCycle	6	0	DTC test completed this operation cycle
warningIndicatorRequested	7	0	Server is not requesting warningIndicator to be active

The following assumptions apply to DTC P0A9B-17 Hybrid Battery Temperature Sensor - circuit voltage above threshold ($0A9B17_{16}$), statusOfDTC of 26_{16} ($0010\ 0110_2$).

Table 338 specifies the statusOfDTC = 26_{16} of DTC P0A9B-17.

Table 338 — statusOfDTC = 26_{16} of DTC P0A9B-17

statusOfDTC: bit field name	Bit #	Bit state	Description
testFailed	0	0	DTC is no longer failed at the time of the request
testFailedThisOperationCycle	1	1	DTC failed on the current operation cycle
pendingDTC	2	1	DTC failed on the current or previous operation cycle
confirmedDTC	3	0	DTC is not confirmed at the time of the request
testNotCompletedSinceLastClear	4	0	DTC test was completed since the last code clear
testFailedSinceLastClear	5	1	DTC test failed at least once since last code clear
testNotCompletedThisOperationCycle	6	0	DTC test completed this operation cycle
warningIndicatorRequested	7	0	Server is not requesting warningIndicator to be active

The following assumptions apply to DTC P2522-1F A/C Request "B" - circuit intermittent ($25221F_{16}$), statusOfDTC of $2F_{16}$ ($0010\ 1111_2$).

Table 339 specifies the statusOfDTC = $2F_{16}$ of DTC P2522-1F.

Table 339 — statusOfDTC = $2F_{16}$ of DTC P2522-1F

statusOfDTC: bit field name	Bit #	Bit state	Description
testFailed	0	1	DTC failed at the time of the request
testFailedThisOperationCycle	1	1	DTC failed on the current operation cycle
pendingDTC	2	1	DTC failed on the current or previous operation cycle
confirmedDTC	3	1	DTC is confirmed at the time of the request
testNotCompletedSinceLastClear	4	0	DTC test was completed since the last code clear
testFailedSinceLastClear	5	1	DTC test failed at least once since last code clear
testNotCompletedThisOperationCycle	6	0	DTC test completed this operation cycle

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

statusOfDTC: bit field name	Bit #	Bit state	Description
warningIndicatorRequested	7	0	Server is not requesting warningIndicator to be active

12.3.5.2.3 Example #1 message flow

In the following example, a count of one is returned to the client because only DTC P2522-1F A/C Request "B" - circuit intermittent ($25221F_{16}$), statusOfDTC of $2F_{16}$ ($0010\ 1111_2$) matches the client defined status mask of 08_{16} ($0000\ 1000_2$).

Table 340 specifies the ReadDTCInformation, SubFunction = reportNumberOfDTCByStatusMask, request message flow example #1.

Table 340 — ReadDTCInformation, SubFunction = reportNumberOfDTCByStatusMask, request message flow example #1

Message direction	client → server		
Message type	Request		
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	ReadDTCInformation Request SID	19_{16}	RDTCI
#2	SubFunction = reportNumberOfDTCByStatusMask, suppressPosRspMsgIndicationBit = FALSE	01_{16}	RNODTCBSM
#3	DTCStatusMask	08_{16}	DTCSM

Table 341 specifies the ReadDTCInformation, SubFunction = reportNumberOfDTCByStatusMask, positive response, example #1.

Table 341 — ReadDTCInformation, SubFunction = reportNumberOfDTCByStatusMask, positive response, example #1

Message direction	server → client		
Message type	Response		
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	ReadDTCInformation Response SID	59_{16}	RDTCIPR
#2	reportType = reportNumberOfDTCByStatusMask	01_{16}	RNODTCBSM
#3	DTCStatusAvailabilityMask	$2F_{16}$	DTCSAM
#4	DTCFormatIdentifier = ISO_14229-1_DTCFormat	01_{16}	14229-1DTCF
#5	DTCCount [DTCCountHighByte]	00_{16}	DTCCHB
#6	DTCCount [DTCCountLowByte]	01_{16}	DTCCLB

12.3.5.3 Example #2 - ReadDTCInformation, SubFunction = reportDTCByStatusMask, matching DTCs returned

12.3.5.3.1 Example #2 overview

This example demonstrates usage of the reportDTCByStatusMask SubFunction parameter, as well as various masking principles in conjunction with unsupported masking bits. This example also applies to the SubFunction parameter reportUserDefMemoryDTCByStatusMask, except that the status mask checks are performed with the DTCs stored in the user defined memory.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

12.3.5.3.2 Example #2 assumptions

The server supports all status bits for masking purposes, except for bit 7 “warningIndicatorRequested”.

The server supports a total of three DTCs (for the sake of simplicity), which have the following states at the time of the client request.

The following assumptions apply to DTC P0A9B-17 Hybrid Battery Temperature Sensor - circuit voltage above threshold ($0A9B17_{16}$), statusOfDTC 24_{16} ($0010\ 0100_2$).

Table 342 specifies the statusOfDTC= 24_{16} of DTC P0A9B-17.

Table 342 — statusOfDTC= 24_{16} of DTC P0A9B-17

statusOfDTC: bit field name	Bit #	Bit state	Description
testFailed	0	0	DTC is no longer failed at the time of the request
testFailedThisOperationCycle	1	0	DTC failed during the current operation cycle
pendingDTC	2	1	DTC failed on the current or previous operation cycle
confirmedDTC	3	0	DTC is not confirmed at the time of the request
testNotCompletedSinceLastClear	4	0	DTC test was completed since the last code clear
testFailedSinceLastClear	5	1	DTC test failed at least once since last code clear
testNotCompletedThisOperationCycle	6	0	DTC test completed this operation cycle
warningIndicatorRequested	7	0	Server is not requesting warningIndicator to be active

The following assumptions apply to DTC P2522-1F A/C Request “B” - circuit intermittent ($25221F_{16}$), statusOfDTC of 00_{16} ($0000\ 0000_2$).

Table 343 specifies the statusOfDTC = 00_{16} of DTC P2522-1F.

Table 343 — statusOfDTC = 00_{16} of DTC P2522-1F

statusOfDTC: bit field name	Bit #	Bit state	Description
testFailed	0	0	DTC is not failed at the time of the request
testFailedThisOperationCycle	1	0	DTC failed during the current operation cycle
pendingDTC	2	0	DTC was not failed on the current or previous operation cycle
confirmedDTC	3	0	DTC is not confirmed at the time of the request
testNotCompletedSinceLastClear	4	0	DTC test was completed since the last code clear
testFailedSinceLastClear	5	0	DTC test never failed since last code clear
testNotCompletedThisOperationCycle	6	0	DTC test completed this operation cycle
warningIndicatorRequested	7	0	Server is not requesting warningIndicator to be active

The following assumptions apply to DTC P0805-11 Clutch Position Sensor - circuit short to ground (080511_{16}), statusOfDTC of $2F_{16}$ ($0010\ 1111_2$).

Table 344 specifies the statusOfDTC = $2F_{16}$ of DTC P0805-11.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 344 — statusOfDTC = $2F_{16}$ of DTC P0805-11

statusOfDTC: bit field name	Bit #	Bit state	Description
testFailed	0	1	DTC is failed at the time of the request
testFailedThisOperationCycle	1	1	DTC failed on the current operation cycle
pendingDTC	2	1	DTC failed on the current or previous operation cycle
confirmedDTC	3	1	DTC is confirmed at the time of the request
testNotCompletedSinceLastClear	4	0	DTC test was completed since the last code clear
testFailedSinceLastClear	5	1	DTC test failed at least once since last code clear
testNotCompletedThisOperationCycle	6	0	DTC test completed this operation cycle
warningIndicatorRequested	7	0	Server is not requesting warningIndicator to be active

12.3.5.3.3 Example #2 message flow

In the following example, DTCs P0A9B-17 ($0A9B17_{16}$) and P0805-11 (080511_{16}) are returned to the client's request. DTC P2522-1F ($25221F_{16}$) is not returned because its status of 00_{16} does not match the DTCStatusMask of 84_{16} (as specified in the client request messsage in the following example). The server shall bypass masking on those status bits it does not support.

Table 345 specifies the ReadDTCInformation, SubFunction = reportDTCByStatusMask, request message flow example #2.

Table 345 — ReadDTCInformation, SubFunction = reportDTCByStatusMask, request message flow example #2

Message direction	client → server		
Message type	Request		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadDTCInformation Request SID	19_{16}	RDTCI
#2	SubFunction = reportDTCByStatusMask, suppressPosRspMsgIndicationBit = FALSE	02_{16}	RDTCBSM
#3	DTCStatusMask	84_{16}	DTCSM

Table 346 specifies the ReadDTCInformation, SubFunction = reportDTCByStatusMask, positive response, example #2.

Table 346 — ReadDTCInformation, SubFunction = reportDTCByStatusMask, positive response, example #2

Message direction	server → client		
Message type	Response		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadDTCInformation Response SID	59_{16}	RDTCIPR
#2	reportType = reportDTCByStatusMask	02_{16}	RDTCBSM
#3	DTCStatusAvailabilityMask	$7F_{16}$	DTCSAM
#4	DTCAndStatusRecord#1 [DTCHighByte]	$0A_{16}$	DTCHB

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Message direction	server → client		
Message type	Response		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#5	DTCAndStatusRecord#1 [DTCMiddleByte]	$9B_{16}$	DTCMB
#6	DTCAndStatusRecord#1 [DTCLowByte]	17_{16}	DTCLB
#7	DTCAndStatusRecord#1 [statusOfDTC]	24_{16}	SODTC
#8	DTCAndStatusRecord#2 [DTCHighByte]	08_{16}	DTCHB
#9	DTCAndStatusRecord#2 [DTCMiddleByte]	05_{16}	DTCMB
#10	DTCAndStatusRecord#2 [DTCLowByte]	11_{16}	DTCLB
#11	DTCAndStatusRecord#2 [statusOfDTC]	$2F_{16}$	SODTC

12.3.5.4 Example #3 - ReadDTCInformation, SubFunction = reportDTCByStatusMask, no matching DTCs returned

12.3.5.4.1 Example #3 overview

This example demonstrates usage of the reportDTCByStatusMask SubFunction parameter, in the situation where no DTCs match the client defined DTCStatusMask.

12.3.5.4.2 Example #3 assumptions

The server supports all status bits for masking purposes, except for bit 7 “warningIndicatorRequested”. The server supports a total of two DTCs (for the sake of simplicity), which have the following states at the time of the client request.

The following assumptions apply to DTC P2522-1F A/C Request “B” - circuit intermittent ($25221F_{16}$), statusOfDTC 24_{16} ($0010\ 0100_2$).

Table 347 specifies the statusOfDTC= 24_{16} of DTC P2522-1F.

Table 347 — statusOfDTC= 24_{16} of DTC P2522-1F

statusOfDTC: bit field name	Bit #	Bit state	Description
testFailed	0	0	DTC is no longer failed at the time of the request
testFailedThisOperationCycle	1	0	DTC failed during the current operation cycle
pendingDTC	2	1	DTC failed on the current or previous operation cycle
confirmedDTC	3	0	DTC is not confirmed at the time of the request
testNotCompletedSinceLastClear	4	0	DTC test was completed since the last code clear
testFailedSinceLastClear	5	1	DTC test failed at least once since last code clear
testNotCompletedThisOperationCycle	6	0	DTC test completed this operation cycle
warningIndicatorRequested	7	0	Server is not requesting warningIndicator to be active

The following assumptions apply to DTC P0A9B-17 Hybrid Battery Temperature Sensor - circuit voltage above threshold ($0A9B17_{16}$), statusOfDTC of 00_{16} ($0000\ 0000_2$).

Table 348 specifies the statusOfDTC = 00_{16} of DTC P0A9B-17.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 348 — statusOfDTC = 00_{16} of DTC P0A9B-17

statusOfDTC: bit field name	Bit #	Bit state	Description
testFailed	0	0	DTC is not failed at the time of the request
testFailedThisOperationCycle	1	0	DTC failed during the current operation cycle
pendingDTC	2	0	DTC was not failed on the current or previous operation cycle
confirmedDTC	3	0	DTC is not confirmed at the time of the request
testNotCompletedSinceLastClear	4	0	DTC test was completed since the last code clear
testFailedSinceLastClear	5	0	DTC test never failed since last code clear
testNotCompletedThisOperationCycle	6	0	DTC test completed this operation cycle
warningIndicatorRequested	7	0	Server is not requesting warningIndicator to be active

The client requests the server to reportByStatusMask all DTCs having bit 0 (TestFailed) set to logical '1'.

12.3.5.4.3 Example #3 message flow

In the following example, none of the above DTCs are returned to the client's request because none of the DTCs has failed the test at the time of the request.

Table 349 specifies the ReadDTCInformation, SubFunction = reportDTCByStatusMask, request message flow example #3.

Table 349 — ReadDTCInformation, SubFunction = reportDTCByStatusMask, request message flow example #3

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadDTCInformation Request SID	19_{16}	RDTCI
#2	SubFunction = reportDTCByStatusMask, suppressPosRspMsgIndicationBit = FALSE	02_{16}	RDTCBSM
#3	DTCStatusMask	01_{16}	DTCSM

Table 350 specifies the ReadDTCInformation, SubFunction = reportDTCByStatusMask, positive response, example #3.

Table 350 — ReadDTCInformation, SubFunction = reportDTCByStatusMask, positive response, example #3

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadDTCInformation Response SID	59_{16}	RDTCIPR
#2	reportType = reportDTCByStatusMask	02_{16}	RDTCBSM
#3	DTCStatusAvailabilityMask	$7F_{16}$	DTCSAM

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

12.3.5.5 Example #4 - ReadDTCInformation, SubFunction = reportDTCSnapshotIdentification

12.3.5.5.1 Example #4 overview

This example demonstrates the usage of the reportDTCSnapshotIdentification SubFunction parameter.

12.3.5.5.2 Example #4 assumptions

The following assumptions apply:

— The server supports the ability to store two DTCSnapshot records for a given DTC.

— The server shall indicate that two DTCSnapshot records are currently stored for DTC number 123456_{16}. For the purpose of this example, assume that this DTC had occurred three times (such that only the first and most recent DTCSnapshot records are stored because of lack of storage space within the server).

— The server shall indicate that one DTCSnapshot record is currently stored for DTC number $789ABC_{16}$.

— All DTCSnapshot records are stored in ascending order.

12.3.5.5.3 Example #4 message flow

In the following example, three DTCSnapshot records are returned to the client's request.

Table 351 specifies the ReadDTCInformation, SubFunction = reportDTCSnapshotIdentification, request message flow example #4.

Table 351 — ReadDTCInformation, SubFunction = reportDTCSnapshotIdentification, request message flow example #4

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadDTCInformation Request SID	19_{16}	RDTCI
#2	SubFunction = reportDTCSnapshotIdentification, suppressPosRspMsgIndicationBit = FALSE	03_{16}	RDTCSSI

Table 352 specifies the ReadDTCInformation, SubFunction = reportDTCSnapshotIdentification, positive response, example #4.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 352 — ReadDTCInformation, SubFunction = reportDTCSnapshotIdentification, positive response, example #4

Message direction		server → client	
Message type		Response	
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	ReadDTCInformation Response SID	59_{16}	RDTCIPR
#2	reportType = reportDTCSnapshotIdentification	03_{16}	RDTCSSI
#3	DTCAndStatusRecord#1 [DTCHighByte]	12_{16}	DTCHB
#4	DTCAndStatusRecord#1 [DTCMiddleByte]	34_{16}	DTCMB
#5	DTCAndStatusRecord#1 [DTCLowByte]	56_{16}	DTCLB
#6	DTCSnapshotRecordNumber#1	01_{16}	DTCEDRC
#7	DTCAndStatusRecord#2 [DTCHighByte]	12_{16}	DTCHB
#8	DTCAndStatusRecord#2 [DTCMiddleByte]	34_{16}	DTCMB
#9	DTCAndStatusRecord#2 [DTCLowByte]	56_{16}	DTCLB
#10	DTCSnapshotRecordNumber#2	02_{16}	DTCEDRC
#11	DTCAndStatusRecord#3 [DTCHighByte]	78_{16}	DTCHB
#12	DTCAndStatusRecord#3 [DTCMiddleByte]	$9A_{16}$	DTCMB
#13	DTCAndStatusRecord#3 [DTCLowByte]	BC_{16}	DTCLB
#14	DTCSnapshotRecordNumber#3	01_{16}	DTCEDRC

12.3.5.6 Example #5 - ReadDTCInformation, SubFunction = reportDTCSnapshotRecordByDTCNumber

12.3.5.6.1 Example #5 overview

This example demonstrates the usage of the reportDTCSnapshotRecordByDTCNumber SubFunction parameter. This example also applies to the SubFunction parameter reportUserDefMemoryDTCSnapshotRecordByDTCNumber, except that the checks are performed with the DTCs stored in the user defined memory.

12.3.5.6.2 Example #5 assumptions

The following assumptions apply:

— The server supports the ability to store two DTCSnapshot records for a given DTC.

— This example assumes a continuation of the previous example.

— Assume that the server requests the second of the two DTCSnapshot records stored by the server for DTC number 123456_{16} (see previous example, where a DTCSnapshotRecordCount of 02_{16} is returned to the client).

— Assume that DTC 123456_{16} has a statusOfDTC of 24_{16}, and that the following environment data is captured each time a DTC occurs.

— The DTCSnapshot record data is referenced via the dataIdentifier 4711_{16}.

Table 353 specifies the DTCSnapshot record content.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

Table 353 — DTCSnapshot record content

Data byte	DTCSnapshot Record Contents	Byte value
#1	DTCSnapshotRecord [data#1] = ECT (Engine Coolant Temperature)	$A6_{16}$
#2	DTCSnapshotRecord [data#2] = TP (Throttle Position)	66_{16}
#3	DTCSnapshotRecord [data#3] = RPM (Engine Speed)	07_{16}
#4	DTCSnapshotRecord [data#4] = RPM (Engine Speed)	50_{16}
#5	DTCSnapshotRecord [data#5] = MAP (Manifold Absolute Pressure)	20_{16}

12.3.5.6.3 Example #5 message flow

In the following example, one DTCSnapshot record is returned in accordance to the client's reportDTCSnapshotRecordByDTCNumber request.

Table 354 specifies the ReadDTCInformation, SubFunction = reportDTCSnapshotRecordByDTCNumber, request message flow example #5.

Table 354 — ReadDTCInformation, SubFunction = reportDTCSnapshotRecordByDTCNumber, request message flow example #5

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadDTCInformation Request SID	19_{16}	RDTCI
#2	SubFunction = reportDTCSnapshotRecordByDTCNumber, suppressPosRspMsgIndicationBit = FALSE	04_{16}	RDTCSSRBDN
#3	DTCMaskRecord [DTCHighByte]	12_{16}	DTCHB
#4	DTCMaskRecord [DTCMiddleByte]	34_{16}	DTCMB
#5	DTCMaskRecord [DTCLowByte]	56_{16}	DTCLB
#6	DTCSnapshotRecordNumber	02_{16}	DTCSSRN

Table 355 specifies the ReadDTCInformation, SubFunction = reportDTCSnapshotRecordByDTCNumber, positive response, example #5.

Table 355 — ReadDTCInformation, SubFunction = reportDTCSnapshotRecordByDTCNumber, positive response, example #5

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadDTCInformation Response SID	59_{16}	RDTCIPR
#2	reportType = reportDTCSnapshotRecordByDTCNumber	04_{16}	RDTCSSRBDN
#3	DTCAndStatusRecord [DTCHighByte]	12_{16}	DTCHB
#4	DTCAndStatusRecord [DTCMiddleByte]	34_{16}	DTCMB
#5	DTCAndStatusRecord [DTCLowByte]	56_{16}	DTCLB
#6	DTCAndStatusRecord [statusOfDTC]	24_{16}	SODTC
#7	DTCSnapshotRecordNumber	02_{16}	DTCEDRN

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Message direction	server → client		
Message type	Response		
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#8	DTCSnapshotRecordNumberOfIdentifiers	01_{16}	DTCSSRNI
#9	dataIdentifier [byte#1] (MSB)	47_{16}	DIDB1
#10	dataIdentifier [byte#2] (LSB)	11_{16}	DIDB2
#11	DTCSnapshotRecord [data#1] = ECT	$A6_{16}$	ED_1
#12	DTCSnapshotRecord [data#2] = TP	66_{16}	ED_2
#13	DTCSnapshotRecord [data#3] = RPM	07_{16}	ED_3
#14	DTCSnapshotRecord [data#4] = RPM	50_{16}	ED_4
#15	DTCSnapshotRecord [data#5] = MAP	20_{16}	ED_5

12.3.5.7 Example #6 - ReadDTCInformation, SubFunction = reportDTCStoredDataByRecordNumber

12.3.5.7.1 Example #6 overview

This example demonstrates the usage of the reportDTCStoredDataByRecordNumber SubFunction parameter.

12.3.5.7.2 Example #6 assumptions

The following assumptions apply:

— The server supports the ability to store two DTCStoredDataRecords for a given DTC.

— This example assumes a continuation of the previous example.

— Assume that the server requests the second of the two DTCStoredDataRecords stored by the server for DTC number 123456_{16} (see previous example, where a DTCStoredDataRecordCount of two is returned to the client).

— Assume that DTC 123456_{16} has a statusOfDTC of 24_{16}, and that the following environment data is captured each time a DTC occurs.

— The DTCStoredData record data is referenced via the dataIdentifier 4711_{16}.

Table 356 specifies the DTCStoredData record content.

Table 356 — DTCStoredData record content

Data byte	DTCSnapshot Record Contents	Byte value
#1	DTCStoredDatatRecord [data#1] = ECT (Engine Coolant Temp.)	$A6_{16}$
#2	DTCStoredDataRecord [data#2] = TP (Throttle Position)	66_{16}
#3	DTCStoredDataRecord [data#3] = RPM (Engine Speed)	07_{16}
#4	DTCStoredDataRecord [data#4] = RPM (Engine Speed)	50_{16}
#5	DTCStoredDataRecord [data#5] = MAP (Manifold Absolute Pressure)	20_{16}

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

12.3.5.7.3 Example #6 message flow

In the following example, DTCStoredData record number two is requested and the server returns the DTC and DTCStoredData record content.

Table 357 specifies the ReadDTCInformation, SubFunction = reportDTCStoredDataByRecordNumber, request message flow example #6.

Table 357 — ReadDTCInformation, SubFunction = reportDTCStoredDataByRecordNumber, request message flow example #6

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadDTCInformation Request SID	19_{16}	RDTCI
#2	SubFunction = reportDTCStoredDataByRecordNumber, suppressPosRspMsgIndicationBit = FALSE	05_{16}	RDTCSDRBRN
#3	DTCStoredDataRecordNumber	02_{16}	DTCSDRN

Table 358 specifies the ReadDTCInformation, SubFunction = reportDTCStoredDataByRecordNumber, positive response, example #6.

Table 358 — ReadDTCInformation, SubFunction = reportDTCStoredDataByRecordNumber, positive response, example #6

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadDTCInformation Response SID	59_{16}	RDTCIPR
#2	reportType = reportDTCStoredDataByRecordNumber	05_{16}	RDTCSDRBRN
#3	DTCStoredDataRecordNumber	02_{16}	DTCSDRN
#4	DTCAndStatusRecord [DTCHighByte]	12_{16}	DTCHB
#5	DTCAndStatusRecord [DTCMiddleByte]	34_{16}	DTCMB
#6	DTCAndStatusRecord [DTCLowByte]	56_{16}	DTCLB
#7	DTCAndStatusRecord [statusOfDTC]	24_{16}	SODTC
#8	DTCStoredDataRecordNumberOfIdentifiers	01_{16}	DTCSDRNI
#9	dataIdentifier [byte#1 (MSB)]	47_{16}	DIDB1
#10	dataIdentifier [byte#2] (LSB)	11_{16}	DIDB2
#11	DTCStoredDataRecord [data#1] = ECT	$A6_{16}$	ED_1
#12	DTCStoredDataRecord [data#2] = TP	66_{16}	ED_2
#13	DTCStoredDataRecord [data#3] = RPM	07_{16}	ED_3
#14	DTCStoredDataRecord [data#4] = RPM	50_{16}	ED_4
#15	DTCStoredDataRecord [data#5] = MAP	20_{16}	ED_5

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

12.3.5.8 Example #7 - ReadDTCInformation, SubFunction = reportDTCExtDataRecordByDTCNumber

12.3.5.8.1 Example #7 overview

This example demonstrates the usage of the reportDTCExtDataRecordByDTCNumber SubFunction parameter. This example also applies to the SubFunction parameter reportUserDefMemory-DTCExtDataRecordByDTCNumber, except that the checks are performed with the DTCs stored in the user defined memory.

12.3.5.8.2 Example #7 assumptions

The following assumptions apply:

— The server supports the ability to store two DTCExtendedData records for a given DTC.

— Assume that the server requests all available DTCExtendedData records stored by the server for DTC number 123456_{16}.

— Assume that DTC 123456_{16} has a statusOfDTC of 24_{16}, and that the following extended data is available for the DTC.

— The DTCExtendedData is referenced via the DTCExtDataRecordNumbers 05_{16} and 10_{16} (see Tables 359 and 360).

Table 359 — DTCExtDataRecordNumber 05_{16} content

Data byte	DTCExtDataRecord Contents for DTCExtDataRecordNumber 05_{16}	Byte value
#1	Warm-up Cycle Counter – Number of warm-up cycles since the DTC commanded the MIL to switch off.	17_{16}

Table 360 — DTCExtDataRecordNumber 10_{16} content

Data byte	DTCExtDataRecord Contents for DTCExtDataRecordNumber 10_{16}	Byte value
#1	DTC Fault Detection Counter – Increments each time the DTC test detects a fault, Decrements each time the test reports no fault.	79_{16}

12.3.5.8.3 Example #7 message flow

In the following example, a DTCMaskRecord including the DTC number and a DTCExtDataRecordNumber with the value of FF_{16} (report all DTCExtDataRecords) is requested by the client. The server returns two DTCExtDataRecords which have been recorded for the DTC number submitted by the client.

Table 361 specifies the ReadDTCInformation, SubFunction = reportDTCExtDataRecordByDTCNumber, request message flow example #7.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

Table 361 — ReadDTCInformation, SubFunction = reportDTCExtDataRecordByDTCNumber, request message flow example #7

Message direction	client → server		
Message type	Request		
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	ReadDTCInformation Request SID	19_{16}	RDTCI
#2	SubFunction = reportDTCExtDataRecordByDTCNumber, suppressPosRspMsgIndicationBit = FALSE	06_{16}	RDTCEDRBDN
#3	DTCMaskRecord [DTCHighByte]	12_{16}	DTCHB
#4	DTCMaskRecord [DTCMiddleByte]	34_{16}	DTCMB
#5	DTCMaskRecord [DTCLowByte]	56_{16}	DTCLB
#6	DTCExtDataRecordNumber	FF_{16}	DTCEDRN

Table 362 specifies the ReadDTCInformation, SubFunction = reportDTCExtDataRecordByDTCNumber, positive response, example #7.

Table 362 — ReadDTCInformation, SubFunction = reportDTCExtDataRecordByDTCNumber, positive response, example #7

Message direction	server → client		
Message type	Response		
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	ReadDTCInformation Response SID	59_{16}	RDTCIPR
#2	reportType = reportDTCExtDataRecordByDTCNumber	06_{16}	RDTCEDRBDN
#3	DTCAndStatusRecord [DTCHighByte]	12_{16}	DTCHB
#4	DTCAndStatusRecord [DTCMiddleByte]	34_{16}	DTCMB
#5	DTCAndStatusRecord [DTCLowByte]	56_{16}	DTCLB
#6	DTCAndStatusRecord [statusOfDTC]	24_{16}	SODTC
#7	DTCExtDataRecordNumber	05_{16}	DTCEDRN
#8	DTCExtDataRecord [byte#1]	17_{16}	ED_1
#9	DTCExtDataRecordNumber	10_{16}	DTCEDRN
#10	DTCExtDataRecord [byte#1]	79_{16}	ED_1

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

12.3.5.9 Example #8 - ReadDTCInformation, SubFunction = reportNumberOfDTC-BySeverityMaskRecord

12.3.5.9.1 Example #8 overview

This example demonstrates the usage of the reportNumberOfDTCBySeverityMaskRecord SubFunction parameter.

12.3.5.9.2 Example #8 assumptions

The server supports a total of three DTCs which have the following states at the time of the client request:

© ISO 2020 – All rights reserved

The following assumptions apply to DTC P0A9B-17 Hybrid Battery Temperature Sensor - circuit voltage above threshold ($0A9B17_{16}$), statusOfDTC 24_{16} ($0010\ 0100_2$), DTCFunctionalUnit = 10_{16}, DTCSeverity = 20_{16}.

NOTE 1 Only bit 7 to 5 of the severity byte are valid.

Table 363 specifies the statusOfDTC = 24_{16} of DTC P0A9B-17.

Table 363 — statusOfDTC = 24_{16} of DTC P0A9B-17

statusOfDTC: bit field name	Bit #	Bit state	Description
testFailed	0	0	DTC is no longer failed at the time of the request
testFailedThisOperationCycle	1	0	DTC failed during the current operation cycle
pendingDTC	2	1	DTC failed on the current or previous operation cycle
confirmedDTC	3	0	DTC is not confirmed at the time of the request
testNotCompletedSinceLastClear	4	0	DTC test was completed since the last code clear
testFailedSinceLastClear	5	1	DTC test failed at least once since last code clear
testNotCompletedThisOperationCycle	6	0	DTC test completed this operation cycle
warningIndicatorRequested	7	0	Server is not requesting warningIndicator to be active

The following assumptions apply to DTC P2522-1F A/C Request "B" - circuit intermittent ($25221F_{16}$), statusOfDTC of 00_{16} ($0000\ 0000_2$), DTCFunctionalUnit = 10_{16}, DTCSeverity = 20_{16}.

NOTE 2 Only bit 7 to 5 of the severity byte are valid.

Table 364 specifies the statusOfDTC = 00_{16} of DTC P2522-1F.

Table 364 — statusOfDTC = 00_{16} of DTC P2522-1F

statusOfDTC: bit field name	Bit #	Bit state	Description
testFailed	0	0	DTC is not failed at the time of the request
testFailedThisOperationCycle	1	0	DTC failed during the current operation cycle
pendingDTC	2	0	DTC was not failed on the current or previous operation cycle
confirmedDTC	3	0	DTC is not confirmed at the time of the request
testNotCompletedSinceLastClear	4	0	DTC test was completed since the last code clear
testFailedSinceLastClear	5	0	DTC test never failed since last code clear
testNotCompletedThisOperationCycle	6	0	DTC test completed this operation cycle
warningIndicatorRequested	7	0	Server is not requesting warningIndicator to be active

The following assumptions apply to DTC P0805-11 Clutch Position Sensor - circuit short to ground (080511_{16}), statusOfDTC of $2F_{16}$ ($0010\ 1111_2$), DTCFunctionalUnit = 10_{16}, DTCSeverity = 40_{16}.

NOTE 3 Only bit 7 to 5 of the severity byte are valid.

Table 365 specifies the statusOfDTC = $2F_{16}$ of DTC P0805-11.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

Table 365 — statusOfDTC = $2F_{16}$ of DTC P0805-11

statusOfDTC: bit field name	Bit #	Bit state	Description
testFailed	0	1	DTC is failed at the time of the request
testFailedThisOperationCycle	1	1	DTC failed on the current operation cycle
pendingDTC	2	1	DTC failed on the current or previous operation cycle
confirmedDTC	3	1	DTC is confirmed at the time of the request
testNotCompletedSinceLastClear	4	0	DTC test was completed since the last code clear
testFailedSinceLastClear	5	1	DTC test failed at least once since last code clear
testNotCompletedThisOperationCycle	6	0	DTC test completed this operation cycle
warningIndicatorRequested	7	0	Server is not requesting warningIndicator to be active

The server supports the testFailed and confirmedDTC status bits for masking purposes.

12.3.5.9.3 Example #8 message flow

In the following example, a count of one is returned to the client because DTC P0805-11 (080511_{16}) match the client defined severity mask record of $C001_{16}$ (DTCSeverityMask = $11\ _{16}xxxx_2$ = $C0_{16}$, DTCStatusMask = $0000\ 0001_2$).

Table 366 specifies the ReadDTCInformation, SubFunction = reportNumberOfDTCBySeverityMaskRecord, request message flow example #8.

Table 366 — ReadDTCInformation, SubFunction = reportNumberOfDTCBySeverityMaskRecord, request message flow example #8

Message direction	client → server		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadDTCInformation Request SID	19_{16}	RDTCI
#2	SubFunction = reportNumberOfDTCBySeverityMaskRecord, suppressPosRspMsgIndicationBit = FALSE	07_{16}	RNODTCBSMR
#3	DTCSeverityMaskRecord(DTCSeverityMask)	$C0_{16}$	DTCSVM
#4	DTCSeverityMaskRecord(DTCStatusMask)	01_{16}	DTCSM

Table 367 specifies the ReadDTCInformation, SubFunction = reportNumberOfDTCBySeverityMaskRecord, positive response, positive response, example #8.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 367 — ReadDTCInformation, SubFunction = reportNumberOfDTCBySeverityMaskRecord, positive response, example #8

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadDTCInformation Response SID	59_{16}	RDTCIPR
#2	reportType = reportNumberOfDTCBySeverityMaskRecord	07_{16}	RNODTCBSMR
#3	DTCStatusAvailabilityMask	09_{16}	DTCSAM
#4	DTCFormatIdentifier = ISO_14229-1_DTCFormat	01_{16}	14229-1DTCF
#5	DTCCount [DTCCountHighByte]	00_{16}	DTCCHB
#6	DTCCount [DTCCountLowByte]	01_{16}	DTCCLB

12.3.5.10 Example #9 - ReadDTCInformation, SubFunction = reportDTCBySeverityMaskRecord

12.3.5.10.1 Example #9 overview

This example demonstrates the usage of the reportDTCBySeverityMaskRecord SubFunction parameter.

12.3.5.10.2 Example #9 assumptions

The assumptions defined in 11.3.5.9.2 and those defined in this subclause apply.

In the following example, the DTC P0805-11 (080511_{16}) match the client defined severity mask record of $C001_{16}$ (DTCSeverityMask = $C0_{16}$ = 110x $XXXX_2$, DTCStatusMask = 01_{16}, 0000 0001_2) and is reported to the client. The severity of DTC P0805-11 (080511_{16}) is 40_{16} (010x $XXXX_2$). The server supports all status bits for masking purposes, except for bit 7 "warningIndicatorRequested".

NOTE Only bit 7 to 5 of the severity mask byte are valid.

12.3.5.10.3 Example #9 message flow

In the following example, one DTCSeverityRecord is returned to the client's request.

Table 368 specifies the ReadDTCInformation, SubFunction = reportDTCBySeverityMaskRecord, request message flow example #9.

Table 368 — ReadDTCInformation, SubFunction = reportDTCBySeverityMaskRecord, request message flow example #9

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadDTCInformation Request SID	19_{16}	RDTCI
#2	SubFunction = reportDTCBySeverityMaskRecord, suppressPosRspMsgIndicationBit = FALSE	08_{16}	RDTCBSMR
#3	DTCSeverityMaskRecord(DTCSeverityMask)	$C0_{16}$	DTCSVM
#4	DTCSeverityMaskRecord(DTCStatusMask)	01_{16}	DTCSM

Table 369 specifies the ReadDTCInformation, SubFunction = reportDTCBySeverityMaskRecord, positive response example #9.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

Table 369 — ReadDTCInformation, SubFunction = reportDTCBySeverityMaskRecord, positive response, example #9

Message direction		server → client	
Message type		Response	
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	ReadDTCInformation Response SID	59_{16}	RDTCIPR
#2	reportType = reportDTCBySeverityMaskRecord	08_{16}	RDTCBSMR
#3	DTCStatusAvailabilityMask	$7F_{16}$	DTCSAM
#4	DTCSeverityRecord#1 [DTCSeverity]	40_{16}	DTCS
#5	DTCSeverityRecord#1 [DTCFunctionalUnit]	10_{16}	DTCFU
#6	DTCSeverityRecord#1 [DTCHighByte]	08_{16}	DTCHB
#7	DTCSeverityRecord#1 [DTCMiddleByte]	05_{16}	DTCMB
#8	DTCSeverityRecord#1 [DTCLowByte]	11_{16}	DTCLB
#9	DTCSeverityRecord#1 [statusOfDTC]	$2F_{16}$	SODTC

12.3.5.11 Example #10 - ReadDTCInformation, SubFunction = reportSeverityInformationOfDTC

12.3.5.11.1 Example #10 overview

This example demonstrates the usage of the reportSeverityInformationOfDTC SubFunction parameter.

12.3.5.11.2 Example #10 assumptions

The assumptions defined in 11.3.5.10.2 apply.

12.3.5.11.3 Example #10 message flow

In the following example, the DTC P0805-11 (080511_{16}), which matches the client defined DTC mask record, is reported to the client.

Table 370 specifies ReadDTCInformation, SubFunction = reportSeverityInformationOfDTC, request message flow example #10.

Table 370 — ReadDTCInformation, SubFunction = reportSeverityInformationOfDTC, request message flow example #10

Message direction		client → server	
Message type		Request	
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	ReadDTCInformation Request SID	19_{16}	RDTCI
#2	SubFunction = reportSeverityInformationOfDTC, suppressPosRspMsgIndicationBit = FALSE	09_{16}	RSIODTC
#3	DTCMaskRecord [DTCHighByte]	08_{16}	DTCHB
#4	DTCMaskRecord [DTCMiddleByte]	05_{16}	DTCMB
#5	DTCMaskRecord [DTCLowByte]	11_{16}	DTCLB

Table 371 specifies ReadDTCInformation, SubFunction = reportSeverityInformationOfDTC, positive response, example #10.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 371 — ReadDTCInformation, SubFunction = reportSeverityInformationOfDTC, positive response, example #10

Message direction	server → client		
Message type	Response		
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	ReadDTCInformation Response SID	59_{16}	RDTCIPR
#2	reportType = reportDTCBySeverityMaskRecord	09_{16}	RSIODTC
#3	DTCStatusAvailabilityMask	$7F_{16}$	DTCSAM
#4	DTCSeverityRecord [DTCSeverity]	40_{16}	DTCS
#5	DTCSeverityRecord [DTCFunctionalUnit]	10_{16}	DTCFU
#6	DTCSeverityRecord [DTCHighByte]	08_{16}	DTCHB
#7	DTCSeverityRecord [DTCMiddleByte]	05_{16}	DTCMB
#8	DTCSeverityRecord [DTCLowByte]	11_{16}	DTCLB
#9	DTCSeverityRecord [statusOfDTC]	$2F_{16}$	SODTC

12.3.5.12 Example #11 – ReadDTCInformation - SubFunction = reportSupportedDTCs

12.3.5.12.1 Example #11 overview

This example demonstrates the usage of the reportSupportedDTCs SubFunction parameter.

12.3.5.12.2 Example #11 assumptions

The assumptions defined in 12.3.5.10.2 apply. Besides the following assumptions apply:

— The server supports a total of three DTCs (for the sake of simplicity), which have the following states at the time of the client request.

— The following assumptions apply to DTC 123456_{16}, statusOfDTC 24_{16} ($0010\ 0100_2$), see Table 372.

Table 372 — statusOfDTC = 24_{16}

statusOfDTC: bit field name	Bit #	Bit state	Description
testFailed	0	0	DTC is not failed at the time of the request
testFailedThisOperationCycle	1	0	DTC failed during the current operation cycle
pendingDTC	2	1	DTC failed on the current or previous operation cycle
confirmedDTC	3	0	DTC was never confirmed
testNotCompletedSinceLastClear	4	0	DTC test were completed since the last code clear
testFailedSinceLastClear	5	1	DTC failed at least once since last code clear
testNotCompletedThisOperationCycle	6	0	DTC test completed this operation cycle
warningIndicatorRequested	7	0	Server is not requesting warningIndicator to be active

The following assumptions apply to DTC 234505_{16}, statusOfDTC of 00_{16} ($0000\ 0000_2$), see Table 373.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 373 — statusOfDTC = 00_{16}

statusOfDTC: bit field name	Bit #	Bit state	Description
testFailed	0	0	DTC is not failed at the time of the request
testFailedThisOperationCycle	1	0	DTC failed during the current operation cycle
pendingDTC	2	0	DTC was not failed on the current or previous operation cycle
confirmedDTC	3	0	DTC is not confirmed at the time of the request
testNotCompletedSinceLastClear	4	0	DTC test were completed since the last code clear
testFailedSinceLastClear	5	0	DTC test never failed since last code clear
testNotCompletedThisOperationCycle	6	0	DTC test completed this operation cycle
warningIndicatorRequested	7	0	Server is not requesting warningIndicator to be active

The following assumptions apply to DTC $ABCD01_{16}$, statusOfDTC of $2F_{16}$ ($0010\ 1111_2$), see Table 374.

Table 374 — statusOfDTC = $2F_{16}$

statusOfDTC: bit field name	Bit #	Bit state	Description
testFailed	0	1	DTC is failed at the time of the request
testFailedThisOperationCycle	1	1	DTC failed on the current operation cycle
pendingDTC	2	1	DTC failed on the current or previous operation cycle
confirmedDTC	3	1	DTC is confirmed at the time of the request
testNotCompletedSinceLastClear	4	0	DTC test were completed since the last code clear
testFailedSinceLastClear	5	1	DTC test failed at least once since last code clear
testNotCompletedThisOperationCycle	6	0	DTC test completed this operation cycle
warningIndicatorRequested	7	0	Server is not requesting warningIndicator to be active

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

12.3.5.12.3 Example #11 message flow

In the following example, all three of the above DTCs are returned to the client's request because all are supported.

Table 375 specifies ReadDTCInformation, SubFunction = reportSupportedDTCs, request message flow example #11.

Table 375 — ReadDTCInformation, SubFunction = reportSupportedDTCs, request message flow example #11

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadDTCInformation Request SID	19_{16}	RDTCI
#2	SubFunction = reportSupportedDTCs, suppressPosRspMsgIndicationBit = FALSE	$0A_{16}$	RSUPDTC

© ISO 2020 – All rights reserved

Table 376 specifies ReadDTCInformation, SubFunction = readSupportedDTCs, positive response, example #11.

Table 376 — ReadDTCInformation, SubFunction = readSupportedDTCs, positive response, example #11

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadDTCInformation Response SID	59_{16}	RDTCIPR
#2	reportType = readSupportedDTCs	$0A_{16}$	RSUPDTC
#3	DTCStatusAvailabilityMask	$7F_{16}$	DTCSAM
#4	DTCAndStatusRecord#1 [DTCHighByte]	12_{16}	DTCHB
#5	DTCAndStatusRecord#1 [DTCMiddleByte]	34_{16}	DTCMB
#6	DTCAndStatusRecord#1 [DTCLowByte]	56_{16}	DTCLB
#7	DTCAndStatusRecord#1 [statusOfDTC]	24_{16}	SODTC
#8	DTCAndStatusRecord#2 [DTCHighByte]	23_{16}	DTCHB
#9	DTCAndStatusRecord#2 [DTCMiddleByte]	45_{16}	DTCMB
#10	DTCAndStatusRecord#2 [DTCLowByte]	05_{16}	DTCLB
#11	DTCAndStatusRecord#2 [statusOfDTC]	00_{16}	SODTC
#12	DTCAndStatusRecord#3 [DTCHighByte]	AB_{16}	DTCHB
#13	DTCAndStatusRecord#3 [DTCMiddleByte]	CD_{16}	DTCMB
#14	DTCAndStatusRecord#3 [DTCLowByte]	01_{16}	DTCLB
#15	DTCAndStatusRecord#3 [statusOfDTC]	$2F_{16}$	SODTC

12.3.5.13 Example #12 - ReadDTCInformation, SubFunction = reportFirstTestFailedDTC, information available

12.3.5.13.1 Example #12 overview

This example demonstrates usage of the reportFirstTestFailedDTC SubFunction parameter, where it is assumed that at least one failed DTC occurred since the last ClearDiagnosticInformation request from the server.

If exactly one DTC failed within the server since the last ClearDiagnosticInformation request from the server, then the server would return the same information in response to a reportMostRecentTestFailedDTC request from the client.

In this example, the status of the DTC returned in response to the reportFirstTestFailedDTC is no longer current at the time of the request (the same phenomenon is possible when requesting the server to report the most recent failed/confirmed DTC).

The general format of request/response messages in the following example is also applicable to SubFunction parameters reportFirstConfirmedDTC, reportMostRecentTestFailedDTC, and reportMostRecent-ConfirmedDTC (for the appropriate DTC status and under similar assumptions).

12.3.5.13.2 Example #12 assumptions

The following assumptions apply:

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

— At least one DTC failed since the last ClearDiagnosticInformation request from the server.

— The server supports all status bits for masking purposes.

— DTC number 123456_{16} = first failed DTC to be detected since the last code clear.

— The following assumptions apply to DTC 123456_{16}, statusOfDTC 26_{16} ($0010\ 0110_2$), see Table 377:

Table 377 — statusOfDTC = 26_{16}

statusOfDTC: bit field name	Bit #	Bit state	Description
testFailed	0	0	DTC is not failed at the time of the request
testFailedThisOperationCycle	1	1	DTC failed during the current operation cycle
pendingDTC	2	1	DTC failed on the current or previous operation cycle
confirmedDTC	3	0	DTC was never confirmed
testNotCompletedSinceLastClear	4	0	DTC test was completed since the last code clear
testFailedSinceLastClear	5	1	DTC failed at least once since last code clear
testNotCompletedThisOperationCycle	6	0	DTC test completed this operation cycle
warningIndicatorRequested	7	0	Server is not requesting warningIndicator to be active

12.3.5.13.3 Example #12 message flow

In the following example DTC 123456_{16} is returned to the client's request, see Table 378.

Table 378 — ReadDTCInformation, SubFunction = reportFirstTestFailedDTC, request message flow example #12

Message direction	client → server		
Message type	Request		
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	ReadDTCInformation Request SID	19_{16}	RDTCI
#2	SubFunction = reportFirstTestFailedDTC, suppressPosRspMsgIndicationBit = FALSE	$0B_{16}$	RFCDTC

Table 379 specifies ReadDTCInformation, SubFunction = reportFirstTestFailedDTC, positive response, example #12.

Table 379 — ReadDTCInformation, SubFunction = reportFirstTestFailedDTC, positive response, example #12

Message direction	server → client		
Message type	Response		
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	ReadDTCInformation Response SID	59_{16}	RDTCIPR
#2	reportType = reportFirstTestFailedDTC	$0B_{16}$	RFCDTC
#3	DTCStatusAvailabilityMask	FF_{16}	DTCSAM

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Message direction	server → client		
Message type	Response		
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#4	DTCAndStatusRecord [DTCHighByte]	12_{16}	DTCHB
#5	DTCAndStatusRecord [DTCMiddleByte]	34_{16}	DTCMB
#6	DTCAndStatusRecord [DTCLowByte]	56_{16}	DTCLB
#7	DTCAndStatusRecord [statusOfDTC]	26_{16}	SODTC

12.3.5.14 Example #13 - ReadDTCInformation, SubFunction = reportFirstTestFailedDTC, no information available

12.3.5.14.1 Example #13 overview

This example demonstrates usage of the reportFirstTestFailedDTC SubFunction parameter, where it is assumed that no failed DTCs have occurred since the last ClearDiagnosticInformation request from the server.

The general format of request/response messages in the following example is also applicable to SubFunction parameters reportFirstConfirmedDTC, reportMostRecentTestFailedDTC, and reportMostRecentConfirmedDTC (for the appropriate DTC status and under similar assumptions).

12.3.5.14.2 Example #13 assumptions

The following assumptions apply:

— No failed DTCs have occurred since the last ClearDiagnosticInformation request from the server.

— The server supports all status bits for masking purposes.

12.3.5.14.3 Example #13 message flow

In Table 380 no DTC is returned to the client's request.

Table 380 — ReadDTCInformation, SubFunction = reportFirstTestFailedDTC, request message flow example #13

Message direction	client → server		
Message type	Request		
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	ReadDTCInformation Request SID	19_{16}	RDTCI
#2	SubFunction = reportFirstTestFailedDTC, suppressPosRspMsgIndicationBit = FALSE	$0B_{16}$	RFCDTC

Table 381 specifies ReadDTCInformation, SubFunction = reportFirstTestFailedDTC, positive response, example #13.

Table 381 — ReadDTCInformation, SubFunction = reportFirstTestFailedDTC, positive response, example #13

Message direction	server → client		
Message type	Response		
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

Message direction	server → client		
Message type	Response		
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	ReadDTCInformation Response SID	59_{16}	RDTCIPR
#2	reportType = reportFirstTestFailedDTC	$0B_{16}$	RFCDTC
#3	DTCStatusAvailabilityMask	FF_{16}	DTCSAM

12.3.5.15 Example #14 - ReadDTCInformation, SubFunction = reportDTCFaultDetectionCounter

12.3.5.15.1 Example #14 overview

This example demonstrates the usage of the reportDTCFaultDetectionCounter SubFunction parameter.

12.3.5.15.2 Example #14 assumptions

The following assumptions apply:

— The server supports FaultDetectionCounter for continuously executed monitors.

12.3.5.15.3 Example #14 message flow

In the following example, the client requests DTCs which has not yet got the status “TestFailed”, i.e. they are prefailed. The server returns one DTC with the status of preFailed to the Client.

Table 382 specifies the ReadDTCInformation, SubFunction = reportDTCFaultDetectionCounter, request message flow example #14.

Table 382 — ReadDTCInformation, SubFunction = reportDTCFaultDetectionCounter, request message flow example #14

Message direction	client → server		
Message type	Request		
A_Data Byte	Description (all values are in hexadecimal)	Byte Value	Mnemonic
#1	ReadDTCInformation Request SID	19_{16}	RDTCI
#2	SubFunction = reportDTCFaultDetectionCounter, suppressPosRspMsgIndicationBit = FALSE	14_{16}	RDTCFDC

Table 383 specifies the ReadDTCInformation, SubFunction = reportDTCFaultDetectionCounter, positive response, example #14.

Table 383 — ReadDTCInformation, SubFunction = reportDTCFaultDetectionCounter, positive response, example #14

Message direction	server → client		
Message type	Response		
A_Data Byte	Description (all values are in hexadecimal)	Byte Value	Mnemonic
#1	ReadDTCInformation Response SID	59_{16}	RDTCIPR
#2	reportType = reportDTCFaultDetectionCounter	14_{16}	RDTCFDC
#3	DTCAndStatusRecord [DTCHighByte]	12_{16}	DTCHB

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Message direction	server → client		
Message type	Response		
A_Data Byte	Description (all values are in hexadecimal)	Byte Value	Mnemonic
#4	DTCAndStatusRecord [DTCMiddleByte]	34_{16}	DTCMB
#5	DTCAndStatusRecord [DTCLowByte]	56_{16}	DTCLB
#6	DTCAndStatusRecord [statusOfDTC]	24_{16}	SODTC
#7	DTCFaultDetectionCounter	60_{16}	DTCFDC

12.3.5.16 Example #15 - ReadDTCInformation, SubFunction = reportDTCExtDataRecordByRecordNumber

12.3.5.16.1 Example #15 overview

This example demonstrates the usage of the reportDTCExtDataRecordByRecordNumber SubFunction parameter.

12.3.5.16.2 Example #15 assumptions

The following assumptions apply:

a) The server supports the ability to store two DTCExtendedData records for all DTCs.

b) Assume that the server requests all available DTCExtendedData records stored by the server for Record number 05_{16}.

c) Assume that DTC 123456_{16} has a statusOfDTC of 24_{16}, and that the following extended data is available for the DTC.

d) The DTCExtendedData is referenced via the DTCExtDataRecordNumber 05_{16}

e) Assume that DTC 234561_{16} has a statusOfDTC of 24_{16}, and that the following extended data is available for the DTC.

f) The DTCExtendedData is referenced via the DTCExtDataRecordNumber 05_{16}.

Table 384 specifies DTCExtDataRecordNumber 05_{16} content for DTC 123456_{16}.

Table 384 — DTCExtDataRecordNumber 05_{16} content for DTC 123456_{16}

Data byte	DTCExtDataRecord Contents for DTCExtDataRecordNumber 05_{16}	Byte value
#1	Warm-up Cycle Counter – Number of warm up cycles since the DTC commanded the MIL to switch off	17_{16}

Table 385 specifies DTCExtDataRecordNumber 05_{16} content for DTC 234561_{16}.

Table 385 — DTCExtDataRecordNumber 05_{16} content for DTC 234561_{16}

Data byte	DTCExtDataRecord Contents for DTCExtDataRecordNumber 05_{16}	Byte value
#1	Warm-up Cycle Counter – Number of warm up cycles since the DTC commanded the MIL to switch off	79_{16}

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

12.3.5.16.3 Example #15 message flow

In the following example, a DTCMaskRecord including the DTC number and a DTCExtDataRecordNumber with the value of 05_{16} (report all DTCExtDataRecords) is requested by the client. The server returns two DTCs which have recorded the DTCExtDataRecordNumber submitted by the client.

Table 386 specifies ReadDTCInformation, SubFunction = reportDTCExtDataRecordByRecordNumber,request message flow example #15.

Table 386 — ReadDTCInformation, SubFunction = reportDTCExtDataRecordByRecordNumber, request message flow example #15

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadDTCInformation Request SID	19_{16}	RDTCI
#2	SubFunction = reportDTCExtDataRecordByRecordNumber, suppressPosRspMsgIndicationBit = FALSE	16_{16}	RDTCEDRBDN
#3	DTCExtDataRecordNumber	05_{16}	DTCEDRN

Table 387 specifies ReadDTCInformation, SubFunction = reportDTCExtDataRecordByRecordNumber, positive response, example #15.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 387 — ReadDTCInformation, SubFunction = reportDTCExtDataRecordByRecordNumber, positive response, example #15

Message direction	server → client		
Message type	Response		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadDTCInformation Response SID	59_{16}	RDTCIPR
#2	reportType = reportDTCExtDataRecordByRecordNumber	16_{16}	RDTCEDRBDN
#3	DTCExtDataRecordNumber	05_{16}	DTCEDRN
#4	DTCAndStatusRecord [DTCHighByte]	12_{16}	DTCHB
#5	DTCAndStatusRecord [DTCMiddleByte]	34_{16}	DTCMB
#6	DTCAndStatusRecord [DTCLowByte]	56_{16}	DTCLB
#7	DTCAndStatusRecord [statusOfDTC]	24_{16}	SODTC
#8	DTCExtDataRecord#1 [ ] = [	17_{16}	ED_1
#9	extendedData#1 byte#1	34_{16}	
#10	extendedData#2 byte#2	56_{16}	
#11	extendedData#3 byte#3]	24_{16}	
#12	DTCAndStatusRecord [DTCHighByte]	12_{16}	DTCHB
#13	DTCAndStatusRecord [DTCMiddleByte]	34_{16}	DTCMB
#14	DTCAndStatusRecord [DTCLowByte]	56_{16}	DTCLB
#15	DTCAndStatusRecord [statusOfDTC]	24_{16}	SODTC
#16	DTCExtDataRecord#2 [ ] = [	17_{16}	ED_1
#17	extendedData#1 byte#1	34_{16}	
#18	extendedData#2 byte#2	21_{16}	
#19	extendedData#3 byte#3]	24_{16}	

12.3.5.17 Example #16 – ReadDTCInformation - SubFunction = reportDTCExtendedDataRecordIdentification

12.3.5.17.1 Example #16 overview

This example demonstrates the usage of the reportDTCExtendedDataRecordIdentification SubFunction parameter.

12.3.5.17.2 Example #16 assumptions

The assumptions defined in 12.3.5.10.2 apply. Besides the following assumptions apply:

— The server supports for three DTCs the DTCExtendedDataRecord 91_{16} (DTC based IUMPR), which have the same states like in example#11 at the time of the client request.

12.3.5.17.3 Example #16 message flow

In the following example, all three of the above DTCs are returned to the client's request because all support DTCExtendedDataRecord 91_{16}.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

Table 388 specifies ReadDTCInformation, SubFunction = reportDTCExtendedDataRecordIdentification, request message flow, example #16.

Table 388 — ReadDTCInformation, SubFunction = reportDTCExtendedDataRecordIdentification, request message flow, example #16

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadDTCInformation Request SID	19_{16}	RDTCI
#2	SubFunction = reportSupportedDTCExtDataRecord, suppressPosRspMsgIndicationBit = FALSE	$1A_{16}$	LEV_ RDTCEDI
#3	DTCExtDataRecordNumber	91_{16}	DTCEDRN

Table 389 specifies ReadDTCInformation, SubFunction = reportDTCExtendedDataRecordIdentification, positive response, example #16.

Table 389 — ReadDTCInformation, SubFunction = reportDTCExtendedDataRecordIdentification, positive response, example #16

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadDTCInformation Response SID	59_{16}	RDTCIPR
#2	reportType = reportSupportedDTCExtDataRecord	$1A_{16}$	RDTCEDI
#3	DTCStatusAvailabilityMask	$7F_{16}$	DTCSAM
#4	DTCExtendedRecordNumber	91_{16}	DTCEDRN
#5	DTCAndStatusRecord#1 [DTCHighByte]	12_{16}	DTCHB
#6	DTCAndStatusRecord#1 [DTCMiddleByte]	34_{16}	DTCMB
#7	DTCAndStatusRecord#1 [DTCLowByte]	56_{16}	DTCLB
#8	DTCAndStatusRecord#1 [statusOfDTC]	24_{16}	SODTC
#9	DTCAndStatusRecord#2 [DTCHighByte]	23_{16}	DTCHB
#10	DTCAndStatusRecord#2 [DTCMiddleByte]	45_{16}	DTCMB
#11	DTCAndStatusRecord#2 [DTCLowByte]	05_{16}	DTCLB
#12	DTCAndStatusRecord#2 [statusOfDTC]	00_{16}	SODTC
#13	DTCAndStatusRecord#3 [DTCHighByte]	AB_{16}	DTCHB
#14	DTCAndStatusRecord#3 [DTCMiddleByte]	CD_{16}	DTCMB
#15	DTCAndStatusRecord#3 [DTCLowByte]	01_{16}	DTCLB
#16	DTCAndStatusRecord#3 [statusOfDTC]	$2F_{16}$	SODTC

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

12.3.5.18 Example #17 - ReadDTCInformation, SubFunction = reportWWHOBDDTCByMaskRecord

12.3.5.18.1 Example #17 overview

This example demonstrates the usage of the reportWWHOBDDTCByMaskRecord SubFunction parameter for confirmed DTCs (DTC status mask 08_{16}). The vehicle uses a CAN bus which connects two emissions-related servers.

The client uses the following request parameter settings:

— FunctionalGroupIdentifier = 33_{16} (emissions system group),

— DTCSeverityMaskRecord.DTCSeverityMask = FF_{16} (report DTCs with any severity and Class status),

— DTCSeverityMaskRecord.DTCStatusMask = 08_{16} (report DTCs with confirmedDTC status = '1').

The servers support the following settings:

— FunctionalGroupIdentifier = 33_{16} (emissions system group),

— DTCStatusAvailabilityMask = FF_{16},

— DTCSeverityAvailabilityMask = FF_{16},

— DTCFormatIdentifier = SAE_J2012-DA_DTCFormat_04 = 04_{16}.

12.3.5.18.2 Example #17 assumptions

All assumptions of example #1 apply.

12.3.5.18.3 Example #17 message flow

In the following example server #1 only reports DTC P2522-1F A/C Request “B” - circuit intermittent ($25221F_{16}$) because the statusOfDTC of $2F_{16}$ ($0010\ 1111_2$) matches the client defined status mask of 08_{16} ($0000\ 1000_2$). Server #2 reports DTC P0235-12 Turbocharger/Supercharger Boost Sensor "A" – circuit short to battery because the statusOfDTC of $2E_{16}$ ($0010\ 1110_2$) matches the client defined status mask of 08_{16} ($0000\ 1000_2$).

Table 390 specifies ReadDTCInformation request, SubFunction = reportNumberOfDTCByStatusMask.

Table 390 — ReadDTCInformation request, SubFunction = reportNumberOfDTCByStatusMask

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadDTCInformation Request SID	19_{16}	RDTCI
#2	SubFunction = reportWWHOBDDTCByMaskRecord, suppressPosRspMsgIndicationBit = FALSE	42_{16}	RWWHOBDDTCBMR
#3	FunctionalGroupIdentifier (FunctionalGroupIdentifier=emissions=33_{16})	33_{16}	FGID
#4	DTCSeverityMaskRecord[] = [DTCStatusMask]	08_{16}	DTCSM
#5	DTCSeverityMaskRecord[] = [DTCSeverityMask]	FF_{16}	DTCSVM

Table 391 specifies ReadDTCInformation response, SubFunction = reportWWHOBDDTCByStatusMask.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

Table 391 — ReadDTCInformation response, SubFunction = reportWWHOBDDTCByStatusMask

Message direction	**server #1 → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadDTCInformation Response SID	59_{16}	RDTCIPR
#2	reportType = reportWWHOBDDTCByMaskRecord	42_{16}	RWWHOBDDTCBMR
#3	FunctionalGroupIdentifier (FunctionalGroupIdentifier=emissions=33_{16})	33_{16}	FGID
#4	DTCStatusAvailabilityMask	FF_{16}	DTCSAM
#5	DTCSeverityAvailabilityMask	FF_{16}	DTCSVAM
#6	DTCFormatIdentifier = [SAE_J2012-DA_DTCFormat_04]	04_{16}	J2012-DADTCF04
#7	DTCAndSeverityRecord[DTCSeverity#1]	20_{16}	DTCASR_DTCS
#8	DTCAndSeverityRecord[DTCHighByte#1]	25_{16}	DTCASR_DTCHB
#9	DTCAndSeverityRecord[DTCMiddleByte#1]	22_{16}	DTCASR_DTCMB
#10	DTCAndSeverityRecord[DTCLowByte#1]	$1F_{16}$	DTCASR_DTCLB
#11	DTCAndSeverityRecord[statusOfDTC#1]	$2F_{16}$	DTCASR_SODTC

Table 392 specifies ReadDTCInformation response, SubFunction = reportOBDDTCByStatusMask.

Table 392 — ReadDTCInformation response, SubFunction = reportOBDDTCByStatusMask

Message direction	**server #2 → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadDTCInformation Response SID	59_{16}	RDTCIPR
#2	reportType = reportWWHOBDDTCByMaskRecord	42_{16}	RWWHOBDDTCBMR
#3	FunctionalGroupIdentifier (FunctionalGroupIdentifier=emissions=33_{16})	33_{16}	FGID
#4	DTCStatusAvailabilityMask	FF_{16}	DTCSAM
#5	DTCSeverityAvailabilityMask	FF_{16}	DTCSVAM
#6	DTCFormatIdentifier = [SAE_J2012-DA_DTCFormat_04]	04_{16}	J2012-DADTCF04
#7	DTCAndSeverityRecord[DTCSeverity#1]	20_{16}	DTCASR_DTCS
#8	DTCAndSeverityRecord[DTCHighByte#1]	02_{16}	DTCASR_DTCHB
#9	DTCAndSeverityRecord[DTCMiddleByte#1]	35_{16}	DTCASR_DTCMB
#10	DTCAndSeverityRecord[DTCLowByte#1]	12_{16}	DTCASR_DTCLB
#11	DTCAndSeverityRecord[statusOfDTC#1]	$2E_{16}$	DTCASR_SODTC

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

12.3.5.19 Example #18 - ReadDTCInformation, SubFunction = reportOBDDTCWithPermanentStatus, matching DTCs returned

12.3.5.19.1 Example #18 overview

This example demonstrates usage of the reportWWHOBDDTCWithPermanentStatus SubFunction parameter. This example also applies to the SubFunction parameter reportPermanentDTC, except that the FunctionalGroupIdentifier is used in reportWWHOBDDTCWithPermanentStatus.

12.3.5.19.2 Example #18 assumptions

The server supports four permanent DTCs to be stored.

The following assumptions apply to DTC P0805-11 Clutch Position Sensor - circuit short to ground (080511_{16}), statusOfDTC of $2F_{16}$ ($0010\ 1111_2$). Since P0805-11 is emission relevant and illuminates MIL, it is also stored as a Permanent DTC.

Table 393 specifies the statusOfDTC = $2F_{16}$ of DTC P0805-11.

Table 393 — statusOfDTC = $2F_{16}$ of DTC P0805-11

statusOfDTC: bit field name	Bit #	Bit state	Description
testFailed	0	1	DTC is failed at the time of the request
testFailedThisOperationCycle	1	1	DTC failed on the current operation cycle
pendingDTC	2	1	DTC failed on the current or previous operation cycle
confirmedDTC	3	1	DTC is confirmed at the time of the request
testNotCompletedSinceLastClear	4	0	DTC test was completed since the last code clear
testFailedSinceLastClear	5	1	DTC test failed at least once since last code clear
testNotCompletedThisOperationCycle	6	0	DTC test completed this operation cycle
warningIndicatorRequested	7	0	Server is not requesting warningIndicator to be active

12.3.5.19.3 Example #18 message flow

In the following example, DTC P0805-11 (080511_{16}) is returned to the client's request.

Table 394 specifies the ReadDTCInformation, SubFunction = reportWWHOBDDTCWithPermanentStatus, request message flow example #2.

Table 394 — ReadDTCInformation, SubFunction = reportWWHOBDDTCWithPermanentStatus, request message flow example #2

Message direction	**client → server**		
Message type	**Request**		
A_Data Byte	**Description (all values are in hexadecimal)**	**Byte Value**	**Mnemonic**
#1	ReadDTCInformation Request SID	19_{16}	RDTCI
#2	SubFunction = reportWWHOBDDTCWithPermanentStatus, suppressPosRspMsgIndicationBit = FALSE	55_{16}	RWWHOBDDTCWPS
#3	FunctionalGroupIdentifier	33_{16}	FGID

Table 395 specifies the ReadDTCInformation, SubFunction = reportWWHOBDDTCWithPermanentStatus, positive response, example #2.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 395 — ReadDTCInformation, SubFunction = reportWWHOBDDTCWithPermanentStatus, positive response, example #18

Message direction		server → client	
Message type		Response	
A_Data Byte	Description (all values are in hexadecimal)	Byte Value	Mnemonic
#1	ReadDTCInformation Response SID	59_{16}	RDTCIPR
#2	reportType = reportWWHOBDDTCWithPermanentStatus	55_{16}	RWWHOBDDTCWPS
#3	FunctionalGroupIdentifier	33_{16}	FGID
#4	DTCStatusAvailabilityMask	FF_{16}	DTCSAM
#5	DTCFormatIdentifier= SAE_J2012-DA_DTCFormat_04	04_{16}	J2012-DADTCF04
#4	DTCAndStatusRecord#2 [DTCHighByte]	08_{16}	DTCHB
#5	DTCAndStatusRecord#2 [DTCMiddleByte]	05_{16}	DTCMB
#6	DTCAndStatusRecord#2 [DTCLowByte]	11_{16}	DTCLB
#7	DTCAndStatusRecord#2 [statusOfDTC]	$2F_{16}$	SODTC

12.3.5.20 Example #19 – ReadDTCInformation - SubFunction = reportDTCByReadinessGroupIdentifier

12.3.5.20.1 Example #19 overview

This example demonstrates the usage of the reportDTCByReadinessGroupIdentifier SubFunction parameter.

12.3.5.20.2 Example #19 assumptions

The assumptions defined in 12.3.5.10.2 apply. Besides the following assumptions apply:

— The server supports the DTCReadinessGroupIdentifier Comprehensive Components Monitoring Group (01_{16}) with three DTCs which have the same states like in example#11 at the time of the client request.

— This DTCs are part of the “emission system group” 33_{16}.

12.3.5.20.3 Example #19 message flow

In the following example, all three of the above DTCs are returned to the client’s request because all belonging to ReadynessGroupID=01_{16} .

Table 396 specifies ReadDTCInformation, SubFunction = reportDTCByReadinessGroupIdentifier, request message flow, example #19.

Table 396 — ReadDTCInformation, SubFunction = reportDTCByReadinessGroupIdentifier, request message flow, example #19

Message direction		client → server	
Message type		Request	
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	ReadDTCInformation Request SID	19_{16}	RDTCI

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Message direction	client → server		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#2	SubFunction = reportDTCInformationByDTCReadinessGroupIdentifier, suppressPosRspMsgIndicationBit = FALSE	56_{16}	LEV_ RDTCBRGI
#3	FunctionalGroupIdentifier	33_{16}	DTCEDRN
#4	DTCReadinessGroupIdentifier	01_{16}	RGID

Table 397 specifies ReadDTCInformation, SubFunction = reportDTCByReadinessGroupIdentifier,positive response, example #19.

Table 397 — ReadDTCInformation, SubFunction = reportDTCByReadinessGroupIdentifier, positive response, example #19

Message direction	server → client		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadDTCInformation Response SID	59_{16}	RDTCIPR
#2	reportType = reportDTCInformationByDTCReadinessGroupIdentifier,	56_{16}	RDTCBRGI
#3	FunctionalGroupIdentifier	33_{16}	DTCSAM
#4	DTCStatusAvailabilityMask	FF_{16}	DTCSAM
#5	DTCFormatIdentifier = [SAE_J2012-DA_DTCFormat_04]	04_{16}	J2012-DADTCF04
#6	DTCReadinessGroupIdentifier	01_{16}	DTCRGID
#7	DTCAndStatusRecord#1 [DTCHighByte]	12_{16}	DTCHB
#8	DTCAndStatusRecord#1 [DTCMiddleByte]	34_{16}	DTCMB
#9	DTCAndStatusRecord#1 [DTCLowByte]	56_{16}	DTCLB
#10	DTCAndStatusRecord#1 [statusOfDTC]	24_{16}	SODTC
#11	DTCAndStatusRecord#2 [DTCHighByte]	23_{16}	DTCHB
#12	DTCAndStatusRecord#2 [DTCMiddleByte]	45_{16}	DTCMB
#13	DTCAndStatusRecord#2 [DTCLowByte]	05_{16}	DTCLB
#14	DTCAndStatusRecord#2 [statusOfDTC]	00_{16}	SODTC
#15	DTCAndStatusRecord#3 [DTCHighByte]	AB_{16}	DTCHB
#16	DTCAndStatusRecord#3 [DTCMiddleByte]	CD_{16}	DTCMB
#17	DTCAndStatusRecord#3 [DTCLowByte]	01_{16}	DTCLB
#18	DTCAndStatusRecord#3 [statusOfDTC]	$2F_{16}$	SODTC

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

13 InputOutput control functional unit

13.1 Overview

Table 398 specifies the InputOutput Control functional unit.

Table 398 — InputOutput Control functional unit

Service	Description
InputOutputControlByIdentifier	The client requests the control of an input/output specific to the server.

13.2 InputOutputControlByIdentifier ($2F_{16}$) service

13.2.1 Service description

The InputOutputControlByIdentifier service is used by the client to substitute a value for an input signal, internal server function and/or force control to a value for an output (actuator) of an electronic system. In general, this service is used for relatively simple (e.g. static) input substitution/output control whereas routineControl is used if more complex input substitution/output control is necessary.

The client request message contains a dataIdentifier to reference the input signal, internal server function, and/or output signal(s) (actuator(s)) (in case of a device control access it might reference a group of signals) of the server. The controlOptionRecord parameter shall include all information required by the server's input signal(s), internal function(s) and/or output signal(s). The vehicle manufacturer may require that the request message contain a controlEnableMask if the dataIdentifier to be controlled references more than one parameter (i.e. the dataIdentifier is packeted or bitmapped). If the vehicle manufacturer chooses to support the EnableMask concept, the controlEnableMask parameter is mandatory on all types of InputOutputControlByIdentifier requests for this service. If inputOutputControlByIdentifier is requested on a dataIdentifier that references a measured output value or feedback value, the server shall be responsible for substituting the correct target value within the server control strategy so that the normal server control strategy will attempt to reach the desired state from the client request message.

The server shall send a positive response message if the request control was successfully started or has reached its desired state. The server shall send a positive response message to a request message with an inputOutputControlParameter of returnControlToECU even if the dataIdentifier is currently not under tester control. In addition, when receiving a returnControlToECU request, a server shall always provide the client the capability of setting the controlMask (if supported) bits all to '1' in order to return control of a packeted or bit-mapped dataIdentifier completely back to the ECU. The format and length of the controlState bytes following the inputOutputControlParameter within the controlOptionRecord parameter of the request message shall exactly match the length and format of the dataRecord of the dataIdentifier being requested. This way it shall be ensured that the actual output or input state can be retrieved by using the service ReadDatabyIdentifier with the same DID.

When utilizing the inputOutputControlByIdentifier service to perform input substitution or output control, there are two fundamental requirements placed on the ECU accepting the request. The first is to disconnect the appropriate data object(s) referenced by the parameter(s) within the dataIdentifier from all upstream control strategies that would otherwise update the data object value. The second is to substitute a value into the appropriate data object(s) that will be used for all downstream activities of the control strategy. For example, a tester request to directly force the headlamps on would need to prevent the headlamp switch position from affecting the headlamp output and substitute the desired state of "On" into the data object(s) used by the functions which ultimately decide the headlamp state desired output.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

The service allows the control of a single dataIdentifier and its corresponding parameter(s) in a single request message. Doing so, the server will respond with a single response message including the dataIdentifier of the request message plus controlStatus information.

IMPORTANT — The server and the client shall meet the request and response message behaviour as specified in 8.7.

13.2.2 Request message

13.2.2.1 Request message definition

Table 399 specifies the request message.

Table 399 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	InputOutputControlByIdentifier Request SID	M	$2F_{16}$	IOCBI
	dataIdentifier [] = [			IOI_
#2	byte#1 (MSB)	M	00_{16} to FF_{16}	B1
#3	byte#2 (LSB)]	M	00_{16} to FF_{16}	B2
	controlOptionRecord [] = [			CSR_
#4	inputOutputControlParameter	M_1	00_{16} to FF_{16}	IOCP_
:	controlState#1	C_1	00_{16} to FF_{16}	CS_
#4+(m-1)	:	:	:	:
	controlState#m]	C_1	00_{16} to FF_{16}	CS_
	controlEnableMaskRecord#1[] = [			CEM_
#4+m	controlMask#1	C_2	00_{16} to FF_{16}	CM_
:	:	:	:	:
#4+m+(r-1)	controlMask#r]	C_2	00_{16} to FF_{16}	CM_

M_1: InputOutputControlParameter shall be implemented as defined in Table E.1.

C_1: The presence of this parameter depends on the dataIdentifier and the inputOutputControlParameter (see Table E.1).

C_2: If the controlEnableMask concept is supported by the vehicle manufacturer, this parameter shall be included if the dataIdentifier consists of more than one parameter (see controlEnableMaskRecord definition).

13.2.2.2 Request message SubFunction parameter $Level (LEV_) definition

This service does not use a SubFunction parameter.

13.2.2.3 Request message data-parameter definition

The following data-parameters (Table 400) are defined for this service:

Table 400 — Request message data-parameter definition

Definition
dataIdentifier This parameter identifies a server local input signal(s), internal parameter(s) and/or output signal(s). The applicable range of values for this parameter can be found in the table of dataIdentifiers defined in C.1.
controlOptionRecord The controlOptionRecord consists of one or multiple bytes (inputOutputControlParameter and controlState#1 to controlState#m). The controlOptionRecord parameter details shall be implemented as defined in Table E.1.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

Definition
controlEnableMaskRecord
The controlEnableMaskRecord consists of one or multiple bytes (controlMask#1 to controlMask#r). The controlEnableMaskRecord shall only be supported when the dataIdentifier to be controlled consists of more than one parameter (i.e. the dataIdentifier is bit-mapped or packeted by definition). There shall be one bit in the controlEnableMaskRecord corresponding to each individual parameter defined within the dataIdentifier. The controlEnableMaskRecord shall not be supported when the dataIdentifier to be controlled consists of only a single parameter. NOTE Each parameter in the dataIdentifier can be any number of bits. The value of each bit within the controlEnableMaskRecord shall determine whether the corresponding parameter in the dataIdentifier will be affected by the request. A bit value of '0' in the controlEnableMaskRecord shall represent that the corresponding parameter is not affected by this request and a bit value of '1' shall represent that the corresponding parameter is affected by this request. The most significant bit of ControlMask#1 shall correspond to the first parameter in the ControlState starting at the most significant bit of ControlState#1, the second most significant bit of ControlMask#1 shall correspond to the second parameter in the ControlState, and continuing on in this fashion utilising as many ControlMask bytes as necessary to mask all parameters. For example, the least significant bit of ControlMask#2 would correspond to the 16th parameter in the controlState. For bitmapped dataIdentifiers, unsupported bits shall also have a corresponding bit in the controlEnableMaskRecord so that the position of the mask bit of every parameter in the controlEnableMaskRecord shall exactly match the position of the corresponding parameter in the controlState.

13.2.3 Positive response message

13.2.3.1 Positive response message definition

Table 401 specifies Positive response message definition.

Table 401 — Positive response message definition

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	InputOutputControlByIdentifier Response SID	M	$6F_{16}$	IOCBIPR
	dataIdentifier [] = [			IOI_
#2	byte#1 (MSB)	M	00_{16} to FF_{16}	B1
#3	byte#2 (LSB)]	M	00_{16} to FF_{16}	B2
	controlStatusRecord [] = [			CSR_
#4	inputOutputControlParameter	M	00_{16} to FF_{16}	IOCP_
#5	controlState#1	C_1	00_{16} to FF_{16}	CS_
:	:	:	:	:
#5+(m-1)	controlState#m]	C_1	00_{16} to FF_{16}	CS_
C_1: The presence of this parameter depends on the dataIdentifier and the inputOutputControlParameter (see Table E.1).				

13.2.3.2 Positive response message data-parameter definition

Table 402 specifies Response message data-parameter definition.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 402 — Response message data-parameter definition

Definition
dataIdentifier This parameter is an echo of the dataIdentifier(s) from the request message.
controlStatusRecord The controlState parameter consists of multiple bytes (InputOutputControlParameter and controlState#1 to controlState#m) which include, e.g. feedback data. The controlStatusRecord parameter details shall be implemented as defined in Table E.1.

13.2.4 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 403. The listed negative responses shall be used if the error scenario applies to the server.

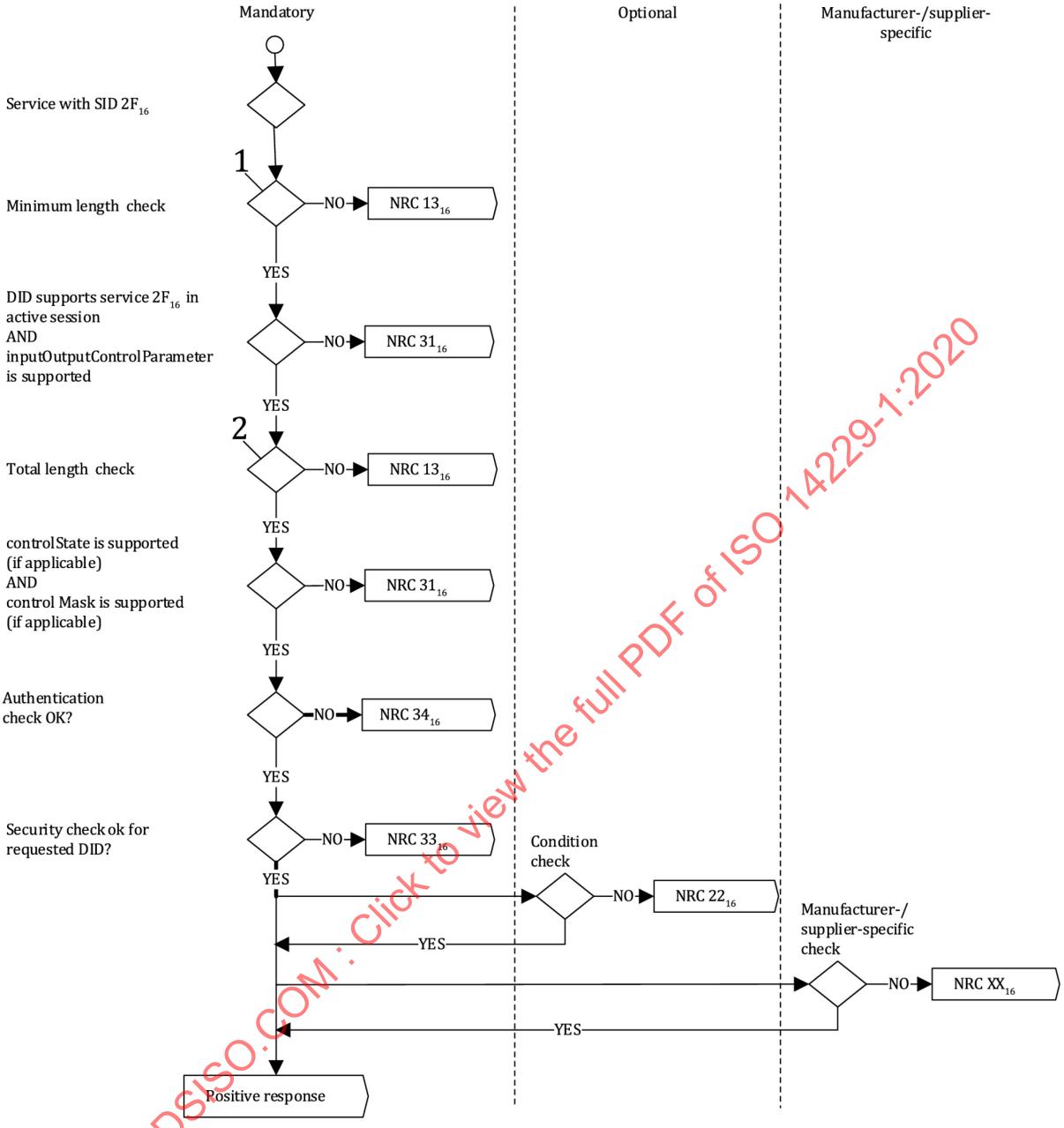
Table 403 — Supported negative response codes

NRC	Description	Mnemonic
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 InputOutputControl are not met.	CNC
31_{16}	**requestOutOfRange** This NRC shall be sent if: — the requested dataIdentifier value is not supported by the device; — the value contained in the inputOutputControlParameter is invalid (see definition of inputOutputControlParameter); — one or multiple of the applicable controlState values of the controlOptionRecord record are invalid; — the combination of bits enabling control in the ControlEnableMaskRecord is not supported by the device.	ROOR
33_{16}	**securityAccessDenied** This NRC shall be returned if a client sends a request with a valid secure dataIdentifier and the server's security feature is currently active.	SAD
34_{16}	**authenticationRequired** This NRC shall be sent if the dataIdentifier is secured and the client has insufficient rights based on its Authentication state.	AR

The evaluation sequence is documented in Figure 29.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved



Key

1 at least 4 (SI+DID+IOCP)

2 if IOCP = shortTermAdjustment, 1 byte SI + 2 byte DID + 1 byte IOCP + nth byte controlState + nth byte controlMask (if applicable), if IOCP <> shortTermAdjustment, 1 byte SI + 2 byte DID + 1 byte IOCP + nth byte controlMask (if applicable)

Figure 29 — NRC handling for InputOutputControlByIdentifier service

13.2.5 Message flow example(s) InputOutputControlByIdentifier

13.2.5.1 Assumptions

The example below shows how the InputOutputControlByIdentifier is used with an HVAC Control Module and assumes that physical communication is performed with a single server.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

13.2.5.2 Example #1 - "Air Inlet Door Position" shortTermAdjustment

The parameter being controlled is the "Air Inlet Door Position" associated with dataIdentifier ($9B00_{16}$).

Conversion: Air Inlet Door Position [%] = decimal(Hex) × 1 [%]

13.2.5.2.1 Step #1: ReadDataByIdentifier

Table 404 to Table 413 specifies an example which uses the ReadDataByIdentifier service to read the current state of the Air Inlet Door Position.

Table 404 — ReadDataByIdentifier request message flow example #1 - step #1

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadDataByIdentifier Request SID	22_{16}	RDBI
#2	dataIdentifier [byte#1] = $9B_{16}$	$9B_{16}$	DID_B1
#3	dataIdentifier [byte#2] = 00_{16} (“Air Inlet Door Position”)	00_{16}	DID_B2

Table 405 — ReadDataByIdentifier positive response message flow example #1 - step #1

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	ReadDataByIdentifier Response SID	62_{16}	RDBIPR
#2	dataIdentifier [byte#1] = $9B_{16}$	$9B_{16}$	DID_B1
#3	dataIdentifier [byte#2] = 00_{16} (“Air Inlet Door Position”)	00_{16}	DID_B2
#4	dataRecord [data#1] = 10 %	$0A_{16}$	DREC_DATA1

13.2.5.2.2 Step #2: shortTermAdjustment

Table 406 — InputOutputControlByIdentifier request message flow example #1 - step #2

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	InputOutputControlByIdentifier Request SID	$2F_{16}$	IOCBI
#2	dataIdentifier [byte#1] = $9B_{16}$	$9B_{16}$	IOI_B1
#3	dataIdentifier [byte#2] = 00_{16} (“Air Inlet Door Position”)	00_{16}	IOI_B2
#4	controlOptionRecord [inputOutputControlParameter] = shortTermAdjustment	03_{16}	IOCP_STA
#5	controlOptionRecord [controlState#1] = 60 %	$3C_{16}$	CS_1

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 407 — InputOutputControlByIdentifier positive response message flow example #1 - step #2

Message direction	server → client		
Message type	Response		
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	InputOutputControlByIdentifier Response SID	$6F_{16}$	IOCBLIPR
#2	dataIdentifier [byte#1] = $9B_{16}$	$9B_{16}$	IOI_B1
#3	dataIdentifier [byte#2] = 00_{16} (“Air Inlet Door Position”)	00_{16}	IOI_B2
#4	controlStatusRecord [inputOutputControlParameter] = shortTermAdjustment	03_{16}	IOCP_STA
#5	controlStatusRecord [controlState#1] = 12 %	$0C_{16}$	CS_1

NOTE The client has sent an inputOutputControlByIdentifier request message as specified above. The server has sent an immediate positive response message, which includes the controlState parameter "Air Inlet Door Position" with the value of 12 %. The air inlet door requires a certain amount of time to move to the requested value of 60 %.

13.2.5.2.3 Step #3: ReadDataByIdentifier

This example uses the readDataByIdentifier service to read the current state of the Air Inlet Door Position.

Table 408 — ReadDataByIdentifier request message flow example #1 - step #3

Message direction	client → server		
Message type	Request		
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	ReadDataByIdentifier Request SID	22_{16}	RDBI
#2	dataIdentifier [byte#1] = $B9_{16}$	$9B_{16}$	DID_B1
#3	dataIdentifier [byte#2] = 00_{16} (“Air Inlet Door Position”)	00_{16}	DID_B2

Table 409 — ReadDataByIdentifier positive response message flow example #1 - step #3

Message direction	server → client		
Message type	Response		
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	ReadDataByIdentifier Response SID	62_{16}	RDBIPR
#2	dataIdentifier [byte#1] = $9B_{16}$	$9B_{16}$	DID_B1
#3	dataIdentifier [byte#2] = 00_{16} (“Air Inlet Door Position”)	00_{16}	DID_B2
#4	dataRecord [data#1] = 60 %	$3C_{16}$	DREC_DATA1

NOTE The client has sent a readDataByIdentifier request message as specified above while inputOutputControlByIdentifier is active. It will take a finite amount of time for the server control strategy to ultimately reach the desired value. The example above reflects when the server has finally reached the desired target value.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

13.2.5.2.4 Step #4: returnControlToECU

Table 410 — InputOutputControlByIdentifier request message flow example #1 - step #4

Message direction	client → server		
Message type	Request		
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	InputOutputControlByIdentifier Request SID	$2F_{16}$	IOCBI
#2	dataIdentifier [byte#1] = $9B_{16}$	$9B_{16}$	IOI_B1
#3	dataIdentifier [byte#2] = 00_{16} (“Air Inlet Door Position”)	00_{16}	IOI_B2
#4	controlOptionRecord [inputOutputControlParameter] = returnControlToECU	00_{16}	RCTECU

Table 411 — InputOutputControlByIdentifier positive response message flow example #1 - step #4

Message direction	server → client		
Message type	Response		
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	InputOutputControlByIdentifier Response SID	$6F_{16}$	IOCBIPR
#2	dataIdentifier [byte#1] = $9B_{16}$	$9B_{16}$	IOI_B1
#3	dataIdentifier [byte#2] = 00_{16} (“Air Inlet Door Position”)	00_{16}	IOI_B2
#4	controlStatusRecord [inputOutputControlParameter] = returnControlToECU	00_{16}	RCTECU
#5	controlStatusRecord [controlState#1] = 58 %	$3A_{16}$	CS_1

13.2.5.2.5 Step #5: freezeCurrentState

Table 412 — InputOutputControlByIdentifier request message flow example #1 - step #5

Message direction	client → server		
Message type	Request		
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	InputOutputControlByIdentifier Request SID	$2F_{16}$	IOCBI
#2	dataIdentifier [byte#1] = $9B_{16}$	$9B_{16}$	IOI_B1
#3	dataIdentifier [byte#2] = 00_{16} (“Air Inlet Door Position”)	00_{16}	IOI_B2
#4	controlOptionRecord [inputOutputControlParameter] = freezeCurrentState	02_{16}	IOCP_FCS

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 413 — InputOutputControlByIdentifier positive response message flow example #1 - step #5

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	InputOutputControlByIdentifier Response SID	$6F_{16}$	IOCBIPR
#2	dataIdentifier [byte#1] = $9B_{16}$	$9B_{16}$	IOI_B1
#3	dataIdentifier [byte#2] = 00_{16} (“Air Inlet Door Position”)	00_{16}	IOI_B2
#4	controlStatusRecord [inputOutputControlParameter] = freezeCurrentState	02_{16}	IOCP_FCS
#5	controlStatusRecord [controlState#1] = 50 %	32_{16}	CS_1

13.2.5.3 Example #2 – EGR and IAC shortTermAdjustment

13.2.5.3.1 Assumptions

This example uses a packeted dataIdentifier 0155_{16} to demonstrate control of individual parameters or multiple parameters within a single request.

This subclause specifies the test conditions for a shortTermAdjustment function and the associated message flow of the example dataIdentifier 0155_{16}. The dataIdentifier supports five individual parameters as described in Table 414 below.

Table 414 — Composite data blocks – DataIdentifier definitions – Example #2

DID	**Data byte**	**Parameter**		**Data record contents**
		Number	**Size**	
0155_{16}	#1 (all bits)	#1	8 bit	dataRecord [data#1] = IAC Pintle Position (n = counts)
	#2 - #3 (all bits)	#2	16 bit	dataRecord [data#2-#3] = RPM (0 = 0 U/min, 65 535 = 65 535 U/min)
	#4 (bits 7 to 4)	#3	4 bit	dataRecord [data#4 (bits 7 to 4)] = Pedal Position A: Linear Scaling, 0 = 0 %, 15 = 120 %
	#4 (bits 3 to 0)	#4	4 bit	dataRecord [data#4 (bits 3 to 0)] = Pedal Position B: Linear Scaling, 0 = 0 %, 15 = 120 %
	#5 (all bits)	#5	8 bit	dataRecord [data#5] = EGR Duty Cycle: Linear Scaling, 0 counts = 0 %, 255 counts = 100 %

DataIdentifier 0155_{16} is packeted by definition and is comprised of five elemental parameters. For individual control purposes, each of these elemental parameters is selectable via a single bit within the ControlEnableMaskRecord. If a given dataIdentifier has a definition other than packeted or bitmapped, the ControlEnableMaskRecord is not present in the request message. The most significant bit of ControlMask#1 is always required to correspond to the first parameter in the dataIdentifier starting at the most significant bit of ControlState#1. This is demonstrated in Table 415.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 415 — ControlEnableMaskRecord– Example #2

ControlEnableMaskRecord for dataIdenitifier 0155_{16}. Total size = 1 byte (i.e. consists only of ControlEnableMask#1)		
Bit Position		**ControlEnableMask#1 – Bit Meaning (1 = affected, 0 = not affected)**
7	MSB	Determines whether or not Parameter#1 (IAC Pintle Position) will be affected by the request
6	---	Determines whether Parameter#2 (RPM) will be affected by the request
5	---	Determines whether Parameter#3 (Pedal Position A) will be affected by the request
4	---	Determines whether Parameter#4 (Pedal Position B) will be affected by the request
3	---	Determines whether Parameter#5 (EGR Duty Cycle) will be affected by the request
2	---	Not affected due to no corresponding parameter
1	---	Not affected due to no corresponding parameter
0	LSB	Not affected due to no corresponding parameter

13.2.5.3.2 Case #1: Control IAC Pintle Position only

Table 416 specifies the InputOutputControlByIdentifier request message flow example #2 – Case #1.

Table 416 — InputOutputControlByIdentifier request message flow example #2 – Case #1

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	InputOutputControlByIdentifier Request SID	$2F_{16}$	IOCBI
#2	dataIdentifier [byte#1] = 01_{16}	01_{16}	IOI_B1
#3	dataIdentifier [byte#2] = 55_{16} (IAC/RPM/PPA/PPB/EGR)	55_{16}	IOI_B2
#4	controlOptionRecord [inputOutputControlParameter] = shortTermAdjustment	03_{16}	IOCP_STA
#5	controlOptionRecord [controlState#1] = IAC Pintle Position (7 counts)	07_{16}	CS_1
#6	controlOptionRecord [controlState#2] = RPM (XX)	XX_{16}	CS_2
#7	controlOptionRecord [controlState#3] = RPM (XX)	XX_{16}	CS_3
#8	controlOptionRecord [controlState#4] = Pedal Position A (Y) and B (Z)	YZ_{16}	CS_4
#9	controlOptionRecord [controlState#5] = EGR Duty Cycle (XX)	XX_{16}	CS_5
#10	controlEnableMask [controlMask#1] = Control IAC Pintle Position ONLY	80	CM_1

NOTE The values transmitted for RPM, Pedal Position A, Pedal Position B, and EGR Duty Cycle in controlState#2 - #5 are irrelevant because the controlMask#1 parameter specifies that only the first parameter in the dataIdentifier will be affected by the request.

Table 417 specifies the InputOutputControlByIdentifier positive response message flow example #2 – Case #1.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

Table 417 — InputOutputControlByIdentifier positive response message flow example #2 – Case #1

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	InputOutputControlByIdentifier Response SID	$6F_{16}$	IOCBLIPR
#2	dataIdentifier [byte#1] = 01_{16}	01_{16}	IOI_B1
#3	dataIdentifier [byte#2] = 55_{16} (IAC/RPM/PPA/PPB/EGR)	55_{16}	IOI_B2
#4	controlStatusRecord [inputOutputControlParameter] = shortTermAdjustment	03_{16}	IOCP_STA
#5	controlStatusRecord [controlState#1] = IAC Pintle Position (7 counts)	07_{16}	CS_1
#6	controlStatusRecord [controlState#2] = RPM (750 U/min)	02_{16}	CS_2
#7	controlStatusRecord [controlState#3] = RPM	EE_{16}	CS_3
#8	controlStatusRecord [controlState#4] = Pedal Position A (8 %), Pedal Position B (16 %)	12_{16}	CS_4
#9	controlStatusRecord [controlState#5] = EGR Duty Cycle (35 %)	59_{16}	CS_5

The value transmitted for all parameters in controlState#1 – controlState#5 shall reflect the current state of the system.

13.2.5.3.3 Case #2: Control RPM Only

Table 418 specifies the InputOutputControlByIdentifier request message flow example #2 – Case #2.

Table 418 — InputOutputControlByIdentifier request message flow example #2 – Case #2

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	InputOutputControlByIdentifier Request SID	$2F_{16}$	IOCBI
#2	dataIdentifier [byte#1] = 01_{16}	01_{16}	IOI_B1
#3	dataIdentifier [byte#2] = 55_{16} (IAC/RPM/EGR)	55_{16}	IOI_B2
#4	controlOptionRecord [inputOutputControlParameter] = shortTermAdjustment	03_{16}	IOCP_STA
#5	controlOptionRecord [controlState#1] = IAC Pintle Position (XX counts)	XX_{16}	CS_1
#6	controlOptionRecord [controlState#2] = RPM ($03E8_{16}$ = 1000 U/min)	03_{16}	CS_2
#7	controlOptionRecord [controlState#3] = RPM	$E8_{16}$	CS_3
#8	controlOptionRecord [controlState#4] = Pedal Position A (Y) and B (Z)	YZ_{16}	CS_4
#9	controlOptionRecord [controlState#5] = EGR Duty Cycle (XX)	XX_{16}	CS_5
#10	controlEnableMask [controlMask#1] = Control RPM ONLY	40_{16}	CM_1

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

NOTE The values transmitted for IAC Pintle Position, Pedal Position A, Pedal Position B, and EGR Duty Cycle in controlState#1 and controlState#4 - #5 are irrelevant because the controlMask#1 parameter specifies that only the second parameter in the dataIdentifier will be affected by the request.

Table 419 specifies the InputOutputControlByIdentifier positive response message flow example #2 – Case #2.

Table 419 — InputOutputControlByIdentifier positive response message flow example #2 – Case #2

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	InputOutputControlByIdentifier Response SID	$6F_{16}$	IOCBLIPR
#2	dataIdentifier [byte#1] = 01_{16}	01_{16}	IOI_B1
#3	dataIdentifier [byte#2] = 55_{16} (IAC/RPM/PPA/PPB/EGR)	55_{16}	IOI_B2
#4	controlStatusRecord [inputOutputControlParameter] = shortTermAdjustment	03_{16}	IOCP_STA
#5	controlStatusRecord [controlState#1] = IAC Pintle Position (9 counts)	09_{16}	CS_1
#6	controlStatusRecord [controlState#2] = RPM (950 U/min)	03_{16}	CS_2
#7	controlStatusRecord [controlState#3] = RPM	$B6_{16}$	CS_3
#8	controlStatusRecord [controlState#4] = Pedal Position A (8 %), Pedal Position B (16 %)	12_{16}	CS_4
#9	controlStatusRecord [controlState#5] = EGR Duty Cycle (35 %)	59_{16}	CS_5

The value transmitted for all parameters in controlState#1 – controlState#5 shall reflect the current state of the system.

13.2.5.3.4 Case #3: Control both Pedal Position A and EGR Duty Cycle

Table 420 specifies the InputOutputControlByIdentifier request message flow example #2 – Case #3.

Table 420 — InputOutputControlByIdentifier request message flow example #2 – Case #3

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	InputOutputControlByIdentifier Request SID	$2F_{16}$	IOCBI
#2	dataIdentifier [byte#1] = 01_{16}	01_{16}	IOI_B1
#3	dataIdentifier [byte#2] = 55_{16} (IAC/RPM/PPA/PPB/EGR)	55_{16}	IOI_B2
#4	controlOptionRecord [inputOutputControlParameter] = shortTermAdjustment	03_{16}	IOCP_STA
#5	controlOptionRecord [controlState#1] = IAC Pintle Position (XX)	XX_{16}	CS_1
#6	controlOptionRecord [controlState#2] = RPM (XX)	XX_{16}	CS_2
#7	controlOptionRecord [controlState#3] = RPM (XX)	XX_{16}	CS_3

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

Message direction	client → server		
Message type	Request		
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#8	controlOptionRecord [controlState#4] = Pedal Position A (3_{16} = 24 %), Pedal Position B (Z)	$3Z_{16}$	CS_4
#9	controlOptionRecord [controlState#5] = EGR Duty Cycle (45 %)	72_{16}	CS_5
#10	controlEnableMask [controlMask#1] = Control Pedal Position A and EGR	28	CM_1

NOTE The values transmitted for IAC Pintle Position, RPM and Pedal Position B in controlState#1 - #3 and controlState#4 (bits 3 to 0) are irrelevant because the controlMask#1 parameter specifies that only the third and fifth parameter in the dataIdentifier will be affected by the request.

Table 421 specifies the InputOutputControlByIdentifier positive response message flow example #2 – Case #3.

Table 421 — InputOutputControlByIdentifier positive response message flow example #2 – Case #3

Message direction	server → client		
Message type	Response		
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	InputOutputControlByIdentifier Response SID	$6F_{16}$	IOCBLIPR
#2	dataIdentifier [byte#1] = 01_{16}	01_{16}	IOI_B1
#3	dataIdentifier [byte#2] = 55_{16} (IAC/RPM/PPA/PPB/EGR)	55_{16}	IOI_B2
#4	controlStatusRecord [inputOutputControlParameter] = shortTermAdjustment	03_{16}	IOCP_STA
#5	controlStatusRecord [controlState#1] = IAC Pintle Position (7 counts)	07_{16}	CS_1
#6	controlStatusRecord [controlState#2] = RPM (850 U/min)	03_{16}	CS_2
#7	controlStatusRecord [controlState#3] = RPM	52_{16}	CS_3
#8	controlStatusRecord [controlState#4] = Pedal Position A (24 %) Pedal Position B (16 %)	32_{16}	CS_4
#9	controlStatusRecord [controlState#4] = EGR Duty Cycle (41 %)	69_{16}	CS_5

The value transmitted for all parameters in controlState#1 – controlState#5 shall reflect the current state of the system.

13.2.5.3.5 Case #4: Return control of all parameters to the ECU

Table 422 specifies the InputOutputControlByIdentifier request message flow example #2 – Case #4.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 422 — InputOutputControlByIdentifier request message flow example #2 – Case #4

Message direction	client → server		
Message type	Request		
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	InputOutputControlByIdentifier Request SID	$2F_{16}$	IOCBI
#2	dataIdentifier [byte#1] = 01_{16}	01_{16}	IOI_B1
#3	dataIdentifier [byte#2] = 55_{16} (IAC/RPM/PPA/PPB/EGR)	55_{16}	IOI_B2
#4	controlOptionRecord [inputOutputControlParameter] = returnControlToECU	00_{16}	RCTECU
#5	controlEnableMask [controlMask#1] = All elemental parameters	FF_{16}	CM_1

Table 423 specifies the InputOutputControlByIdentifier positive response message flow example #2 – Case #4.

Table 423 — InputOutputControlByIdentifier positive response message flow example #2 – Case #4

Message direction	server → client		
Message type	Response		
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	InputOutputControlByIdentifier Response SID	$6F_{16}$	IOCBLIPR
#2	dataIdentifier [byte#1] = 01_{16}	01_{16}	IOI_B1
#3	dataIdentifier [byte#2] = 55_{16} (IAC/RPM/PPA/PPB/EGR)	55_{16}	IOI_B2
#4	controlStatusRecord [inputOutputControlParameter] = returnControlToECU	00_{16}	RCTECU
#5	controlStatusRecord [controlState#1] = IAC Pintle Position (9 counts)	09_{16}	CS_1
#6	controlStatusRecord [controlState#2] = RPM (850 U/min)	03_{16}	CS_2
#7	controlStatusRecord [controlState#3] = RPM	52_{16}	CS_3
#8	controlStatusRecord [controlState#4] = Pedal Position A (8 %) Pedal Position B (16 %)	12_{16}	CS_4
#9	controlStatusRecord [controlState#5] = EGR Duty Cycle (35 %)	59_{16}	CS_5

The value transmitted for all parameters in controlState#1 – controlState#5 shall reflect the current state of the system.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

14 Routine functional unit

14.1 Overview

Table 424 specifies the Routine functional unit.

© ISO 2020 – All rights reserved

Table 424 — Routine functional unit

Service	Description
RoutineControl	The client requests to start, stop a routine in the server(s) or requests the routine results.

This functional unit specifies the services of remote activation of routines, as they shall be implemented in servers and client. The following subclause describes two different methods of implementation (methods "A" and "B"). There may be other methods of implementation possible. Methods "A" and "B" shall be used as a guideline for implementation of routine services.

NOTE Each method can feature the functionality to request routine results service after the routine has been stopped. The selection of method and the implementation is the responsibility of the vehicle manufacturer and system supplier.

The following is a brief description of method "A" and "B":

— Method "A":

  — This method is based on the assumption that after a routine has been started by the client in the server's memory the client shall be responsible to stop the routine.

  — The server routine shall be started in the server's memory some time between the completion of the RoutineControl request message that starts the routine and the completion of the first response message (if "positive" based on the server's conditions).

  — The server routine shall be stopped in the server's memory some time after the completion of the StopRoutine request message and the completion of the first response message (if "positive" based on the server's conditions).

  — The client may request routine results after the routine has been stopped.

— Method "B":

  — This method is based on the assumption that after a routine has been started by the client in the server's memory that the server shall be responsible to stop the routine.

  — The server routine shall be started in the server's memory some time between the completion of the RoutineControl request message that starts the routine and the completion of the first response message (if "positive" based on the server's conditions).

  — The server routine shall be stopped any time as programmed or previously initialized in the server's memory.

14.2 RoutineControl (31_{16}) service

14.2.1 Service description

The RoutineControl service is used by the client to execute a defined sequence of steps and obtain any relevant results. There is a lot of flexibility with this service, but typical usage may include functionality such as erasing memory, resetting or learning adaptive data, running a self-test, overriding the normal server control strategy, and controlling a server value to change over time including predefined sequences (e.g. close convertible roof) to name a few. In general, when used to control outputs this service is used for more complex type control whereas inputOutputControlByIdentifier is used for relatively simple (e.g. static) output control.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

14.2.1.1 Overview

The RoutineControl service is used by the client to:

— start a routine;

— stop a routine; and

— request routine results.

A routine is referenced by a 2-byte routineIdentifier.

The following subclauses specify start routine, stop routine, and request routine results referenced by a routineIdentifier.

IMPORTANT — The server and the client shall meet the request and response message behaviour as specified in 8.7.

14.2.1.2 Start a routine referenced by a routineIdentifier

The routine shall be started in the server's memory some time between the completion of the StartRoutine request message and the completion of the first response message if the response message is positive or negative, indicating that the request is already performed or in progress to be performed.

The routines could either be tests that run instead of normal operating code or could be routines that are enabled and executed with the normal operating code running. In particular in the first case, it might be necessary to switch the server in a specific diagnostic session using the DiagnosticSessionControl service or to unlock the server using the SecurityAccess service prior to using the StartRoutine service.

14.2.1.3 Stop a routine referenced by a routineIdentifier

The server routine shall be stopped in the server's memory some time after the completion of the StopRoutine request message and the completion of the first response message if the response message is positive or negative, indicating that the request to stop the routine is already performed or in progress to be performed.

The server routine shall be stopped any time as programmed or previously initialized in the server's memory.

14.2.1.4 Request routine results referenced by a routineIdentifier

This SubFunction is used by the client to request results (e.g. exit status information) referenced by a routineIdentifier and generated by the routine which was executed in the server's memory.

Based on the routine results, which may have been received in the positive response message of the stopRoutine SubFunction parameter (e.g. normal/abnormal Exit With Results) the requestRoutineResults SubFunction shall be used.

An example of routineResults could be data collected by the server, which could not be transmitted during routine execution because of server performance limitations.

14.2.2 Request message

14.2.2.1 Request message definition

Table 425 specifies the request message.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

Table 425 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	RoutineControl Request SID	M	31_{16}	RC
#2	SubFunction = [routineControlType]	M	 00_{16} to FF_{16}	LEV_ RCTP_
 #3 #4	routineIdentifier [] = [byte#1 (MSB) byte#2 (LSB)]	 M M	 00_{16} to FF_{16} 00_{16} to FF_{16}	RI_ B1 B2
 #5 : #n	routineControlOptionRecord[] = [routineControlOption#1 : routineControlOption#m]	 U : U	 00_{16} to FF_{16} : 00_{16} to FF_{16}	RCEOR_ RCO_ : RCO _

14.2.2.2 Request message SubFunction parameter $Level (LEV_) definition

The SubFunction parameters are used by this service to select the control of the routine. Explanations and usage of the possible levels are detailed in Table 426 (suppressPosRspMsgIndicationBit (bit 7) not shown).

Table 426 — Request message SubFunction definition

Bits 6 to 0	Description	Cvt	Mnemonic
00_{16}	**ISOSAEReserved** This value is reserved by this document for future definition.	M	ISOSAERESRVD
01_{16}	**startRoutine** This parameter specifies that the server shall start the routine specified by the routineIdentifier.	M	STR
02_{16}	**stopRoutine** This parameter specifies that the server shall stop the routine specified by the routineIdentifier.	U	STPR
03_{16}	**requestRoutineResults** This parameter specifies that the server shall return result values of the routine specified by the routineIdentifier.	U	RRR
04_{16} to $7F_{16}$	**ISOSAEReserved** This value is reserved by this document for future definition.	M	ISOSAERESRVD

14.2.2.3 Request message data-parameter definition

Table 427 specifies the data-parameters of the request message.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 427 — Request message data-parameter definition

Definition
routineIdentifier This parameter shall identify a server local routine and be out of the range of the defined dataIdentifiers specified in Table F.1.
routineControlOptionRecord This parameter record contains either: — routine entry option parameters, which optionally specify start conditions of the routine (e.g. timeToRun, startUpVariables, etc.), or — routine exit option parameters which optionally specify stop conditions of the routine(e.g. timeToExpireBeforeRoutineStops, variables, etc.), or — request Routine Results option parameters.

14.2.3 Positive response message

14.2.3.1 Positive response message definition

Table 428 specifies the positive response message.

Table 428 — Positive response message definition

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	RoutineControl Response SID	M	71_{16}	RCPR
#2	routineControlType	M	00_{16} to $7F_{16}$	RCTP_
	routineIdentifier [] = [			RI_
#3	byte#1 (MSB)	M	00_{16} to FF_{16}	B1
#4	byte#2 (LSB)]	M	00_{16} to FF_{16}	B2
#5	routineInfo	C_1	00_{16} to FF_{16}	RINF_
	routineStatusRecord[] = [			RSR_
#6	routineStatus#1	U	00_{16} to FF_{16}	RS_
:	:	:	:	:
#n	routineStatus#m]	U	00_{16} to FF_{16}	RS _

C_1: The RoutineInfo byte specifies a scheme (e.g. StartRoutine, StopRoutine, RequestRoutineResults), to allow for generic external test equipment handling of any routine. This parameter is mandatory for any routine where the routineStatusRecord is defined by the ISO/SAE specifications (e.g. ISO 27145-3, SAE J1979-DA, ISO 26021) even if the ISO/SAE defined size of the routineStatusRecord equals "0" data bytes. For routines where the routineStatusRecord is completely defined by the vehicle manufacturer, the support of this parameter is optional. The definition of this byte shall be left to the vehicle manufacturer.

U: The RoutineStatusByte #m shall only be included in the routineStatusRecord[] if specified for the routineIdentifier (RID) by the vehicle manufacturer.

14.2.3.2 Positive response message data-parameter definition

Table 429 specifies the data-parameters of the positive response message.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

Table 429 — Response message data-parameter definition

Definition
routineControlType This parameter is an echo of bits 6 - 0 of the SubFunction parameter from the request message.
routineIdentifier This parameter is an echo of the routineIdentifier from the request message.
routineInfo The RoutineInfo byte encoding is vehicle manufacuter specific and provides a mechanism for the vehicle manufacturer to support generic external test equipment handling of all implemented routines (e.g. if stopRoutine or requestRoutineResults are required) based upon this returned value.
routineStatusRecord This parameter record is used to give to the client either: — additional information about the status of the server following the start of the routine, or — additional information about the status of the server after the routine has been stopped (e.g. total run time, results generated by the routine before stopped, etc.), or — results (exit status information) of the routine, which has been stopped previously in the server.

14.2.4 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 430. The listed negative responses shall be used if the error scenario applies to the server.

Table 430 — Supported negative response codes

NRC	Description	Mnemonic
12_{16}	**SubFunctionNotSupported** This NRC shall be sent if the requested SubFunction is either generally not supported or is not supported for the requested RoutineIdentifier.	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 RoutineControl are not met.	CNC
24_{16}	**requestSequenceError** This NRC shall be returned if: — the routine is currently active and can not be restarted when the 'startRoutine' SubFunction is received (it is up to the vehicle manufacturer whether a given routine can be restarted while active), — the routine is not currently active when the 'stopRoutine' SubFunction is received, — routine results are not available when the 'requestRoutineResults' SubFunction is received (e.g. the requested routineIdentifier has never been started).	RSE

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

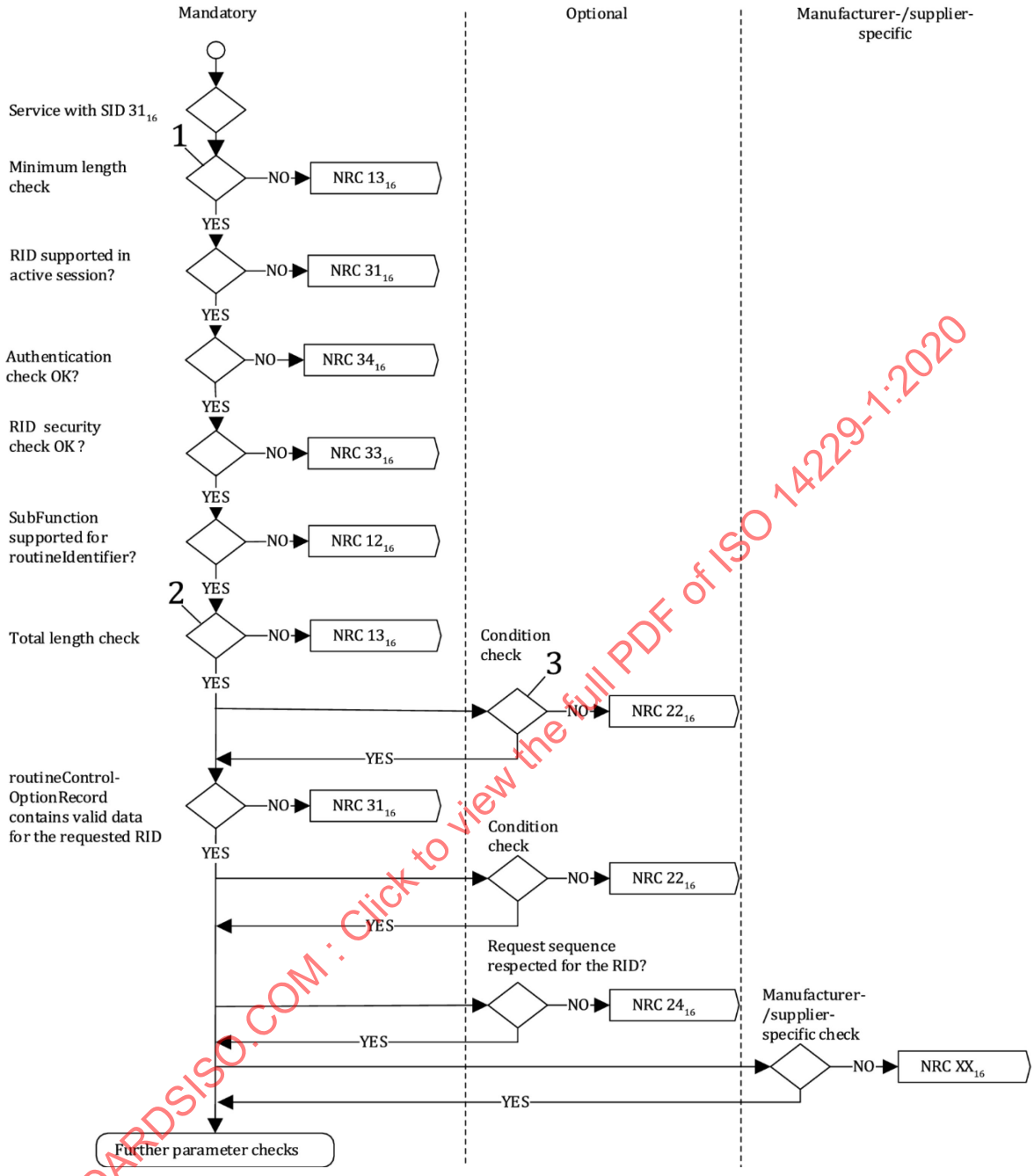
© ISO 2020 – All rights reserved

NRC	Description	Mnemonic
31_{16}	**requestOutOfRange** This NRC shall be returned if: — the server does not support the requested routineIdentifier, — the user optional routineControlOptionRecord contains invalid data for the requested routineIdentifier.	ROOR
33_{16}	**securityAccessDenied** This NRC shall be sent if a client sends a request with a valid secure routineIdentifier and the server's security feature is currently active.	SAD
72_{16}	**GeneralProgrammingFailure** This NRC shall be returned if the server detects an error when performing a routine, which accesses server internal memory. An example is when the routine erases or programs a certain memory location in the permanent memory device (e.g. Flash Memory) and the access to that memory location fails.	GPF

The evaluation sequence is documented in Figure 30.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved



STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

Key

1 at least 4 (SI+SubFunction+RID Parameter)

2 1 byte SI + 1 byte SF + 2 byte RID + nth byte routineControlOptionRecord required for the specific RID

3 optional condition check independent from data content

Figure 30 — NRC handling for RoutineControl service

14.2.5 Message flow example(s) RoutineControl

14.2.5.1 Example #1: SubFunction = startRoutine

This subclause specifies the test conditions to start a routine in the server to continuously test (as fast as possible) all input and output signals on signal intermittence. Then, a technician can "wiggle" all

© ISO 2020 – All rights reserved

wiring harness connectors of the system under test. After the doing this, the routine in the server is stopped. The routineIdentifier references this routine by the routineIdentifier 0201_{16}.

Test conditions: ignition = on, engine = off, vehicle speed = 0 [kph].

The client requests to have a response message by setting the suppressPosRspMsgIndicationBit (bit 7 of the SubFunction parameter) to "FALSE" ('0').

Table 431 specifies the RoutineControl request message flow - example #1.

Table 431 — RoutineControl request message flow - example #1

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	RoutineControl Request SID	31_{16}	RC
#2	SubFunction = startRoutine, suppressPosRspMsgIndicationBit = FALSE	01_{16}	LEV_STR
#3	routineIdentifier [byte#1] (MSB)	02_{16}	RI_B1
#4	routineIdentifier [byte#2] (LSB)	01_{16}	RI_B2

Table 432 specifies the positive response message flow - example #1.

Table 432 — positive response message flow - example #1

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	RoutineControl Response SID	71_{16}	RCPR
#2	routineControlType = startRoutine	01_{16}	STR
#3	routineIdentifier [byte#1] (MSB)	02_{16}	RI_B1
#4	routineIdentifier [byte#2] (LSB)	01_{16}	RI_B2
#5	routineStatusRecord [routineStatus#1] = vehicle manufactuer specific	32_{16}	RRS_

14.2.5.2 Example #2: SubFunction = stopRoutine

This subclause specifies the test conditions to start a routine in the server to continuously test (as fast as possible) all input and output signals on signal intermittence. Then, a technician can "wiggle" all wiring harness connectors of the system under test. After the doing this, the routine in the server is stopped. The routineIdentifier references this routine by the routineIdentifier 0201_{16}.

Test conditions: ignition = on, engine = off, vehicle speed = 0 [kph].

The client requests to have a response message by setting the suppressPosRspMsgIndicationBit (bit 7 of the SubFunction parameter) to "FALSE" ('0').

Table 433 specifies the RoutineControl request message flow - example #2.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

Table 433 — RoutineControl request message flow - example #2

Message direction	client → server		
Message type	Request		
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	RoutineControl Request SID	31_{16}	RC
#2	SubFunction = stopRoutine, suppressPosRspMsgIndicationBit = FALSE	02_{16}	SPR
#3	routineIdentifier [byte#1] (MSB)	02_{16}	RI_B1
#4	routineIdentifier [byte#2] (LSB)	01_{16}	RI_B2

Table 434 specifies the RoutineControl positive response message flow - example #2.

Table 434 — RoutineControl positive response message flow - example #2

Message direction	server → client		
Message type	Response		
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	StopRoutine Response SID	71_{16}	RCPR
#2	routineControlType = stopRoutine	02_{16}	SPR
#3	routineIdentifier [byte#1] (MSB)	02_{16}	RI_B1
#4	routineIdentifier [byte#2] (LSB)	01_{16}	RI_B2
#5	routineStatusRecord [routineStatus#1] = vehicle manufactuer specific	30_{16}	RRS_

14.2.5.3 Example #3: SubFunction = requestRoutineResults

This subclause specifies the test conditions to start a routine in the server to continuously test (as fast as possible) all input and output signals on signal intermittence. Then, a technician can "wiggle" all wiring harness connectors of the system under test. After the doing this, the routine in the server is stopped. The routineIdentifier references this routine by the routineIdentifier 0201_{16}.

Test conditions: ignition = on, engine = off, vehicle speed = 0 [kph].

The client requests to have a response message bysetting the suppressPosRspMsgIndicationBit (bit 7 of the SubFunction parameter) to "FALSE" ('0').

Table 435 specifies the RequestRoutineResults request message flow – example #3.

Table 435 — RequestRoutineResults request message flow – example #3

Message direction	client → server		
Message type	Request		
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	RoutineControl Request SID	31_{16}	RC
#2	SubFunction = requestRoutineResults, suppressPosRspMsgIndicationBit = FALSE	03_{16}	RRR
#3	routineIdentifier [byte#1] (MSB)	02_{16}	RI_B1
#4	routineIdentifier [byte#2] (LSB)	01_{16}	RI_B2

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 436 specifies the RequestRoutineResults positive response message flow – example #3.

Table 436 — RequestRoutineResults positive response message flow - example #3

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	RoutineControl Response SID	71_{16}	RCPR
#2	routineControlType = requestRoutineResults	03_{16}	RRR
#3	routineIdentifier [byte#1] (MSB)	02_{16}	RI_B1
#4	routineIdentifier [byte#2] (LSB)	01_{16}	RI_B2
#5	routineStatusRecord [routineStatus#1] = vehicle manufactuer-specific	30_{16}	RRS_
#6	routineStatusRecord [routineStatus#2] = inputSignal#1	33_{16}	RRS_
:	:	:	:
#n	routineStatusRecord [routineStatus#m] = inputSignal#m	$8F_{16}$	RRS_

14.2.5.4 Example #4: SubFunction = startRoutine with routineControlOption

This subclause specifies the test conditions to start a routine in a transmission control unit to calibrate the gear shift for a certain gear in a special mode. The gear could be any from #1 to #20 and the mode can be bench, stand alone and in-vehicle. The routineIdentifier references this routine by the routineIdentifier 0202_{16}.

Test conditions: ignition = on, engine = off, vehicle speed = 0 [kph].

The client requests to have a response message by setting the suppressPosRspMsgIndicationBit (bit 7 of the SubFunction parameter) to "FALSE" ('0').

Table 437 specifies the RoutineControl request message flow - example #4.

Table 437 — RoutineControl request message flow - example #4

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	RoutineControl Request SID	31_{16}	RC
#2	SubFunction = startRoutine, suppressPosRspMsgIndicationBit = FALSE	01_{16}	STR
#3	routineIdentifier [byte#1] (MSB)	02_{16}	RI_B1
#4	routineIdentifier [byte#2] (LSB)	02_{16}	RI_B2
#5	routineControlOption#1 [selected gear] = vehicle manufacturer-specific	06_{16}	RCO_
#6	routineControlOption#2 [test condition]	01_{16}	RCO_

Table 438 specifies the RoutineControl positive response message flow - example #4.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

Table 438 — RoutineControl positive response message flow - example #4

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	RoutineControl Response SID	71_{16}	RCPR
#2	routineControlType = startRoutine	01_{16}	STR
#3	routineIdentifier [byte#1] (MSB)	02_{16}	RI_B1
#4	routineIdentifier [byte#2] (LSB)	02_{16}	RI_B2
#5	routineStatusRecord [routineStatus#1] = vehicle manufacturer-specific	32_{16}	RRS_
#6	routineStatusRecord [routineStatus#2]= response time	33_{16}	RRS_
:	:	:	:
#n	routineStatusRecord [routineStatus#m]= inputSignal#m	$8F_{16}$	RRS_

15 Upload download functional unit

15.1 Overview

Table 439 specifies the Upload Download functional unit.

Table 439 — Upload Download functional unit

Service	Description
RequestDownload	The client requests the negotiation of a data transfer from the client to the server.
RequestUpload	The client requests the negotiation of a data transfer from the server to the client.
TransferData	The client transmits data to the server (download) or requests data from the server (upload).
RequestTransferExit	The client requests the termination of a data transfer.
RequestFileTransfer	The client requests the negotiation of a file transfer between server and client.

15.2 RequestDownload (34_{16}) service

15.2.1 Service description

The requestDownload service is used by the client to initiate a data transfer from the client to the server (download).

After the server has received the requestDownload request message, the server shall take all necessary actions to receive data before it sends a positive response message.

IMPORTANT — The server and the client shall meet the request and response message behaviour as specified in 8.7.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

15.2.2 Request message

15.2.2.1 Request message definition

Table 440 specifies the request message.

Table 440 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	RequestDownload Request SID	M	34_{16}	RD
#2	dataFormatIdentifier	M	00_{16} to FF_{16}	DFI_
#3	addressAndLengthFormatIdentifier	M	00_{16} to FF_{16}	ALFID
#4 : #(m-1)+4	memoryAddress[] = [byte#1 (MSB) : byte#m]	M : C_1	00_{16} to FF_{16} : 00_{16} to FF_{16}	MA_ B1 : Bm
#n-(k-1) : #n	memorySize[] = [byte#1 (MSB) : byte#k]	M : C_2	00_{16} to FF_{16} : 00_{16} to FF_{16}	MS_ B1 : Bk

C_1: The presence of this parameter depends on address length information parameter of the addressAndLengthFormatIdentifier.

C_2: The presence of this parameter depends on the memory size length information of the addressAndLengthFormatIdentifier.

15.2.2.2 Request message SubFunction parameter $Level (LEV_) definition

This service does not use a SubFunction parameter.

15.2.2.3 Request message data-parameter definition

Table 441 specifies the data-parameters of the request message.

Table 441 — Request message data-parameter definition

Definition
dataFormatIdentifier This data-parameter is a one Byte value with each nibble encoded separately. The high nibble specifies the "compressionMethod" and the low nibble specifies the "encryptingMethod". The value 00_{16} specifies that neither compressionMethod nor encryptingMethod is used. Values other than 00_{16} are vehicle manufacturer specific.
addressAndLengthFormatIdentifier This parameter is a one Byte value with each nibble encoded separately (see Table H.1 for example values): — bit 7 - 4: Length (number of bytes) of the memorySize parameter — bit 3 - 0: Length (number of bytes) of the memoryAddress parameter

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Definition
memoryAddress The parameter memoryAddress is the starting address of the server memory where the data is to be written to. The number of bytes used for this address is defined by the low nibble (bit 3 - 0) of the addressAndLengthFormatIdentifier. Byte#m in the memoryAddress parameter is always the least significant byte of the address being referenced in the server. The most significant byte(s) of the address can be used as a memory identifier. An example of the use of a memory identifier would be a dual processor server with 16 bit addressing and memory address overlap (when a given address is valid for either processor but yields a different physical memory device or internal and external flash is used). In this case, an otherwise unused byte within the memoryAddress parameter can be specified as a memory identifier used to select the desired memory device. Usage of this functionality shall be as defined by vehicle manufacturer/system supplier.
memorySize This parameter shall be used by the server to compare the memory size with the total amount of data transferred during the TransferData service. This increases the programming security. The number of bytes used for this size is defined by the high nibble (bit 7 - 4) of the addressAndLengthFormatIdentifier. If data compression is used, it is vehicle manufacturer specific whether or not the memory size represents the compressed or uncompressed size.

15.2.3 Positive response message

15.2.3.1 Positive response message definition

Table 442 specifies the positive response message.

Table 442 — Positive response message definition

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	RequestDownload Response SID	M	74_{16}	RDPR
#2	lengthFormatIdentifier	M	00_{16} to $F0_{16}$	LFID
#3 : #n	maxNumberOfBlockLength = [byte#1 (MSB) : byte#m]	M : M	00_{16} to FF_{16} : 00_{16} to FF_{16}	MNROB_B1 : Bm

15.2.3.2 Positive response message data-parameter definition

Table 443 specifies the data-parameters of the positive response message.

Table 443 — Response message data-parameter definition

Definition
lengthFormatIdentifier This parameter is a one Byte value with each nibble encoded separately: — bit 7 - 4: Length (number of bytes) of the maxNumberOfBlockLength parameter. — bit 3 - 0: reserved by document, to be set to '0'. The format of this parameter is compatible to the format of the addressAndLengthFormatIdentifier parameter contained in the request message, except that the lower nibble shall be set to '0'.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Definition
maxNumberOfBlockLength This parameter is used by the requestDownload positive response message to inform the client how many data bytes (maxNumberOfBlockLength) to include in each TransferData request message from the client. This length reflects the complete message length, including the service identifier and the data-parameters present in the TransferData request message. This parameter allows the client to adapt to the receive buffer size of the server before it starts transferring data to the server. A server is required to accept transferData requests that are equal in length to its reported maxNumberOfBlockLength. It is server specific what transferData request lengths less than maxNumberOfBlockLength are accepted (if any). The last transferData request within a given block can be required to be less than maxNumberOfBlockLength. It is not allowed for a server to write additional data bytes (i.e. pad bytes) not contained within the transferData message (either in a compressed or uncompressed format), as this would affect the memory address of where the subsequent transferData request data would be written.

15.2.4 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 444. The listed negative responses shall be used if the error scenario applies to the server.

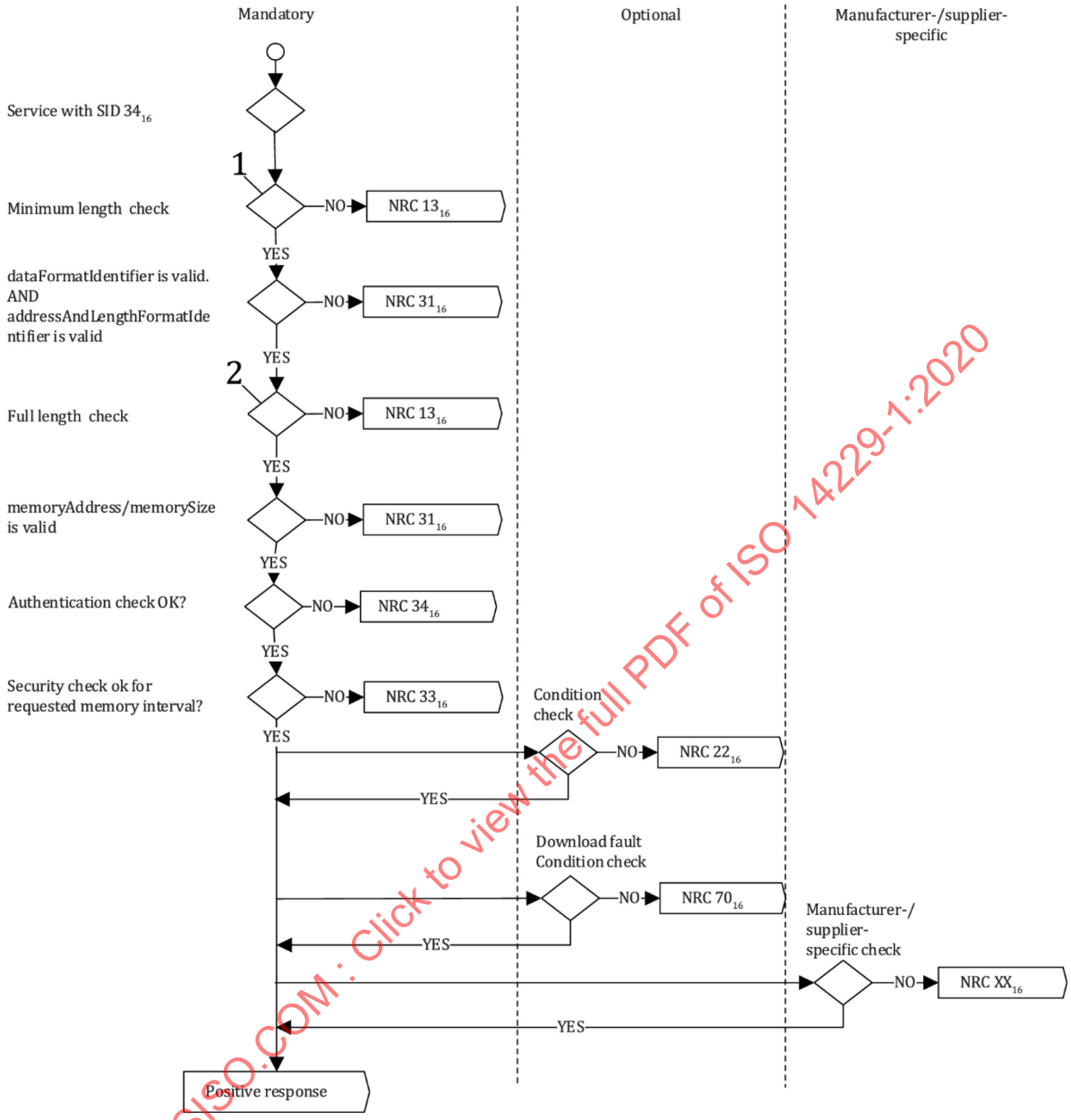
Table 444 — Supported negative response codes

NRC	Description	Mnemonic
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 a server receives a request for this service while in the process of receiving a download of a software or calibration module. This could occur if there is a data size mismatch between the server and the client during the download of a module.	CNC
31_{16}	**requestOutOfRange** This NRC shall be returned if: — the specified dataFormatIdentifier is not valid. — the specified addressAndLengthFormatIdentifier is not valid. — the specified memoryAddress/memorySize is not valid.	ROOR
33_{16}	**securityAccessDenied** This NRC shall be returned if the server is secure (for server's that support the SecurityAccess service) when a request for this service has been received.	SAD
34_{16}	**authenticationRequired** This NRC shall be sent if the dataIdentifier is secured and the client has insufficient rights based on its Authentication state.	AR
70_{16}	**uploadDownloadNotAccepted** This NRC indicates that an attempt to download to a server's memory cannot be accomplished due to some fault conditions.	UDNA

The evaluation sequence is documented in Figure 31.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved



Key

1 at least 5 (SI + DFI_ + ALFID + minimum MA_ + minimum MS_)

2 length can be computed from addressAndLengthFormatIdentifier

Figure 31 — NRC handling for RequestDownload service

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

15.2.5 Message flow example(s) RequestDownload

See 15.5.5 for a complete message flow example.

15.3 RequestUpload (35_{16}) service

15.3.1 Service description

The RequestUpload service is used by the client to initiate a data transfer from the server to the client (upload).

© ISO 2020 – All rights reserved

After the server has received the requestUpload request message the server shall take all necessary actions to send data before it sends a positive response message.

IMPORTANT — The server and the client shall meet the request and response message behaviour as specified in 8.7.

15.3.2 Request message

15.3.2.1 Request message definition

Table 445 specifies the request message.

Table 445 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	RequestUpload Request SID	M	35_{16}	RU
#2	dataFormatIdentifier	M	00_{16} to FF_{16}	DFI_
#3	addressAndLengthFormatIdentifier	M	00_{16} to FF_{16}	ALFID
#4 : #(m-1)+4	memoryAddress[] = [byte#1 (MSB) : byte#m]	M : C_1	00_{16} to FF_{16} : 00_{16} to FF_{16}	MA_ B1 : Bm
#n-(k-1) : #n	memorySize[] = [byte#1 (MSB) : byte#k]	M : C_2	00_{16} to FF_{16} : 00_{16} to FF_{16}	MS_ B1 : Bk

C_1: The presence of this parameter depends on address length information parameter of the addressAndLengthFormatIdentifier.

C_2: The presence of this parameter depends on the memory size length information of the addressAndLengthFormatIdentifier.

15.3.2.2 Request message SubFunction parameter $Level (LEV_) definition

This service does not use a SubFunction parameter.

15.3.2.3 Request message data-parameter definition

Table 446 specifies the data-parameters of the request message.

Table 446 — Request message data-parameter definition

Definition
dataFormatIdentifier This data-parameter is a one Byte value with each nibble encoded separately. The high nibble specifies the "compressionMethod", and the low nibble specifies the "encryptingMethod". The value 00_{16} specifies that neither compressionMethod nor encryptingMethod is used. Values other than 00_{16} are vehicle manufacturer specific.
addressAndLengthFormatIdentifier This parameter is a one Byte value with each nibble encoded separately (see Table H.1 for example values): — bit 7 - 4: Length (number of bytes) of the memorySize parameter — bit 3 - 0: Length (number of bytes) of the memoryAddress parameter

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

Definition
memoryAddress The parameter memoryAddress is the starting address of server memory from which data is to be retrieved. The number of bytes used for this address is defined by the low nibble (bit 3 - 0) of the addressAndLengthFormatIdentifier. Byte#m in the memoryAddress parameter is always the least significant byte of the address being referenced in the server. The most significant byte(s) of the address can be used as a memory identifier. An example of the use of a memory identifier would be a dual processor server with 16 bit addressing and memory address overlap (when a given address is valid for either processor but yields a different physical memory device or internal and external flash is used). In this case, an otherwise unused byte within the memoryAddress parameter can be specified as a memory identifier used to select the desired memory device. Usage of this functionality shall be as defined by vehicle manufacturer/system supplier.
memorySize This parameter shall be used by the server to compare the memory size with the total amount of data transferred during the TransferData service. This increases the programming security. The number of bytes used for this size is defined by the high nibble (bit 4) of the addressAndLengthFormatIdentifier. If data compression is used, it is vehicle manufacturer specific whether or not the memory size represents the compressed or uncompressed size.

15.3.3 Positive response message

15.3.3.1 Positive response message definition

Table 447 specifies the positive response message.

Table 447 — Positive response message definition

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	RequestUpload Response SID	M	75_{16}	RUPR
#2	lengthFormatIdentifier	M	00_{16} to $F0_{16}$	LFID
#3 : #n	maxNumberOfBlockLength = [byte#1 (MSB) : byte#m]	M : M	00_{16} to FF_{16} : 00_{16} to FF_{16}	MNROB_ B1 : Bm

15.3.3.2 Positive response message data-parameter definition

Table 448 specifies the data-parameters of the positive response message.

Table 448 — Response message data-parameter definition

Definition
lengthFormatIdentifier This parameter is a one Byte value with each nibble encoded separately: — bit 7 to 4: Length (number of bytes) of the maxNumberOfBlockLength parameter; — bit 3 to 0: reserved by document, to be set to 0_{16}; The format of this parameter is compatible to the format of the addressAndLengthFormatIdentifier parameter contained in the request message, except that the lower nibble shall be set to 0_{16}.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Definition
maxNumberOfBlockLength This parameter is used by the requestUpload positive response message to inform the client how many data bytes shall be included in each TransferData positive response message from the server. This length reflects the complete message length, including the service identifier and the data-parameters present in the TransferData positve response message. This parameter allows the client to adapt to the send buffer size of the server before the server starts transferring data to the client. A client is required to accept transferData responses that are equal in length to the reported maxNumberOfBlockLength. It is server-specific what transferData response lengths less than maxNumberOfBlockLength are sent (if any). NOTE The last transferData response within a given block can be required to be less than maxNumberOfBlockLength.

15.3.4 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 449. The listed negative responses shall be used if the error scenario applies to the server.

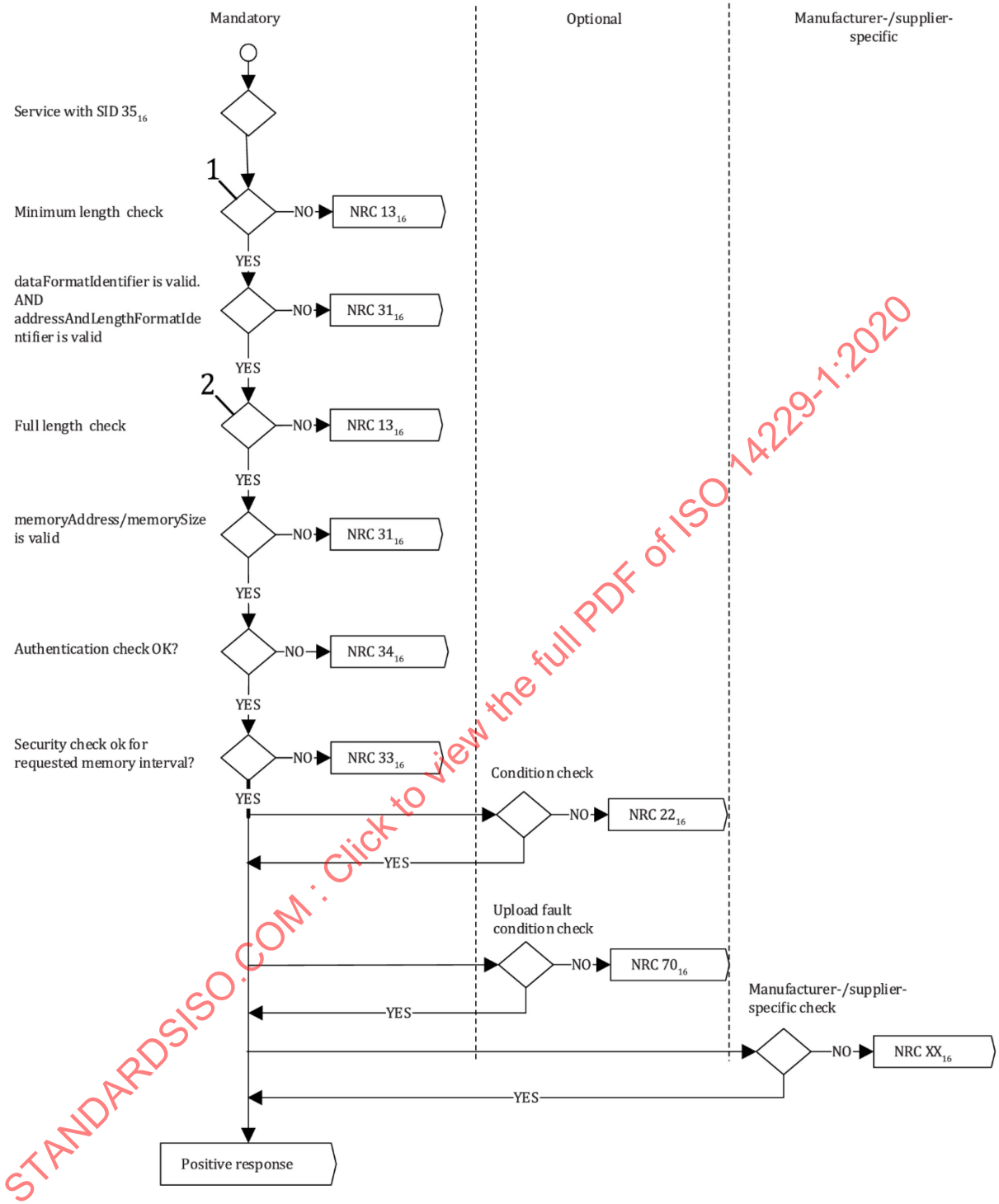
Table 449 — Supported negative response codes

NRC	Description	Mnemonic
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 requestUpload are not met. This could occur if a server receives a request for this service while a requestUpload is already active, but not yet completed.	CNC
31_{16}	**requestOutOfRange** This NRC shall be returned if: — The specified dataFormatIdentifier is not valid; — The specified addressAndLengthFormatIdentifier is not valid; — The specified memoryAddress/memorySize is not valid.	ROOR
33_{16}	**securityAccessDenied** This NRC shall be returned if the server is secure (for server's that support the SecurityAccess service) when a request for this service has been received.	SAD
34_{16}	**authenticationRequired** This NRC shall be sent if the dataIdentifier is secured and the client has insufficient rights based on its Authentication state.	AR
70_{16}	**uploadDownloadNotAccepted** This NRC indicates that an attempt to upload to a server's memory cannot be accomplished due to some fault conditions.	UDNA

The evaluation sequence is documented in Figure 32.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved



STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

Key

1 at least 5 (SI + DFI_ + ALFID + minimum MA_ + minimum MS_)

2 length can be computed from addressAndLengthFormatIdentifier

Figure 32 — NRC handling for RequestUpload service

15.3.5 Message flow example(s) RequestUpload

See 15.5.5 for a complete message flow example.

© ISO 2020 – All rights reserved

15.4 TransferData (36_{16}) service

15.4.1 Service description

The TransferData service is used by the client to transfer data either from the client to the server (download) or from the server to the client (upload).

The data transfer direction is defined by the preceding RequestDownload or RequestUpload service. If the client initiated a RequestDownload the data to be downloaded is included in the parameter(s) transferRequestParameter in the TransferData request message(s). If the client initiated a RequestUpload the data to be uploaded is included in the parameter(s) transferResponseParameter in the TransferData response message(s).

The TransferData service request includes a blockSequenceCounter to allow for an improved error handling in case a TransferData service fails during a sequence of multiple TransferData requests. The blockSequenceCounter of the server shall be initialized to one when receiving a RequestDownload (34_{16}), RequestUpload (35_{16}) or RequestFileTransfer (38_{16}) request message. This means that the first TransferData (36_{16}) request message following the RequestDownload (34_{16}), RequestUpload (35_{16}) or RequestFileTransfer (38_{16}) request message starts with a blockSequenceCounter of one.

IMPORTANT — The server and the client shall meet the request and response message behaviour as specified in 8.7.

15.4.2 Request message

15.4.2.1 Request message definition

Table 450 specifies the request message.

Table 450 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	TransferData Request SID	M	36_{16}	TD
#2	blockSequenceCounter	M	00_{16} to FF_{16}	BSC
#3 : #n	transferRequestParameterRecord[] = [transferRequestParameter#1 : transferRequestParameter#m]	C : U	00_{16} to FF_{16} : 00_{16} to FF_{16}	TRPR_ TRTP_ : TRTP_
C = Conditional: this parameter is mandatory if a download is in progress.				

15.4.2.2 Request message SubFunction parameter $Level (LEV_) definition

This service does not use a SubFunction parameter.

15.4.2.3 Request message data-parameter definition

Table 451 specifies the data-parameters of the request message.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

Table 451 — Request message data-parameter definition

Definition
blockSequenceCounter The blockSequenceCounter parameter value starts at 01_{16} with the first TransferData request that follows the RequestDownload (34_{16}) or RequestUpload (35_{16}) or RequestFileTransfer (38_{16}) service. Its value is incremented by 1 for each subsequent TransferData request. At the value of FF_{16} the blockSequenceCounter rolls over and starts at 00_{16} with the next TransferData request message. Example use cases: — If a TransferData request to download data is correctly received and processed in the server, but the positive response message does not reach the client, then the client would determine an application layer timeout and would repeat the same request (including the same blockSequenceCounter). The server would receive the repeated TransferData request and could determine based on the included blockSequenceCounter that this TransferData request is repeated. The server would send the positive response message immediately without writing the data once again into its memory. — If the TransferData request to download data is not received correctly in the server, then the server would not send a positive response message. The client would determine an application layer timeout and would repeat the same request (including the same blockSequenceCounter). The server would receive the repeated TransferData request and could determine based on the included blockSequenceCounter that this is a new TransferData. The server would process the service and would send the positive response message. — If a TransferData request to upload data is correctly received and processed in the server but the positive response message does not reach the client, then the client would determine an application layer timeout and would repeat the same request (including the same blockSequenceCounter). The server would receive the repeated TransferData request and could determine based on the included blockSequenceCounter that this TransferData request is repeated. The server would send the positive response message immediately accessing the previously provided data once again in its memory. — If the TransferData request to upload data is not received correctly in the server, then the server would not send a positive response message. The client would determine an application layer timeout and would repeat the same request (including the same blockSequenceCounter). The server would receive the repeated TransferData request and could determine based on the included blockSequenceCounter that this is a new TransferData. The server would process the service and would send the positive response message.
transferRequestParameterRecord This parameter record contains parameter(s) which are required by the server to support the transfer of data. Format and length of this parameter(s) are vehicle manufacturer specific. EXAMPLE For a download, the transferRequestParameterRecord include the data to be transferred.

15.4.3 Positive response message

15.4.3.1 Positive response message definition

Table 452 specifies the positive response message.

Table 452 — Positive response message definition

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	TransferData Response SID	M	76_{16}	TDPR
#2	blockSequenceCounter	M	00_{16} to FF_{16}	BSC

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#3 : #n	transferResponseParameterRecord[] = [transferResponseParameter#1 : transferResponseParameter#m]	C : U	00_{16} to FF_{16} : 00_{16} to FF_{16}	TREPR_ TREP_ : TREP
C = Conditional: this parameter is mandatory if an upload is in progress.				

15.4.3.2 Positive response message data-parameter definition

Table 453 specifies the data-parameters of the positive reponse message.

Table 453 — Response message data-parameter definition

Definition
blockSequenceCounter This parameter is an echo of the blockSequenceCounter parameter from the request message.
transferResponseParameterRecord This parameter shall contain parameter(s), which are required by the client to support the transfer of data. Format and length of this parameter(s) are vehicle manufacturer specific. Examples: For a download, the parameter transferResponseParameterRecord could include a checksum computed by the server. For an upload, the parameter transferResponseParameterRecord include the uploaded data. For a download, the parameter transferResponseParameterRecord should not repeat the transferRequestParameterRecord.

15.4.4 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 454. The listed negative responses shall be used if the error scenario applies to the server.

Table 454 — Supported negative response codes

NRC	Description	Mnemonic
13_{16}	**incorrectMessageLengthOrInvalidFormat** This NRC shall be sent if the length of the message is wrong(e.g. message length does not meet requirements of maxNumberOfBlockLength parameter returned in the positive response to the requestDownload service).	IMLOIF
24_{16}	**requestSequenceError** The server shall use this response code: if the RequestDownload, RequestUpload or RequestFileTransfer service is not active when a request for this service is received; — if the RequestDownload, RequestUpload or RequestFileTransfer service is active, but the server has already received all data as determined by the memorySize parameter in the active RequestDownload, RequestUpload or RequestFileTransfer service. The repetition of a TransferData request message with a blockSequenceCounter equal to the one included in the previous TransferData request message shall be accepted by the server.	RSE

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

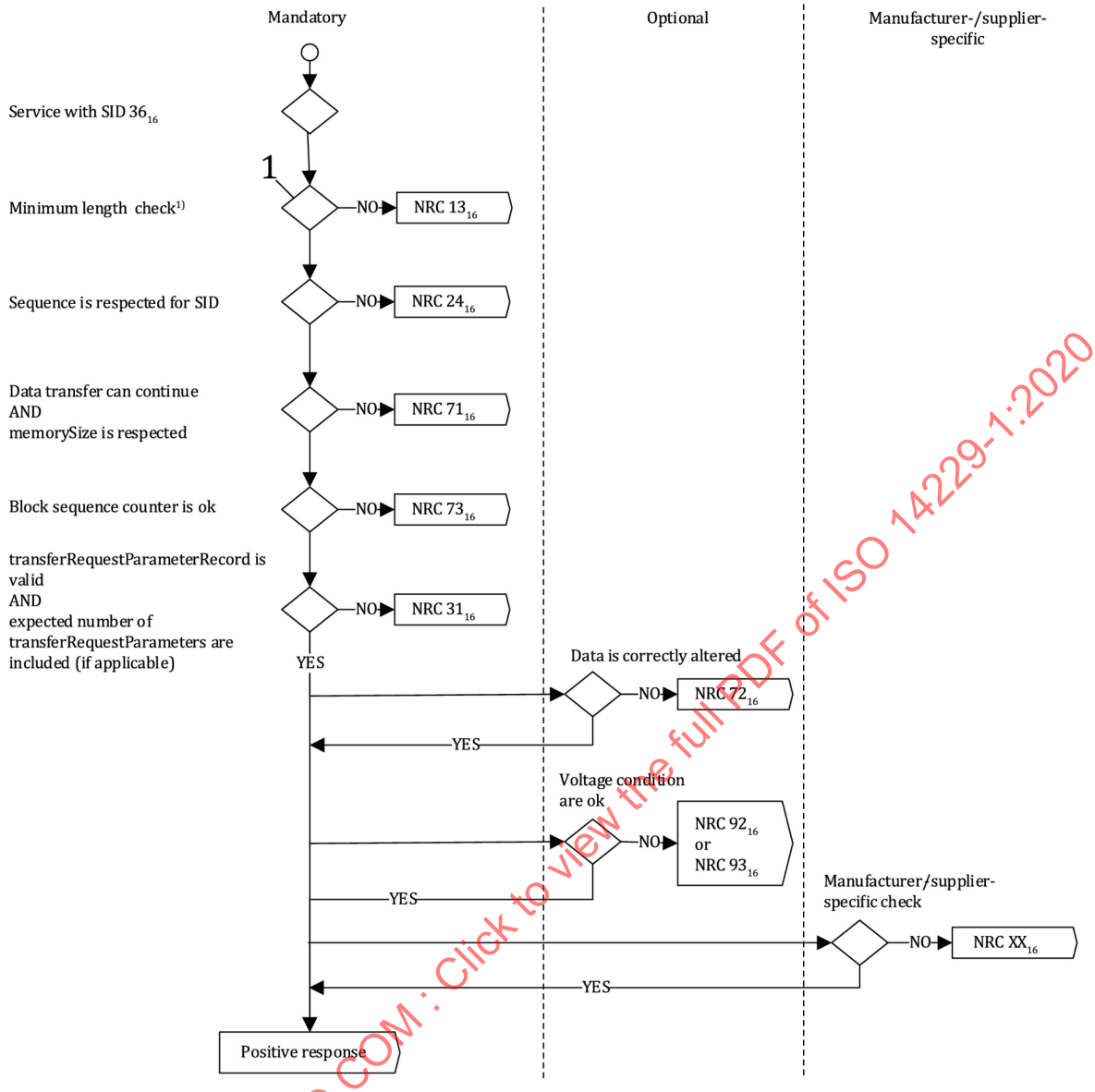
© ISO 2020 – All rights reserved

NRC	Description	Mnemonic
31_{16}	**requestOutOfRange** This NRC shall be returned if: — the transferRequestParameterRecord contains additional control parameters (e.g. additional address information) and this control information is invalid. — the transferRequestParameterRecord is not consistent with the requestDownload, requestUpload or RequestFileTransfer service parameter maxNumberOfBlockLength. — the transferRequestParameterRecord is not consistent with the server's memory alignment constraints.	ROOR
71_{16}	**transferDataSuspended** This NRC shall be returned if the download module length does not meet the requirements of the memorySize parameter sent in the request message of the requestDownload service.	TDS
72_{16}	**generalProgrammingFailure** This NRC shall be returned if the server detects an error when erasing or programming a memory location in the permanent memory device (e.g. Flash Memory) during the download of data.	GPF
73_{16}	**wrongBlockSequenceCounter** This NRC shall be returned if the server detects an error in the sequence of the blockSequenceCounter. The repetition of a TransferData request message with a blockSequenceCounter equal to the one included in the previous TransferData request message shall be accepted by the server.	WBSC
$92_{16}/93_{16}$	**voltageTooHigh/voltageTooLow** This return code shall be sent as applicable if the voltage measured at the primary power pin of the server is out of the acceptable range for downloading data into the server's permanent memory (e.g. Flash Memory).	VTH/VTL

The evaluation sequence is documented in Figure 33.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved



Key

1 shall be 2 if a RequestUpload is in progress (SI + BSC), at least 3 if a RequestDownload is in progress (SI + BSC + minimum TRPR_)

Figure 33 — NRC handling for TransferData service

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

15.4.5 Message flow example(s) TransferData

See 15.5.5 for a complete message flow example.

15.5 RequestTransferExit (37_{16}) service

15.5.1 Service description

This service is used by the client to terminate a data transfer between client and server (upload or download).

IMPORTANT — The server and the client shall meet the request and response message behaviour as specified in 8.7.

© ISO 2020 – All rights reserved

15.5.2 Request message

15.5.2.1 Request message definition

Table 455 specifies the request message.

Table 455 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	RequestTransferExit Request SID	M	37_{16}	RTE
#2 : #n	transferRequestParameterRecord[] = [transferRequestParameter#1 : transferRequestParameter#m]	U : U	00_{16} to FF_{16} : 00_{16} to FF_{16}	TRPR_ TRTP_ : TRTP_

15.5.2.2 Request message SubFunction parameter $Level (LEV_) definition

This service does not use a SubFunction parameter.

15.5.2.3 Request message data-parameter definition

Table 456 specifies the data-parameters of the request message.

Table 456 — Request message data-parameter definition

Definition
transferRequestParameterRecord This parameter record contains parameter(s), which are required by the server to support the transfer of data. Format and length of this parameter(s) are vehicle manufacturer specific.

15.5.3 Positive response message

15.5.3.1 Positive response message definition

Table 457 specifies the positive response message.

Table 457 — Positive response message definition

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	RequestTransferExit Response SID	M	77_{16}	RTEPR
#2 : #n	transferResponseParameterRecord[] = [transferResponseParameter#1 : transferResponseParameter#m]	U : U	00_{16} to FF_{16} : 00_{16} to FF_{16}	TREPR_ TREP_ : TREP

15.5.3.2 Positive response message data-parameter definition

Table 458 specifies the data-parameters of the positive response message.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 458 — Response message data-parameter definition

Definition
transferResponseParameterRecord This parameter shall contain parameter(s) which are required by the client to support the transfer of data. Format and length of this parameter(s) are vehicle manufacturer specific.

15.5.4 Supported negative response codes (NRC_)

The following negative response codes shall be implemented for this service. The circumstances under which each negative response code would occur are documented in Table 459. The listed negative responses shall be used if the error scenario applies to the server.

Table 459 — Supported negative response codes

NRC	Description	Mnemonic
13_{16}	**incorrectMessageLengthOrInvalidFormat** This NRC shall be returned if the length of the message is wrong.	IMLOIF
24_{16}	**requestSequenceError** This NRC shall be returned if: — the programming process is not completed when a request for this service is received; — the RequestDownload, RequestUpload or RequestFileTransfer service is not active.	RSE
31_{16}	**requestOutOfRange** This NRC shall be returned if the transferRequestParameterRecord contains invalid data.	ROOR
72_{16}	**generalProgrammingFailure** This NRC shall be returned if the server detects an error when finalizing the data transfer between the client and server (e.g. via an integrity check).	GPF

The evaluation sequence is documented in Figure 34.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

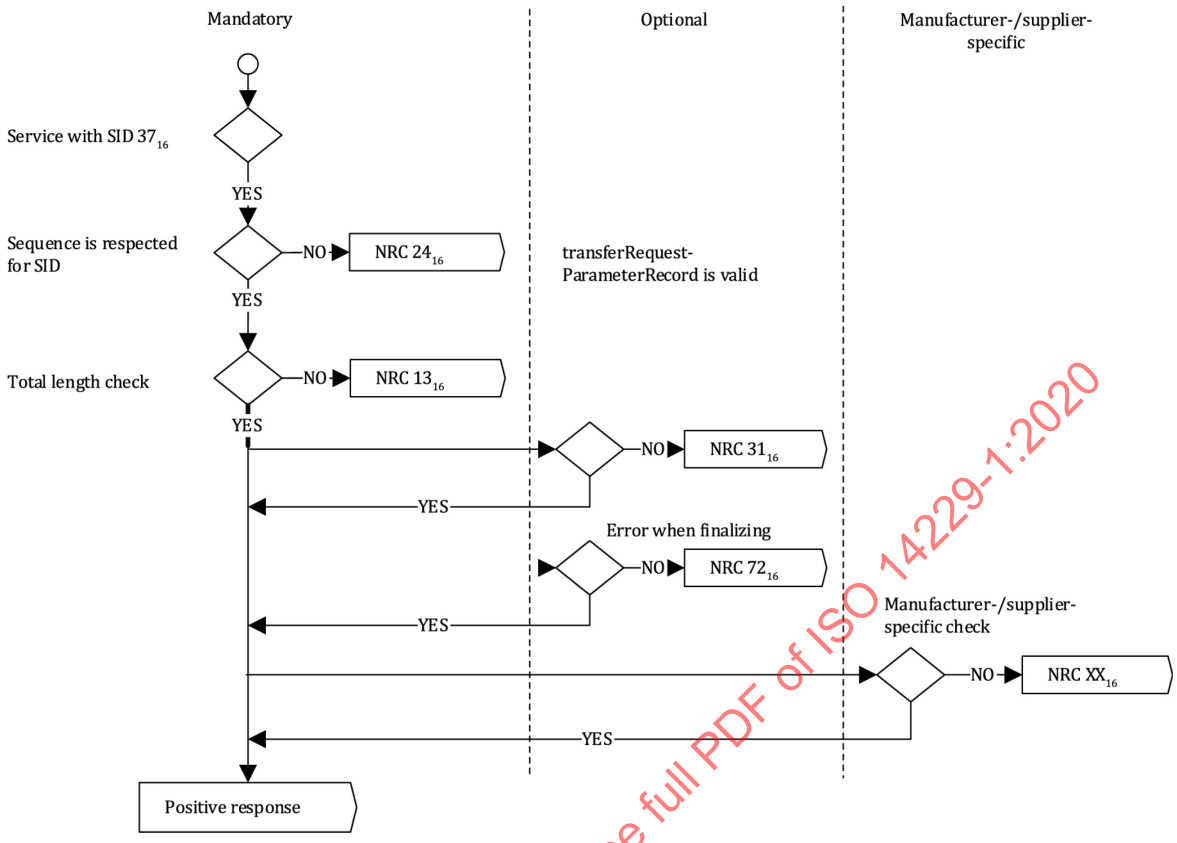


Figure 34 — NRC handling for RequestTransferExit service

15.5.5 Message flow example(s) for downloading/uploading data

15.5.5.1 Download data to a server

15.5.5.1.1 Assumptions

This subclause specifies the conditions to transfer data (download) from the client to the server.

The example consists of three steps.

In the 1st step the client and the server execute a RequestDownload service. With this service the following information is exchanged as parameters in the request and positive response message between client and the server.

Table 460 specifies the transferRequestParameter values.

Table 460 — Definition of transferRequestParameter values

Data parameter name	Data parameter value(s)	Data parameter description
memoryAddress (3 bytes)	602000_{16}	memoryAddress (start) to download data to
dataFormatIdentifier	11_{16}	dataFormatIdentifier: — compressionMethod: $1X_{16}$ — encryptingMethod: $X1_{16}$

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Data parameter name	Data parameter value(s)	Data parameter description
MemorySize (3 bytes)	$00FFFF_{16}$	MemorySize (65 535 bytes) This parameter value shall be used by the server to compare to the actual number of bytes transferred during the execution of the requestTransferExit service.

Table 461 specifies the transferResponseParameter value.

Table 461 — Definition of transferResponseParameter value

Data parameter name	Data parameter value(s)	Data parameter description
maximumNumberOfBlockLength	0081_{16}	maximumNumberOfBlockLength: (serviceId + BlockSequenceCounter (1 byte) + 127 server data bytes = 129 data bytes)

In the 2nd step the client transfers 65 535 bytes of data to the flash memory starting at memoryaddress 602000_{16} to the server.

In the 3rd step the client terminates the data transfer to the server with a requestTransferExit service.

Test conditions: ignition = on, engine = off, vehicle speed = 0 [kph].

It is assumed, that for this example the server supports a three byte memoryAddress and a three byte MemorySize. If the MemorySize contains the uncompressed size, the number of TransferData services with 127 data bytes can not be calculated because the compression method and its compression ratio is not standardized. If the MemorySize contains the compressed size, the total number of TransferData services with 127 data bytes would be 516, followed by a single TransferData request with three bytes. Therefore, it is assumed that the last TransferData request message contains a blockSequenceCounter equal to 05_{16}.

15.5.5.1.2 Step #1: Request for download

Table 462 specifies the RequestDownload request message flow example.

Table 462 — RequestDownload request message flow example

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	RequestDownload Request SID	34_{16}	RD
#2	dataFormatIdentifier	11_{16}	DFI
#3	addressAndLengthFormatIdentifier	33_{16}	ALFID
#4	memoryAddress [byte#1] (MSB)	60_{16}	MA_B1
#5	memoryAddress [byte#2]	20_{16}	MA_B2
#6	memoryAddress [byte#3] (LSB)	00_{16}	MA_B3
#7	MemorySize [byte#1] (MSB)	00_{16}	UCMS_B1
#8	MemorySize [byte#2]	FF_{16}	UCMS_B2
#9	MemorySize [byte#3] (LSB)	FF_{16}	UCMS_B3

Table 463 specifies RequestDownload positive response message flow example.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 463 — RequestDownload positive response message flow example

Message direction		server → client	
Message type		Response	
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	RequestDownload Response SID	74_{16}	RDPR
#2	LengthFormatIdentifier	20_{16}	LFID
#3	maxNumberOfBlockLength [byte#1] (MSB)	00_{16}	MNROB_B1
#4	maxNumberOfBlockLength [byte#2] (LSB)	81_{16}	MNROB_B1

15.5.5.1.3 Step #2: Transfer data

Table 464 specifies TransferData request message flow example.

Table 464 — TransferData request message flow example

Message direction		client → server	
Message type		Request	
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	TransferData Request SID	36_{16}	TD
#2	blockSequenceCounter	01_{16}	BSC
#3	transferRequestParameterRecord [transferRequestParameter#1] = dataByte#3	XX_{16}	TRTP_1
:	:	:	:
#129	transferRequestParameterRecord [transferRequestParameter#127] = dataByte#129	XX_{16}	TRTP_127

Table 465 specifies TransferData positive response message flow example.

Table 465 — TransferData positive response message flow example

Message direction		server → client	
Message type		Response	
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	TransferData Response SID	76_{16}	TDPR
#2	blockSequenceCounter	01_{16}	BSC

Table 466 specifies TransferData request message flow example.

Table 466 — TransferData request message flow example

Message direction		client → server	
Message type		Request	
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	TransferData Request SID	36_{16}	TD
#2	blockSequenceCounter	05_{16}	BSC

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Message direction	client → server		
Message type	Request		
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#3	transferRequestParameterRecord [transferRequestParameter#1] = dataByte#3	XX_{16}	TRTP_1
:	:	:	:
#n+2	transferRequestParameterRecord [transferRequestParameter#n-2] = dataByte#n	XX_{16}	TRTP_n-2

Table 467 specifies TransferData positive response message flow example.

Table 467 — TransferData positive response message flow example

Message direction	server → client		
Message type	Response		
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	TransferData Response SID	76_{16}	TDPR
#2	blockSequenceCounter	05_{16}	BSC

15.5.5.1.4 Step #3: Request Transfer exit

Table 468 specifies RequestTransferExit request message flow example.

Table 468 — RequestTransferExit request message flow example

Message direction	client → server		
Message type	Request		
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	RequestTransferExit Request SID	37_{16}	RTE

Table 469 specifies RequestTransferExit positive response message flow example.

Table 469 — RequestTransferExit positive response message flow example

Message direction	server → client		
Message type	Response		
A_Data byte	Description (all values are in hexadecimal)	Byte value	Mnemonic
#1	RequestTransferExit Response SID	77_{16}	RTEPR

15.5.5.2 Upload data from a server

This subclause specifies the conditions to transfer data (upload) from a server to the client.

The example consists of three steps.

In the 1st step the client and the server execute a requestUpload service. With this service, the following information (see Table 470) is exchanged, as parameters in the request and positive response message between client and the server.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

Table 470 — Definition of transferRequestParameter values

Data parameter name	Data value(s)	Data parameter description
memoryAddress (3 bytes)	201000_{16}	memoryAddress (start) to upload data from
dataFormatIdentifier	11_{16}	dataFormatIdentifier — compressionMethod: $1X_{16}$ — encryptingMethod: $X1_{16}$
MemorySize (3 bytes)	$0001FF_{16}$	MemorySize (511 bytes) This parameter value shall indicate how many data bytes shall be transferred and shall be used by the server to compare to the actual number of bytes transferred during execution of the requestTransferExit service.

Table 471 specifies Definition of transferResponseParameter value.

Table 471 — Definition of transferResponseParameter value

Data parameter name	Data value(s)	Data parameter description
maximumNumberOfBlockLength	0081_{16}	maximumNumberOfBlockLength: (serviceId + BlockSequenceCounter (1 byte) + 127 server data bytes = 129 data bytes)

In the 2nd step the server transfers 511 data bytes (4 transferData services with 129 (127 server data bytes + 1 ServiceId data byte + 1 blockSequenceCounter byte) data bytes and 1 transferData service with 5 (3 server data bytes + 1 serviceId data byte + 1 blockSequenceCounter byte) data bytes from the external RAM starting at memoryaddress 201000_{16} in the server.

In the 3rd step the client terminates the data transfer to the server with a requestTransferExit service.

Test conditions: ignition = on, engine = off, vehicle speed = 0 [kph]

It is assumed, that for this example the server supports a three byte memoryAddress and a three byte MemorySize. Furthermore it is assumed that the server supports a blockSequenceCounter in the TransferData (36_{16}) service.

15.5.5.2.1 Step #1: Request for upload

Table 472 specifies RequestUpload request message flow example.

Table 472 — RequestUpload request message flow example

Message direction	**client → server**		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	RequestUpload Request SID	35_{16}	RU
#2	dataFormatIdentifier	11_{16}	DFI
#3	addressAndLengthFormatIdentifier	33_{16}	ALFID
#4	memoryAddress [byte#1] (MSB)	20_{16}	MA_B1

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Message direction	client → server		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#5	memoryAddress [byte#2]	10_{16}	MA_B2
#6	memoryAddress [byte#3] (LSB)	00_{16}	MA_B3
#7	MemorySize [byte#1] (MSB)	00_{16}	UCMS_B1
#8	MemorySize [byte#2]	01_{16}	UCMS_B2
#9	MemorySize [byte#3] (LSB)	FF_{16}	UCMS_B3

Table 473 specifies RequestUpload positive response message flow example.

Table 473 — RequestUpload positive response message flow example

Message direction	server → client		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	RequestUpload Response SID	75_{16}	RUPR
#2	lengthFormatIdentifier	20_{16}	LFID
#3	maxNumberOfBlockLength [byte#1] (MSB)	00_{16}	MNROB_B1
#4	maxNumberOfBlockLength [byte#2] (LSB)	81_{16}	MNROB_B1

15.5.5.2.2 Step #2: Transfer data

Table 474 specifies TransferData request message flow example.

Table 474 — TransferData request message flow example

Message direction	client → server		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	TransferData Request SID	36_{16}	TD
#2	blockSequenceCounter	01_{16}	BSC

Table 475 specifies TransferData positive response message flow example.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 475 — TransferData positive response message flow example

Message direction	server → client		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	TransferData Response SID	76_{16}	TDPR
#2	blockSequenceCounter	01_{16}	BSC
#3 : #129	transferResponseParameterRecord [transferResponseParameter#1] = dataByte3 : transferResponseParameterRecord [transferResponseParameter#127] = dataByte129	XX_{16} : XX_{16}	TREP_1 : TREP_127

Table 476 specifies TransferData request message flow example.

Table 476 — TransferData request message flow example

Message direction	client → server		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	TransferData Request SID	36_{16}	TD
#2	blockSequenceCounter	05_{16}	BSC

Table 477 specifies TransferData positive response message flow example.

Table 477 — TransferData positive response message flow example

Message direction	server → client		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	TransferData Response SID	76_{16}	TDPR
#2	blockSequenceCounter	05_{16}	BSC
#3 : #5	transferResponseParameterRecord [transferResponseParameter#1] = dataByte3 : transferResponseParameterRecord [transferResponseParameter#3] = dataByte5	XX_{16} : XX_{16}	TREP_1 : TREP_3

15.5.5.2.3 Step #3: Request Transfer exit

Table 478 specifies RequestTransferExit request message flow example.

Table 478 — RequestTransferExit request message flow example

Message direction	client → server		
Message type	**Request**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	RequestTransferExit Request SID	37_{16}	RTE

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 479 specifies RequestTransferExit positive response message flow example.

Table 479 — RequestTransferExit positive response message flow example

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	RequestTransferExit Response SID	77_{16}	RTEPR

15.6 RequestFileTransfer (38_{16}) service

15.6.1 Service description

The requestFileTransfer service is used by the client to initiate a file data transfer from either the client to the server or from the server to the client (download or upload). Additionally, this service has capabilities to retrieve information about the file system.

This service is intended as an alternative solution to the RequestDownload and RequestUpload service supporting data upload and download functionality if a server implements a file system for data storage. When configuring a download or upload process to or from a file system, the RequestFileTransfer service shall be used replacing the RequestDownload or RequestUpload. The actual data transfer and termination of the data transfer are implemented by using the TransferData and RequestTransferExit as used with the RequestDownload or RequestUpload service. This service also includes functionality for deleting files or directories on the server's file system. For this use case the TransferData and RequestTransferExit service are not applicable.

After the server has received the RequestFileTransfer request message the server shall take all necessary actions to receive or transmit data before it sends a positive response message.

IMPORTANT — The server and the client shall meet the request and response message behaviour as specified in 8.7.

15.6.2 Request message

15.6.2.1 Request message definition

Table 480 specifies the request message.

Table 480 — Request message definition

A_Data byte	**Parameter Name**	**Cvt**	**Byte value**	**Mnemonic**
#1	RequestFileTransfer Request SID	M	38_{16}	RFT
#2	modeOfOperation	M	01_{16} to 06_{16}	MOOP
#3 #4	filePathAndNameLength [byte#1 (MSB) byte#2] (LSB)	M M	00_{16} to FF_{16} 00_{16} to FF_{16}	FPL_B1 FPL_B2
#5 : #5+n-1	filePathAndName = [byte#1 (MSB) : byte#n]	M : C_1	00_{16} to FF_{16} : 00_{16} to FF_{16}	FP_B1 : FP_Bn
#5+n	dataFormatIdentifier	C_2	00_{16} to FF_{16}	DFI_
#5+n+1	fileSizeParameterLength	C_2	00_{16} to FF_{16}	FSL

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#5+n+2 : #5+n+2+k-1	fileSizeUnCompressed= [byte#1 (MSB) : byte#k]	C_2 : $C_{2,3}$	00_{16} to FF_{16} : 00_{16} to FF_{16}	FSUC_B1 : FSUC_Bk
#5+n+2+k : #5+n+1+2k	fileSizeCompressed= [byte#1 (MSB) : byte#k]	C_2 : $C_{2,3}$	00_{16} to FF_{16} : 00_{16} to FF_{16}	FSC_B1 : FSC_Bk

C_1: The length (number of bytes) of this message parameter is defined by the filePathAndNameLength parameter.

C_2: The presence of these parameters depends on the modeOfOperation parameter.

C_3: The length (number of bytes) of this message parameter is defined by the fileSizeParameterLength.

15.6.2.2 Request message SubFunction parameter $Level (LEV_) definition

This service does not use a SubFunction parameter.

15.6.2.3 Request message data-parameter definition

Table 481 specifies the data-parameters of the request message.

Table 481 — Request message data-parameter definition

Definition
modeOfOperation This data-parameter specifies the type of operation to be applied to the file or directory indicated in the filePathAndName parameter. The values of the data-parameter shall be defined as specified in Annex G.
filePathAndNameLength Specifies the length in byte for the parameter filePath.
filePathAndName Specifies the file system location of the server where the file which shall be added, deleted, replaced or read from depending on the parameter modeOfOperation parameter. In addition this parameter includes the file name of the file which shall be added, deleted, replaced or read as part of the file path. If the modeOfOperation parameter equals 05_{16} (ReadDir), this parameter indicates the directory to be read. Each byte of this parameter shall be encoded in ASCII format.
dataFormatIdentifier This data-parameter is a one Byte value with each nibble encoded separately. The high nibble specifies the "compressionMethod, and the low nibble specifies the "encryptingMethod". The value 00_{16} specifies that neither compressionMethod nor encryptingMethod is used. Values other than 00_{16} are vehicle manufacturer specific. If the modeOfOperation parameter equals to 02_{16} (DeleteFile) and 05_{16} (ReadDir) this parameter shall not be included in the request message.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Definition
fileSizeParameterLength Specifies the length in bytes for both parameters fileSizeUncompressed and fileSizeCompressed. If the modeOfOperation parameter equals to 02_{16} (DeleteFile), 04_{16} (ReadFile) or 05_{16} (ReadDir) this parameter shall not be included in the request message.
fileSizeUncompressed Specifies the size of the uncompressed file in bytes. If the modeOfOperation parameter equals 02_{16} (DeleteFile), 04_{16} (ReadFile) or 05_{16} (ReadDir) this parameter shall not be included in the request message.
fileSizeCompressed Specifies the size of the compressed file in bytes. If an uncompressed file is transferred all bytes of this parameter shall be set to the size information used in the parameter fileSizeUncompressed. If the modeOfOperation parameter equals to 02_{16} (DeleteFile), 04_{16} (ReadFile) or 05_{16} (ReadDir) this parameter shall not be included in the request message.

15.6.3 Positive response message

15.6.3.1 Positive response message definition

Table 482 specifies the positive response message.

Table 482 — Positive response message definition

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	RequestFileTransfer Response SID	M	78_{16}	RRFT
#2	modeOfOperation	M	01_{16} to 06_{16}	MOOP
#3	lengthFormatIdentifier	C_1	00_{16} to FF_{16}	LFID
#4 : #4+(m-1)	maxNumberOfBlockLength = [byte#1 (MSB) : byte#m]	 $C_{1,2}$: $C_{1,2}$	 00_{16} to FF_{16} : 00_{16} to FF_{16}	MNROB_ B1 : Bm
#4+m	dataFormatIdentifier	C_1	00_{16} to FF_{16}	DFI_
#4+m+1 #4+m+2	fileSizeOrDirInfoParameterLength [byte#1 (MSB) byte#2 (LSB)]	 C_1 C_1	 00_{16} to FF_{16} 00_{16} to FF_{16}	 FSDIL_B1 FSDIL_B2
#4+m+3 : #4+m+3+k-1	fileSizeUncompressedOrDirInfoLength= [byte#1 (MSB) : byte#k]	 $C_{1,3}$: $C_{1,3}$	 00_{16} to FF_{16} : 00_{16} to FF_{16}	 FSUDIL_B1 : FSUDIL_Bk
#4+m+3+k : #4+m+3+2k-1	fileSizeCompressed= [byte#1 (MSB) : byte#k]	 $C_{1,3}$: $C_{1,3}$	 00_{16} to FF_{16} : 00_{16} to FF_{16}	 FSC_B1 : FSC_Bk

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#4+m+4+2k	filePosition[Byte1] (MSB)	C_1	00_{16} to FF_{16}	FPOS_B1
#4+m+5+2k	filePosition[Byte2]	C_1	00_{16} to FF_{16}	FPOS_B2
#4+m+6+2k	filePosition[Byte3]	C_1	00_{16} to FF_{16}	FPOS_B3
#4+m+7+2k	filePosition[Byte4]	C_1	00_{16} to FF_{16}	FPOS_B4
#4+m+8+2k	filePosition[Byte5]	C_1	00_{16} to FF_{16}	FPOS_B5
#4+m+9+2k	filePosition[Byte6]	C_1	00_{16} to FF_{16}	FPOS_B6
#4+m+10+2k	filePosition[Byte7]	C_1	00_{16} to FF_{16}	FPOS_B7
#4+m+11+2k	filePosition[Byte8] (LSB)	C_1	00_{16} to FF_{16}	FPOS_B8

C_1: The presence of these parameters depends on the modeOfOperation parameter.

C_2: The length (number of bytes) of this message parameter is defined by the lengthFormatIdentifier parameter.

C_3: The length (number of bytes) of this message parameter is defined by the fileSizeOrDirInfoParameterLength parameter.

15.6.3.2 Positive response message data-parameter definition

Table 483 specifies the data-parameters of the positive response message.

Table 483 — Response message data-parameter definition

Definition
modeOfOperation This is parameter echoes the value of the request.
lengthFormatIdentifier Specifies the length (number of bytes) of the maxNumberOfBlockLength parameter. If the modeOfOperation parameter equals to 02_{16} (DeleteFile) this parameter shall not be included in the response message.
maxNumberOfBlockLength This parameter is used by the requestFileTransfer positive response message to inform the client how many data bytes (maxNumberOfBlockLength) to include in each TransferData request message from the client or how many data bytes the server will include in a TransferData positive response when uploading data. This length reflects the complete message length, including the service identifier and the data parameters present in the TransferData request message or positive response message. This parameter allows either the client to adapt to the receive buffer size of the server before it starts transferring data to the server or to indicate how many data bytes will be included in each TransferData positive response in the event that data is uploaded. A server is required to accept transferData requests that are equal in length to its reported maxNumberOfBlockLength. It is server specific what transferData request lengths less than maxNumberOfBlockLength are accepted (if any). NOTE The last transferData request within a given block can be required to be less than maxNumberOfBlockLength. It is not allowed for a server to write additional data bytes (i.e. pad bytes) not contained within the transferData message (either in a compressed or uncompressed format), as this would affect the memory address of where the subsequent transferData request data would be written. If the modeOfOperation parameter equals to 02_{16} (DeleteFile) this parameter shall be not be included in the response message.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Definition
dataFormatIdentifier This is parameter echoes the value of the request. If the modeOfOperation parameter equals to 02_{16} (DeleteFile) this parameter shall not be included in the response message.) If the modeOfOperation parameter equals to 05_{16} (ReadDir) the value of this parameter shall be equal to 00_{16}.
fileSizeOrDirInfoParameterLength Specifies the length in bytes for both parameters fileSizeUncompressedOrDirInfoLength and fileSizeCompressed. If the modeOfOperation parameter equals to 01_{16} (AddFile), 02_{16} (DeleteFile), 03_{16} (ReplaceFile) or 06_{16} (ResumeFile) this parameter shall not be included in the response message.
fileSizeUncompressedOrDirInfoLength Specifies the size of the uncompressed file to be uploaded or the length of the directory information to be read in bytes. If the modeOfOperation parameter equals to 01_{16} (AddFile), 02_{16} (DeleteFile), 03_{16} (ReplaceFile) or 06_{16} (ResumeFile) this parameter shall not be included in the response message.
fileSizeCompressed Specifies the size of the compressed file in bytes. If the modeOfOperation parameter equals to 01_{16} (AddFile), 02_{16} (DeleteFile, 03_{16} (ReplaceFile)), 05_{16} (ReadDir) or 06_{16} (ResumeFile) this parameter shall not be included in the response message.
filePosition Specifies the byte position within the file at which the Tester will resume downloading after an initial download is suspended. A download is suspended when the ECU stops receiving TransferData requests and does not receive the RequestTransferExit request before returning to the defaultSession. The filePosition is relative to the compressed size or uncompressed size, depending if the file was originally sent compressed or uncompressed during the initiating ModeOfOperation = AddFile or ReplaceFile. If the modeOfOperation parameter equals to 01_{16} (AddFile), 02_{16} (DeleteFile), 03_{16} (ReplaceFile), 04_{16} (ReadFile), or 05_{16} (ReadDir) this parameter shall not be included in the request.

15.6.4 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 484. The listed negative responses shall be used if the error scenario applies to the server.

Table 484 — Supported negative response codes

NRC	Description	Mnemonic
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 a server receives a request for this service while in the process of downloading or uploading data or other conditions to be able to execute this service are not met.	CNC

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

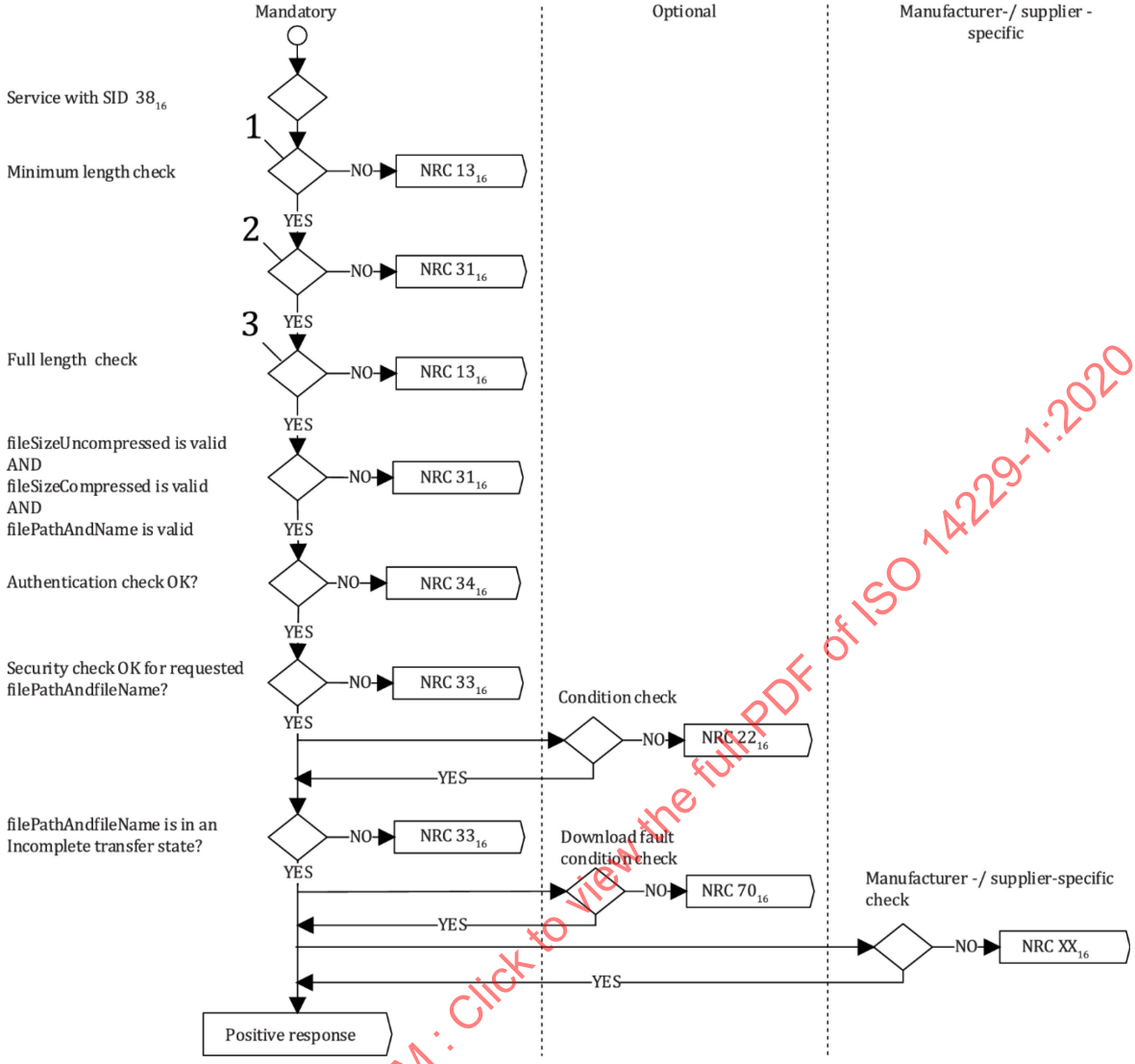
© ISO 2020 – All rights reserved

NRC	Description	Mnemonic
24_{16}	**requestSequenceError** This NRC shall be returned when modeOfOperation is 06_{16} (ResumeFile) and the requested file has already been completely transferred and therefore is not in a state to be resumed.	RSE
31_{16}	**requestOutOfRange** This NRC shall be returned if: — The specified dataFormatIdentifier is not valid. — The specified modeOfOperation is not valid. — The specified fileSizeParameterLength is not valid. — The specified filePathAndNameLength is not valid. — The specified fileSizeUncompressed is not valid. — The specified fileSizeCompressed is not valid. — The specified filePathAndName is not valid.	ROOR
33_{16}	**securityAccessDenied**	SAD
	This NRC shall be returned if the server is secured (for server's that support the SecurityAccess service) when a request for this service has been received.	
34_{16}	**authenticationRequired** This NRC shall be sent if the dataIdentifier is secured and the client has insufficient rights based on its Authentication state.	AR
70_{16}	**uploadDownloadNotAccepted** This NRC indicates that an attempt to download to a server's memory cannot be accomplished due to some fault conditions.	UDNA

The evaluation sequence is documented in Figure 35.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved



Key

1 minimum length: 5 byte (SI + MOOP + FPL_B1 + FPL_B2 + FP_B1)

2 the validity check of the message parameters depends on the modeOfOperation parameter

3 maximum length can be computed using fileSizeParamterLength and filePathAndNameLength

Figure 35 — NRC handling for requestFileTransfer service

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

15.6.5 Message flow example(s) RequestFileTransfer

15.6.5.1 Assumptions

This sub-clause specifies the conditions applicable for this message flow example.

NOTE This example is limited to the description of the requestFileTransfer request and the requestFileTransfer positive response. The usage of transferData and requestTransferExit in this context is identical with the usage of these services with requestDownload or requestUpload, thus the examples describing the download/upload sequence apply as well.

Table 485 specifies the message parameter values.

© ISO 2020 – All rights reserved

Table 485 — Definition RequestFileTransfer message parameter values

Data parameter name	Data parameter value(s)	Data parameter description
modeOfOperation	01_{16}	AddFile
filePathAndNameLength	$001E_{16}$	The length of parameter filePathAndName is 30.
filePathAndName	"D:\mapdata\europe\germany1.yxz"	Path including the file name.
dataFormatIdentifier	11_{16}	compressionMethod = $1X_{16}$; encryptingMethod = $X1_{16}$
fileSizeParameterLength	02_{16}	The length of both file size parameters is 2 bytes.
fileSizeUncompressed	$C350_{16}$	50 kbyte
fileSizeCompressed	7530_{16}	30 kbyte

15.6.5.2 Request file transfer

Table 486 and Table 487 show an example of the RequestFileTransfer request and response message flow.

Table 486 — RequestFileTransfer request message example

Message direction	**server → client**		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	RequestFileTransfer Request SID	38_{16}	RFT
#2	modeOfOperation	01_{16}	MOOP
#3 #4	filePathAndNameLength [byte#1 (MSB) byte#2] (LSB)	 00_{16} $1E_{16}$	 FPL_B1 FPL_B2

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Message direction	server → client		
Message type	Response		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
	filePathAndName = [		
#5	byte#1 (MSB)	44_{16}	FP_B1
#6	byte#2	$3A_{16}$	FP_B2
#7	byte#3	$5C_{16}$	FP_B3
#8	byte#4	$6D_{16}$	FP_B4
#9	byte#5	61_{16}	FP_B5
#10	byte#6	70_{16}	FP_B6
#11	byte#7	64_{16}	FP_B7
#12	byte#8	61_{16}	FP_B8
#13	byte#9	74_{16}	FP_B9
#14	byte#10	61_{16}	FP_B10
#15	byte#11	$5C_{16}$	FP_B11
#16	byte#12	65_{16}	FP_B12
#17	byte#13	75_{16}	FP_B13
#18	byte#14	72_{16}	FP_B14
#19	byte#15	$6F_{16}$	FP_B15
#20	byte#16	70_{16}	FP_B16
#21	byte#17	65_{16}	FP_B17
#22	byte#18	$5C_{16}$	FP_B18
#23	byte#19	67_{16}	FP_B19
#24	byte#20	65_{16}	FP_B20
#25	byte#21	72_{16}	FP_B21
#26	byte#22	$6D_{16}$	FP_B22
#27	byte#23	61_{16}	FP_B23
#28	byte#24	$6E_{16}$	FP_B24
#29	byte#25	79_{16}	FP_B25
#30	byte#26	31_{16}	FP_B26
#31	byte#27	$2E_{16}$	FP_B27
#32	byte#28	79_{16}	FP_B28
#33	byte#29	78_{16}	FP_B29
#34	byte#30]	$7A_{16}$	FP_B30
#35	dataFormatIdentifier	11_{16}	DFI_
#36	fileSizeParameterLength	02_{16}	FSL
	fileSizeUnCompressed= [		
#37	byte#1 (MSB)	$C3_{16}$	FSUC_B1
#38	byte#2]	50_{16}	FSUC_Bk
	fileSizeCompressed= [		
#39	byte#1 (MSB)	75_{16}	FSC_B1
#40	byte#2]	30_{16}	FSC_Bk

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 487 — RequestFileTransfer positive response request message example

Message direction	server → client		
Message type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value**	**Mnemonic**
#1	RequestFileTransfer Response SID	78_{16}	RRFT
#2	modeOfOperation	01_{16}	MOOP
#3	lengthFormatIdentifier	02_{16}	LFID
#4 #5	maxNumberOfBlockLength = [byte#1 (MSB) byte#m]	 $C3_{16}$ 50_{16}	MNROB_ B1 B2
#6	dataFormatIdentifier	11_{16}	DFI_

16 Security sub-layer definition

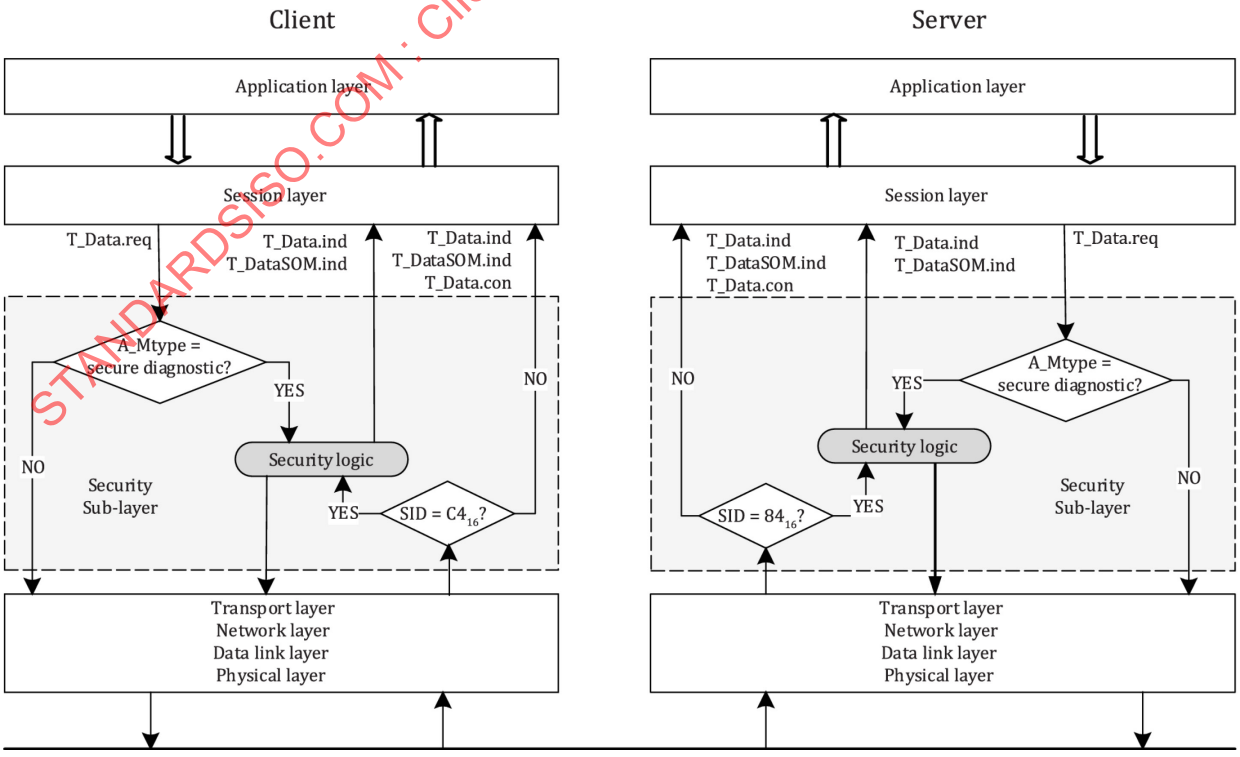
16.1 General

16.1.1 Purpose

The purpose of the security sub-layer is to transmit data that is protected against attacks from third parties - which could endanger data security.

16.1.2 Security sub-layer description

Figure 36 illustrates the security sub-layer. The security sub-layer shall be added in the server and client application for the purpose of performing diagnostic services in a secured mode.



STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Figure 36 — Security sub-layer implementation

There are two methods to perform diagnostic service data transfer between the client and server(s):

— Unsecured data transmission mode

The application uses the diagnostic services and application layer service primitives with A_Mtype set either to diagnostic or to remote diagnostics defined in this document to exchange data between a client and a server. The message of SecureDataTransmission is not allowed. All other messages bypass the security sub-layer and are processed as regular messages.

— Secured data transmission mode

The application uses the diagnostic services and application layer service primitives with A_Mtype set either to secure diagnostic or to secure remote diagnostics defined in this document to exchange data between a client and a server. The security sub-layer uses the SecuredDataTransmission service for the transmission/reception of the secured data. Depending on the key distribution secured links may be point-to-point communication only.

The task of the security sub-layer when performing a diagnostic service in a secured mode is to encrypt data provided by the "Application Layer", to decrypt data provided by the "Network Layer", to add an authentication code, to verify an authentication code, and to add, check, and remove security specific data elements. The security sub-layer uses the SecuredDataTransmission (84_{16}) request message of the application layer to transmit and receive the entire diagnostic message or message according to an external protocol (request and response), which shall be exchanged in a secured mode.

16.1.3 Security sub-layer access

The concept of accessing the security sub-layer for a secured service execution is similar to the application layer interface as described in this document. The security sub-layer makes use of the application layer service primitives.

The following describes the execution of confirmed diagnostic service in a secured mode:

— The client application uses the security sub-layer SecuredServiceExecution service request to perform a diagnostic service in a secured mode (A_Mtype = secureDiag). The security sub-layer performs the required action to establish a link with the server(s), adds the specific security related parameters, encrypts the service data of the diagnostic service to be executed in a secured mode if needed and uses the application layer SecuredDataTransmission service request message to transmit the secured data to the server.

— The server receives an application layer SecuredDataTransmission service indication, which is handled by the security sub-layer of the server. The security sub-layer of the server checks the security specific parameters decrypts encrypted data, confirms authenticity of the message and presents the data of the service to be executed in a secured mode to the application with message type set to secureDiag. The application service cannot be the SecuredDataTransmission (84_{16}) service request message.

— When the authentication of the SecuredDataTransmission service failed a regular negative response message is responded with using the SecuredDataTransmission (84_{16}) request message service identifier and a NRC.

— The application executes the service and uses the Mtype = secureDiag to respond to the service in a secured mode. The security sub-layer of the server adds the specific security related parameters, encrypts the response message data if needed and uses the application layer SecuredDataTransmission service response message to transmit the response data to the client.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

— The client receives an application layer SecuredDataTransmission service confirmation primitive, which is handled by the security sub-layer of the client. The security sub-layer of the client checks the security specific parameters, decrypts encrypted response data and presents the data via the security sub-layer SecuredServiceExecution confirmation to the application.

The ΔP2 timing highly varies with each ECU due to the used security algorithm and performance of the ECU. On timeout the message transmission had failed and the client application can assume that the request is not processed in the server.

Figure 37 and Figure 38 show the physical communication flow with normal and enhanced response timing.

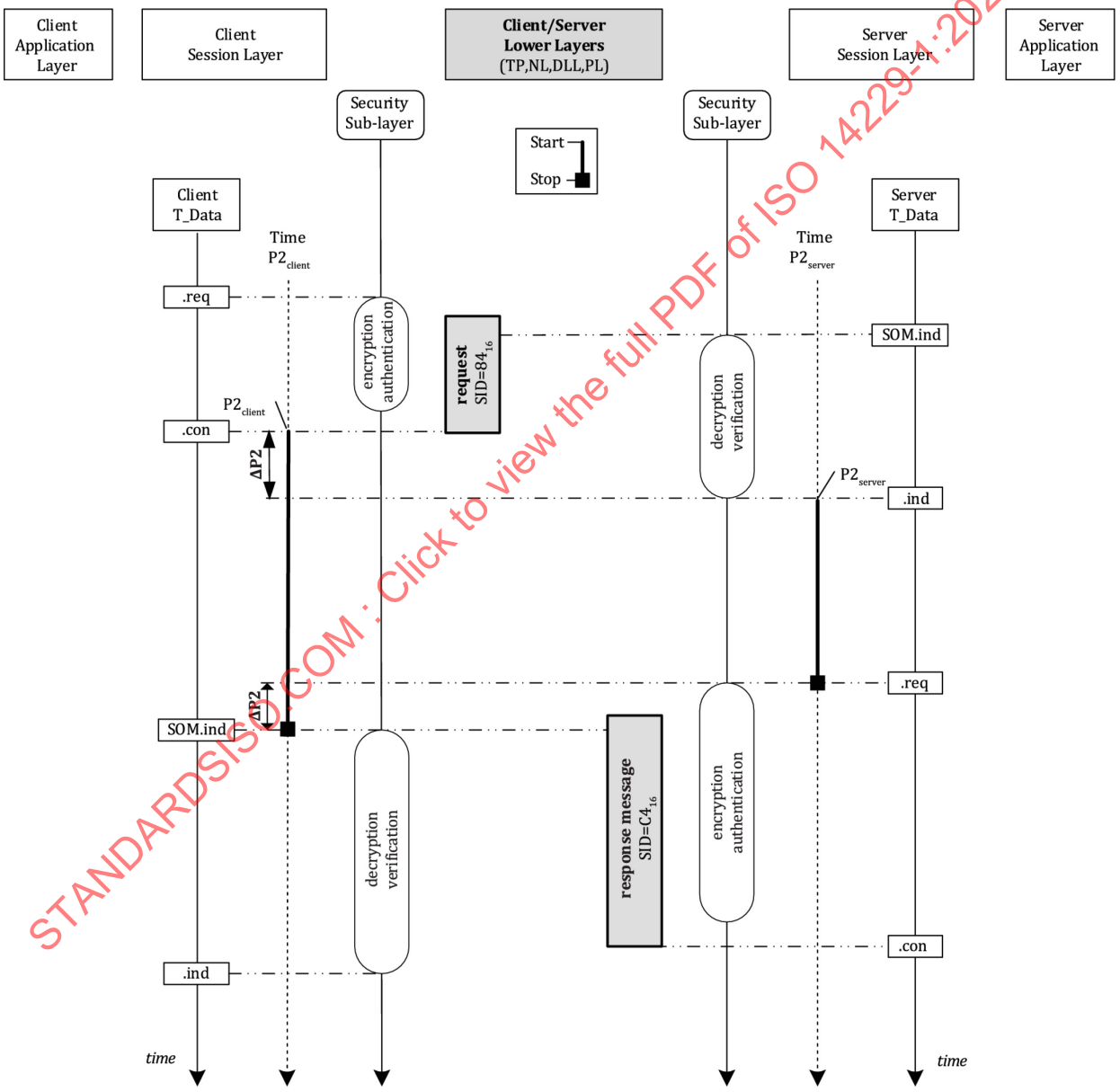


Figure 37 — Physical communication during defaultSession - with SOM.ind and security sub-layer

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

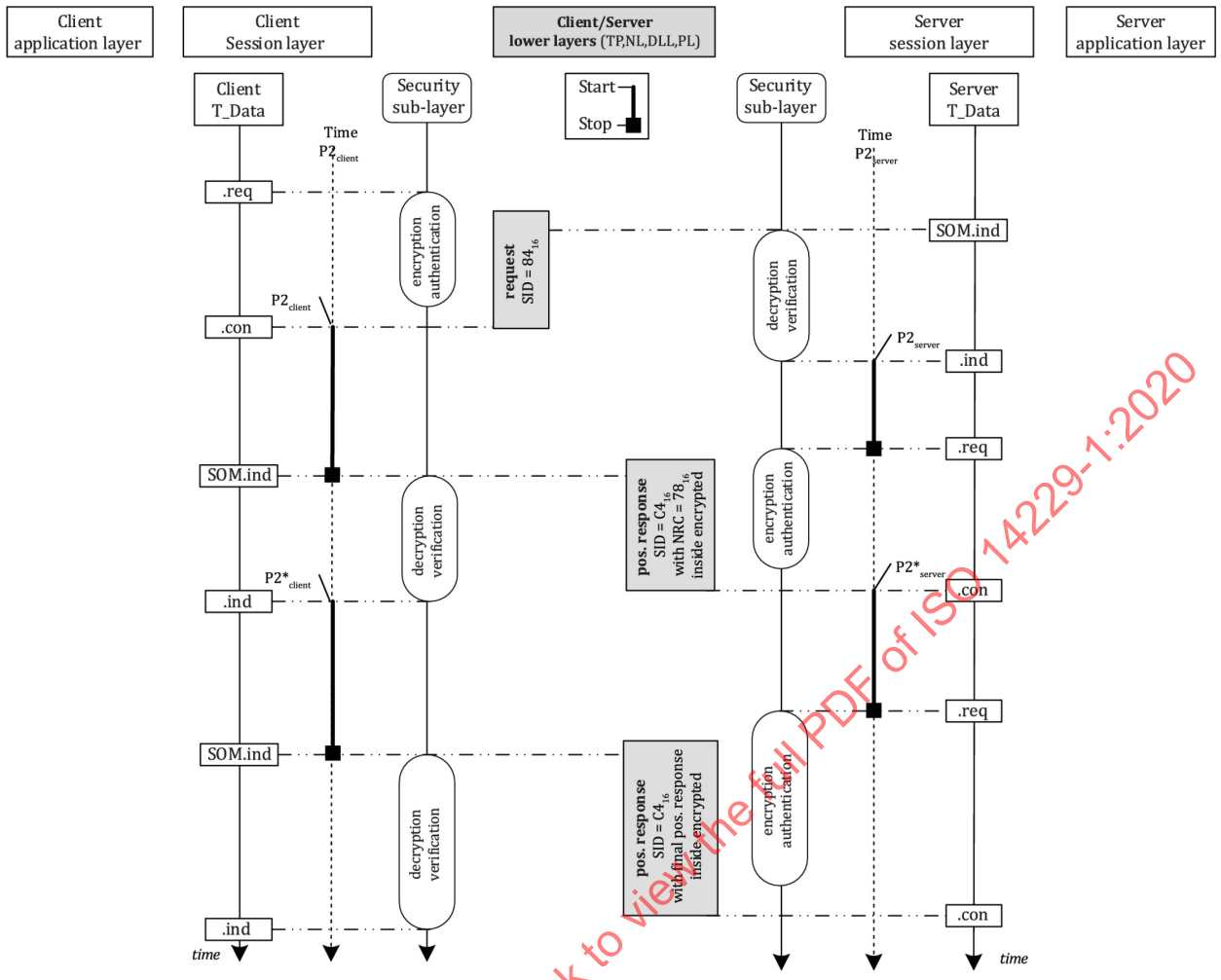


Figure 38 — Physical communication during defaultSession - with enhanced response timing and security sub-layer

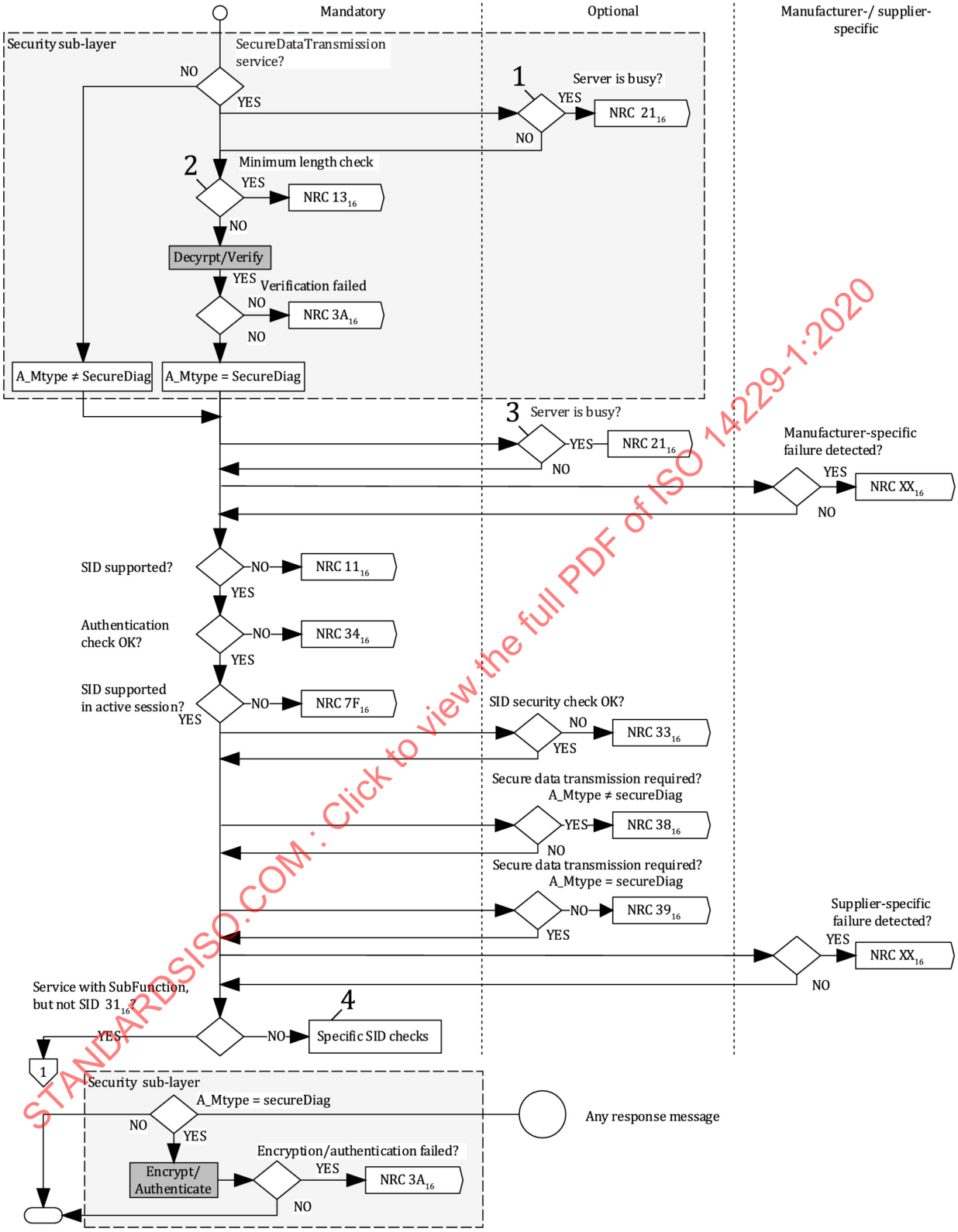
STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

IMPORTANT — The client shall process messages with A_Mtype = secureDiag in the security sub-layer. The server shall process the SecuredDataTransmission service in the security sub-layer. The SecuredDataTransmission service does not have any state or timing behaviour like the P2 timing. Therefore, for complex security layer calculations, the value of ΔP2 shall be considered. Additionally, the server and the client only meet the request and response message behaviour as specified in 8.7 for SID 84_{16}. The SecuredDataTransmission service is also not aware of any session or security level restrictions.

16.1.4 General server response behaviour

The general server response behaviour specified in Figure 39 is mandatory for all request messages in case the SecureDataTransmission service is supported. The validation steps start with the reception of the request message.

© ISO 2020 – All rights reserved



STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

Key

1 diagnostic request cannot be accepted because another security task is already requested and in progress

2 minimum length: (9 + signature length) byte

3 diagnostic request cannot be accepted because another diagnostic task is already requested and in progress by a different client

4 refer to the response behaviour (supported negative response codes) of each service

© ISO 2020 – All rights reserved

Figure 39 — Security sub-layer – Server response behaviour

16.2 SecuredDataTransmission (84_{16}) service

16.2.1 Service description

The SecuredDataTransmission service is applicable if a client intends to use diagnostic services defined in this document in a secured mode. It may also be used to transmit external data, which conform to some other application protocol, in a secured mode between a client and a server. A secured mode in this context means that the data transmitted is protected by cryptographic methods.

16.2.2 Request message

16.2.2.1 Request message definition

The security sub-layer generates the application layer SecuredDataTransmission request message parameters.

Table 488 specifies the request message using plain text (unencrypted).

NOTE: Encrypted messages would make the internal SID unreadable without decryption.

Table 488 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#1	SecuredDataTransmission Request SID	M	84_{16}	SDT
#2	Apar - Administrative Parameter = [	M		APAR
#3	byte#1 (MSB)		00_{16} to FF_{16}	
	byte#2]		00_{16} to FF_{16}	
#4	Signature/Encryption Calculation	M	00_{16} to FF_{16}	SIGENCRYPT
#5	Signature Length = [	M		SIGLEN
#6	byte#1 (MSB)		00_{16} to FF_{16}	
	byte#2]		00_{16} to FF_{16}	
#7	Anti-replay Counter = [			ANTIREPLAYCNT
#8	byte#1 (MSB)		00_{16} to FF_{16}	
	byte#2]		00_{16} to FF_{16}	
#9	Internal Message Service Request ID	M	00_{16} to FF_{16}	INTMSGREQID
.	Service Specific Parameters #1	U	00_{16} to FF_{16}	SRVSPECPARAM
.	:	.	:	
.	:	.	:	
#9+n	Service Specific Parameters #n	U	00_{16} to FF_{16}	
#9+n+1	Signature/MAC Byte #1	U	00_{16} to FF_{16}	SIGMACBYTE
.	:	.	:	
.	:	.	:	
#9+n+m	Signature/MAC Byte #m	U	00_{16} to FF_{16}	

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

16.2.2.2 Request message SubFunction parameter $Level (LEV_) definition

This service does not use a SubFunction parameter.

16.2.2.3 Request message data-parameter definition

Table 489 specifies the data-parameters of the request message.

Table 489 — Request message data-parameter definition

Definition
Administrative Parameter This parameter identifies the features of security used in the message.
Signature/Encryption Calculation This parameter contains an identifier for the algorithm used to sign or encrypt the message.
Signature Length This parameter identifies the length of the signature portion of the message.
Anti-replay Counter This parameter is an always incrementing counter to prevent a replay attack. It shall be reset to 0 after maximum limit value has been reached. The synchronisation and consistency mechanism between the client and the server(s) depend on the security algorithm and are manufacturer specific.
Internal Message Service Request ID This parameter is the internal Service Request ID for the message being encapsulated.
Service Specific Parameters This parameter contains any additional data required by the internal Service Request.
Signature/MAC Byte This parameter contains the signature used to verify the message. It shall be calculated using the Anti-replay Counter, the Internal Message Service and the Service Specific Parameters.

The Administrative Parameter is a bit string of 16 bits length. The bit assignment shall be according to Table 490. If a bit is set to 1 then the corresponding feature is requested. If it is set to 0 then the corresponding feature is not requested.

Table 490 — Definition of Administrative Parameter

Bit number	Meaning
0	Message is request message. (If not, it is a response message.)
1 to 2	ISO Reserved – Backwards Compatibility
3	A pre-established key is used. (If not, a key established in a secured link setup is used.)
4	Message is encrypted.
5	Message is signed.
6	Signature on the response is requested.
7 to 10	ISO reserved – Backwards compatibility
11 to 15	ISO reserved

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 491 specifies Definition of Signature/Encryption calculation parameter.

Table 491 — Definition of Signature/Encryption calculation parameter

Range	Description	Cvt	Mnemonic
00_{16} to $7F_{16}$	**VehicleManufacturerSpecific** This range of values shall be used to reference vehicle manufacturer specific signature/encryption calculation parameter.	U	VMS
80_{16} to $8F_{16}$	**SystemSupplier** This range of values shall be used to reference system supplier specific signature/encryption calculation parameter.	U	SSS
90_{16} to FF_{16}	**ISOSAEReserved** This range of values shall be reserved by this document for future definition.	U	ISOSAERESRVD

16.2.3 Positive response message for successful internal message

16.2.3.1 Positive response message definition for successful internal message

Table 492 specifies the positive response message where the internal service request is successful.

Table 492 — Positive response message definition

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
1	SecuredDataTransmission Response SID	M	$C4_{16}$	SDTPR
	Apar - Administrative Parameter = [	M		APAR
#2	byte#1 (MSB)		00_{16} to FF_{16}	
#3	byte#2]		00_{16} to FF_{16}	
#4	Signature/Encryption Calculation	M	00_{16} to FF_{16}	SIGENCRYPT
	Signature Length = [	M		SIGLEN
#5	byte#1 (MSB)		00_{16} to FF_{16}	
#6	byte#2]		00_{16} to FF_{16}	
	Anti-replay Counter = [	M		ANTIREPLAYCNT
#7	byte#1 (MSB)		00_{16} to FF_{16}	
#8	byte#2]		00_{16} to FF_{16}	
#9	Internal Message Service Response ID	M	00_{16} to FF_{16}	INTMSGRSPID
#9+1	Response Specific Parameters #1	U	00_{16} to FF_{16}	RSPSPECPARAM
:	:	:	:	
:	:	:	:	
#9+n	Response Specific Parameters #n	U	00_{16} to FF_{16}	

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
#9+n+1	Signature/MAC Byte #1	U	00_{16} to FF_{16}	SIGMACBYTE
:	:	.	:	
:	:	.	:	
#9+n+m	Signature/MAC Byte #m	U	00_{16} to FF_{16}	

16.2.3.2 Positive response message data-parameter definition

Table 493 specifies the data-parameter of the positive response message.

Table 493 — Response message data-parameter definition

Definition
Administrative Parameter This parameter identifies the features of security used in the message.
Signature/Encryption Calculation This parameter contains an identifier for the algorithm used to sign or encrypt the message.
Signature Length This parameter identifies the length of the signature portion of the message.
Anti-replay Counter This parameter is an always incrementing counter to prevent a replay attack. In this response message it is an echo of the Anti-replay Counter value used in the request message.
Internal Message Service Response ID This parameter is the internal Service Response ID for the message being encapsulated.
Response Specific Parameters This parameter contains any additional data required by the internal Service Request.
Signature/MAC Byte This parameter contains the signature used to verify the message. It shall be calculated using the Anti-replay Counter, the Internal Message Service and the Service Specific Parameters.

16.2.3.3 Positive response message definition for unsuccessful internal message

Table 494 specifies the positive response message where the internal service request is successful.

Table 494 — Positive response message definition

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
1	SecuredDataTransmission Response SID	M	$C4_{16}$	SDTPR
	Apar - Administrative Parameter = [	M		APAR
#2	byte#1 (MSB)		00_{16} to FF_{16}	
#3	byte#2]		00_{16} to FF_{16}	
#4	Signature/Encryption Calculation	M	00_{16} to FF_{16}	SIGENCRYPT

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

A_Data byte	Parameter Name	Cvt	Byte value	Mnemonic
	Signature Length = [	M		SIGLEN
#5	byte#1 (MSB)		00_{16} to FF_{16}	
#6	byte#2]		00_{16} to FF_{16}	
	Anti-replay Counter = [	M		ANTIREPLAYCNT
#7	byte#1 (MSB)		00_{16} to FF_{16}	
#8	byte#2]		00_{16} to FF_{16}	
#9	Internal Message Negative Response SID	M	$7F_{16}$	INTMSGNRSPID
#10	Internal Message Service Request ID	M	00_{16} to FF_{16}	INTMSGREQID
#11	Internal Message responseCode	M	00_{16} to FF_{16}	INTMSGRSPCODE
#12	Signature/MAC Byte #1	U	00_{16} to FF_{16}	SIGMACBYTE
:	:	:	:	
:	:	:	:	
#12+m	Signature/MAC Byte #m	U	00_{16} to FF_{16}	

16.2.3.4 Positive response message data-parameter definition

Table 495 specifies the data-parameter of the positive response message.

Table 495 — Response message data-parameter definition

Definition
Administrative Parameter This parameter identifies the features of security used in the message.
Signature/Encryption Calculation This parameter contains an identifier for the algorithm used to sign or encrypt the message.
Signature Length This parameter identifies the length of the signature portion of the message.
Anti-replay Counter This parameter is an always incrementing counter to prevent a replay attack. It shall be reset to 0 after maximum limit value has been reached. The synchronisation and consistency mechanism between the client and the server(s) depend on the security algorithm and are manufacturer-specific.
Signature/MAC Byte This parameter contains the signature used to verify the message. It shall be calculated using the Anti-replay Counter, the Internal Message Service and the Service Specific Parameters.

16.2.4 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 496. The listed negative responses shall be used if the error scenario applies to the server.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 496 — Supported negative response codes

NRC	Description	Mnemonic
13_{16}	**incorrectMessageLengthOrInvalidFormat** The server shall use this response code, if the length of the request A_PDU is not correct.	IMLOIF
$3A_{16}$	**secureDataVerificationFailed** This NRC indicates that the message failed in the security sub-layer.	SDVF

NOTE 1 The response codes listed above apply to the SecuredDataTransmission (84_{16}) service. In case the internal diagnostic service performed in a secured mode requires a negative response then this negative response is sent to the client in a secured mode via a SecuredDataTransmission positive response message.

NOTE 2 The negative response code secureDataTransmissionRequired (38_{16}) can be used for other diagnostic services if the requested action is required to be sent using a secured communication channel. The negative response code secureDataTransmissionNotAllowed (39_{16}) can be used for internal message service if the requested action is not allowed to be sent using a secured communication channel.

16.2.5 Message flow example SecuredDataTransmission

16.2.5.1 Assumptions

For the example messages, the message will send a signed, but unencrypted message. The signature calculation used was 1's complement.

16.2.5.2 Example #1: SecuredDataTransmission to Write DID

The security sub-layer generates the application layer SecuredDataTransmission request message parameters. This example shows the request and a positive response.

Table 497 specifies the request message.

Table 497 — SecuredDataTransmission request message flow example #1

A_Data byte	Parameter Name	Byte value	Mnemonic
#1	SecuredDataTransmission Request SID	84_{16}	SDT
#2 #3	Apar - Administrative Parameter = Message is a Request Message, Message is signed, Signature on the response is requested	00_{16} 61_{16}	APAR
#4	Signature/Encryption Calculation	00_{16}	SIGENCRYPT
#5 #6	Signature Length = [byte#1 (MSB) byte#2]	 00_{16} 06_{16}	SIGLEN
#7 #8	Anti-replay Counter = [byte#1 (MSB) byte#2]	 01_{16} 24_{16}	ANTIREPLAYCNT
#9	Internal Message Service Request ID	$2E_{16}$	INTMSGREQID

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

A_Data byte	Parameter Name	Byte value	Mnemonic
#10	Service Specific Parameters #1 – DID MSB	$F1_{16}$	SRVSPECPARAM
#11	Service Specific Parameters #2 – DID LSB	23_{16}	
#12	Service Specific Parameters #3 – Data	AA_{16}	
#13	Service Specific Parameters #4 – Data	55_{16}	
#14	Signature/MAC Byte #1	DB_{16}	SIGMACBYTE
#15	Signature/MAC Byte #2	$D1_{16}$	
#16	Signature/MAC Byte #3	$0E_{16}$	
#17	Signature/MAC Byte #4	DC_{16}	
#18	Signature/MAC Byte #5	55_{16}	
#19	Signature/MAC Byte #6	AA_{16}	

Table 498 specifies the SecuredDataTransmission positive response message flow example #1.

Table 498 — SecuredDataTransmission response message flow example #1

A_Data byte	Parameter Name	Byte value	Mnemonic
#1	SecuredDataTransmission Request SID	$C4_{16}$	SDTPR
#2	Apar - Administrative Parameter Message is signed	00_{16}	APAR
#3		20_{16}	
#4	Signature/Encryption Calculation	00_{16}	SIGENCRYPT
#5	Signature Length = [		SIGLEN
#6	byte#1 (MSB) byte#2]	00_{16} 06_{16}	
#7	Anti-replay Counter = [		ANTIREPLAYCNT
#8	byte#1 (MSB) byte#2]	01_{16} 24_{16}	
#9	Internal Message Service Response ID	$6E_{16}$	INTMSGRSPID
#10	Service Specific Parameters #1	$F1_{16}$	SRVSPECPARAM
#11	Service Specific Parameters #2	23_{16}	
#12	Signature/MAC Byte #1	FE_{16}	SIGMACBYTE
#13	Signature/MAC Byte #2	DB_{16}	
#14	Signature/MAC Byte #3	91_{16}	
#15	Signature/MAC Byte #4	$0E_{16}$	
#16	Signature/MAC Byte #5	DC_{16}	
#17	Signature/MAC Byte #6	FF_{16}	

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

16.2.5.3 Example #2: SecuredDataTransmission to Write DID

The security sub-layer generates the application layer SecuredDataTransmission request message parameters. This example shows the request and a negative response to the internal service request. The reject occurs because the data size for writing the VIN (DID $F190_{16}$) is too short (only two bytes length).

© ISO 2020 – All rights reserved

Table 499 specifies the request message.

Table 499 — SecuredDataTransmission request message flow example #2

A_Data byte	Parameter Name	Byte value	Mnemonic
#1	SecuredDataTransmission Request SID	84_{16}	SDT
#2 #3	Apar - Administrative Parameter = Message is Request Message, Message is signed, Signature on the response is requested	00_{16} 61_{16}	APAR
#4	Signature/Encryption Calculation	00_{16}	SIGENCRYPT
#5 #6	Signature Length = [byte#1 (MSB) byte#2]	 00_{16} 06_{16}	SIGLEN
#7 #8	Anti-replay Counter = [byte#1 (MSB) byte#2]	 01_{16} 36_{16}	ANTIREPLAYCNT
#9	Internal Message Service Request ID	$2E_{16}$	INTMSGREQID
#10	Service Specific Parameters #1 – DID MSB	$F1_{16}$	SRVSPECPARAM
#11	Service Specific Parameters #2 – DID LSB	90_{16}	
#12	Service Specific Parameters #3 – Data	57_{16}	
#13	Service Specific Parameters #4 – Data	30_{16}	
#14	Signature/MAC Byte #1	$C9_{16}$	SIGMACBYTE
#15	Signature/MAC Byte #2	$D1_{16}$	
#16	Signature/MAC Byte #3	$0E_{16}$	
#17	Signature/MAC Byte #4	$6F_{16}$	
#18	Signature/MAC Byte #5	$A8_{16}$	
#19	Signature/MAC Byte #6	CF_{16}	

Table 500 specifies the SecuredDataTransmission positive response message flow example #2.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

Table 500 — SecuredDataTransmission response message flow example #2

A_Data byte	Parameter Name	Byte value	Mnemonic
#1	SecuredDataTransmission Request SID	$C4_{16}$	SDTPR
#2	Apar - Administrative Parameter Message is signed	00_{16}	APAR
#3		20_{16}	
#4	Signature/Encryption Calculation	00_{16}	SIGENCRYPT
#5	Signature Length = [		SIGLEN
#6	byte#1 (MSB) byte#2]	00_{16} 06_{16}	
#7	Anti-replay Counter = [		ANTIREPLAYCNT
#8	byte#1 (MSB) byte#2]	01_{16} 36_{16}	
#9	Internal Message Negative Response ID	$7F_{16}$	INTMSGNRSPID
#10	Internal Message Negative Request ID	$2E_{16}$	
#11	Internal Message Negative Response Code	13_{16}	
#12	Signature/MAC Byte #1	FE_{16}	SIGMACBYTE
#13	Signature/MAC Byte #2	$C9_{16}$	
#14	Signature/MAC Byte #3	$A1_{16}$	
#15	Signature/MAC Byte #4	80_{16}	
#16	Signature/MAC Byte #5	EC_{16}	
#17	Signature/MAC Byte #6	FF_{16}	

17 Non-volatile server memory programming process

17.1 General information

This subclause defines a framework for the physically oriented download of one or multiple application software/data modules into non-volatile server memory. The defined non-volatile server memory programming sequence addresses:

a) vehicle manufacturer specific needs in performing certain steps during the programming process, while being compliant with the general service execution requirements as specified in this document and ISO 14229-2 (such as the sequential order of services and the session management),

b) to support networks with multiple nodes connected, which interact with each other, using normal communication messages,

c) use of either a physically oriented vehicle approach (point-to-point communication — servers do not support functional diagnostic communication) or a functionally oriented vehicle approach (point-to-point and point-to-multiple communication — servers support functional diagnostic communication). A single vehicle shall only support one of the above mentioned vehicle approaches.

The programming sequence is divided into two programming phases. All steps are categorized based on the following types:

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

— Standardized steps: this type of step is mandatory. The client and the server shall behave as specified.

— Optional/recommended steps: this type of step is optional. These optional steps require the usage of a specific diagnostic service identifier (as described in the step) and contain recommendations on how an operation shall be performed. Where the specified functionality is used, then the client and the server shall behave as specified.

— Vehicle manufacturer specific steps: this type of step is optional. The usage and content (e.g. diagnostic service identifiers used) of these optional steps is left to the discretion of the vehicle manufacturer and shall be in accordance with this document and ISO 14229-2.

The defined steps can either be:

— functionally addressed to all nodes on the network (functionally oriented vehicle approach, servers support functional diagnostic communication), or

— physically addressed to each node on the network (physically oriented vehicle approach).

Each step of the two programming phases of the programming procedure will specify the allowed addressing method for that step. The vehicle manufacturer specific steps can either by functionally or physically addressed (depends on the OEM requirements).

Figure 40 depicts the non-volatile server memory programming process overview.

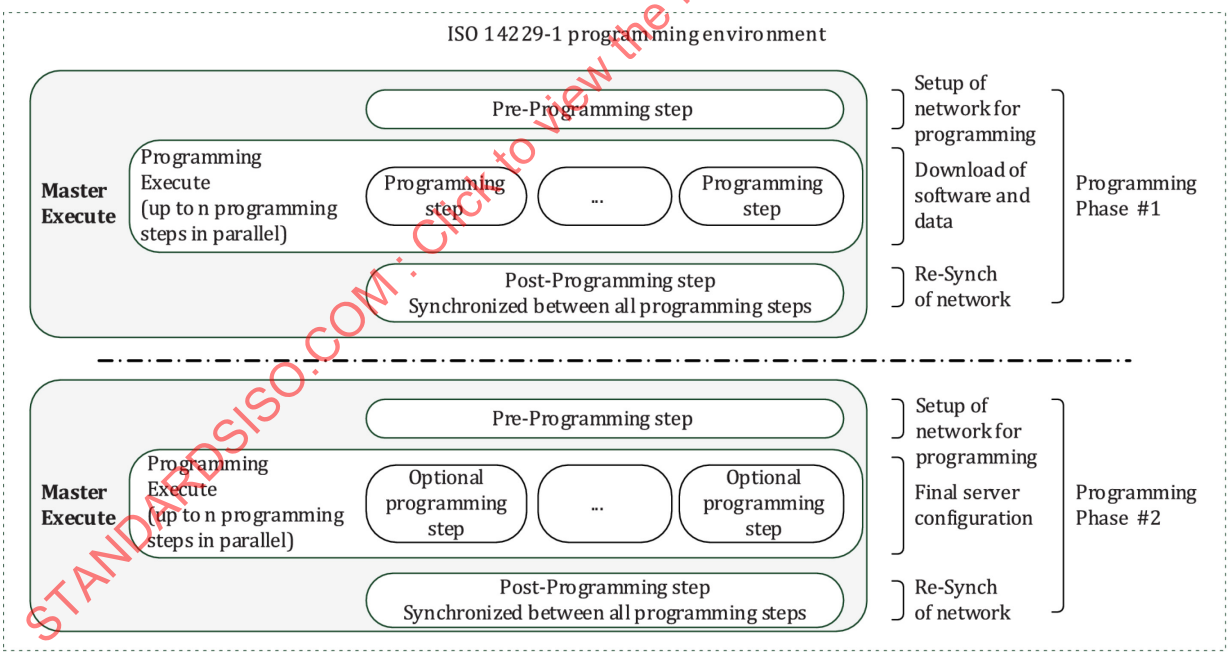


Figure 40 — Non-volatile server memory programming process overview

The programming process the client is required to follow consists of two distinct types of diagnostic service executions:

— Master execute

All steps that are required to be synchronized between multiple programming steps which run in parallel shall be coordinated as they are intended for vehicle wide functions (e.g. typically using functional addressing). This is achieved via the "master execute" of the client. The steps defined for

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

the “Pre-Programming Step” and the "Post-programming Step" of the individual programming phases are executed by the "master execute" of the client. The programming process requires synchronisation between the individual "Programming steps" (e.g. the transition of the vehicle network into a mode of operation that allows for programming of individual ECUs, or at the point in time when the individual parallel "Programming steps" reach the point where a conclusion of a programming phase is required). The master execute shall maintain the vehicle in the mode of operation it has transitioned to.

— Programming execute

All steps that are not required to be synchronized between multiple "Programming steps" do not require any coordination by the client and can run in parallel, therefore no “master execute” is required in the client during the execution of these steps. The "Programming steps" of the individual ECUs can be executed individually in parallel by the client until they are concluded and require the execution of the "Post-programming phase". All steps controlled by the "programming execute" are ECU oriented steps (e.g. physically addressed to the ECU to be programmed).

d) Programming phase #1 — download of application software and/or application data

1) Within programming phase #1, the application software/data is transferred to the server.

i) Optional Pre-Programming step — Setup of vehicle network for programming

The preprogramming step of phase #1 is optional and used to prepare the vehicle network for a programming event of one or multiple servers. This step provides certain hooks where a vehicle manufacturer can insert specific operations that are required for the OEM vehicle’s network (perform wake-up, determine communication parameters, read server identification data, etc.).

This step also contains provisions to increase the bit rate to improve download performance. The usage of this functionality is optional and can only be performed in case of a functionally oriented vehicle approach (functional diagnostic communication supported by the servers).

The request messages of this step can either be physically or functionally addressed.

2) Server Programming step — Download of application software and application data

The server programming step of phase #1 is used to program one or multiple servers (download of application software and/or application data and/or boot software).

Within this step, only physical addressing is used by the client, which allows for parallel or sequential programming of multiple nodes. In the case where the preprogramming step is not used, then the DiagnosticSessionControl (10_{16}) withSubFunction programmingSession can also be performed using functional addressing.

At the end of this step, a physical reset of the reprogrammed server(s) is optional. The use of the reset leads to the requirement to implement programming phase #2 in order to finally conclude the programming event by physically clearing DTCs in the reprogrammed server(s), because after the physical reset during this step the reprogrammed server(s) enable(s) the default session and perform(s) their normal mode of operation while the remaining server(s) have still disabled normal communication. The reprogrammed server(s) will potentially set DTCs.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved

Furthermore, it shall be considered that the reprogrammed server could activate a new set of diagnostic address, which differs from the ones used when performing a programming event (see 17.3).

If either the server that was reprogrammed does not change its communication parameters or the client knows the changed communication parameters, then following the reset certain configuration data can be written to the reprogrammed server.

3) Post-Programming step — Re-synchronisation of vehicle network after programming

The post-programming step of phase #1 concludes the programming phase #1. This step is performed when the programming step of each reprogrammed server is finished.

The request messages of this step can either be physically or functionally addressed.

The vehicle network is transitioned to its normal mode of operation. This can either be done via a reset using the ECUReset (11_{16}) service or an explicit transition to the default session via the DiagnosticSessionControl (10_{16}) service.

e) Programming phase #2 — Server configuration (optional)

1) Programming phase #2 is an optional phase in which the client can perform further actions that are needed to finally conclude a programming event (write the VIN, trigger Immobilizer learn-routine, etc.). For example, if the server(s) that has (have) been reprogrammed is (are) physically reset during the server programming step of programming phase #1, then DTCs shall be cleared in this server(s).

2) When executing this phase, the downloaded application software/application data is running/activated in the server and the server provides its full diagnostic functionality.

— Pre-Programming step — Setup of vehicle network for server configuration

The preprogramming step of phase #2 is used to prepare the vehicle network for the programming step of phase #2. This step is an optional step and provides certain hooks where a vehicle manufacturer can insert specific operations that are required for OEM vehicle's network (e.g. wake-up, determine communication parameters).

The request messages of these steps can either be physically or functionally addressed.

— Programming step — Final server configuration

The programming step is used to, for example, write data (e.g. VIN), after the server reset.

The content of this step is vehicle manufacturer specific.

If the server(s) that has (have) been reprogrammed is physically reset at the end of the server programming step of programming phase #1, then DTCs shall be cleared in this server(s) during the programming step of phase #2.

The request messages of these steps are physically addressed.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

© ISO 2020 – All rights reserved

— Post-Programming step — Re-synchronisation of vehicle network after final server configuration

The post-programming step concludes programming phase #2. This step is performed when the programming step of each reprogrammed server is finished. The vehicle network is transitioned to its normal mode of operation.

This step can either be functionally oriented (servers support functional diagnostic communication) or physically oriented.

The request messages of these steps can either be physically or functionally addressed.

17.2 Detailed programming sequence

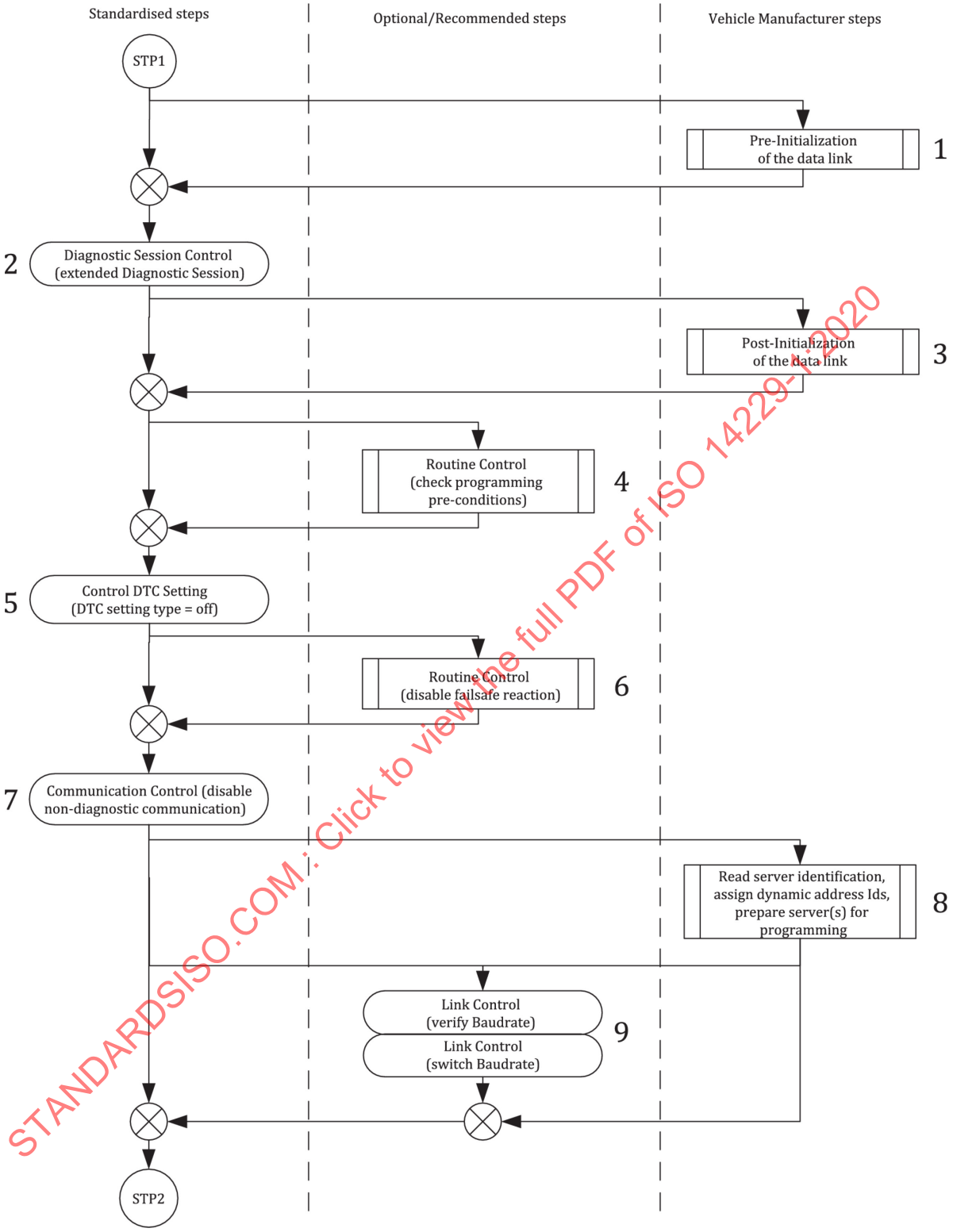
17.2.1 Programming phase #1 — Download of application software and/or application data

17.2.1.1 Pre-Programming step of phase #1 — Setup of vehicle network for programming

Figure 41 graphically depicts the functionality embedded in the preprogramming step.

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved



STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

Key

1 prior to any communication on the data link the network shall be initialized (e.g. perform an initial wake-up on the network). The wake-up method and strategy is vehicle manufacturer specific and optional to be used. Furthermore, this step allows for a determination of the server communication parameters such as the network configuration parameter and server diagnostic address used by the server(s)

2 in order to be able to disable the normal communication between the servers and the setting of DTCs, it is required to start a non-defaultSession in each server where normal communication and DTCs shall be disabled. This is achieved via a DiagnosticSessionControl (10_{16}) service with sessionType equal to extendedDiagnosticSession. The request is either transmitted functionally addressed to all servers with a single request message, or physically

© ISO 2020 – All rights reserved

addressed to each server in a separate request message [requires a physically addressed TesterPresent ($3E_{16}$) request message to be transmitted to each server that is transitioned into a non-defaultSession]. It is vehicle manufacturer specific whether response messages are required or not

3 following the transition into the extendedDiagnosticSession, further vehicle manufacturer specific data link initialization steps can optionally be performed

EXAMPLE A vehicle manufacturer specific additional initialization step can be to issue a request that causes gateway devices to perform a wake-up on all data links which are not accessible by the client directly through the diagnostic connector. The gateway will keep the data link(s) awake as long as the non-defaultSession is kept active in the gateway.

4 this optional routineIdentifier (number chosen by the vehicle manufacturer) allows a client to check whether all pre-conditions to transition to the programmingSession are fulfilled prior to attempting the transition

5 the client disables the setting of DTCs in each server using the ControlDTCSetting (85_{16}) service with DTCSettingType equal to "off". The request is either transmitted functionally addressed to all servers with a single request message, or transmitted physically addressed to each server in a separate request message. It is vehicle manufacturer specific whether response messages are required or not

6 This optional routineIdentifier (number chosen by the vehicle manufacturer) allows a client to enable or disable the failsafe reaction of an ECU if needed for safety reasons.

7 the client disables the transmission and reception of non-diagnostic messages using the CommunicationControl (28_{16}) service. The controlType parameter and communicationType parameter values are vehicle manufacturer specific (one OEM might disable the transmission only while another OEM might disable the transmission and the reception based on vehicle manufacturer specific needs). The request is either transmitted functionally addressed to all servers with a single request message, or transmitted physically addressed to each server in a separate request message. It is vehicle manufacturer specific whether response messages are required or not

8 after disabling normal communication an optional vehicle manufacturer specific step follows, which allows the following:

Reading the status of the server(s) to be programmed (e.g. application software/data programmed).

Reading server identification data from the server(s) to be programmed:

—identification (see dataIdentifier definitions): applicationSoftwareIdentification, applicationDataIdentification,

—fingerprint (see dataIdentifier definitions): applicationSoftwareFingerprint, applicationDataFingerprint.

Communication configuration such as dynamic assignment of address identifiers for a "Service ECU".

Preparation of non-programmable servers for the upcoming programming event in order to allow them to optimize their data link hardware acceptance filtering in a way that they can handle a 100 % bus utilization without dropping data link frames (only accept the function request address identifier and its own physical request address identifier)

9 it is optional to increase the bandwidth for the programming event in order to decrease the overall programming time and to gain additional bandwidth to be able to program multiple servers in parallel. A LinkControl (87_{16}) service with linkControl equal to either verifyBaudrateTransitionWithFixedMode or verifyBaudrateTransitionWithSpecificMode is transmitted functionally or physically addressed to all servers with a single request message with responseRequired equal to "yes". This service is used to verify if a mode transition at the associated data link can be performed. At this point the transition is not performed. A second LinkControl (87_{16}) service withSubFunction transitionMode is transmitted functionally addressed to all servers with a single request message with responseRequired equal to "no".

Once the request message is successfully transmitted, the client and all servers transition to the previously verified mode for the programming event. The servers shall transition the individual data link specific mode within a vehicle manufacturer specific timing window. For this duration plus a safety margin, the client is not allowed to transmit any request message onto the vehicle network (including the TesterPresent request message). When the transition is successfully performed, then the requested mode shall stay active for the duration the server switches between non-defaultSessions. Once the server transitions to the defaultSession, it shall re-enable the normal mode of the vehicle link it is connected to.

The usage of mode switches requires the support of functional diagnostic communication in each server on a single data link that shall be transitioned to the associated data link dependent mode

Figure 41 — Preprogramming step of phase 1 (STP1)

STANDARDSISO.COM : Click to view the full PDF of ISO 14229-1:2020

 © ISO 2020 – All rights reserved