INTERNATIONAL STANDARD

ISO 14229-1

Second edition
2013-03-15

Road vehicles — Unified diagnostic services (UDS) —

Part 1: Specification and requirements

Véhicules routiers — Services de diagnostic unifiés (SDU) —

Partie 1: Spécification et exigences

Reference number
ISO 14229-1:2013(E)

© ISO 2013

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

COPYRIGHT PROTECTED DOCUMENT

© ISO 2013

All rights reserved. Unless otherwise specified, 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
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

 © ISO 2013 – All rights reserved

Contents

Page

Foreword vi

Introduction vii

1 Scope 1

2 Normative references 1

3 Terms, definitions, symbols and abbreviated terms 1
3.1 Terms and definitions 1
3.2 Abbreviated terms 4

4 Conventions 5

5 Document overview 6

6 Application layer services 7
6.1 General 7
6.2 Format description of application layer services 9
6.3 Format description of service primitives 9
6.4 Service data unit specification 12

7 Application layer protocol 15
7.1 General definition 15
7.2 Protocol data unit specification 16
7.3 Application protocol control information 16
7.4 Negative response/confirmation service primitive 18
7.5 Server response implementation rules 18

8 Service description conventions 29
8.1 Service description 29
8.2 Request message 30
8.3 Positive response message 33
8.4 Supported negative response codes (NRC_) 34
8.5 Message flow examples 34

9 Diagnostic and Communication Management functional unit 35
9.1 Overview 35
9.2 DiagnosticSessionControl (0x10) service 36
9.3 ECUReset (0x11) service 43
9.4 SecurityAccess (0x27) service 47
9.5 CommunicationControl (0x28) service 53
9.6 TesterPresent (0x3E) service 58
9.7 AccessTimingParameter (0x83) service 61
9.8 SecuredDataTransmission (0x84) service 66
9.9 ControlDTCSetting (0x85) service 71
9.10 ResponseOnEvent (0x86) service 75
9.11 LinkControl (0x87) service 99

10 Data Transmission functional unit 106
10.1 Overview 106
10.2 ReadDataByIdentifier (0x22) service 106
10.3 ReadMemoryByAddress (0x23) service 113
10.4 ReadScalingDataByIdentifier (0x24) service 119
10.5 ReadDataByPeriodicIdentifier (0x2A) service 126
10.6 DynamicallyDefineDataIdentifier (0x2C) service 140
10.7 WriteDataByIdentifier (0x2E) service 162
10.8 WriteMemoryByAddress (0x3D) service 167

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

11 Stored Data Transmission functional unit **174**
11.1 Overview **174**
11.2 ClearDiagnosticInformation (0x14) Service **175**
11.3 ReadDTCInformation (0x19) Service **178**

12 InputOutput Control functional unit **245**
12.1 Overview **245**
12.2 InputOutputControlByIdentifier (0x2F) service **245**

13 Routine functional unit **259**
13.1 Overview **259**
13.2 RoutineControl (0x31) service **260**

14 Upload Download functional unit **270**
14.1 Overview **270**
14.2 RequestDownload (0x34) service **270**
14.3 RequestUpload (0x35) service **275**
14.4 TransferData (0x36) service **280**
14.5 RequestTransferExit (0x37) service **285**
14.6 RequestFileTransfer (0x38) service **295**

15 Non-volatile server memory programming process **303**
15.1 General information **303**
15.2 Detailed programming sequence **307**
15.3 Server reprogramming requirements **315**
15.4 Non-volatile server memory programming message flow examples **319**

Annex A (normative) **Global parameter definitions** **325**
A.1 Negative response codes **325**

Annex B (normative) **Diagnostic and communication management functional unit data-parameter definitions** **333**
B.1 communicationType parameter definition **333**
B.2 eventWindowTime parameter definition **334**
B.3 linkControlModeIdentifier parameter definition **334**
B.4 nodeIdentificationNumber parameter definition **335**

Annex C (normative) **Data transmission functional unit data-parameter definitions** **337**
C.1 DID parameter definitions **337**
C.2 scalingByte parameter definitions **343**
C.3 scalingByteExtension parameter definitions **345**
C.4 transmissionMode parameter definitions **351**
C.5 Coding of UDS version number **352**

Annex D (normative) **Stored data transmission functional unit data-parameter definitions** **353**
D.1 groupOfDTC parameter definition **353**
D.2 DTCStatusMask and statusOfDTC bit definitions **353**
D.3 DTC severity and class definition **366**
D.4 DTCFormatIdentifier definition **369**
D.5 FunctionalGroupIdentifier definition **369**
D.6 DTCFaultDetectionCounter operation implementation example **371**
D.7 DTCAgingCounter example **372**

Annex E (normative) **Input output control functional unit data-parameter definitions** **374**
E.1 InputOutputControlParameter definitions **374**

Annex F (normative) **Routine functional unit data-parameter definitions** **375**
F.1 RoutineIdentifier (RID) definition **375**

Annex G (normative) **Upload and download functional unit data-parameter** **376**
G.1 Definition of modeOfOperation values **376**

Annex H (informative) **Examples for addressAndLengthFormatIdentifier parameter values** **377**
H.1 addressAndLengthFormatIdentifier example values **377**

Annex I (normative) **Security access state chart** **379**

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

 © ISO 2013 – All rights reserved

I.1 General **379**
I.2 Disjunctive normal form based state transition definitions **379**

Annex J (informative) **Recommended implementation for multiple client environments** **385**
J.1 Introduction **385**
J.2 Implementation specific limitations **385**
J.3 Use cases relevant for system design **386**
J.4 Use Case Evaluation: **388**
J.5 Multiple client server level implementation **389**

Bibliography **391**

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – 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.

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

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

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

ISO 14229-1 was prepared by Technical Committee ISO/TC 22, *Road vehicles*, Subcommittee SC 3, *Electrical and electronic equipment*.

This second edition cancels and replaces the first edition (ISO 14229-1:2006), which has been technically revised.

ISO 14229 consists of the following parts, under the general title *Road vehicles — Unified diagnostic services (UDS)*:

— *Part 1: Specification and requirements*

— *Part 2: Session layer services*

— *Part 3: Unified diagnostic services on CAN implementation (UDSonCAN)*

— *Part 4: Unified diagnostic services on FlexRay implementation (UDSonFR)*

— *Part 5: Unified diagnostic services on Internet Protocol implementation (UDSonIP)*

— *Part 6: Unified diagnostic services on K-Line implementation (UDSonK-Line)*

The following part is under preparation:

— *Part 7: Unified diagnostic services on Local Interconnect Network implementation (UDSonLIN)*

The titles of future parts will be drafted as follows:

— *Part n: Unified diagnostic services on … implementation (UDSon…)*

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – 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 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 ISO 14229-1, ISO 14229-3 UDSonCAN, ISO 14229-4 UDSonFR, ISO 14229-5 UDSonIP, ISO 14229-6 UDSonK-Line, ISO 14229-7 UDSonLIN, further standards and ISO 27145-3 WWH-OBD.

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

— 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 27145-4 WWH-OBD.

— 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 27145-4 WWH-OBD.

— 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 and further standards, ISO 27145-4 WWH-OBD.

— 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 and further standards, ISO 27145-4 WWH-OBD.

NOTE The diagnostic services in this standard 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. It is required that future modifications to this standard provide long-term backward compatibility with the implementation standards as described above.

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

<table>
<tr><th>Applicability</th><th>OSI seven layer</th><th colspan="7">Enhanced diagnostics services</th><th>WWH-OBD</th></tr>
<tr><td rowspan="8">Seven layer according to ISO/IEC 7498-1 and ISO/IEC 10731</td><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, 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>further standards</td><td rowspan="6">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>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>
</table>

© ISO 2013 – All rights reserved

SHW3SZH 2013-09-29 NormMaster
Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18
SHW3SZH 2013-09-29 NormMaster

Road vehicles — Unified diagnostic services (UDS) —

Part 1: Specifications and requirements

1 Scope

This part of ISO 14229 specifies data link independent requirements of diagnostic services, which allow a diagnostic tester (client) to control diagnostic functions in an on-vehicle Electronic Control Unit (ECU, server) such as an electronic fuel injection, automatic gear box, 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 part of ISO 14229 does not apply to non-diagnostic message transmission on the vehicle's communication data link between two Electronic Control Units. However, this part of ISO 14229 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 part of ISO 14229 does not specify any implementation requirements.

2 Normative references

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

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

3 Terms, definitions, symbols and abbreviated terms

3.1 Terms and definitions

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

3.1.1
boot manager
part of the boot software that executes immediately after an ECU power on or reset whose primary purpose is to check whether a valid application is available to execute as compared to transferring control to the reprogramming software

NOTE The boot manager may also take into account other conditions for transitioning control to the reprogramming software.

3.1.2
boot memory partition
area of the server memory in which the boot software is located

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

3.1.3
boot software
software which is executed in a special part of server memory which is used primarily to boot the ECU and perform server programming

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

NOTE 2 See also 3.1.1 and 3.1.17.

3.1.4
client
function that is part of the tester and that makes use of the diagnostic services

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

3.1.5
diagnostic data
data that is located in the memory of an electronic control unit which may be inspected and/or possibly modified by the tester

NOTE 1 Diagnostic data includes analogue inputs and outputs, digital inputs and outputs, intermediate values and various status information.

NOTE 2 Examples of diagnostic data are vehicle speed, throttle angle, mirror position, system status, etc. Three types of values are defined for diagnostic data:

— the current value: the value currently used by (or resulting from) the normal operation of the electronic control unit;

— a stored value: an internal copy of the current value made at specific moments (e.g. when a malfunction occurs or periodically); this copy is made under the control of the electronic control unit;

— a static value: e.g. VIN.

The server 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 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).

3.1.6
diagnostic routine
routine that is embedded in an electronic control unit and that may be started by a server upon a request from the client

NOTE It could either run instead of a normal operating program, or could be enabled in this mode and executed with the normal operating program. In the first case, normal operation for the server is not possible. In the second case, multiple diagnostic routines may be enabled that run while all other parts of the electronic control unit are functioning normally.

3.1.7
diagnostic service
information exchange initiated by a client in order to require diagnostic information from a server or/and to modify its behaviour for diagnostic purpose

3.1.8
diagnostic session
state within the server in which a specific set of diagnostic services and functionality is enabled

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

 © ISO 2013 – All rights reserved

3.1.9
diagnostic trouble code
DTC
numerical common identifier for a fault condition identified by the on-board diagnostic system

3.1.10
ECU
electronic control unit, containing at least one server

NOTE Systems considered as Electronic Control Units include Anti-lock Braking System (ABS) and Engine Management System.

3.1.11
functional unit
set of functionally close or complementary diagnostic services

3.1.12
integer type
simple type with distinguished values which are the positive and the negative whole numbers, including zero

NOTE The range of type integer is not specified within this part of ISO 14229.

3.1.13
local client
client that is connected to the same local network as the server and is part of the same address space as the server

3.1.14
local server
server that is connected to the same local network as the client and is part of the same address space as the client

3.1.15
OSI
open systems interconnection

3.1.16
permanent DTC
diagnostic trouble code (DTC) 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 Refer to the relevant legislation for all necessary requirements.

3.1.17
record
one or more diagnostic data elements that are referred to together by a single means of identification

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

3.1.18
remote server
server that is not directly connected to the main diagnostic network

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

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

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

3.1.19
remote client
client that is not directly connected to the main diagnostic network

NOTE 1 A remote client is identified by means of a remote address.

NOTE 2 Remote addresses represent an own address space that is independent from the addresses on the main network.

3.1.20
reprogramming software
part of the boot software that allows for reprogramming of the electronic control unit

3.1.21
security
mechanism for protecting vehicle modules from "unauthorized" intrusion through a vehicle diagnostic data link

3.1.22
server
function that is part of an electronic control unit and that provides the diagnostic services

NOTE This international standard differentiates between the server (i.e. the function) and the electronic control unit so that this standard remains independent from the implementation.

3.1.23
supported DTC
diagnostic trouble code which is currently configured/calibrated and enabled to execute under pre-defined vehicle conditions

3.1.24
tester
system that controls functions such as test, inspection, monitoring, or diagnosis of an on-vehicle electronic control unit and may 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 The tester is also referenced as the client.

3.2 Abbreviated terms

.con	service primitive .confirmation
.ind	service primitive .indication
.req	service primitive .request
A_PCI	application layer protocol control information
ECU	electronic control unit
EDR	event data recorder
N/A	not applicable
NR_SI	negative response service identifier
NRC	negative response code
OSI	open systems interconnection

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

RA remote address

SA source address

SI service identifier

TA target address

TA_type target address type

4 Conventions

This part of ISO 14229 is based on the conventions discussed in the OSI Service Conventions (ISO/IEC 10731:1994) as they apply for diagnostic services.

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

The distinction between service and protocol is summarised in Figure 1.

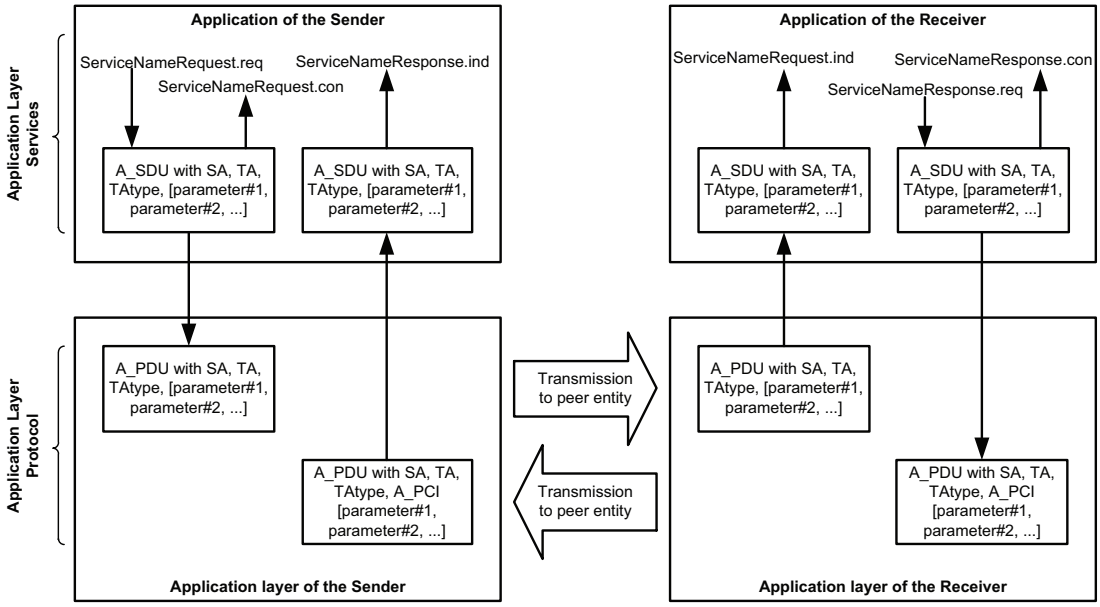


Figure 1 — The services and the protocol

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

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

5 Document overview

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

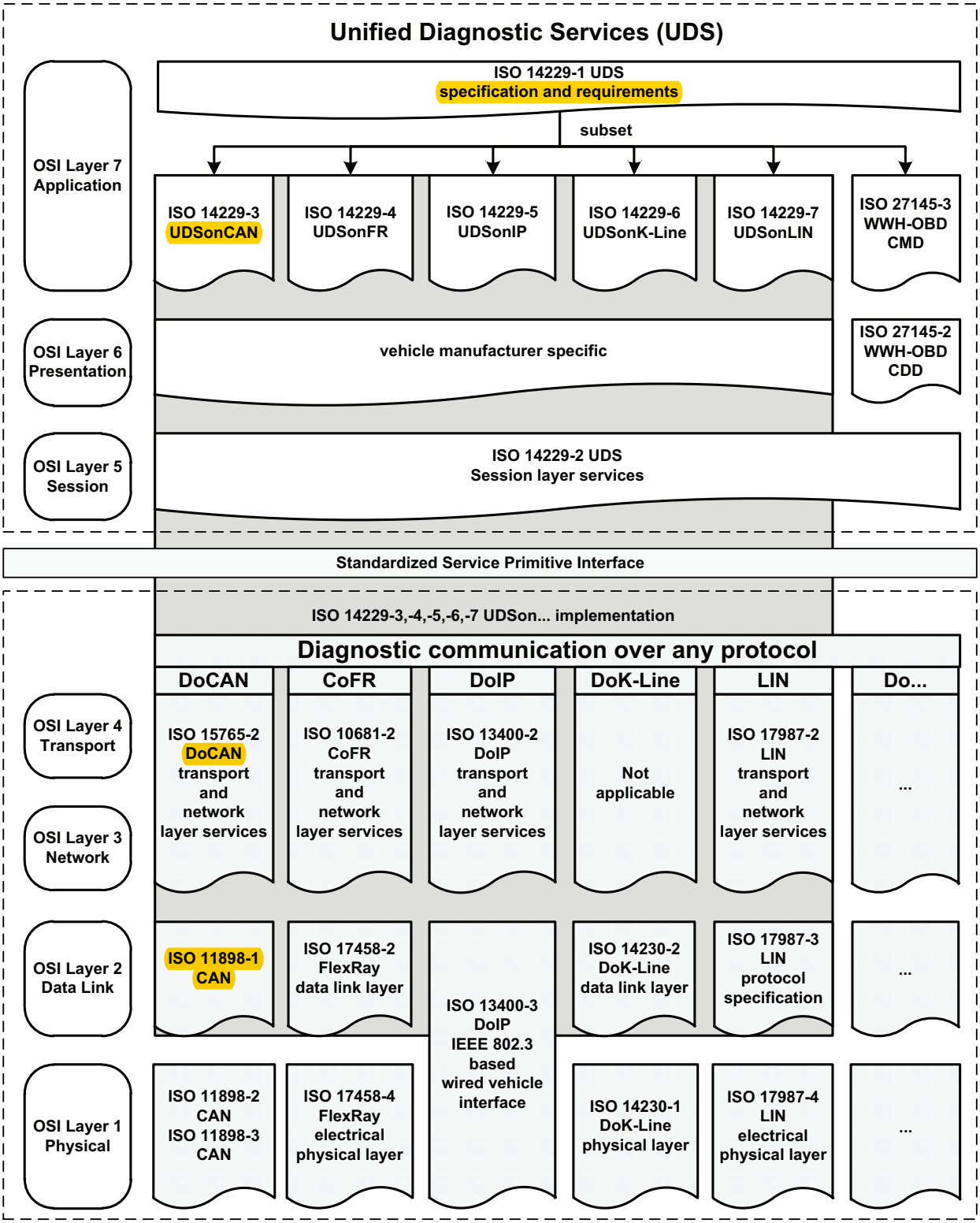


Figure 2 — Implementation of UDS document reference according to OSI model

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

6 Application layer services

6.1 General

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

The service access point of the diagnostics application layer provides a number of services that all have the same general structure. For each service, six service primitives are specified:a **service request primitive**, used by the client function in the diagnostic tester application, to pass data about a requested diagnostic service to the diagnostics application layer;

— a **service request primitive**, used by the client function in the diagnostic tester application, to pass data about a requested diagnostic service to the diagnostics application layer;

— a **service request-confirmation primitive**, used by the client function in the diagnostic tester application, to indicate that the data passed in the service request primitive is successfully sent on the vehicle communication bus the diagnostic tester is connected to

— a **service indication primitive**, used by the diagnostics application layer, to pass data to the server function of the ECU diagnostic application;

— a **service response primitive**, used by the server function in the ECU diagnostic application, to pass response data provided by the requested diagnostic service to the diagnostics application layer;

— a **service response-confirmation primitive**, used by the server function in the ECU diagnostic application, to indicate that the data passed in the service response primitive is successfully sent on the vehicle communication bus the ECU received the diagnostic request on;

— a **service confirmation primitive** used by the diagnostics application layer to pass data to the client function in the diagnostic tester application.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

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

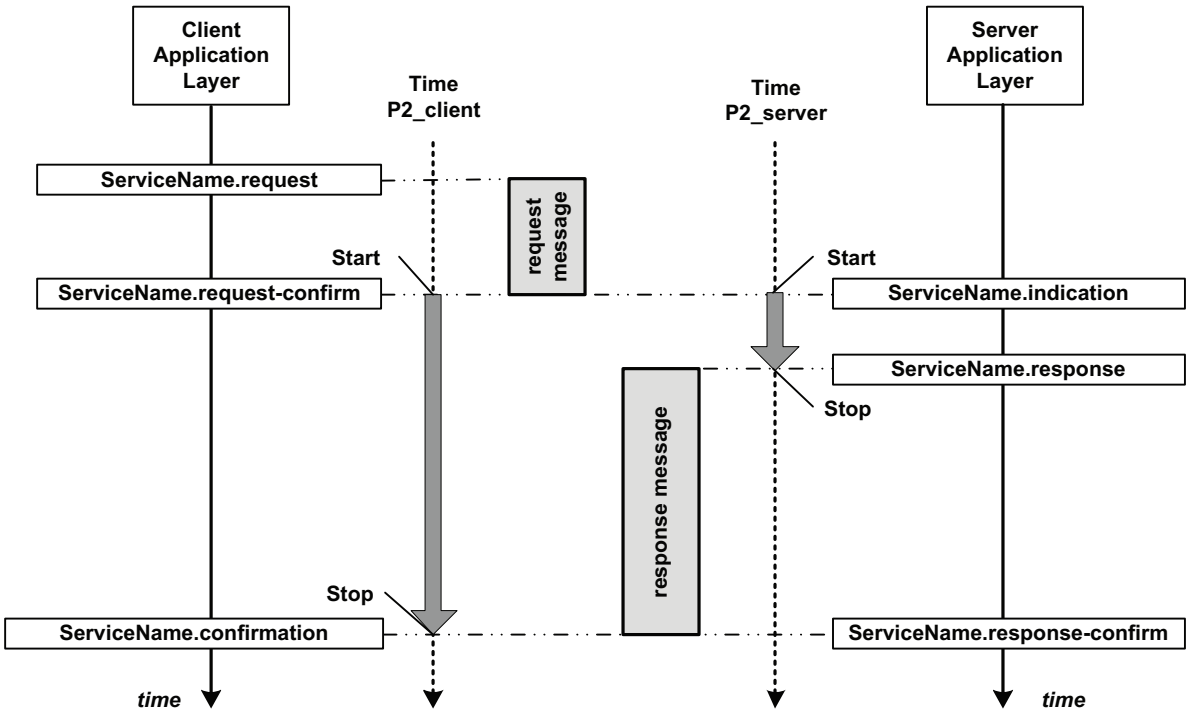


Figure 3 — Application layer service primitives - confirmed service

Figure 4 depicts the application layer service primitives - unconfirmed service.

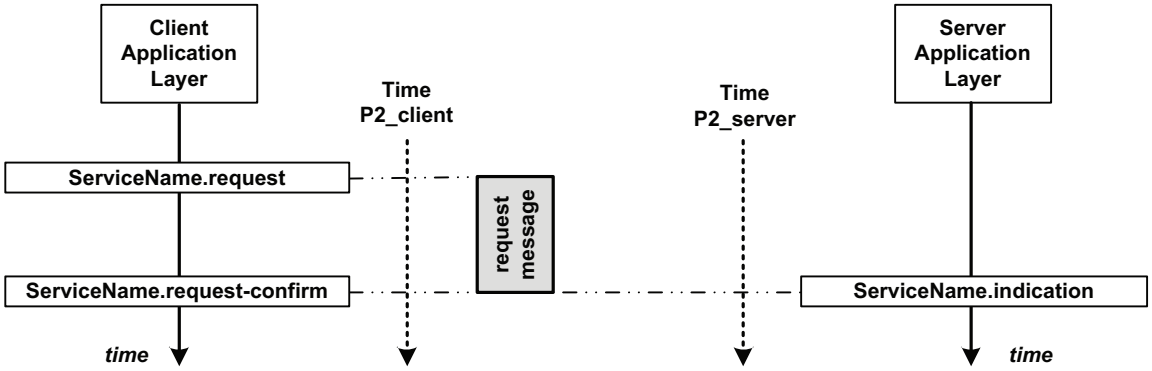


Figure 4 — Application layer service primitives - unconfirmed service

For a given service, the request-confirmation primitive and the response-confirmation primitive always have the same service data unit. The purpose of these service primitives is to indicate the completion of an earlier request or response service primitive invocation. The service descriptions in this International Standard will not make use of those service primitives, but the data link specific implementation documents might use those to define e.g. 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).

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

6.2 Format description of application layer services

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

The different formats for application layer services are specified in 6.3.

6.3 Format description of service primitives

6.3.1 General definition

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

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

Where:

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

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

— "parameter A, ..." is the A_SDU (Application layer Service Data Unit) as a list of values passed by the service primitive (addressing information),

— "parameter A, parameter B, parameter C" are mandatory parameters that shall be included in all service calls,

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

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

6.3.2 Service request and service indication primitives

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

```
service_name.request  (
                      A_MType,
                      A_SA,
                      A_TA,
                      A_TA_type,
                      [A_AE],
                      A_Length,
                      A_Data[,parameter 1, ...],
                      )
```

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

```
service_name.indication   (
                      A_MType,
                      A_SA,
                      A_TA,
                      A_TA_type,
                      [A_AE],
                      A_Length
                      A_Data[,parameter 1, ...],
                      )
```

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

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

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

6.3.3 Service response and service confirm primitives

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

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

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

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

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

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

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

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

6.3.4 Service request-confirm and service response-confirm primitives

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

```
service_name.req_confirm  (
                          A_Mtype,
                          A_SA,
                          A_TA,
                          A_TA_type,
                          [A_AE],
                          A_Result
                          )
```

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

```
service_name.rsp_confirm  (
                          A_Mtype,
                          A_SA,
                          A_TA,
                          A_TA_type,
                          [A_AE],
                          A_Result
                          )
```

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

6.4 Service data unit specification

6.4.1 Mandatory parameters

6.4.1.1 General definition

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

6.4.1.2 A_Mtype, Application layer message type

Type: enumeration

Range: diagnostics, remote diagnostics

Description:

The parameter Mtype shall be used to identify the format of the vehicle diagnostic system as specified in 6.2. This part of ISO 14229 specifies a range of two 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.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

6.4.1.3 A_SA, Application layer source address

Type: 2 byte unsigned integer value

Range: 0x0000 – 0xFFFF

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.

6.4.1.4 A_TA, Application layer target address

Type: 2 byte unsigned integer value

Range: 0x0000 – 0xFFFF

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.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

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

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

6.4.1.5 A_TA_Type, Application layer target address type

Type: enumeration

Range: physical, functional

Description:

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

6.4.1.6 A_Result

Type: enumeration

Range: ok, error

Description:

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

6.4.1.7 A_Length

Type: 4 byte unsigned integer value

Range: $0_d – (2^{32}-1)_d$

Description:

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

6.4.1.8 A_Data

Type: string of bytes

Range: not applicable

Description:

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

6.4.2 Vehicle system requirements

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

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

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

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

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

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

6.4.3 Optional parameters - A_AE, Application layer remote address

Type: 2 byte unsigned integer value

Range: 0x0000 – 0xFFFF

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 its 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.

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

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

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

7 Application layer protocol

7.1 General definition

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

The only exception from this rule shall be a few cases when functional addressing is used or the request/indication specifies that no response/confirmation shall be generated. In order not to burden the system with many unnecessary messages, there are a few cases when a negative response messages 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 specification (e.g., see 7.5).

The application layer protocol shall be handled in parallel with the session layer protocol. This mean 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).

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

7.2 Protocol data unit specification

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

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

Where:

— "Mtype, SA, TA, TA_type, RA, Length " are the same parameters as used in the A_SDU;

— "A_Data" is a string of byte data defined for each individual application layer service. The A_Data string shall start with the A_PCI followed by all service specific parameters from the A_SDU as specified for each service. The brackets indicate that this part of the parameter list may be empty;

— "Length" determines the number of bytes of A_Data;

7.3 Application protocol control information

7.3.1 PCI, Protocol Control Information

The A_PCI consists of two formats. The formats identified by the value of the first byte of the A_PCI parameter. For all service requests and for service responses with first byte unequal to 0x7F, the following definition shall apply:

```
A_PCI   (
        SI
        )
```

Where:

— "SI" is the parameter Service identifier;

For service responses with first byte equal to 0x7F, 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 (0x2A, see 10.5) no A_PCI is present in the application layer protocol data unit (A_PDU).

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

7.3.2 SI, Service Identifier

Type: 1 byte unsigned integer value

Range: 0x00 – 0xFF according to definitions in Table 2.

Table 2 — Service identifier values

Service identifier (SI)	Service type (bit 6)	Where defined
0x10 – 0x3E	ISO 14229-1 service requests	ISO 14229-1
0x3F	Not applicable	Reserved by document
0x50 – 0x7E	ISO 14229-1 positive service responses	ISO 14229-1
0x7F	Negative response service identifier	ISO 14229-1
0x80 – 0x82	Not applicable	Reserved by ISO 14229-1
0x83 – 0x88	ISO 14229-1 service requests	ISO 14229-1
0x89 – 0xB9	Not applicable	Reserved by ISO 14229-1
0xBA – 0xBE	Service requests	Defined by system supplier
0xBF – 0xC2	Not applicable	Reserved by ISO 14229-1
0xC3 – 0xC8	ISO 14229-1 positive service responses	ISO 14229-1
0xC9 – 0xF9	Not applicable	Reserved by ISO 14229-1
0xFA – 0xFE	Positive service responses	Defined by system supplier
0xFF	Not applicable	Reserved by document

NOTE There is a one-to-one correspondence between service identifiers for request messages and service identifiers for positive response messages, with bit 6 of the SI 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 (0x2A, see 10.5) service.

Description:

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

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

7.3.3 NR_SI, Negative response service identifier

Type: 1 byte unsigned integer value

Fixed value: 0x7F

Description:

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

NOTE The NR_SI value is co-ordinated 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.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

7.4 Negative response/confirmation service primitive

Each diagnostic service has a negative response/negative confirmation message specified with message A_Data bytes according to Table 3. The first A_Data byte (A_PCI.NR_SI) is always the specific negative response service identifier. The second A_Data byte (A_PCI.SI) shall be a copy of the service identifier value from the service request/indication message 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	0xXXXX	SA
TA	Target Address	M	0xXXXX	TA
TAtype	Target Address type	M	0xXX	TAT
RA	Remote Address (optional)	C	0xXXXX	RA
A_Data.A_PCI.NR_SI	Negative Response SID	M	0x7F	SIDNR
A_Data.A_PCI. SI	<Service Name> Request SID	M	0xXX	SIDRQ
A_Data.Parameter 1	responseCode	M	0xXX	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.

7.5 Server response implementation rules

7.5.1 General definitions

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

Legend

Abbreviation	Description
suppressPosRspMsgIndicationBit	TRUE = server shall NOT send a positive response message (exception see Annex A.1 in definition of NRC 0x78) 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 meesage
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 must be supported by the server
None	None of the requested data-parameter of the client request message is supported by the server

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

 © ISO 2013 – All rights reserved

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 (serviceNotSupportedIn-ActiveSession), SFNS (sub-functionNotSupported), SFNSIAS (sub-functionNotSupportedInActive-Session), and ROOR (requestOutOfRange) shall not be transmitted when functional addressing was used for the request message (exception see Annex A.1 in definition of NRC 0x78).

7.5.2 General server response behaviour

The general server response behaviour specified in this subclause is mandatory for all request messages. The validation steps starts 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,

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

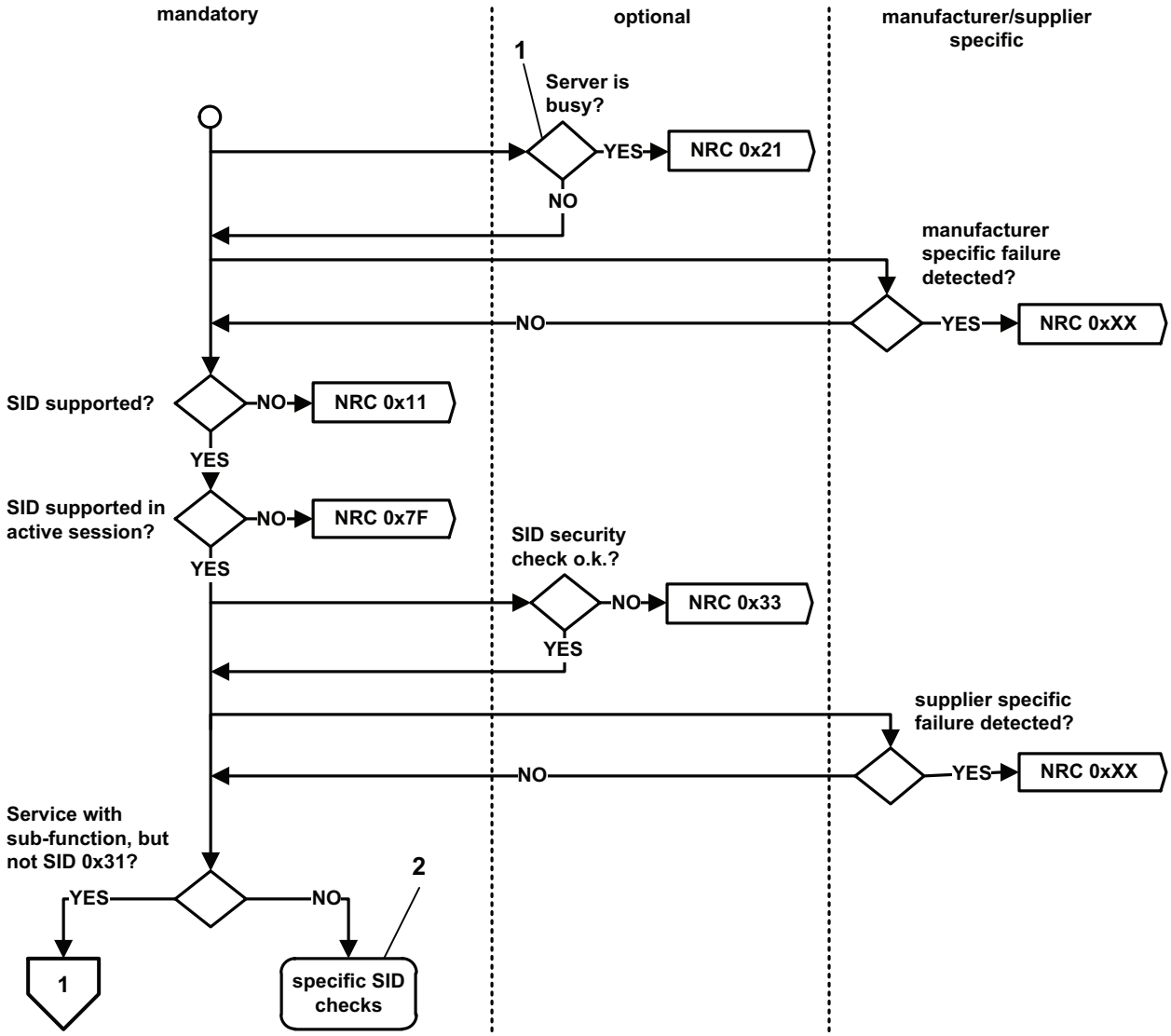
NOTE Depending on the choices implemented in all figures specifying NRC handling, a specific NRC is not guaranteed for all possible test pattern sequences.

Figure 5 depicts the general server response behaviour.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved



Key

1 Diagnostic request can not 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

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

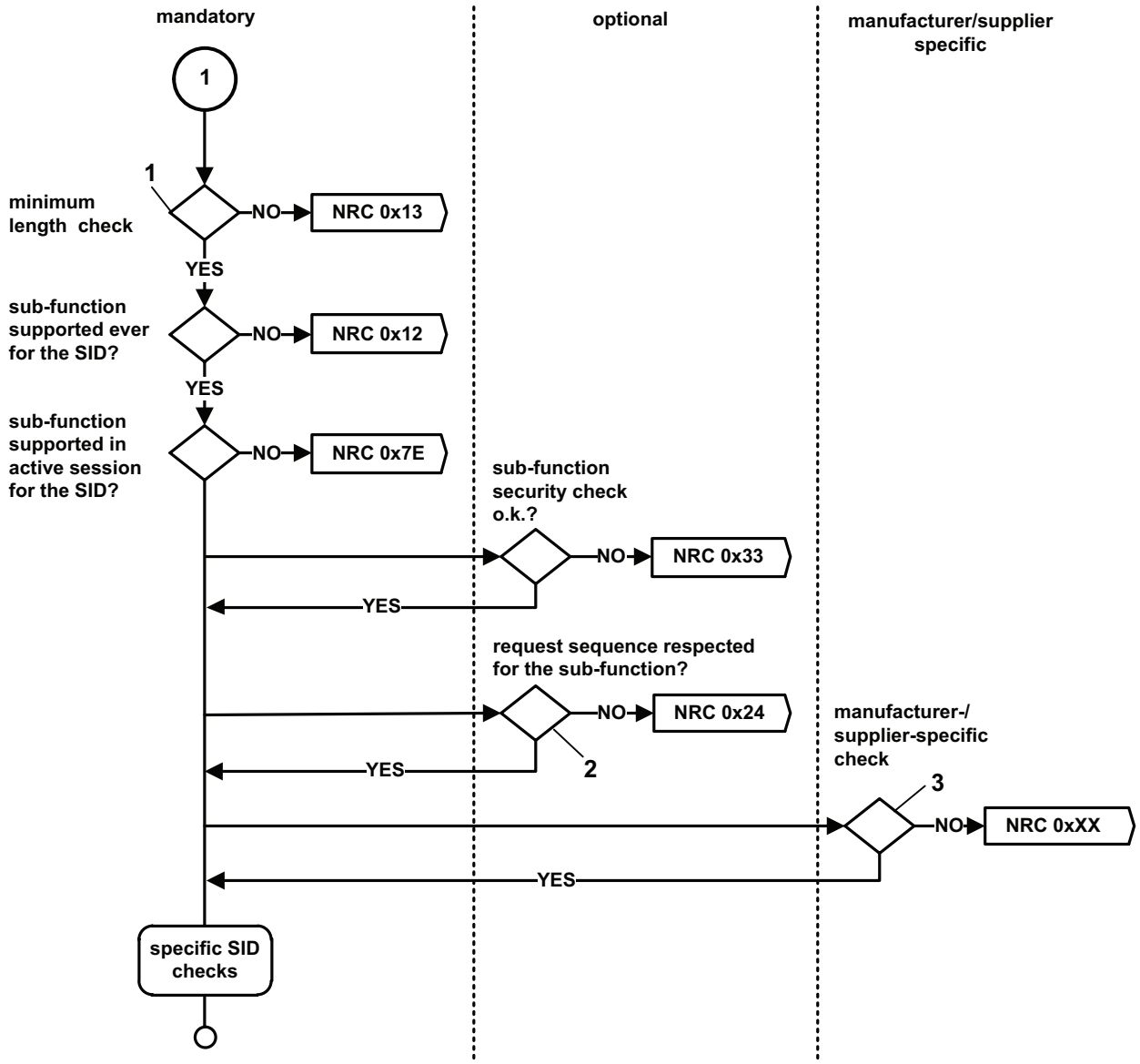
© ISO 2013 – All rights reserved

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

7.5.3.1 General server response behaviour for request messages with sub-function parameter

The general server response behaviour specified in this subclause is mandatory for all request messages with sub-function parameter. A request message in the context of this section is defined as a service request message adhering to the formatting requirements defined in this part of ISO 14229.

Figure 6 depicts the general server response behaviour for request messages with sub-function parameter.



Key

1 at least 2 (SID+SubFunction Parameter).

2 if sub-function 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 sub-function parameter

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

7.5.3.2 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 sub-function parameter in the physically addressed request message received from the client.

Table 4 shows possible communication schemes with physical addressing.

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

Server case #	Client request message		Server capability			Server response		Comments to server response
	Addressing scheme	sub-function (suppress-PosRspMsg-Indication-Bit)	SI supported	SubFunction supported	Data parameter 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= 0xXX	Server sends negative response because error occurred reading the data-parameters of the request message
c)					None		NRC= ROOR	Negative response with NRC 0x31
d)			NO	--	--		NRC= SNS or SNSIAS	Negative response with NRC 0x11 or NRC 0x7F
e)			YES	NO	--		NRC= SFNS or SFNSIAS	Negative response with NRC 0x12 or NRC 0x7E
f)		TRUE (bit = 1)	YES	YES	At least 1	NoRsp	---	Server does NOT send a response
g)					At least 1	NegRsp	NRC= 0xXX	Server sends negative response because error occurred reading the data-parameters of the request message
h)					None		NRC= ROOR	Negative response with NRC 0x31
i)			NO	--	--		NRC= SNS	Negative response with NRC 0x11
j)			YES	NO	--		NRC= SFNS	Negative response with NRC 0x12

Description of server response cases on physically addressed client request messages with sub-function:

a) Server sends a positive response message because the service identifier and sub-function parameter is supported of the client's request with indication for a response message.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

b) Server sends a negative response message (e.g., IMLOIF: incorrectMessageLengthOrIncorrectFormat) because the service identifier and sub-function parameter is supported of the client's request, but some other error appeared (e.g. wrong PDU length according to service identifier and sub-function parameter in the request message) during processing of the sub-function.

c) Server sends a negative response message with the negative response code ROOR (requestOutOfRange) because the service identifier and sub-function 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 (sub-functionNotSupported) or SFNSIAS (sub-FunctionNotSupportedInActiveSession) because the service identifier is supported and the sub-function 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 sub-function parameter is supported of the client's request with indication for no response message.

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

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

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

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

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

7.5.3.3 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 sub-function parameter in the functionally addressed request message received from the client.

Table 5 shows possible communication schemes with functional addressing.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

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

Server case #	Client request message		Server capability			Server response		Comments to server response
	Addressing scheme	sub-function (suppressPosRspMsgIndicationBit)	SI supported	Sub-Function supported	Data parameter 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= 0xXX	Server sends negative response because error occurred reading 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= 0xXX	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 sub-function:

a) Server sends a positive response message because the service identifier and sub-function parameter is 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 sub-function parameter is supported of the client's request, but some other error appeared (e.g. wrong PDU length according to service identifier and sub-function parameter in the request message) during processing of the sub-function.

c) Server sends no response message because the negative response code ROOR (requestOutOfRange, which is identified by the server because the service identifier and sub-function parameter are supported but a required data-parameter is not supported of the client's request) is always suppressed in case of a functionally addressed request message. The suppressPosRspMsgIndicationBit does not matter in such case.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – 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 (sub-functionNotSupported) and SFNSIAS (sub-functionNotSupportedInActiveSession), which are identified by the server because the service identifier is supported and the sub-function 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 sub-function parameter is supported of the client's request with indication for no response message.

NOTE 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 sub-function 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 (sub-functionNotSupported) and SFNSIAS (sub-functionNotSupportedInActiveSession), which are identified by the server because the service identifier is supported and the sub-function parameter is not supported of the client's request, are always suppressed in case of a functionally addressed request message. The suppressPosRspMsgIndicationBit does not matter in such case.

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

7.5.4.1 General server response behaviour for request messages without sub-function parameter

There is no general server response behaviour available for request messages without sub-function parameter. A request message in the context of this section is defined as a service request message adhering to the formatting requirements defined in this part of ISO 14229.

7.5.4.2 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 sub-function parameter but a data-parameter in the physically addressed request message received from the client.

Table 6 shows possible communication schemes with physical addressing.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

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

Server case #	Client request message	Server capability		Server response		Comments to server response
	Addressing scheme	SI supported	Parameter supported	Message	NRC	
a)	physical	YES	ALL	PosRsp	---	Server sends positive response
b)			At least 1		---	Server sends positive response
c)			At least 1	NegRsp	NRC=0xXX	Server sends negative response because error occured reading data-parameters of request message
d)			NONE		NRC= ROOR	Negative response with NRC 0x31
e)		NO	---		NRC=SNS or SNSIAS	Negative response with NRC 0x11 or NRC 0x7F

Description of server response cases on physically addressed client request messages without sub-function (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.

7.5.4.3 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 sub-function parameter but a data-parameter in the functionally addressed request message received from the client.

Table 7 shows possible communication schemes with functional addressing.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

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

Server case #	Client request message	Server capability		Server response		Comments to server response
	Addressing scheme	SI supported	Parameter 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= 0xXX	Server sends negative response because error occured reading 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 sub-function (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.

7.5.5 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_sub-function:                          /* test if service with sub-function is supported */
      IF (message_length >= 2) THEN                         /* check minimum length of message with sub-function */
         SWITCH (A_PDU.A_Data.A_Data.Parameter1 & 0x7F)
                                                            /* get sub-function parameter value without bit 7 */
            {
            CASE sub-function_00:                           /* test if sub-function parameter value is supported */
               IF (message_length == expected_sub-function_message_length) THEN
                  :                                         /* prepare response message */
```

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

```
                    responseCode = positiveResponse;        /* positive response message; set internal NRC = 0x00
*/
                 ELSE
                    responseCode = IMLOIF;                  /* NRC 0x13: incorrectMessageLengthOrInvalidFormat */
                 ENDIF
                 BREAK;
              CASE sub-function_01:                         /* test if sub-function parameter value is supported */
                 :                                          /* prepare response message */
                 responseCode = positiveResponse;           /* positive response message; set internal NRC = 0x00
*/
                 :
              CASE sub-function_127:                        /* test if sub-function parameter value is supported */
                 :  /* prepare response message */
                 responseCode = positiveResponse;           /* positive response message; set internal NRC = 0x00
*/
                 BREAK;
              DEFAULT:
                 responseCode = SFNS;                       /* NRC 0x12: sub-functionNotSupported */
              }
       ELSE
          responseCode = IMLOIF;                            /* NRC 0x13: incorrectMessageLengthOrInvalidFormat */
       ENDIF
       suppressPosRspMsgIndicationBit = (A_PDU.A_Data.Parameter1 & 0x80);
                                                            /* results in either 0x00 or 0x80 */
       IF ( (suppressPosRspMsgIndicationBit) && (responseCode == positiveResponse) &&
          ("not yet a NRC 0x78 response sent")) THEN        /*  test  if  positive  response  is  required  and  if
responseCode
                                                                  is positive 0x00 */
          suppressResponse = TRUE;                          /* flag to NOT send a positive response message */
       ELSE
          suppressResponse = FALSE;                         /* flag to send the response message */
       ENDIF
       BREAK;
    CASE Service_without_sub-function:                      /* test if service without sub-function 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 = 0x00
*/
          ELSE
              responseCode = ROOR;                          /* NRC 0x31: requestOutOfRange */
          ENDIF
       ELSE
          responseCode = IMLOIF;                            /* NRC 0x13: incorrectMessageLengthOrInvalidFormat */
       ENDIF
       BREAK;
    DEFAULT:
       responseCode = SNS;                                  /* NRC 0x11: serviceNotSupported */
    }
IF (A_PDU.TA_type == functional && ((responseCode == SNS) ¦¦ (responseCode == SFNS) ¦¦ (responseCode == SNSIAS) ¦¦
(responseCode == SFNSIAS) ¦¦ (responseCode == ROOR)) &&
("not yet a NRC 0x78 response sent")) THEN
                                                            /* suppress negative response message */
ELSE
    IF (suppressResponse == TRUE) THEN                      /* suppress positive response message */
    ELSE                                                    /* send negative or positive response */
    ENDIF
ENDIF
```

When functional addressing is used for the request message, and the negative response message with NRC=RCRRP (requestCorrectlyReceivedResponsePending) needs to be sent, then the final negative response message using NRC=SNS (serviceNotSupported), NRC=SNSIAS (serviceNotSupportedIn-ActiveSession), NRC=SFNS (sub-functionNotSupported), NRC=SFNSIAS (sub-functionNotSupportedIn-ActiveSession) or NRC=ROOR (requestOutOfRange) shall also be sent if it is the result of the PDU analysis of the received request message.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

7.5.6 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 have to 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 has to 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 one or more legislated diagnostic requests and one of these requests is received while a non-legislated service (e.g., enhanced diagnostics) is active, then the active service shall be aborted, the default session shall be started and the legislated diagnostic service shall be processed. This requirement does not apply if the programming session is active.

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

8 Service description conventions

8.1 Service description

This subclause defines how each diagnostic service is described in this specification. It defines 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 defines the A_PDU parameter conventions.

Table 8 — A_PDU parameter conventions

Type	Name	Description
M	Mandatory	The parameter has to be present in the A_PDU.
C	Conditional	The parameter can be present in the A_PDU, based on certain criteria (e.g. sub-function/parameters within the A_PDU).
S	Selection	Indicates that the parameter is mandatory (unless otherwise specified) and is a selection from a parameter list.
U	User option	The parameter may or may not be present, depending on dynamic usage by the user.
NOTE The "<Service Name> Request SID" marked as 'M' (Mandatory), shall not imply that this service has to 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.		

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

8.2 Request message

8.2.1 Request message definition

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

Table 9 defines the request A_PDU definition with sub-function

Table 9 — Request A_PDU definition with sub-function

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	sub-function = [parameter]	S	xx	LEV_PARAM
A_Data.Parameter 2 : A_Data.Parameter k	data-parameter#1 : data-parameter#k-1	U : U	xx : xx	DP_…#1 : 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 defines the request A_PDU definition without sub-function.

Table 10 — Request A_PDU definition without sub-function

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 : A_Data.Parameter k	data-parameter#1 : data-parameter#k	U : U	xx : xx	DP_…#1 : 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.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

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

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

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

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

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

Table 11 — SubFunction parameter structure

Bit position	Description
7	**suppressPosRspMsgIndicationBit** This bit indicates if a positive response message shall be suppressed by the server. '0' = FALSE, do not suppress a positive response message (a positive response message is required). '1' = TRUE, suppress response message (a positive response message shall not be sent; the server being addressed shall not send a positive response message). Independent of the suppressPosRspMsgIndicationBit, negative response messages are sent by the server(s) according to the restrictions specified in 7.5. Even if a positive response is not required (i.e., SPRMIB = true), the execution of the service must be completely passed to keep the implementation consistent regardless of SPRMIB value. suppressPosRspMsgIndicationBit values of both '0' and '1' shall be supported for all sub-function parameter values (i.e., bits 6-0 of the sub-function structure) supported by the server for any given service.
6-0	**sub-function parameter value** The bits 0-6 of the sub-function parameter contain the sub-function parameter value of the service (0x00 – 0x7F).

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

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

Each service contains a table that defines values for the sub-function parameter values, taking 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 0x19 SF 0x0A), as this would defeat the purpose of SPRMIB. For the server implementation it is recommended to send NRC 0x78 (RCRRP) and subsequently also send the positive response, in case paged-buffer-handling is used while SPRMIB is TRUE.

Table 12 defines the request message sub-function parameter definition.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 12 — Request message sub-function parameter definition

Bits 6 – 0	Description	Cvt	Mnemonic
xx	**sub-function#1** description of sub-function parameter#1	M/U	SUBFUNC1
:	:	:	:
xx	**sub-function#m** description of sub-function parameter#m	M/U	SUBFUNCm

The convention (Cvt) column in the Table 12 above shall be interpreted as defined in Table 13.

Table 13 — SubFunction parameter conventions

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

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

Table 14 defines the calculation of the sub-function byte value.

Table 14 — Calculation of the sub-function byte value

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

8.2.3 Request message data-parameter definition

This subclause defines the data-parameter(s) $DataParam (DP_) for the request/indication of the service <Service Name>. This subclause does not contain any definition 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 defines 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

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

8.3 Positive response message

8.3.1 Positive response message definition

This subclause defines the A_PDU parameters for the service response / confirmation (see 7.2 for a detailed description of the application layer protocol data unit A_PDU). There might be a separate table for each sub-function parameter $Level when the response messages of the different sub-function parameters $Level differ in the structure of the A_Data parameters.

NOTE 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 defines 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
A_Data.Parameter 1 : A_Data.Parameter k	data-parameter#1 : data-parameter#k	U : U	xx : xx	DP_…#1 : DP_…#k
C: The RA (Remote Address) PDU parameter is only present in case of remote addressing.				

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

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

8.3.2 Positive response message data-parameter definition

This subclause defines 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 defines the response data-parameters.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 17 — Response data-parameter definition

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

8.4 Supported negative response codes (NRC_)

This subclause defines the negative response codes that shall be implemented for this service. The circumstances under which each response code would occur are documented in a table as given below. The definition of the negative response message can be found in 7.4. The server shall use the negative response A_PDU for the indication of an identified error condition.

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

Table 18 defines the supported negative response codes.

Table 18 — Supported negative response codes

NRC	Description	Mnemonic
0xXX	NegativeResponseCode#1 1. condition#1 : m. condition#m	NRC_
:	:	NRC_
0xXX	NegativeResponseCode#n 1. condition#1 : k. condition#k	NRC_

8.5 Message flow examples

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

Table 19 defines the request message flow example.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

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	0xXX	SIDRQ
#2	sub-function/data-parameter#1	0xXX	LEV_/DP_
:	:	0xXX	DP_
#n	data-parameter#m	0xXX	DP_

Table 20 defines 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	0xXX	SIDPR
#2	data-parameter#1	0xXX	DP_
:	:	:	:
#n	data-parameter#n-1	0xXX	DP_

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

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

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	0x7F	SIDRSIDNRQ
#2 (A_PCI.SI)	<Service Name> Request SID	0xXX	SIDRQ
#3	responseCode	0xXX	NRC_

9 Diagnostic and Communication Management functional unit

9.1 Overview

Table 22 defines the Diagnostic and Communication Management functional unit.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

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.
AccessTimingParameter	The client uses this service to read/modify the timing parameters for an active communication.
SecuredDataTransmission	The client uses this service to perform data transmission with an extended data link security.
ControlDTCSetting	The client controls the setting of DTCs in the server.
ResponseOnEvent	The client requests to setup and/or control an event mechanism in the server.
LinkControl	The client requests control of the communication baudrate.

9.2 DiagnosticSessionControl (0x10) service

9.2.1 Service description

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

A diagnostic session enables a specific set of diagnostic services and/or functionality in the server(s). This service provides the capability that the server(s) can report data link layer specific parameter values valid for the enabled diagnostic session (e.g. timing parameter values). The user of this International Standard 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).

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

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

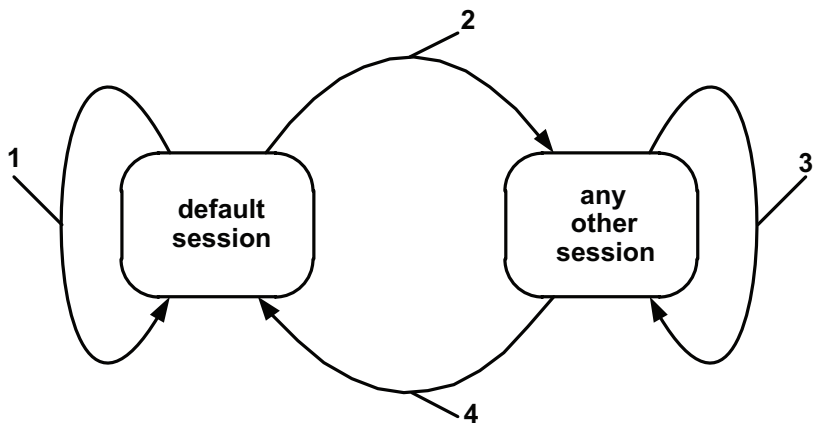
© ISO 2013 – All rights reserved

— 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 0xF4 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).

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.



Key

1 default session: 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 session: When the server transitions from the defaultSession to any other session than the defaultSession then the server shall only stop the events (similar to stopResponseOnEvent) that have been configured in the server via the ResponseOnEvent (0x86) 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 (0x86) service shall be stopped.

ii) Security shall be relocked. Note that 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 default session: When the server transitions from any diagnostic session other than the default session to the defaultSession then the server shall stop each event that has been configured in the server via the ResponseOnEvent (0x86) service and security shall be enabled. 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

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 23 defines the services, which are allowed during the defaultSession and the non-defaultSession (timed services). Any non-defaultSession is tied to a diagnostic session timer that has to be kept active by the client.

Table 23 — Services allowed during default and non-default diagnostic session

Service	defaultSession	non-defaultSession
DiagnosticSessionControl – 0x10	x	x
ECUReset – 0x11	x	x
SecurityAccess – 0x27	not applicable	x
CommunicationControl – 0x28	not applicable	x
TesterPresent – 0x3E	x	x
AccessTimingParameter – 0x83	not applicable	x
SecuredDataTransmission – 0x84	not applicable	x
ControlDTCSetting – 0x85	not applicable	x
ResponseOnEvent – 0x86	x[a]	x
LinkControl – 0x87	not applicable	x
ReadDataByIdentifier – 0x22	x[b]	x
ReadMemoryByAddress – 0x23	x[c]	x
ReadScalingDataByIdentifier – 0x24	x[b]	x
ReadDataByPeriodicIdentifier – 0x2A	not applicable	x
DynamicallyDefineDataIdentifier – 0x2C	x[d]	x
WriteDataByIdentifier – 0x2E	x[b]	x
WriteMemoryByAddress – 0x3D	x[c]	x
ClearDiagnosticInformation – 0x14	x	x
ReadDTCInformation – 0x19	x	x
InputOutputControlByIdentifier – 0x2F	not applicable	x
RoutineControl – 0x31	x[e]	x
RequestDownload – 0x34	not applicable	x
RequestUpload – 0x35	not applicable	x
TransferData – 0x36	not applicable	x
RequestTransferExit – 0x37	not applicable	X
RequestFileTransfer – 0x38	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 7.5.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

9.2.2 Request message

9.2.2.1 Request message definition

Table 24 defines the request message.

Table 24 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	DiagnosticSessionControl Request SID	M	0x10	DSC
#2	sub-function = [diagnosticSessionType]	M	0x00 – 0xFF	LEV_DS_

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

The sub-function parameter diagnosticSessionType is used by the DiagnosticSessionControl service to select the specific behaviour of the server. Explanations and usage of the possible diagnostic sessions are detailed in Table 25.

The following sub-function values are specified (suppressPosRspMsgIndicationBit (bit 7) not shown).

Table 25 — Request message sub-function parameter definition

Bit 6-0	Description	Cvt	Mnemonic
0x00	**ISOSAEReserved** This value is reserved by this document.	M	ISOSAERESRVD
0x01	**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. NOTE 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

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 25 — *(continued)*

Bit 6-0	Description	Cvt	Mnemonic
0x02	**ProgrammingSession** This diagnosticSession enables all diagnostic services required to support the memory programming of a server. In case the server runs the programmingSession in the boot software, the programmingSession shall only be left via an ECUReset (0x11) service initiated by the client, a DiagnosticSessionControl (0x10) 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 (0x10) 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.).	U	PRGS
0x03	**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
0x04	**safetySystemDiagnosticSession** This diagnosticSession enables all diagnostic services required to support safety system related functions (e.g., airbag deployment).	U	SSDS
0x05 – 0x3F	**ISOSAEReserved** This value is reserved by this document for future definition.	M	ISOSAERESRVD
0x40 – 0x5F	**vehicleManufacturerSpecific** This range of values is reserved for vehicle manufacturer specific use.	U	VMS
0x60 – 0x7E	**systemSupplierSpecific** This range of values is reserved for system supplier specific use.	U	SSS
0x7F	**ISOSAEReserved** This value is reserved by this document for future definition.	M	ISOSAERESRVD

9.2.2.3 Request message data-parameter definition

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

9.2.3 Positive response message

9.2.3.1 Positive response message definition

Table 26 defines the positive response message definition.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 26 — Positive response message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	DiagnosticSessionControl Response SID	M	0x50	DSCPR
#2	sub-function = [diagnosticSessionType]	M	0x00 – 0xFF	LEV_DS_
#3 : #6	sessionParameterRecord[]#1 = [data#1 : data#4]	M : M	0x00 – 0xFF : 0x00 – 0xFF	SPREC_ DATA_1 : DATA_m

9.2.3.2 Positive response message data-parameter definition

Table 27 defines the response message data-parameter definition.

Table 27 — Response message data-parameter definition

Definition
diagnosticSessionType This parameter is an echo of bits 6 - 0 of the sub-function parameter from the request message.
sessionParameterRecord This parameter record contains session specific parameter values reported by the server. The content 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	0x00 – 0xFF	P2SMH
#2	$P2_{Server_max}$ (low byte)	M	0x00 – 0xFF	P2SML
#3	$P2^*_{Server_max}$ (high byte)	M	0x00 – 0xFF	P2ESMH
#4	$P2^*_{Server_max}$ (low byte)]	M	0x00 – 0xFF	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 0x78) $P2_{Server_max}$ 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 $P2^*_{Server}$.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

9.2.4 Supported negative response codes (NRC_)

The following negative response codes shall be implemented for this service. The circumstances under which each response code would occur are documented in Table 30. The listed negative responses shall be used if the error scenario applies to the server.

Table 30 — Supported negative response codes

NRC	Description	Mnemonic
0x12	**sub-functionNotSupported** This NRC shall be sent if the sub-function parameter is not supported.	SFNS
0x13	**incorrectMessageLengthOrInvalidFormat** This NRC shall be sent if the length of the message is wrong.	IMLOIF
0x22	**conditionsNotCorrect** This NRC shall be returned if the criteria for the request DiagnosticSessionControl are not met.	CNC

9.2.5 Message flow example(s) DiagnosticSessionControl

9.2.5.1 Example #1 - Start programmingSession

This message flow shows how to enable the diagnostic session "programmingSession" in a server. The client requests to have a response message by setting the suppressPosRspMsgIndicationBit (bit 7 of the sub-function parameter) to "FALSE" ('0'). For the given example it is assumed that the $P2_{Server_max}$ is equal to 50 ms and the $P2^*_{Server_max}$ is equal to 5 000 ms.

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

Table 31 — DiagnosticSessionControl 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	DiagnosticSessionControl Request SID	0x10	DSC
#2	diagnosticSessionType = programmingSession, suppressPosRspMsgIndicationBit = FALSE	0x02	DS_ECUPRGS

Table 32 defines the DiagnosticSessionControl positive response message flow example #1.

Table 32 — DiagnosticSessionControl positive response message flow example #1

Message direction	**server → client**		
Message Type	**Response**		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte Value**	**Mnemonic**
#1	DiagnosticSessionControl Response SID	0x50	DSCPR
#2	diagnosticSessionType = programmingSession	0x02	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)]	0x00 0x32 0x01 0xF4	SPREC_1 SPREC_2 SPREC_3 SPREC_4

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

9.3 ECUReset (0x11) service

9.3.1 Service description

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

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

IMPORTANT — The server and the client shall meet the request and response message behaviour as specified in 7.5.

This part of ISO 14229 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.

9.3.2 Request message

9.3.2.1 Request message definition

Table 33 defines the request message definition.

Table 33 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ECUReset Request SID	M	0x11	ER
#2	sub-function = [resetType]	M	0x00 – 0xFF	LEV_RT_

9.3.2.2 Request message sub-function Parameter $Level (LEV_) definition

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

Table 34 defines the request message sub-function parameter definition.

Table 34 — Request message sub-function parameter definition

Bits 6 – 0	Description	Cvt	Mnemonic
0x00	**ISOSAEReserved** This value is reserved by this document.	M	ISOSAERESRVD
0x01	**hardReset** This value 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 the standard. It might result in the re-initialization of both volatile memory and non-volatile memory locations to predetermined values.	U	HR

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 34 — *(continued)*

Bits 6 – 0	Description	Cvt	Mnemonic
0x02	**keyOffOnReset** This value 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 the standard. Typically the values of non-volatile memory locations are preserved; volatile memory will be initialized.	U	KOFFONR
0x03	**softReset** This value 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 the standard. A typical action is to restart the application without reinitializing of previously learned configuration data, adaptive factors and other long-term adjustments.	U	SR
0x04	**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 sub-function is only applicable to a server supporting a stand-by-mode!	U	ERPSD
0x05	**disableRapidPowerShutDown** This value requests the server to disable the previously enabled "rapid power shut down" function.	U	DRPSD
0x06 – 0x3F	**ISOSAEReserved** This range of values is reserved by this document for future definition.	M	ISOSAERESRVD
0x40 – 0x5F	**vehicleManufacturerSpecific** This range of values is reserved for vehicle manufacturer specific use.	U	VMS
0x60 – 0x7E	**systemSupplierSpecific** This range of values is reserved for system supplier specific use.	U	SSS
0x7F	**ISOSAEReserved** This value is reserved by this document for future definition.	M	ISOSAERESRVD

9.3.2.3 Request message data-parameter definition

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

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

9.3.3 Positive response message

9.3.3.1 Positive response message definition

Table 35 defines the positive response message.

Table 35 — Positive response message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ECUReset Response SID	M	0x51	ERPR
#2	sub-function = [resetType]	M	0x00 – 0x7F	LEV_RT_
#3	powerDownTime	C	0x00 – 0xFF	PDT
C: This parameter is present if the sub-function parameter is set to the enableRapidPowerShutDown value (0x04);				

9.3.3.2 Positive response message data-parameter definition

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

Table 36 — Response message data-parameter definition

Definition
resetType This parameter is an echo of bits 6 - 0 of the sub-function 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: — 0x00 – 0xFE: 0 – 254 seconds powerDownTime, — 0xFF: indicates a failure or time not available.

9.3.4 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.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 37 — Supported negative response codes

NRC	Description	Mnemonic
0x12	**sub-functionNotSupported** This NRC shall be sent if the sub-function parameter is not supported.	SFNS
0x13	**incorrectMessageLengthOrInvalidFormat** This NRC shall be sent if the length of the message is wrong.	IMLOIF
0x22	**conditionsNotCorrect** This NRC shall be returned if the criteria for the ECUReset request is not met.	CNC
0x33	**securityAccessDenied** This NRC shall be sent if the requested reset is secured and the server is not in an unlocked state.	SAD

9.3.5 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 sub-function parameter) to 'FALSE'.

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

Table 38 defines 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	0x11	ER
#2	ResetType = hardReset, suppressPosRspMsgIndicationBit = FALSE	0x01	RT_HR

Table 39 defines 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	0x51	ERPR
#2	resetType = hardReset	0x01	RT_HR

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

9.4 SecurityAccess (0x27) service

9.4.1 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),

— 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' sub-function 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 0x03 is active and a tester request is successful in unlocking the security level associated with requestSeed 0x01, then only the secured functionality supported by the security level associated with requestSeed 0x01 shall be unlocked at that time. Any additional secured functionality that was previously unlocked by the security level associated with requestSeed 0x03 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 requires the client to start over from the beginning with a SecurityAccess 'requestSeed' message (refer to Annex I for additional details regarding security access handling details).

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.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

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,

— Secured diagnostic service.

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 7.5.

9.4.2 Request message

9.4.2.1 Request message definition

Table 40 defines the request message definition - sub-function = requestSeed.

Table 40 — Request message definition - sub-function = requestSeed

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	SecurityAcces Request SID	M	0x27	SA
#2	sub-function = [securityAccessType = requestSeed]	M	0x01, 0x03, 0x05, 0x07 – 0x7D	LEV_ SAT_RSD
#3 : #n	securityAccessDataRecord[] = [parameter#1 : parameter#m]	U : U	0x00 – 0xFF : 0x00 – 0xFF	SECACCDR_ PARA1 : PARAm

Table 41 defines the request message definition - sub-function = sendKey.

Table 41 — Request message definition - sub-function = sendKey

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	SecurityAcces Request SID	M	0x27	SA
#2	sub-function = [securityAccessType = sendKey]	M	0x02, 0x04, 0x06, 0x08 – 0x7E	LEV_SAT_SK
#3 : #n	securityKey[] = [key#1 (high byte) : key#m (low byte)]	M : U	0x00 – 0xFF : 0x00 – 0xFF	SECKEY_ KEY1HB : KEYmLB

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

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

The sub-function 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 = 0x01” identifies a fixed relationship between “requestSeed = 0x01” and “sendKey = 0x02”

— “requestSeed = 0x03” identifies a fixed relationship between “requestSeed = 0x03” and “sendKey = 0x04”

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

Table 42 — Request message sub-function parameter definition

Bits 6 – 0	Description	Cvt	Mnemonic
0x00	**ISOSAEReserved** This value is reserved by this document.	M	ISOSAERESRVD
0x01	**requestSeed** RequestSeed with the level of security defined by the vehicle manufacturer.	U	RSD
0x02	**sendKey** SendKey with the level of security defined by the vehicle manufacturer.	U	SK
0x03, 0x05, 0x07 – 0x41	**requestSeed** RequestSeed with different levels of security defined by the vehicle manufacturer.	U	RSD
0x04, 0x06, 0x08 – 0x42	**sendKey** SendKey with different levels of security defined by the vehicle manufacturer.	U	SK
0x43 – 0x5E	**ISOSAEReserved** This value is reserved by this document for future definition.	M	ISOSAERESRVD
0x5F	**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
0x60	**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
0x61 – 0x7E	**systemSupplierSpecific** This range of values is reserved for system supplier specific use.	U	SSS
0x7F	**ISOSAEReserved** This value is reserved by this document for future definition.	M	ISOSAERESRVD

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

9.4.2.3 Request message data-parameter definition

Table 43 defines 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 e.g. contain an identification of the client that is verified in the server.

9.4.3 Positive response message

9.4.3.1 Positive response message definition

Table 44 defines 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	sub-function = [securityAccessType]	M	00-7F	LEV_SAT_SK
#3 : #n	securitySeed[] = [seed#1 (high byte) : seed#m (low byte)]	C : C	0x00 – 0xFF : 0x00 – 0xFF	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.				

9.4.3.2 Positive response message data-parameter definition

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

Table 45 — Response message data-parameter definition

Definition
securityAccessType This parameter is an echo of bits 6 - 0 of the sub-function 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 sub-function set to a value which requests the seed of the server.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

9.4.4 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
0x12	**sub-functionNotSupported** This NRC shall be sent if the sub-function parameter is not supported.	SFNS
0x13	**incorrectMessageLengthOrInvalidFormat** This NRC shall be sent if the length of the message is wrong.	IMLOIF
0x22	**conditionsNotCorrect** This NRC shall be returned if the criteria for the request SecurityAccess are not met.	CNC
0x24	**requestSequenceError** Send if the ‘sendKey’ sub-function is received without first receiving a ‘requestSeed’ request message.	RSE
0x31	**requestOutOfRange** This NRC shall be sent if the user optional securityAccessDataRecord contains invalid data.	ROOR
0x35	**invalidKey** Send if an expected 'sendKey' sub-function value is received and the value of the key does not match the server's internally stored/calculated key.	IK
0x36	**exceededNumberOfAttempts** Send if the delay timer is active due to exceeding the maximum number of allowed false access attempts.	ENOA
0x37	**requiredTimeDelayNotExpired** Send if the delay timer is active and a request is transmitted.	RTDNE

9.4.5 Message flow example(s) SecurityAccess

9.4.5.1 Assumptions

For the below given message flow examples the following conditions have to be fulfilled to successfully unlock the server if it is in a “locked” state:

— sub-function to request the seed: 0x01 (requestSeed)

— sub-function to send the key: 0x02 (sendKey)

— seed of the server (2 bytes): 0x3657

— key of the server (2 bytes): 0xC9A9 (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 sub-function parameter) to "FALSE" ('0').

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

9.4.5.2 Example #1 - server is in a “locked” state

9.4.5.2.1 Step #1: Request the Seed

Table 47 defines 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	0x27	SA
#2	SecurityAccessType = requestSeed, suppressPosRspMsgIndicationBit = FALSE	0x01	SAT_RSD

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

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	0x67	SAPR
#2	securityAccessType = requestSeed	0x01	SAT_RSD
#3	securitySeed [byte#1] = seed#1 (high byte)	0x36	SECHB
#4	securitySeed [byte#2] = seed#2 (low byte)	0x57	SECLB

9.4.5.2.2 Step #2: Send the Key

Table 49 defines 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	0x27	SA
#2	securityAccessType = sendKey, suppressPosRspMsgIndicationBit = FALSE	0x02	SAT_SK
#3	securityKey [byte#1] = key#1 (high byte)	0xC9	SECKEY_HB
#4	securityKey [byte#2] = key#2 (low byte)	0xA9	SECKEY_LB

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 50 defines 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	0x67	SAPR
#2	securityAccessType = sendKey	0x02	SAT_SK

9.4.5.3 Example #2 - server is in an "unlocked" state

9.4.5.3.1 Step #1: Request the Seed

Table 51 defines 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	0x27	SA
#2	securityAccessType = requestSeed, suppressPosRspMsgIndicationBit = FALSE	0x01	SAT_RSD

Table 52 defines 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	0x67	SAPR
#2	securityAccessType = requestSeed	0x01	SAT_RSD
#3	securitySeed [byte#1] = seed#1 (high byte)	0x00	SECHB
#4	securitySeed [byte#2] = seed#2 (low byte)	0x00	SECLB

9.5 CommunicationControl (0x28) service

9.5.1 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).

IMPORTANT — The server and the client shall meet the request and response message behaviour as specified in 7.5.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

9.5.2 Request message

9.5.2.1 Request message definition

Table 53 defines the request message.

Table 53 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	CommunicationControl Request SID	M	0x28	CC
#2	sub-function = [controlType]	M	0x00 – 0xFF	LEV_CTRLTP
#3	communicationType	M	0x00 – 0xFF	CTP
#4	nodeIdentificationNumber (high byte)	C[a]	0x00 – 0xFF	NIN
#5	nodeIdentificationNumber (low byte)	C[a]	0x00 – 0xFF	NIN
[a] The presence of the C parameter requires the controlType either being 0x04 or 0x05.				

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

The sub-function 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 sub-function parameter definition

Bits 6 – 0	Description	Cvt	Mnemonic
0x00	**enableRxAndTx** This value indicates that the reception and transmission of messages shall be enabled for the specified communicationType.	U	ERXTX
0x01	**enableRxAndDisableTx** This value indicates that the reception of messages shall be enabled and the transmission shall be disabled for the specified communicationType.	U	ERXDTX
0x02	**disableRxAndEnableTx** This value indicates that the reception of messages shall be disabled and the transmission shall be enabled for the specified communicationType.	U	DRXETX
0x03	**disableRxAndTx** This value indicates that the reception and transmission of messages shall be disabled for the specified communicationType.	U	DRXTX
0x04	**enableRxAndDisableTxWithEnhancedAddressInformation** This value indicates that the addressed bus master shall switch the related sub-bus segment to the diagnostic-only scheduling mode.	U	ERXDTXWEAI
0x05	**enableRxAndTxWithEnhancedAddressInformation** This value indicates that the addressed bus master shall switch the related sub-bus segment to the application scheduling mode.	U	ERXTXWEAI
0x06 – 0x3F	**ISOSAEReserved** This range of values is reserved by this document for future definition.	M	ISOSAERESRVD
0x40 – 0x5F	**vehicleManufacturerSpecific** This range of values is reserved for vehicle manufacturer specific use.	U	VMS

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 54 — *(continued)*

Bits 6 – 0	Description	Cvt	Mnemonic
0x60 – 0x7E	**systemSupplierSpecific** This range of values is reserved for system supplier specific use.	U	SSS
0x7F	**ISOSAEReserved** This value is reserved by this document for future definition.	M	ISOSAERESRVD

9.5.2.3 Request message data-parameter definition

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

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 Annex 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 can not be addressed using the addressing methods of the lower OSI layers 1 to 6. This parameter is only present, if the sub-function parameter controlType is set to 0x04 or 0x05 (see Annex B.4 for the coding of the nodeIdentificationNumber data-parameter)

9.5.3 Positive response message

9.5.3.1 Positive response message definition

Table 56 defines the positive response message.

Table 56 — Positive response message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	CommunicationControl Response SID	M	0x68	CCPR
#2	sub-function = [controlType]	M	0x00 – 0x7F	LEV_CTRLTP

9.5.3.2 Positive response message data-parameter definition

Table 57 defines 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 - 0 of the sub-function parameter from the request message.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

9.5.4 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.

Table 58 — Supported negative response codes

NRC	Description	Mnemonic
0x12	**sub-functionNotSupported** This NRC shall be sent if the sub-function parameter is not supported.	SFNS
0x13	**incorrectMessageLengthOrInvalidFormat** This NRC shall be sent if the length of the message is wrong.	IMLOIF
0x22	**conditionsNotCorrect** Used when the server is in a critical normal mode activity and therefore cannot disable/enable the requested communication type.	CNC
0x31	**requestOutOfRange** The server shall use this response code, if it detects an error in the communicationType or nodeIdentificationNumber parameter.	ROOR

9.5.5 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 sub-function parameter) to "FALSE" ('0').

Table 59 defines 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	0x28	CC
#2	controlType = enableRxAndDisableTx, suppressPosRspMsgIndicationBit = FALSE	0x01	ERXTX
#3	communicationType = network management	0x02	NWMCP

Table 60 defines 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	0x68	CCPR
#2	ControlType	0x01	CTRLTP

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

9.5.6 Message flow example CommunicationControl (switch a remote network into the diagnostic-only scheduling mode where the node with address 0x000A is connected to)

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

Table 61 defines the CommunicationControl request message flow example.

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	0x28	CC
#2	controlType = enableRxAndDisableTxWithEnhancedAddressInformation, suppressPosRspMsgIndicationBit = FALSE	0x04	ERXTX
#3	communicationType = normal messages	0x01	NMCP
#4	nodeIdentificationNumber (high byte)	0x00	NIN
#5	nodeIdentificationNumber (low byte)	0x0A	NIN

Table 62 defines 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	0x68	CCPR
#2	controlType = enableRxAndDisableTxWithEnhancedAddressInformation, suppressPosRspMsgIndicationBit = FALSE	0x04	ERXTX

9.5.7 Message flow example CommunicationControl (switch to application scheduling mode with enhanced address information, the node 0x000A, 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 sub-function parameter) to "FALSE" ('0').

Table 63 defines the CommunicationControl request message flow example.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

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	0x28	CC
#2	controlType = enableRxAndTxWithEnhancedAddressInformation, suppressPosRspMsgIndicationBit = FALSE	0x05	ERXTX
#3	communicationType = normal messages	0x01	NMCP
#4	nodeIdentificationNumber (high byte)	0x00	NIN
#5	nodeIdentificationNumber (low byte)	0x0A	NIN

Table 64 defines the CommunicationControl positive response message flow example.

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	0x68	CCPR
#2	controlType = enableRxAndTxWithEnhancedAddressInformation, suppressPosRspMsgIndicationBit = FALSE	0x05	ERXTX

SHW3SZH 2013-09-29 NormMaster

9.6 TesterPresent (0x3E) service

9.6.1 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 7.5.

9.6.2 Request message

9.6.2.1 Request message definition

Table 65 defines the request message.

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 65 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	TesterPresent Request SID	M	0x3E	TP
#2	sub-function = [zeroSubFunction]	M	0x00 / 0x80	LEV_ZSUBF

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

Table 66 specifies the sub-function parameter values defined for this service (suppressPosRspMsgIndicationBit (bit 7) not shown).

Table 66 — Request message sub-function parameter definition

Bits 6 – 0	Description	Cvt	Mnemonic
0x00	**zeroSubFunction** This parameter value is used to indicate that no sub-function value beside the suppressPosRspMsgIndicationBit is supported by this service.	M	ZSUBF
0x01 – 0x7F	**ISOSAEReserved** This range of values is reserved by this document.	M	ISOSAERESRVD

9.6.2.3 Request message data-parameter definition

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

9.6.3 Positive response message

9.6.3.1 Positive response message definition

Table 67 defines the positive response message.

Table 67 — Positive response message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	TesterPresent Response SID	M	0x7E	TPPR
#2	sub-function = [zeroSubFunction]	M	0x00	LEV_ZSUBF

9.6.3.2 Positive response message data-parameter definition

Table 68 defines the data-parameter of the positive response message.

Table 68 — Response message data-parameter definition

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

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

9.6.4 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 69. The listed negative responses shall be used if the error scenario applies to the server.

Table 69 — Supported negative response codes

NRC	Description	Mnemonic
0x12	**sub-functionNotSupported** This NRC shall be sent if the sub-function parameter is not supported.	SFNS
0x13	**incorrectMessageLengthOrInvalidFormat** This NRC shall be sent if the length of the message is wrong.	IMLOIF

9.6.5 Message flow example(s) TesterPresent

9.6.5.1 Example #1 - TesterPresent (suppressPosRspMsgIndicationBit = FALSE)

Table 70 defines the TesterPresent request message flow example #1.

Table 70 — 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	0x3E	TP
#2	zeroSubFunction, suppressPosRspMsgIndicationBit = FALSE	0x00	ZSUBF

Table 71 defines the TesterPresent positive response message flow example #1.

Table 71 — 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	0x7E	TPPR
#2	zeroSubFunction, suppressPosRspMsgIndicationBit = FALSE	0x00	ZSUBF

9.6.5.2 Example #2 - TesterPresent (suppressPosRspMsgIndicationBit = TRUE)

Table 72 defines the TesterPresent request message flow example #2.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 72 — 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	0x3E	TP
#2	zeroSubFunction, suppressPosRspMsgIndicationBit = TRUE	0x80	ZSUBF

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

9.7 AccessTimingParameter (0x83) service

9.7.1 Service description

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

The use of this service is complex and depends on the server's capability and the data link topology. Only one extended timing parameter set will be supported per diagnostic session. It is recommended to use this service only with physical addressing, because of the different sets of extended timing parameters supported by the servers.

It is recommended to use the following sequence of services:

— DiagnosticSessionControl (diagnosticSessionType) service;

— AccessTimingParameter (readExtendedTimingParameterSet) service;

— AccessTimingParameter (setTimingParametersToGivenValues) service;

For the case a response is required to be sent by the server the client and server shall activate the new timing parameter settings after the server has sent the AccessTimingParameter positive response message. In case no response message is allowed the client and the server shall activate the new timing parameter after the transmission/reception of the request message.

The server and the client shall reset their timing parameters to the default values after a successful switching to another or the same diagnostic session (e.g. via DiagnosticSessionControl, ECUReset service, or a session timing timeout).

The AccessTimingParameter service provides four different modes for the access to the server timing parameters:

— readExtendedTimingParameterSet;

— setTimingParametersToDefaultValues;

— readCurrentlyActiveTimingParameters;

— setTimingParametersToGivenValues;

IMPORTANT — The server and the client shall meet the request and response message behaviour as specified in 7.5.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

9.7.2 Request message

9.7.2.1 Request message definition

Table 73 defines the request message.

Table 73 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	AccessTimingParameter Request SID	M	0x83	ATP
#2	sub-function = [timingParameterAccessType]	M	0x00 – 0xFF	LEV_TPAT_
#3 : #n	TimingParameterRequestRecord [byte#1 : byte#m]	 C : C	 0x00 – 0xFF : 0x00 – 0xFF	TPREQR_ B1 : Bm
C: The TimingParameterRequestRecord is only present if timingParameterAccessType = setTimingParametersToGivenValues. The structure and content of the TimingParameterRequestRecord is data link layer dependend and therefore defined in the implementation specification(s) of ISO 14229.				

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

The sub-function parameter timingParameterAccessType is used by the AccessTimingParameter service to select the specific behaviour of the server. Explanations and usage of the possible timingParameterIdentifiers are detailed in Table 74. The following sub-function values are specified (suppressPosRspMsgIndicationBit (bit 7) not shown):

Table 74 — Request message sub-function parameter definition

Bits 6 – 0	Description	Cvt	Mnemonic
0x00	**ISOSAEReserved** This value is reserved by this document.	M	ISOSAERESRVD
0x01	**readExtendedTimingParameterSet** Upon receiving an AccessTimingParameter indication primitive with timingParameterAccessType = readExtendedTimingParameterSet, the server shall read the extended timing parameter set, i.e. the values that the server is capable of supporting. If the read access to the timing parameter set is successful, the server shall send an AccessTimingParameter response primitive with the positive response parameters. If the read access to the timing parameters set is not successful, the server shall send a negative response message with the appropriate negative response code. This sub-function is used to provide an extra set of timing parameters for the currently active diagnostic session. With the timingParameterAccessType = setTimingParametersToGivenValues only this set (read by timingParameterAccessType = readExtendedTimingParameterSet) of timing parameters can be set.	U	RETPS

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 74 — *(continued)*

Bits 6 – 0	Description	Cvt	Mnemonic
0x02	**setTimingParametersToDefaultValues** Upon receiving an AccessTimingParameter indication primitive with timingParameterAccessType = setTimingParametersToDefaultValues, the server shall change all timing parameters to the default values and send an AccessTimingParameter response primitive with the positive response parameters before the default timing parameters become active (if suppressPosRspMsgIndicationBit is set to 'FALSE', otherwise the timing parameters shall become active after the successful evaluation of the request message). If the timing parameters cannot be changed to default values for any reason, the server shall maintain the currently active timing parameters and send a negative response message with the appropriate negative response code. The definition of the default timing values depends on the used data link and is specified in the implementation specification(s) of ISO 14229.	U	STPTDV
0x03	**readCurrentlyActiveTimingParameters** Upon receiving an AccessTimingParameter indication primitive with timingParameterAccessType = readCurrentlyActiveTimingParameters, the server shall read the currently used timing parameters. If the read access to the timing parameters is successful, the server shall send an AccessTimingParameter response primitive with the positive response parameters. If the read access to the currently used timing parameters is impossible for any reason, the server shall send a negative response message with the appropriate negative response code.	U	RCATP
0x04	**setTimingParametersToGivenValues** Upon receiving an AccessTimingParameter indication primitive with timingParameterAccessType = setTimingParametersToGivenValues, the server shall check if the timing parameters can be changed under the present conditions. If the conditions are valid, the server shall perform all actions necessary to change the timing parameters and send an AccessTimingParameter response primitive with the positive response parameters before the new timing parameter values become active (suppressPosRspMsgIndicationBit is set to 'FALSE', otherwise the timing parameters shall become active after the successful evaluation of the request message). If the timing parameters cannot be changed by any reason, the server shall maintain the currently active timing parameters and send a negative response message with the appropriate negative response code. It is not possible to set the timing parameters of the server to any set of values between the minimum and maximum values read via timingParameterAccessType = readExtendedTimingParameterSet. The timing parameters of the server can only be set to exactly the timing parameters read via timingParameterAccessType = readExtendedTimingParameterSet. A request to do so shall be rejected by the server.	U	STPTGV
0x05 – 0xFF	**ISOSAEReserved** This value is reserved by this document for future definition.	M	ISOSAERESRVD

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

9.7.2.3 Request message data-parameter definition

Table 75 defines the data-parameter of the request message.

Table 75 — Request message data-parameter definition

Definition
TimingParameterRequestRecord This parameter record contains the timing parameter values to be set in the server via timingParameterAccessType = setTimingParametersToGivenValues. The content and structure of this parameter record is data link layer specific and can be found in the implementation specification(s) of ISO 14229 e.g. ISO 14229-3.

9.7.3 Positive response message

9.7.3.1 Positive response message definition

Table 76 defines the positive response message.

Table 76 — Positive response message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	AccessTimingParameter Response SID	M	0xC3	ATPPR
#2	timingParameterAccessType	M	0x00 – 0x7F	TPAT_
#3 : #n	TimingParameterResponseRecord [byte#1 : byte#m]	C : C	0x00 – 0xFF : 0x00 – 0xFF	TPRSPR_ B1 : Bm
C: The TimingParameterResponseRecord is only present if timingParameterAccessType = readExtendedTimingParameterSet or readCurrentlyActiveTimingParameters. The structure and content of the TimingParameterResponseRecord is data link layer dependend and therefore defined in the implementation specification(s) of ISO 14229.				

9.7.3.2 Positive response message data-parameter definition

Table 77 defines the data-parameters of the positive response message.

Table 77 — Response message data-parameter definition

Definition
timingParameterAccessType This parameter is an echo of bits 6 - 0 of the sub-function parameter from the request message.
TimingParameterResponseRecord This parameter record contains the timing parameter values read from the server via timingParameterAccessType = readExtendedTimingParameterSet or readCurrentlyActiveTimingParameters. The content and structure of this parameter record is data link layer specific and can be found in the implementation specification(s) of ISO 14229.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

9.7.4 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 78. The listed negative responses shall be used if the error scenario applies to the server.

Table 78 — Supported negative response codes

NRC	Description	Mnemonic
0x12	**sub-functionNotSupported** This NRC shall be sent if the sub-function parameter is not supported.	SFNS
0x13	**incorrectMessageLengthOrInvalidFormat** The length of the message or the format is wrong.	IMLOIF
0x22	**conditionsNotCorrect** This NRC shall be returned if the criteria for the request AccessTimingParameter are not met.	CNC
0x31	**requestOutOfRange** This NRC shall be sent if the TimingParameterRequestRecord contains invalid timing parameter values.	ROOR

9.7.5 Message flow example(s) AccessTimingParameter

9.7.5.1 Example #1 – set timing parameters to default values

This message flow shows how to set the default timing parameters in a server. The client requests to have a response message by setting the suppressPosRspMsgIndicationBit (bit 7 of the sub-function parameter) to "FALSE" ('0').

Table 79 defines the AccessTimingParameter request message flow example #1.

Table 79 — AccessTimingParameter 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	AccessTimingParameter Request SID	0x83	ATP
#2	timingParameterAccessType = setTimingParametersToDefaultValues; suppressPosRspMsgIndicationBit = FALSE	0x02	TPAT_STPTDV

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 80 defines the AccessTimingParameter positive response message flow example #1.

Table 80 — AccessTimingParameter 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	AccessTimingParameter Response SID	0xC3	ATPPR
#2	timingParameterAccessType = setTimingParametersToDefaultValues	0x02	TPAT_STPTDV

9.8 SecuredDataTransmission (0x84) service

9.8.1 Service description

9.8.1.1 Purpose

The purpose of this service is to transmit data that is protected against attacks from third parties - which could endanger data security.

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.

9.8.1.2 Security sub-layer

Figure 8 illustrates the security sub-layer. The security sub-layer has to be added in the server and client application for the purpose of performing diagnostic services in a secured mode.

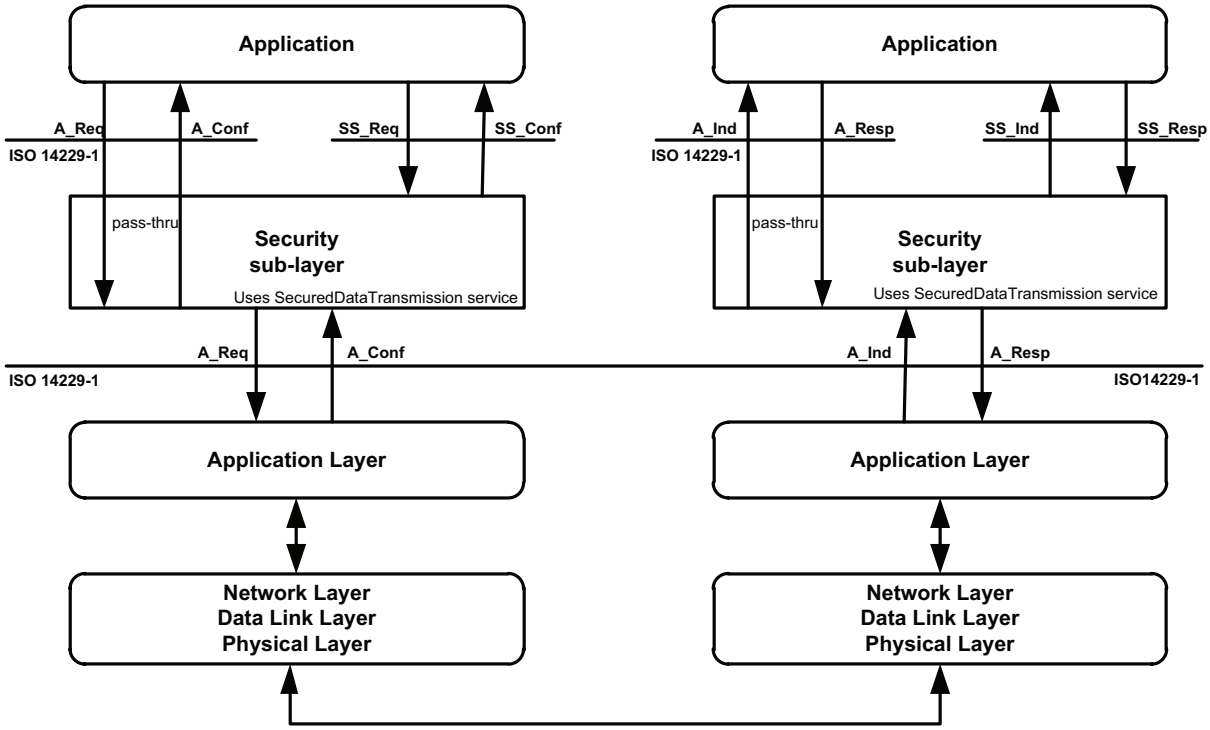


Figure 8 — Security sub-layer implementation

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

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 defined in this document to exchange data between a client and a server. The Security Sub-Layer performs a "Pass-Thru" of data between "Application" and "Application Layer" in the client and the server.

— Secured data transmission mode

The application uses the diagnostic services or external services and the Security sub-layer Service Primitives 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. Secured links must be point-to-point communication. Therefore only physical addressing is allowed, which means that only one server is involved.

The interface of the security sub-layer to the application is according to the ISO/OSI model conventions and therefore provides the following four security sub-layer (SS_) service primitives

— SS_SecuredMode.req: Security sub-layer Request

— SS_SecuredMode.ind: Security sub-layer Indication

— SS_SecuredMode.resp: Security sub-layer Response

— SS_SecuredMode.conf: Security sub-layer Confirmation

This part of ISO 14229 defines both, confirmed and unconfirmed services. In a secured mode only confirmed services are allowed (suppressPosRspMsgIndicationBit = FALSE). Based on this requirement the following services are not allowed to be executed in a secured mode:

— ResponseOnEvent (0x86);

— ReadDataByPeriodicIdentifier (0x2A);

— TesterPresent (0x3E);

The confirmed services (suppressPosRspMsgIndicationBit = FALSE) use the four application layer service primitives request, indication, response and confirmation. Those are mapped onto the four security sub-layer service primitives and vice versa when executing a confirmed diagnostic service in a secured mode.

The task of the Security sub-layer when performing a diagnostic service in a secured mode is to encrypt data provided by the "Application", to decrypt data provided by the "Application Layer" and to add, check, and remove security specific data elements. The Security sub-layer uses the SecuredDataTransmission (0x84) service 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.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

The security sub-layer provides the service "SecuredServiceExecution" to the application for the purpose of a secured execution of diagnostic services.

The security sub-layer request and indication primitive of the "SecuredServiceExecution" service are specified according to the following general format:

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


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

The security sub-layer layer response and confirm primitive of the SecuredServiceExecution service are specified according to the following general format:

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


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

The addressing information shown in the security sub-layer service primitives is mapped directly onto the addressing information of the application layer and vice versa.

9.8.1.3 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.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

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. 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 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 and presents the data of the service to be executed in a secured mode to the application via the security sub-layer SecuredServiceExecution service indiciation. The application executes the service and uses the security sub-layer SecuredServiceExecution service response 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 to transmit the response data to the client.

— 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.

Figure 9 graphically shows the interaction of the security sub-layer, the application layer and the application when executing a confirmed diagnostic service in a secured mode.

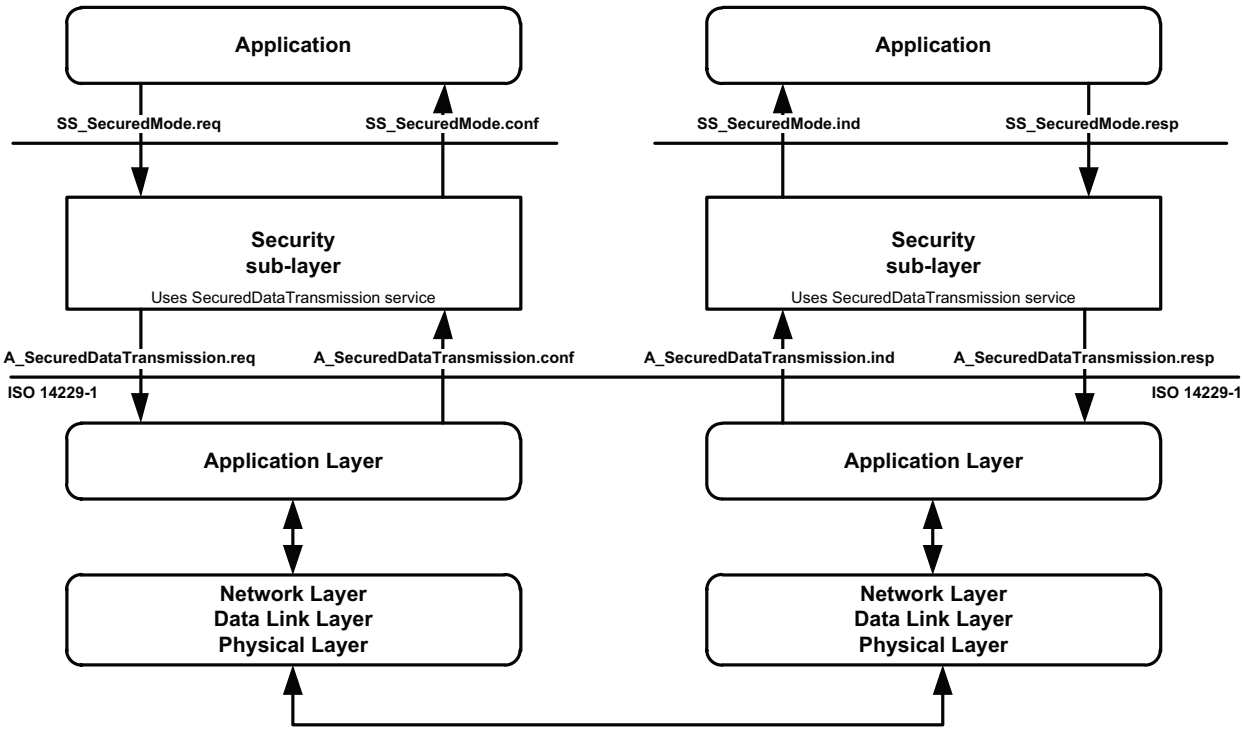


Figure 9 — Security sub-layer, application layer and application interaction

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

IMPORTANT — The server and the client shall meet the request and response message behaviour as specified in 7.5.

9.8.2 Request message

9.8.2.1 Request message definition

The security sub-layer generates the application layer SecuredDataTransmission request message parameters.

Table 81 defines the request message.

Table 81 — Request message definition

A_Data Byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	SecuredDataTransmission Request SID	M	0x84	SDT
#2 : #n	securityDataRequestRecord[] = [securityDataParameter#1 : securityDataParameter#m]	M : M	0x00 – 0xFF : 0x00 – 0xFF	SECDRQR_ SDP_ : SDP_

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

This service does not use a sub-function parameter.

9.8.2.3 Request message data-parameter definition

Table 82 defines the data-parameters of the request message.

Table 82 — Request message data-parameter definition

Definition
securityDataRequestRecord This parameter contains the data as processed by the Security Sub-Layer.

9.8.3 Positive response message

9.8.3.1 Positive response message definition

Table 83 defines the positive response message.

Table 83 — Positive response message definition

A_Data Byte	Parameter Name	Cvt	Byte Value	Mnemonic
1	SecuredDataTransmission Response SID	M	0xC4	SDTPR
2 : n	securityDataResponseRecord[] = [securityDataParameter#1 : securityDataParameter#m]	M : M	0x00 – 0xFF : 0x00 – 0xFF	SECDRQR_ SDP_ : SDP_

9.8.3.2 Positive response message data-parameter definition

Table 84 defines the data-parameter of the positive response message:

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 84 — Response message data-parameter definition

Definition
securityDataResponseRecord This parameter contains the data as processed by the Security Sub-Layer.

9.8.4 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 85. The response codes are always sent without encryption, even if according to the configurationProfile in the request A_PDU the response A_PDU has to be encrypted. The listed negative responses shall be used if the error scenario applies to the server.

Table 85 — Supported negative response codes

NRC	Description	Mnemonic
0x13	**incorrectMessageLengthOrInvalidFormat** The server shall use this response code, if the length of the request A_PDU is not correct.	IMLOIF
0x38 – 0x4F	**reservedByExtendedDataLinkSecurityDocument** This range of values is reserved by extended data link security.	RBEDLSD

NOTE The response codes listed above apply to the SecuredDataTransmission (0x84) service. In case the diagnostic service performed in a secured mode requires a negative response then this negative response is send to the client in a secured mode via a SecuredDataTransmission positive response message.

9.9 ControlDTCSetting (0x85) service

9.9.1 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 sub-function 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 sub-function set to either "on" or "off" even if the requested DTC setting state is already active.

If a ClearDiagnosticInformation (0x14) 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 - 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

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

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).

IMPORTANT — The server and the client shall meet the request and response message behaviour as specified in 7.5.

9.9.2 Request message

9.9.2.1 Request message definition

Table 86 defines the request message.

Table 86 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ControlDTCSetting Request SID	M	0x85	CDTCS
#2	sub-function = [DTCSettingType]	M	0x00 – 0xFF	LEV_DTCSTP_
#3 : #n	DTCSettingControlOptionRecord [] = [parameter#1 : parameter#m	U : U	0x00 – 0xFF : 0x00 – 0xFF	DTCSCOR_ PARA1 : PARAm

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

The sub-function 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 87).

Table 87 — Request message sub-function parameter definition

Bits 6 – 0	Description	Cvt	Mnemonic
0x00	**ISOSAEReserved** This value is reserved by this document.	M	ISOSAERESRVD
0x01	**on** The server(s) shall resume the updating of diagnostic trouble code status bits according to normal operating conditions	M	ON
0x02	**off** The server(s) shall stop the updating of diagnostic trouble code status bits.	M	OFF
0x03 – 0x3F	**ISOSAEReserved** This range of values is reserved by this document for future definition.	M	ISOSAERESRVD
0x40 – 0x5F	**vehicleManufacturerSpecific** This range of values is reserved for vehicle manufacturer specific use.	U	VMS
0x60 – 0x7E	**systemSupplierSpecific** This range of values is reserved for system supplier specific use.	U	SSS
0x7F	**ISOSAEReserved** This value is reserved by this document for future definition.	M	ISOSAERESRVD

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

9.9.2.3 Request message data-parameter definition

Table 88 defines the data-parameter of the request message.

Table 88 — 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).

9.9.3 Positive response message

9.9.3.1 Positive response message definition

Table 89 defines the positive response message.

Table 89 — Positive response message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ControlDTCSetting Response SID	M	0xC5	CDTCSPR
#2	DTCSettingType	M	00-7F	DTCSTP

9.9.3.2 Positive response message data-parameter definition

Table 90 defines the data-parameter of the positive response message.

Table 90 — Response message data-parameter definition

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

9.9.4 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 91. The listed negative responses shall be used if the error scenario applies to the server.

Table 91 — Supported negative response codes

NRC	Description	Mnemonic
0x12	**sub-functionNotSupported** This NRC shall be sent if the sub-function parameter is not supported.	SFNS
0x13	**incorrectMessageLengthOrInvalidFormat** This NRC shall be sent if the length of the message is wrong.	IMLOIF
0x22	**conditionsNotCorrect** Used when the server is in a critical normal mode activity and therefore cannot perform the requested DTC control functionality.	CNC

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 91 — *(continued)*

NRC	Description	Mnemonic
0x31	**requestOutOfRange** The server shall use this response code, if it detects an error in the DTCSettingControlOptionRecord.	ROOR

9.9.5 Message flow example(s) ControlDTCSetting

9.9.5.1 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 sub-function parameter) to "FALSE" ('0').

Table 92 defines the ControlDTCSetting request message flow example #1.

Table 92 — 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	0x85	RDTCS
#2	DTCSettingType = off, suppressPosRspMsgIndicationBit = FALSE	0x02	DTCSTP_OFF

Table 93 defines the ControlDTCSetting positive response message flow example #1.

Table 93 — 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	0xC5	RDTCSPR
#2	DTCSettingType = off	0x02	DTCSTP_OFF

9.9.5.2 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 sub-function parameter) to "FALSE" ('0').

Table 94 defines the ControlDTCSetting request message flow example #2.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 94 — 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	0x85	ENC
#2	DTCSettingType = on, suppressPosRspMsgIndicationBit = FALSE	0x01	DTCSTP_ON

Table 95 defines the ControlDTCSetting positive response message flow example #2.

Table 95 — 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	0xC5	RDTCSPR
#2	DTCSettingType = on	0x01	DTCSTP_ON

9.10 ResponseOnEvent (0x86) service

9.10.1 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 10 for a brief overview about the client and server behaviour.

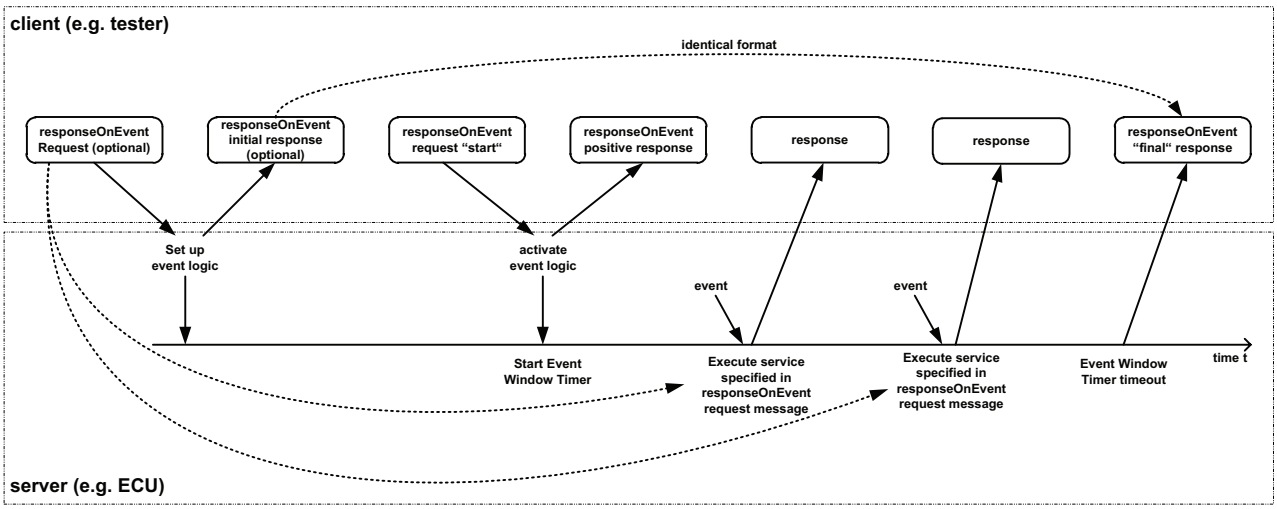


Figure 10 — ResponseOnEvent service - client and server behaviour

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

NOTE The figure above 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 sub-function and data content of the ResponseOnEvent request message at the time of the reception. This includes the following sub-function 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 0x31 shall be sent. The serviceToRespondToRecord is not part of this evaluation. The serviceToRespondToRecord parameter will be evaluated when the specified event occurs, which 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. In case conditions are not correct a negative response message with the appropriate negative response code shall be sent. Multiple events shall be signalled in the order of their occurrence.

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.

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 0x78) 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 In some circumstances due to the postponing of the ServiceToRespondTo the data contained in the ServiceToRespondTo Records may not correspond with the data values which caused the event.

c) In case multiple events occur while another event is in progress, the handling of the multiple events (e.g., ignoring all but the first/last event or stockpiling of events) shall be vehicleManufacturerSpecific.

d) 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.

e) 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.

f) 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.

g) 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.

h) If a ResponseOnEvent service has been set up then the following shall apply:

   1) If Bit 6 of the eventType subfunction 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).

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

 © ISO 2013 – All rights reserved

2) If Bit 6 of the eventType subfunction parameter is set to 1 (store event), it shall resume sending serviceToRespondTo-responses according to the ResponseOnEvent-set up after a power cycle of the server. StoreEvent is therefore only allowed in combination with infinite eventWindowTime.

i) The “suppressPosResponseMessageIndicattionBit” = “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.

j) The server shall return a ResponseOnEvent ”final” response (see fig. 15) to indicate the ResponseOnEvent (0x86) 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.

k) In order to avoid interference with normal diagnostic operation, it is recommended to implement ResponseOnEvent service only to be applied to transient events and conditions. The server shall return a response once per event occurrence. For a condition that is continuously sustained over a period of time, the response service shall be executed only one time at the initial occurrence. In case the eventType is defined so that serviceToRespondTo-responses could occur at a high frequency, then appropriate measures have to be taken in order to prevent back to back serviceToRespondTo-responses. A minimum separation time between serviceToRespondTo-responses could be part of the eventTypeRecord (vehicle-manufacturer-specific).

It is recommended to use only the services listed in Table 96 for the service to be performed in case the specified event occurs. (serviceToRespondToRecord request service identifier).

Table 96 — Recommended services to be used with the ResponseOnEvent service

Recommended services (ServiceToRespondTo)	Request SID	Response SID
ReadDataByIdentifier	0x22	0x62
ReadDTCInformation	0x19	0x59
RoutineControl	0x31	0x71
InputOutputControlByIdentifier	0x2F	0x6F

For performance reasons (e.g. avoid missed execution of serviceToRespondToRecord request service identifier) it is recommended to respect the following suggestions:

— 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 7.5.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

9.10.2 Request message

9.10.2.1 Request message definition

Table 97 defines the request message.

Table 97 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ResponseOnEvent Request SID	M	0x86	ROE
#2	sub-function = [eventType]	M	0x00 – 0xFF	LEV_ETP
#3	eventWindowTime	M	0x00 – 0xFF	EWT
#4 : #(m-1)+4	eventTypeRecord[] = [eventTypeParameter 1 : eventTypeParameter m]	C1 : C1	0x00 – 0xFF : 0x00 – 0xFF	ETR_ ETP1 : ETPm
#n-(r-1)-1 #n-(r-1) : #n	serviceToRespondToRecord[] = [serviceId serviceParameter 1 : serviceParameter r]	C2 C3 : C3	0x00 – 0xFF 0x00 – 0xFF : 0x00 – 0xFF	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 sub-function 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				

9.10.2.2 Request message sub-function Parameter $Level (LEV_) Definition

9.10.2.2.1 ResponseOnEvent request message sub-function Parameter definition

The sub-function parameter eventType is used by the ResponseOnEvent request message to specify the event to be configured in the server and to control the ResponseOnEvent set up. Each sub-function parameter value given in Table 98 also specifies the length of the applicable eventTypeRecord (suppressPosRspMsgIndicationBit (bit 7) not shown in table below).

Bit 6 of the eventType sub-function 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).

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 98 — eventType sub-function bit 6 definition - storageState

Bit 6 value	Description	Cvt	Mnemonic
0x00	**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).	M	DNSE
0x01	**storeEvent** This value indicates 1) On ROE start or stop request in the default session, that the event shall resume or stop sending serviceToRespondTo-responses according to the ResponseOnEvent-set up after a reset or power on (i.e. the ResponseOnEvent service is resumed). 2) On any ROE setup event logic request, that requested event logic shall be stored persistently till the event logic is explicitly cleared (via clearResponseOnEvent) or the event logic is overwritten by a new ROE setup event logic request of the same category.	U	SE

Table 99 defines the request message sub-function parameter.

Table 99 — Request message sub-function parameter definition

Bits 5 - 0 Value	Description	Cvt	Type of ROE sub-function	Mnemonic
0x00	**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 sub-function parameter. Length of eventTypeRecord: 0 byte	U	control	STPROE
0x01	**onDTCStatusChange** This value identifies the event as a new DTC detected matching the DTCStatusMask specified for this event. Length of eventTypeRecord: 1 byte UImplementation hint:U A server resident DTC count algorithm shall count the number of DTCs satisfying the client defined DTCStatusMask at a certain periodic rate (e.g. approximately 1 second). If the count is different from that which was calculated on the previous execution, the client shall generate the event that causes the execution of the serviceToRespondTo. The latest count shall then be stored as a reference for the next calculation. This eventType requires the specification of the DTCStatusMask in the request message (eventTypeParameter#1).	U	setup	ONDTCS
0x02	**onTimerInterrupt** This value identifies the event as a timer interrupt, but the timer and its values are not part of the ResponseOnEvent service. This eventType requires the specification of more details in the request message (eventTypeRecord). Length of eventTypeRecord: 1 byte	U	setup	OTI

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 99 — *(continued)*

Bits 5 - 0 Value	Description	Cvt	Type of ROE sub-function	Mnemonic
0x03	**onChangeOfDataIdentifier** This value identifies the event as a new internal data record identified by dataIdentifier. The data values are vehicle manufacturer specific. This eventType requires the specification of more details in the request message (eventTypeRecord). Length of eventTypeRecord: 2 bytes	U	setup	OCODID
0x04	**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
0x05	**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 start sending responses on event. Length of eventTypeRecord: 0 byte.	M	control	STRTROE
0x06	**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
0x07	**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 sub-function 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 defines the kind of comparison. The event occurs if the comparison result is positive. Length of eventTypeRecord: 10 bytes	U	setup	OCOV
0x08 – 0x1F	**ISOSAEReserved** This range of values is reserved by this document for future definition.	M	-	ISOSAERESRVD
0x20 – 0x2F	**VehicleManufacturerSpecific** This range of values is reserved for vehicle manufacturer specific use.	U	setup	VMS
0x30 – 0x3E	**SystemSupplierSpecific** This range of values is reserved for system supplier specific use.	U	setup	SSS
0x3F	**ISOSAEReserved** This value is reserved by this document for future definition.	M	-	ISOSAERESRVD

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

9.10.2.2.2 Detailed request message sub-function onTimerInterrupt parameters specification

With subfunction onTimerInterrupt the Server is allowed to trigger events in a timer configurable period of time.

The eventTypeRecord defines the timer value with the following timer schedule:

— Slow rate,

— Medium rate,

— Fast rate.

It is the manufacturer's specific task to define the time rate associated to each timer schedule option. See Table 100.

Table 100 — Comparison logic parameter definition

eventTypeRecord	Event will be generated	Timer type
0x01	Every time the slow rate timer value elapses.	Slow rate timer. E.g. timer elapses after 1 second
0x02	Every time the medium rate timer elapses.	Medium rate timer. E.g. timer elapses after 300 ms
0x03	Every time the fast rate timer elapses.	Fast rate timer. E.g. timer elapses after 25 ms

9.10.2.2.3 Detailed request message sub-function onChangeOfDataIdentifier parameters specification

With subfunction 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 loose some changes and triggers less events than expected.

The eventTypeRecord sets the two byte DID value that have to be monitored for any change.

9.10.2.2.4 Detailed request message sub-function onComparisonOfValues parameters specification

Table 101 – Table 103 specify the parameters for the request message with sub-function onComparisonOfValues parameters in case of serviceToRespondToRecord specifying a comparison between read DIDs.

Table 101 — sub-function onComparisonOfValues parameters definition

Data Byte	ParameterName	Byte Value	Comment	Detailed requirement
1	ServiceID	0x86	Request SID	---
2	eventType	0x07	sub-function onComparisonOfValues	---
3	eventWindowTime	0x02	InfiniteTimeWindow specification	---
4	eventTypeRecord byte1	0x01	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	0x04	DataIdentifier (DID) low byte	---

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 101 — *(continued)*

Data Byte	ParameterName	Byte Value	Comment	Detailed requirement
6	eventTypeRecord byte 3	0x01	Comparison logic, see Table 102	The eventTypeRecord byte 3 sets the logics of the comparison method,
7	eventTypeRecord byte 4	0x00	Raw reference comparison value MSB	The eventTypeRecord byte 4, 5, 6, 7 sets the reference comparison value.
8	eventTypeRecord byte 5	0x00	Raw reference comparison value	---
9	eventTypeRecord byte 6	0x01	Raw reference comparison value	---
10	eventTypeRecord byte 7	0x00	Raw reference comparison value LSB	---
11	eventTypeRecord byte 8	0x0A	hysteresis value	The eventTypeRecord byte 8 defines an histeresys value in percentage from 0% (0x00) to 100% (0x64).
12	eventTypeRecord byte 9	0xBC	Localization byte 1 MSB, see Table 103	The eventTypeRecord byte 9, 10 defines localization of value within the data identifier, see Table 103.
13	eventTypeRecord byte 10	0x58	Localization byte 2 LSB, see Table 103	---
14	serviceToRespondTo byte 1	0x22	SID	The serviceToRespondToRecord sets the service and the DID to be read and compared. In the first positive response message the numberOfIdentifiedEvents field is always set to 0x00.
15	serviceToRespondTo byte 2	0xA1	DID1	---
16	serviceToRespondTo byte 3	0x00	DID2	---

Table 102 defines the comparison logic parameter definition.

Table 102 — Comparison logic parameter definition

Comparison logic ID	Event will be generated when
0x01	Comparison Parameter < Measured Value
0x02	Comparison Parameter > Measured Value
0x03	Comparison Parameter = Measured Value
0x04	Comparison Parameter <> Measured Value

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 103 defines the localization of value 16 bit bitfield parameter definition.

Table 103 — 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 0x00 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 1023 bits.

9.10.2.3 Request message data-parameter definition

Table 104 defines the data-parameters of the request message.

Table 104 — 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 0x02 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 sub-function.
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.

9.10.3 Positive response message

9.10.3.1 Positive response message definition

Table 105 defines the positive response message for all sub-functions but reportActivatedEvents.

Table 105 — Positive response message definition for all sub-functions but reportActivatedEvents

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ResponseOnEvent Response SID	M	0xC6	ROEPR
#2	eventType	M	0x00 – 0x7F	ETP
#3	numberOfIdentifiedEvents	M	0x00 – 0xFF	NOIE
#4	eventWindowTime	M	0x00 – 0xFF	EWT

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 105 — *(continued)*

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
	eventTypeRecord[] = [			ETR_
#5	eventTypeParameter 1	C1	0x00 – 0xFF	ETP1
:	:	:	:	:
#(m-1)+5	eventTypeParameter m]	C1	0x00 – 0xFF	ETPm
	serviceToRespondToRecord[] = [			STRTR_
#n-(r-1)-1	serviceId	M	0x00 – 0xFF	SI
#n-(r-1)	serviceParameter 1	C2	0x00 – 0xFF	SP1
:	:	:	:	:
#n	serviceParameter r]	C2	0x00 – 0xFF	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 106 defines the positive response message for sub-function = reportActivatedEvents.

Table 106 — Positive response message definition for sub-function = reportActivatedEvents

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ResponseOnEvent Response SID	M	0xC6	ROEPR
#2	eventType = reportActivatedEvents	M	0x04	ETP_RAE
#3	numberOfActivatedEvents	M	0x00 – 0xFF	NOIE
#4	eventTypeOfActiveEvent#1	C1	0x00 – 0xFF	EVOAE
#5	eventWindowTime#1	C1	0x00 – 0xFF	EWT
	eventTypeRecord#1[] = [			ETR_
#6	eventTypeParameter 1	C2	0x00 – 0xFF	ETP1
:	:	:	:	:
#(m-1)+6	eventTypeParameter m]	C2	0x00 – 0xFF	ETPm
	serviceToRespondToRecord#1[] = [			STRTR_
#p-(o-1)-1	serviceId	C3	0x00 – 0xFF	SI
#p-(o-1)	serviceParameter 1	C4	0x00 – 0xFF	SP1
:	:	:	:	:
#p	serviceParameter o]	C4	0x00 – 0xFF	SPo
:	:	:	:	:
#n-(r-1)-4-(q-1)	eventTypeOfActiveEvent#k	C1	0x00 – 0xFF	EVOAE
#n-(r-1)-3-(q-1)	eventWindowTime#k	C1	0x00 – 0xFF	EWT
	eventTypeRecord#k[] = [			ETR_
#n-(r-1)-2-(q-1)	eventTypeParameter 1	C2	0x00 – 0xFF	ETP1
:	:	:	:	:
#n-(r-1)-2	eventTypeParameter q]	C2	0x00 – 0xFF	ETPm
	serviceToRespondToRecord#k[] = [			STRTR_
#n-(r-1)-1	serviceId	C3	0x00 – 0xFF	SI
#n-(r-1)	serviceParameter 1	C4	0x00 – 0xFF	SP1
:	:	:	:	:
#n	serviceParameter r]	C4	0x00 – 0xFF	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.				

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

9.10.3.2 Positive response message data-parameter definition

Table 107 defines the data-parameters of the positive response message.

Table 107 — Response message data-parameter definition

Definition
eventType This parameter is an echo of bits 6 - 0 of the sub-function parameter of the request message.
eventTypeOfActiveEvent This parameter is an echo of the sub-function parameter of the request message that was issued to set-up the active event. The applicable values are the ones specified for the eventType sub-function 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 send 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.
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.
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.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

9.10.4 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 108. The listed negative responses shall be used if the error scenario applies to the server.

Table 108 — Supported negative response codes

NRC	Description	Mnemonic
0x12	**sub-functionNotSupported** This NRC shall be sent if the sub-function parameter is not supported	SFNS
0x13	**incorrectMessageLengthOrInvalidFormat** This NRC shall be sent if the length of the request message is wrong.	IMLOIF
0x22	**conditionsNotCorrect** Used when the server is in a critical normal mode activity and therefore cannot perform the requested functionality.	CNC
0x31	**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; if a combination of finite event window and storageState equal to storeEvent is requested	ROOR

9.10.5 Message flow example(s) ResponseOnEvent

9.10.5.1 Assumptions

For the message flow examples it is assumed, that the eventWindowTime equal to 0x08 defines an event window of 80 seconds (eventWindowTime * 10 seconds). The client requests to have a response message by setting the suppressPosRspMsgIndicationBit (bit 7 of the sub-function 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 has to send the initial positive response message of the ResponseOnEvent service. To activate the event logic the client has to request ResponseOnEvent sub-function 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 has to respond immediately with the response message corresponding to the serviceToRespondToRecord in the ResponseOnEvent request message.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

 © ISO 2013 – All rights reserved

— **Close event window:**
It is recommended to close the event window of the server according to the parameter eventWindowTime. After this action, the server has to 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.

9.10.5.2 Example #1 - ResponseOnEvent (finite event window)

Table 109 defines the setup of ResponseOnEvent request message flow example #1.

Table 109 — 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	0x86	ROE
#2	eventTypeRecord [eventType] = onDTCStatusChange, storageState = doNotStoreEvent suppressPosRspMsgIndicationBit = FALSE	0x01	ET_ODTCSC
#3	eventWindowTime = 80 seconds	0x08	EWT
#4	eventTypeRecord [eventTypeParameter] = testFailed status	0x01	ETP1
#5	serviceToRespondToRecord [serviceId] = ReadDTCInformation	0x19	RDTCI
#6	serviceToRespondToRecord [sub-function] = reportNumberOfDTCByStatusMask	0x01	RNDTC
#7	serviceToRespondToRecord [DTCStatusMask] = testFailed status	0x01	DTCSM

Table 110 defines the ResponseOnEvent initial positive response message flow example #1.

Table 110 — 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	0xC6	ROEPR
#2	eventType = onDTCStatusChange	0x01	ET_ODTCSC
#3	numberOfIdentifiedEvents = 0	0x00	NOIE
#4	eventWindowTime = 80 seconds	0x08	EWT
#5	eventTypeRecord [eventTypeParameter] = testFailed status	0x01	ETP1
#6	serviceToRespondToRecord [serviceId] = ReadDTCInformation	0x19	RDTCI
#7	serviceToRespondToRecord [sub-function] = reportNumberOfDTCByStatusMask	0x01	RNDTC
#8	serviceToRespondToRecord [DTCStatusMask] = testFailed status	0x01	DTCSM

The event logic is set up; now it has to be activated.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 111 defines the start of the ResponseOnEvent request message flow example #1.

Table 111 — 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	0x86	ROE
#2	eventTypeRecord [eventType] = startResponseOnEvent, storageState = doNotStoreEvent, suppressPosRspMsgIndicationBit = FALSE	0x05	ET_STRTROE
#3	eventWindowTime (will not be evaluated)	0x08	EWT

Table 112 defines the ResponseOnEvent positive response message flow example #1.

Table 112 — 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	0xC6	ROEPR
#2	eventType = onDTCStatusChange	0x01	ET_ODTCSC
#3	numberOfIdentifiedEvents = 0	0x00	NOIE
#4	eventWindowTime	0x08	EWT

In case the specified event occurs the server sends the response message according to the specified serviceToRespondToRecord.

Table 113 defines the ReadDTCInformation positive response message flow example #1.

Table 113 — 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	0x59	RDTCI
#2	DTCStatusAvailibilityMask	0xFF	DTCSAM
#3	DTCCount [DTCCountHighByte] = 0	0x00	DTCCNT_HB
#4	DTCCount [DTCCountLowByte] = 4	0x04	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.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 114 defines the ResponseOnEvent request number of active events message flow example #1.

Table 114 — 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	0x86	ROE
#2	eventTypeRecord [eventType] = reportActivatedEvents, storageState = doNotStoreEvent, suppressPosRspMsgIndicationBit = FALSE	0x04	ET_RAE
#3	eventWindowTime (will not be evaluated)	0x08	EWT

Table 115 defines the ResponseOnEvent reportActivatedEvents positive response message flow example #1.

Table 115 — 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	0xC6	ROEPR
#2	eventType = reportActivatedEvents	0x04	ET_RAE
#3	numberOfActivatedEvents = 1	0x01	NOAE
#4	eventTypeOfActiveEvent = onDTCStatusChange	0x01	ET_ODTCSC
#5	eventWindowTime = 80 seconds	0x08	EWT
#6	eventTypeRecord [eventTypeParameter] = testFailed status	0x01	ETP1
#7	serviceToRespondToRecord [serviceId] = ReadDTCInformation	0x19	RDTCI
#8	serviceToRespondToRecord [sub-function] = reportNumberOfDTCByStatusMask	0x01	RNDTC
#9	serviceToRespondToRecord [DTCStatusMask] = testFailed status	0x01	DTCSM

If the specified event window time has expired the server shall send a final positive response.

Table 116 defines the ResponseOnEvent final positive response message flow example #1.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 116 — 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	0xC6	ROEPR
#2	eventType = onDTCStatusChange	0x01	ET_ODTCSC
#3	numberOfIdentifiedEvents = 1	0x01	NOIE
#4	eventWindowTime = 80 seconds	0x08	EWT
#5	eventTypeRecord [eventTypeParameter] = testFailed status	0x01	ETP1
#6	serviceToRespondToRecord [serviceId] = ReadDTCInformation	0x19	RDTCI
#7	serviceToRespondToRecord [sub-function] = reportNumberOfDTCByStatusMask	0x01	RNDTC
#8	serviceToRespondToRecord [DTCStatusMask] = testFailed status	0x01	DTCSM

9.10.5.2.1 Example #1 - flowcharts

The following flowcharts show two different kind of server behaviour:

— no event occurs within the finite event window. In this case the server has to 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..#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.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Figure 11 depicts the finite event window – no event during active event window.

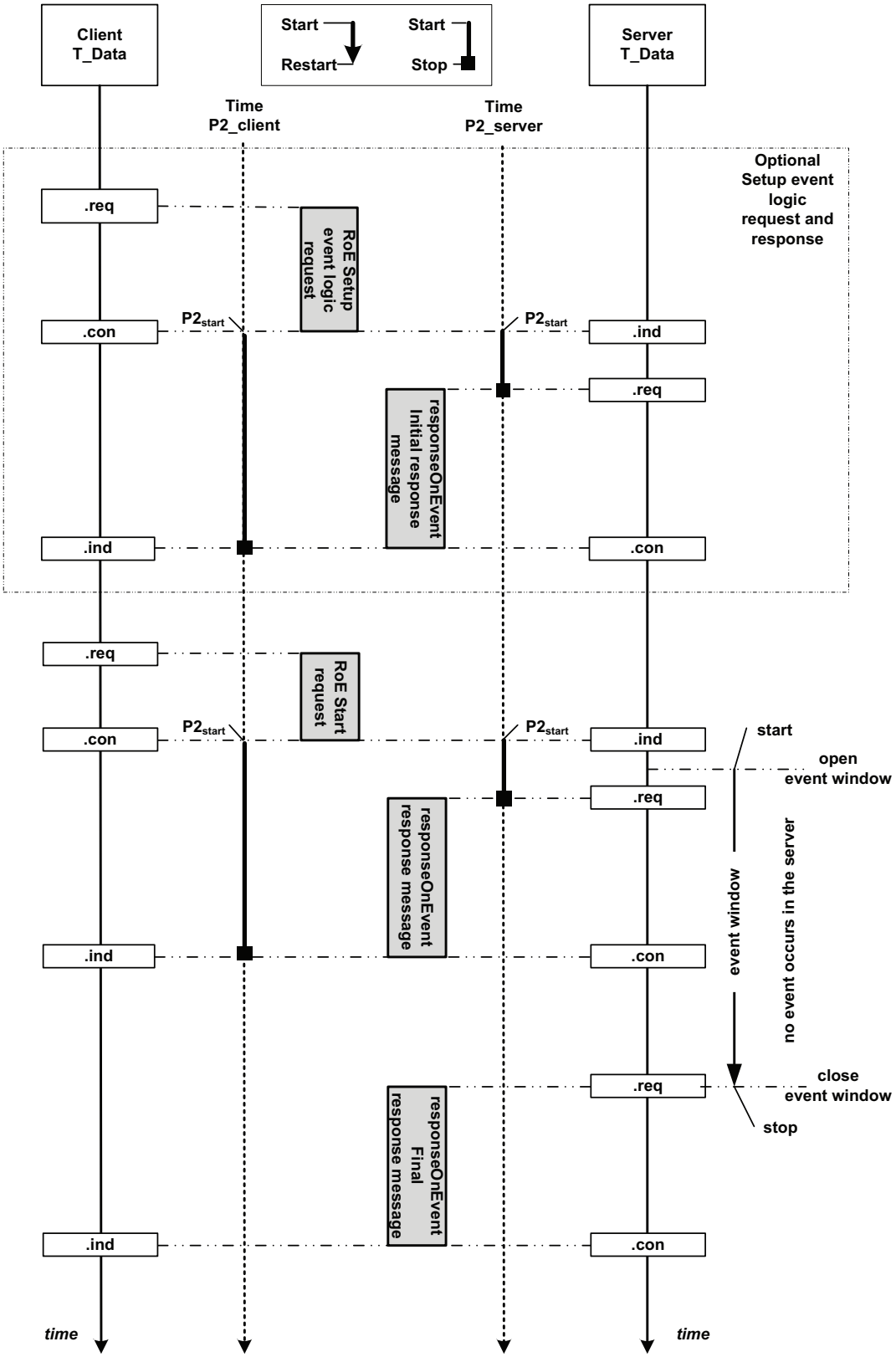


Figure 11 — Finite event window - no event during active event window

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Figure 12 depicts the finite event window – multiple events during active event window.

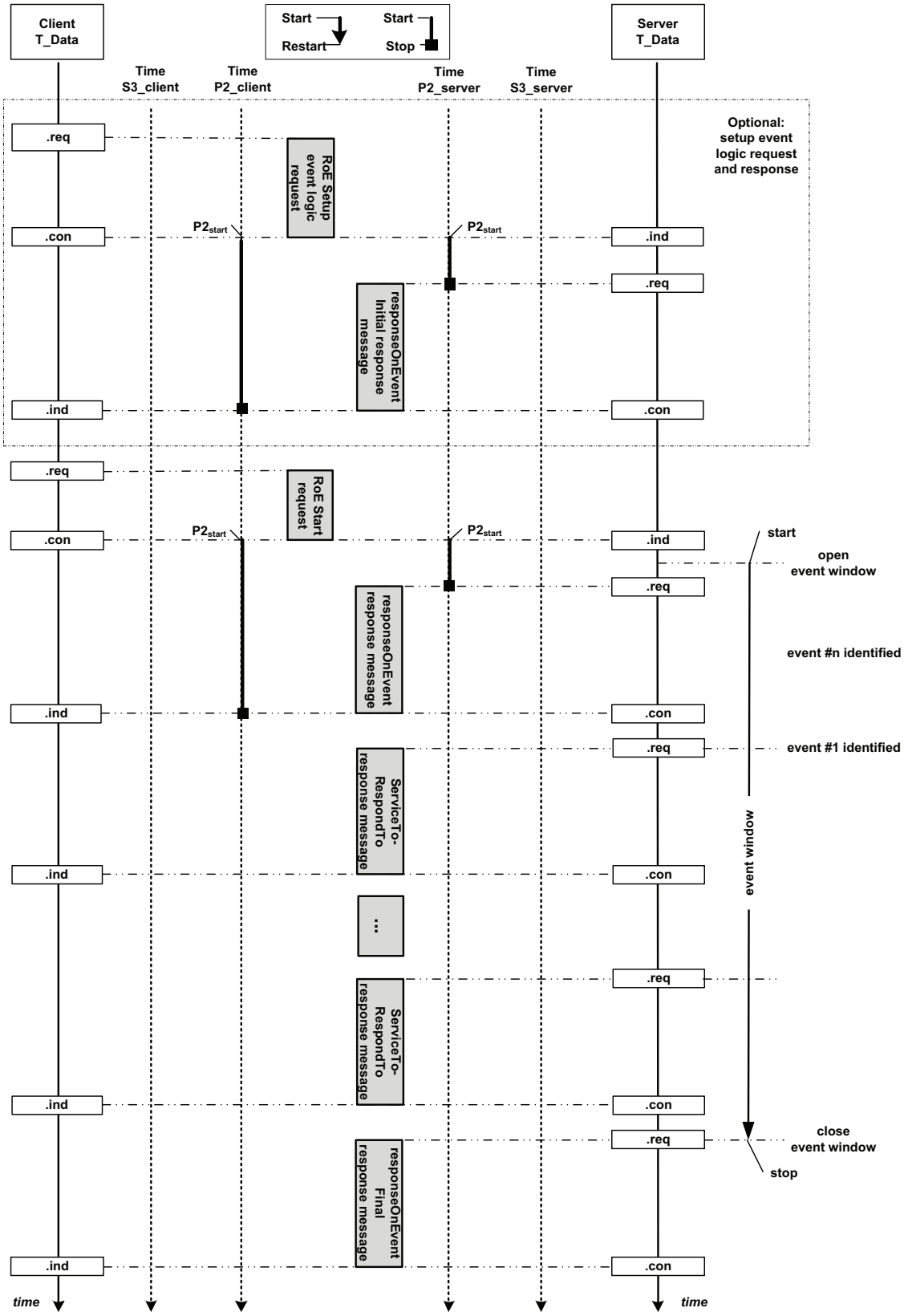


Figure 12 — Finite event window - multiple events during active event window

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

9.10.5.3 Example #2 - ResponseOnEvent (infinite event window)

Table 117 defines the ResponseOnEvent request message flow example #2.

Table 117 — 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	0x86	ROE
#2	eventTypeRecord [eventType] = onDTCStatusChange, storageState = doNotStoreEvent, suppressPosRspMsgIndicationBit = FALSE	0x01	ET_ODTCSC
#3	eventWindowTime = infinite	0x02	EWT
#4	eventTypeRecord [eventTypeParameter] = testFailed status	0x01	ETP1
#5	serviceToRespondToRecord [serviceId] = ReadDTCInformation	0x19	RDTCI
#6	serviceToRespondToRecord [sub-function] = reportNumberOfDTCByStatusMask	0x01	RNDTC
#7	serviceToRespondToRecord [DTCStatusMask] = testFailed status	0x01	DTCSM

Table 118 defines the ResponseOnEvent initial positive response message flow example #2.

Table 118 — 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	0xC6	ROEPR
#2	eventType = onDTCStatusChange	0x01	ET_ODTCSC
#3	numberOfIdentifiedEvents = 0	0x00	NOIE
#4	eventWindowTime = infinite	0x02	EWT
#5	eventTypeRecord [eventTypeParameter] = testFailed status	0x01	ETP1
#6	serviceToRespondToRecord [serviceId] = ReadDTCInformation	0x19	RDTCI
#7	serviceToRespondToRecord [sub-function] = reportNumberOfDTCByStatusMask	0x01	RNDTC
#8	serviceToRespondToRecord [DTCStatusMask] = testFailed status	0x01	DTCSM

The event logic is set up; now it has to be activated.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 119 defines the start of ResponseOnEvent request message flow example #2.

Table 119 — 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	0x86	ROE
#2	eventTypeRecord [eventType] = startResponseOnEvent, storageState = doNotStoreEvent, suppressPosRspMsgIndicationBit = FALSE	0x05	ET_STRTROE
#3	eventWindowTime (will not be evaluated)	0x02	EWT

Table 120 defines the ResponseOnEvent positive response message flow example #2.

Table 120 — 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	0xC6	ROEPR
#2	eventType = onDTCStatusChange	0x05	ET_ODTCSC
#3	numberOfIdentifiedEvents = 0	0x00	NOIE
#4	eventWindowTime	0x02	EWT

In case the specified event occurs the server sends the response message according to the specified serviceToRespondToRecord.

Table 121 defines the ReadDTCInformation positive response message flow example #2.

Table 121 — 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	0x59	RDTCI
#2	DTCStatusAvailibilityMask	0xXX	DTCSAM
#3	DTCCount [DTCCountHighByte] = 0	0x00	DTCCNT_HB
#4	DTCCount [DTCCountLowByte] = 4	0x04	DTCCNT_LB

9.10.5.3.1 Example #2 - Flowcharts

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..#n) and shall have the same service identifier (SI) but might have different content.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Figure 13 depicts the infinite event window – no event during active event window.

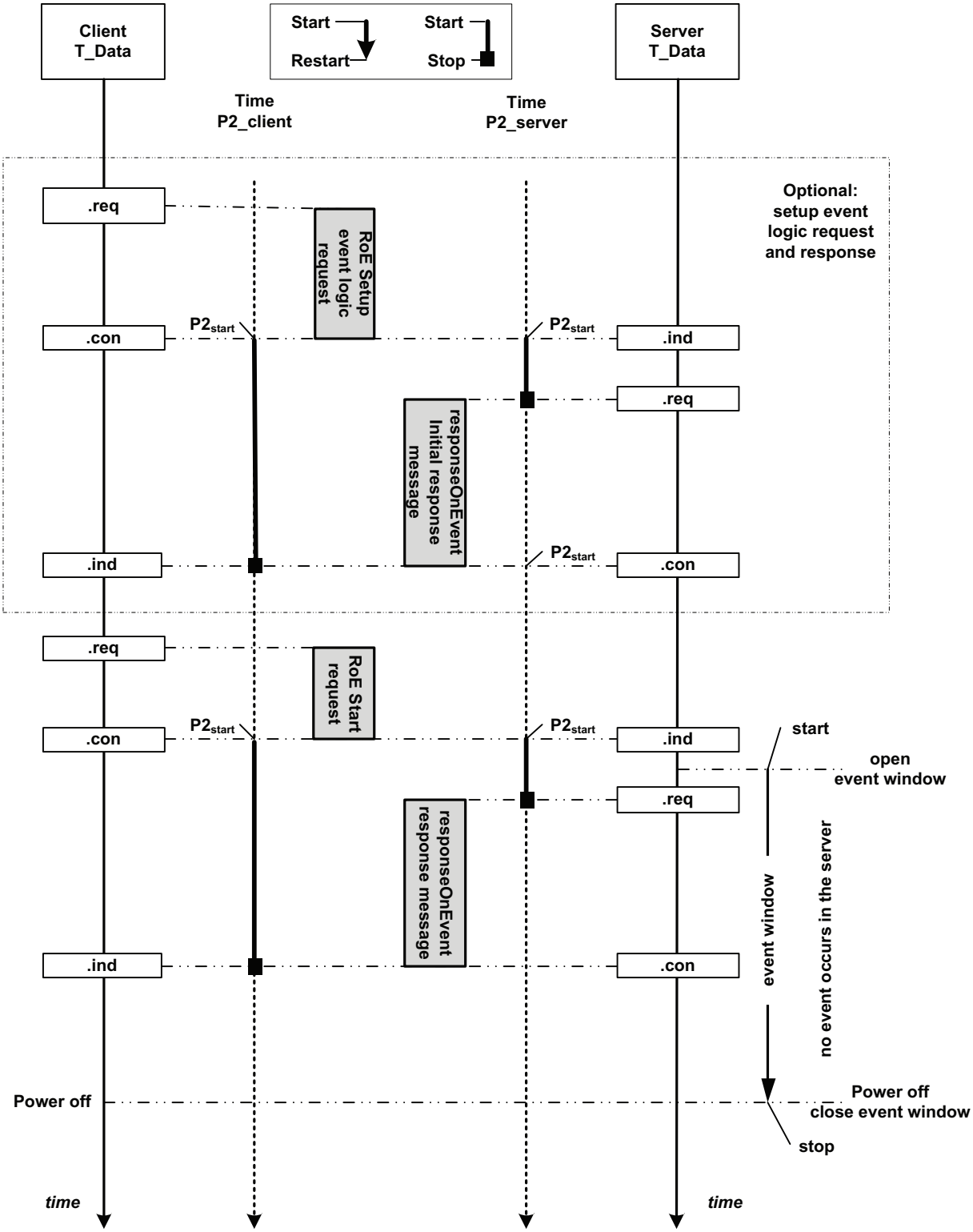


Figure 13 — Infinite event window – no event during active event window

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Figure 14 depicts the infinite event window – multiple events during active event window.

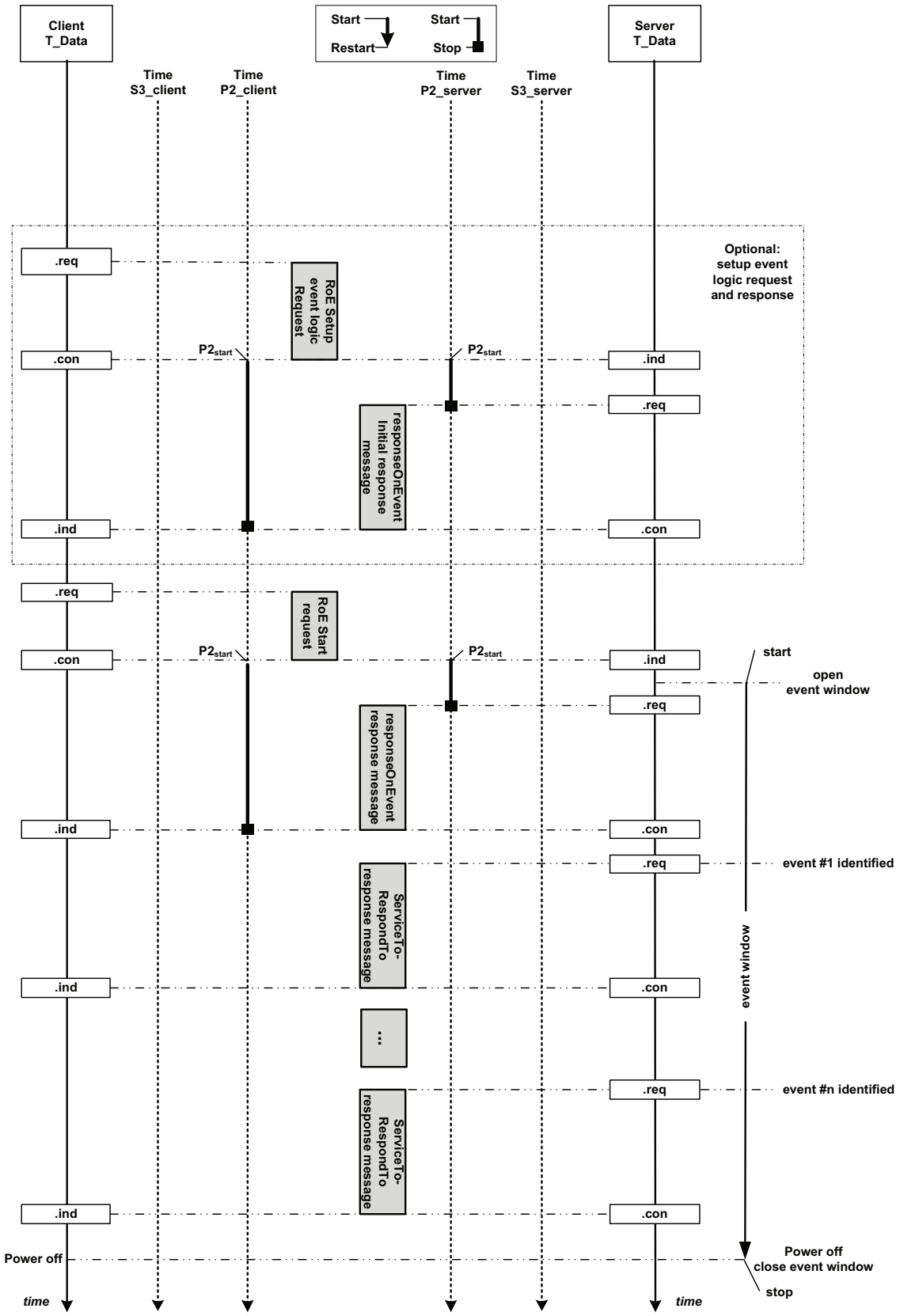


Figure 14 — Infinite event window – multiple events during active event window

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

9.10.5.4 Example #3 - ResponseOnEvent (infinite event window) – sub-function parameter “onComparisonOfValues”

This example only explains the utilisation of sub-function 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 set up 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 0x22 – ReadDataByIdentifier is chosen as the serviceToRespondTo,

— the dataIdentifier 0x0104 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 0x22),

— 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 0x01 – “MV > CP”,

— as hysteresis value 0x0A – 10 % is chosen,

— as eventWindowTime the value 0x02 – “infinite” is chosen,

— as storageState (eventType sub-function bit 6) the value 1 binary – “storeEvent” is chosen,

— in any case a response is requested.

Definition for examples:

— Byte#4&5: dataIdentifier 0x0104

— Byte#6&7: Localisation of reading & 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 1000b 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 1111b Bit#15: Sign entry: 1=signed, 0=unsigned

EXAMPLE 3 Total assignment would be:

— 1011 1100 0101 1000b= 0xBC58 thus byte#6 contains 0xBC, byte#7 contains 0x58

— Byte#8: Comparison operation (operator)

EXAMPLE 4 operator MV > CP = 0x01

— 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 dec = 0x0000 147A,

— byte#9 = 0x00, byte#10 = 0x00, byte#11 = 0x14 and byte#12 = 0x7A

— 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% = 0x0A

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 122 defines the ResponseOnEvent request message example #3.

Table 122 — 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	0x86	ROE
#2	eventTypeRecord [eventType] = onComparisonOfValues, storageState = storeEvent suppressPosRspMsgIndicationBit=FALSE	0x47	ET_OCOV
#3	eventWindowTime = infinite	0x02	EWT
#4	eventTypeRecord [eventTypeParameter#1] = recordDataIdentifier (High Byte)	0x01	ETR_ETP1
#5	eventTypeRecord [eventTypeParameter#2] = recordDataIdentifier (Low Byte)	0x04	ETR_ETP2
#6	eventTypeRecord [eventTypeParameter#3] = Valueinfo#1	0xBC	ETR_ETP3
#7	eventTypeRecord [eventTypeParameter#4] = Valueinfo#2	0x58	ETR_ETP4
#8	eventTypeRecord [eventTypeParameter#5] = Operator	0x01	ETR_ETP5
#9	eventTypeRecord [eventTypeParameter#6] = Comparison Parameter (Byte#4)	0x00	ETR_ETP6
#10	eventTypeRecord [eventTypeParameter#7] = Comparison Parameter (Byte#3)	0x00	ETR_ETP7
#11	eventTypeRecord [eventTypeParameter#8] = Comparison Parameter (Byte#2)	0x14	ETR_ETP8
#12	eventTypeRecord [eventTypeParameter#9] = Comparison Parameter (Byte#1)	0x7A	ETR_ETP9
#13	eventTypeRecord [eventTypeParameter#10] = Hysteresis [%]	0x0A	ETR_ETP10
#14	serviceToRespondToRecord [serviceID] = ReadDataByIdentifier	0x22	RDBI
#15	serviceToRespondToRecord [serviceParameter#1] = dataIdentifier (MSB)	0x01	DID_B1
#16	serviceToRespondToRecord [serviceParameter#2] = dataIdentifier (LSB)	0x04	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 123 defines the ReadDataByIdentifier positive response message example #3.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 123 — 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	0x62	RDBIPR
#2	dataIdentifier [byte#1] (MSB)	0x01	DID_B1
#3	dataIdentifier [byte#2] (LSB)	0x04	DID_B2
#4	dataRecord [data#1]	0xXX	DREC_DATA1
#5	dataRecord [data#2]	0xXX	DREC_DATA2
#6	dataRecord [data#3]	0xXX	DREC_DATA3
#7	dataRecord [data#4]	0xXX	DREC_DATA4
#8	dataRecord [data#5]	0xXX	DREC_DATA5
#9	dataRecord [data#6]	0xXX	DREC_DATA6
#10	dataRecord [data#7]	0xXX	DREC_DATA7
#11	dataRecord [data#8]	0xXX	DREC_DATA8
#12	dataRecord [data#9]	0xXX	DREC_DATA9
#13	dataRecord [data#10]	0xXX	DREC_DATA10
#14	dataRecord [data#11] data content of byte#11: 0x14	0x14	DREC_DATA11
#15	dataRecord [data#12] data content of byte#12: 0x7B	0x7B	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 was 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.

9.11 LinkControl (0x87) service

9.11.1 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.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

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 has to 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 has to 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.

The linkControlType parameter in the request message in conjunction with the conditional linkControlModeIdentifier/linkRecord parameter provides a mechanism to transition with either a pre-defined mode transition parameter or a specifically defined mode transition parameter.

NOTE 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 (0x11) is performed. Once a data link mode transition has taken place, any additional non-defaultSession request(s) shall not cause a re-transition 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 7.5.

9.11.2 Request message

9.11.2.1 Request message definition

Table 124 defines the request message (linkControlType = verifyModeTransitionWithFixedParameter).

Table 124 — Request message definition (linkControlType = verifyModeTransitionWithFixedParameter)

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	LinkControl Request SID	M	0x87	LC
#2	sub-function = [linkControlType]	M	0x01	LEV_LCTP_
#3	linkControlModeIdentifier	M	0x00 – 0xFF	LCMI_

Table 125 defines the request message (linkControlType = verifyModeTransitionWithSpecificParameter).

Table 125 — Request message definition (linkControlType = verifyModeTransitionWithSpecificParameter)

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	LinkControl Request SID	M	0x87	LC
#2	sub-function = [linkControlType]	M	0x02	LEV_LCTP_
	linkRecord[] = [			LBR_
#3	modeParameterHighByte	M	0x00 – 0xFF	MPHB
#4	modeParameterMiddleByte	M	0x00 – 0xFF	MPMB
#5	modeParameterLowByte]	M	0x00 – 0xFF	MPLB

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

 © ISO 2013 – All rights reserved

Table 126 defines the request message (linkControlType = transitionMode).

Table 126 — Request message definition (linkControlType = transitionMode)

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	LinkControl Request SID	M	0x87	LC
#2	sub-function = [linkControlType]	M	0x03	LEV_LCTP_

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

The sub-function 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 127 defines the request message sub-function parameters.

Table 127 — Request message sub-function parameter definition

Bits 6 – 0	Description	Cvt	Mnemonic
0x00	**ISOSAEReserved** This value is reserved by this document.	M	ISOSAERESRVD
0x01	**verifyModeTransitionWithFixedParameter** This parameter is used to verify if a transition with a pre-defined parameter, which is specified by the linkControlModeIdentifier data-parameter can be performed.	U	VMTWFP
0x02	**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
0x03	**transitionMode** This sub-function parameter requests the server(s) to transition the data link into the mode which was requested in the preceding verification message.	U	TM
0x04 – 0x3F	**ISOSAEReserved** This range of values is reserved by this document for future definition.	M	ISOSAERESRVD
0x40 – 0x5F	**vehicleManufacturerSpecific** This range of values is reserved for vehicle manufacturer specific use.	U	VMS
0x60 – 0x7E	**systemSupplierSpecific** This range of values is reserved for system supplier specific use.	U	SSS
0x7F	**ISOSAEReserved** This value is reserved by this document for future definition.	M	ISOSAERESRVD

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

9.11.2.3 Request message data-parameter definition

Table 128 defines the data-parameters of the request message.

Table 128 — 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 sub-function parameter indicates that a specific parameter is used. The format of the linkRecord is specified in the individual data links specific diagnostic specification (UDSonXYZ).

9.11.3 Positive response message

9.11.3.1 Positive response message definition

Table 129 defines the positive response message.

Table 129 — Positive response message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	LinkControl Response SID	M	0xC7	LCPR
#2	linkControlType	M	00-7F	LCTP

9.11.3.2 Positive response message data-parameter definition

Table 130 defines the data-parameter of the positive response message.

Table 130 — Response message data-parameter definition

Definition
linkControlType This parameter is an echo of bits 6 - 0 of the linkControlType sub-function parameter from the request message.

9.11.4 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 131. The listed negative responses shall be used if the error scenario applies to the server.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 131 — Supported negative response codes

NRC	Description	Mnemonic
0x12	**sub-functionNotSupported** This NRC shall be sent if the sub-function parameter is not supported.	SFNS
0x13	**incorrectMessageLengthOrInvalidFormat** This NRC shall be sent if the length of the message is wrong.	IMLOIF
0x22	**conditionsNotCorrect** This NRC shall be returned if the criteria for the requested LinkControl are not met.	CNC
0x24	**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
0x31	**requestOutOfRange** This NRC shall be returned if the requested linkControlModeIdentifier is invalid; the specific modeParameter (linkRecord) is invalid;	ROOR

9.11.5 Message flow example(s) LinkControl

9.11.5.1 Example #1 - Transition baudrate to fixed baudrate (PC baudrate 115200 kBit/s)

9.11.5.1.1 Step#1: Verify if all criteria are met for a baudrate switch

Table 132 defines the LinkControl request message flow example #1 - step #1.

Table 132 — 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	0x87	LC
#2	linkControlType = verifyModeTransitionWithFixedParameter, suppressPosRspMsgIndicationBit = FALSE	0x01	VMTWFP
#3	linkControlModeIdentifier = PC115200Baud	0x05	BI_PC115200

Table 133 defines the LinkControl positive response message flow example #1 - step #1.

Table 133 — 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	0xC7	LCPR
#2	linkControlType = verifyModeTransitionWithFixedParameter	0x01	VMTWFP

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

9.11.5.1.2 Step#2: Transition the baudrate

Table 134 defines the LinkControl request message flow example #1 - step #2.

Table 134 — 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	0x87	LC
#2	linkControlType = transitionMode, suppressPosRspMsgIndicationBit = TRUE	0x83	TM

There is no response from the server(s). The client and the server(s) have to transition the baudrate of their communication link.

9.11.5.2 Example #2 - Transition baudrate to specific baudrate (150kBit/s)

9.11.5.2.1 Step#1: Verify if all criteria are met for a baudrate switch

Table 135 defines the LinkControl request message flow example #2 - step #1.

Table 135 — 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	0x87	LC
#2	linkControlType = verifyModeTransitionWithSpecificParameter, suppressPosRspMsgIndicationBit = FALSE	0x02	VMTWSP
#3	linkRecord [modeParameterHighByte] (150kBit/s)	0x02	MPHB
#4	linkRecord [modeParameterMiddleByte]	0x49	MPMB
#5	linkRecord [modeParameterLowByte]	0xF0	MPLB

Table 136 defines the LinkControl positive response message flow example #2 - step #1.

Table 136 — 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	0xC7	LCPR
#2	linkControlType = verifyModeTransitionWithSpecificParameter	0x02	VMTWSP

9.11.5.2.2 Step#2: Transition the baudrate

Table 137 defines the LinkControl request message flow example #2 - step #2.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 137 — 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	0x87	LC
#2	linkControlType = transitionMode, suppressPosRspMsgIndicationBit = TRUE	0x83	TM

There is no response from the server(s). The client and the server(s) have to transition the baudrate of their communication link.

9.11.5.3 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).

9.11.5.3.1 Step#1: Verify if all criteria are met for a scheduler switch

Table 138 defines the LinkControl request message flow example #3 - step #1.

Table 138 — 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	0x87	LC
#2	linkControlType = verifyModeTransitionWithFixedParameter, suppressPosRspMsgIndicationBit = FALSE	0x01	VMTWFP
#3	linkControlModeIdentifier = ProgrammingSetup	0x20	PROGSU

Table 139 defines the LinkControl positive response message flow example #3 - step #1.

Table 139 — 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	0xC7	LCPR
#2	linkControlType = verifyModeTransitionWithFixedParameter	0x01	VMTWFP

9.11.5.3.2 Step#2: Transition to programming scheduler

Table 140 defines the LinkControl request message flow example #3 - step #2.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 140 — 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	0x87	LC
#2	linkControlType = transitionMode, suppressPosRspMsgIndicationBit = TRUE	0x83	TM

There is no response from the server(s). The client and the server(s) have to transition the cycle design of the FlexRay communication link.

10 Data Transmission functional unit

10.1 Overview

Table 141 defines the Data Transmission functional unit.

Table 141 — 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.
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.

10.2 ReadDataByIdentifier (0x22) service

10.2.1 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

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

 © ISO 2013 – All rights reserved

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 7.5.

10.2.2 Request message

10.2.2.1 Request message definition

Table 142 defines the request message.

Table 142 — Request message definition

A_Data Byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ReadDataByIdentifier Request SID	M	0x22	RDBI
#2 #3	dataIdentifier[]#1 = [byte#1 (MSB) byte#2]	M M	0x00 – 0xFF 0x00 – 0xFF	DID_ HB LB
:	:	:	:	:
#n-1 #n	dataIdentifier[]#m = [byte#1 (MSB) byte#2]	U U	0x00 – 0xFF 0x00 – 0xFF	DID_ HB LB

10.2.2.2 Request message sub-function parameter $Level (LEV_) Definition

This service does not use a sub-function parameter.

10.2.2.3 Request message data-parameter definition

Table 143 defines the data-parameter for the request message.

Table 143 — 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).

10.2.3 Positive response message

10.2.3.1 Positive response message definition

Table 144 defines the positive response message.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 144 — Positive response message definition

A_Data Byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ReadDataByIdentifier Response SID	M	0x62	RDBIPR
#2 #3	dataIdentifier[]#1 = [byte#1 (MSB) byte#2]	M M	0x00 – 0xFF 0x00 – 0xFF	DID_ HB LB
#4 : #(k-1)+4	dataRecord[]#1 = [data#1 : data#k]	M : U	0x00 – 0xFF : 0x00 – 0xFF	DREC_ DATA_1 : DATA_m
:	:	:	:	:
#n-(o-1)-2 #n-(o-1)-1	dataIdentifier[]#m = [byte#1 (MSB) byte#2]	U U	0x00 – 0xFF 0x00 – 0xFF	DID_ HB LB
#n-(o-1) : #n	dataRecord[]#m = [data#1 : data#o]	U : U	0x00 – 0xFF : 0x00 – 0xFF	DREC_ DATA_1 : DATA_k

10.2.3.2 Positive response message data-parameter definition

Table 145 defines the data-parameters of the positive response message.

Table 145 — 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.

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

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 146 — Supported negative response codes

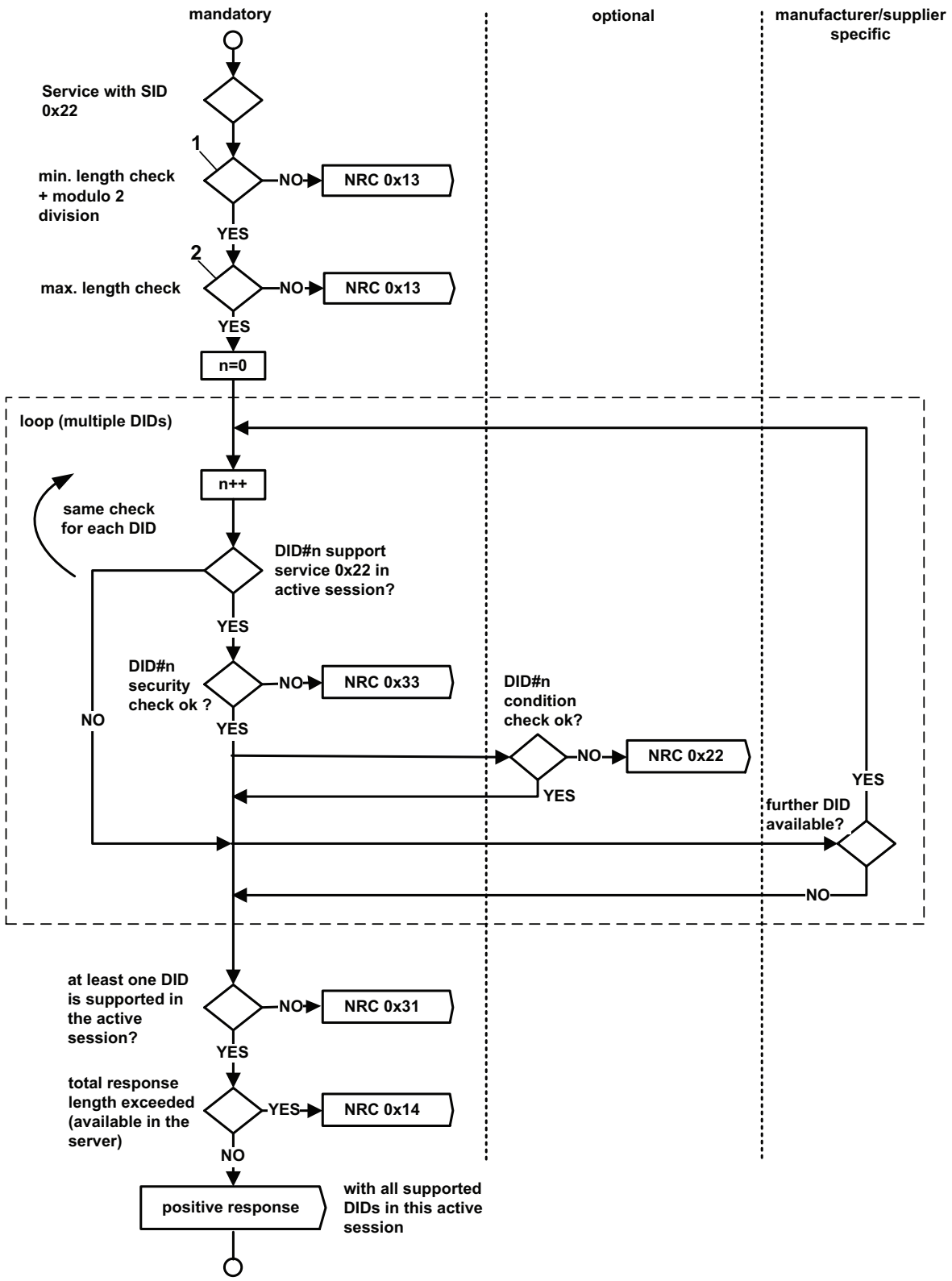
NRC	Description	Mnemonic
0x13	**incorrectMessageLengthOrInvalidFormat** This NRC shall be sent if the length of the request message is invalid or the client exceeded the maximum number of dataIdentifiers allowed to be requested at a time.	IMLOIF
0x14	**responseTooLong** This NRC shall be sent if the 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
0x22	**conditionsNotCorrect** This NRC shall be sent if the operating conditions of the server are not met to perform the required action.	CNC
0x31	**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
0x33	**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

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

The evaluation sequence is documented in Figure 15.



Key

1 minimum length is 3 byte (SI + DID)

2 maximum length is 1 byte (SI) + 2*n bytes (DID(s))

Figure 15 — NRC handling for ReadDataByIdentifier service

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

10.2.5 Message flow example ReadDataByIdentifier

10.2.5.1 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 [6] for further details regarding accepted terms/definitions/acronyms for emission related systems.

The first example reads a single 2 byte dataIdentifier containing a single piece of information (where dataIdentifier 0xF190 contains the VIN number).

The second example demonstrates requesting of multiple dataIdentifiers with a single request (where dataIdentifier 0x010A 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 0x0110 contains battery positive voltage).

10.2.5.2 Example #1: read single dataIdentifier 0xF190 (VIN number)

Table 147 defines the ReadDataByIdentifier request message flow example #1.

Table 147 — 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	0x22	RDBI
#2	dataIdentifier [byte#1] (MSB)	0xF1	DID_B1
#3	dataIdentifier [byte#2]	0x90	DID_B2

Table 148 defines the ReadDataByIdentifier positive response message flow example #1.

Table 148 — 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	0x62	RDBIPR
#2	dataIdentifier [byte#1] (MSB)	0xF1	DID_B1
#3	dataIdentifier [byte#2]	0x90	DID_B2
#4	dataRecord [data#1] = VIN Digit 1 = “W”	0x57	DREC_DATA1
#5	dataRecord [data#2] = VIN Digit 2 = “0”	0x30	DREC_DATA2
#6	dataRecord [data#3] = VIN Digit 3 = “L”	0x4C	DREC_DATA3
#7	dataRecord [data#4] = VIN Digit 4 = “0”	0x30	DREC_DATA4
#8	dataRecord [data#5] = VIN Digit 5 = “0”	0x30	DREC_DATA5
#9	dataRecord [data#6] = VIN Digit 6 = “0”	0x30	DREC_DATA6
#10	dataRecord [data#7] = VIN Digit 7 = “0”	0x30	DREC_DATA7

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 148 — *(continued)*

Message direction	**server → client**		
Message Type	**Response**		
A_Data Byte	**Description (all values are in hexadecimal)**	**Byte Value**	**Mnemonic**
#11	dataRecord [data#8] = VIN Digit 8 = “4”	0x34	DREC_DATA8
#12	dataRecord [data#9] = VIN Digit 9 = “3”	0x33	DREC_DATA9
#13	dataRecord [data#10] = VIN Digit 10 = “M”	0x4D	DREC_DATA10
#14	dataRecord [data#11] = VIN Digit 11 = “B”	0x42	DREC_DATA11
#15	dataRecord [data#12] = VIN Digit 12 = “5”	0x35	DREC_DATA12
#16	dataRecord [data#13] = VIN Digit 13 = “4”	0x34	DREC_DATA13
#17	dataRecord [data#14] = VIN Digit 14 = “1”	0x31	DREC_DATA14
#18	dataRecord [data#15] = VIN Digit 15 = “3”	0x33	DREC_DATA15
#19	dataRecord [data#16] = VIN Digit 16 = “2”	0x32	DREC_DATA16
#20	dataRecord [data#17] = VIN Digit 17 = “6”	0x36	DREC_DATA17

10.2.5.3 Example #2: Read multiple dataIdentifiers 0x010A and 0x0110

Table 149 defines the ReadDataByIdentifier request message flow example #2.

Table 149 — 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	0x22	RDBI
#2	dataIdentifier#1 [byte#1] (MSB)	0x01	DID_B1
#3	dataIdentifier#1 [byte#2]	0x0A	DID_B2
#4	dataIdentifier#2 [byte#1] (MSB)	0x01	DID_B1
#5	dataIdentifier#2 [byte#2]	0x10	DID_B2

Table 150 defines the ReadDataByIdentifier positive response message flow example #2.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 150 — 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	0x62	RDBIPR
#2	dataIdentifier [byte#1] (MSB)	0x01	DID_B1
#3	dataIdentifier [byte#2] (LSB)	0x0A	DID_B2
#4	dataRecord [data#1] = ECT	0xA6	DREC_DATA1
#5	dataRecord [data#2] = TP	0x66	DREC_DATA2
#6	dataRecord [data#3] = RPM	0x07	DREC_DATA3
#7	dataRecord [data#4] = RPM	0x50	DREC_DATA4
#8	dataRecord [data#5] = MAP	0x20	DREC_DATA5
#9	dataRecord [data#6] = MAF	0x1A	DREC_DATA6
#10	dataRecord [data#7] = VSS	0x00	DREC_DATA7
#11	dataRecord [data#8] = BARO	0x63	DREC_DATA8
#12	dataRecord [data#9] = LOAD	0x4A	DREC_DATA9
#13	dataRecord [data#10] = IAC	0x82	DREC_DATA10
#14	dataRecord [data#11] = APP	0x7E	DREC_DATA11
#15	dataIdentifier [byte#1] (MSB)	0x01	DID_B1
#16	dataIdentifier [byte#2] (LSB)	0x10	DID_B2
#17	dataRecord [data#1] = B+	0x8C	DREC_DATA1

10.3 ReadMemoryByAddress (0x23) service

10.3.1 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 0x00 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 7.5.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

10.3.2 Request message

10.3.2.1 Request message definition

Table 151 defines the request message.

Table 151 — Request message definition

A_Data Byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ReadMemoryByAddress Request SID	M	0x23	RMBA
#2	addressAndLengthFormatIdentifier	M	0x00 – 0xFF	ALFID
#3 : #(m-1)+3	memoryAddress[] = [byte#1 (MSB) : byte#m]	M : C1	0x00 – 0xFF : 0x00 – 0xFF	MA_ B1 : Bm
#n-(k-1) : #n	memorySize[] = [byte#1 (MSB) : byte#k]	M : C2	0x00 – 0xFF : 0x00 – 0xFF	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.				

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

This service does not use a sub-function parameter.

10.3.2.3 Request message data-parameter definition

Table 152 defines the data-parameters of the request message.

Table 152 — Request message data-parameter definition

Definition
addressAndLengthFormatIdentifier This parameter is a one byte value with each nibble encoded separately (see 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.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

10.3.3 Positive response message

10.3.3.1 Positive response message definition

Table 153 defines the positive response message.

Table 153 — Positive response message definition

A_Data Byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ReadMemoryByAddress Response SID	M	0x63	RMBAPR
#2 : #n	dataRecord[] = [data#1 : data#m]	M : U	0x00 – 0xFF : 0x00 – 0xFF	DREC_ DATA_1 : DATA_m

10.3.3.2 Positive response message data-parameter definition

Table 154 defines the data parameter of the positive response message.

Table 154 — 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..

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

Table 155 — Supported negative response codes

NRC	Description	Mnemonic
0x13	**incorrectMessageLengthOrInvalidFormat** This NRC shall be sent if the length of the message is wrong.	IMLOIF
0x22	**conditionsNotCorrect** This NRC shall be sent if the operating conditions of the server are not met to perform the required action.	CNC
0x31	**requestOutOfRange** This NRC shall be sent if: — Any memory address within the interval [0xMA, (0xMA + 0xMS -0x1)] is invalid; — Any memory address within the interval [0xMA, (0xMA + 0xMS -0x1)] 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

SHW3SZH 2013-09-29 NormMaster

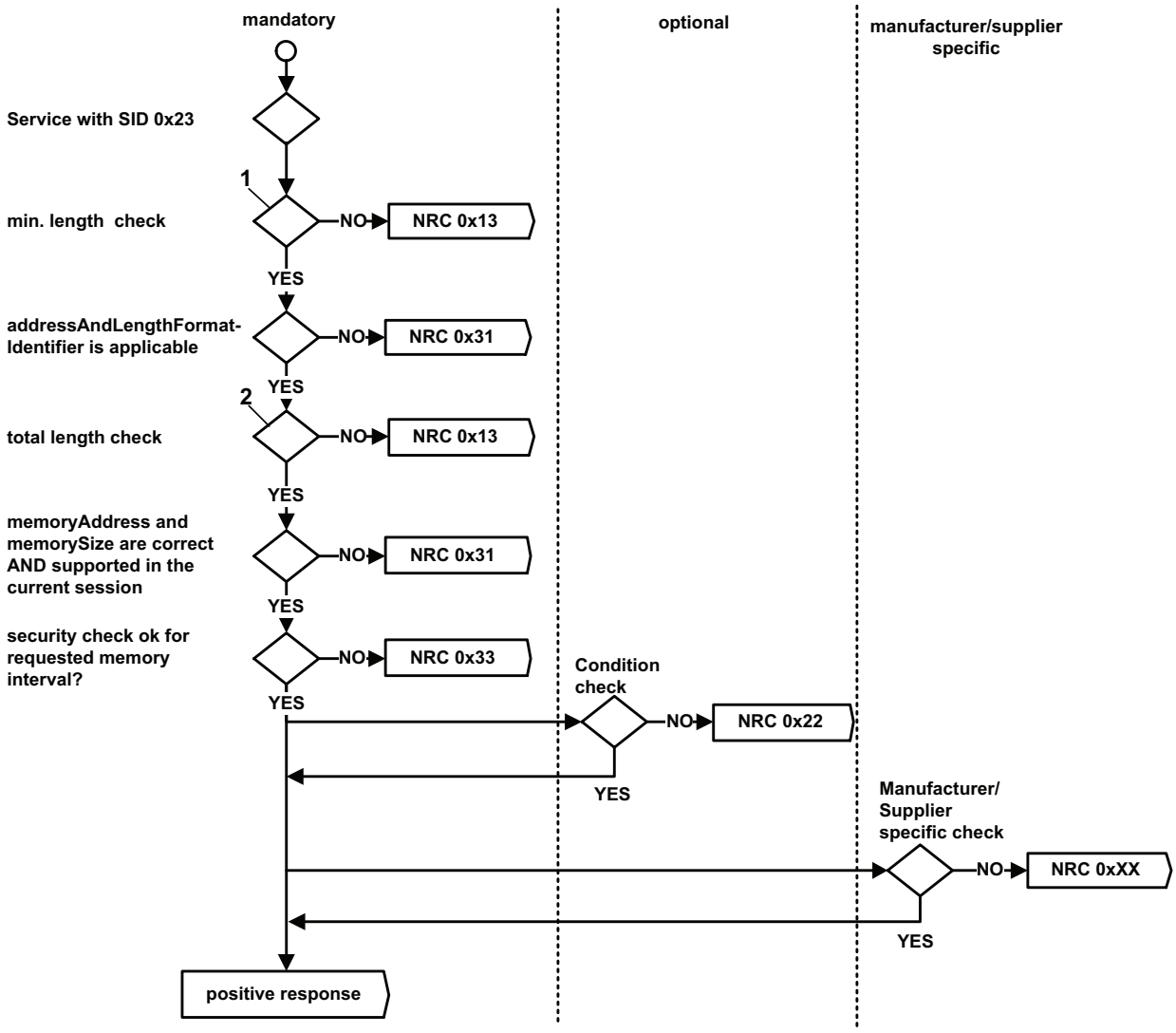
Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 155 — *(continued)*

0x33	**SecurityAccessDenied** This NRC shall be sent if any memory address within the interval [0xMA, (0xMA + 0xMS - 0x1)] is secure and the server is locked.	SAD

The evaluation sequence is documented in Figure 16.



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 16 — NRC handling for ReadMemoryByAddress service

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

10.3.5 Message flow example ReadMemoryByAddress

10.3.5.1 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.

10.3.5.2 Example #1: ReadMemoryByAddress - 4-byte (32-bit) addressing

The client reads 259 data bytes from the server's memory starting at memory address 0x2048 1392.

Table 156 defines the ReadMemoryByAddress request message flow example #1.

Table 156 — 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	0x23	RMBA
#2	addressAndLengthFormatIdentifier	0x24	ALFID
#3	memoryAddress [byte#1] (MSB)	0x20	MA_B1
#4	memoryAddress [byte#2]	0x48	MA_B2
#5	memoryAddress [byte#3]	0x13	MA_B3
#6	memoryAddress [byte#4]	0x92	MA_B4
#7	memorySize [byte#1] (MSB)	0x01	MS_B1
#8	memorySize [byte#2]	0x03	MS_B2

Table 157 defines the ReadMemoryByAddress positive response message flow example #1.

Table 157 — 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	0x63	RMBAPR
#2	dataRecord [data#1] (memory cell#1)	0x00	DREC_DATA_1
:	:	:	:
#259+1	dataRecord [data#259] (memory cell#259)	0x8C	DREC_DATA_259

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

10.3.5.3 Example #2: ReadMemoryByAddress - 2-byte (16-bit) addressing.

The client reads five data bytes from the server's memory starting at memory address 0x4813.

Table 158 defines the ReadMemoryByAddress request message flow example #2.

Table 158 — 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	0x23	RMBA
#2	addressAndLengthFormatIdentifier	0x12	ALFID
#3	memoryAddress [byte#1 (MSB)]	0x48	MA_B1
#4	memoryAddress [byte#2 (LSB)]	0x13	MA_B2
#5	memorySize [byte#1]	0x05	MS_B1

Table 159 defines the ReadMemoryByAddress positive response message flow example #2.

Table 159 — 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	0x63	RMBAPR
#2	dataRecord [data#1] (memory cell#1)	0x43	DREC_DATA_1
#3	dataRecord [data#2] (memory cell#2)	0x2A	DREC_DATA_2
#4	dataRecord [data#3] (memory cell#3)	0x07	DREC_DATA_3
#5	dataRecord [data#4] (memory cell#4)	0x2A	DREC_DATA_4
#6	dataRecord [data#5] (memory cell#5)	0x55	DREC_DATA_5

10.3.5.4 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 0x204813.

Table 160 defines the ReadMemoryByAddress request message flow example #3.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 160 — 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	0x23	RMBA
#2	addressAndLengthFormatIdentifier	0x23	ALFID
#3	memoryAddress [byte#1 (MSB)]	0x20	MA_B1
#4	memoryAddress [byte#2]	0x48	MA_B2
#5	memoryAddress [byte#3 (LSB)]	0x13	MA_B3
#6	memorySize [byte#1 (MSB)]	0x00	MS_B1
#7	memorySize [byte#2 (LSB)]	0x03	MS_B2

Table 161 defines the ReadMemoryByAddress first positive response message, example #3.

Table 161 — 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	0x63	RMBAPR
#2	dataRecord [data#1] (memory cell#1)	0x00	DREC_DATA_1
#3	dataRecord [data#2] (memory cell#2)	0x01	DREC_DATA_2
#4	dataRecord [data#3] (memory cell#3)	0x8C	DREC_DATA_3

10.4 ReadScalingDataByIdentifier (0x24) service

10.4.1 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 7.5.

10.4.2 Request message

10.4.2.1 Request message definition

Table 162 defines the request message.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 162 — Request message definition

A_Data Byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ReadScalingDataByIdentifier Request SID	M	0x24	RSDBI
	dataIdentifier[] = [			DID_
#2	byte#1 (MSB)	M	0x00 – 0xFF	HB
#3	byte#2]	M	0x00 – 0xFF	LB

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

This service does not use a sub-function parameter.

10.4.2.3 Request message data-parameter definition

Table 163 defines the data-parameter of the request message.

Table 163 — 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).

10.4.3 Positive response message

10.4.3.1 Positive response message definition

Table 164 defines the positive response message.

Table 164 — Positive response message definition

A_Data Byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ReadScalingDataByIdentifier Response SID	M	0x64	RSDBIPR
	dataIdentifier[] = [			DID_
#2	byte#1 (MSB)	M	0x00 – 0xFF	HB
#3	byte#2 (LSB)]	M	0x00 – 0xFF	LB
#4	scalingByte#1	M	0x00 – 0xFF	SB_1
	scalingByteExtension[]#1 = [			SBE_
#5	scalingByteExtensionParameter#1	C1	0x00 – 0xFF	PAR1
:	:	:	:	:
#(p-1)+5	scalingByteExtensionParameter#p]	C1	0x00 – 0xFF	PARp
:	:	:	:	:
#n-r	scalingByte#k	C2	0x00 – 0xFF	SB_k
	scalingByteExtension[]#k = [			SBE_
#n-(r-1)	scalingByteExtensionParameter#1	C1	0x00 – 0xFF	PAR1
:	:	:	:	:
#n	scalingByteExtensionParameter#r]	C1	0x00 – 0xFF	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.				

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

10.4.3.2 Positive response message data-parameter definition

Table 165 defines the data-parameters of the positive response message.

Table 165 — 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).

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

Table 166 — Supported negative response codes

NRC	Description	Mnemonic
0x13	**incorrectMessageLengthOrInvalidFormat** This NRC shall be sent if the length of the request message is invalid.	IMLOIF
0x22	**conditionsNotCorrect** This NRC shall be sent if the operating conditions of the server are not met to perform the required action.	CNC
0x31	**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
0x33	**securityAccessDenied** This NRC shall be sent if the dataIdentifier is secured and the server is not in an unlocked state.	SAD

The evaluation sequence is documented in Figure 17.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

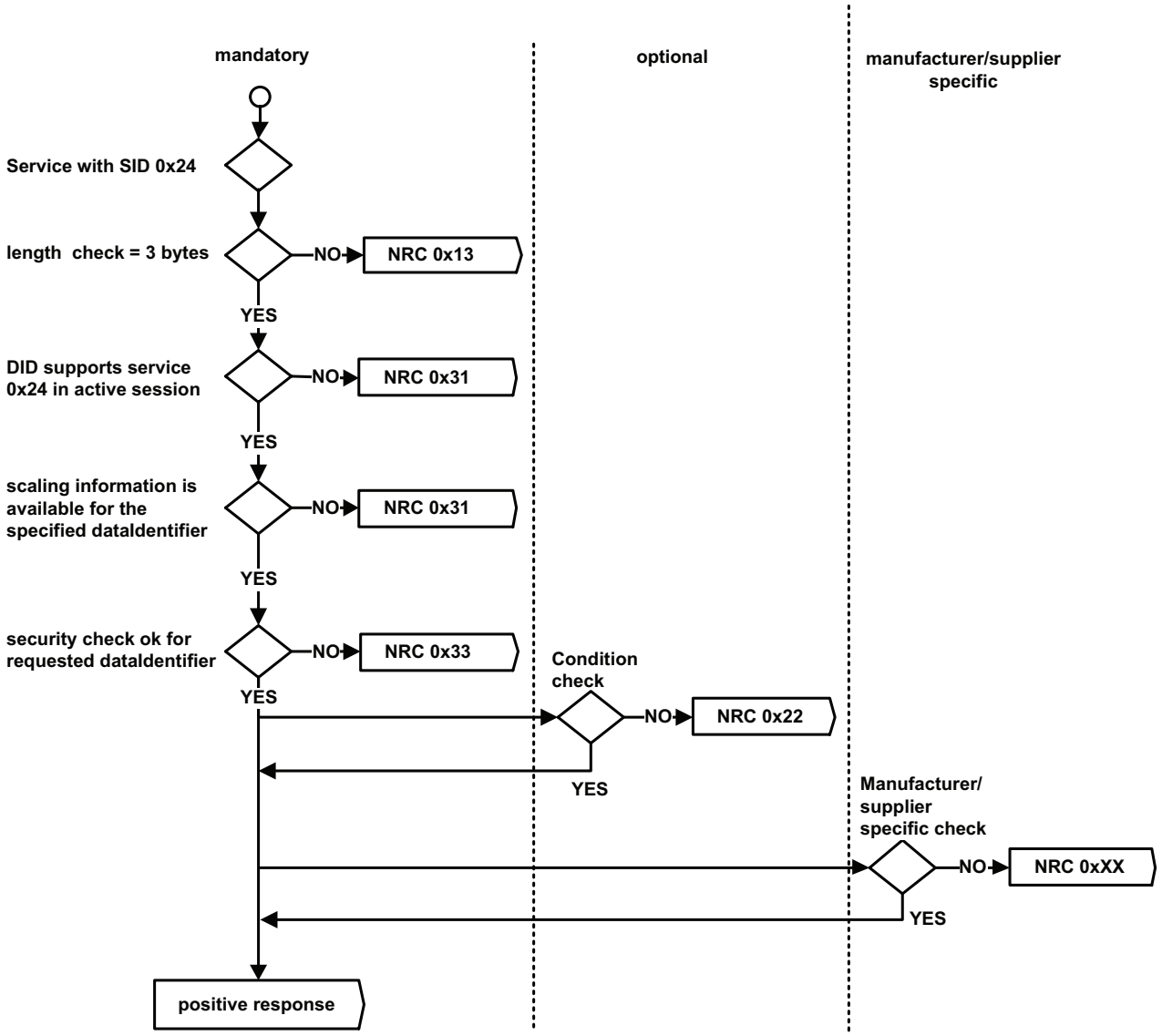


Figure 17 — NRC handling for ReadScalingDataByIdentifier service

10.4.5 Message flow example ReadScalingDataByIdentifier

10.4.5.1 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 0xF190, 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.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

10.4.5.2 Example #1: readScalingDataByIdentifier wth dataIdentifier 0xF190 (VIN number)

Table 167 defines the ReadScalingDataByIdentifier request message flow example #1.

Table 167 — 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	0x24	RSDBI
#2	dataIdentifier [byte#1] (MSB)	0xF1	DID_B1
#3	dataIdentifier [byte#2] (LSB)	0x90	DID_B2

Table 168 defines the ReadScalingDataByIdentifier positive response message flow example #1.

Table 168 — 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	0x64	RSDBIPR
#2	dataIdentifier [byte#1] (MSB)	0xF1	DID_B1
#3	dataIdentifier [byte#2] (LSB)	0x90	DID_B2
#4	scalingByte#1 {ASCII, 15 data bytes}	0x6F	SB_1
#5	scalingByte#2 {ASCII, 2 data bytes}	0x62	SB_2

10.4.5.3 Example #2: readScalingDataByIdentifier wth dataIdentifier 0x0105 (Vehicle Speed)

Table 169 defines the ReadScalingDataByIdentifier request message flow example #2.

Table 169 — 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	0x24	RSDBI
#2	dataIdentifier [byte#1] (MSB)	0x01	DID_B1
#3	dataIdentifier [byte#2] (LSB)	0x05	DID_B2

Table 170 defines the ReadScalingDataByIdentifier positive response message flow example #2.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 170 — 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	0x64	RSDBIPR
#2	dataIdentifier [byte#1] (MSB)	0x01	DID_B1
#3	dataIdentifier [byte#2] (LSB)	0x05	DID_B2
#4	scalingByte#1 {unsigned numeric, 1 data byte}	0x01	SBYT_1
#5	scalingByte#2 {formula, 5 data bytes}	0x95	SB_2
#6	scalingByteExtension#2 [byte#1] {formulaIdentifier = C0 * x + C1}	0x00	SBE_21
#7	scalingByteExtension#2 [byte#2] {C0 high byte}	0xE0	SBE_22
#8	scalingByteExtension#2 [byte#3] {C0 low byte} [C0 = 75 * 10P-2P]	0x4B	SBE_23
#9	scalingByteExtension#2 [byte#4] {C1 high byte}	0x00	SBE_24
#10	scalingByteExtension#2 [byte#5] {C1 low byte} [C1 = 30 * 10P0P]	0x1E	SBE_25
#11	scalingByte#3 {unit / format, 1 data byte}	0xA1	SB_3
#12	scalingByteExtension#3 [byte#1] {unit ID, km/h}	0x30	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 0x0105.

10.4.5.4 Example #3: readScalingDataByIdentifier wth dataIdentifier 0x0967

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 (0x0967) is defined in Table 171.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 171 — Example data definition

Data Byte	Bit(s)	Description
#1	7-4	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
	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 172 defines the ReadScalingDataByIdentifier request message flow example #3.

Table 172 — 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	0x24	RSDBI
#2	dataIdentifier [byte#1] (MSB)	0x09	DID_B1
#3	dataIdentifier [byte#2] (LSB)	0x67	DID_B2

Table 173 defines the ReadScalingDataByIdentifier positive response message flow example #3.

Table 173 — 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	0x64	RSDBIPR
#2	dataIdentifier [byte#1] (MSB)	0x09	DID_HB
#3	dataIdentifier [byte#2] (LSB)	0x67	DID_LB
#4	scalingByte#1 {bitMappedReportedWithOutMask, 2 data bytes}	0x22	SBYT_1
#5	scalingByteExtension#1 [byte#1] {dataRecord#1 Validity Mask}	0x03	SBYE_11
#6	scalingByteExtension#1 [byte#2] {dataRecord#2 Validity Mask}	0x43	SBYE_12

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

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.

10.5 ReadDataByPeriodicIdentifier (0x2A) service

10.5.1 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 (0xF2XX, see C.1 for allowed periodicDataIdentifier values), e.g. the periodicDataIdentifier 0xE3 used in this service is the dataIdentifier 0xF2E3.

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:

— the call rate of the periodic scheduler,

— the number of available protocol specific periodic data response message address information IDs allocated per scheduler call (e.g., CAN identifier on CAN)

— 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.

NOTE Note the periodic rate is an integer multiple of the periodic scheduler call rate.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

 © ISO 2013 – All rights reserved

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 10ms, 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 10.6.5.

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 7.5.

10.5.2 Request message

10.5.2.1 Request message definition

Table 174 defines the request message.

Table 174 — Request message definition

A_Data Byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ReadDataByPeriodicIdentifier Request SID	M	0x2A	RDBPI
#2	transmissionMode	M	0x00 – 0xFF	TM
#3	periodicDataIdentifier[]#1	C	0x00 – 0xFF	PDID1
:	:	:	:	:
#m+2	periodicDataIdentifier[]#m	U	0x00 – 0xFF	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.				

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

This service does not use a sub-function parameter.

10.5.2.3 Request message data-parameter definition

Table 175 defines the data-parameters of the request message.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 175 — 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.

10.5.3 Positive response message

10.5.3.1 Positive response message definition

It has to be distinguished 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 176 defines the initial positive response message to be transmitted by the server when it accepts the request.

Table 176 — Positive response message definition

A_Data Byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ReadDataByPeriodicIdentifier Response SID	M	0x6A	RDBPIPR

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 177 defines the periodic data response message data definition.

Table 177 — Periodic data response message data definition

A_Data Byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	periodicDataIdentifier	M	0x00 – 0xFF	PDID
#2 : #k+2	dataRecord[] = [data#1 : data#k]	M : U	0x00 – 0xFF : 0x00 – 0xFF	DREC_ DATA_1 : DATA_k

10.5.3.2 Positive response message data-parameter definition

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

Table 178 defines the periodic message data-parameters of the defined periodic data response message.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 178 — 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.

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

Table 179 — Supported negative response codes

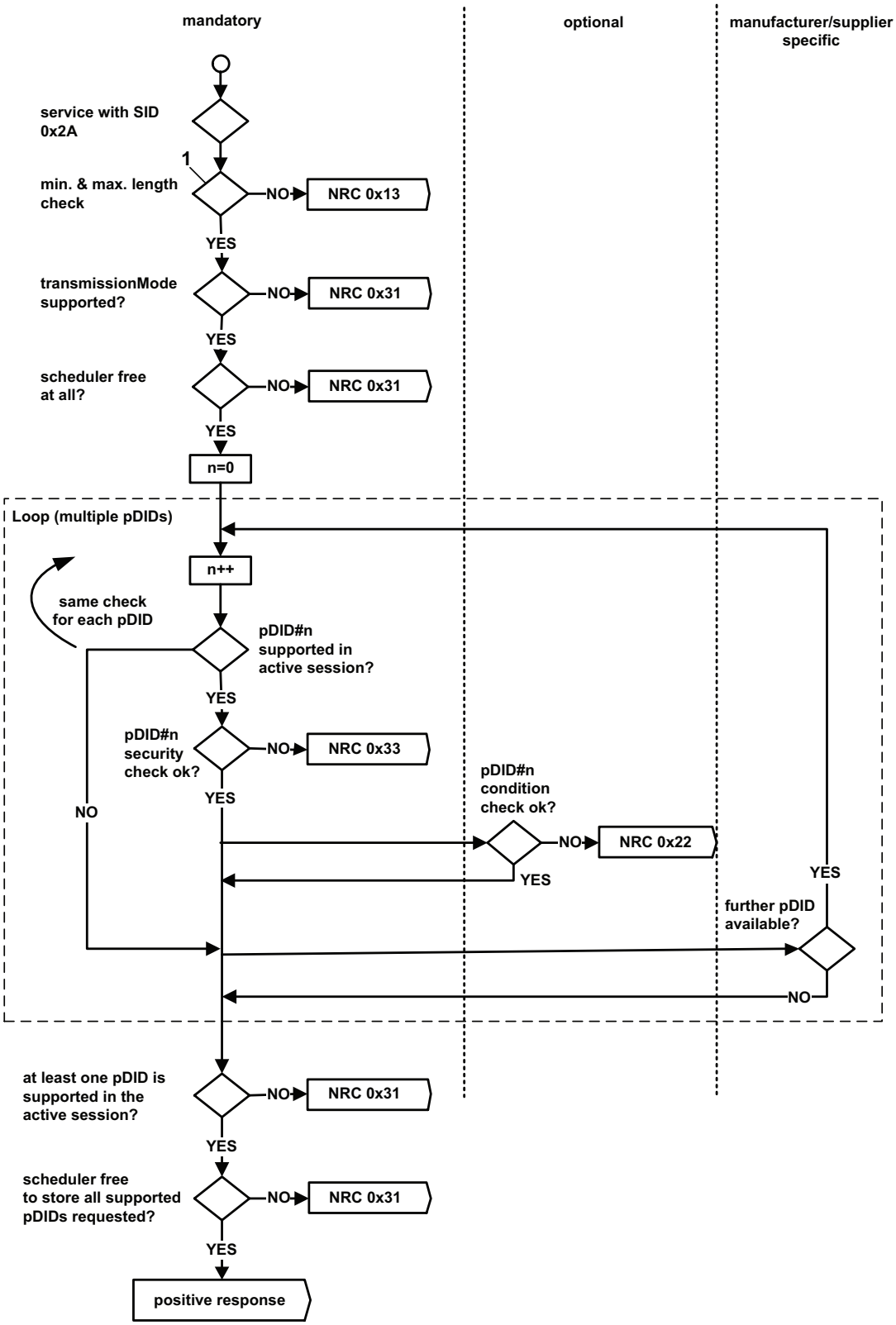
NRC	Description	Mnemonic
0x13	**incorrectMessageLengthOrInvalidFormat** This NRC shall be sent if the length of the request message is invalid or the client exceeded the maximum number of periodicDataIdentifiers allowed to be requested at a time.	IMLOIF
0x22	**conditionsNotCorrect** This NRC shall be sent if the operating conditions of the server are not met to perform the required action. E.g. this could occur if the client requests periodicDataIdentifiers with different transmissionModes and the server does not support multiple transmissionModes simultaneously.	CNC
0x31	**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
0x33	**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

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

The evaluation sequence is documented in Figure 18.



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) + 1byte (TM) + n bytes (pDID(s))

Figure 18 — NRC handling for ReadDataByPeriodicIdentifier service

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

10.5.5 Message flow example ReadDataByPeriodicIdentifier

10.5.5.1 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 [6] for further details regarding accepted terms/definitions/acronyms for emission related systems.

10.5.5.2 Example #1 - Read multiple periodicDataIdentifiers 0xE3 and 0x24 at medium rate

10.5.5.2.1 Assumptions

The example demonstrates requesting of multiple dataIdentifiers with a single request (where periodicDataIdentifier 0xE3 (= dataIdentifier 0xF2E3) contains engine coolant temperature, throttle position, engine speed, vehicle speed sensor, and periodicDataIdentifier 0x24 (= dataIdentifier 0xF224) 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 0xE3 only.

10.5.5.2.2 Step #1: Request periodic transmission of the periodicDataIdentifiers

Table 180 defines the ReadDataByPeriodicIdentifier request message flow example – step #1.

Table 180 — 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	0x2A	RDBPI
#2	transmissionMode = sendAtMediumRate	0x02	TM_SAMR
#3	periodicDataIdentifier#1	0xE3	PDID1
#4	periodicDataIdentifier#2	0x24	PDID2

Table 181 defines the ReadDataByPeriodicIdentifier initial positive response message flow example – step #1.

Table 181 — 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	0x6A	RDBPIPR

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 182 defines the ReadDataByPeriodicIdentifier sub-sequent positive response message #1 flows – step #1.

Table 182 — ReadDataByPeriodicIdentifier sub-sequent 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	0xE3	PDID1
#2	dataRecord [data#1] = ECT	0xA6	DREC_DATA_1
#3	dataRecord [data#2] = TP	0x66	DREC_DATA_2
#4	dataRecord [data#3] = RPM	0x07	DREC_DATA_3
#5	dataRecord [data#4] = RPM	0x50	DREC_DATA_4
#6	dataRecord [data#5] = VSS	0x00	DREC_DATA_5

Table 183 defines the ReadDataByPeriodicIdentifier sub-sequent positive response message #2 flows – step #1.

Table 183 — ReadDataByPeriodicIdentifier sub-sequent 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#2	0x24	PDID2
#2	dataRecord [data#1] = B+	0x8C	DREC_DATA_1
#3	dataRecord [data#2] = MAP	0x20	DREC_DATA_2
#4	dataRecord [data#3] = MAF	0x1A	DREC_DATA_3
#5	dataRecord [data#4] = BARO	0x63	DREC_DATA_4
#6	dataRecord [data#5] = LOAD	0x4A	DREC_DATA_5

The server transmits the above shown sub-sequent response messages at the medium rate as applicable to the server.

10.5.5.2.3 Step #2: Stop the transmission of the periodicDataIdentifiers

Table 184 defines the ReadDataByIdentifier request message flow example – step #2.

Table 184 — 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	0x2A	RDBPI
#2	transmissionMode = stopSending	0x04	TM_SS
#3	periodicDataIdentifier#1	0xE3	PDID

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 185 defines the ReadDataByIdentifier positive response message flow example – step #2.

Table 185 — 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	0x6A	RDBPIPR

The server stops the transmission of the periodicDataIdentifier 0xE3 only. The periodicDataIdentifier 0x24 is still transmitted at the server medium rate.

10.5.5.3 Example #2 - Graphical and tabular example of ReadDataByPeriodicIdentifier service periodic schedule rates

10.5.5.3.1 ReadDataByPeriodicIdentifier example overview

This subclause contains an example of scheduled periodic data, with both a graphical and tabular example of the ReadDataByPeriodicIdentifier (0x2A) 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.

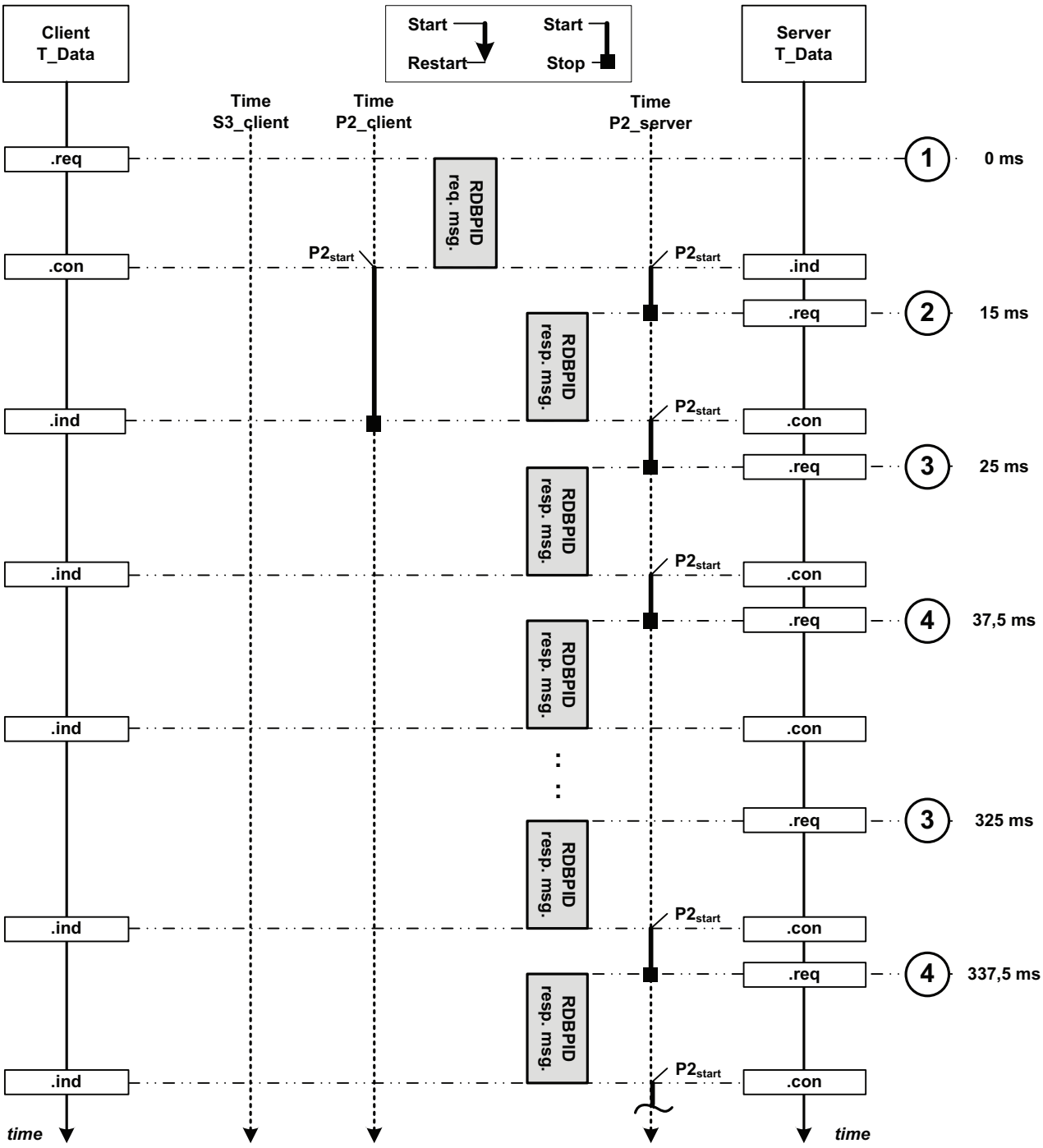
SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

10.5.5.3.2 Example #2 – Read multiple periodicDataIdentifiers 0xE3 and 0x24 at medium rate

At t = 0,0 ms, the client begins sending the request to schedule 2 periodicDataIdentifiers (0xF2E3 and 0xF224) 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.



Key

1 ReadDataByPeriodicIdentifier (0x2A, 0x02, 0xF2E3, 0xF224) request message (sendAtMediumRate)
2 ReadDataByPeriodicIdentifier positive response message (0x6A, no data included)
3 ReadDataByPeriodicIdentifier periodic data response message (0xE3, 0xXX, …, 0xXX)
4 ReadDataByPeriodicIdentifier periodic data response message (0x24, 0xXX, …, 0xXX)

Figure 19 — Example #2 – periodicDataIdentifiers scheduled at medium rate

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 186 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 186 — Example #2: Periodic scheduler table

time t ms	periodic scheduler transmit Index	periodic identifier sent	periodic scheduler loop #	scheduler[0] transmit count	scheduler[1] transmit count
25,0	0	0xE3	1	0->24	0
37,5	1	0x24	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	0xE3	25	0->24	1
337,5	1	0x24	26	23	0->24
350,0	0	None	27	22	23
362,5	0	None	28	21	22

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

10.5.5.4 Example #3 - Graphical and tabular example of ReadDataByPeriodicIdentifier service periodic schedule rates

10.5.5.4.1 ReadDataByPeriodicIdentifier example overview

This subclause contains an example of scheduled periodic data with both a graphical and tabular example of the ReadDataByPeriodicIdentifier (0x2A) service.

The example is based on the example given in 10.5.5.3. 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.

10.5.5.4.2 Read multiple periodicDataIdentifiers at different periodic rates

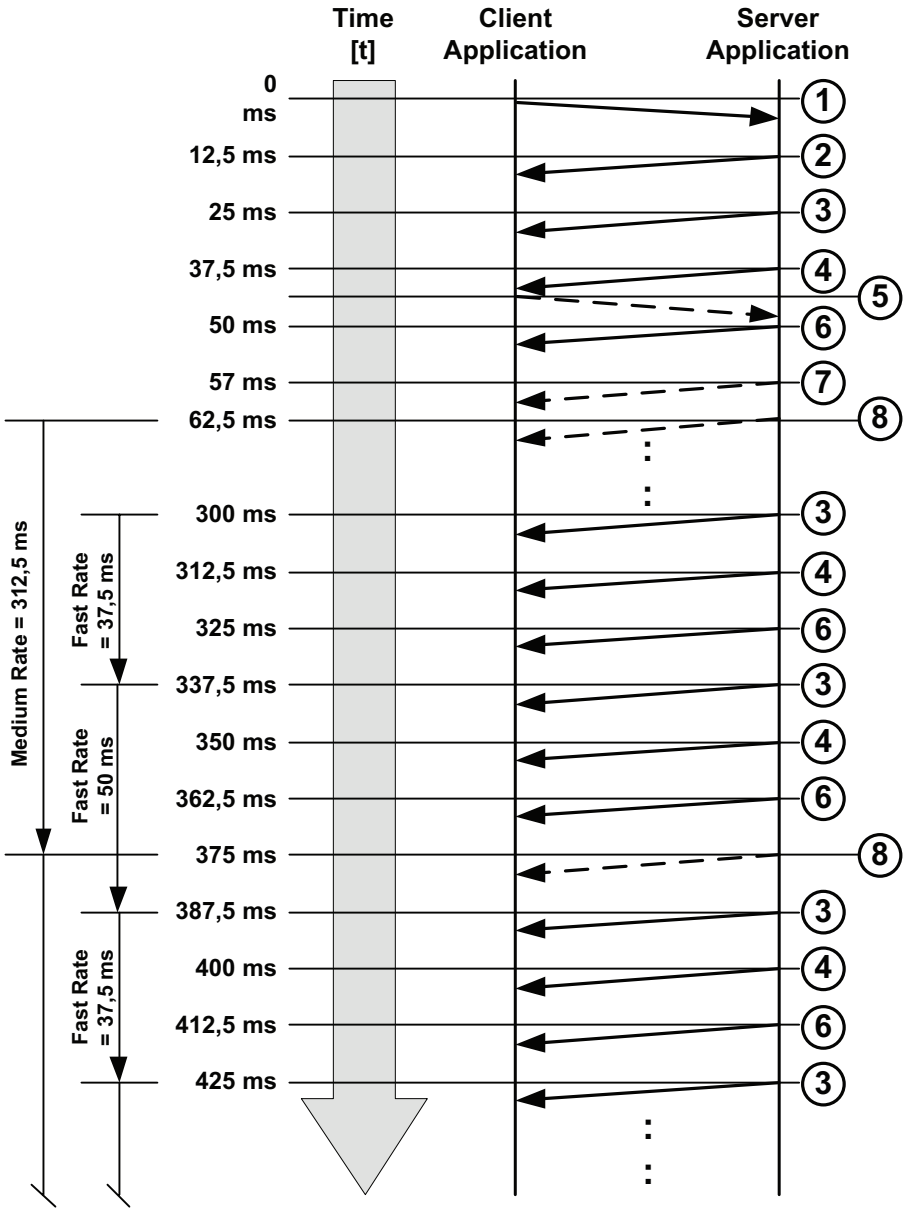
In this example, three periodicDataIdentifiers (for simplicity 0x01, 0x02, 0x03) are scheduled at the fast periodic rate (25 ms) and then another request is sent for a single periodicDataIdentifier (0x04) to be scheduled at the medium periodic rate (300 ms). For the purposes of this example, the server 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 20 depicts the example #3 – periodicDataIdentifiers scheduled at fast and medium rate.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

 © ISO 2013 – All rights reserved



Key

1 ReadDataByPeriodicIdentifier (0x2A, 0x03, 0xF201, 0xF202, 0xF203) request message (sendAtFastRate)
2 ReadDataByPeriodicIdentifier positive response message (0x6A, no data included)
3 ReadDataByPeriodicIdentifier periodic data response message (0x01, 0xXX, …, 0xXX)
4 ReadDataByPeriodicIdentifier periodic data response message (0x02, 0xXX, …, 0xXX)
5 ReadDataByPeriodicIdentifier (0x2A, 0x02, 0xF204) request message (sendAtMediumRate)
6 ReadDataByPeriodicIdentifier periodic data response message (0x03, 0xXX, …, 0xXX)
7 ReadDataByPeriodicIdentifier positive response message (0x6A, no data included)
8 ReadDataByPeriodicIdentifier periodic data response message (04, 0xXX, …, 0xXX)

Figure 20 — Example #3 – periodicDataIdentifiers scheduled at fast and medium rate

Table 187 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.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 187 — Example #3: Periodic scheduler table

time t ms	periodic scheduler transmit index	periodic identifier sent	periodic scheduler loop #	scheduler[0] transmit count	scheduler[1] transmit count	scheduler[2] transmit count	scheduler[3] transmit count
25,0	0	0x01	1	0->2	0	0	N/A
37,5	1	0x02	2	1	0->2	0	N/A
50,0	2	0x03	3	0	1	0->2	0
62,5	3	0x04	4	0	0	1	0->24
75,0	0	0x01	5	0->2	0	0	23
87,5	1	0x02	6	1	0->2	0	22
100,0	2	0x03	7	0	1	0->2	21
112,5	3	0x01	8	0->2	0	1	20
125,0	1	0x02	9	1	0->2	0	19
137,5	2	0x03	10	0	1	0->2	18
150,0	3	0x01	11	0->2	0	1	17
162,5	1	0x02	12	1	0->2	0	16
175,0	2	0x03	13	0	1	0->2	15
187,5	3	0x01	14	0->2	0	1	14
200,0	1	0x02	15	1	0->2	0	13
212,5	2	0x03	16	0	1	0->2	12
225,0	3	0x01	17	0->2	0	1	11
237,5	1	0x02	18	1	0->2	0	10
250,0	2	0x03	19	0	1	0->2	9
262,5	3	0x01	20	0->2	0	1	8
275,0	1	0x02	21	1	0->2	0	7
287,5	2	0x03	22	0	1	0->2	6
300,0	3	0x01	23	0->2	0	1	5
312,5	1	0x02	24	1	0->2	0	4
325,0	2	0x03	25	0	1	0->2	3
337,5	3	0x01	26	0->2	0	1	2
350,0	1	0x02	27	1	0->2	0	1
362,5	2	0x03	28	0	1	0->2	0
375,0	3	0x04	29	0	0	1	0->24
387,5	0	0x01	30	0->2	0	0	23

10.5.5.5 Example #4 - Tabular example of ReadDataByPeriodicIdentifier service periodic schedule rates

10.5.5.5.1 ReadDataByPeriodicIdentifier example overview

This subclause contains an example of scheduled periodic data with a tabular example of the ReadDataByPeriodicIdentifier (0x2A) service. The example contains a table which shows a possible

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

 © ISO 2013 – All rights reserved

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 0x01, 0x02) 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 188 — Example #4: Periodic scheduler table

Time t ms	Response message ID#	Periodic identifier sent	Periodic scheduler loop #
10	1	0x01	1
10	2	0x02	1
20	1	0x01	2
20	2	0x02	2
30	1	0x01	3
30	2	0x02	3
40	1	0x01	4
40	2	0x02	4
50	1	0x01	5
50	2	0x02	5
60	1	0x01	6
60	2	0x02	6
70	1	0x01	7
70	2	0x02	7
80	1	0x01	8
80	2	0x02	8
90	1	0x01	9
90	2	0x02	9
100	1	0x01	10
100	2	0x02	10

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

10.5.5.6 Example #5 - Tabular example of ReadDataByPeriodicIdentifier service periodic schedule rates

10.5.5.6.1 ReadDataByPeriodicIdentifier example overview

This subclause uses the same assumptions as 10.5.5.5.1. 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 0x01, 0x02, 0x03) 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 189 — Example #5: Periodic scheduler table

Time t ms	Response message ID#	Periodic identifier sent	Periodic scheduler loop #
10	1	0x01	1
10	2	0x02	1
20	1	0x03	2
20	2	0x01	2
30	1	0x02	3
30	2	0x03	3
40	1	0x01	4
40	2	0x02	4
50	1	0x03	5
50	2	0x01	5
60	1	0x02	6
60	2	0x03	6
70	1	0x01	7
70	2	0x02	7
80	1	0x03	8
80	2	0x01	8
90	1	0x02	9
90	2	0x03	9
100	1	0x01	10
100	2	0x02	10

10.6 DynamicallyDefineDataIdentifier (0x2C) service

10.6.1 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.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

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 has to 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.

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 has to reference the starting point in the composite block of data and the size parameter has to 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 has to be set to one and the size parameter has to 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.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

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 7.5.

10.6.2 Request message

10.6.2.1 Request message definition

Table 190 defines the request message – sub-function = defineByIdentifier.

Table 190 — Request message definition - sub-function = defineByIdentifier

A_Data Byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	DynamicallyDefineDataIdentifier Request SID	M	0x2C	DDDI
#2	sub-function = [definitionType = defineByIdentifier]	M	0x01	LEV_ DBID
#3 #4	dynamicallyDefinedDataIdentifier[] = [byte#1 (MSB) byte#2 (LSB)]	M M	0xF2 / 0xF3 0x00 – 0xFF	DDDDI_ HB LB
#5 #6	sourceDataIdentifier[]#1 = [byte#1 (MSB) byte#2 (LSB)]	M M	0x00 – 0xFF 0x00 – 0xFF	SDI_ HB LB
#7	positionInSourceDataRecord#1	M	0x01 – 0xFF	PISDR#1
#8	memorySize#1	M	0x00 – 0xFF	MS#1
:	:	:	:	:
#n-3 #n-2	sourceDataIdentifier[]#m = [byte#1 (MSB) byte#2 (LSB)]	U U	0x00 – 0xFF 0x00 – 0xFF	SDI_ HB LB
#n-1	positionInSourceDataRecord#m	U	0x01 – 0xFF	PISDR#m
#n	memorySize#m	U	0x00 – 0xFF	MS#m

Table 191 defines the request message – sub-function = defineByMemoryAddress.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 191 — Request message definition - sub-function = defineByMemoryAddress

A_Data Byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	DynamicallyDefineDataIdentifier Request SID	M	0x2C	DDDI
#2	sub-function = [definitionType = defineByMemoryAddress]	M	0x02	LEV_ DBMA
#3 #4	dynamicallyDefinedDataIdentifier[] = [byte#1 (MSB) byte#2 (LSB)]	M M	0xF2 / 0xF3 0x00 – 0xFF	DDDDI_ HB LB
#5	addressAndLengthFormatIdentifier	M1	0x00 – 0xFF	ALFID
#6 : #(m-1)+6	memoryAddress[] = [byte#1 (MSB) : byte#m]	M : C1	0x00 – 0xFF : 0x00 – 0xFF	MA_ B1 : Bm
#m+6 : #m+6+(k-1)	memorySize[] = [byte#1 (MSB) : byte#k]	M : C2	0x00 – 0xFF : 0x00 – 0xFF	MS_ B1 : Bk
:	:	:	:	:
#n-k-(m-1) : #n-k	memoryAddress[] = [byte#1 (MSB) : byte#m]	U : U/C1	0x00 – 0xFF : 0x00 – 0xFF	MA_ B1 : Bm
#n-(k-1) : #n	memorySize[] = [byte#1 (MSB) : byte#k]	U : U/C2	0x00 – 0xFF : 0x00 – 0xFF	MS_ B1 : Bk
M1: The addressAndLengthFormatIdentifier parameter is only present once at the very beginning of the request message and defines 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 192 defines the request message – sub-function = clearDynamicallyDefinedDataIdentifier.

Table 192 — Request message definition - sub-function = clearDynamicallyDefinedDataIdentifier

A_Data Byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	DynamicallyDefineDataIdentifier Request SID	M	0x2C	DDDI
#2	sub-function = [definitionType = clearDynamicallyDefinedDataIdentifier]	M	0x03	LEV_ CDDDID
#3 #4	dynamicallyDefinedDataIdentifier[] = [byte#1 (MSB) byte#2 (LSB)]	C C	0xF2 / 0xF3 0x00 – 0xFF	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.				

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

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

The sub-parameters defined as valid for the request message of this service are indicated in Table 193 (suppressPosRspMsgIndicationBit (bit 7) not shown).

Table 193 — Request message sub-function 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 doesn't exist at the time of the request. However, 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

10.6.2.3 Request message data-parameter definition

Table 194 defines the data-parameters of the request message.

Table 194 — 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 sub-function = 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 sub-function = 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.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

 © ISO 2013 – All rights reserved

Table 194 — *(continued)*

Definition

addressAndLengthFormatIdentifier
This parameter is a one byte value with each nibble encoded separately (see H.1 for example values):
bit 7 - 4: Length (number of bytes) of the memorySize parameter(s);
bit 3 - 0: Length (number of bytes) of the memoryAddress parameter(s);

memoryAddress This parameter is only present for sub-function = 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 - 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. In case of sub-function = 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 sub-function = 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 - 4) of the addressAndLengthFormatIdentifier.

10.6.3 Positive response message

10.6.3.1 Positive response message definition

Table 195 defines the positive response message.

Table 195 — Positive response message definition

A_Data Byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	DynamicallyDefineDataIdentifier Response SID	M	0x6C	DDDIPR
#2	sub-function = [definitionType]	M	0x00 – 0x7F	DM
	dynamicallyDefinedDataIdentifier [] = [			DDDDI_
#3	byte#1 (MSB)	C	0xF2 / 0xF3	HB
#4	byte#2 (LSB)]	C	0x00 – 0xFF	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.				

10.6.3.2 Positive response message data-parameter definition

Table 196 defines the data-parameters of the positive response message.

Table 196 — Response message data-parameter definition

Definition
definitionType This parameter is an echo of bits 6 - 0 of the sub-function parameter from the request message.
dynamicallyDefinedDataIdentifier This parameter is an echo of the data-parameter dynamicallyDefinedDataIdentifier from the request message.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

10.6.4 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 197. The listed negative responses shall be used if the error scenario applies to the server.

Table 197 — Supported negative response codes

NRC	Description	Mnemonic
0x12	**sub-functionNotSupported** This NRC shall be sent if the sub-function parameter is not supported.	SFNS
0x13	**incorrectMessageLengthOrInvalidFormat** This NRC shall be sent if the length of the message is wrong.	IMLOIF
0x22	**conditionsNotCorrect** This NRC shall be sent if the operating conditions of the server are not met to perform the required action.	CNC
0x31	**requestOutOfRange** 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;	ROOR
0x33	**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

10.6.5 Message flow examples DynamicallyDefineDataIdentifier

10.6.5.1 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.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

 © ISO 2013 – All rights reserved

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 198 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 [6] 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 sub-function parameter) to "FALSE" ('0').

Table 198 — Composite data blocks - DataIdentifier definitions

Data Identifier (block)	Data Byte	Data Record Contents	Byte Value
0x010A	#1	dataRecord [data#1] = B+	0x8C
	#2	dataRecord [data#2] = ECT	0xA6
	#3	dataRecord [data#3] = TP	0x66
	#4	dataRecord [data#4] = RPM	0x07
	#5	dataRecord [data#5] = RPM	0x50
	#6	dataRecord [data#6] = MAP	0x20
	#7	dataRecord [data#7] = MAF	0x1A
	#8	dataRecord [data#8] = VSS	0x00
	#9	dataRecord [data#9] = BARO	0x63
	#10	dataRecord [data#10] = LOAD	0x4A
	#11	dataRecord [data#11] = IAC	0x82
	#12	dataRecord [data#12] = APP	0x7E
0x050B	#1	dataRecord [data#1] = SPARKADV	0x00
	#2	dataRecord [data#2] = KS	0x91

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 199 defines the elemental data records – DID definitions.

Table 199 — Elemental data records - DID definitions

Data Identifier (DID)	Data Byte	Data Record Contents	Byte Value
0x1234	#1	EOT (MSB)	0x4C
	#2	EOT (LSB)	0x36
0x5678	#1	AAT	0x4D
0x9ABC	#1	EOL (MSB)	0x49
	#2	EOL	0x21
	#3	EOL	0x00
	#4	EOL (LSB)	0x17

Table 200 defines the memory data records – Memory Address definitions.

Table 200 — Memory data records - Memory Address definitions

Memory Address	Data Byte	Data Record Contents	Byte Value
0x21091968	#1	dataRecord [data#1] = B+	0x8C
	#2	dataRecord [data#2] = ECT	0xA6
	#3	dataRecord [data#3] = TP	0x66
	#4	dataRecord [data#4] = RPM	0x07
	#5	dataRecord [data#5] = RPM	0x50
	#6	dataRecord [data#6] = MAP	0x20
	#7	dataRecord [data#7] = MAF	0x1A
	#8	dataRecord [data#8] = VSS	0x00
	#9	dataRecord [data#9] = BARO	0x63
	#10	dataRecord [data#10] = LOAD	0x4A
	#11	dataRecord [data#11] = IAC	0x82
	#12	dataRecord [data#12] = APP	0x7E
0x13101994	#1	dataRecord [data#1] = SPARKADV	0x00
	#2	dataRecord [data#2] = KS	0x91

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

10.6.5.2 Example #1: DynamicallyDefineDataIdentifier, sub-function = defineByIdentifier

The example in Table 201 will build up a dynamic data identifier (DDDI 0xF301) containing engine oil temperature, ambient air temperature, and engine oil level using the 2-byte DIDs as the reference for the required data.

Table 201 — DynamicallyDefineDataIdentifier request DDDDI 0xF301 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	0x2C	DDDI
#2	sub-function = defineByIdentifier, suppressPosRspMsgIndicationBit = FALSE	0x01	DBID
#3	dynamicallyDefinedDataIdentifier [byte#1] (MSB)	0xF3	DDDDI_B1
#4	dynamicallyDefinedDataIdentifier [byte#2] (LSB)	0x01	DDDDI_B2
#5	sourceDataIdentifier#1 [byte#1] (MSB) - Engine Oil Temperature	0x12	SDI_B1
#6	sourceDataIdentifier#1 [byte#2]	0x34	SDI_B2
#7	positionInSourceDataRecord#1	0x01	PISDR#1
#8	memorySize#1	0x02	MS#1
#9	sourceDataIdentifier#2 [byte#1] (MSB) - Ambient Air Temperature	0x56	SDI_B1
#10	sourceDataIdentifier#2 [byte#2]	0x78	SDI_B2
#11	positionInSourceDataRecord#2	0x01	PISDR#2
#12	memorySize#2	0x01	MS#2
#13	sourceDataIdentifier#3 [byte#1] (MSB) - Engine Oil Level	0x9A	SDI_B1
#14	sourceDataIdentifier#3 [byte#2]	0xBC	SDI_B2
#15	positionInSourceDataRecord#3	0x01	PISDR#3
#16	memorySize#3	0x04	MS#3

Table 202 defines the DynamicallyDefineDataIdentifier positive response DDDDI 0xF301 message flow example #1.

Table 202 — DynamicallyDefineDataIdentifier positive response DDDDI 0xF301 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	0x6C	DDDIPR
#2	definitionMode = defineByIdentifier	0x01	DBID
#3	dynamicallyDefinedDataIdentifier [byte#1] (MSB)	0xF3	DDDDI_B1
#4	dynamicallyDefinedDataIdentifier [byte#2] (LSB)	0x01	DDDDI_B2

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 203 defines the ReadDataByIdentifier request DDDDI 0xF301 message flow example #1.

Table 203 — ReadDataByIdentifier request DDDDI 0xF301 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	0x22	RDBI
#2	dataIdentifier [byte#1] (MSB)	0xF3	DID_B1
#3	dataIdentifier [byte#2] (LSB)	0x01	DID_B2

Table 204 defines the ReadDataByIdentifier positive response DDDDI 0xF301 message flow example #1.

Table 204 — ReadDataByIdentifier positive response DDDDI 0xF301 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	0x62	RDBIPR
#2	dataIdentifier [byte#1] (MSB)	0xF3	DID_B1
#3	dataIdentifier [byte#2] (LSB)	0x01	DID_B2
#4	dataRecord [data#1] = EOT	0x4C	DREC_DATA_1
#5	dataRecord [data#2] = EOT	0x36	DREC_DATA_2
#6	dataRecord [data#3] = AAT	0x4D	DREC_DATA_3
#7	dataRecord [data#4] = EOL	0x49	DREC_DATA_4
#8	dataRecord [data#5] = EOL	0x21	DREC_DATA_5
#9	dataRecord [data#6] = EOL	0x00	DREC_DATA_6
#10	dataRecord [data#7] = EOL	0x17	DREC_DATA_7

10.6.5.3 Example #2: DynamicallyDefineDataIdentifier, sub-function = defineByIdentifier

The example in Table 205 will build up a dynamic data identifier (DDDDI 0xF302) containing engine coolant temperature (from data record 0x010A), engine speed (from data record 0x010A), IAC Pintle Position (from data record 0x010A) and knock sensor (from data record 0x050B).

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 205 — DynamicallyDefineDataIdentifier request DDDDI 0xF302 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	0x2C	DDDI
#2	sub-function = defineByIdentifier, suppressPosRspMsgIndicationBit = FALSE	0x01	DBID
#3	dynamicallyDefinedDataIdentifier [byte#1] (MSB)	0xF3	DDDDI_B1
#4	dynamicallyDefinedDataIdentifier [byte#2] (LSB)	0x02	DDDDI_B2
#5	sourceDataIdentifier#1 [byte#1] (MSB)	0x01	SDI_B1
#6	sourceDataIdentifier#1 [byte#2] (LSB)	0x0A	SDI_B2
#7	positionInSourceDataRecord#1 - Engine Coolant Temperature	0x02	PISDR#1
#8	memorySize#1	0x01	MS#1
#9	sourceDataIdentifier#2 [byte#1] (MSB)	0x01	SDI_B1
#10	sourceDataIdentifier#2 [byte#2] (LSB)	0x0A	SDI_B2
#11	positionInSourceDataRecord#2 - Engine Speed	0x04	PISDR#2
#12	memorySize#2	0x02	MS#2
#13	sourceDataIdentifier#3 [byte#1] (MSB)	0x01	SDI_B1
#14	sourceDataIdentifier#3 [byte#2] (LSB)	0x0A	SDI_B2
#15	positionInSourceDataRecord#3 – Idle Air Control	0x0B	PISDR#3
#16	memorySize#3	0x01	MS#3
#17	sourceDataIdentifier#4 [byte#1] (MSB)	0x05	SDI_B1
#18	sourceDataIdentifier#4 [byte#2] (LSB)	0x0B	SDI_B2
#19	positionInSourceDataRecord#4 - Knock Sensor	0x02	PISDR#4
#20	memorySize#4	0x01	MS#4

Table 206 defines the DynamicallyDefineDataIdentifier positive response DDDDI 0xF302 message flow example #2.

Table 206 — DynamicallyDefineDataIdentifier positive response DDDDI 0xF302 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	0x6C	DDDIPR
#2	definitionMode = defineByIdentifier	0x01	DBID
#3	dynamicallyDefinedDataIdentifier [byte#1] (MSB)	0xF3	DDDDI_B1
#4	dynamicallyDefinedDataIdentifier [byte#2] (LSB)	0x02	DDDDI_B2

Table 207 defines the ReadDataByIdentifier request DDDDI 0xF302 message flow example #2.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 207 — ReadDataByIdentifier request DDDDI 0xF302 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	0x22	RDBI
#2	dataIdentifier [byte#1] (MSB)	0xF3	DID_B1
#3	dataIdentifier [byte#2] (LSB)	0x02	DID_B2

Table 208 defines the ReadDataByIdentifier positive response DDDDI 0xF302 message flow example #2.

Table 208 — ReadDataByIdentifier positive response DDDDI 0xF302 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	0x62	RDBIPR
#2	dataIdentifier [byte#1] (MSB)	0xF3	DID_B1
#3	dataIdentifier [byte#2] (LSB)	0x02	DID_B2
#4	dataRecord [data#1] = ECT	0xA6	DREC_DATA_1
#5	dataRecord [data#2] = RPM	0x07	DREC_DATA_2
#6	dataRecord [data#3] = RPM	0x50	DREC_DATA_3
#7	dataRecord [data#4] = IAC	0x82	DREC_DATA_4
#8	dataRecord [data#5] = KS	0x91	DREC_DATA_5

10.6.5.4 Example #3: DynamicallyDefineDataIdentifier, sub-function = defineByMemoryAddress

The example in Table 209 will build up a dynamic data identifier (DDDDI 0xF302) containing engine coolant temperature (from memory block starting at memory address 0x21091969), engine speed (from memory block starting at memory address 0x2109196B), and knock sensor (from memory block starting at memory address 0x13101995).

Table 209 — DynamicallyDefineDataIdentifier request DDDDI 0xF302 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	0x2C	DDDI
#2	sub-function = defineByMemoryAddress, suppressPosRspMsgIndicationBit = FALSE	0x02	DBMA
#3	dynamicallyDefinedDataIdentifier [byte#1] (MSB)	0xF3	DDDDI_B1
#4	dynamicallyDefinedDataIdentifier [byte#2] (LSB)	0x02	DDDDI_B2
#5	addressAndLengthFormatIdentifier	0x14	ALFID
#6	memoryAddress#1 [byte#1] (MSB) - Engine Coolant Temperature	0x21	MA_1_B1
#7	memoryAddress#1 [byte#2]	0x09	MA_1_B2

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 209 — *(continued)*

Message direction	**client → server**		
Message Type	**Request**		
A_Data Byte	**Description (all values are in hexadecimal)**	**Byte Value**	**Mnemonic**
#8	memoryAddress#1 [byte#3]	0x19	MA_1_B3
#9	memoryAddress#1 [byte#4]	0x69	MA_1_B4
#10	memorySize#1	0x01	MS#1
#11	memoryAddress#2 [byte#1] (MSB) - Engine Speed	0x21	MA_2_B1
#12	memoryAddress#2 [byte#2]	0x09	MA_2_B2
#13	memoryAddress#2 [byte#3]	0x19	MA_2_B3
#14	memoryAddress#2 [byte#4]	0x6B	MA_2_B4
#15	memorySize#2	0x02	MS#2
#16	memoryAddress#3 [byte#1] (MSB) - Knock Sensor	0x13	MA_3_B1
#17	memoryAddress#3 [byte#2]	0x10	MA_3_B2
#18	memoryAddress#3 [byte#3]	0x19	MA_3_B3
#19	memoryAddress#3 [byte#4]	0x95	MA_3_B4
#20	memorySize#3	0x01	MS#3

Table 210 defines the DynamicallyDefineDataIdentifier positive response DDDDI 0xF302 message flow example #3.

Table 210 — DynamicallyDefineDataIdentifier positive response DDDDI 0xF302 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	0x6C	DDDIPR
#2	definitionMode = defineByMemoryAddress	0x02	DBMA
#3	dynamicallyDefinedDataIdentifier [byte#1] (MSB)	0xF3	DDDDI_B1
#4	dynamicallyDefinedDataIdentifier [byte#2] (LSB)	0x02	DDDDI_B2

Table 211 defines the ReadDataByIdentifier request DDDDI 0xF302 message flow example #3.

Table 211 — ReadDataByIdentifier request DDDDI 0xF302 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	0x22	RDBI
#2	dataIdentifier [byte#1] (MSB)	0xF3	DID_B1
#3	dataIdentifier [byte#2] (LSB)	0x02	DID_B2

Table 212 defines the ReadDataByIdentifier positive response DDDDI 0xF302 message flow example #3.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 212 — ReadDataByIdentifier positive response DDDDI 0xF302 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	0x62	RDBIPR
#2	dataIdentifier [byte#1] (MSB)	0xF3	DID_B1
#3	dataIdentifier [byte#2] (LSB)	0x02	DID_B2
#4	dataRecord [data#1] = ECT	0xA6	DREC_DATA_1
#5	dataRecord [data#2] = RPM	0x07	DREC_DATA_2
#6	dataRecord [data#3] = RPM	0x50	DREC_DATA_3
#7	dataRecord [data#4] = KS	0x91	DREC_DATA_4

10.6.5.5 Example #4: DynamicallyDefineDataIdentifier, sub-function = defineByIdentifier

The example in Table 213 will build up a dynamic data identifier (DDDDI 0xF2E7) containing engine coolant temperature (from data record 0x010A), engine speed (from data record 0x010A), and knock sensor (from data record 0x050B).

The value for the dynamic data identifier is choosen 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).

Table 213 — DynamicallyDefineDataIdentifier request DDDDI 0xF2E7 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	0x2C	DDDI
#2	sub-function = defineByIdentifier, suppressPosRspMsgIndicationBit = FALSE	0x01	DBID
#3	dynamicallyDefinedDataIdentifier [byte#1] (MSB)	0xF2	DDDDI_B1
#4	dynamicallyDefinedDataIdentifier [byte#2] (LSB)	0xE7	DDDDI_B2
#5	sourceDataIdentifier#1 [byte#1] (MSB)	0x01	SDI_B1
#6	sourceDataIdentifier#1 [byte#2] (LSB)	0x0A	SDI_B2
#7	positionInSourceDataRecord#1 - Engine Coolant Temperature	0x02	PISDR
#8	memorySize#1	0x01	MS
#9	sourceDataIdentifier#2 [byte#1] (MSB)	0x01	SDI_B1
#10	sourceDataIdentifier#2 [byte#2] (LSB)	0x0A	SDI_B2
#11	positionInSourceDataRecord#2 - Engine Speed	0x04	PISDR
#12	memorySize#2	0x02	MS
#13	sourceDataIdentifier#3 [byte#1] (MSB)	0x05	SDI_B1
#14	sourceDataIdentifier#3 [byte#2] (LSB)	0x0B	SDI_B2
#15	positionInSourceDataRecord#3 - Knock Sensor	0x02	PISDR
#16	memorySize#3	0x01	MS

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 214 defines the DynamicallyDefineDataIdentifier positive response DDDDI 0xF2E7 message flow example #4.

Table 214 — DynamicallyDefineDataIdentifier positive response DDDDI 0xF2E7 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	0x6C	DDDIPR
#2	definitionMode = defineByIdentifier	0x01	DBID
#3	dynamicallyDefinedDataIdentifier [byte#1] (MSB)	0xF2	DDDDI_B1
#4	dynamicallyDefinedDataIdentifier [byte#2] (LSB)	0xE7	DDDDI_B2

Table 215 defines the ReadDataByPeriodicIdentifier request DDDDI 0xF2E7 message flow example #4.

Table 215 — ReadDataByPeriodicIdentifier request DDDDI 0xF2E7 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	0x2A	RDBPI
#2	transmissionMode = sendAtFastRate	0x04	TM
#3	PeriodicDataIdentifier	0xE7	PDID

Table 216 defines the ReadDataByPeriodicIdentifier initial positive message flow example #4.

Table 216 — 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	0x6A	RDBPIPR

Table 217 and Table 218 define the ReadDataByPeriodicIdentifier periodic data response #1 DDDDI 0xF2E7 message flow example #4.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 217 — ReadDataByPeriodicIdentifier periodic data response #1 DDDDI 0xF2E7 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	0xE7	PDID
#2	dataRecord [data#1] = ECT	0xA6	DREC_DATA_1
#3	dataRecord [data#2] = RPM	0x07	DREC_DATA_2
#4	dataRecord [data#3] = RPM	0x50	DREC_DATA_3
#5	dataRecord [data#4] = KS	0x91	DREC_DATA_4

NOTE Multiple response messages with different byte values are not shown in this example.

Table 218 — ReadDataByPeriodicIdentifier periodic data response #n DDDDI 0xF2E7 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	0xE7	PDID
#2	dataRecord [data#1] = ECT	0xA6	DREC_DATA_1
#3	dataRecord [data#2] = RPM	0x07	DREC_DATA_2
#4	dataRecord [data#3] = RPM	0x55	DREC_DATA_3
#5	dataRecord [data#4] = KS	0x98	DREC_DATA_4

10.6.5.6 Example #5: DynamicallyDefineDataIdentifier, sub-function = clearDynamicallyDefined-DataIdentifier

The example in Table 219 demonstrates the clearing of a dynamicallyDefinedDataIdentifier, and assumes that DDDDI 0xF303 exists at the time of the request.

Table 219 — DynamicallyDefineDataIdentifier request clear DDDDI 0xF303 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	0x2C	DDDI
#2	sub-function = clearDynamicallyDefinedDataIdentifier, suppressPosRspMsgIndicationBit = FALSE	0x03	CDDDI
#3	dynamicallyDefinedDataIdentifier [byte#1] (MSB)	0xF3	DDDDI_B1
#4	dynamicallyDefinedDataIdentifier [byte#2] (LSB)	0x03	DDDDI_B2

Table 220 defines the DynamicallyDefineDataIdentifier positive response clear DDDDI 0xF303 message flow example #5.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

 © ISO 2013 – All rights reserved

Table 220 — DynamicallyDefineDataIdentifier positive response clear DDDDI 0xF303 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	0x6C	DDDIPR
#2	definitionMode = clearDynamicallyDefinedDataIdentifier	0x03	CDDDI
#3	dynamicallyDefinedDataIdentifier [byte#1] (MSB)	0xF3	DDDDI_B1
#4	dynamicallyDefinedDataIdentifier [byte#2] (LSB)	0x03	DDDDI_B2

10.6.5.7 Example #6: DynamicallyDefineDataIdentifier, concatenation of definitions (defineByIdentifier / defineByAddress)

This example will build up a dynamic data identifier (DDDI 0xF301) 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 referecend by 2-byte DID.

10.6.5.7.1 Step #1: DynamicallyDefineDataIdentifier, sub-function = defineByIdentifier (1 st portion)

Table 221 defines the DynamicallyDefineDataIdentifier request DDDI 0xF301 message flow example #6 definition of 1st portion (defineByIdentifier).

Table 221 — DynamicallyDefineDataIdentifier request DDDI 0xF301 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	0x2C	DDDI
#2	sub-function = defineByIdentifier, suppressPosRspMsgIndicationBit = FALSE	0x01	DBID
#3	dynamicallyDefinedDataIdentifier [byte#1] (MSB)	0xF3	DDDDI_B1
#4	dynamicallyDefinedDataIdentifier [byte#2] (LSB)	0x01	DDDDI_B2
#5	sourceDataIdentifier#1 [byte#1] (MSB) - Engine Oil Temperature	0x12	SDI_B1
#6	sourceDataIdentifier#1 [byte#2]	0x34	SDI_B2
#7	positionInSourceDataRecord#1	0x01	PISDR#1

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 221 — *(continued)*

Message direction	**client → server**		
Message Type	**Request**		
A_Data Byte	**Description (all values are in hexadecimal)**	**Byte Value**	**Mnemonic**
#8	memorySize#1	0x02	MS#1
#9	sourceDataIdentifier#2 [byte#1] (MSB) - Ambient Air Temperature	0x56	SDI_B1
#10	sourceDataIdentifier#2 [byte#2] (LSB)	0x78	SDI_B2
#11	positionInSourceDataRecord#2	0x01	PISDR#2
#12	memorySize#2	0x01	MS#2

Table 222 defines the DynamicallyDefineDataIdentifier positive response DDDI 0xF301 message flow example #6 definition of first portion (defineByIdentifier).

Table 222 — DynamicallyDefineDataIdentifier positive response DDDI 0xF301 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	0x6C	DDDIPR
#2	definitionMode = defineByIdentifier	0x01	DBID
#3	dynamicallyDefinedDataIdentifier [byte#1] (MSB)	0xF3	DDDDI_B1
#4	dynamicallyDefinedDataIdentifier [byte#2] (LSB)	0x01	DDDDI_B2

10.6.5.7.2 Step #2: DynamicallyDefineDataIdentifier, sub-function = defineByMemoryAddress (2 nd portion)

Table 223 defines the DynamicallyDefineDataIdentifier request DDDDI 0xF301 message flow example #6 definition of 2nd portion (defineByMemoryAddress).

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 223 — DynamicallyDefineDataIdentifier request DDDDI 0xF301 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	0x2C	DDDI
#2	sub-function = defineByMemoryAddress, suppressPosRspMsgIndicationBit = FALSE	0x02	DBMA
#3	dynamicallyDefinedDataIdentifier [byte#1] (MSB)	0xF3	DDDDI_B1
#4	dynamicallyDefinedDataIdentifier [byte#2] (LSB)	0x01	DDDDI_B2
#5	addressAndLengthFormatIdentifier	0x14	ALFID
#6	memoryAddress#1 [byte#1] (MSB) - Engine Coolant Temperature	0x21	MA_B1#1
#7	memoryAddress#1 [byte#2]	0x09	MA_B2#1
#8	memoryAddress#1 [byte#3]	0x19	MA_B3#1
#9	memoryAddress#1 [byte#4]	0x69	MA_B4#1
#10	memorySize#1	0x01	MS#1
#11	memoryAddress#2 [byte#1] (MSB) - Engine Speed	0x21	MA_B1#2
#12	memoryAddress#2 [byte#2]	0x09	MA_B2#2
#13	memoryAddress#2 [byte#3]	0x19	MA_B3#2
#14	memoryAddress#2 [byte#4]	0x6B	MA_B4#2
#15	memorySize#2	0x02	MS#2

Table 222 defines the DynamicallyDefineDataIdentifier positive response DDDI 0xF301 message flow example #6.

Table 224 — DynamicallyDefineDataIdentifier positive response DDDI 0xF301 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	0x6C	DDDIPR
#2	definitionMode = defineByMemoryAddress	0x02	DBMA
#3	dynamicallyDefinedDataIdentifier [byte#1] (MSB)	0xF3	DDDDI_B1
#4	dynamicallyDefinedDataIdentifier [byte#2] (LSB)	0x01	DDDDI_B2

10.6.5.7.3 Step #3: DynamicallyDefineDataIdentifier, sub-function = defineByIdentifier (3 rd portion)

Table 225 defines the DynamicallyDefineDataIdentifier request DDDI 0xF301 message flow example #6 definition of 3rd portion (defineByIdentifier).

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 225 — DynamicallyDefineDataIdentifier request DDDI 0xF301 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	0x2C	DDDI
#2	sub-function = defineByIdentifier, suppressPosRspMsgIndicationBit = FALSE	0x01	DBID
#3	dynamicallyDefinedDataIdentifier [byte#1] (MSB)	0xF3	DDDDI_B1
#4	dynamicallyDefinedDataIdentifier [byte#2] (LSB)	0x01	DDDDI_B2
#5	sourceDataIdentifier#1 [byte#1] (MSB) - Engine Oil Level	0x9A	SDI_B1
#6	sourceDataIdentifier#1 [byte#2]	0xBC	SDI_B2
#7	positionInSourceDataRecord#1	0x01	PISDR#3
#8	memorySize#1	0x04	MS#3

Table 226 defines the DynamicallyDefineDataIdentifier positive response DDDI 0xF301 message flow example #6.

Table 226 — DynamicallyDefineDataIdentifier positive response DDDI 0xF301 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	0x6C	DDDIPR
#2	definitionMode = defineByIdentifier	0x01	DBID
#3	dynamicallyDefinedDataIdentifier [byte#1] (MSB)	0xF3	DDDDI_B1
#4	dynamicallyDefinedDataIdentifier [byte#2] (LSB)	0x01	DDDDI_B2

10.6.5.7.4 Step #4: ReadDataByIdentifier - dataIdentifier = DDDDI 0xF301

Table 227 defines the ReadDataByIdentifier request DDDDI 0xF301 message flow example #6.

Table 227 — ReadDataByIdentifier request DDDDI 0xF301 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	0x22	RDBI
#2	dataIdentifier [byte#1] (MSB)	0xF3	DID_B1
#3	dataIdentifier [byte#2] (LSB)	0x01	DID_B2

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 228 defines the ReadDataByIdentifier positive response DDDDI 0xF301 message flow example #6.

Table 228 — ReadDataByIdentifier positive response DDDDI 0xF301 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	0x62	RDBIPR
#2	dataIdentifier [byte#1] (MSB)	0xF3	DID_B1
#3	dataIdentifier [byte#2] (LSB)	0x01	DID_B2
#4	dataRecord [data#1] = EOT (MSB)	0x4C	DREC_DATA_1
#5	dataRecord [data#2] = EOT	0x36	DREC_DATA_2
#6	dataRecord [data#3] = AAT	0x4D	DREC_DATA_3
#7	dataRecord [data#4] = ECT	0xA6	DREC_DATA_4
#8	dataRecord [data#5] = RPM	0x07	DREC_DATA_5
#9	dataRecord [data#6] = RPM	0x50	DREC_DATA_6
#10	dataRecord [data#7] = EOL (MSB)	0x49	DREC_DATA_7
#11	dataRecord [data#8] = EOL	0x21	DREC_DATA_8
#12	dataRecord [data#9] = EOL	0x00	DREC_DATA_9
#13	dataRecord [data#10] = EOL	0x17	DREC_DATA_10

10.6.5.7.5 Step #5: DynamicallyDefineDataIdentifier - clear definition of DDDDI 0xF301

Table 229 defines the DynamicallyDefineDataIdentifier request clear DDDDI 0xF301 message flow example #6.

Table 229 — DynamicallyDefineDataIdentifier request clear DDDDI 0xF301 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	0x2C	DDDI
#2	sub-function = clearDynamicallyDefinedDataIdentifier, suppressPosRspMsgIndicationBit = FALSE	0x03	CDDDI
#3	dynamicallyDefinedDataIdentifier [byte#1] (MSB)	0xF3	DDDDI_B1
#4	dynamicallyDefinedDataIdentifier [byte#2] (LSB)	0x01	DDDDI_B2

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 230 defines the DynamicallyDefineDataIdentifier positive response clear DDDDI 0xF301 message flow example #6.

Table 230 — DynamicallyDefineDataIdentifier positive response clear DDDDI 0xF301 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	0x6C	DDDIPR
#2	definitionMode = clearDynamicallyDefinedDataIdentifier	0x03	CDDDI
#3	dynamicallyDefinedDataIdentifier [byte#1] (MSB)	0xF3	DDDDI_B1
#4	dynamicallyDefinedDataIdentifier [byte#2] (LSB)	0x01	DDDDI_B2

10.7 WriteDataByIdentifier (0x2E) service

10.7.1 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,

— Setting option content.

NOTE The server may 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 7.5.

10.7.2 Request message

10.7.2.1 Request message definition

Table 231 defines the request message.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

 © ISO 2013 – All rights reserved

Table 231 — Request message definition

A_Data Byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	WriteDataByIdentifier Request SID	M	0x2E	WDBI
	dataIdentifier[] = [			DID_
#2	byte#1 (MSB)	M	0x00 – 0xFF	HB
#3	byte#2]	M	0x00 – 0xFF	LB
	dataRecord[] = [			DREC_
#4	data#1	M	0x00 – 0xFF	DATA_1
:	:	:	:	:
#m+3	data#m]	U	0x00 – 0xFF	DATA_m

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

This service does not use a sub-function parameter.

10.7.2.3 Request message data-parameter definition

Table 232 defines the data-parameters of the request message.

Table 232 — 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.

10.7.3 Positive response message

10.7.3.1 Positive response message definition

Table 233 defines the positive response message.

Table 233 — Positive response message definition

A_Data Byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	WriteDataByIdentifier Response SID	M	0x6E	WDBIPR
	dataIdentifier[] = [			DID_
#2	byte#1 (MSB)	M	0x00 – 0xFF	HB
#3	byte#2]	M	0x00 – 0xFF	LB

10.7.3.2 Positive response message data-parameter definition

Table 234 defines the data-parameters of the positive response message.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 234 — Response message data-parameter definition

Definition
dataIdentifier This parameter is an echo of the data-parameter dataIdentifier 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 235. The listed negative responses shall be used if the error scenario applies to the server.

Table 235 — Supported negative response codes

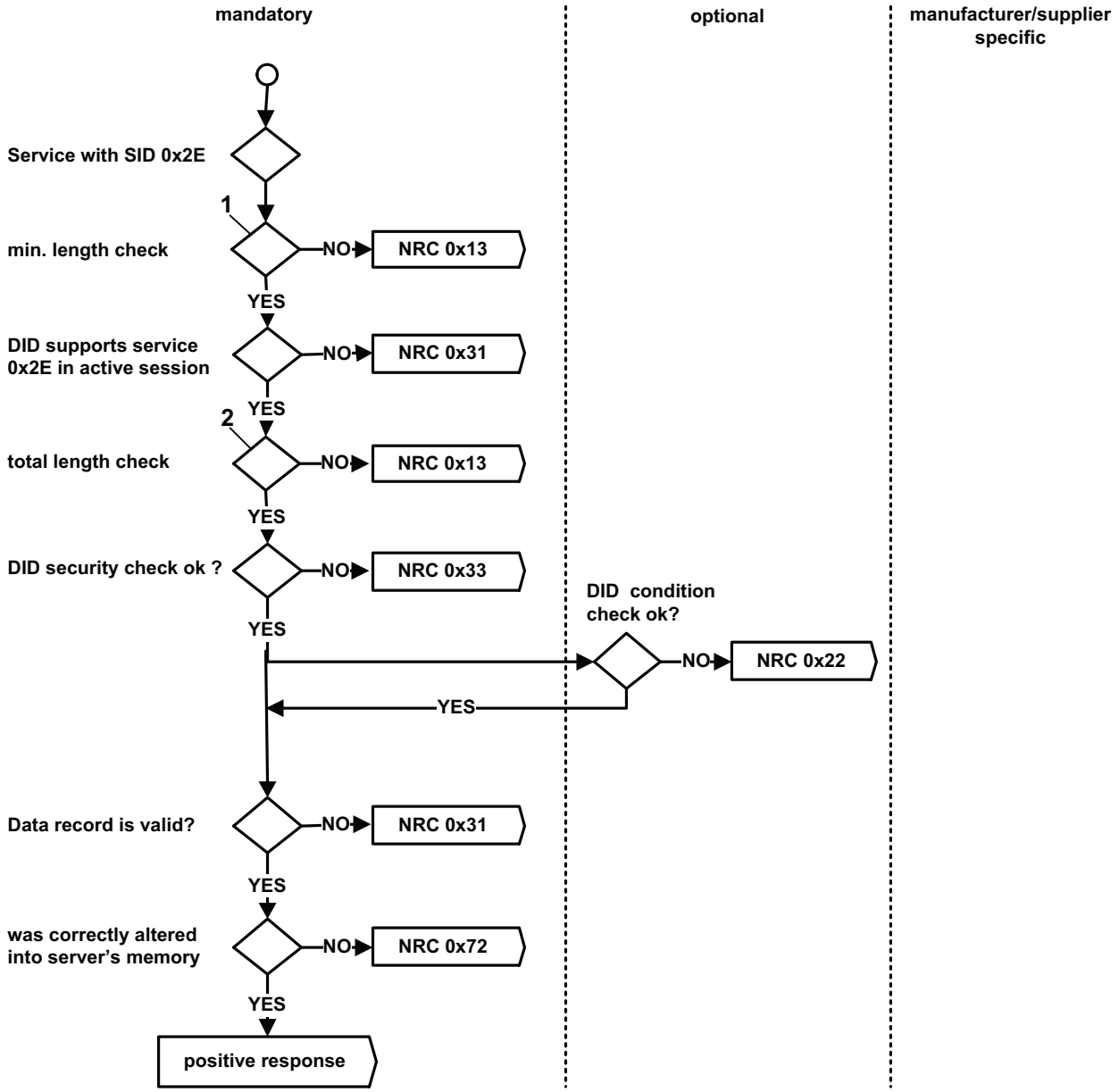
NRC	Description	Mnemonic
0x13	**incorrectMessageLengthOrInvalidFormat** This NRC shall be sent if the length of the message is wrong.	IMLOIF
0x22	**conditionsNotCorrect** This NRC shall be sent if the operating conditions of the server are not met to perform the required action.	CNC
0x31	**requestOutOfRange** 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
0x33	**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
0x72	**generalProgrammingFailure** This NRC shall be returned if the server detects an error when writing to a memory location.	GPF

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

The evaluation sequence is documented in Figure 21.



Key

1 minimum length is 4 byte (SI + DID + DREC)

2 total length is 1 byte (SI + 2 byte DID + nth byte DREC)

Figure 21 — NRC handling for WriteDataByIdentifier service

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

10.7.5 Message flow example WriteDataByIdentifier

10.7.5.1 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 0xF190.

10.7.5.2 Example #1: write dataIdentifier 0xF190 (VIN)

Table 236 defines the WriteDataByIdentifier request message flow example #1.

Table 236 — 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	0x2E	WDBI
#2	dataIdentifier [byte#1] (MSB)	0xF1	DID_B1
#3	dataIdentifier [byte#2]	0x90	DID_B2
#4	dataRecord [data#1] = VIN Digit 1= “W”	0x57	DREC_DATA1
#5	dataRecord [data#2] = VIN Digit 2= “0”	0x30	DREC_DATA2
#6	dataRecord [data#3] = VIN Digit 3= “L”	0x4C	DREC_DATA3
#7	dataRecord [data#4] = VIN Digit 4= “0”	0x30	DREC_DATA4
#8	dataRecord [data#5] = VIN Digit 5= “0”	0x30	DREC_DATA5
#9	dataRecord [data#6] = VIN Digit 6= “0”	0x30	DREC_DATA6
#10	dataRecord [data#7] = VIN Digit 7= “0”	0x30	DREC_DATA7
#11	dataRecord [data#8] = VIN Digit 8= “4”	0x34	DREC_DATA8
#12	dataRecord [data#9] = VIN Digit 9= “3”	0x33	DREC_DATA9
#13	dataRecord [data#10] = VIN Digit 10 = “M”	0x4D	DREC_DATA10
#14	dataRecord [data#11] = VIN Digit 11 = “B”	0x42	DREC_DATA11
#15	dataRecord [data#12] = VIN Digit 12 = “5”	0x35	DREC_DATA12
#16	dataRecord [data#13] = VIN Digit 13 = “4”	0x34	DREC_DATA13
#17	dataRecord [data#14] = VIN Digit 14 = “1”	0x31	DREC_DATA14
#18	dataRecord [data#15] = VIN Digit 15 = “3”	0x33	DREC_DATA15
#19	dataRecord [data#16] = VIN Digit 16 = “2”	0x32	DREC_DATA16
#20	dataRecord [data#17] = VIN Digit 17 = “6”	0x36	DREC_DATA17

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 237 defines the WriteDataByIdentifier positive response message flow example #1.

Table 237 — 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	0x6E	WDBIPR
#2	dataIdentifier [byte#1] (MSB)	0xF1	DID_B1
#3	dataIdentifier [byte#2] (LSB)	0x90	DID_B2

10.8 WriteMemoryByAddress (0x3D) service

10.8.1 Service description

The WriteMemoryByAddress service allows the client to write information into the server at one or more contiguous memory locations.

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 0x00 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;

— Change calibration values;

IMPORTANT — The server and the client shall meet the request and response message behaviour as specified in 7.5.

10.8.2 Request message

10.8.2.1 Request message definition

Table 238 defines the request message definition.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 238 — Request message definition

A_Data Byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	WriteMemoryByAddress Request SID	M	0x3D	WMBA
#2	addressAndLengthFormatIdentifier	M	0x00 – 0xFF	ALFID
#3 : #m+2	memoryAddress[] = [byte#1 (MSB) : byte#m]	M : C1	0x00 – 0xFF : 0x00 – 0xFF	MA_ B1 : Bm
#n-r-2-(k-1) : #n-r-2	memorySize[] = [byte#1 (MSB) : byte#k]	M : C2	0x00 – 0xFF : 0x00 – 0xFF	MS_ B1 : Bk
#n-(r-1) : #n	dataRecord[] = [data#1 : data#r]	M : U	0x00 – 0xFF : 0x00 – 0xFF	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.				

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

This service does not use a sub-function parameter.

10.8.2.3 Request message data-parameter definition

Table 239 defines the data-parameters of the request message.

Table 239 — Request message data-parameter definition

Definition
addressAndLengthFormatIdentifier This parameter is a one byte value with each nibble encoded separately (see 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.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 239 — *(continued)*

dataRecord This parameter provides the data that the client is actually attempting to write into the server memory addresses within the interval {0xMA, (0xMA + 0xMS - 0x01)}.

10.8.3 Positive response message

10.8.3.1 Positive response message definition

Table 240 defines the positive response message.

Table 240 — Positive response message definition

A_Data Byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	WriteMemoryByAddress Response SID	M	0x7D	WMBAPR
#2	addressAndLengthFormatIdentifier	M	0x00 – 0xFF	ALFID
#3 : #(m-1)+3	memoryAddress[] = [byte#1 (MSB) : byte#m]	M : C1	0x00 – 0xFF : 0x00 – 0xFF	MA_ B1 : Bm
#n-(k-1) : #n	memorySize[] = [byte#1 (MSB) : byte#k]	M : C2	0x00 – 0xFF : 0x00 – 0xFF	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.				

10.8.3.2 Positive response message data-parameter definition

Table 241 defines the data-parameters of the positive response message.

Table 241 — 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.

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

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 242 — Supported negative response codes

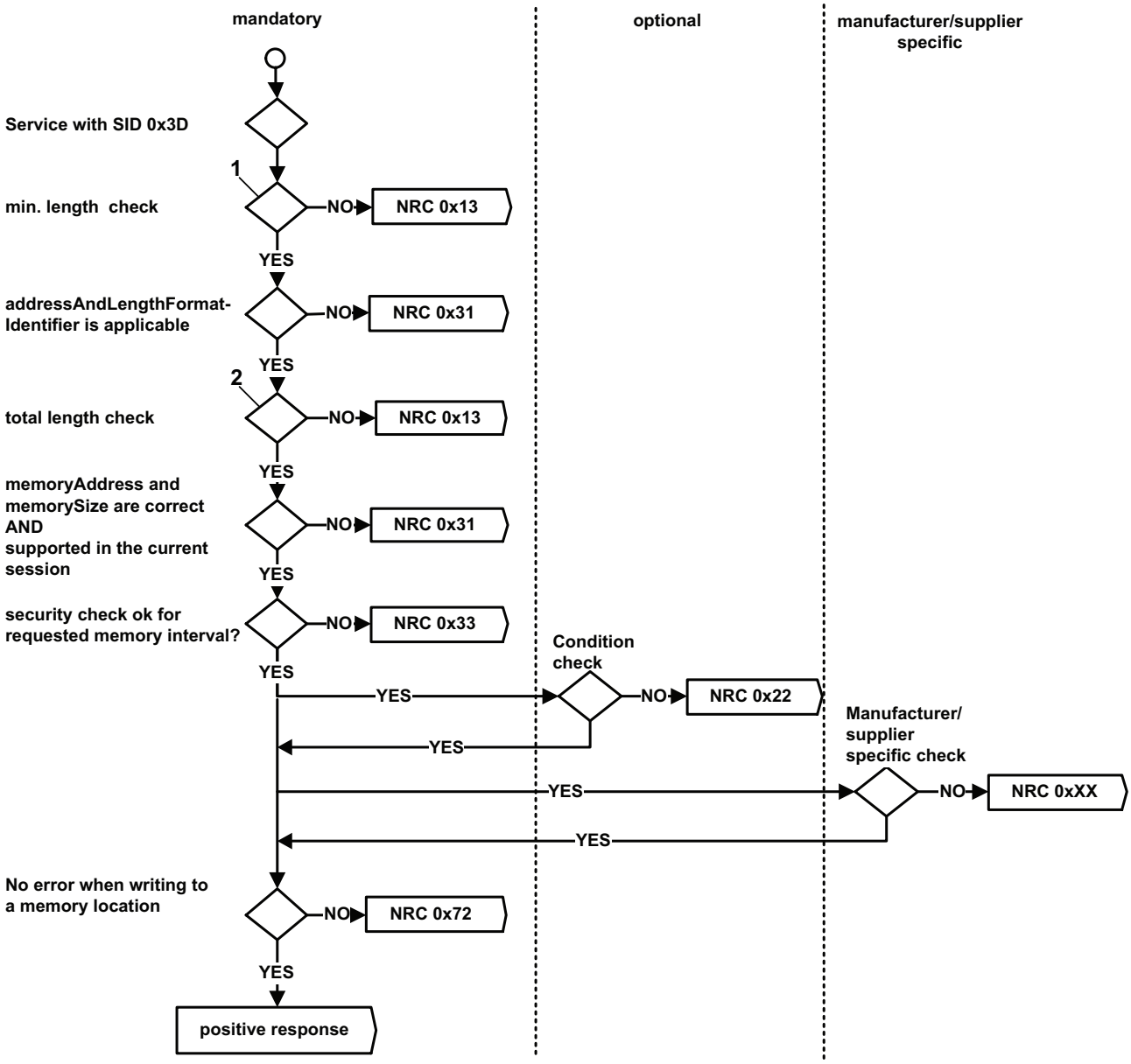
NRC	Description	Mnemonic
0x13	**incorrectMessageLengthOrInvalidFormat** This NRC shall be sent if the length of the message is wrong.	IMLOIF
0x22	**conditionsNotCorrect** This NRC shall be sent if the operating conditions of the server are not met to perform the required action.	CNC
0x31	**requestOutOfRange** This NRC shall be sent if: — Any memory address within the interval [0xMA, (0xMA + 0xMS -0x1)] is invalid; — Any memory address within the interval [0xMA, (0xMA + 0xMS -0x1)] 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
0x33	**securityAccessDenied** This NRC shall be sent if any memory address within the interval [0xMA, (0xMA + 0xMS -0x1)] is secure and the server is locked.	SAD
0x72	**generalProgrammingFailure** This NRC shall be returned if the server detects an error when writing to a memory location.	GPF

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

The evaluation sequence is documented in Figure 22.



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 22 — NRC handling for WriteMemoryByAddress service

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

10.8.5 Message flow example WriteMemoryByAddress

10.8.5.1 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.

10.8.5.2 Example #1: WriteMemoryByAddress, 2-byte (16-bit) addressing

Table 243 defines the WriteMemoryByAddress request message flow example #1.

Table 243 — 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	0x3D	WMBA
#2	addressAndLengthFormatIdentifier	0x12	ALFID
#3	memoryAddress [byte#1] (MSB)	0x20	MA_B1
#4	memoryAddress [byte#2] (LSB)	0x48	MA_B2
#5	memorySize [byte#1]	0x02	MS_B1
#6	dataRecord [data#1]	0x00	DREC_DATA_1
#7	dataRecord [data#2]	0x8C	DREC_DATA_2

Table 244 defines the WriteMemoryByAddress positive response message flow example #1.

Table 244 — 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	0x7D	WMBAPR
#2	addressAndLengthFormatIdentifier	0x12	ALFID
#3	memoryAddress [byte#1] (MSB)	0x20	MA_B1
#4	memoryAddress [byte#2] (LSB)	0x48	MA_B2
#5	memorySize [byte#1]	0x02	MS_B1

10.8.5.3 Example #2: WriteMemoryByAddress, 3-byte (24-bit) addressing

Table 245 defines the WriteMemoryByAddress request message flow example #2.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 245 — 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	0x3D	WMBA
#2	addressAndLengthFormatIdentifier	0x13	ALFID
#3	memoryAddress [byte#1]	0x20	MA_B1
#4	memoryAddress [byte#2]	0x48	MA_B2
#5	memoryAddress [byte#3]	0x13	MA_B3
#6	memorySize [byte#1]	0x03	MS_B1
#7	dataRecord [data#1]	0x00	DREC_DATA_1
#8	dataRecord [data#2]	0x01	DREC_DATA_2
#9	dataRecord [data#3]	0x8C	DREC_DATA_3

Table 246 defines the WriteMemoryByAddress positive response message flow example #2.

Table 246 — 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	0x7D	WMBAPR
#2	addressAndLengthFormatIdentifier	0x13	ALFID
#3	memoryAddress [byte#1]	0x20	MA_B1
#4	memoryAddress [byte#2]	0x48	MA_B2
#5	memoryAddress [byte#3]	0x13	MA_B3
#6	memorySize [byte#1]	0x03	MS_B1

10.8.5.4 Example #3: WriteMemoryByAddress, 4-byte (32-bit) addressing

Table 247 defines the WriteMemoryByAddress request message flow example #3.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 247 — 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	0x3D	WMBA
#2	addressAndLengthFormatIdentifier	0x14	ALFID
#3	memoryAddress [byte#1] (MSB)	0x20	MA_B1
#4	memoryAddress [byte#2]	0x48	MA_B2
#5	memoryAddress [byte#3]	0x13	MA_B3
#6	memoryAddress [byte#4] (LSB)	0x09	MA_B4
#7	memorySize [byte#1]	0x05	MS_B1
#8	dataRecord [data#1]	0x00	DREC_DATA_1
#9	dataRecord [data#2]	0x01	DREC_DATA_2
#10	dataRecord [data#3]	0x8C	DREC_DATA_3
#11	dataRecord [data#4]	0x09	DREC_DATA_4
#12	dataRecord [data#5]	0xAF	DREC_DATA_5

Table 248 defines the WriteMemoryByAddress positive response message flow example #3.

Table 248 — 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	0x7D	WMBAPR
#2	addressAndLengthFormatIdentifier	0x14	ALFID
#3	memoryAddress [byte#1] (MSB)	0x20	MA_B1
#4	memoryAddress [byte#2]	0x48	MA_B2
#5	memoryAddress [byte#3]	0x13	MA_B3
#6	memoryAddress [byte#4] (LSB)	0x09	MA_B4
#7	memorySize [byte#1]	0x05	MS_B1

11 Stored Data Transmission functional unit

11.1 Overview

Table 249 — 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.)

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

11.2 ClearDiagnosticInformation (0x14) Service

11.2.1 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) 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 disconnect during the power-latch phase) this may cause data inconsistency.

The behaviour of the individual DTC status bits shall be implemented according to the definitions in D.2, Figure D.1 - 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 11.3),

— captured DTC snapshot data (DTCSnapshotData, see ReadDTCInformation service in 11.3),

— captured DTC extended data (DTCExtendedData, see ReadDTCInformation service in 11.3),

— other DTC related data such as first/most recent DTC, flags, counters, timers, etc. specific to DTCs,

Any DTC information stored in an optionally available DTC mirror memory in the server is not affected by this service (see ReadDTCInformation (0x19) service in 11.3 for DTC mirror memory definition).

IMPORTANT — The server and the client shall meet the request and response message behaviour as specified in 7.5.

11.2.2 Request message

11.2.2.1 Request message definition

Table 250 defines the request message.

Table 250 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ClearDiagnosticInformation Request SID	M	0x14	CDTCI
	groupOfDTC[] = [			GODTC_
#2	groupOfDTCHighByte	M	0x00 – 0xFF	HB
#3	groupOfDTCMiddleByte	M	0x00 – 0xFF	MB
#4	groupOfDTCLowByte]	M	0x00 – 0xFF	LB

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

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

There are no sub-function parameters used by this service.

11.2.2.3 Request message data-parameter definition

Table 251 defines the data-parameter of the request message.

Table 251 — 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.

11.2.3 Positive response message

11.2.3.1 Positive response message definition

Table 252 defines the positive response message.

Table 252 — Positive response message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ClearDiagnosticInformation Positive Response SID	M	0x54	CDTCIPR

11.2.3.2 Positive response message data-parameter definition

There are no data-parameters used by this service in the positive response message.

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

Table 253 — Supported negative response codes

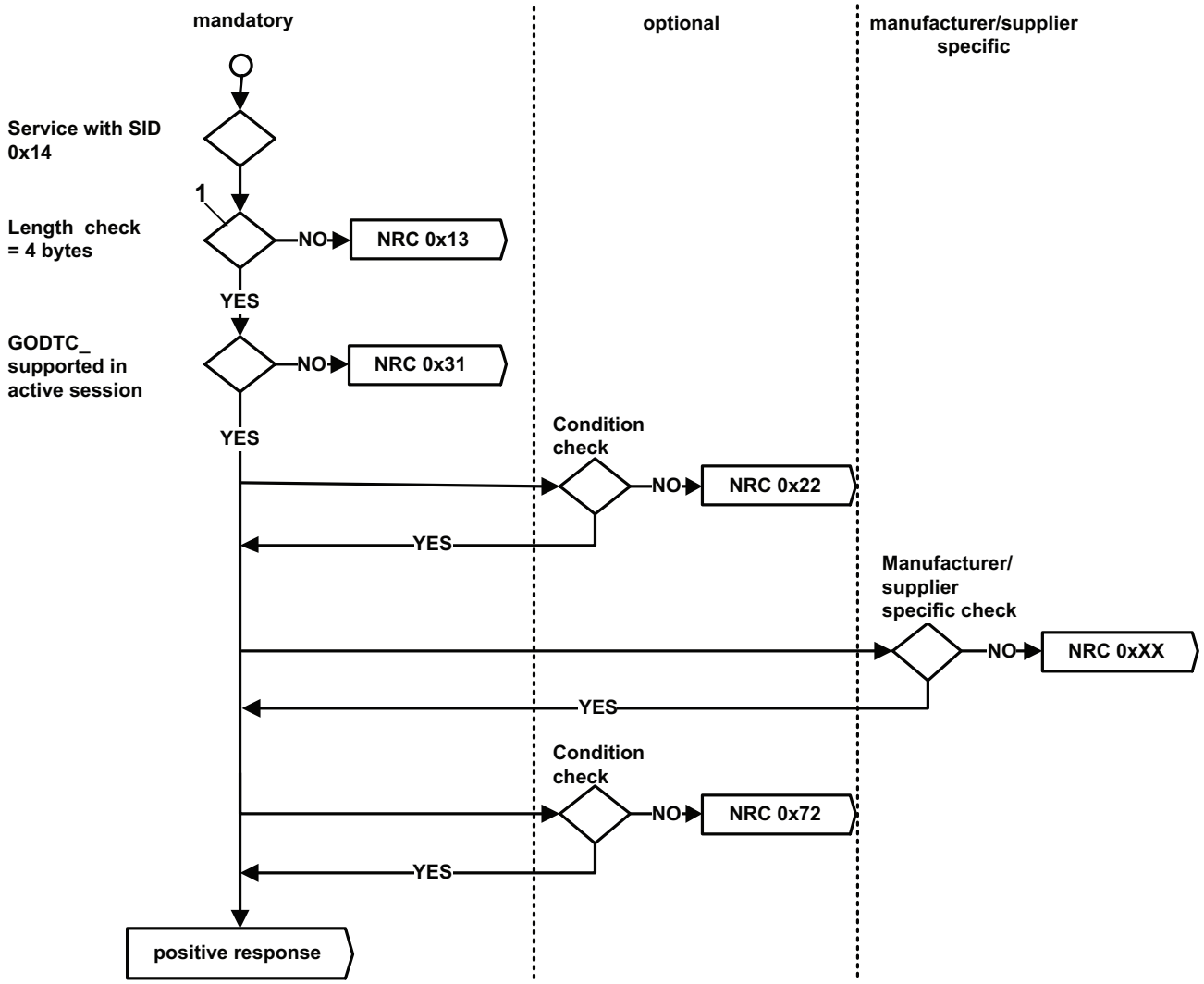
NRC	Description	Mnemonic
0x13	**incorrectMessageLengthOrInvalidFormat** This NRC shall be sent if the length of the message is wrong.	IMLOIF
0x22	**conditionsNotCorrect** This NRC shall be used if internal conditions within the server prevent the clearing of DTC related information stored in the server.	CNC
0x31	**requestOutOfRange** This NRC shall be returned if the specified groupOfDTC parameter is not supported.	ROOR
0x72	**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 23.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved



Key

1 CDTCI + GODTC_

Figure 23 — NRC handling for ClearDiagnosticInformation service

11.2.5 Message flow example ClearDiagnosticInformation

The client sends a ClearDiagnosticInformation request message to a single server. Table 254 defines the ClearDiagnosticInformation request message flow example #1. The client sends a ClearDiagnosticInformation request message to a single server.

Table 254 — 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	0x14	CDTCI
#2	groupOfDTC [DTCHighByte] (“Emissions-related systems”)	0xFF	DTCHB
#3	groupOfDTC [DTCMiddleByte]	0xFF	DTCMB
#4	groupOfDTC [DTCLowByte]	0x33	DTCLB

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 255 defines the ClearDiagnosticInformation positive response message flow example #1.

Table 255 — 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	0x54	CDTCIPR

11.3 ReadDTCInformation (0x19) Service

11.3.1 Service description

11.3.1.1 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 particular subfunction, 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 DTCExtendedData associated with a client defined DTC and status mask combination out of the DTC memory or the DTC mirror memory. DTCExtendedData consists of extended status information associated with a DTC. 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, e.g.

  — 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,

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

— 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 the list of DTCs out of the DTC mirror memory matching a client defined DTC status mask,

— Retrieve mirror memory DTCExtendedData record data for a client defined DTC mask and a client defined DTCExtendedData record number out of the DTC mirror memory,

— Retrieve the number of DTCs out of the DTC mirror memory matching a client defined DTC status mask,

— Retrieve the number of "only" emissions-related OBD DTCs matching a client defined DTC status mask. Emissions-related OBD DTCs cause the malfunction indicator to be turned on/display a message in case such DTC is detected,

— Retrieve the status of "only" emissions-related OBD DTCs matching a client defined DTC status mask. Emissions-related OBD DTCs cause the malfunction indicator to be turned on/display a message in case such DTC is detected,

— 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 mask and a client defined DTCExtendedData record number out of the user defined DTC mirror memory,

— Retrieve user defined DTC memory DTCSnapshotRecord data for a client defined DTC mask out of the user defined DTC memory,

This service uses a sub-function to determine which type of diagnostic information the client is requesting. Further details regarding each sub-function 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.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

— **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, e.g. in the event of a calibrated number of failure free cycles.

A given DTC value (e.g., 0x080511) 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 sub-function = 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 0x000000 and DTC status 0x00. 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 7.5.

11.3.1.2 Retrieving the number of DTCs that match a client defined status mask (sub-function = 0x01 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 sub-function 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.

The sub-function reportNumberOfMirrorMemoryDTCByStatusMask has the same functionality as the sub-function reportNumberOfDTCByStatusMask with the difference that it returns the number of DTCs out of DTC mirror memory.

11.3.1.3 Retrieving the list of DTCs that match a client defined status mask (sub-function = 0x02 reportDTCByStatusMask)

The client can retrieve a list of DTCs, which satisfy a client defined status mask by sending a request with the sub-function byte set to reportDTCByStatusMask. This sub-function allows the client to request the server to report all DTCs 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 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

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

 © ISO 2013 – All rights reserved

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).

11.3.1.4 Retrieving DTCSnapshot record identification (sub-function = 0x03 reportDTCSnapshotIdentification)

A client can retrieve DTCSnapshot record identification information for all captured DTCSnapshot records by sending a request for this service with the sub-function 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).

NOTE A server may 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).

11.3.1.5 Retrieving DTCSnapshot record data for a client defined DTC mask (sub-function = 0x04 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 sub-function 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 DTCSnapshotRecordNumber 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 DTCSnapshotRecordNumber 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 sub-function parameter. It is recommended that the client first requests the identification of DTCSnapshot records stored using the sub-function parameter reportDTCSnapshotIdentification before requesting a specific DTCSnapshotRecordNumber via the reportDTCSnapshotRecordByDTCNumber request..

It is also recommended to support the sub-function 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 pre-defined DTCSnapshotRecord in response to the client's request, if a failure has been identified for the client defined DTCMaskRecord and DTCSnapshotRecordNumber parameters (DTCSnapshotRecordNumber unequal 0xFF).

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

NOTE 2 The exact failure criteria shall be 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 DTCSnapshotRecordNumber to 0xFF, 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 0xFF 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).

11.3.1.6 Retrieving DTCStoredData record data for a client defined record number (sub-function = 0x05 reportDTCStoredDataByRecordNumber)

A client can retrieve captured DTCStoredData record data for a DTCStoredDataRecordNumber by sending a request for this service with the sub-function 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.

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 shall be 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.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

 © ISO 2013 – All rights reserved

The server shall report one DTCStoredDataRecord in a single response message, except the client has set the DTCStoredDataRecordNumber to 0xFF, 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 has to 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 is to 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). 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).

11.3.1.7 Retrieving DTCExtendedData record data for a client defined DTC mask and a client defined DTCExtendedData record number (sub-function = 0x06 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 sub-function 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 pre-defined DTCExtendedData record in response to the client's request (DTCExtDataRecordNumber unequal to 0xFE or 0xFF).

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.

The server shall report one DTCExtendedData record in a single response message, except the client has set the DTCExtDataRecordNumber to 0xFE or 0xFF, because this shall cause the server to response 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 0xFE is sent by the client, but no OBD extended data records (0x90 – 0xEF) are supported by the server. This is to 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).

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

11.3.1.8 Retrieving the number of DTCs that match a client defined severity mask record (sub-function = 0x07 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 sub-function 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.

NOTE 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 sub-function reportDTCByStatusMask, because the request to read the DTCs is done at a different point in time.

11.3.1.9 Retrieving severity and functional unit information that match a client defined severity mask record (sub-function = 0x08 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 sub-function byte set to reportDTCBySeverityMaskRecord. This sub-function 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,

(((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.

11.3.1.10 Retrieving severity and functional unit information for a client defined DTC (sub-function = 0x09 reportSeverityInformationOfDTC)

A client can retrieve severity and functional unit information for a client defined DTCMaskRecord by sending a request for this service with the sub-function 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)).

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

 © ISO 2013 – All rights reserved

11.3.1.11 Retrieving the status of all DTCs supported by the server (sub-function = 0x0A reportSupportedDTC)

A client can retrieve the status of all DTCs supported by the server by sending a request for this service with the sub-function 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.

11.3.1.12 Retrieving the first / most recent failed DTC (sub-function = 0x0B reportFirstTestFailedDTC, sub-function = 0x0D reportMostRecentTestFailedDTC)

The client can retrieve the first / most recent failed DTC from the server by sending a request with the sub-function 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 only one DTC became 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 aging 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).

11.3.1.13 Retrieving the first / most recently detected confirmed DTC (sub-function = 0x0C reportFirstConfirmedDTC, subfunction = 0x0E reportMostRecentConfirmedDTC)

The client can retrieve the first / most recent confirmed DTC from the server by sending a request with the sub-function 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 became confirmed since the last time 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 became 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 became 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.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

11.3.1.14 Retrieving the list of DTCs out of the server DTC mirror memory that match a client defined status mask (sub-function = 0x0F reportMirrorMemoryDTCByStatusMask)

The handling of the sub-function reportMirrorMemoryDTCByStatusMask is identical to the handling as defined for reportDTCByStatusMask, except that all status mask checks are performed with the DTCs stored in the DTC mirror memory of the server. The DTC mirror memory is an additional optional error memory in the server that cannot be erased by the ClearDiagnosticInformation (0x14) service. The DTC mirror memory mirrors the normal DTC memory and can be used for example if the normal error memory is erased.

11.3.1.15 Retrieving mirror memory DTCExtendedData record data for a client defined DTC mask and a client defined DTCExtendedData record number out of the DTC mirror memory (sub-function = 0x10 reportMirrorMemoryDTCExtDataRecordByDTCNumber)

The handling of the sub-function reportMirrorMemoryDTCExtDataRecordByDTCNumber is identical to the handling as defined for reportDTCExtDataRecordByDTCNumber, except that the data is retrieved out of the DTC mirror memory. The DTC mirror memory is an additional optional error memory in the server that cannot be erased by the ClearDiagnosticInformation (0x14) service. The DTC mirror memory mirrors the normal DTC memory and can be used for example if the normal error memory is erased.

11.3.1.16 Retrieving the number of mirror memory DTCs that match a client defined status mask (sub-function = 0x11 reportNumberOfMirrorMemoryDTCByStatusMask)

A client can retrieve a count of the number of mirror memory DTCs matching a client defined status mask by sending a request for this service with the sub-function set to reportNumberOfMirrorMemoryDTCByStatusMask. 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.

11.3.1.17 Retrieving the number of "only emissions-related OBD" DTCs that match a client defined status mask (sub-function = 0x12 reportNumberOfEmissionsOBDDTCByStatusMask)

A client can retrieve a count of the number of "only emissions-related OBD" DTCs matching a client defined status mask by sending a request for this service with the sub-function set to reportNumberOfEmissionsOBDDTCByStatusMask. 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 "only emissions-related OBD" DTCs available in the server's memory based on the status mask provided by the client.

11.3.1.18 Retrieving the list of "only emissions-related OBD" DTCs that match a client defined status mask (sub-function = 0x13 reportEmissionsOBDDTCByStatusMask)

The client can retrieve a list of "only emissions-related OBD" DTCs, which satisfy a client defined status mask by sending a request with the sub-function byte set to reportEmissionsOBDDTCByStatusMask. This sub-function allows the client to request the server to report all "emissions-related OBD" DTCs 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 mask specified in the client’s request and the actual status associated with each "emissions-related OBD" DTC supported by the server. In addition to the DTCStatusAvailabilityMask, the server shall return all "emissions-related OBD" 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 "emissions-related OBD" 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.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

 © ISO 2013 – All rights reserved

"Emissions-related OBD" 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).

11.3.1.19 Retrieving a list of "prefailed" DTC status (sub-function = 0x14 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 can not 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.

11.3.1.20 Retrieving a list of DTCs with "permanent DTC" status (sub-function = 0x15 reportDTCWithPermanentStatus)

The client can retrieve a list of DTCs with "permanent DTC" status as described in 3.1.

11.3.1.21 Retrieving DTCExtendedData record data for a client defined DTCExtendedData record number (sub-function = 0x16 reportDTCExtDataRecordByRecordNumber)

A client can retrieve DTCExtendedData for a client defined DTCExtendedData record number by sending a request for this service with the sub-function 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 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).

11.3.1.22 Retrieving the list of WWH-OBD DTCs from a functional group that match a client defined status mask (sub-function = 0x42 reportWWHOBDDTCByMaskRecord)

The implementation and usage of DTCSeverityMask (with severity and class) is defined in ISO°27145-3 [17].

11.3.1.23 Retrieving a list of WWH-OBD DTCs with "permanent DTC" status (sub-function = 0x55 reportWWHOBDDTCWithPermanentStatus)

The client can retrieve a list of WWH-OBD DTCs with the "permanent DTC" status as described in 3.1.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

11.3.1.24 Retrieving the list of DTCs out of the server's user defined DTC memory that match a client defined DTC status mask (sub-function = 0x17 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 sub-function byte set to reportUserDefMemoryDTCByStatusMask. This sub-function 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 not be cleared upon a successful ClearDiagnosticInformation request from the client, but by a manufacturer specific routine control.

11.3.1.25 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 (sub-function = 0x18 reportUserDefMemoryDTCSnapshotRecordByDTCNumber)

A client can retrieve captured DTCSnapshot record data for a client defined DTCMaskRecord in conjunction with a DTCSnapshot record number and an user defined memory identifier by sending a request for this service with the sub-function 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 pre-defined DTCSnapshotRecord from the specific user memory in response to the client’s request, if a failure has been identified for the client defined DTCMaskRecord and DTCSnapshotRecordNumber parameters (DTCSnapshotRecordNumber unequal 0xFF) and that specific memory.

NOTE 2 The exact failure criteria shall be 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 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.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

 © ISO 2013 – All rights reserved

The server shall report one DTCSnapshot record in a single response message, except the client has set the DTCSnapshotRecordNumber to 0xFF, 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, DTCSnapshotRecordNumber, 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 DTCSnapshotRecordNumber 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).

11.3.1.26 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 (sub-function = 0x19 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 sub-function 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 pre-defined DTCExtendedData record in response to the client's request (DTCExtDataRecordNumber unequal to 0xFE or 0xFF).

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 0xFE or 0xFF, 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 is to 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).

It is in the responsibility of the vehicle manufacturer to specify the rules for the deletion of stored DTCs and DTC extended data in the user defined memory.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

11.3.2 Request message

11.3.2.1 Request message definition

Table 256 defines the structure of the ReadDTCInformation request message based on the used sub-function parameter.

Table 256 — Request message definition - sub-function = reportNumberOfDTCByStatusMask, reportDTCByStatusMask, reportMirrorMemoryDTCByStatusMask, reportNumberOfMirrorMemoryDTCByStatusMask, reportNumberOfEmissionsOBDDTCByStatusMask, reportEmissionsOBDDTCByStatusMask

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ReadDTCInformation Request SID	M	0x19	RDTCI
#2	sub-function = [reportType = reportNumberOfDTCByStatusMask reportDTCByStatusMask reportMirrorMemoryDTCByStatusMask reportNumberOfMirrorMemoryDTCByStatusMask reportNumberOfEmissionsOBDDTCByStatusMask reportEmissionsOBDDTCByStatusMask]	M	 0x01 0x02 0x0F 0x11 0x12 0x13	LEV_ RNODTCBSM RDTCBSM RMMDTCBSM RNOMMDTCBSM RNOOEBDDTCBSM ROBDDTCBSM
#3	DTCStatusMask	M	0x00 – 0xFF	DTCSM

Table 257 defines the structure of the ReadDTCInformation request message based on the used sub-function parameter.

Table 257 — Request message definition - sub-function = reportDTCSnapshotIdentification, reportDTCSnapshotRecordByDTCNumber

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ReadDTCInformation Request SID	M	0x19	RDTCI
#2	sub-function = [reportType = reportDTCSnapshotIdentification reportDTCSnapshotRecordByDTCNumber]	M	 0x03 0x04	LEV_ RDTCSSI RDTCSSBDTC
 #3 #4 #5	DTCMaskRecord[] = [DTCHighByte DTCMiddleByte DTCLowByte]	 C C C	 0x00 – 0xFF 0x00 – 0xFF 0x00 – 0xFF	DTCMREC_ DTCHB DTCMB DTCLB
#6	DTCSnapshotRecordNumber	C	0x00 – 0xFF	DTCSSRN
C: The DTCMaskRecord record and DTCSnapshotRecordNumber parameters are only present in case the sub-function parameter is equal to reportDTCSnapshotRecordByDTCNumber.				

Table 258 defines the structure of the ReadDTCInformation request message based on the used sub-function parameter.

Table 258 — Request message definition - sub-function = reportDTCStoredDataByRecordNumber

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ReadDTCInformation Request SID	M	0x19	RDTCI
#2	sub-function = [reportType = reportDTCStoredDataByRecordNumber]	M	 0x05	LEV_ RDTCSDBRN
#3	DTCStoredDataRecordNumber	M	0x00 – 0xFF	DTCSDRN

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 259 defines the structure of the ReadDTCInformation request message based on the used sub-function parameter.

Table 259 — Request message definition - sub-function = reportDTCExtDataRecordByDTCNumber, reportMirrorMemoryDTCExtDataRecordByDTCNumber

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ReadDTCInformation Request SID	M	0x19	RDTCI
#2	sub-function = [reportType = reportDTCExtDataRecordByDTCNumber reportMirrorMemoryDTCExtDataRecordByDTCNumber]	M	 0x06 0x10	LEV_ RDTCEDRBDN RMDEDRBDN
 #3 #4 #5	DTCMaskRecord[] = [DTCHighByte DTCMiddleByte DTCLowByte]	 M M M	 0x00 – 0xFF 0x00 – 0xFF 0x00 – 0xFF	DTCMREC_ DTCHB DTCMB DTCLB
#6	DTCExtDataRecordNumber	M	0x00 – 0xFF	DTCEDRN

Table 260 defines the structure of the ReadDTCInformation request message based on the used sub-function parameter.

Table 260 — Request message definition - sub-function = reportNumberOfDTCBySeverityMaskRecord, reportDTCSeverityInformation

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ReadDTCInformation Request SID	M	0x19	RDTCI
#2	sub-function = [reportType = reportNumberOfDTCBySeverityMaskRecord reportDTCBySeverityMaskRecord]	M	 0x07 0x08	LEV_ RNODTCBSMR RDTCBSMR
 #3 #4	DTCSeverityMaskRecord[] = [DTCSeverityMask DTCStatusMask]	 M M	 0x00 – 0xFF 0x00 – 0xFF	DTCSVMREC_ DTCSVM DTCSM

Table 261 defines the structure of the ReadDTCInformation request message based on the used sub-function parameter.

Table 261 — Request message definition - sub-function = reportSeverityInformationOfDTC

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ReadDTCInformation Request SID	M	0x19	RDTCI
#2	sub-function = [reportType = reportSeverityInformationOfDTC]	M	 0x09	LEV_ RSIODTC
 #3 #4 #5	DTCMaskRecord[] = [DTCHighByte DTCMiddleByte DTCLowByte]	 M M M	 0x00 – 0xFF 0x00 – 0xFF 0x00 – 0xFF	DTCMREC_ DTCHB DTCMB DTCLB

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 262 defines the structure of the ReadDTCInformation request message based on the used sub-function parameter.

Table 262 — Request message definition - sub-function = reportSupportedDTC, reportFirstTestFailedDTC, reportFirstConfirmedDTC, reportMostRecentTestFailedDTC, reportMostRecentConfirmedDTC, reportDTCFaultDetectionCounter, reportDTCWithPermanentStatus

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ReadDTCInformation Request SID	M	0x19	RDTCI
#2	sub-function = [reportType = reportSupportedDTC reportFirstTestFailedDTC reportFirstConfirmedDTC reportMostRecentTestFailedDTC reportMostRecentConfirmedDTC reportDTCFaultDetectionCounter reportDTCWithPermanentStatus]	M	 0x0A 0x0B 0x0C 0x0D 0x0E 0x14 0x15	LEV_ RSUPDTC RFTFDTC RFCDTC RMRTFDTC RMRCDTC RDTCFDC RDTCWPS

Table 263 defines the structure of the ReadDTCInformation request message based on the used sub-function parameter.

Table 263 — Request message definition - sub-function = reportDTCExtDataRecordByRecordNumber

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ReadDTCInformation Request SID	M	0x19	RDTCI
#2	sub-function = [reportType = reportDTCExtDataRecordByRecordNumber]	M	0x16	LEV_ RDTCEDRBR
#3	DTCExtDataRecordNumber	M	0x00 – 0xEF	DTCEDRN

Table 264 defines the structure of the ReadDTCInformation request message based on the used sub-function parameter.

Table 264 — Request message definition - sub-function = reportUserDefMemoryDTCByStatusMask

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ReadDTCInformation Request SID	M	0x19	RDTCI
#2	sub-function = [reportType = reportUserDefMemoryDTCByStatusMask	M	0x17	LEV_ RUDMDTCBSM
#3	DTCStatusMask	M	0x00 –0xFF	DTCSM
#4	MemorySelection	M	0x00 – 0xFF	MEMYS

Table 265 defines the structure of the ReadDTCInformation request message based on the used sub-function parameter.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 265 — Request message definition - sub-function = reportUserDefMemoryDTCSnapshotRecordByDTCNumber

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ReadDTCInformation Request SID	M	0x19	RDTCI
#2	sub-function = [reportType = reportUserDefMemoryDTCSnapshotRecordByDTCNumber]	M	0x18	LEV_ RUDMDTCSSBDTC
	DTCMaskRecord[] = [			DTCMREC_
#3	DTCHighByte	M	0x00 – 0xFF	DTCHB
#4	DTCMiddleByte	M	0x00 – 0xFF	DTCMB
#5	DTCLowByte]	M	0x00 – 0xFF	DTCLB
#6	DTCSnapshotRecordNumber	M	0x00 – 0xFF	DTCSSRN
#7	MemorySelection	M	0x00 – 0xFF	MEMYS

Table 266 defines the structure of the ReadDTCInformation request message based on the used sub-function parameter.

Table 266 — Request message definition - sub-function = reportUserDefMemoryDTCExtDataRecordByDTCNumber

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ReadDTCInformation Request SID	M	0x19	RDTCI
#2	sub-function = [reportType = reportUserDefMemoryDTCExtDataRecordByDTCNumber]	M	0x19	LEV_ RUDMDTCEDRBDN
	DTCMaskRecord[] = [			DTCMREC_
#3	DTCHighByte	M	0x00 – 0xFF	DTCHB
#4	DTCMiddleByte	M	0x00 – 0xFF	DTCMB
#5	DTCLowByte]	M	0x00 – 0xFF	DTCLB
#6	DTCExtDataRecordNumber	M	0x00 – 0xFF	DTCEDRN
#7	MemorySelection	M	0x00 – 0xFF	MEMYS

Table 267 defines the structure of the ReadDTCInformation request message based on the used sub-function parameter.

Table 267 — Request message definition - sub-function = reportWWHOBDDTCByMaskRecord

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ReadDTCInformation Request SID	M	0x19	RDTCI
#2	sub-function = [reportType = reportWWHOBDDTCByMaskRecord]	M	0x42	LEV_ ROBDDTCBMR
#3	FunctionalGroupIdentifier	M	0x00 – 0xFF	FGID
	DTCSeverityMaskRecord[] = [			DTCSVMREC_
#4	DTCStatusMask	M	0x00 – 0xFF	DTCSM
#5	DTCSeverityMask]	M	0x00 – 0xFF	DTCSVM

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 268 defines the structure of the ReadDTCInformation request message based on the used sub-function parameter.

Table 268 — Request message definition - sub-function = reportWWHOBDDTCWithPermanentStatus

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ReadDTCInformation Request SID	M	0x19	RDTCI
#2	sub-function = [reportType = reportWWHOBDDTCWithPermanentStatus]	M	0x55	LEV_ RWWHOBDDTCWPS
#3	FunctionalGroupIdentifier	M	0x00 – 0xFF	FGID

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

The sub-function parameters are used by this service to select one of the DTC report types specified in Table 269. Explanations and usage of the possible levels are detailed below (suppressPosRspMsgIndicationBit (bit 7) not shown).

Table 269 — Request message sub-function definition

Bits 6 – 0	Description	Cvt	Mnemonic
0x00	**ISOSAEReserved** This value is reserved by this document for future definition.	M	ISOSAERESRVD
0x01	**reportNumberOfDTCByStatusMask** This parameter specifies that the server shall transmit to the client the number of DTCs matching a client defined status mask.	U	RNODTCBSM
0x02	**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
0x03	**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
0x04	**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 (0xFF for all records).	U	RDTCSSBDTC
0x05	**reportDTCStoredDataByRecordNumber** This parameter specifies that the server shall transmit to the client the DTCStoredDatarecord(s) associated with a client defined DTCStoredData record number (0xFF for all records).	U	RDTCSDBRN
0x06	**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 (0xFF for all records, 0xFE for all OBD records).	U	RDTCEDRBDN
0x07	**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

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 269 — *(continued)*

Bits 6 – 0	Description	Cvt	Mnemonic
0x08	**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
0x09	**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
0x0A	**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
0x0B	**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 sub-function parameter shall be independent of whether or not the DTC was confirmed or aged.	U	RFTFDTC
0x0C	**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 sub-function 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
0x0D	**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 sub-function parameter shall be independent of whether or not the DTC was confirmed or aged.	U	RMRTFDTC
0x0E	**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 sub-function 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
0x0F	**reportMirrorMemoryDTCByStatusMask** This parameter specifies that the server shall transmit to the client a list of DTCs out of the DTC mirror memory and corresponding statuses matching a client defined status mask.	U	RMMDTCBSM

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 269 — *(continued)*

Bits 6 – 0	Description	Cvt	Mnemonic
0x10	**reportMirrorMemoryDTCExtDataRecordByDTCNumber** This parameter specifies that the server shall transmit to the client the DTCExtendedData record(s) - out of the DTC mirror memory - associated with a client defined DTC number and DTCExtendedData record number (0xFF for all records, 0xFE for all OBD records) DTCs.	U	RMMDEDRBDN
0x11	**reportNumberOfMirrorMemoryDTCByStatusMask** This parameter specifies that the server shall transmit to the client the number of DTCs out of mirror memory matching a client defined status mask.	U	RNOMMDTCBSM
0x12	**reportNumberOfEmissionsOBDDTCByStatusMask** This parameter specifies that the server shall transmit to the client the number of emissions-related OBD DTCs matching a client defined status mask. The number of OBD DTCs reported shall only be those which are required to be compatible with emissions-related legal requirements.	U	RNOOEOBDDTCBSM
0x13	**reportEmissionsOBDDTCByStatusMask** This parameter specifies that the server shall transmit to the client a list of emissions-related OBD DTCs and corresponding statuses matching a client defined status mask. The list of OBD DTCs reported shall only be those which are required to be compatible with emissions-related legal requirements.	U	ROBDDTCBSM
0x14	**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 (0x7F) 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., 0x01 – 0x7E) 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 value less than +127 (0x7F) 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.	U	RDTCFDC

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 269 — *(continued)*

Bits 6 – 0	Description	Cvt	Mnemonic
0x15	**reportDTCWithPermanentStatus** This parameter specifies that the server shall transmit to the client a list of DTCs with "permanent DTC" status as described in 3.1.	U	RDTCWPS
0x16	**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 0xF0.	U	RDTCEDBR
0x17	**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
0x18	**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 (0xFF for all records).	U	RUDMDTCSSBDTC
0x19	**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 (0xFF for all records).	U	RUDMDTCEDRBDN
0x1A – 0x41	**ISOSAEReserved** This value is reserved by this document for future definition.	M	ISOSAERESRVD
0x42	**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
0x43 – 0x54	**ISOSAEReserved** This value is reserved by this document for future definition.	M	ISOSAERESRVD
0x55	**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 3.1.	U	RWWHOBDDTCWPS
0x56 – 0x7F	**ISOSAEReserved** This value is reserved by this document for future definition.	M	ISOSAERESRVD

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

11.3.2.3 Request message data-parameter definition

Table 270 specifies the data-parameters of the request message.

Table 270 — 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 [12] 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 part of ISO 14229 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 [19] 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 [5] 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 [16] specification. This format is identified by the DTCFormatIdentifier = SAE_J2012-DA_WWH-OBD_DTCFormat.
DTCSnapshotRecordNumber DTCSnapshotRecordNumber is a 1-byte value indicating the number of the specific DTCSnapshot data record requested for a client defined DTCMaskRecord via the reportDTCSnapshotByDTCNumber sub-function. DTCSnapshot data record number 0x00 shall be reserved for legislated purposes (e.g., WWH-OBD). DTCSnapshot records in range of 0x01 through 0xFE shall be available for vehicle manufacturer specific usage. A value of 0xFF 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 sub-function. DTCStoredDataRecordNumber 0x00 shall be reserved for legislated purposes. DTCStoredData records in range of 0x01 through 0xFE shall be available for vehicle manufacturer specific usage. A value of 0xFF requests the server to report all stored DTCStoredData data records at once.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

 © ISO 2013 – All rights reserved

Table 270 — *(continued)*

Definition
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 sub-function. For emissions-related servers (OBD compliant ECUs) the DTCExtDataRecordNumber 0x00 shall be reserved for future OBD use. The following DTCExtDataRecordNumber ranges are reserved: — A value of 0x00 is reserved by ISO/SAE. — A value of 0x01 – 0x8F requests the server to report the vehicle manufacturer specific stored DTCExtendedData records. — A value of 0x90 – 0xEF requests the server to report legislated OBD stored DTCExtendedData records. — A value of 0xF0 – 0xFD is reserved by ISO/SAE for future reporting of groups in a single response message. — A value of 0xFE requests the server to report all legislated OBD stored DTCExtendedData records in a single response message. — A value of 0xFF requests the server to report all stored DTCExtendedData records in a single response message.
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.

11.3.3 Positive response message

11.3.3.1 Positive response message definition

Positive response(s) to the service ReadDTCInformation requests depend on the sub-function in the service request.

Table 271 defines the positive response message format of the sub-function parameter.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 271 — Response message definition - sub-function = reportNumberOfDTCByStatusMask, reportNumberOfDTCBySeverityMaskRecord, reportNumberOfMirrorMemoryDTCByStatusMask, reportNumberOfEmissionsOBDDTCByStatusMask

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ReadDTCInformation Response SID	M	0x59	RDTCIPR
#2	reportType = [reportNumberOfDTCByStatusMask reportNumberOfDTCBySeverityMaskRecord reportNumberOfMirrorMemoryDTCByStatusMask reportNumberOfEmissionsOBDDTCByStatusMask]	M	 0x01 0x07 0x11 0x12	LEV_ RNODTCBSM RNODTCBSMR RNOMMDTCBSM RNOOEOBDDTCBSM
#3	DTCStatusAvailabilityMask	M	0x00 – 0xFF	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	 0x00 0x01 0x02 0x03 0x04	DTCFID_ J2012-DADTCF00 14229-1DTCF J1939-73DTCF 11992-4DTCF J2012-DADTCF04
 #5 #6	DTCCount[] = [DTCCountHighByte DTCCountLowByte]	 M M	 0x00 – 0xFF 0x00 – 0xFF	DTCC_ DTCCHB DTCCLB

Table 272 defines the positive response message format of the sub-function parameter.

Table 272 — Response message definition - sub-function = reportDTCByStatusMask, reportSupportedDTCs, reportFirstTestFailedDTC, reportFirstConfirmedDTC, reportMostRecentTestFailedDTC, reportMostRecentConfirmedDTC, reportMirrorMemoryDTCByStatusMask, reportEmissionsOBDDTCByStatusMask, reportDTCWithPermanentStatus

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ReadDTCInformation Response SID	M	0x59	RDTCIPR
#2	reportType = [reportDTCByStatusMask reportSupportedDTCs reportFirstTestFailedDTC reportFirstConfirmedDTC reportMostRecentTestFailedDTC reportMostRecentConfirmedDTC reportMirrorMemoryDTCByStatusMask reportEmissionsOBDDTCByStatusMask reportDTCWithPermanentStatus]	M	 0x02 0x0A 0x0B 0x0C 0x0D 0x0E 0x0F 0x13 0x15	LEV_ RDTCBSM RSUPDTC RFTFDTC RFCDTC RMRTFDTC RMRCDTC RMMDTCBSM ROBDDTCBSM RDTCWPS
#3	DTCStatusAvailabilityMask	M	0x00 – 0xFF	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	 0x00 – 0xFF 0x00 – 0xFF 0x00 – 0xFF 0x00 – 0xFF 0x00 – 0xFF 0x00 – 0xFF 0x00 – 0xFF 0x00 – 0xFF : 0x00 – 0xFF 0x00 – 0xFF 0x00 – 0xFF 0x00 – 0xFF	DTCASR_ DTCHB DTCMB DTCLB SODTC DTCHB DTCMB DTCLB SODTC : DTCHB DTCMB DTCLB SODTC

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 272 — *(continued)*

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
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, reportMirrorMemoryDTCByStatusMask, reportEmissionsOBDDTCByStatusMask, reportDTCWithPermanentStatus and more than one DTC information is available to be reported.				

Table 273 defines the positive response message format of the sub-function parameter.

Table 273 — Response message definition - sub-function = reportSnapshotIdentification

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ReadDTCInformation Response SID	M	0x59	RDTCIPR
#2	reportType = [reportDTCSnapshotIdentification]	M	0x03	LEV_ RDTCSSI
#3 #4 #5	DTCRecord[]#1 = [DTCHighByte#1 DTCMiddleByte#1 DTCLowByte#1]	C_1 C_1 C_1	0x00 – 0xFF 0x00 – 0xFF 0x00 – 0xFF	DTCASR_ DTCHB DTCMB DTCLB
#6	DTCSnapshotRecordNumber#1	C_1	0x00 – 0xFF	DTCSSRN
:	:	:	:	:
#n-3 #n-2 #n-1	DTCRecord[]#m = [DTCHighByte#m DTCMiddleByte#m DTCLowByte#m]	C_2 C_2 C_2	0x00 – 0xFF 0x00 – 0xFF 0x00 – 0xFF	DTCASR_ DTCHB DTCMB DTCLB
#n	DTCSnapshotRecordNumber#m	C_2	0x00 – 0xFF	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 274 defines the positive response message format of the sub-function parameter.

Table 274 — Response message definition - sub-function = reportDTCSnapshotRecordByDTCNumber

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ReadDTCInformation Response SID	M	0x59	RDTCIPR
#2	reportType = [reportDTCSnapshotRecordByDTCNumber]	M	0x04	LEV_ RDTCSSBDTC
#3 #4 #5 #6	DTCAndStatusRecord[] = [DTCHighByte DTCMiddleByte DTCLowByte statusOfDTC]	M M M M	0x00 – 0xFF 0x00 – 0xFF 0x00 – 0xFF 0x00 – 0xFF	DTCASR_ DTCHB DTCMB DTCLB SODTC
#7	DTCSnapshotRecordNumber#1	C_1	0x00 – 0xFF	DTCSSRN
#8	DTCSnapshotRecordNumberOfIdentifiers#1	C_1	0x00 – 0xFF	DTCSSRNI

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 274 — *(continued)*

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
	DTCSnapshotRecord[]#1 = [			DTCSSR_
#9	dataIdentifier#1 byte#1 (MSB)	C_1	0x00 – 0xFF	DIDB11
#10	dataIdentifier#1 byte#2 (LSB)	C_1	0x00 – 0xFF	DIDB12
#11	snapshotData#1 byte#1	C_1	0x00 – 0xFF	SSD11
:	:	C_1	:	:
# 11+(p-1)	snapshotData#1 byte#p	C_1	0x00 – 0xFF	SSD1p
:	:	:	:	:
#r-(m-1)-2	dataIdentifier#w byte#1 (MSB)	C_2	0x00 – 0xFF	DIDB21
#r-(m-1)-1	dataIdentifier#w byte#2 (LSB)	C_2	0x00 – 0xFF	DIDB22
#r-(m-1)	snapshotData#w byte#1	C_2	0x00 – 0xFF	SSD21
:	:	C_2	:	:
#r	snapshotData#w byte#m]	C_2	0x00 – 0xFF	SSD2m
:	:	:	:	:
#t	DTCSnapshotRecordNumber#x	C_3	0x00 – 0xFF	DTCSSRN
#t+1	DTCSnapshotRecordNumberOfIdentifiers#x	C_3	0x00 – 0xFF	DTCSSRNI
	DTCSnapshotRecord[]#x = [			DTCSSR_
#t+2	dataIdentifier#1 byte#1 (MSB)	C_3	0x00 – 0xFF	DIDB11
#t+3	dataIdentifier#1 byte#2 (LSB)	C_3	0x00 – 0xFF	DIDB12
#t+5	snapshotData#1 byte#1	C_3	0x00 – 0xFF	SSD11
:	:	C_3	:	:
#t+5+(p-1)	snapshotData#1 byte#p	C_3	0x00 – 0xFF	SSD1p
:	:	:	:	:
#n-(u-1)-2	dataIdentifier#w byte#1 (MSB)	C_4	0x00 – 0xFF	DIDB21
#n-(u-1)-1	dataIdentifier#w byte#2 (LSB)	C_4	0x00 – 0xFF	DIDB22
#n-(u-1)	snapshotData#w byte#1	C_4	0x00 – 0xFF	SSD21
:	:	C_4	:	:
#n	snapshotData#w byte#u]	C_4	0x00 – 0xFF	SSD2u

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 e.g. 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 0xFF in the request) and more than one record is available to be reported.

Table 275 defines the positive response message format of the sub-function parameter.

Table 275 — Response message definition - sub-function = reportDTCStoredDataByRecordNumber

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ReadDTCInformation Response SID	M	0x59	RDTCIPR
#2	reportType = [reportDTCStoredDataByRecordNumber]	M	0x05	LEV_ RDTCSDBRN
#3	DTCStoredDataRecordNumber#1	M	0x00 – 0xFF	DTCSDRN
	DTCAndStatusRecord[]#1 = [			DTCASR_
#4	DTCHighByte	C_1	0x00 – 0xFF	DTCHB
#5	DTCMiddleByte	C_1	0x00 – 0xFF	DTCMB
#6	DTCLowByte	C_1	0x00 – 0xFF	DTCLB
#7	statusOfDTC]	C_1	0x00 – 0xFF	SODTC

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 275 — *(continued)*

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#8	DTCStoredDataRecordNumberOfIdentifiers#1	C_1	0x00 – 0xFF	DTCSDRNI
	DTCStoredDataRecord[]#1 = [			DTCSDR_
#9	dataIdentifier#1 byte#1 (MSB)	C_1	0x00 – 0xFF	DIDB11
#10	dataIdentifier#1 byte#2 (LSB)	C_1	0x00 – 0xFF	DIDB12
#11	DTCstoredData#1 byte#1	C_1	0x00 – 0xFF	DSD11
:	:	:	:	:
#11+(p-1)	DTCstoredData#1 byte#p	C_1	0x00 – 0xFF	DSD1p
:	:	:	:	:
#r-(m-1)-2	dataIdentifier#w byte#1 (MSB)	C_2	0x00 – 0xFF	DIDB21
#r-(m-1)-1	dataIdentifier#w byte#2 (LSB)	C_2	0x00 – 0xFF	DIDB22
#r-(m-1)	DTCstoredData#w byte#1	C_2	0x00 – 0xFF	DSD21
:	:	:	:	:
#r	DTCstoredData#w byte#m]	C_2	0x00 – 0xFF	DSD2m
:	:	:	:	:
#t	DTCStoredDataRecordNumber#x	C_3	0x00 – 0xFF	DTCSDRN
	DTCAndStatusRecord[]#x = [			DTCASR_
#t+1	DTCHighByte	C_3	0x00 – 0xFF	DTCHB
#t+2	DTCMiddleByte	C_3	0x00 – 0xFF	DTCMB
#t+3	DTCLowByte	C_3	0x00 – 0xFF	DTCLB
#t+4	statusOfDTC]	C_3	0x00 – 0xFF	SODTC
#t+5	DTCStoredDataRecordNumberOfIdentifiers#x	C_3	0x00 – 0xFF	DTCSDRNI
	DTCStoredDataRecord[]#x = [			DTCSDR_
#t+6	dataIdentifier#1 byte#1 (MSB)	C_3	0x00 – 0xFF	DIDB11
#t+7	dataIdentifier#1 byte#2 (LSB)	C_3	0x00 – 0xFF	DIDB12
#t+8	DTCstoredData#1 byte#1	C_3	0x00 – 0xFF	DSD11
:	:	:	:	:
#t+8+(p-1)	DTCstoredDataa#1 byte#p	C_3	0x00 – 0xFF	DSD1p
:	:	:	:	:
#n-(u-1)-2	dataIdentifier#w byte#1 (MSB)	C_4	0x00 – 0xFF	DIDB21
#n-(u-1)-1	dataIdentifier#w byte#2 (LSB)	C_4	0x00 – 0xFF	DIDB22
#n-(u-1)	DTCstoredData#w byte#1	C_4	0x00 – 0xFF	DSD21
:	:	:	:	:
#n	DTCstoredData#w byte#u]	C_4	0x00 – 0xFF	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 e.g. 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 0xFF in the request) and more than one record is available to be reported.

Table 276 defines the positive response message format of the sub-function parameter.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 276 — Response message definition - sub-function = reportDTCExtDataRecordByDTCNumber and reportMirrorMemoryDTCExtDataRecordByDTCNumber

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ReadDTCInformation Response SID	M	0x59	RDTCIPR
#2	reportType = [reportDTCExtDataRecordByDTCNumber reportMirrorMemoryDTCExtDataRecordByDTCNumber]	M	 0x06 0x10	LEV_ RDTCEDRBD RMDEDRBDN
 #3 #4 #5 #6	DTCAndStatusRecord[] = [DTCHighByte DTCMiddleByte DTCLowByte statusOfDTC]	 M M M M	 0x00 – 0xFF 0x00 – 0xFF 0x00 – 0xFF 0x00 – 0xFF	DTCASR_ DTCHB DTCMB DTCLB SODTC
#7	DTCExtDataRecordNumber#1	C_1	0x00-0xFD	DTCEDRN
 #8 : #8+(p-1)	DTCExtDataRecord[]#1 = [extendedData#1 byte#1 : extendedData#1 byte#p]	 C_1 C_1 C_1	 0x00 – 0xFF : 0x00 – 0xFF	DTCSSR_ EDD11 : EDD1p
:	:	:	:	:
#t	DTCExtDataRecordNumber#x	C_2	0x00 – 0xFD	DTCEDRN
 #t+1 : #t+1+(q-1)	DTCExtDataRecord[]#x = [extendedData#x byte#1 : extendedData#x byte#q]	 C_2 C_2 C_2	 0x00 – 0xFF : 0x00 – 0xFF	DTCSSR_ EDDx1 : 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 0xFE or 0xFF in the request) and more than one record is available to be reported.				

Table 277 defines the positive response message format of the sub-function parameter.

Table 277 — Response message definition - sub-function = reportDTCBySeverityMaskRecord, reportSeverityInformationOfDTC

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ReadDTCInformation Response SID	M	0x59	RDTCIPR
#2	reportType = [reportDTCBySeverityMaskRecord reportSeverityInformationOfDTC]	M	 0x08 0x09	LEV_ RDTCBSMR RSIODTC
#3	DTCStatusAvailabilityMask	M	0x00 – 0xFF	DTCSAM

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 277 — *(continued)*

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
	DTCAndSeverityRecord[] = [			DTCASR_
#4	DTCSeverity#1	C_1	0x00 – 0xFF	DTCS
#5	DTCFunctionalUnit#1	C_1	0x00 – 0xFF	DTCFU
#6	DTCHighByte#1	C_1	0x00 – 0xFF	DTCHB
#7	DTCMiddleByte#1	C_1	0x00 – 0xFF	DTCMB
#8	DTCLowByte#1	C_1	0x00 – 0xFF	DTCLB
#9	statusOfDTC#1	C_1	0x00 – 0xFF	SODTC
:	:	:	:	:
#n-5	DTCSeverity#m	C_2	0x00 – 0xFF	DTCS
#n-4	DTCFunctionalUnit#m	C_2	0x00 – 0xFF	DTCFU
#n-3	DTCHighByte#m	C_2	0x00 – 0xFF	DTCHB
#n-2	DTCMiddleByte#m	C_2	0x00 – 0xFF	DTCMB
#n-1	DTCLowByte#m	C_2	0x00 – 0xFF	DTCLB
#n	statusOfDTC#m]	C_2	0x00 – 0xFF	SODTC
C_1: In case of reportDTCBySeverityMaskRecord this parameter has to be present if at least one DTC matches the client defined DTC severity mask. In case of reportSeverityInformationOfDTC this parameter has to 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 has to be present if more than one DTC matches the client defined DTC severity mask.				

Table 278 defines the positive response message format of the sub-function parameter.

Table 278 — Response message definition - sub-function = reportDTCFaultDetectionCounter

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ReadDTCInformation Response SID	M	0x59	RDTCIPR
#2	reportType = [reportDTCFaultDetectionCounter]	M	0x14	LEV_ RDTCFDC
	DTCFaultDetectionCounterRecord[] = [			DTCFDCR_
#3	DTCHighByte#1	C_1	0x00 – 0xFF	DTCHB
#4	DTCMiddleByte#1	C_1	0x00 – 0xFF	DTCMB
#5	DTCLowByte#1	C_1	0x00 – 0xFF	DTCLB
#6	DTCFaultDetectionCounter#1	C_1	0x01 – 0xFF	DTCFDC
#7	DTCHighByte#2	C_2	0x00 – 0xFF	DTCHB
#8	DTCMiddleByte#2	C_2	0x00 – 0xFF	DTCLB
#9	DTCLowByte#2	C_2	0x00 – 0xFF	DTCFT
#10	DTCFaultDetectionCounter#2	C_2	0x01 – 0xFF	DTCFDC
#n-3	:	:	:	:
#n-2	DTCHighByte#m	C_2	0x00 – 0xFF	DTCHB
#n-1	DTCMiddleByte#m	C_2	0x00 – 0xFF	DTCMB
#n	DTCLowByte#m	C_2	0x00 – 0xFF	DTCLB
	DTCFaultDetectionCounter#m]	C_2	0x01 – 0xFF	DTCFDC
C_1: This parameter is only present if at least one DTC has a DTCFaultDetectionCounter with a positive value less than 0x7F. C_2: This parameter record is only present if more than one DTC has a DTCFaultDetectionCounter with a positive value less than 0x7F.				

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 279 defines the positive response message format of the sub-function parameter.

Table 279 — Response message definition - sub-function = reportDTCExtDataRecordByRecordNumber

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ReadDTCInformation Response SID	M	0x59	RDTCIPR
#2	reportType = [reportDTCExtDataRecordByRecordNumber]	M	0x16	LEV_ RDTCEDRBR
#3	DTCExtDataRecordNumber	M	0x00 – 0xEF	DTCEDRN
	DTCAndStatusRecord[]#1 = [			DTCASR_
#4	DTCHighByte	C_1	0x00 – 0xFF	DTCHB
#5	DTCMiddleByte	C_1	0x00 – 0xFF	DTCMB
#6	DTCLowByte	C_1	0x00 – 0xFF	DTCLB
#7	statusOfDTC]	C_1	0x00 – 0xFF	SODTC
	DTCExtDataRecord[]#1= [			DTCEDR_
#8	extendedData#1 byte#1	C_1	0x00 – 0xFF	EDD11
:	:	:	:	:
#8+(p-1)	extendedData#1 byte#p]	C_1	0x00 – 0xFF	EDD1p
:	:	:	:	:
	DTCAndStatusRecord[]#x = [			DTCSSR
#t	DTCHighByte	C_2	0x00 – 0xFF	
#t+1	DTCMiddleByte	C_2	0x00 – 0xFF	
#t+2	DTCLowByte	C_2	0x00 – 0xFF	
#t+3	statusOfDTC]	C_2	0x00 – 0xFF	
	DTCExtDataRecord[]#x = [			DTCEDR_
#t+4	extendedData#x byte#1	C_2	0x00 – 0xFF	EDDx1
:	:	:	:	:
#t+4+(p-1)	extendedData#x byte#p]	C_2	0x00 – 0xFF	EDDxp

C_1: The DTCAndStatusRecord and the DTCExtDataRecord parameters are only present if at least one DTCExtDataRecord is available to be reported.

C_2: 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 280 defines the positive response message format of the sub-function parameter.

Table 280 — Response message definition - sub-function = 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-FF	MEMYS
#4	DTCStatusAvailabilityMask	M	00-FF	DTCSAM

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 280 — *(continued)*

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
	DTCAndStatusRecord[] = [			DTCASR_
#5	DTCHighByte#1	C_1	00-FF	DTCHB
#6	DTCMiddleByte#1	C_1	00-FF	DTCMB
#7	DTCLowByte#1	C_1	00-FF	DTCLB
#8	statusOfDTC#1	C_1	00-FF	SODTC
#9	DTCHighByte#2	C_2	00-FF	DTCHB
#10	DTCMiddleByte#2	C_2	00-FF	DTCMB
#11	DTCLowByte#2	C_2	00-FF	DTCLB
#12	statusOfDTC#2	C_2	00-FF	SODTC
:	:	:	:	:
#n-3	DTCHighByte#m	C_2	00-FF	DTCHB
#n-2	DTCMiddleByte#m	C_2	00-FF	DTCMB
#n-1	DTCLowByte#m	C_2	00-FF	DTCLB
#n	statusOfDTC#m]	C_2	00-FF	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 281 defines the positive response message format of the sub-function parameter.

Table 281 — Response message definition - sub-function = reportUserDefMemoryDTCSnapshotRecordByDTCNumber

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ReadDTCInformation Response SID	M	0x59	RDTCIPR
#2	reportType = [reportUserDefMemoryDTCSnapshotRecordByDTCNumber]	M	0x18	LEV_ RUDMDTCSSBDTC
#3	MemorySelection	M	0x00-0xFF	MEMYS
	DTCAndStatusRecord[] = [			DTCASR_
#4	DTCHighByte	M	0x00 – 0xFF	DTCHB
#5	DTCMiddleByte	M	0x00 – 0xFF	DTCMB
#6	DTCLowByte	M	0x00 – 0xFF	DTCLB
#7	statusOfDTC]	M	0x00 – 0xFF	SODTC
#8	DTCSnapshotRecordNumber#1	C_1	0x00 – 0xFF	DTCSSRN
#9	DTCSnapshotRecordNumberOfIdentifiers#1	C_1	0x00 – 0xFF	DTCSSRNI
	DTCSnapshotRecord[]#1 = [			DTCSSR_
#10	dataIdentifier#1 byte#1 (MSB)	C_1	0x00 – 0xFF	DIDB11
#11	dataIdentifier#1 byte#2 (LSB)	C_1	0x00 – 0xFF	DIDB12
#12	snapshotData#1 byte#1	C_1	0x00 – 0xFF	SSD11
:	:	C_1	:	:
# 12+(p-1)	snapshotData#1 byte#p	C_1	0x00 – 0xFF	SSD1p
:	:	:	:	:
#r-(m-1)-2	dataIdentifier#w byte#1 (MSB)	C_2	0x00 – 0xFF	DIDB21
#r-(m-1)-1	dataIdentifier#w byte#2 (LSB)	C_2	0x00 – 0xFF	DIDB22
#r-(m-1)	snapshotData#w byte#1	C_2	0x00 – 0xFF	SSD21
:	:	C_2	:	:
#r	snapshotData#w byte#m]	C_2	0x00 – 0xFF	SSD2m
:	:	:	:	:

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 281 — *(continued)*

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#t	DTCSnapshotRecordNumber#x	C_3	0x00 – 0xFF	DTCSSRN
#t+1	DTCSnapshotRecordNumberOfIdentifiers#x	C_3	0x00 – 0xFF	DTCSSRNI
	DTCSnapshotRecord[]#x = [			DTCSSR_
#t+2	dataIdentifier#1 byte#1 (MSB)	C_3	0x00 – 0xFF	DIDB11
#t+3	dataIdentifier#1 byte#2 (LSB)	C_3	0x00 – 0xFF	DIDB12
#t+5	snapshotData#1 byte#1	C_3	0x00 – 0xFF	SSD11
:	:	C_3	:	:
#t+5+(p-1)	snapshotData#1 byte#p	C_3	0x00 – 0xFF	SSD1p
:	:	:	:	:
#n-(u-1)-2	dataIdentifier#w byte#1 (MSB)	C_4	0x00 – 0xFF	DIDB21
#n-(u-1)-1	dataIdentifier#w byte#2 (LSB)	C_4	0x00 – 0xFF	DIDB22
#n-(u-1)	snapshotData#w byte#1	C_4	0x00 – 0xFF	SSD21
:	:	C_4	:	:
#n	snapshotData#w byte#u]	C_4	0x00 – 0xFF	SSD2u

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 e.g. 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 0xFF in the request) and more than one record is available to be reported.

Table 282 defines the positive response message format of the sub-function parameter.

Table 282 — Response message definition - sub-function = reportUserDefMemoryDTCExtDataRecordByDTCNumber

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ReadDTCInformation Response SID	M	0x59	RDTCIPR
#2	reportType = [reportUserDefMemoryDTCExtDataRecordByDTCNumber]	M	0x19	LEV_ RUDMDTCEDRBDN
#3	MemorySelection	M	0x00-0xFF	MEMYS
	DTCAndStatusRecord[] = [			DTCASR_
#4	DTCHighByte	M	0x00 – 0xFF	DTCHB
#5	DTCMiddleByte	M	0x00 – 0xFF	DTCMB
#6	DTCLowByte	M	0x00 – 0xFF	DTCLB
#7	statusOfDTC]	M	0x00 – 0xFF	SODTC
#8	DTCExtDataRecordNumber#1	C_1	0x00-0xFE	DTCEDRN
	DTCExtDataRecord[]#1 = [			DTCSSR_
#9	extendedData#1 byte#1	C_1	0x00 – 0xFF	EDD11
:	:	C_1	:	:
#9+(p-1)	extendedData#1 byte#p]	C_1	0x00 – 0xFF	EDD1p
:	:	:	:	:

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 282 — *(continued)*

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
	DTCExtDataRecord[]#x = [			DTCSSR_
#t+1	extendedData#x byte#1	C_2	0x00 – 0xFF	EDDx1
:	:	C_2	:	:
#t+1+(q-1)	extendedData#x byte#q]	C_2	0x00 – 0xFF	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 0xFE or 0xFF in the request) and more than one record is available to be reported.				

Table 283 defines the positive response message format of the sub-function parameter.

Table 283 — Response message definition - sub-function = reportWWHOBDDTCByMaskRecord

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ReadDTCInformation Response SID	M	0x59	RDTCIPR
#2	reportType = [reportWWHOBDDTCByMaskRecord]	M	0x42	LEV_ RWWHOBDDTCBSMR
#3	FunctionalGroupIdentifier	M	0x00 – 0xFF	FGID
#4	DTCStatusAvailabilityMask	M	0x00 – 0xFF	DTCSAM
#5	DTCSeverityAvailabilityMask	M	0x00 – 0xFF	DTCSVAM
#6	DTCFormatIdentifier = [SAE_J2012-DA_DTCFormat_04 SAE_J1939-73_DTCFormat]	M	0x04 0x02	DTCFID_ J2012-DADTCF04 J1939-73DTCF
	DTCAndSeverityRecord[] = [			DTCASR_
#7	DTCSeverity#1	C_1	0x00 – 0xFF	DTCS
#8	DTCHighByte#1 (MSB)	C_1	0x00 – 0xFF	DTCHB
#9	DTCMiddleByte#1	C_1	0x00 – 0xFF	DTCMB
#10	DTCLowByte#1	C_1	0x00 – 0xFF	DTCLB
#11	statusOfDTC#1	C_1	0x00 – 0xFF	SODTC
:	:	:	:	:
#n-4	DTCSeverity#m	C_2	0x00 – 0xFF	DTCS
#n-3	DTCHighByte#m (MSB)	C_2	0x00 – 0xFF	DTCHB
#n-2	DTCMiddleByte#m	C_2	0x00 – 0xFF	DTCMB
#n-1	DTCLowByte#m	C_2	0x00 – 0xFF	DTCLB
#n	statusOfDTC#m]	C_2	0x00 – 0xFF	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 284 defines the positive response message format of the sub-function parameter.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 284 — Response message definition - sub-function = reportWWHOBDDTCWithPermanentStatus

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	ReadDTCInformation Response SID	M	0x59	RDTCIPR
#2	reportType = [reportWWHOBDDTCWithPermanentStatus]	M	0x55	LEV_ RWWHOBDDTCWPS
#3	FunctionalGroupIdentifier	M	0x00 – 0xFF	FGID
#4	DTCStatusAvailabilityMask	M	0x00 – 0xFF	DTCSAM
#5	DTCFormatIdentifier = [SAE_J2012-DA_DTCFormat_04 SAE_J1939-73_DTCFormat]	M	0x04 0x02	DTCFID_ J2012-DADTCF04 J1939-73DTCF
	DTCAndStatusRecord[] = [			DTCASR_
#6	DTCHighByte#1 (MSB)	C_1	0x00 – 0xFF	DTCHB
#7	DTCMiddleByte#1	C_1	0x00 – 0xFF	DTCMB
#8	DTCLowByte#1	C_1	0x00 – 0xFF	DTCLB
#9	statusOfDTC#1	C_1	0x00 – 0xFF	SODTC
:	:	:	:	:
#n-3	DTCHighByte#m (MSB)	C_2	0x00 – 0xFF	DTCHB
#n-2	DTCMiddleByte#m	C_2	0x00 – 0xFF	DTCMB
#n-1	DTCLowByte#m	C_2	0x00 – 0xFF	DTCLB
#n	statusOfDTC#m]	C_2	0x00 – 0xFF	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.				

11.3.3.2 Positive response message data-parameter definition

Table 285 specifies the data-parameters of the positive response message.

Table 285 — Response data-parameter definition

Definition
reportType This parameter is an echo of bits 6 - 0 of the sub-function 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 [12] 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 [18].

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

 © ISO 2013 – All rights reserved

Table 285 — *(continued)*

Definition
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 [12] 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 [19] 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 [5] 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 [16] specification. This format is identified by the DTCFormatIdentifier = SAE_2012-DA_WWHOBD_DTCFormat.
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 defines 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 [12] 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 [19]. — ISO_11992-4_DTCFormat: This parameter value identifies the DTC format reported by the server as defined in ISO 11992-4 [5] specification. — SAE_J2012-DA_DTCFormat_04: This parameter value identifies the DTC format reported by the server as defined in ISO 27145-2 [16] specification. The definition of the byte values contained in the DTCFormatIdentifier byte can be found in D.4 of this specification. A given server shall support only one DTCFormatIdentifier.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 285 — *(continued)*

Definition
DTCCount This 2-byte parameter refers collectively to the DTCCountHighByte and DTCCountLowByte parameters that are sent in response to a reportNumberOfDTCByStatusMask or reportNumberOfMirrorMemoryDTC request. DTCCount provides a count of the number of DTCs that match the DTCStatusMask defined in the client's request.
DTCSnapshotRecordNumber Either the echo of the DTCSnapshotRecordNumber parameter specified by the client in the reportDTCSnapshotRecordByDTCNumber request, or the actual DTCSnapshotRecordNumber of a stored DTCSnapshot record.
DTCSnapshotRecordNumberOfIdentifiers This single byte parameter shows the number of dataIdentifiers in the immediately following DTCSnapshotRecord. A value of 0x00 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.
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 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.
FunctionalGroupIdentifier A one byte identifier which contains the functional system group the DTC(s) are related to e.g. 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.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

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

Table 286 — Supported negative response codes

NRC	Description	Mnemonic
0x12	**sub-functionNotSupported** This NRC shall be sent if the sub-function parameter is not supported.	SFNS
0x13	**incorrectMessageLengthOrInvalidFormat** This NRC shall be sent if the length of the message is wrong.	IMLOIF
0x31	**requestOutOfRange** 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. Note that this is to be differentiated from the case where the DTCSnapshotRecordNumber 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 MemorySelection identifier was not recognized by the server.	ROOR

11.3.5 Message flow examples – ReadDTCInformation

11.3.5.1 General assumption

For all examples the client requests to have a response message by setting the suppressPosRspMsgIndicationBit (bit 7 of the sub-function parameter) to "FALSE" ('0').

11.3.5.2 Example #1 - ReadDTCInformation, sub-function = reportNumberOfDTCByStatusMask

11.3.5.2.1 Example #1 overview

This example demonstrates the usage of the reportNumberOfDTCByStatusMask sub-function parameter for confirmed DTCs (DTC status mask 0x08), as well as various masking principles. The DTCStatusAvailabilityMask for this sever = 0x2F.

11.3.5.2.2 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 (0x080511), statusOfDTC 0x24 ($0010\ 0100_b$).

Table 287 defines the statusOfDTC = 0x24 of DTC P0805-11.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 287 — statusOfDTC = 0x24 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 never 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 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

The following assumptions apply to DTC P0A9B-17 Hybrid Battery Temperature Sensor - circuit voltage above threshold (0x0A9B17), statusOfDTC of 0x26 ($0010\ 0110_b$).

Table 288 defines the statusOfDTC = 0x26 of DTC P0A9B-17.

Table 288 — statusOfDTC = 0x26 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 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

The following assumptions apply to DTC P2522-1F A/C Request “B” - circuit intermittent (0x25221F), statusOfDTC of 0x2F ($0010\ 1111_b$).

Table 289 defines the statusOfDTC = 0x2F of DTC P2522-1F.

Table 289 — statusOfDTC = 0x2F 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 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

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

11.3.5.2.3 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 (0x25221F), statusOfDTC of 0x2F (0010 1111_b) matches the client defined status mask of 0x08 (0000 1000_b).

Table 290 defines the ReadDTCInformation, sub-function = reportNumberOfDTCByStatusMask, request message flow example #1.

Table 290 — ReadDTCInformation, sub-function = 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	0x19	RDTCI
#2	sub-function = reportNumberOfDTCByStatusMask, suppressPosRspMsgIndicationBit = FALSE	0x01	RNODTCBSM
#3	DTCStatusMask	0x08	DTCSM

Table 291 defines the ReadDTCInformation, sub-function = reportNumberOfDTCByStatusMask, positive response, example #1.

Table 291 — ReadDTCInformation, sub-function = 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	0x59	RDTCIPR
#2	reportType = reportNumberOfDTCByStatusMask	0x01	RNODTCBSM
#3	DTCStatusAvailabilityMask	0x2F	DTCSAM
#4	DTCFormatIdentifier = ISO_14229-1_DTCFormat	0x01	14229-1DTCF
#5 #6	DTCCount [DTCCountHighByte] DTCCount [DTCCountLowByte]	0x00 0x01	DTCCHB DTCCLB

11.3.5.3 Example #2 - ReadDTCInformation, sub-function = reportDTCByStatusMask, matching DTCs returned

11.3.5.3.1 Example #2 overview

This example demonstrates usage of the reportDTCByStatusMask sub-function parameter, as well as various masking principles in conjunction with unsupported masking bits. This example also applies to the sub-function parameter reportMirrorMemoryDTCByStatusMask and the sub-function parameter reportUserDefMemoryDTCByStatusMask, except that the status mask checks are performed with the DTCs stored in the DTC mirror memory or in the user defined memory.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

11.3.5.3.2 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 (0x0A9B17), statusOfDTC 0x24 ($0010\ 0100_b$).

Table 292 defines the statusOfDTC= 0x24 of DTC P0A9B-17.

Table 292 — statusOfDTC= 0x24 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 never 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 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

The following assumptions apply to DTC P2522-1F A/C Request “B” - circuit intermittent (0x25221F), statusOfDTC of 0x00 ($0000\ 0000_b$).

Table 293 defines the statusOfDTC = 0x00 of DTC P2522-1F.

Table 293 — statusOfDTC = 0x00 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 never failed on 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 P0805-11 Clutch Position Sensor - circuit short to ground (0x080511), statusOfDTC of 0x2F ($0010\ 1111_b$).

Table 294 defines the statusOfDTC = 0x2F of DTC P0805-11.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 294 — statusOfDTC = 0x2F 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 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

11.3.5.3.3 Example #2 message flow

In the following example, DTCs P0A9B-17 (0x0A9B17) and P0805-11 (0x080511) are returned to the client's request. DTC P2522-1F (0x25221F) is not returned because its status of 0x00 does not match the DTCStatusMask of 0x84 (as specified in the client request messsage in the following example). The server shall bypass masking on those status bits it doesn't support.

Table 295 defines the ReadDTCInformation, sub-function = reportDTCByStatusMask, request message flow example #2.

Table 295 — ReadDTCInformation, sub-function = 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	0x19	RDTCI
#2	sub-function = reportDTCByStatusMask, suppressPosRspMsgIndicationBit = FALSE	0x02	RDTCBSM
#3	DTCStatusMask	0x84	DTCSM

Table 296 defines the ReadDTCInformation, sub-function = reportDTCByStatusMask, positive response, example #2.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 296 — ReadDTCInformation, sub-function = 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	0x59	RDTCIPR
#2	reportType = reportDTCByStatusMask	0x02	RDTCBSM
#3	DTCStatusAvailabilityMask	0x7F	DTCSAM
#4	DTCAndStatusRecord#1 [DTCHighByte]	0x0A	DTCHB
#5	DTCAndStatusRecord#1 [DTCMiddleByte]	0x9B	DTCMB
#6	DTCAndStatusRecord#1 [DTCLowByte]	0x17	DTCLB
#7	DTCAndStatusRecord#1 [statusOfDTC]	0x24	SODTC
#8	DTCAndStatusRecord#2 [DTCHighByte]	0x08	DTCHB
#9	DTCAndStatusRecord#2 [DTCMiddleByte]	0x05	DTCMB
#10	DTCAndStatusRecord#2 [DTCLowByte]	0x11	DTCLB
#11	DTCAndStatusRecord#2 [statusOfDTC]	0x2F	SODTC

11.3.5.4 Example #3 - ReadDTCInformation, sub-function = reportDTCByStatusMask, no matching DTCs returned

11.3.5.4.1 Example #3 overview

This example demonstrates usage of the reportDTCByStatusMask sub-function parameter, in the situation where no DTCs match the client defined DTCStatusMask.

11.3.5.4.2 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 (0x25221F), statusOfDTC 0x24 ($0010\ 0100_b$).

Table 297 defines the statusOfDTC= 0x24 of DTC P2522-1F.

Table 297 — statusOfDTC= 0x24 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 never 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 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

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

The following assumptions apply to DTC P0A9B-17 Hybrid Battery Temperature Sensor - circuit voltage above threshold (0x0A9B17), statusOfDTC of 0x00 ($0000\ 0000_b$).

Table 298 defines the statusOfDTC = 0x00 of DTC P0A9B-17.

Table 298 — statusOfDTC = 0x00 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 never failed on 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 client requests the server to reportByStatusMask all DTCs having bit 0 (TestFailed) set to logical ‘1’.

11.3.5.4.3 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 299 defines the ReadDTCInformation, sub-function = reportDTCByStatusMask, request message flow example #3.

Table 299 — ReadDTCInformation, sub-function = 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	0x19	RDTCI
#2	sub-function = reportDTCByStatusMask, suppressPosRspMsgIndicationBit = FALSE	0x02	RDTCBSM
#3	DTCStatusMask	0x01	DTCSM

Table 300 defines the ReadDTCInformation, sub-function = reportDTCByStatusMask, positive response, example #3.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 300 — ReadDTCInformation, sub-function = 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	0x59	RDTCIPR
#2	reportType = reportDTCByStatusMask	0x02	RDTCBSM
#3	DTCStatusAvailabilityMask	0x7F	DTCSAM

11.3.5.5 Example #4 - ReadDTCInformation, sub-function = reportDTCSnapshotIdentification

11.3.5.5.1 Example #4 overview

This example demonstrates the usage of the reportDTCSnapshotIdentification sub-function parameter.

11.3.5.5.2 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 0x123456. 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 0x789ABC.

— All DTCSnapshot records are stored in ascending order.

11.3.5.5.3 Example #4 message flow

In the following example, three DTCSnapshot records are returned to the client's request.

Table 301 defines the ReadDTCInformation, sub-function = reportDTCSnapshotIdentification, request message flow example #4.

Table 301 — ReadDTCInformation, sub-function = 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	0x19	RDTCI
#2	sub-function = reportDTCSnapshotIdentification, suppressPosRspMsgIndicationBit = FALSE	0x03	RDTCSSI

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 302 defines the ReadDTCInformation, sub-function = reportDTCSnapshotIdentification, positive response, example #4.

Table 302 — ReadDTCInformation, sub-function = 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	0x59	RDTCIPR
#2	reportType = reportDTCSnapshotIdentification	0x03	RDTCSSI
#3	DTCAndStatusRecord#1 [DTCHighByte]	0x12	DTCHB
#4	DTCAndStatusRecord#1 [DTCMiddleByte]	0x34	DTCMB
#5	DTCAndStatusRecord#1 [DTCLowByte]	0x56	DTCLB
#6	DTCSnapshotRecordNumber#1	0x01	DTCEDRC
#7	DTCAndStatusRecord#2 [DTCHighByte]	0x12	DTCHB
#8	DTCAndStatusRecord#2 [DTCMiddleByte]	0x34	DTCMB
#9	DTCAndStatusRecord#2 [DTCLowByte]	0x56	DTCLB
#10	DTCSnapshotRecordNumber#2	0x02	DTCEDRC
#11	DTCAndStatusRecord#3 [DTCHighByte]	0x78	DTCHB
#12	DTCAndStatusRecord#3 [DTCMiddleByte]	0x9A	DTCMB
#13	DTCAndStatusRecord#3 [DTCLowByte]	0xBC	DTCLB
#14	DTCSnapshotRecordNumber#3	0x01	DTCEDRC

11.3.5.6 Example #5 - ReadDTCInformation, sub-function = reportDTCSnapshotRecord-ByDTCNumber

11.3.5.6.1 Example #5 overview

This example demonstrates the usage of the reportDTCSnapshotRecordByDTCNumber sub-function parameter. This example also applies to the sub-function parameter reportUserDefMemory-DTCSnapshotRecordByDTCNumber, except that the checks are performed with the DTCs stored in the user defined memory.

11.3.5.6.2 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 0x123456 (see previous example, where a DTCSnapshotRecordCount of 0x02 is returned to the client).

— Assume that DTC 0x123456 has a statusOfDTC of 0x24, and that the following environment data is captured each time a DTC occurs.

— The DTCSnapshot record data is referenced via the dataIdentifier 0x4711.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 303 defines the DTCSnapshot record content.

Table 303 — DTCSnapshot record content

Data Byte	DTCSnapshot Record Contents	Byte Value
#1	DTCSnapshotRecord [data#1] = ECT (Engine Coolant Temperature)	0xA6
#2	DTCSnapshotRecord [data#2] = TP (Throttle Position)	0x66
#3	DTCSnapshotRecord [data#3] = RPM (Engine Speed)	0x07
#4	DTCSnapshotRecord [data#4] = RPM (Engine Speed)	0x50
#5	DTCSnapshotRecord [data#5] = MAP (Manifold Absolute Pressure)	0x20

11.3.5.6.3 Example #5 message flow

In the following example, one DTCSnapshot record is returned in accordance to the client's reportDTCSnapshotRecordByDTCNumber request.

Table 304 defines the ReadDTCInformation, sub-function = reportDTCSnapshotRecordByDTCNumber, request message flow example #5.

Table 304 — ReadDTCInformation, sub-function = 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	0x19	RDTCI
#2	sub-function = reportDTCSnapshotRecordByDTCNumber, suppressPosRspMsgIndicationBit = FALSE	0x04	RDTCSSRBDN
#3	DTCMaskRecord [DTCHighByte]	0x12	DTCHB
#4	DTCMaskRecord [DTCMiddleByte]	0x34	DTCMB
#5	DTCMaskRecord [DTCLowByte]	0x56	DTCLB
#6	DTCSnapshotRecordNumber	0x02	DTCSSRN

Table 305 defines the ReadDTCInformation, sub-function = reportDTCSnapshotRecordByDTCNumber, positive response, example #5.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 305 — ReadDTCInformation, sub-function = 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	0x59	RDTCIPR
#2	reportType = reportDTCSnapshotRecordByDTCNumber	0x04	RDTCSSRBDN
#3	DTCAndStatusRecord [DTCHighByte]	0x12	DTCHB
#4	DTCAndStatusRecord [DTCMiddleByte]	0x34	DTCMB
#5	DTCAndStatusRecord [DTCLowByte]	0x56	DTCLB
#6	DTCAndStatusRecord [statusOfDTC]	0x24	SODTC
#7	DTCSnapshotRecordNumber	0x02	DTCEDRN
#8	DTCSnapshotRecordNumberOfIdentifiers	0x01	DTCSSRNI
#9	dataIdentifier [byte#1] (MSB)	0x47	DIDB1
#10	dataIdentifier [byte#2] (LSB)	0x11	DIDB2
#11	DTCSnapshotRecord [data#1] = ECT	0xA6	ED_1
#12	DTCSnapshotRecord [data#2] = TP	0x66	ED_2
#13	DTCSnapshotRecord [data#3] = RPM	0x07	ED_3
#14	DTCSnapshotRecord [data#4] = RPM	0x50	ED_4
#15	DTCSnapshotRecord [data#5] = MAP	0x20	ED_5

11.3.5.7 Example #6 - ReadDTCInformation, sub-function = reportDTCStoredDataByRecordNumber

11.3.5.7.1 Example #6 overview

This example demonstrates the usage of the reportDTCStoredDataByRecordNumber sub-function parameter.

11.3.5.7.2 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 0x123456 (see previous example, where a DTCStoredDataRecordCount of two is returned to the client).

— Assume that DTC 0x123456 has a statusOfDTC of 0x24, and that the following environment data is captured each time a DTC occurs.

— The DTCStoredData record data is referenced via the dataIdentifier 0x4711.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 306 defines the DTCStoredData record content.

Table 306 — DTCStoredData record content

Data Byte	DTCSnapshot Record Contents	Byte Value
#1	DTCStoredDatatRecord [data#1] = ECT (Engine Coolant Temp.)	0xA6
#2	DTCStoredDataRecord [data#2] = TP (Throttle Position)	0x66
#3	DTCStoredDataRecord [data#3] = RPM (Engine Speed)	0x07
#4	DTCStoredDataRecord [data#4] = RPM (Engine Speed)	0x50
#5	DTCStoredDataRecord [data#5] = MAP (Manifold Absolute Pressure)	0x20

11.3.5.7.3 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 307 defines the ReadDTCInformation, sub-function = reportDTCStoredDataByRecordNumber, request message flow example #6.

Table 307 — ReadDTCInformation, sub-function = 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	0x19	RDTCI
#2	sub-function = reportDTCStoredDataByRecordNumber, suppressPosRspMsgIndicationBit = FALSE	0x05	RDTCSDRBRN
#3	DTCStoredDataRecordNumber	0x02	DTCSDRN

Table 308 defines the ReadDTCInformation, sub-function = reportDTCStoredDataByRecordNumber, positive response, example #6.

Table 308 — ReadDTCInformation, sub-function = 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	0x59	RDTCIPR
#2	reportType = reportDTCStoredDataByRecordNumber	0x05	RDTCSDRBRN
#3	DTCStoredDataRecordNumber	0x02	DTCSDRN
#4	DTCAndStatusRecord [DTCHighByte]	0x12	DTCHB
#5	DTCAndStatusRecord [DTCMiddleByte]	0x34	DTCMB
#6	DTCAndStatusRecord [DTCLowByte]	0x56	DTCLB
#7	DTCAndStatusRecord [statusOfDTC]	0x24	SODTC

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 308 — *(continued)*

Message direction	**server → client**		
Message Type	**Response**		
A_Data Byte	**Description (all values are in hexadecimal)**	**Byte Value**	**Mnemonic**
#8	DTCStoredDataRecordNumberOfIdentifiers	0x01	DTCSDRNI
#9	dataIdentifier [byte#1 (MSB)]	0x47	DIDB1
#10	dataIdentifier [byte#2] (LSB)	0x11	DIDB2
#11	DTCStoredDataRecord [data#1] = ECT	0xA6	ED_1
#12	DTCStoredDataRecord [data#2] = TP	0x66	ED_2
#13	DTCStoredDataRecord [data#3] = RPM	0x07	ED_3
#14	DTCStoredDataRecord [data#4] = RPM	0x50	ED_4
#15	DTCStoredDataRecord [data#5] = MAP	0x20	ED_5

11.3.5.8 Example #7 - ReadDTCInformation, sub-function = reportDTCExtDataRecordByDTCNumber

11.3.5.8.1 Example #7 overview

This example demonstrates the usage of the reportDTCExtDataRecordByDTCNumber sub-function parameter. This example also applies to the sub-function parameter reportUserDefMemory-DTCExtDataRecordByDTCNumber, except that the checks are performed with the DTCs stored in the user defined memory.

11.3.5.8.2 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 0x123456.

— Assume that DTC 0x123456 has a statusOfDTC of 0x24, and that the following extended data is available for the DTC.

— The DTCExtendedData is referenced via the DTCExtDataRecordNumbers 0x05 and 0x10

Table 309 — DTCExtDataRecordNumber 0x05 content

Data Byte	**DTCExtDataRecord Contents for DTCExtDataRecordNumber 0x05**	**Byte Value**
#1	Warm-up Cycle Counter – Number of warm up cycles since the DTC commanded the MIL to switch off	0x17

Table 310 — DTCExtDataRecordNumber 0x10 content

Data Byte	**DTCExtDataRecord Contents for DTCExtDataRecordNumber 0x10**	**Byte Value**
#1	DTC Fault Detection Counter – Increments each time the DTC test detects a fault, Decrements each time the test reports no fault.	0x79

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

11.3.5.8.3 Example #7 message flow

In the following example, a DTCMaskRecord including the DTC number and a DTCExtDataRecordNumber with the value of 0xFF (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 311 defines the ReadDTCInformation, sub-function = reportDTCExtDataRecordByDTCNumber, request message flow example #7.

Table 311 — ReadDTCInformation, sub-function = 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	0x19	RDTCI
#2	sub-function = reportDTCExtDataRecordByDTCNumber, suppressPosRspMsgIndicationBit = FALSE	0x06	RDTCEDRBDN
#3	DTCMaskRecord [DTCHighByte]	0x12	DTCHB
#4	DTCMaskRecord [DTCMiddleByte]	0x34	DTCMB
#5	DTCMaskRecord [DTCLowByte]	0x56	DTCLB
#6	DTCExtDataRecordNumber	0xFF	DTCEDRN

Table 312 defines the ReadDTCInformation, sub-function = reportDTCExtDataRecordByDTCNumber, positive response, example #7.

Table 312 — ReadDTCInformation, sub-function = 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	0x59	RDTCIPR
#2	reportType = reportDTCExtDataRecordByDTCNumber	0x06	RDTCEDRBDN
#3	DTCAndStatusRecord [DTCHighByte]	0x12	DTCHB
#4	DTCAndStatusRecord [DTCMiddleByte]	0x34	DTCMB
#5	DTCAndStatusRecord [DTCLowByte]	0x56	DTCLB
#6	DTCAndStatusRecord [statusOfDTC]	0x24	SODTC
#7	DTCExtDataRecordNumber	0x05	DTCEDRN
#8	DTCExtDataRecord [byte#1]	0x17	ED_1
#9	DTCExtDataRecordNumber	0x10	DTCEDRN
#10	DTCExtDataRecord [byte#1]	0x79	ED_1

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

11.3.5.9 Example #8 - ReadDTCInformation, sub-function = reportNumberOfDTC-BySeverityMaskRecord

11.3.5.9.1 Example #8 overview

This example demonstrates the usage of the reportNumberOfDTCBySeverityMaskRecord sub-function parameter.

11.3.5.9.2 Example #8 assumptions

The server supports a total of three DTCs which have the following states at the time of the client request:

The following assumptions apply to DTC P0A9B-17 Hybrid Battery Temperature Sensor - circuit voltage above threshold (0x0A9B17), statusOfDTC 0x24 (0010 0100b), DTCFunctionalUnit = 0x10, DTCSeverity = 0x20:

NOTE Only bit 7 to 5 of the severity byte are valid.

Table 313 defines the statusOfDTC = 0x24 of DTC P0A9B-17.

Table 313 — statusOfDTC = 0x24 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 never 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 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

The following assumptions apply to DTC P2522-1F A/C Request “B” - circuit intermittent (0x25221F), statusOfDTC of 0x00 (0000 0000 binary), DTCFunctionalUnit = 0x10, DTCSeverity = 0x20:

NOTE Only bit 7 to 5 of the severity byte are valid.

Table 314 defines the statusOfDTC = 0x00 of DTC P2522-1F.

Table 314 — statusOfDTC = 0x00 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 never failed on 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

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

The following assumptions apply to DTC P0805-11 Clutch Position Sensor - circuit short to ground (0x080511), statusOfDTC of 0x2F ($0010\ 1111_b$), DTCFunctionalUnit = 0x10, DTCSeverity = 0x40:

NOTE Only bit 7 to 5 of the severity byte are valid.

Table 315 defines the statusOfDTC = 0x2F of DTC P0805-11.

Table 315 — statusOfDTC = 0x2F 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 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

The server supports the testFailed and confirmedDTC status bits for masking purposes.

11.3.5.9.3 Example #8 message flow

In the following example, a count of one is returned to the client because DTC P0805-11 (0x080511) match the client defined severity mask record of 0xC001 (DTCSeverityMask = $110x\ xxxx_b$ = 0xC0, DTCStatusMask = $0000\ 0001_b$).

Table 316 defines the ReadDTCInformation, sub-function = reportNumberOfDTCBySeverityMaskRecord, request message flow example #8.

Table 316 — ReadDTCInformation, sub-function = 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	0x19	RDTCI
#2	sub-function = reportNumberOfDTCBySeverityMaskRecord, suppressPosRspMsgIndicationBit = FALSE	0x07	RNODTCBSMR
#3	DTCSeverityMaskRecord(DTCSeverityMask)	0xC0	DTCSVM
#4	DTCSeverityMaskRecord(DTCStatusMask)	0x01	DTCSM

Table 317 defines the ReadDTCInformation, sub-function = reportNumberOfDTCBySeverityMaskRecord, positive response, positive response, example #8.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

 © ISO 2013 – All rights reserved

Table 317 — ReadDTCInformation, sub-function = 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	0x59	RDTCIPR
#2	reportType = reportNumberOfDTCBySeverityMaskRecord	0x07	RNODTCBSMR
#3	DTCStatusAvailabilityMask	0x09	DTCSAM
#4	DTCFormatIdentifier = ISO_14229-1_DTCFormat	0x01	14229-1DTCF
#5	DTCCount [DTCCountHighByte]	0x00	DTCCHB
#6	DTCCount [DTCCountLowByte]	0x01	DTCCLB

11.3.5.10 Example #9 - ReadDTCInformation, sub-function = reportDTCBySeverityMaskRecord

11.3.5.10.1 Example #9 overview

This example demonstrates the usage of the reportDTCBySeverityMaskRecord sub-function parameter.

11.3.5.10.2 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 (0x080511) match the client defined severity mask record of 0xC001 (DTCSeverityMask = 0xC0 = 110x XXXX_b, DTCStatusMask = 0x01, $0000\ 0001_b$) and is reported to the client. The severity of DTC P0805-11 (0x080511) is 0x40 (010x XXXX_b). 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.

11.3.5.10.3 Example #9 message flow

In the following example, one DTCSeverityRecord is returned to the client’s request.

Table 318 defines the ReadDTCInformation, sub-function = reportDTCBySeverityMaskRecord, request message flow example #9.

Table 318 — ReadDTCInformation, sub-function = 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	0x19	RDTCI
#2	sub-function = reportDTCBySeverityMaskRecord, suppressPosRspMsgIndicationBit = FALSE	0x08	RDTCBSMR
#3	DTCSeverityMaskRecord(DTCSeverityMask)	0xC0	DTCSVM
#4	DTCSeverityMaskRecord(DTCStatusMask)	0x01	DTCSM

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 319 — ReadDTCInformation, sub-function = 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	0x59	RDTCIPR
#2	reportType = reportDTCBySeverityMaskRecord	0x08	RDTCBSMR
#3	DTCStatusAvailabilityMask	0x7F	DTCSAM
#4	DTCSeverityRecord#1 [DTCSeverity]	0x40	DTCS
#5	DTCSeverityRecord#1 [DTCFunctionalUnit]	0x10	DTCFU
#6	DTCSeverityRecord#1 [DTCHighByte]	0x08	DTCHB
#7	DTCSeverityRecord#1 [DTCMiddleByte]	0x05	DTCMB
#8	DTCSeverityRecord#1 [DTCLowByte]	0x11	DTCLB
#9	DTCSeverityRecord#1 [statusOfDTC]	0x2F	SODTC

11.3.5.11 Example #10 - ReadDTCInformation, sub-function = reportSeverityInformationOfDTC

11.3.5.11.1 Example #10 overview

This example demonstrates the usage of the reportSeverityInformationOfDTC sub-function parameter.

11.3.5.11.2 Example #10 assumptions

The assumptions defined in 11.3.5.10.2 apply.

11.3.5.11.3 Example #10 message flow

In the following example, the DTC P0805-11 (0x080511), which matches the client defined DTC mask record, is reported to the client.

Table 320 — ReadDTCInformation, sub-function = 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	0x19	RDTCI
#2	sub-function = reportSeverityInformationOfDTC, suppressPosRspMsgIndicationBit = FALSE	0x09	RSIODTC
#3	DTCMaskRecord [DTCHighByte]	0x08	DTCHB
#4	DTCMaskRecord [DTCMiddleByte]	0x05	DTCMB
#5	DTCMaskRecord [DTCLowByte]	0x11	DTCLB

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 321 — ReadDTCInformation, sub-function = 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	0x59	RDTCIPR
#2	reportType = reportDTCBySeverityMaskRecord	0x09	RSIODTC
#3	DTCStatusAvailabilityMask	0x7F	DTCSAM
#4	DTCSeverityRecord [DTCSeverity]	0x40	DTCS
#5	DTCSeverityRecord [DTCFunctionalUnit]	0x10	DTCFU
#6	DTCSeverityRecord [DTCHighByte]	0x08	DTCHB
#7	DTCSeverityRecord [DTCMiddleByte]	0x05	DTCMB
#8	DTCSeverityRecord [DTCLowByte]	0x11	DTCLB
#9	DTCSeverityRecord [statusOfDTC]	0x2F	SODTC

11.3.5.12 Example #11 – ReadDTCInformation - sub-function = reportSupportedDTCs

11.3.5.12.1 Example #11 overview

This example demonstrates the usage of the reportSupportedDTCs sub-function parameter.

11.3.5.12.2 Example #11 assumptions

The assumptions defined in 11.3.5.10.2 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 0x123456, statusOfDTC 0x24 ($0010\ 0100_b$)

Table 322 — statusOfDTC = 0x24

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 never failed on 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 0x234505, statusOfDTC of 0x00 ($0000\ 0000_b$)

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 323 — statusOfDTC = 0x00

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 never failed on 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 0xABCD01, statusOfDTC of 0x2F ($0010\ 1111_b$)

Table 324 — statusOfDTC = 0x2F

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

11.3.5.12.3 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 325 — ReadDTCInformation, sub-function = 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	0x19	RDTCI
#2	sub-function = reportSupportedDTCs, suppressPosRspMsgIndicationBit = FALSE	0x0A	RSUPDTC

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 326 — ReadDTCInformation, sub-function = 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	0x59	RDTCIPR
#2	reportType = readSupportedDTCs	0x0A	RSUPDTC
#3	DTCStatusAvailabilityMask	0x7F	DTCSAM
#4	DTCAndStatusRecord#1 [DTCHighByte]	0x12	DTCHB
#5	DTCAndStatusRecord#1 [DTCMiddleByte]	0x34	DTCMB
#6	DTCAndStatusRecord#1 [DTCLowByte]	0x56	DTCLB
#7	DTCAndStatusRecord#1 [statusOfDTC]	0x24	SODTC
#8	DTCAndStatusRecord#2 [DTCHighByte]	0x23	DTCHB
#9	DTCAndStatusRecord#2 [DTCMiddleByte]	0x45	DTCMB
#10	DTCAndStatusRecord#2 [DTCLowByte]	0x05	DTCLB
#11	DTCAndStatusRecord#2 [statusOfDTC]	0x00	SODTC
#12	DTCAndStatusRecord#3 [DTCHighByte]	0xAB	DTCHB
#13	DTCAndStatusRecord#3 [DTCMiddleByte]	0xCD	DTCMB
#14	DTCAndStatusRecord#3 [DTCLowByte]	0x01	DTCLB
#15	DTCAndStatusRecord#3 [statusOfDTC]	0x2F	SODTC

11.3.5.13 Example #12 - ReadDTCInformation, sub-function = reportFirstTestFailedDTC, information available

11.3.5.13.1 Example #12 overview

This example demonstrates usage of the reportFirstTestFailedDTC sub-function 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 sub-function parameters reportFirstConfirmedDTC, reportMostRecentTestFailedDTC, and reportMostRecent-ConfirmedDTC (for the appropriate DTC status and under similar assumptions).

11.3.5.13.2 Example #12 assumptions

The following assumptions apply:

— At least one DTC failed since the last ClearDiagnosticInformation request from the server.

— The server supports all status bits for masking purposes.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

— DTC number 0x123456 = first failed DTC to be detected since the last code clear.

— The following assumptions apply to DTC 0x123456, statusOfDTC 0x26 ($0010\ 0110_b$):

Table 327 — statusOfDTC = 0x26

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 never failed on 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

11.3.5.13.3 Example #12 message flow

In the following example DTC 0x123456 is returned to the client's request.

Table 328 — ReadDTCInformation, sub-function = 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	0x19	RDTCI
#2	sub-function = reportFirstTestFailedDTC, suppressPosRspMsgIndicationBit = FALSE	0x0B	RFCDTC

Table 329 — ReadDTCInformation, sub-function = 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	0x59	RDTCIPR
#2	reportType = reportFirstTestFailedDTC	0x0B	RFCDTC
#3	DTCStatusAvailabilityMask	0xFF	DTCSAM
#4	DTCAndStatusRecord [DTCHighByte]	0x12	DTCHB
#5	DTCAndStatusRecord [DTCMiddleByte]	0x34	DTCMB
#6	DTCAndStatusRecord [DTCLowByte]	0x56	DTCLB
#7	DTCAndStatusRecord [statusOfDTC]	0x26	SODTC

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

11.3.5.14 Example #13 - ReadDTCInformation, sub-function = reportFirstTestFailedDTC, no information available

11.3.5.14.1 Example #13 overview

This example demonstrates usage of the reportFirstTestFailedDTC sub-function 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 sub-function parameters reportFirstConfirmedDTC, reportMostRecentTestFailedDTC, and reportMostRecentConfirmedDTC (for the appropriate DTC status and under similar assumptions).

11.3.5.14.2 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.

11.3.5.14.3 Example #13 message flow

In the following example no DTC is returned to the client's request.

Table 330 — ReadDTCInformation, sub-function = 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	0x19	RDTCI
#2	sub-function = reportFirstTestFailedDTC, suppressPosRspMsgIndicationBit = FALSE	0x0B	RFCDTC

Table 331 — ReadDTCInformation, sub-function = reportFirstTestFailedDTC, positive response, example #13

Message direction	**server → client**		
Message Type	**Response**		
A_Data Byte	**Description (all values are in hexadecimal)**	**Byte Value**	**Mnemonic**
#1	ReadDTCInformation Response SID	0x59	RDTCIPR
#2	reportType = reportFirstTestFailedDTC	0x0B	RFCDTC
#3	DTCStatusAvailabilityMask	0xFF	DTCSAM

11.3.5.15 Example #14 - ReadDTCInformation, sub-function = reportNumberOfEmissionsOBD-DTCByStatusMask

11.3.5.15.1 Example #14 overview

This example demonstrates the usage of the reportNumberOfEmissionsOBDDTCByStatusMask sub-function parameter, as well as various masking principles.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

11.3.5.15.2 Example #14 assumptions

The server supports all status bits for masking purposes. Furthermore the server supports a total of three emissions-related OBD DTCs (for the sake of simplicity!), which have the following states at the time of the client request:

The following assumptions apply to emissions-related OBD DTC P0005-00 Fuel Shutoff Valve "A" Control Circuit/Open (0x000500), statusOfDTC 0xAE ($1010\ 1110_b$).

Table 332 — statusOfDTC = 0xAE of DTC P0005-00

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 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 failed at least once since last code clear
testNotCompletedThisOperationCycle	6	0	DTC test completed this operation cycle
warningIndicatorRequested	7	1	Server is requesting warningIndicator to be active (OBD DTC)

The following assumptions apply to emissions-related OBD DTC P022F-00 Intercooler Bypass Control "B" Circuit High (0x022F00), statusOfDTC of 0xAC ($1010\ 1100_b$).

Table 333 — statusOfDTC = 0xAC of DTC P022F-00

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 never 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 failed at least once since last code clear
testNotCompletedThisOperationCycle	6	0	DTC test completed this operation cycle,
warningIndicatorRequested	7	1	Server is requesting warningIndicator to be active (OBD DTC)

The following assumptions apply to emissions-related OBD DTC P0A09-00 DC/DC Converter Status Circuit Low Input (0x0A0900), statusOfDTC of 0xAF ($1010\ 1111_b$).

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 334 — statusOfDTC = AF of DTC P0A09-00

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 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	1	Server is requesting warningIndicator to be active (OBD DTC)

11.3.5.15.3 Example #14 message flow

In the following example, a count of three is returned to the client because all DTCs defined in the assumptions match the client defined status mask of 0x08 – confirmedDTC (0000 1000_b).

Table 335 — ReadDTCInformation, sub-function = reportNumberOfEmissionsOBD-DTCByStatusMask, 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	0x19	RDTCI
#2	sub-function = reportNumberOfEmissionsOBDDTCByStatusMask, suppressPosRspMsgIndicationBit = FALSE	0x12	RNOOEOBDDTCBSM
#3	DTCStatusMask	0x08	DTCSM

Table 336 — ReadDTCInformation, sub-function = reportNumberOfEmissionsOBD-DTCByStatusMask, 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	0x59	RDTCIPR
#2	reportType = reportNumberOfEmissionsOBDDTCByStatusMask	0x12	RNOOEOBDDTCBSM
#3	DTCStatusAvailabilityMask	0xFF	DTCSAM
#4	DTCFormatIdentifier = SAE_J2012-DA_DTCFormat_00	0x00	J2012-DADTCF00
#5	DTCCount [DTCCountHighByte]	0x00	DTCCHB
#6	DTCCount [DTCCountLowByte]	0x03	DTCCLB

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

11.3.5.16 Example #15 - ReadDTCInformation, sub-function = reportEmissionsOBDDTC-ByStatusMask, all matching OBD DTCs returned

11.3.5.16.1 Example #15 overview

This example demonstrates usage of the reportEmissionsOBDDTCByStatusMask sub-function parameter, as well as various masking principles in conjunction with unsupported masking bits.

11.3.5.16.2 Example #15 assumptions

The server supports all status bits for masking purposes. The server supports a total of three DTCs (for the sake of simplicity!) as defined in 11.3.5.15.2.

11.3.5.16.3 Example #15 message flow

In the following example, emissions-related OBD DTC P0005-AE Fuel Shutoff Valve "A" Control Circuit/Open (0x000500), P022F-00 Intercooler Bypass Control "B" Circuit High (0x022F00) and P0A09-00 DC/DC Converter Status Circuit Low Input (0x0A0900) are returned to the client's request because all DTCs defined in the assumptions match the client defined status mask of 0x80 – warningIndicatorRequested (1000 0000 $_b$).

NOTE The server shall bypass masking on those status bits it doesn't support.

Table 337 — ReadDTCInformation, sub-function = reportEmissionsOBDDTCByStatusMask, 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	0x19	RDTCI
#2	sub-function = reportEmissionsOBDDTCByStatusMask, suppressPosRspMsgIndicationBit = FALSE	0x13	ROBDDTCBSM
#3	DTCStatusMask	0x80	DTCSM

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

 © ISO 2013 – All rights reserved

Table 338 — ReadDTCInformation, sub-function = reportDTCByStatusMask, 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	0x59	RDTCIPR
#2	reportType = reportEmissionsOBDDTCByStatusMask	0x13	ROBDDTCBSM
#3	DTCStatusAvailabilityMask	0xFF	DTCSAM
#4	DTCAndStatusRecord#1 [DTCHighByte]	0x00	DTCHB
#5	DTCAndStatusRecord#1 [DTCMiddleByte]	0x05	DTCMB
#6	DTCAndStatusRecord#1 [DTCLowByte]	0x00	DTCLB
#7	DTCAndStatusRecord#1 [statusOfDTC]	0xAE	SODTC
#8	DTCAndStatusRecord#2 [DTCHighByte]	0x02	DTCHB
#9	DTCAndStatusRecord#2 [DTCMiddleByte]	0x2F	DTCMB
#10	DTCAndStatusRecord#2 [DTCLowByte]	0x00	DTCLB
#11	DTCAndStatusRecord#2 [statusOfDTC]	0xAC	SODTC
#12	DTCAndStatusRecord#3 [DTCHighByte]	0x0A	DTCHB
#13	DTCAndStatusRecord#3 [DTCMiddleByte]	0x09	DTCMB
#14	DTCAndStatusRecord#3 [DTCLowByte]	0x00	DTCLB
#15	DTCAndStatusRecord#3 [statusOfDTC]	0xAF	SODTC

11.3.5.17 Example #16 - ReadDTCInformation, sub-function = reportEmissionsOBDDTC-ByStatusMask (confirmedDTC and warningIndicatorRequested), matching DTCs returned

11.3.5.17.1 Example #16 overview

This example demonstrates usage of the reportEmissionsOBDDTCByStatusMask sub-function parameter, as well as the masking principle to request the server to report emissions-related OBD DTCs which are of the status "confirmedDTC" and "warningIndicatorRequested (MIL = ON)" in conjunction with unsupported masking bits. This example shows a typical OBD Scan Tool type request for emissions-related OBD DTCs which cause the MIL to be turned ON and therefore do not pass the I/M (Inspection and Maintenance) test.

11.3.5.17.2 Example #16 assumptions

The server does not support bit 0 (testFailed), bit 4 (testNotCompletedSinceLastClear) and bit 5 (testFailedSinceLastClear) for masking purposes. This results in a DTCStatusAvailabilityMask value of 0xCE ($1100\ 1110_b$).

The client uses a DTC status mask with the value of 0x88 ($1000\ 1000_b$) because only DTCs with the status "confirmedDTC = 1" and " warningIndicatorRequested = 1" shall be displayed to the technician. 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 P010A-14 Mass or Volume Air Flow "A" - circuit short to ground or open (0x010A14), statusOfDTC 0x00 ($0000\ 0000_b$):

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 339 — statusOfDTC = 0x00 of DTC P010A-14

statusOfDTC: bit field name	Bit #	Bit state	Description
testFailed	0	0	Not applicable
testFailedThisOperationCycle	1	0	DTC never failed on 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	Not applicable
testFailedSinceLastClear	5	0	Not applicable
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 P0180-17 Fuel Temperature Sensor A - circuit voltage above threshold (0x018017), statusOfDTC of 0x8E ($1000\ 1110_b$):

Table 340 — statusOfDTC = 0x8E of DTC P0180-17

statusOfDTC: bit field name	Bit #	Bit state	Description
testFailed	0	0	Not applicable
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	Not applicable
testFailedSinceLastClear	5	0	Not applicable
testNotCompletedThisOperationCycle	6	0	DTC test completed this operation cycle
warningIndicatorRequested	7	1	Server is requesting warningIndicator to be active (OBD DTC)

The following assumptions apply to DTC P0190-1D Fuel Rail Pressure Sensor "A" - circuit current out of range (0x01901D), statusOfDTC of 0x8E ($1000\ 1110_b$):

Table 341 — statusOfDTC = 0x8E of DTC P0190-1D

statusOfDTC: bit field name	Bit #	Bit state	Description
testFailed	0	0	Not applicable
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	Not applicable
testFailedSinceLastClear	5	0	Not applicable
testNotCompletedThisOperationCycle	6	0	DTC test completed this operation cycle
warningIndicatorRequested	7	1	Server is requesting warningIndicator to be active (OBD DTC)

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

11.3.5.17.3 Example #16 message flow

In the following example, P0180-17 (0x018017) and P0190-1D (0x01901D) are returned to the client's request.

NOTE The server shall bypass masking on those status bits it doesn't support.

Table 342 — ReadDTCInformation, sub-function = reportEmissionsOBDDTCByStatusMask, 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	0x19	RDTCI
#2	sub-function = reportEmissionsOBDDTCByStatusMask, suppressPosRspMsgIndicationBit = FALSE	0x13	ROBDDTCBSM
#3	DTCStatusMask	0x88	DTCSM

Table 343 — ReadDTCInformation, sub-function = reportDTCByStatusMask, 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	0x59	RDTCIPR
#2	reportType = reportEmissionsOBDDTCByStatusMask	0x13	ROBDDTCBSM
#3	DTCStatusAvailabilityMask	0xCE	DTCSAM
#8	DTCAndStatusRecord#1 [DTCHighByte]	0x01	DTCHB
#9	DTCAndStatusRecord#1 [DTCMiddleByte]	0x80	DTCMB
#10	DTCAndStatusRecord#1 [DTCLowByte]	0x17	DTCLB
#11	DTCAndStatusRecord#1 [statusOfDTC]	0x8E	SODTC
#12	DTCAndStatusRecord#2 [DTCHighByte]	0x01	DTCHB
#13	DTCAndStatusRecord#2 [DTCMiddleByte]	0x90	DTCMB
#14	DTCAndStatusRecord#2 [DTCLowByte]	0x1D	DTCLB
#15	DTCAndStatusRecord#2 [statusOfDTC]	0x8E	SODTC

11.3.5.18 Example #17 - ReadDTCInformation, sub-function = reportDTCExtDataRecordByRecordNumber

11.3.5.18.1 Example #17 overview

This example demonstrates the usage of the reportDTCExtDataRecordByRecordNumber sub-function parameter.

11.3.5.18.2 Example #17 assumptions

The following assumptions apply:

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

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 0x05.

c) Assume that DTC 0x123456 has a statusOfDTC of 0x24, and that the following extended data is available for the DTC.

d) The DTCExtendedData is referenced via the DTCExtDataRecordNumber 0x05

e) Assume that DTC 0x234561 has a statusOfDTC of 0x24, and that the following extended data is available for the DTC.

f) The DTCExtendedData is referenced via the DTCExtDataRecordNumber 0x05.

Table 344 — DTCExtDataRecordNumber 0x05 content for DTC 0x123456

Data Byte	DTCExtDataRecord Contents for DTCExtDataRecordNumber 0x05	Byte Value
#1	Warm-up Cycle Counter – Number of warm up cycles since the DTC commanded the MIL to switch off	0x17

Table 345 — DTCExtDataRecordNumber 0x05 content for DTC 0x234561

Data Byte	DTCExtDataRecord Contents for DTCExtDataRecordNumber 0x05	Byte Value
#1	Warm-up Cycle Counter – Number of warm up cycles since the DTC commanded the MIL to switch off	0x79

11.3.5.18.3 Example #17 message flow

In the following example, a DTCMaskRecord including the DTC number and a DTCExtDataRecordNumber with the value of 0x05 (report all DTCExtDataRecords) is requested by the client. The server returns two DTCs which have recorded the DTCExtDataRecordNumber submitted by the client.

Table 346 — ReadDTCInformation, sub-function = reportDTCExtDataRecordByRecordNumber, 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	0x19	RDTCI
#2	sub-function = reportDTCExtDataRecordByRecordNumber, suppressPosRspMsgIndicationBit = FALSE	0x16	RDTCEDRBDN
#3	DTCExtDataRecordNumber	0x05	DTCEDRN

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 347 — ReadDTCInformation, sub-function = reportDTCExtDataRecordByRecordNumber, 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	0x59	RDTCIPR
#2	reportType = reportDTCExtDataRecordByRecordNumber	0x16	RDTCEDRBDN
#3	DTCAndStatusRecord [DTCHighByte]	0x12	DTCHB
#4	DTCAndStatusRecord [DTCMiddleByte]	0x34	DTCMB
#5	DTCAndStatusRecord [DTCLowByte]	0x56	DTCLB
#6	DTCAndStatusRecord [statusOfDTC]	0x24	SODTC
#7	DTCExtDataRecordNumber	0x05	DTCEDRN
#8	DTCExtDataRecord [byte#1]	0x17	ED_1
#9	DTCAndStatusRecord [DTCHighByte]	0x23	DTCHB
#10	DTCAndStatusRecord [DTCMiddleByte]	0x45	DTCMB
#11	DTCAndStatusRecord [DTCLowByte]	0x61	DTCLB
#12	DTCAndStatusRecord [statusOfDTC]	0x24	SODTC
#13	DTCExtDataRecordNumber	0x05	DTCEDRN
#14	DTCExtDataRecord [byte#1]	0x79	ED_1

11.3.5.19 Example #18 - ReadDTCInformation, sub-function = reportWWHOBDDTCByMaskRecord

11.3.5.19.1 Example #18 overview

This example demonstrates the usage of the reportWWHOBDDTCByMaskRecord sub-function parameter for confirmed DTCs (DTC status mask 0x08). The vehicle uses a CAN bus which connects two emissions-related servers.

The client uses the following request parameter settings:

— FunctionalGroupIdentifier = 0x33 (emissions system group),

— DTCSeverityMaskRecord.DTCSeverityMask = 0xFF (report DTCs with any severity and Class status),

— DTCSeverityMaskRecord.DTCStatusMask = 0x08 (report DTCs with confirmedDTC status = '1').

The servers support the following settings:

— FunctionalGroupIdentifier = 0x33 (emissions system group),

— DTCStatusAvailabilityMask = 0xFF,

— DTCSeverityAvailabilityMask = 0xFF,

— DTCFormatIdentifier = SAE_J2012-DA_DTCFormat_04 = 0x04.

11.3.5.19.2 Example #18 assumptions

All assumptions of example #1 apply.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

11.3.5.19.3 Example #18 message flow

In the following example server #1 only reports DTC P2522-1F A/C Request “B” - circuit intermittent (0x25221F) because the statusOfDTC of 0x2F (0010 1111 binary) matches the client defined status mask of 0x08 (0000 1000 $_b$). Server #2 reports DTC P0235-12 Turbocharger/Supercharger Boost Sensor "A" – circuit short to battery because the statusOfDTC of 0x2E (0010 1110 $_b$) matches the client defined status mask of 0x08 (0000 1000 $_b$).

Table 348 — ReadDTCInformation request, sub-function = reportNumberOfDTCByStatusMask

Message direction	**client → server**		
Message Type	**Request**		
A_Data Byte	**Description (all values are in hexadecimal)**	**Byte Value**	**Mnemonic**
#1	ReadDTCInformation Request SID	0x19	RDTCI
#2	sub-function = reportWWHOBDDTCByMaskRecord, suppressPosRspMsgIndicationBit = FALSE	0x42	RWWHOBDDTCBMR
#3	FunctionalGroupIdentifier (FunctionalGroupIdentifier=emissions=0x33)	0x33	FGID
#4	DTCSeverityMaskRecord[] = [DTCStatusMask]	0x08	DTCSM
#5	DTCSeverityMaskRecord[] = [DTCSeverityMask]	0xFF	DTCSVM

Table 349 — ReadDTCInformation response, sub-function = 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	0x59	RDTCIPR
#2	reportType = reportWWHOBDDTCByMaskRecord	0x42	RWWHOBDDTCBMR
#3	FunctionalGroupIdentifier (FunctionalGroupIdentifier=emissions=0x33)	0x33	FGID
#4	DTCStatusAvailabilityMask	0xFF	DTCSAM
#5	DTCSeverityAvailabilityMask	0xFF	DTCSVAM
#6	DTCFormatIdentifier = [SAE_J2012-DA_DTCFormat_04]	0x04	J2012-DADTCF04
#7	DTCAndSeverityRecord[DTCSeverity#1]	0x20	DTCASR_DTCS
#8	DTCAndSeverityRecord[DTCHighByte#1]	0x25	DTCASR_DTCHB
#9	DTCAndSeverityRecord[DTCMiddleByte#1]	0x22	DTCASR_DTCMB
#10	DTCAndSeverityRecord[DTCLowByte#1]	0x1F	DTCASR_DTCLB
#11	DTCAndSeverityRecord[statusOfDTC#1]	0x2F	DTCASR_SODTC

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 350 — ReadDTCInformation response, sub-function = 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	0x59	RDTCIPR
#2	reportType = reportWWHOBDDTCByMaskRecord	0x42	RWWHOBDDTCBMR
#3	FunctionalGroupIdentifier (FunctionalGroupIdentifier=emissions=0x33)	0x33	FGID
#4	DTCStatusAvailabilityMask	0xFF	DTCSAM
#5	DTCSeverityAvailabilityMask	0xFF	DTCSVAM
#6	DTCFormatIdentifier = [SAE_J2012-DA_DTCFormat_04]	0x04	J2012-DADTCF04
#7	DTCAndSeverityRecord[DTCSeverity#1]	0x20	DTCASR_DTCS
#8	DTCAndSeverityRecord[DTCHighByte#1]	0x02	DTCASR_DTCHB
#9	DTCAndSeverityRecord[DTCMiddleByte#1]	0x35	DTCASR_DTCMB
#10	DTCAndSeverityRecord[DTCLowByte#1]	0x12	DTCASR_DTCLB
#11	DTCAndSeverityRecord[statusOfDTC#1]	0x2E	DTCASR_SODTC

12 InputOutput Control functional unit

12.1 Overview

Table 351 defines the InputOutput Control functional unit.

Table 351 — InputOutput Control functional unit

Service	Description
InputOutputControlByIdentifier	The client requests the control of an input/output specific to the server.

12.2 InputOutputControlByIdentifier (0x2F) service

12.2.1 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 manafuacturer 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.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

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.

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 7.5.

12.2.2 Request message

12.2.2.1 Request message definition

Table 352 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	InputOutputControlByIdentifier Request SID	M	0x2F	IOCBI
	dataIdentifier [] = [			IOI_
#2	byte#1 (MSB)	M	0x00 – 0xFF	B1
#3	byte#2 (LSB)]	M	0x00 – 0xFF	B2
	controlOptionRecord [] = [			CSR_
#4	inputOutputControlParameter	M_1	0x00 – 0xFF	IOCP_
:	controlState#1	C_1	0x00 – 0xFF	CS_
#4+(m-1)	:	:	:	:
	controlState#m]	C_1	0x00 – 0xFF	CS_
	controlEnableMaskRecord#1[] = [			CEM_
#4+m	controlMask#1	C_2	0x00 – 0xFF	CM_
:	:	:	:	:
#4+m+(r-1)	controlMask#r]	C_2	0x00 – 0xFF	CM_

M_1 InputOutputControlParameter shall be implemented as defined in E.1.

C_1: The presence of this parameter depends on the dataIdentifier and the inputOutputControlParameter (see 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)

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

This service does not use a sub-function parameter.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

 © ISO 2013 – All rights reserved

12.2.2.3 Request message data-parameter definition

The following data-parameters are defined for this service:

Table 353 — Request message data-parameter definition

Definition
dataIdentifier This parameter identifies an 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 E.1.
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.

12.2.3 Positive response message

12.2.3.1 Positive response message definition

Table 354 — Positive response message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	InputOutputControlByIdentifier Response SID	M	0x6F	IOCBIPR
	dataIdentifier [] = [			IOI_
#2	byte#1 (MSB)	M	0x00 – 0xFF	B1
#3	byte#2 (LSB)]	M	0x00 – 0xFF	B2
	controlStatusRecord [] = [			CSR_
#4	inputOutputControlParameter	M	0x00 – 0xFF	IOCP_
#5	controlState#1	C_1	0x00 – 0xFF	CS_
:	:	:	:	:
#5+(m-1)	controlState#m]	C_1	0x00 – 0xFF	CS_
C_1: The presence of this parameter depends on the dataIdentifier and the inputOutputControlParameter (see E.1).				

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

12.2.3.2 Positive response message data-parameter definition

Table 355 — 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 E.1

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

Table 356 — Supported negative response codes

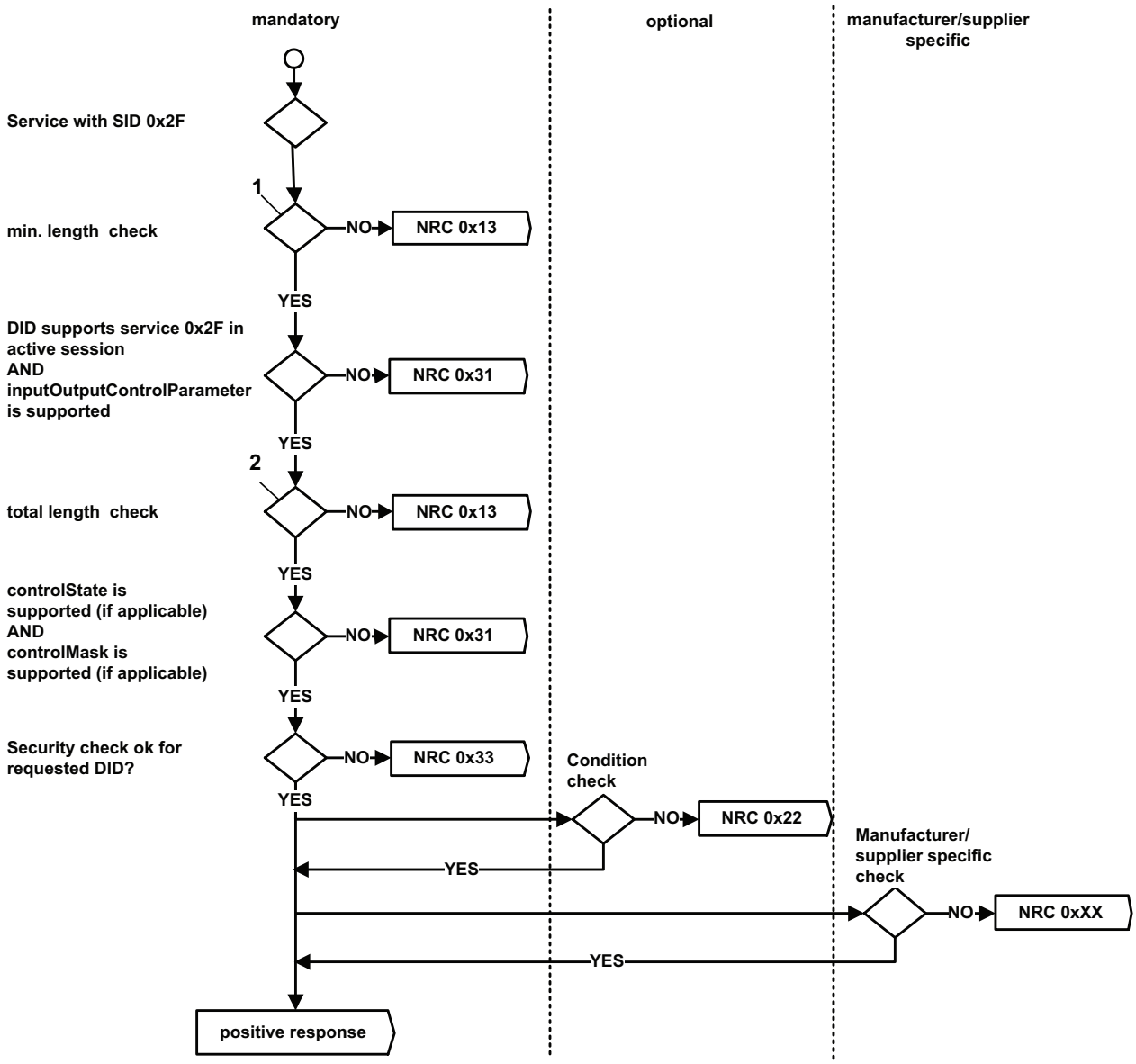
NRC	Description	Mnemonic
0x13	**incorrectMessageLengthOrInvalidFormat** This NRC shall be sent if the length of the message is wrong.	IMLOIF
0x22	**conditionsNotCorrect** This NRC shall be returned if the criteria for the request InputOutputControl are not met.	CNC
0x31	**requestOutOfRange** This NRC shall be sent if: — the requested dataIdentifier value is not supported by the device; — the value contained in the inputOuptputControlParameter 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
0x33	**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

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

The evaluation sequence is documented in Figure 24.



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 24 — NRC handling for InputOutputControlByIdentifier service

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

12.2.5 Message flow example(s) InputOutputControlByIdentifier

12.2.5.1 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.

12.2.5.2 Example #1 - ”Air Inlet Door Position” shortTermAdjustment

The parameter being controlled is the "Air Inlet Door Position" associated with dataIdentifier (0x9B00).

Conversion: Air Inlet Door Position [%] = decimal(Hex) * 1 [%]

12.2.5.2.1 Step #1: ReadDataByIdentifier

This example uses the ReadDataByIdentifier service to read the current state of the Air Inlet Door Position.

Table 357 — 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	0x22	RDBI
#2	dataIdentifier [byte#1] = 0x9B	0x9B	DID_B1
#3	dataIdentifier [byte#2] = 0x00 (“Air Inlet Door Position”)	0x00	DID_B2

Table 358 — 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	0x62	RDBIPR
#2	dataIdentifier [byte#1] = 0x9B	0x9B	DID_B1
#3	dataIdentifier [byte#2] = 0x00 (“Air Inlet Door Position”)	0x00	DID_B2
#4	dataRecord [data#1] = 10%	0x0A	DREC_DATA1

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

12.2.5.2.2 Step #2: shortTermAdjustment

Table 359 — 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	0x2F	IOCBI
#2	dataIdentifier [byte#1] = 0x9B	0x9B	IOI_B1
#3	dataIdentifier [byte#2] = 0x00 (“Air Inlet Door Position”)	0x00	IOI_B2
#4	controlOptionRecord [inputOutputControlParameter] = shortTermAdjustment	0x03	IOCP_STA
#5	controlOptionRecord [controlState#1] = 60%	0x3C	CS_1

Table 360 — 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	0x6F	IOCBLIPR
#2	dataIdentifier [byte#1] = 0x9B	0x9B	IOI_B1
#3	dataIdentifier [byte#2] = 0x00 (“Air Inlet Door Position”)	0x00	IOI_B2
#4	controlStatusRecord [inputOutputControlParameter] = shortTermAdjustment	0x03	IOCP_STA
#5	controlStatusRecord [controlState#1] = 12%	0x0C	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%.

12.2.5.2.3 Step #3: ReadDataByIdentifier

This example uses the readDataByIdentifier service to read the current state of the Air Inlet Door Position.

Table 361 — 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	0x22	RDBI
#2	dataIdentifier [byte#1] = 0xB9	0x9B	DID_B1
#3	dataIdentifier [byte#2] = 0x00 (“Air Inlet Door Position”)	0x00	DID_B2

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 362 — 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	0x62	RDBIPR
#2	dataIdentifier [byte#1] = 0x9B	0x9B	DID_B1
#3	dataIdentifier [byte#2] = 0x00 (“Air Inlet Door Position”)	0x00	DID_B2
#4	dataRecord [data#1] = 60%	0x3C	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.

12.2.5.2.4 Step #4: returnControlToECU

Table 363 — 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	0x2F	IOCBI
#2	dataIdentifier [byte#1] = 0x9B	0x9B	IOI_B1
#3	dataIdentifier [byte#2] = 0x00 (“Air Inlet Door Position”)	0x00	IOI_B2
#4	controlOptionRecord [inputOutputControlParameter] = returnControlToECU	0x00	RCTECU

Table 364 — 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	0x6F	IOCBIPR
#2	dataIdentifier [byte#1] = 0x9B	0x9B	IOI_B1
#3	dataIdentifier [byte#2] = 0x00 (“Air Inlet Door Position”)	0x00	IOI_B2
#4	controlStatusRecord [inputOutputControlParameter] = returnControlToECU	0x00	RCTECU
#5	controlStatusRecord [controlState#1] = 58%	0x3A	CS_1

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

12.2.5.2.5 Step #5: freezeCurrentState

Table 365 — 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	0x2F	IOCBI
#2	dataIdentifier [byte#1] = 0x9B	0x9B	IOI_B1
#3	dataIdentifier [byte#2] = 0x00 (“Air Inlet Door Position”)	0x00	IOI_B2
#4	controlOptionRecord [inputOutputControlParameter] = freezeCurrentState	0x02	IOCP_FCS

Table 366 — 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	0x6F	IOCBIPR
#2	dataIdentifier [byte#1] = 0x9B	0x9B	IOI_B1
#3	dataIdentifier [byte#2] = 0x00 (“Air Inlet Door Position”)	0x00	IOI_B2
#4	controlStatusRecord [inputOutputControlParameter] = freezeCurrentState	0x02	IOCP_FCS
#5	controlStatusRecord [controlState#1] = 50%	0x32	CS_1

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

12.2.5.3 Example #2 – EGR and IAC shortTermAdjustment

12.2.5.3.1 Assumptions

This example uses a packeted dataIdentifier 0x0155 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 0x0155. The dataIdentifier supports five individual parameters as described in Table 367 below.

Table 367 — Composite data blocks – DataIdentifier definitions – Example #2

DID	Data Byte	Parameter		Data Record Contents
		Number	Size	
0x0155	#1 (all bits)	#1	8 bits	dataRecord [data#1] = IAC Pintle Position (n = counts)
	#2 - #3 (all bits)	#2	16 bits	dataRecord [data#2-#3] = RPM (0 = 0 U/min, 65 535 = 65 535 U/min)
	#4 (bits 7-4)	#3	4 bits	dataRecord [data#4 (bits 7-4)] = Pedal Position A: Linear Scaling, 0 = 0%, 15 = 120 %
	#4 (bits 3-0)	#4	4 bits	dataRecord [data#4 (bits 3-0)] = Pedal Position B: Linear Scaling, 0 = 0%, 15 = 120 %
	#5 (all bits)	#5	8 bits	dataRecord [data#5] = EGR Duty Cycle: Linear Scaling, 0 counts = 0%, 255 counts = 100 %

DataIdentifier 0x0155 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 368.

Table 368 — ControlEnableMaskRecord– Example #2

ControlEnableMaskRecord for dataIdenitifier 0x0155. Total size = 1 byte (i.e., consists only of ControlEnableMask#1)		
Bit Position		**ControlEnableMask#1 – Bit Meaning (1 = affected, 0 = not affected)**
7	(Most Significant Bit)	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		No affect due to no corresponding parameter
1		No affect due to no corresponding parameter
0	(Least Significant Bit)	No affect due to no corresponding parameter

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

12.2.5.3.2 Case #1: Control IAC Pintle Position only

Table 369 defines the InputOutputControlByIdentifier request message flow example #2 – Case #1.

Table 369 — 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	0x2F	IOCBI
#2	dataIdentifier [byte#1] = 0x01	0x01	IOI_B1
#3	dataIdentifier [byte#2] = 0x55 (IAC / RPM / PPA / PPB / EGR)	0x55	IOI_B2
#4	controlOptionRecord [inputOutputControlParameter] = shortTermAdjustment	0x03	IOCP_STA
#5	controlOptionRecord [controlState#1] = IAC Pintle Position (7 counts)	0x07	CS_1
#6	controlOptionRecord [controlState#2] = RPM (XX)	0xXX	CS_2
#7	controlOptionRecord [controlState#3] = RPM (XX)	0xXX	CS_3
#8	controlOptionRecord [controlState#4] = Pedal Position A (Y) and B (Z)	0xYZ	CS_4
#9	controlOptionRecord [controlState#5] = EGR Duty Cycle (XX)	0xXX	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 370 defines the InputOutputControlByIdentifier positive response message flow example #2 – Case #1.

Table 370 — 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	0x6F	IOCBLIPR
#2	dataIdentifier [byte#1] = 0x01	0x01	IOI_B1
#3	dataIdentifier [byte#2] = 0x55 (IAC / RPM / PPA / PPB / EGR)	0x55	IOI_B2
#4	controlStatusRecord [inputOutputControlParameter] = shortTermAdjustment	0x03	IOCP_STA
#5	controlStatusRecord [controlState#1] = IAC Pintle Position (7 counts)	0x07	CS_1
#6	controlStatusRecord [controlState#2] = RPM (750 U/min)	0x02	CS_2
#7	controlStatusRecord [controlState#3] = RPM	0xEE	CS_3
#8	controlStatusRecord [controlState#4] = Pedal Position A (8%), Pedal Position B (16%)	0x12	CS_4
#9	controlStatusRecord [controlState#5] = EGR Duty Cycle (35%)	0x59	CS_5

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

NOTE The value transmitted for all parameters in controlState#1 – controlState#5 shall reflect the current state of the system.

12.2.5.3.3 Case #2: Control RPM Only

Table 371 defines the InputOutputControlByIdentifier request message flow example #2 – Case #2.

Table 371 — 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	0x2F	IOCBI
#2	dataIdentifier [byte#1] = 0x01	0x01	IOI_B1
#3	dataIdentifier [byte#2] = 0x55 (IAC / RPM / EGR)	0x55	IOI_B2
#4	controlOptionRecord [inputOutputControlParameter] = shortTermAdjustment	0x03	IOCP_STA
#5	controlOptionRecord [controlState#1] = IAC Pintle Position (XX counts)	0xXX	CS_1
#6	controlOptionRecord [controlState#2] = RPM (0x03E8 = 1000 U/min)	0x03	CS_2
#7	controlOptionRecord [controlState#3] = RPM	0xE8	CS_3
#8	controlOptionRecord [controlState#4] = Pedal Position A (Y) and B (Z)	0xYZ	CS_4
#9	controlOptionRecord [controlState#5] = EGR Duty Cycle (XX)	0xXX	CS_5
#10	controlEnableMask [controlMask#1] = Control RPM ONLY	0x40	CM_1

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.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

 © ISO 2013 – All rights reserved

Table 372 defines the InputOutputControlByIdentifier positive response message flow example #2 – Case #2.

Table 372 — 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	0x6F	IOCBLIPR
#2	dataIdentifier [byte#1] = 0x01	0x01	IOI_B1
#3	dataIdentifier [byte#2] = 0x55 (IAC / RPM / PPA / PPB / EGR)	0x55	IOI_B2
#4	controlStatusRecord [inputOutputControlParameter] = shortTermAdjustment	0x03	IOCP_STA
#5	controlStatusRecord [controlState#1] = IAC Pintle Position (9 counts)	0x09	CS_1
#6	controlStatusRecord [controlState#2] = RPM (950 U/min)	0x03	CS_2
#7	controlStatusRecord [controlState#3] = RPM	0xB6	CS_3
#8	controlStatusRecord [controlState#4] = Pedal Position A (8 %), Pedal Position B (16 %)	0x12	CS_4
#9	controlStatusRecord [controlState#5] = EGR Duty Cycle (35 %)	0x59	CS_5

NOTE The value transmitted for all parameters in controlState#1 – controlState#5 shall reflect the current state of the system.

12.2.5.3.4 Case #3: Control both Pedal Position A and EGR Duty Cycle

Table 373 defines the InputOutputControlByIdentifier request message flow example #2 – Case #3.

Table 373 — 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	0x2F	IOCBI
#2	dataIdentifier [byte#1] = 0x01	0x01	IOI_B1
#3	dataIdentifier [byte#2] = 0x55 (IAC / RPM / PPA / PPB / EGR)	0x55	IOI_B2
#4	controlOptionRecord [inputOutputControlParameter] = shortTermAdjustment	0x03	IOCP_STA
#5	controlOptionRecord [controlState#1] = IAC Pintle Position (XX)	0xXX	CS_1
#6	controlOptionRecord [controlState#2] = RPM (XX)	0xXX	CS_2
#7	controlOptionRecord [controlState#3] = RPM (XX)	0xXX	CS_3
#8	controlOptionRecord [controlState#4] = Pedal Position A (0x3 = 24 %), Pedal Position B (Z)	0x3Z	CS_4
#9	controlOptionRecord [controlState#5] = EGR Duty Cycle (45 %)	0x72	CS_5
#10	controlEnableMask [controlMask#1] = Control Pedal Position A and EGR	28	CM_1

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

NOTE The values transmitted for IAC Pintle Position, RPM and Pedal Position B in controlState#1 - #3 and controlState#4 (bits 3-0) 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 374 defines the InputOutputControlByIdentifier positive response message flow example #2 – Case #3.

Table 374 — 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	0x6F	IOCBLIPR
#2	dataIdentifier [byte#1] = 0x01	0x01	IOI_B1
#3	dataIdentifier [byte#2] = 0x55 (IAC / RPM / PPA / PPB / EGR)	0x55	IOI_B2
#4	controlStatusRecord [inputOutputControlParameter] = shortTermAdjustment	0x03	IOCP_STA
#5	controlStatusRecord [controlState#1] = IAC Pintle Position (7 counts)	0x07	CS_1
#6	controlStatusRecord [controlState#2] = RPM (850 U/min)	0x03	CS_2
#7	controlStatusRecord [controlState#3] = RPM	0x52	CS_3
#8	controlStatusRecord [controlState#4] = Pedal Position A (24%) Pedal Position B (16%)	0x32	CS_4
#9	controlStatusRecord [controlState#4] = EGR Duty Cycle (41%)	0x69	CS_5

NOTE The value transmitted for all parameters in controlState#1 – controlState#5 shall reflect the current state of the system.

12.2.5.3.5 Case #4: Return control of all parameters to the ECU

Table 375 defines the InputOutputControlByIdentifier request message flow example #2 – Case #4.

Table 375 — 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	0x2F	IOCBI
#2	dataIdentifier [byte#1] = 0x01	0x01	IOI_B1
#3	dataIdentifier [byte#2] = 0x55 (IAC / RPM / PPA / PPB / EGR)	0x55	IOI_B2
#4	controlOptionRecord [inputOutputControlParameter] = returnControlToECU	0x00	RCTECU
#5	controlEnableMask [controlMask#1] = All elemental parameters	0xFF	CM_1

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

 © ISO 2013 – All rights reserved

Table 376 defines the InputOutputControlByIdentifier positive response message flow example #2 – Case #4.

Table 376 — 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	0x6F	IOCBLIPR
#2	dataIdentifier [byte#1] = 0x01	0x01	IOI_B1
#3	dataIdentifier [byte#2] = 0x55 (IAC / RPM / PPA / PPB / EGR)	0x55	IOI_B2
#4	controlStatusRecord [inputOutputControlParameter] = returnControlToECU	0x00	RCTECU
#5	controlStatusRecord [controlState#1] = IAC Pintle Position (9 counts)	0x09	CS_1
#6	controlStatusRecord [controlState#2] = RPM (850 U/min)	0x03	CS_2
#7	controlStatusRecord [controlState#3] = RPM	0x52	CS_3
#8	controlStatusRecord [controlState#4] = Pedal Position A (8%) Pedal Position B (16%)	0x12	CS_4
#9	controlStatusRecord [controlState#4] = EGR Duty Cycle (35%)	0x59	CS_5

NOTE The value transmitted for all parameters in controlState#1 – controlState#5 shall reflect the current state of the system.

13 Routine functional unit

13.1 Overview

Table 377 defines the Routine functional unit.

Table 377 — 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 may 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.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

— 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.

13.2 RoutineControl (0x31) service

13.2.1 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.

13.2.1.1 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 7.5.

13.2.1.2 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.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

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.

13.2.1.3 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.

NOTE The server routine shall be stopped any time as programmed or previously initialized in the server's memory.

13.2.1.4 Request routine results referenced by a routineIdentifier

This sub-function 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 sub-function parameter (e.g. normal / abnormal Exit With Results) the requestRoutineResults sub-function 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.

13.2.2 Request message

13.2.2.1 Request message definition

Table 378 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	RoutineControl Request SID	M	0x31	RC
#2	sub-function = [routineControlType]	M	 0x00 – 0xFF	LEV_ RCTP_
 #3 #4	routineIdentifier [] = [byte#1 (MSB) byte#2 (LSB)]	 M M	 0x00 – 0xFF 0x00 – 0xFF	RI_ B1 B2
 #5 : #n	routineControlOptionRecord[] = [routineControlOption#1 : routineControlOption#m]	 C/U : C/U	 0x00 – 0xFF : 0x00 – 0xFF	RCEOR_ RCO_ : RCO _
C: This parameter is user optional to be present for sub-function parameter startRoutine and stopRoutine.				

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

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

The sub-function parameters are used by this service to select the control of the routine. Explanations and usage of the possible levels are detailed in Table 379 (suppressPosRspMsgIndicationBit (bit 7) not shown).

Table 379 — Request message sub-function definition

Bits 6 – 0	Description	Cvt	Mnemonic
0x00	**ISOSAEReserved** This value is reserved by this document for future definition.	M	ISOSAERESRVD
0x01	**startRoutine** This parameter specifies that the server shall start the routine specified by the routineIdentifier.	M	STR
0x02	**stopRoutine** This parameter specifies that the server shall stop the routine specified by the routineIdentifier.	U	STPR
0x03	**requestRoutineResults** This parameter specifies that the server shall return result values of the routine specified by the routineIdentifier.	U	RRR
0x04 – 0x7F	**ISOSAEReserved** This value is reserved by this document for future definition.	M	ISOSAERESRVD

13.2.2.3 Request message data-parameter definition

Table 380 defines the data-parameters of the request message.

Table 380 — Request message data-parameter definition

Definition
routineIdentifier This parameter identifies a server local routine and is out of the range of defined dataIdentifiers (see 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.).

13.2.3 Positive response message

13.2.3.1 Positive response message definition

Table 381 defines the positive response message.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 381 — Positive response message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	RoutineControl Response SID	M	0x71	RCPR
#2	routineControlType	M	00-7F	RCTP_
#3 #4	routineIdentifier [] = [byte#1 (MSB) byte#2 (LSB)]	M M	0x00 – 0xFF 0x00 – 0xFF	RI_ B1 B2
#5	routineInfo	C_1	0x00 – 0xFF	RINF_
#6 : #n	routineStatusRecord[] = [routineStatus#1 : routineStatus#m]	U : U	0x00 – 0xFF : 0x00 – 0xFF	RSR_ RS_ : 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 is only to be included in the routineStatusRecord[] if specified for the routineIdentifier (RID) by the vehicle manufacturer.

13.2.3.2 Positive response message data-parameter definition

Table 382 defines the data-parameters of the positive response message.

Table 382 — Response message data-parameter definition

Definition
routineControlType This parameter is an echo of bits 6 - 0 of the sub-function 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.

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

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 383 — Supported negative response codes

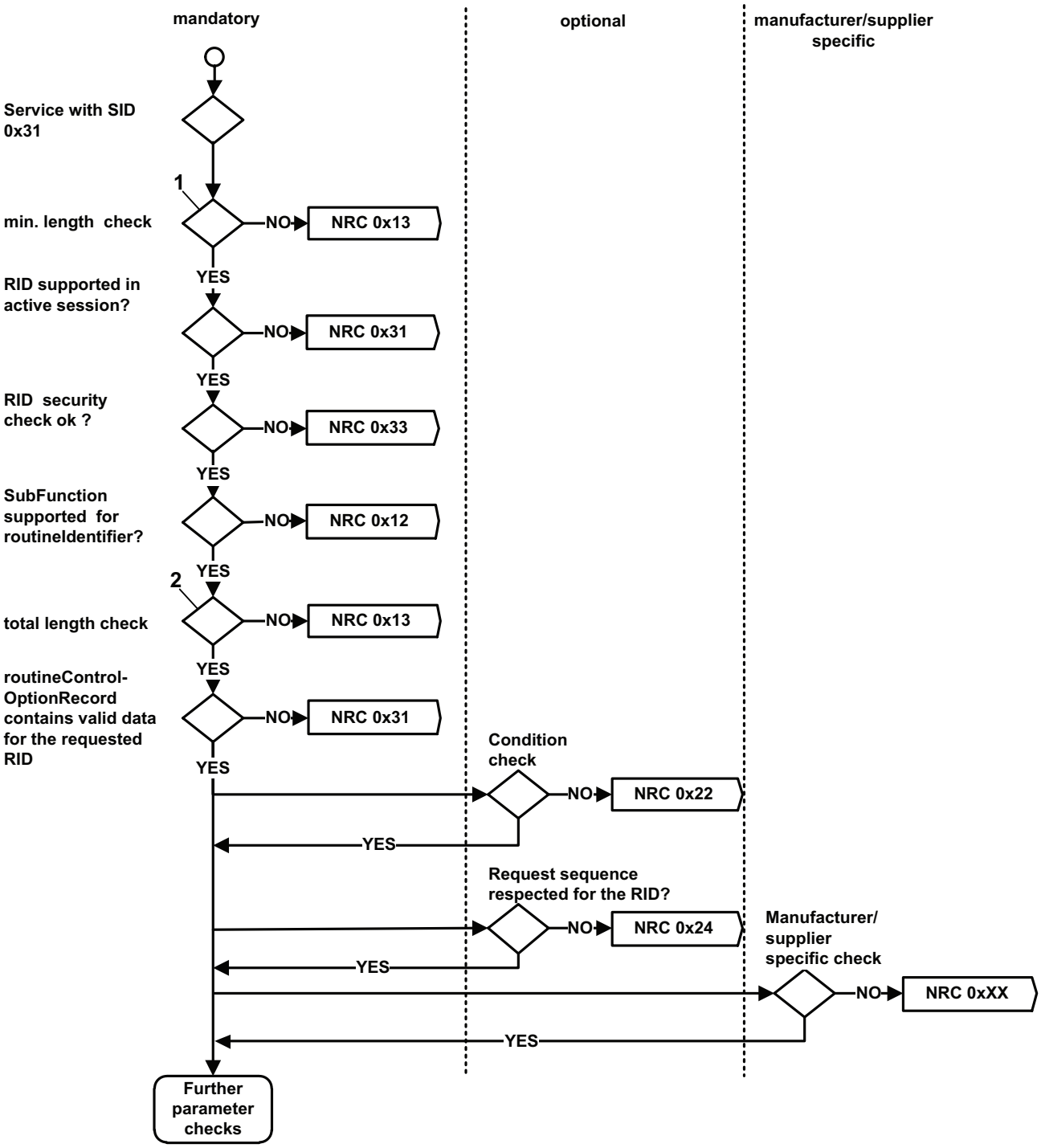
NRC	Description	Mnemonic
0x12	**sub-functionNotSupported** This NRC shall be sent if the requested sub-function is either generally not supported or is not supported for the requested RoutineIdentifier.	SFNS
0x13	**incorrectMessageLengthOrInvalidFormat** This NRC shall be sent if the length of the message is wrong.	IMLOIF
0x22	**conditionsNotCorrect** This NRC shall be returned if the criteria for the request RoutineControl are not met.	CNC
0x24	**requestSequenceError** This NRC shall be returned if • the routine is currently active and can not be restarted when the 'startRoutine' sub-function 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' sub-function is received, • routine results are not available when the 'requestRoutineResults' sub-function is received (e.g., the requested routineIdentifier has never been started).	RSE
0x31	**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
0x33	**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
0x72	**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

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

The evaluation sequence is documented in Figure 25.



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

Figure 25 — NRC handling for RoutineControl service

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

13.2.5 Message flow example(s) RoutineControl

13.2.5.1 Example #1: sub-function = 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 intermittent while a technician would "wiggle" all wiring harness connectors of the system under test. The routineIdentifier references this routine by the routineIdentifier 0x0201.

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 sub-function parameter) to "FALSE" ('0').

Table 384 defines the RoutineControl request message flow - example #1.

Table 384 — 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	0x31	RC
#2	sub-function = startRoutine, suppressPosRspMsgIndicationBit = FALSE	0x01	LEV_STR
#3	routineIdentifier [byte#1] (MSB)	0x02	RI_B1
#4	routineIdentifier [byte#2] (LSB)	0x01	RI_B2

Table 385 defines the positive response message flow - example #1.

Table 385 — 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	0x71	RCPR
#2	routineControlType = startRoutine	0x01	STR
#3	routineIdentifier [byte#1] (MSB)	0x02	RI_B1
#4	routineIdentifier [byte#2] (LSB)	0x01	RI_B2
#5	routineStatusRecord [routineStatus#1] = vehicle manufactuer specific	0x32	RRS_

13.2.5.2 Example #2: sub-function = stopRoutine

This subclause specifies the test conditions to stop a routine in the server which has continuously tested (as fast as possible) all input and output signals on intermittence while a technician would have been "wiggled" all wiring harness connectors of the system under test. The routineIdentifier references this routine by the routineIdentifier 0x0201.

Test conditions: ignition = on, engine = off, vehicle speed = 0 [kph]

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

 © ISO 2013 – All rights reserved

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

Table 386 defines the RoutineControl request message flow - example #2.

Table 386 — 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	0x31	RC
#2	sub-function = stopRoutine, suppressPosRspMsgIndicationBit = FALSE	0x02	SPR
#3	routineIdentifier [byte#1] (MSB)	0x02	RI_B1
#4	routineIdentifier [byte#2] (LSB)	0x01	RI_B2

Table 387 defines the RoutineControl positive response message flow - example #2.

Table 387 — 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	0x71	RCPR
#2	routineControlType = stopRoutine	0x02	SPR
#3	routineIdentifier [byte#1] (MSB)	0x02	RI_B1
#4	routineIdentifier [byte#2] (LSB)	0x01	RI_B2
#5	routineStatusRecord [routineStatus#1] = vehicle manufactuer specific	0x30	RRS_

13.2.5.3 Example #3: sub-function = requestRoutineResults

This example shows how to retrieve result values after a routine has been finished. The routine has continuously tested (as fast as possible) all input and output singals on intermittence while a technician would have been "wiggled" at all wiring harness connectors of the system under test. The routineIdentifier to reference this routine is 0x0201. 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 sub-function parameter) to "FALSE" ('0').

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 388 defines the RequestRoutineResults request message flow – example #3.

Table 388 — 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	0x31	RC
#2	sub-function = requestRoutineResults, suppressPosRspMsgIndicationBit = FALSE	0x03	RRR
#3	routineIdentifier [byte#1] (MSB)	0x02	RI_B1
#4	routineIdentifier [byte#2] (LSB)	0x01	RI_B2

Table 389 defines the RequestRoutineResults positive response message flow – example #3.

Table 389 — 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	0x71	RCPR
#2	routineControlType = requestRoutineResults	0x03	RRR
#3	routineIdentifier [byte#1] (MSB)	0x02	RI_B1
#4	routineIdentifier [byte#2] (LSB)	0x01	RI_B2
#5	routineStatusRecord [routineStatus#1] = Vehicle Manufactuer Specific	0x30	RRS_
#6	routineStatusRecord [routineStatus#2] = inputSignal#1	0x33	RRS_
:	:	:	:
#n	routineStatusRecord [routineStatus#m] = inputSignal#m	0x8F	RRS_

13.2.5.4 Example #4: sub-function = 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 0x0202.

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 sub-function parameter) to "FALSE" ('0').

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 390 defines the RoutineControl request message flow - example #4.

Table 390 — 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	0x31	RC
#2	sub-function = startRoutine, suppressPosRspMsgIndicationBit = FALSE	0x01	STR
#3	routineIdentifier [byte#1] (MSB)	0x02	RI_B1
#4	routineIdentifier [byte#2] (LSB)	0x02	RI_B2
#5	routineControlOption#1 [selected gear] = vehicle manufacturer specific	0x06	RCO_
#6	routineControlOption#2 [test condition]	0x01	RCO_

Table 391 defines the RoutineControl positive response message flow - example #4.

Table 391 — 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	0x71	RCPR
#2	routineControlType = startRoutine	0x01	STR
#3	routineIdentifier [byte#1] (MSB)	0x02	RI_B1
#4	routineIdentifier [byte#2] (LSB)	0x02	RI_B2
#5	routineStatusRecord [routineStatus#1] = vehicle manufacturer specific	0x32	RRS_
#6	routineStatusRecord [routineStatus#2]= response time	0x33	RRS_
.	:	:	:
#n	routineStatusRecord [routineStatus#m]= inputSignal#m	0x8F	RRS_

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

14 Upload Download functional unit

14.1 Overview

Table 392 defines the Upload Download functional unit.

Table 392 — 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.

14.2 RequestDownload (0x34) service

14.2.1 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 7.5.

14.2.2 Request message

14.2.2.1 Request message definition

Table 393 defines the request message.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 393 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	RequestDownload Request SID	M	0x34	RD
#2	dataFormatIdentifier	M	0x00 – 0xFF	DFI_
#3	addressAndLengthFormatIdentifier	M	0x00 – 0xFF	ALFID
#4 : #(m-1)+4	memoryAddress[] = [byte#1 (MSB) : byte#m]	M : C_1	0x00 – 0xFF : 0x00 – 0xFF	MA_ B1 : Bm
#n-(k-1) : #n	memorySize[] = [byte#1 (MSB) : byte#k]	M : C_2	0x00 – 0xFF : 0x00 – 0xFF	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.				

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

This service does not use a sub-function parameter.

14.2.2.3 Request message data-parameter definition

Table 394 defines the data-parameters of the request message.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 394 — 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 0x00 specifies that neither compressionMethod nor encryptingMethod is used. Values other than 0x00 are vehicle manufacturer specific.
addressAndLengthFormatIdentifier This parameter is a one byte value with each nibble encoded separately (see 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 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.

14.2.3 Positive response message

14.2.3.1 Positive response message definition

Table 395 defines the positive response message.

Table 395 — Positive response message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	RequestDownload Response SID	M	0x74	RDPR
#2	lengthFormatIdentifier	M	0x00 – 0xF0	LFID
#3 : #n	maxNumberOfBlockLength = [byte#1 (MSB) : byte#m]	M : M	0x00 – 0xFF : 0x00 – 0xFF	MNROB_ B1 : Bm

14.2.3.2 Positive response message data-parameter definition

Table 396 defines the data-parameters of the positive response message.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 396 — 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 has to be set to '0'.
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). Note that the last transferData request within a given block may 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.

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

Table 397 — Supported negative response codes

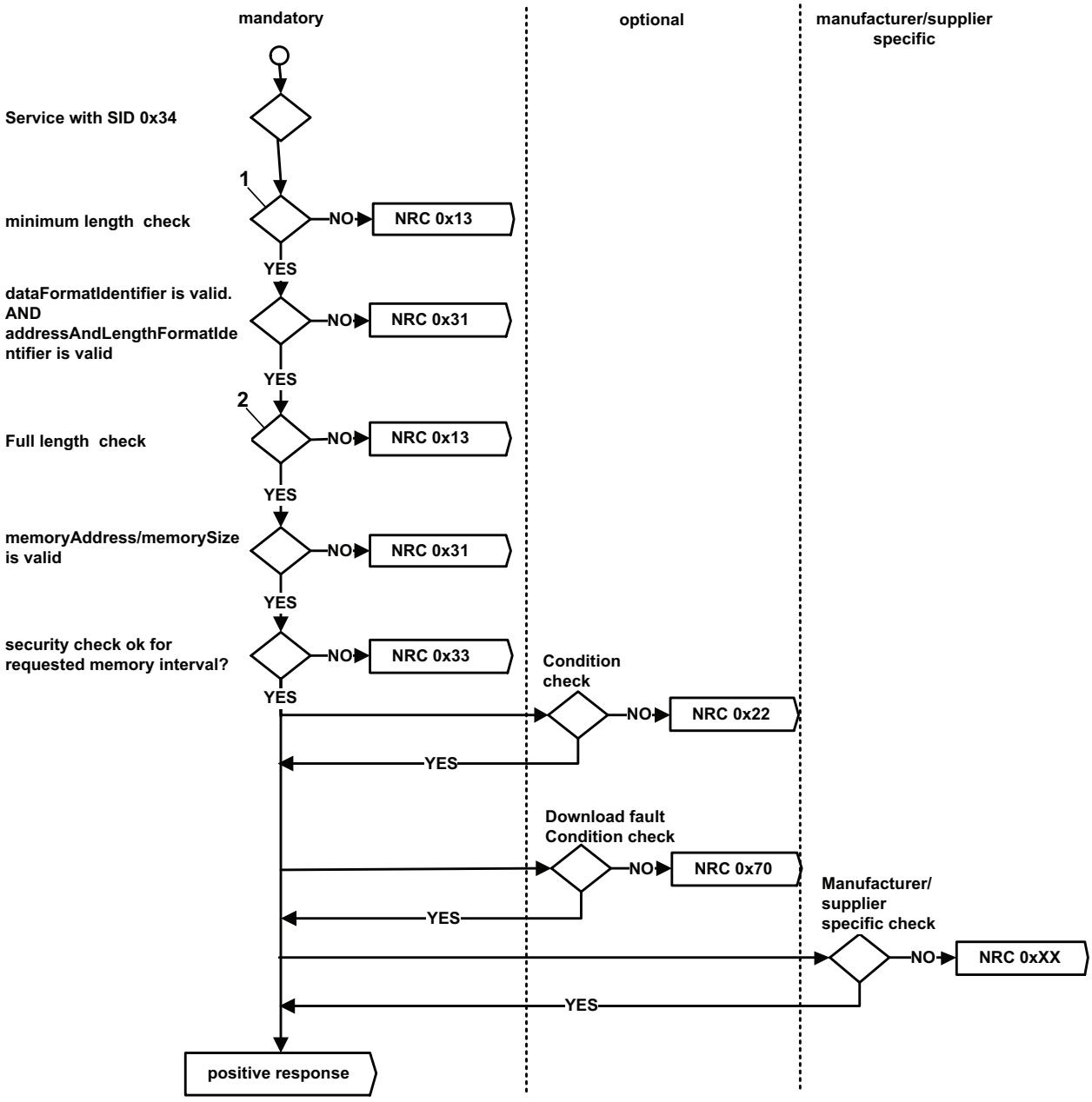
NRC	Description	Mnemonic
0x13	**incorrectMessageLengthOrInvalidFormat** This NRC shall be sent if the length of the message is wrong.	IMLOIF
0x22	**conditionsNotCorrect** This NRC shall be returned if a server receives a request for this service while in the process of receiving a download of a software 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
0x31	**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
0x33	**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
0x70	**uploadDownloadNotAccepted** This NRC indicates that an attempt to download to a server's memory cannot be accomplished due to some fault conditions.	UDNA

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

The evaluation sequence is documented in Figure 26.



Key

1 at least 5 (SI + DFI_ + ALFID + minimum MA_ + minimum MS_)

2 length can be computed from addressAndLengthFormatIdentifier

Figure 26 — NRC handling for RequestDownload service

14.2.5 Message flow example(s) RequestDownload

See 14.5.5 for a complete message flow example.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

14.3 RequestUpload (0x35) service

14.3.1 Service description

The RequestUpload service is used by the client to initiate a data transfer from the server to the client (upload).

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 7.5.

14.3.2 Request message

14.3.2.1 Request message definition

Table 398 defines the request message.

Table 398 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	RequestUpload Request SID	M	0x35	RU
#2	dataFormatIdentifier	M	0x00 – 0xFF	DFI_
#3	addressAndLengthFormatIdentifier	M	0x00 – 0xFF	ALFID
#4 : #(m-1)+4	memoryAddress[] = [byte#1 (MSB) : byte#m]	M : C_1	0x00 – 0xFF : 0x00 – 0xFF	MA_ B1 : Bm
#n-(k-1) : #n	memorySize[] = [byte#1 (MSB) : byte#k]	M : C_2	0x00 – 0xFF : 0x00 – 0xFF	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.				

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

This service does not use a sub-function parameter.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

14.3.2.3 Request message data-parameter definition

Table 399 defines the data-parameters of the request message.

Table 399 — 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 0x00 specifies that neither compressionMethod nor encryptingMethod is used. Values other than 0x00 are vehicle manufacturer specific.
addressAndLengthFormatIdentifier This parameter is a one byte value with each nibble encoded separately (see 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 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.

14.3.3 Positive response message

14.3.3.1 Positive response message definition

Table 400 defines the positive response message.

Table 400 — Positive response message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	RequestUpload Response SID	M	0x75	RUPR
#2	lengthFormatIdentifier	M	0x00 – 0xF0	LFID
#3 : #n	maxNumberOfBlockLength = [byte#1 (MSB) : byte#m]	M : M	0x00 – 0xFF : 0x00 – 0xFF	MNROB_ B1 : Bm

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

14.3.3.2 Positive response message data-parameter definition

Table 401 defines the data-parameters of the positive response message.

Table 401 — 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 0x0; The format of this parameter is compatible to the format of the addressAndLengthFormatIdentifier parameter contained in the request message, except that the lower nibble has to be set to 0x0.
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 may be required to be less than maxNumberOfBlockLength.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

14.3.4 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 402. The listed negative responses shall be used if the error scenario applies to the server.

Table 402 — Supported negative response codes

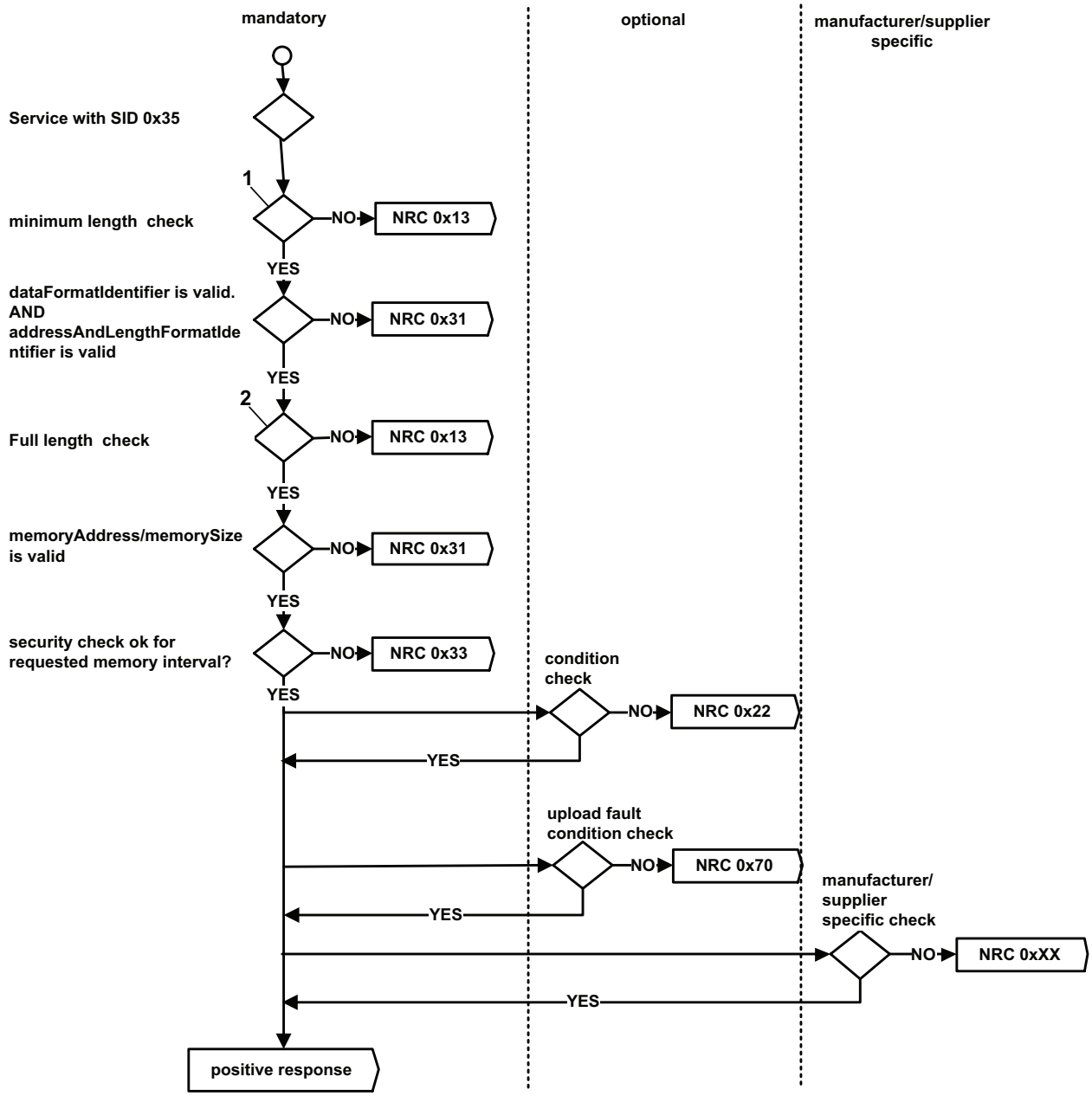
NRC	Description	Mnemonic
0x13	**incorrectMessageLengthOrInvalidFormat** This NRC shall be sent if the length of the message is wrong.	IMLOIF
0x22	**conditionsNotCorrect** This NRC shall be returned if the criteria for the requestUpload are not met. This could occur if a server receives a request for this service while a requestUpload is already active, but not yet completed.	CNC
0x31	**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
0x33	**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
0x70	**uploadDownloadNotAccepted** This NRC indicates that an attempt to upload to a server's memory cannot be accomplished due to some fault conditions.	UDNA

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

The evaluation sequence is documented in Figure 27.



Key

1 at least 5 (SI + DFI_ + ALFID + minimum MA_ + minimum MS_)

2 length can be computed from addressAndLengthFormatIdentifier

Figure 27 — NRC handling for RequestUpload service

14.3.5 Message flow example(s) RequestUpload

See 14.5.5 for a complete message flow example.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

14.4 TransferData (0x36) service

14.4.1 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 (0x34) or RequestUpload (0x35) request message. This means that the first TransferData (0x36) request message following the RequestDownload (0x34) or RequestUpload (0x35) 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 7.5.

14.4.2 Request message

14.4.2.1 Request message definition

Table 403 defines the request message.

Table 403 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	TransferData Request SID	M	0x36	TD
#2	blockSequenceCounter	M	0x00 – 0xFF	BSC
#3 : #n	transferRequestParameterRecord[] = [transferRequestParameter#1 : transferRequestParameter#m]	C : U	0x00 – 0xFF : 0x00 – 0xFF	TRPR_ TRTP_ : TRTP_
C = Conditional: this parameter is mandatory if a download is in progress.				

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

This service does not use a sub-function parameter.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

14.4.2.3 Request message data-parameter definition

Table 404 defines the data-parameters of the request message.

Table 404 — Request message data-parameter definition

Definition
blockSequenceCounter The blockSequenceCounter parameter value starts at 0x01 with the first TransferData request that follows the RequestDownload (0x34) or RequestUpload (0x35) service. Its value is incremented by 1 for each subsequent TransferData request. At the value of 0xFF the blockSequenceCounter rolls over and starts at 0x00 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.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

14.4.3 Positive response message

14.4.3.1 Positive response message definition

Table 405 defines the positive response message.

Table 405 — Positive response message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	TransferData Response SID	M	0x76	TDPR
#2	blockSequenceCounter	M	0x00 – 0xFF	BSC
#3 : #n	transferResponseParameterRecord[] = [transferResponseParameter#1 : transferResponseParameter#m]	C : U	0x00 – 0xFF : 0x00 – 0xFF	TREPR_ TREP_ : TREP
C = Conditional: this parameter is mandatory if an upload is in progress.				

14.4.3.2 Positive response message data-parameter definition

Table 406 defines the data-parameters of the positive reponse message.

Table 406 — 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.

14.4.4 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 407. The listed negative responses shall be used if the error scenario applies to the server.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 407 — Supported negative response codes

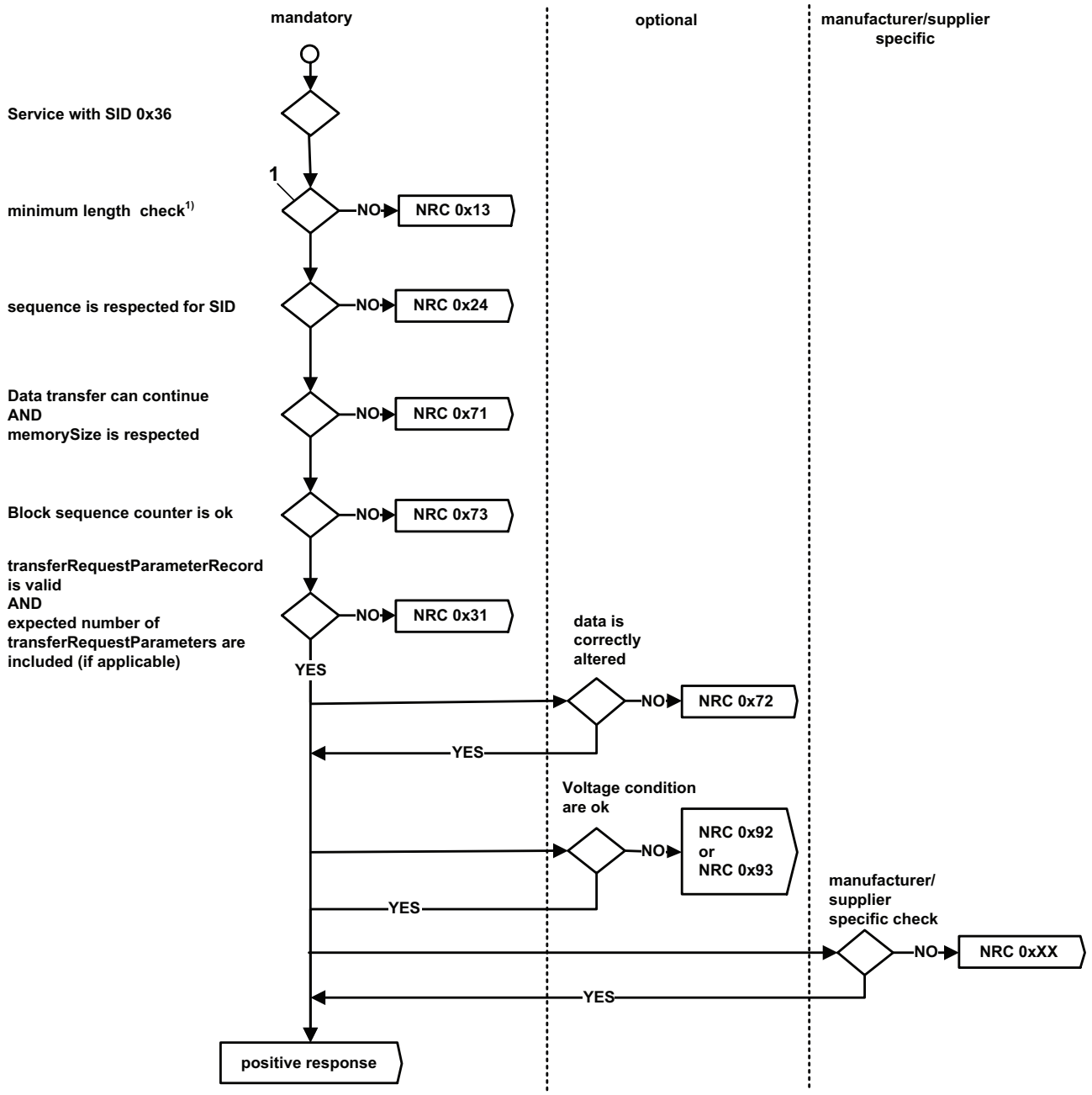
NRC	Description	Mnemonic
0x13	**incorrectMessageLengthOrInvalidFormat** This NRC shall be sent if the length of the message is wrong.(e.g., message length does not meet requirements of maxNumberOfBlockLength parameter returned in the positive response to the requestDownload service).	IMLOIF
0x24	**requestSequenceError** The server shall use this response code: — If the RequestDownload or RequestUpload service is not active when a request for this service is received; — If the RequestDownload or RequestUpload service is active, but the server has already received all data as determined by the memorySize parameter in the active RequestDownlod or RequestUpload service; NOTE 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
0x31	**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 or requestUpload service parameter maxNumberOfBlockLength. — The transferRequestParameterRecord is not consistent with the server's memory alignment constraints.	ROOR
0x71	**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
0x72	**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
0x73	**wrongBlockSequenceCounter** This NRC shall be returned if the server detects an error in the sequence of the blockSequenceCounter. NOTE 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
0x92 / 0x93	**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

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

The evaluation sequence is documented in Figure 28.



Key

1 must be 2 if a RequestUpload is in progress (SI + BSC),
at least 3 if a RequestDownload is in progress (SI + BSC + minimum TRPR_)

Figure 28 — NRC handling for TransferData service

14.4.5 Message flow example(s) TransferData

See 14.5.5 for a complete message flow example.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

14.5 RequestTransferExit (0x37) service

14.5.1 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 7.5.

14.5.2 Request message

14.5.2.1 Request message definition

Table 408 defines the request message.

Table 408 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	RequestTransferExit Request SID	M	0x37	RTE
#2 : #n	transferRequestParameterRecord[] = [transferRequestParameter#1 : transferRequestParameter#m]	U : U	0x00 – 0xFF : 0x00 – 0xFF	TRPR_ TRTP_ : TRTP_

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

This service does not use a sub-function parameter.

14.5.2.3 Request message data-parameter definition

Table 409 defines the data-parameters of the request message.

Table 409 — 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.

14.5.3 Positive response message

14.5.3.1 Positive response message definition

Table 410 defines the positive response message.

Table 410 — Positive response message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	RequestTransferExit Response SID	M	0x77	RTEPR
#2 : #n	transferResponseParameterRecord[] = [transferResponseParameter#1 : transferResponseParameter#m]	U : U	0x00 – 0xFF : 0x00 – 0xFF	TREPR_ TREP_ : TREP

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

14.5.3.2 Positive response message data-parameter definition

Table 411 defines the data-parameters of the positive response message.

Table 411 — 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.

14.5.4 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 412. The listed negative responses shall be used if the error scenario applies to the server.

Table 412 — Supported negative response codes

NRC	Description	Mnemonic
0x13	**incorrectMessageLengthOrInvalidFormat** This NRC shall be returned if the length of the message is wrong.	IMLOIF
0x24	**requestSequenceError** This NRC shall be returned if: — The programming process is not completed when a request for this service is received; — The RequestDownload or RequestUpload service is not active;	RSE
0x31	**requestOutOfRange** This NRC shall be returned if the transferRequestParameterRecord contains invalid data.	ROOR
0x72	**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 29.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

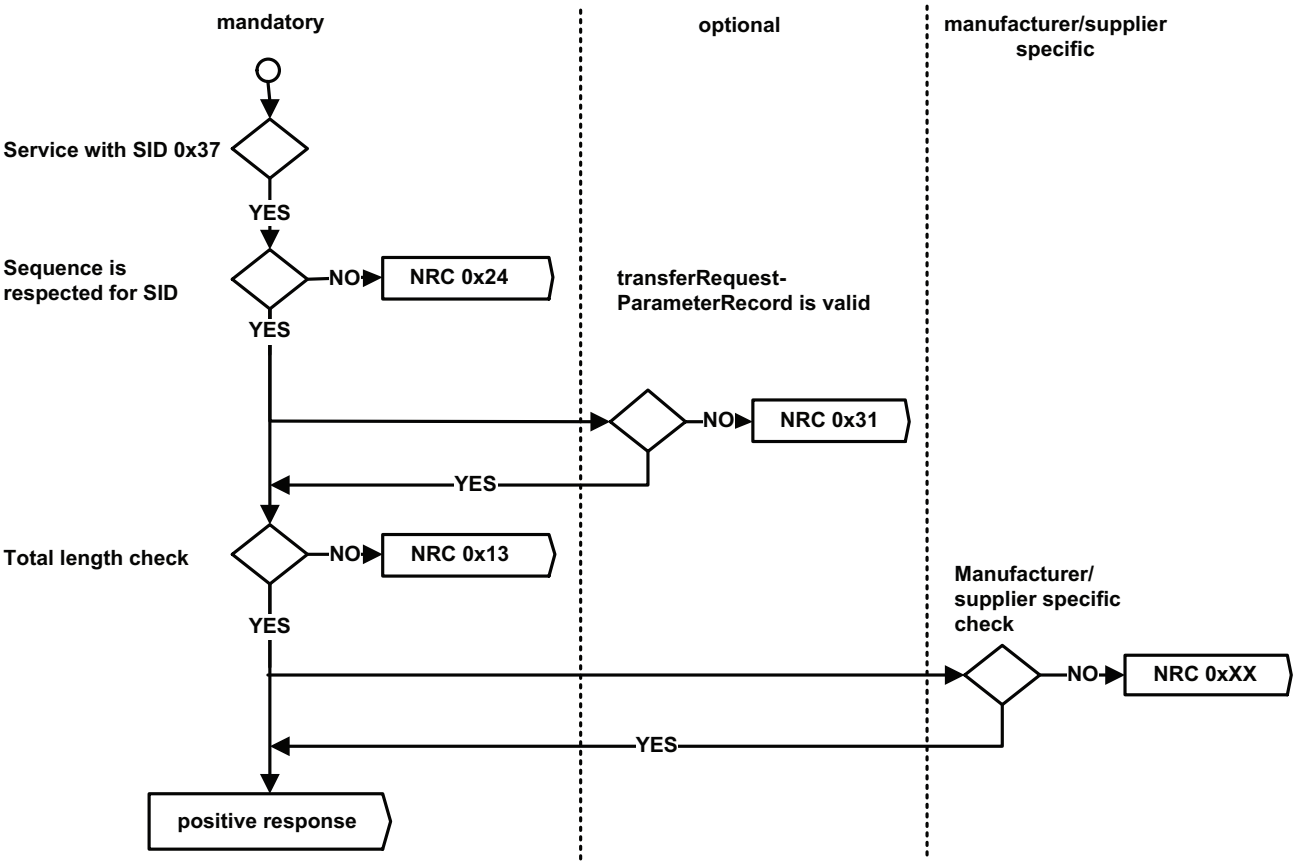


Figure 29 — NRC handling for RequestTransferExit service

14.5.5 Message flow example(s) for downloading/uploading data

14.5.5.1 Download data to a server

14.5.5.1.1 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 413 defines the transferRequestParameter values.

Table 413 — Definition of transferRequestParameter values

Data Parameter Name	Data Parameter Value(s)	Data Parameter Description
memoryAddress (3 bytes)	0x602000	memoryAddress (start) to download data to
dataFormatIdentifier	0x11	dataFormatIdentifier: — compressionMethod = 0x1X — encryptingMethod = 0xX1

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 413 — *(continued)*

Data Parameter Name	Data Parameter Value(s)	Data Parameter Description
MemorySize (3 bytes)	0x00FFFF	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 414 defines the transferResponseParameter value.

Table 414 — Definition of transferResponseParameter value

Data Parameter Name	Data Parameter Value(s)	Data Parameter Description
maximumNumberOfBlockLength	0x0081	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 0x602000 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 0x05.

14.5.5.1.2 Step #1: Request for download

Table 415 defines the RequestDownload request message flow example.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 415 — 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	0x34	RD
#2	dataFormatIdentifier	0x11	DFI
#3	addressAndLengthFormatIdentifier	0x33	ALFID
#4	memoryAddress [byte#1] (MSB)	0x60	MA_B1
#5	memoryAddress [byte#2]	0x20	MA_B2
#6	memoryAddress [byte#3] (LSB)	0x00	MA_B3
#7	MemorySize [byte#1] (MSB)	0x00	UCMS_B1
#8	MemorySize [byte#2]	0xFF	UCMS_B2
#9	MemorySize [byte#3] (LSB)	0xFF	UCMS_B3

Table 416 — 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	0x74	RDPR
#2	LengthFormatIdentifier	0x20	LFID
#3	maxNumberOfBlockLength [byte#1] (MSB)	0x00	MNROB_B1
#4	maxNumberOfBlockLength [byte#2] (LSB)	0x81	MNROB_B1

14.5.5.1.3 Step #2: Transfer data

Table 417 — 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	0x36	TD
#2	blockSequenceCounter	0x01	BSC
#3	transferRequestParameterRecord [transferRequestParameter#1] = dataByte#3	0xXX	TRTP_1
:	:	:	:
#129	transferRequestParameterRecord [transferRequestParameter#127] = dataByte#129	0xXX	TRTP_127

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 418 — 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	0x76	TDPR
#2	blockSequenceCounter	0x01	BSC

⋮

Table 419 — 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	0x36	TD
#2	blockSequenceCounter	0x05	BSC
#3 : #n+2	transferRequestParameterRecord [transferRequestParameter#1] = dataByte#3 : transferRequestParameterRecord [transferRequestParameter#n-2] = dataByte#n	0xXX : 0xXX	TRTP_1 : TRTP_n-2

Table 420 — 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	0x76	TDPR
#2	blockSequenceCounter	0x05	BSC

14.5.5.1.4 Step #3: Request Transfer exit

Table 421 — 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	0x37	RTE

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 422 — 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	0x77	RTEPR

14.5.5.2 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 is exchanged as parameters in the request and positive response message between client and the server:

Table 423 — Definition of transferRequestParameter values

Data parameter name	**Data value(s)**	**Data parameter description**
memoryAddress (3 bytes)	0x201000	memoryAddress (start) to upload data from
dataFormatIdentifier	0x11	dataFormatIdentifier — compressionMethod = 0x1X — encryptingMethod = 0xX1
MemorySize (3 bytes)	0x0001FF	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 424 — Definition of transferResponseParameter value

Data parameter name	**Data value(s)**	**Data parameter description**
maximumNumberOfBlockLength	0x0081	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 0x201000 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 (0x36) service.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

14.5.5.2.1 Step #1: Request for upload

Table 425 — 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	0x35	RU
#2	dataFormatIdentifier	0x11	DFI
#3	addressAndLengthFormatIdentifier	0x33	ALFID
#4	memoryAddress [byte#1] (MSB)	0x20	MA_B1
#5	memoryAddress [byte#2]	0x10	MA_B2
#6	memoryAddress [byte#3] (LSB)	0x00	MA_B3
#7	MemorySize [byte#1] (MSB)	0x00	UCMS_B1
#8	MemorySize [byte#2]	0x01	UCMS_B2
#9	MemorySize [byte#3] (LSB)	0xFF	UCMS_B3

Table 426 — 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	0x75	RUPR
#2	lengthFormatIdentifier	0x20	LFID
#3	maxNumberOfBlockLength [byte#1] (MSB)	0x00	MNROB_B1
#4	maxNumberOfBlockLength [byte#2] (LSB)	0x81	MNROB_B1

14.5.5.2.2 Step #2: Transfer data

Table 427 — 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	0x36	TD
#2	blockSequenceCounter	0x01	BSC

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 428 — 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	0x76	TDPR
#2	blockSequenceCounter	0x01	BSC
#3 : #129	transferResponseParameterRecord [transferResponseParameter#1] = dataByte3 : transferResponseParameterRecord [transferResponseParameter#127] = dataByte129	xx : xx	TREP_1 : TREP_127

⋮

Table 429 — 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	0x36	TD
#2	blockSequenceCounter	0x05	BSC

Table 430 — 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	0x76	TDPR
#2	blockSequenceCounter	0x05	BSC
#3 : #5	transferResponseParameterRecord [transferResponseParameter#1] = dataByte3 : transferResponseParameterRecord [transferResponseParameter#3] = dataByte5	0xXX : 0xXX	TREP_1 : TREP_3

14.5.5.2.3 Step #3: Request Transfer exit

Table 431 — 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	0x37	RTE

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 432 — 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	0x77	RTEPR

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

14.6 RequestFileTransfer (0x38) service

14.6.1 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 7.5.

14.6.2 Request message

14.6.2.1 Request message definition

Table 433 defines the request message.

Table 433 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	RequestFileTransfer Request SID	M	0x38	RFT
#2	modeOfOperation	M	0x01 – 0x05	MOOP
	filePathAndNameLength [			
#3	byte#1 (MSB)	M	0x00 – 0xFF	FPL_B1
#4	byte#2] (LSB)	M	0x00 – 0xFF	FPL_B2
	filePathAndName = [			
#5	byte#1 (MSB)	M	0x00 – 0xFF	FP_B1
:	:	:	:	:
#5+n-1	byte#n]	C_1	0x00 – 0xFF	FP_Bn
#5+n	dataFormatIdentifier	C_2	0x00 – 0xFF	DFI_
#5+n+1	fileSizeParameterLength	C_2	0x00 – 0xFF	FSL
	fileSizeUnCompressed= [			
#5+n+2	byte#1 (MSB)	C_2	0x00 – 0xFF	FSUC_B1
:	:	:	:	:
#5+n+2+k-1	byte#k]	$C_{2,3}$	0x00 – 0xFF	FSUC_Bk
	fileSizeCompressed= [			
#5+n+2+k	byte#1 (MSB)	C_2	0x00 – 0xFF	FSC_B1
:	:	:	:	:
#5+n+1+2k	byte#k]	$C_{2,3}$	0x00 – 0xFF	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.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

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

This service does not use a sub-function parameter.

14.6.2.3 Request message data-parameter definition

Table 434 defines the data-parameters of the request message.

Table 434 — Request message data-parameter definition

Definition
modeOfOperation This data-parameter defines the type of operation to be applied to the file or directory indicated in the filePathAndName parameter. The values of the data-parameter are defined in Annex G.
filePathAndNameLength Defines the length in byte for the parameter filePath.
filePathAndName Defines 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 0x05 (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 0x00 specifies that neither compressionMethod nor encryptingMethod is used. Values other than 0x00 are vehicle manufacturer specific. If the modeOfOperation parameter equals to 0x02 (DeleteFile) and 0x05 (ReadDir) this parameter shall not be included in the request message.
fileSizeParameterLength Defines the length in bytes for both parameters fileSizeUncompressed and fileSizeCompressed. If the modeOfOperation parameter equals to 0x02 (DeleteFile), 0x04 (ReadFile) or 0x05 (ReadDir) this parameter shall not be included in the request message.
fileSizeUncompressed Defines the size of the uncompressed file in bytes. If the modeOfOperation parameter equals 0x02 (DeleteFile), 0x04 (ReadFile) or 0x05 (ReadDir) this parameter shall not be included in the request message.
fileSizeCompressed Defines 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 0x02 (DeleteFile), 0x04 (ReadFile) or 0x05 (ReadDir) this parameter shall not be included in the request message.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

14.6.3 Positive response message

14.6.3.1 Positive response message definition

Table 435 defines the positive response message.

Table 435 — Positive response message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	RequestFileTransfer Response SID	S	0x78	RRFT
#2	modeOfOperation	M	0x01 – 0x05	MOOP
#3	lengthFormatIdentifier	C_1	0x00 – 0xFF	LFID
#4 : #4+(m-1)	maxNumberOfBlockLength = [byte#1 (MSB) : byte#m]	 $C_{1,2}$: $C_{1,2}$	 0x00 – 0xFF : 0x00 – 0xFF	MNROB_ B1 : Bm
#4+m	dataFormatIdentifier	C_1	0x00 – 0xFF	DFI_
#4+m+1 #4+m+2	fileSizeOrDirInfoParameterLength [byte#1 (MSB) byte#2 (LSB)]	 C_1 C_1	 0x00 – 0xFF 0x00 – 0xFF	 FSDIL_B1 FSDIL_B2
#4+m+3 : #4+m+3+k-1	fileSizeUncompressedOrDirInfoLength= [byte#1 (MSB) : byte#k]	 $C_{1,3}$: $C_{1,3}$	 0x00 – 0xFF : 0x00 – 0xFF	 FSUDIL_B1 : FSUDIL_Bk
#4+m+3+k : #4+m+3+2k-1	fileSizeCompressed= [byte#1 (MSB) : byte#k]	 $C_{1,3}$: $C_{1,3}$	 0x00 – 0xFF : 0x00 – 0xFF	 FSC_B1 : FSC_Bk

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 fileSizeOrDirInfoParameterLength parameter

C_3: The length (number of bytes) of this message parameter is defined by the lengthFormatIdentifier parameter

14.6.3.2 Positive response message data-parameter definition

Table 436 defines the data-parameters of the positive response message.

Table 436 — Response message data-parameter definition

Definition
modeOfOperation This is parameter echoes the value of the request.
lengthFormatIdentifier Defines the length (number of bytes) of the maxNumberOfBlockLength parameter. If the modeOfOperation parameter equals to 0x02 (DeleteFile) this parameter shall be not be included in the response message.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 436 — *(continued)*

Definition
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 may 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 0x02 (DeleteFile) this parameter shall be not be included in the response message.
dataFormatIdentifier This is parameter echoes the value of the request. If the modeOfOperation parameter equals to 0x02 (DeleteFile) this parameter shall not be included in the response message.) If the modeOfOperation parameter equals to 0x05 (ReadDir) the value of this parameter shall be equal to 0x00.
fileSizeOrDirInfoParameterLength Defines the length in bytes for both parameters fileSizeUncompressedOrDirInfoLength and fileSizeCompressed. If the modeOfOperation parameter equals to 0x01 (AddFile), 0x02 (DeleteFile) or 0x03 (ReplaceFile) this parameter shall not be included in the response message.
fileSizeUncompressedOrDirInfoLength Defines 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 0x01 (AddFile), 0x02 (DeleteFile) or 0x03 (ReplaceFile) this parameter shall not be included in the response message.
fileSizeCompressed Defines the size of the compressed file in bytes. If the modeOfOperation parameter equals to 0x01 (AddFile), 0x02 (DeleteFile, 0x03 (ReplaceFile)) or 0x05 (ReadDir) this parameter shall not be included in the response message.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

14.6.4 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 437. The listed negative responses shall be used if the error scenario applies to the server.

Table 437 — Supported negative response codes

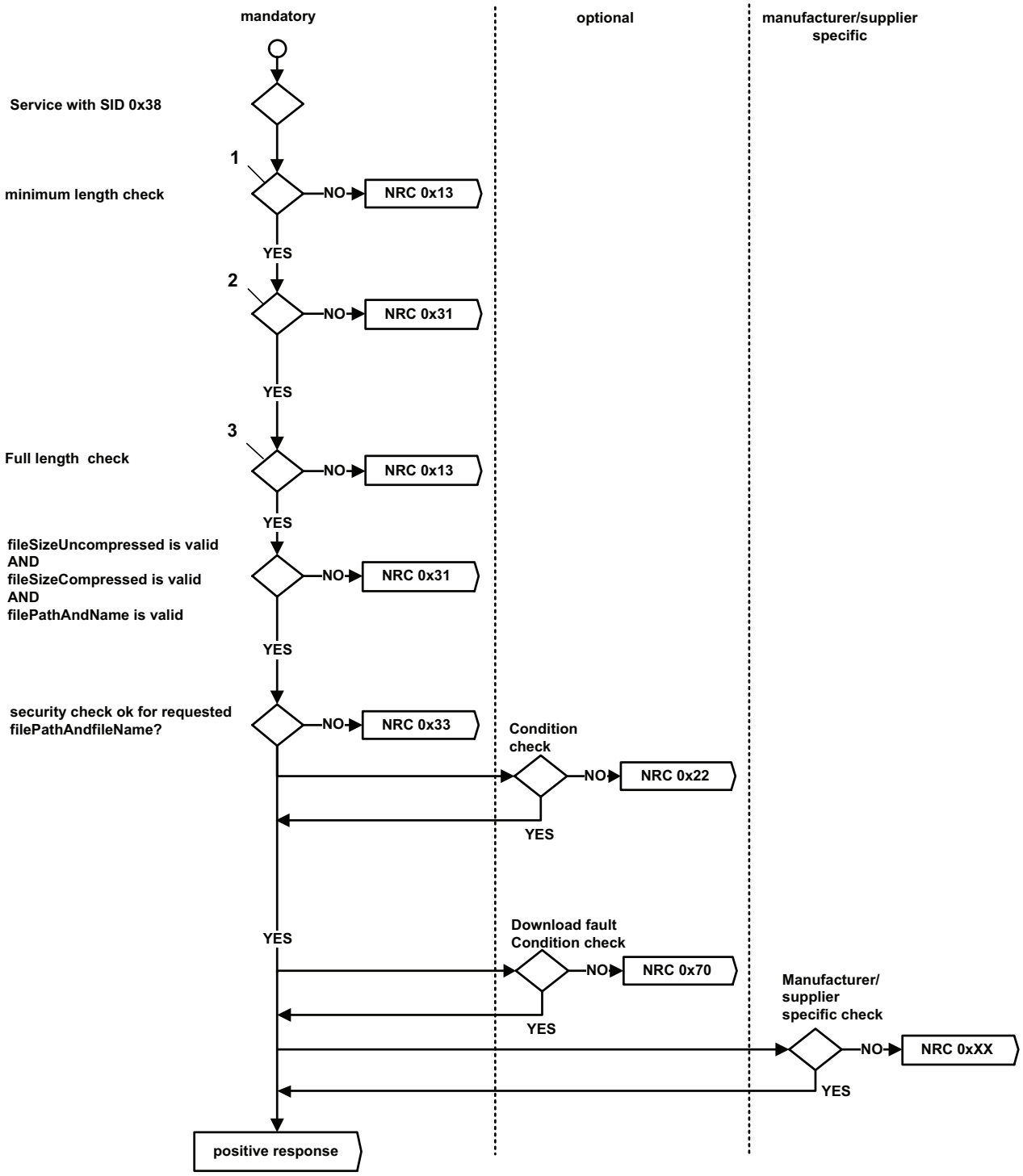
NRC	Description	Mnemonic
0x13	**incorrectMessageLengthOrInvalidFormat** This NRC shall be sent if the length of the message is wrong.	IMLOIF
0x22	**conditionsNotCorrect** This NRC shall be returned if a server receives a request for this service while in the process of downloading or uploading data or other conditions to be able to execute this service are not met.	CNC
0x31	**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
0x33	**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
0x70	**uploadDownloadNotAccepted** This NRC indicates that an attempt to download to a server's memory cannot be accomplished due to some fault conditions.	UDNA

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

The evaluation sequence is documented in Figure 30.



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 30 — Response evaluation sequence requestFileTransfer

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

14.6.5 Message flow example(s) RequestFileTransfer

14.6.5.1 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 438 defines the message parameter values.

Table 438 — Definition RequestFileTransfer message parameter values

Data Parameter Name	Data Parameter Value(s)	Data Parameter Description
modeOfOperation	0x01	AddFile
filePathAndNameLength	0x001E	The length of parameter filePathAndName is 30.
filePathAndName	"D:\mapdata\europe\germany1.yxz"	Path including the file name.
dataFormatIdentifier	0x11	compressionMethod = 0x1X; encryptingMethod = 0xX1
fileSizeParameterLength	0x02	The length of both file size parameters is 2 bytes.
fileSizeUncompressed	0xC350	50 KByte
fileSizeCompressed	0x7530	30 KByte

14.6.5.2 Request file transfer

Table 439 and Table 440 show an example of the RequestFileTransfer request and response message flow.

Table 439 — 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	0x38	RFT
#2	modeOfOperation	0x01	MOOP
#3 #4	filePathAndNameLength [byte#1 (MSB) byte#2] (LSB)	0x00 0x1E	FPL_B1 FPL_B2

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 439 — *(continued)*

Message direction	**server → client**		
Message Type	**Response**		
A_Data Byte	**Description (all values are in hexadecimal)**	**Byte Value**	**Mnemonic**
	filePathAndName = [		
#5	byte#1 (MSB)	0x44	FP_B1
#6	byte#2	0x3A	FP_B2
#7	byte#3	0x5C	FP_B3
#8	byte#4	0x6D	FP_B4
#9	byte#5	0x61	FP_B5
#10	byte#6	0x70	FP_B6
#11	byte#7	0x64	FP_B7
#12	byte#8	0x61	FP_B8
#13	byte#9	0x74	FP_B9
#14	byte#10	0x61	FP_B10
#15	byte#11	0x5C	FP_B11
#16	byte#12	0x65	FP_B12
#17	byte#13	0x75	FP_B13
#18	byte#14	0x72	FP_B14
#19	byte#15	0x6F	FP_B15
#20	byte#16	0x70	FP_B16
#21	byte#17	0x65	FP_B17
#22	byte#18	0x5C	FP_B18
#23	byte#19	0x67	FP_B19
#24	byte#20	0x65	FP_B20
#25	byte#21	0x72	FP_B21
#26	byte#22	0x6D	FP_B22
#27	byte#23	0x61	FP_B23
#28	byte#24	0x6E	FP_B24
#29	byte#25	0x79	FP_B25
#30	byte#26	0x31	FP_B26
#31	byte#27	0x2E	FP_B27
#32	byte#28	0x79	FP_B28
#33	byte#29	0x78	FP_B29
#34	byte#30]	0x7A	FP_B30
#35	dataFormatIdentifier	0x11	DFI_
#36	fileSizeParameterLength	0x02	FSL
	fileSizeUnCompressed= [		
#37	byte#1 (MSB)	0xC3	FSUC_B1
#38	byte#2]	0x50	FSUC_Bk
	fileSizeCompressed= [		
#39	byte#1 (MSB)	0x75	FSC_B1
#40	byte#2]	0x30	FSC_Bk

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 440 — 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	0x78	RRFT
#2	modeOfOperation	0x01	MOOP
#3	lengthFormatIdentifier	0x02	LFID
#4 #5	maxNumberOfBlockLength = [byte#1 (MSB) byte#m]	0xC3 0x50	MNROB_ B1 B2
#6	dataFormatIdentifier	0x11	DFI_

15 Non-volatile server memory programming process

15.1 General information

This clause 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 part of ISO 14229 and Part 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:

— 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 ISO 14229-1 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).

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

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 31 depicts the non-volatile server memory programming process overview.

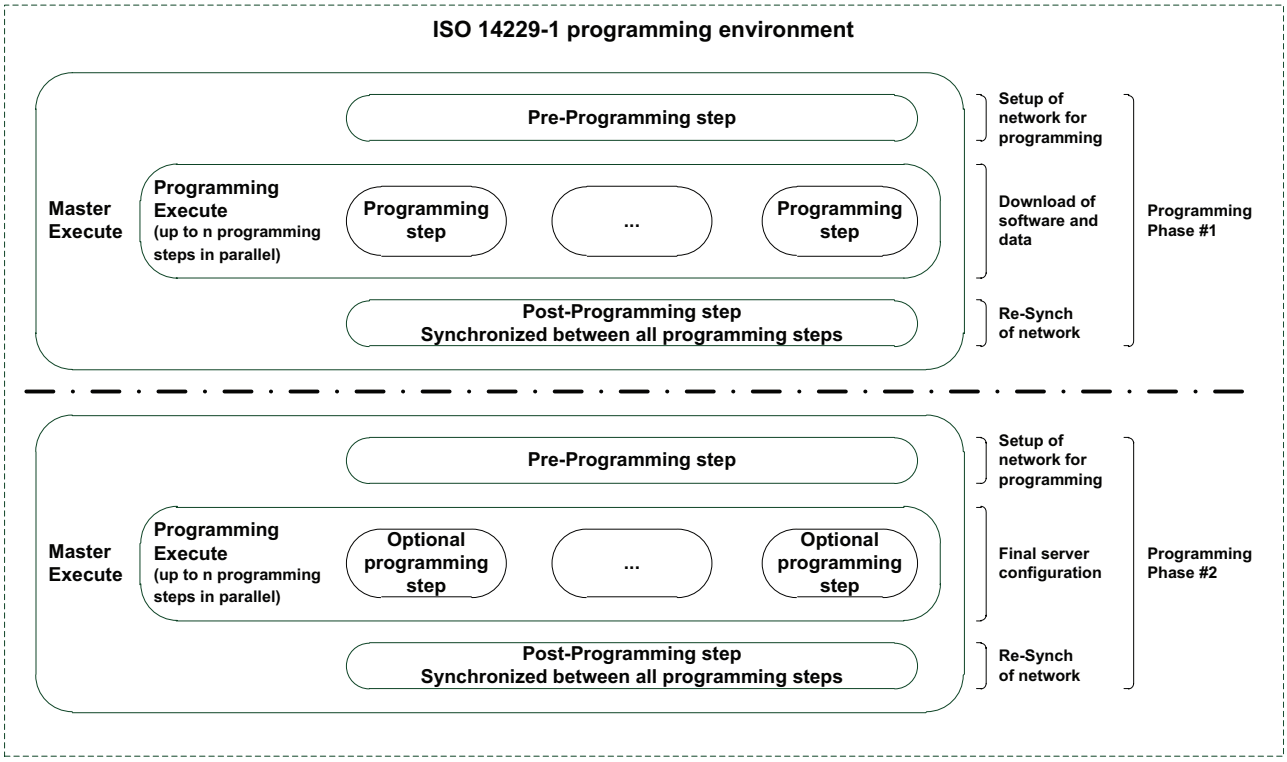


Figure 31 — Non-volatile server memory programming process overview

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 have to 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 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 synchronization 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 has to 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" don’t need to be coordinated 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).

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

a) **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 pre-programming 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 baud 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 pre-programming step is not used, then the DiagnosticSessionControl (0x10) with subfunction programmingSession can also be performed using functional addressing.

At the end of this step, a physical reset of the re-programmed 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 re-programmed server(s), because after the physical reset during this step the re-programmed 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 re-programmed server(s) will potentially set DTCs.

Furthermore, it shall be considered that the re-programmed server could activate a new set of diagnostic address, which differs from the ones used when performing a programming event (see 15.3).

If either the server that was re-programmed 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 re-programmed server.

3) Post-Programming step — Re-synchronization 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 (0x11) service or an explicit transition to the default session via the DiagnosticSessionControl (0x10) service.

b) **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 re-programmed is (are) physically reset

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

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 pre-programming 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 re-programmed are 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.

— Post-Programming step — Re-synchronization 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.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

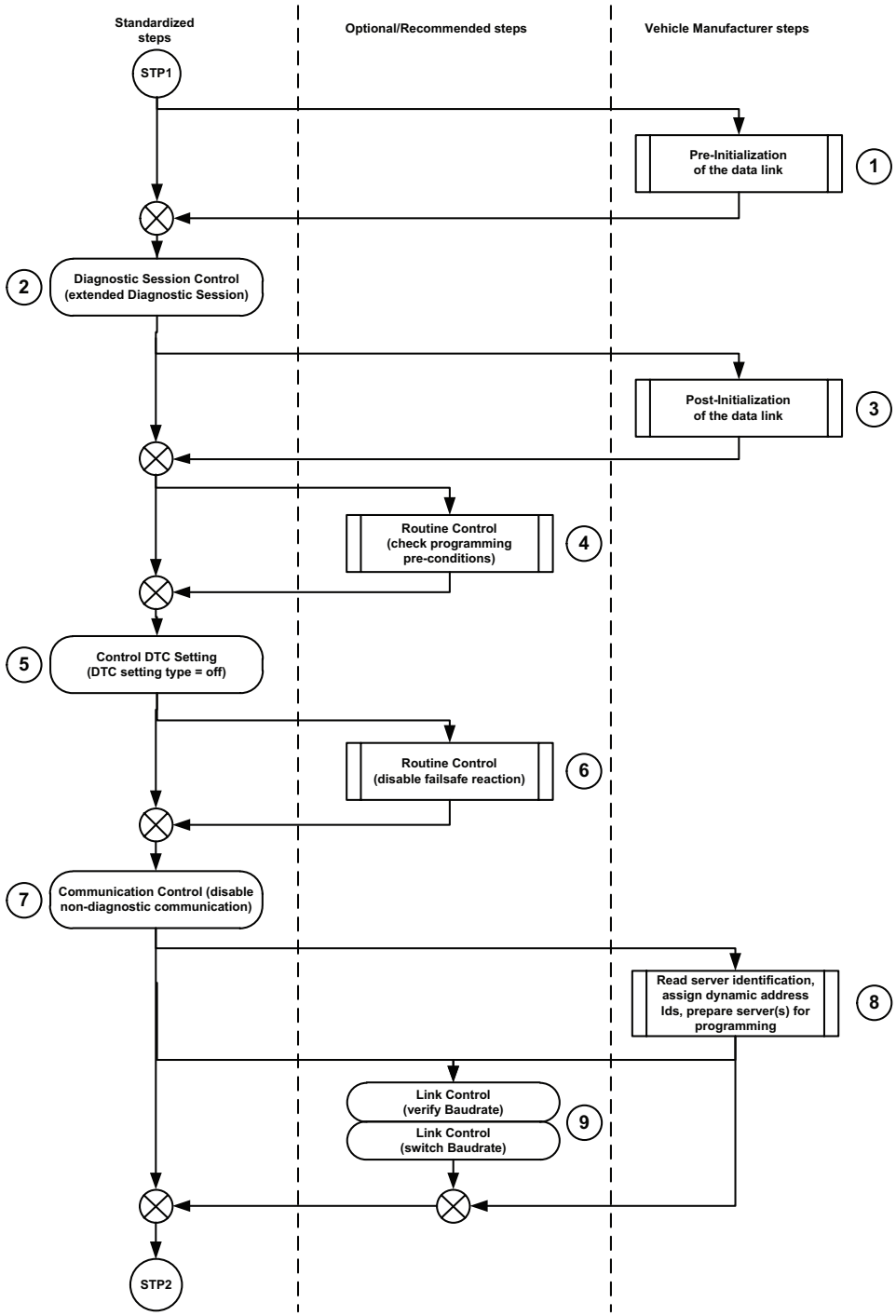
© ISO 2013 – All rights reserved

15.2 Detailed programming sequence

15.2.1 Programming phase #1 — Download of application software and/or application data

15.2.1.1 Pre-Programming step of phase #1 — Setup of vehicle network for programming

Figure 32 graphically depicts the functionality embedded in the pre-programming step.



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).

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

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 (0x10) service with sessionType equal to extendedDiagnosticSession. The request is either transmitted functionally addressed to all servers with a single request message, or physically addressed to each server in a separate request message (requires a physically addressed TesterPresent (0x3E) 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 (0x85) 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 (0x28) 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 (0x87) 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 (0x87) service with subfunction 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 have to 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 32 — Pre-programming step of phase 1 (STP1)

SHW3SZH 2013-09-29 NormMaster

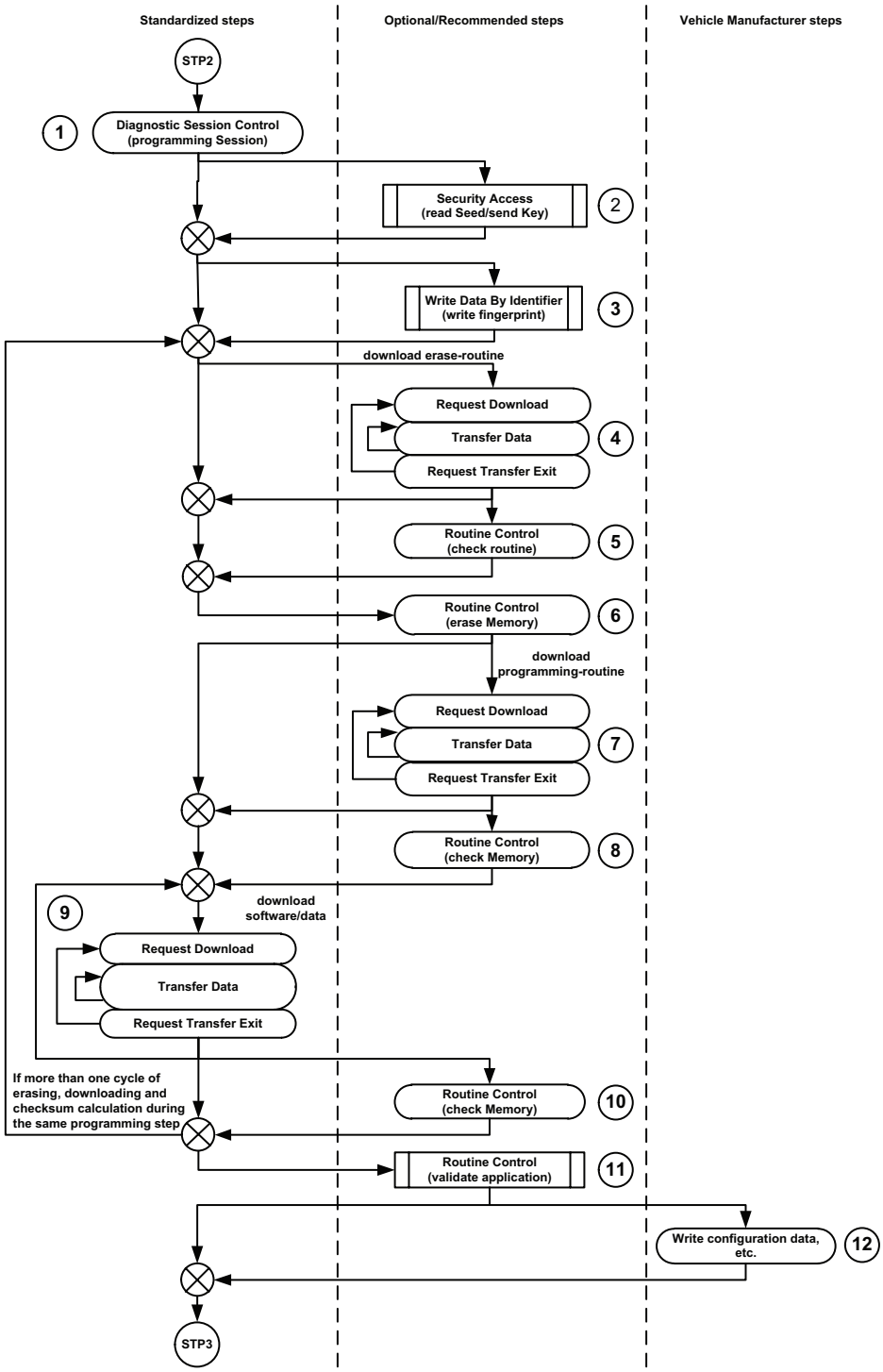
Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

 © ISO 2013 – All rights reserved

15.2.1.2 Programming step of phase #1 — Download of application software and data

Following the pre-programming step, the programming of one or multiple servers is performed. The programming sequence applies for a programming event of a single server and is therefore physically oriented. When multiple servers are programmed, then multiple programming events either run in parallel or will be performed sequentially.

Figure 33 graphically depicts the functionality embedded in the programming step of phase #1.



Key

1 The programming event is started in the server(s) via a physically/functionally addressed request of the DiagnosticSessionControl (0x10) service with sessionType equal to programmingSession. When the server(s)

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

receive(s) the request, it/they shall allocate all necessary resources required for programming. It is implementation specific whether the server(s) start(s) executing out of boot software.

2 A programming event should be secured. The SecurityAccess (0x27) service shall be mandatory for emissions-related and safety systems. Other systems are not required to implement this service. The method on how a security access is performed is specified in this part of ISO 14229.

3 It is vehicle manufacturer specific to write a "fingerprint" into the server memory prior to the download of any data (e.g., application software) into the ECU. The "fingerprint" identifies the one who modifies the server memory. When using this option then the dataIdentifiers bootSoftwareFingerprint, applicationSoftwareFingerprint and applicationDataFingerprint shall be used to write the fingerprint information (see dataIdentifier definitions).

4 Where the server does not have the memory erase routine stored in permanent memory, then a download of the memory erase routine shall be performed. The download shall follow the specified sequence with RequestDownload (…), TransferData, and RequestTransferExit.

5 It is vehicle manufacturer specific if a RoutineControl (0x31) is used to check whether the download of the memory erase routine was successful. Alternative methods are to provide the result in the RequestTransferExit positive response message or via a negative response message including the appropriate negative response code to the RequestTransferExit request message.

6 The memory of the server shall be erased when required by the memory technology (e.g., flash memory) in order to allow an application software/data download. This is achieved via a routineIdentifier, using the RoutineControl (0x31) service to execute the erase routine.

7 Where the server does not have the memory programming routine stored in permanent memory, then a download of the memory programming routine shall be performed. The download shall follow the specified sequence with RequestDownload (0x34), TransferData (0x36), and RequestTransferExit (0x37). Note that the memory programming algorithm may be downloaded along with the memory erase algorithm (see footnote d).

8 It is vehicle manufacturer specific if a RoutineControl (0x31) is used to check whether the download of the memory program routine was successful. Alternative methods are to provide the result in the RequestTransferExit positive response message or via a negative response message including the appropriate negative response code to the RequestTransferExit request message.

9 Each download of a contiguous block of application software/data to a non-volatile server memory location (either a complete application software/data module or part of a software/data module) shall always follow the general data transfer method using the following service sequence:

— RequestDownload (0x34);

— TransferData (0x36);

— RequestTransferExit (0x37).

A single application software/data block might require multiple TransferData (0x36) request messages to be completely transmitted (this is the case if the length of the block exceeds the maximum network layer buffer size).

10 It is vehicle manufacturer specific if a RoutineControl (0x31) is used to check whether the download of the memory was successful. Alternative methods are to provide the result in the RequestTransferExit positive response message or via a negative response message including the appropriate negative response code to the RequestTransferExit request message.

11 This optional routineIdentifier (number chosen by the vehicle manufacturer) allows a client to verify if the download has been performed successfully once all application software/data blocks/modules are completely downloaded. This routine typically triggers the server to check any and all reprogramming dependencies and to perform all necessary action to prove that the download and programming into non-volatile memory was successful and valid (e.g., checksum, signature, DTCs, hardware/software compatibility, etc.). The details are left to the discretion of the vehicle manufacturer.

Following the download of the application software/data, it is optional to reset the re-programmed server in order to enable the downloaded application software/data. It shall be considered that the re-programmed server could activate a new set of diagnostic identifiers, which differs to the ones used when performing the programming event. If either the server that was re-programmed does not change its communication parameters or the programming environment know the changed communication parameters, then following the reset certain configuration data can be written to the re- programmed server.

12 Following the download of the application software/data, it is vehicle manufacturer specific to perform further operations such as writing configuration data (e.g. VIN, etc.) back to the server. This also depends on the functionally that is supported by the re-programmed server when running out of boot software.

Figure 33 — Programming step of phase 1 (STP2)

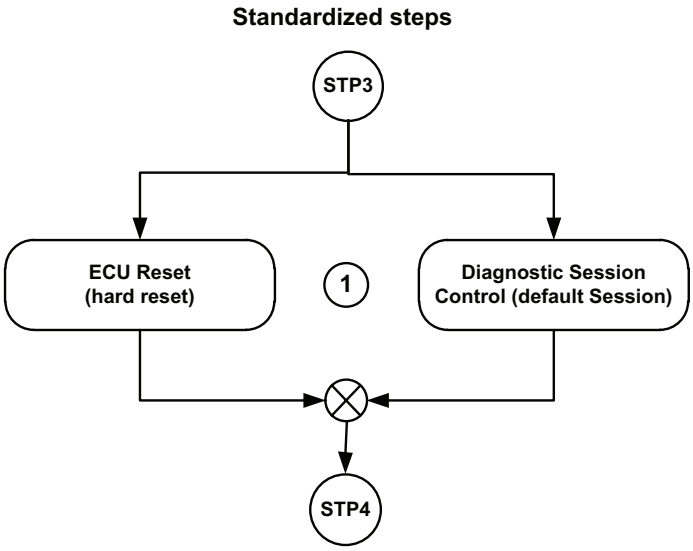
SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

 © ISO 2013 – All rights reserved

15.2.1.3 Post-Programming step of phase #1 — Re-synchronization of vehicle network

Figure 34 graphically depicts the functionality embedded in the post-programming step of phase #1.



Key

1 The client transmits either an ECUReset (0x11) service request message onto the vehicle network with resetType equal to hardReset or DiagnosticSessionControl (0x10) with sessionType equal to defaultSession. This can either be done functionally addressed or physically addressed (depends on the supported vehicle approach). Further it is vehicle manufacturer specific whether a response message is required or not.

When a baud rate switch has been performed, then this step shall be performed functionally, not requiring a response message, because the servers perform a baud rate transition to their normal speed of operation.

The reception of the ECUReset (0x11) request message causes the server(s) to perform a reset and to start the defaultSession.

Figure 34 — Post-programming step of phase 1 (STP3)

15.2.1.4 Pre-programming step of phase #2 — Server configuration

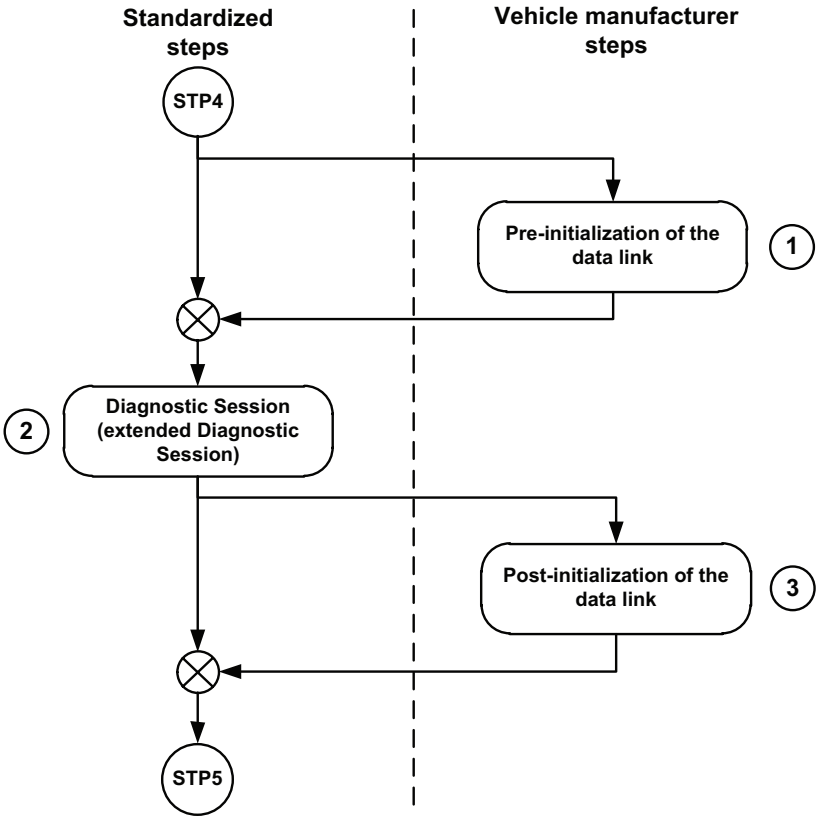
The pre-programming step of phase #2 is optional and should be used when there is the need to perform certain action after the software reset of the reprogrammed server. This will be the case when the server does not provide the required functionality to finally conclude the programming event when running out of boot software during the programming step of phase #1.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Figure 35 graphically depicts the functionality embedded in the pre-programming step of phase #2.



Key

1 Prior to any communication on the data link the network shall be initialized, which means that an initial wake-up of the vehicle network shall be performed. 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 server diagnostic address and the data link identifiers used by the server(s).

2 In order to be able to perform certain services in the programming step of phase #2, a non-defaultSession shall be started in each server on the data link that is involved in the conclusion of the programming event. This is performed via a DiagnosticSessionControl (0x10) service with sessionType equal to extendedDiagnosticSession.

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.

Figure 35 — Pre-programming step of phase 2 (STP4)

15.2.1.5 Programming step of phase #2 — Final server configuration

The programming step of phase #2 is optional and contains any action that needs to take place with the reprogrammed server after the reset (when the application software is running) such as writing specific identification information. This step might be required in case the server does not provide the required functionality to perform an action when running out of boot software during the programming step of phase #1.

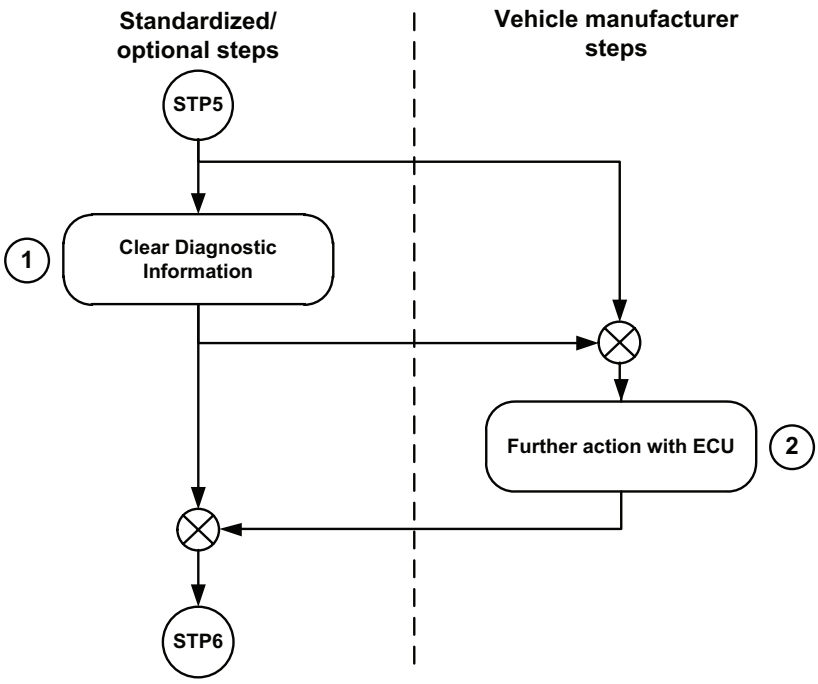
SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

 © ISO 2013 – All rights reserved

When multiple servers require performing additional functions, then multiple programming steps can run in parallel or will be performed sequentially.

Figure 36 depicts the programming step of phase 2 (STP5).



Key

1 In case the re-programmed server(s) has (have) been reset during the programming step of programming phase #1, then any diagnostic information that might have been stored in the re-programmed server(s) may be cleared via a physically addressed ClearDiagnosticInformation (0x14) service.

2 The client performs any operation that is required in order to conclude the programming event with the server, such as writing configuration data (e.g. VIN).

Figure 36 — Programming step of phase 2 (STP5)

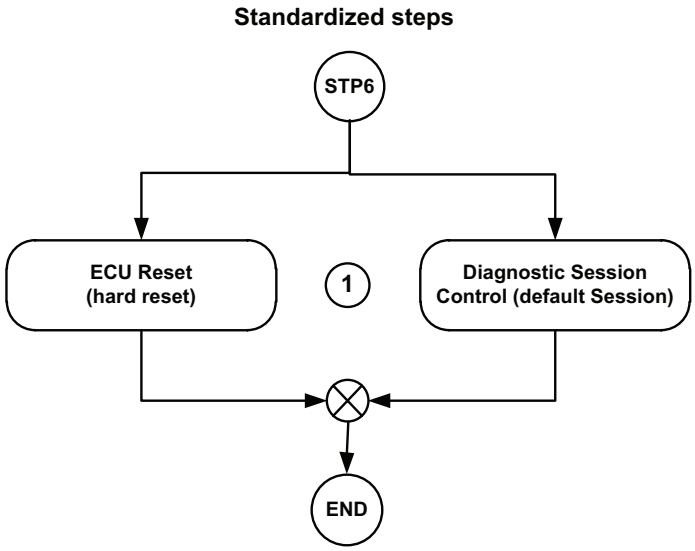
SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

15.2.1.6 Post-programming step of phase #2 — Re-synchronization of vehicle network

Figure 37 depicts the Post-programming step of phase 2 (STP6).



Key

1 The client transmits either an ECUReset (0x11) service request message onto the vehicle network with resetType equal to hardReset or DiagnosticSessionControl (0x10) with sessionType equal to defaultSession. This can either be done functionally addressed or physically addressed (depends on the supported vehicle approach). Further it is vehicle manufacturer-specific whether a response message is required or not.
When a baud rate switch has been performed, then this step shall be performed functionally, not requiring a response message, because the servers perform a baud rate transition to their normal speed of operation.
The reception of the ECUReset (0x11) request message causes the server(s) to perform a reset and to start the defaultSession.

Figure 37 — Post-programming step of phase 2 (STP6)

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

15.3 Server reprogramming requirements

15.3.1 Requirements for servers to support programming

During a programming session, servers shall default their physical I/O pins (wherever possible and without risk of damage to the server/vehicle and without risk of safety hazards) to a predefined state which minimizes current draw.

15.3.1.1 Boot software description and requirements

15.3.1.1.1 Boot software general requirements

All programmable servers that support programming of the application software shall contain boot software in a boot memory partition. Servers that support boot software typically continue to execute out of the boot software until a complete set of application software and application data is programmed (e.g., it is possible for some servers to begin executing application software despite not having 100% of application data programmed).

The boot memory partition shall be protected against inadvertent erasure such that a failed attempt to modify application data or application software does not prohibit the server's ability to recover and be programmed after the failed attempt. The server shall be able to recover and be reprogrammed if any of the following error conditions occur during the programming process:

a) loss of supplied power connection.

b) loss of the ground connection.

c) disruption of data link communication.

d) over- or under-voltage conditions.

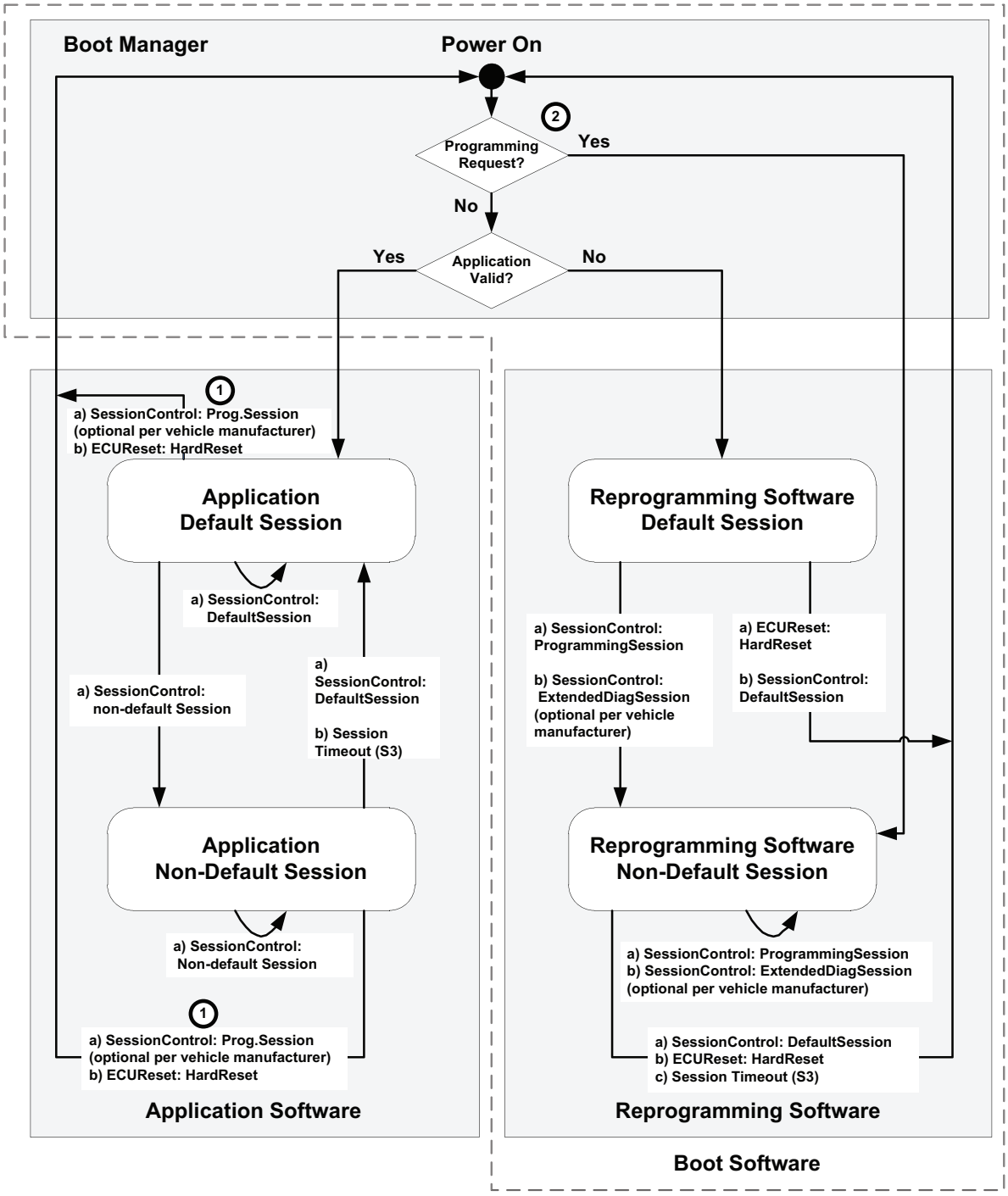
The boot software can be protected via hardware (e.g., via settings in a control register which prevents certain sectors of the memory from being erased or written to) or software (e.g., address range restrictions in the programming routines). It is recommended that the boot software not be capable of being modified by the same programming erase/write routines that are used to modify the application software and application data. Programming the boot software as part of the programming process may be allowed, provided that a mechanism is in place to ensure that there is no possibility that the server could fail at a point of the programming process where it cannot recover and be programmed with a subsequent programming event.

Boot software resides in the boot memory partition and is the software that a server begins executing upon power-up. Transfer of program control to the boot software also occurs once the server is informed that it is about to be programmed (e.g., reference the DiagnosticSessionControl service and the programming process defined in 15.2.1.2). A typical implementation showing the interactions and transitions between the boot software and the application software is shown in Figure 38.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved



Key

1 Some implementations may have the capability to transition to the Reprogramming Software programmingSession without going through a full power on reset.

2 This check can serve two purposes. One is to check whether the application requested a transition into the reprogramming software's programmingSession. The other is an alternative entry into the reprogramming software by other conditions (e.g., by scanning for a SessionControl programmingSession request over a small time window).

Figure 38 — Example of typical interaction and transitions between application and boot software

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

15.3.1.1.2 Boot software diagnostic service requirements

Table 441 to Table 443 define the minimum diagnostic service requirements for the boot software of a programmable server. The listed services have to be supported in order to fulfil the requirements for performing non-volatile server memory programming during programming phase #1. The tables make use of the steps defined for programming phase #1 (see 15.2.1). The service(s) to be supported for steps (1), (3) and (8) shall be defined by the vehicle manufacturer.

Table 441 — Boot software diagnostic service support during pre-programming step of phase #1

Service	Subfunction/Data parameter	Sequence step No.	Remark
DiagnosticSessionControl (0x10)	sessionType = extendedDiagnosticSession (0x03)	(2)	Mandatory: Required for session management ($S3_{Server}$ timeout, especially when performing a baudrate transition and SecurityAccess service).
CommunicationControl (0x28)	controlType = vehicle manufacturer specific (disable non-diagnostic communication messages)	(7)	Mandatory: The server does not need to perform any special action (non-diagnostic messages are disabled when running out of boot), except the transmission of a positive response message.
RoutineControl (0x31)	routineIdentifier = vehicle manufacturer specific	(4), (6)	Optional: Required if check programming pre-conditions or disable failsafe reaction are supported.
ControlDTCSetting (0x85)	DTCSettingType = off (0x02)	(5)	Mandatory: The server does not need to perform any special action (DTCs are disabled when running out of boot), except the transmission of a positive response message.
ReadDataByIdentifier (0x22)	dataIdentifier = vehicle manufacturer specific	(8)	Optional: Required to be supported when reading software/data identification data.
LinkControl (0x87)	linkControlType = verifyWithFixedBaudrate (0x01), verifyWithSpecifcBaudrate (0x02), transitionBaudrate (0x03)	(9)	Optional: Required to be supported when performing a baudrate switch.

NOTE Table 441 only applies if the vehicle manufacturer supports the pre-programming step of phase #1.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 442 — Boot software diagnostic service support during programming step of phase #1

Service	Subfunction/Data parameter	Sequence step No.	Remark
DiagnosticSessionControl (0x10)	sessionType = programmingSession (0x02)	(1)	Mandatory: Required for compatibility with application software in order to allow for the identical handling in the programming application of the client.
SecurityAccess (0x27)	securityAccessType = requestSeed (0x01), sendKey (0x02)	(2)	Optional: Required to be supported by theft-, emission- and safety-related systems.
WriteDataByIdentifier (0x2E)	bootSoftwareFingerprint, appSoftwareFingerprint, appDataFingerprint, vehicle manufacturer specific	(3)	Optional: Required for writing the fingerprint and other identification data.
RequestDownload (0x34)	vehicle manufacturer specific	(4), (7), (9)	Mandatory: In general required for the transfer of data from the client to the server when running out of boot.
TransferData (0x36)	routine data, application software, or application data		
RequestTransferExit (0x37)	vehicle manufacturer specific		
RoutineControl (0x31)	routineControlType = startRoutine (0x01) routineIdentifier = refer to sequence step details for required numbers	(5), (6), (8), (10), (11)	Optional: Required if any of the sequence steps are supported by the vehicle manufacturer.
ECUReset (0x11)	resetType = hardReset (0x01)	(12)	Mandatory: Required for a reset of the re-programmed server at the end of the programming step. The server(s) that have been reprogrammed are forced to perform a reset in order to start the application software.
The service(s) to be supported for step (m) shall be defined by the vehicle manufacturer.			

Table 443 — Boot software diagnostic service support during post-programming step of phase #1

Service	Subfunction / Data parameter	Sequence step	Remark
ECUReset (0x11)	resetType = hardReset (0x01)	(1)	Mandatory: The server(s) that have been reprogrammed are forced to perform a reset in order to start the application software.

15.3.1.2 Security requirements

All programmable servers that have emission, safety or theft related features shall employ a seed and key security feature, accessible via the SecurityAccess (0x27) service, to protect the programmed server from inadvertent erasure and unauthorized programming. All such field service replacement servers shall be shipped to the field with the security feature activated (i.e., a programming tool cannot gain access to the server without first gaining access through the SecurityAccess service).

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

 © ISO 2013 – All rights reserved

15.3.2 Software, data identification and fingerprints

15.3.2.1 Software and data identification

The boot software, application software and application data may be identified via the dataIdentifiers according to C.1. The structure of the dataRecord for bootSoftwareIdentification, applicationSoftwareIdentification and applicationDataIdentification is vehicle manufacturer specific.

The bootSoftwareIdentification, applicationSoftwareIdentification and applicationDataIdentification shall be part of each module that is downloaded into the server; therefore any write operation to the defined dataIdentifiers shall be rejected by the server.

15.3.2.2 Software and data fingerprints

A fingerprint uniquely identifies the programming tool that erased and/or reprogrammed the server software/data. If the server software/data is separated in several modules, the fingerprint could also identify which software/data module is manipulated (e.g., boot software, application software, and application data). If supported a fingerprint shall be written into non-volatile memory of the server before any software/data manipulation occurs (e.g. before erasing the flash memory).

The boot software, application software and application data fingerprints may be identified via the dataIdentifiers according to C.1.

The structure of the dataRecord for bootSoftwareFingerprint, applicationSoftwareFingerprint, and applicationDataFingerprint is vehicle manufacturer specific.

15.3.3 Server routine access

Routines are used to perform non-volatile memory access such as erasing non-volatile memory and checking the successful download of a module.

Table 444 defines the standardized routineIdentifiers for non-volatile memory access. Other routineIdentifier numbers used in the programming sequence are specified by the vehicle manufacturer.

Table 444 — routineIdentifiers for non-volatile memory access

Byte Value	Description	Mnemonic
0xFF00	**eraseMemory**	EM
	This value shall be used to start the servers memory erase routine. The Control option and status record format shall be ECU-specific and defined by the vehicle manufacturer.	

15.4 Non-volatile server memory programming message flow examples

15.4.1 General information

The following example presents CAN message traffic for a non-volatile server memory-programming event of a single server. The given message flows are based on a single server and the transfer of two modules, where each module has a length of 511 bytes. The network layer buffer size of the server that is re-programmed is 255 bytes (reported in the RequestDownload positive response message). The programming example uses the 11 bit OBD CAN Identifiers as specified in ISO 15765-4. Therefore, all frames must be padded with filler bytes (DLC = 8). All CAN frames of a request message are padded with a filler byte of 0x55. All CAN frames of a response message are padded with a filler byte of 0xAA.

NOTE Filler bytes can have any value.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

15.4.2 Programming phase #1 — Pre-Programming step

See Table 445 through Table 447.

Table 445 — StartDiagnosticSessionControl(extendedSession)

Relative Time	Ch.#	CAN ID	Client Request/Server Response	DLC	PCI and frame data bytes	Comments
27.2174	1	7DF	Func. Request	8	02 10 03 55 55 55 55 55	DSC message-SF
0,0001	1	7E8	Response	8	06 50 03 00 96 17 70 AA	DSC message-SF
0,0002	1	7E9	Response	8	06 50 03 00 96 17 70 AA	DSC message-SF

Table 446 — ControlDTCSetting(off)

Relative Time	Ch. #	CAN ID	Client Request/Server Response	DLC	PCI and frame data bytes	Comments
0,0505	1	7DF	Func. Request	8	02 85 02 55 55 55 55 55	CDTCS message-SF
0,0001	1	7E8	Response	8	02 C5 02 AA AA AA AA AA	CDTCS message-SF
0,0001	1	7E9	Response	8	02 C5 02 AA AA AA AA AA	CDTCS message-SF

Table 447 — CommunicationControl(disableRxAndTx in the application)

Relative Time	Ch. #	CAN ID	Client Request/Server Response	DLC	PCI and frame data bytes	Comments
1,0007	1	7DF	Func. Request	8	03 28 03 01 55 55 55 55	CC message-SF
0,0001	1	7E8	Response	8	02 68 03 AA AA AA AA AA	CC message-SF
0,0001	1	7E9	Response	8	02 68 03 AA AA AA AA AA	CC message-SF
NOTE After the successful execution of the CommunicationControl with the subfunction disableRxAndTx in the application, a functional addressed TesterPresent message with suppressPosRspMsgIndicationBit (bit 7 of subfunction) = TRUE (1) (no response) is sent approx. every 2 s to keep all servers in this state in order to not send normal communication messages.						

15.4.3 Programming phase #1 — Programming step

See Table 448 through Table 463

Table 448 — DiagnosticSessionControl(programmingSession)

Relative Time	Ch. #	CAN ID	Client Request/Server Response	DLC	PCI and frame data bytes	Comments
1.6964	1	7E0	Phys. Request	8	02 10 02 55 55 55 55 55	DSC message-SF
0,0012	1	7E8	Response	8	06 50 02 00 FA 0B B8 AA	DSC message-SF
1,9987	1	7DF	Func. Request	8	02 3E 80 55 55 55 55 55	TP message-SF

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 449 — SecurityAccess(requestSeed)

Relative Time	Ch. #	CAN ID	Client Request/Server Response	DLC	PCI and frame data bytes	Comments
1,0000	1	7E0	Phys. Request	8	02 27 01 55 55 55 55 55	SA message-SF
0,0008	1	7E8	Response	8	04 67 01 21 74 AA AA AA	SA message-SF
0,9989	1	7DF	Func. Request	8	02 3E 80 55 55 55 55 55	TP message-SF

Table 450 — SecurityAccess(sendKey)

Relative Time	Ch. #	CAN ID	Client Request/Server Response	DLC	PCI and frame data bytes	Comments
1,9998	1	7E0	Phys. Request	8	04 27 02 47 11 55 55 55	SA message-SF
0,0002	1	7DF	Func. Request	8	02 3E 80 55 55 55 55 55	TP message-SF
0,0008	1	7E8	Response	8	02 67 02 AA AA AA AA AA	SA message-SF
1,9992	1	7DF	Func. Request	8	02 3E 80 55 55 55 55 55	TP message-SF

Table 451 — RoutineControl(eraseMemory)

Relative Time	Ch. #	CAN ID	Client Request/Server Response	DLC	PCI and frame data bytes	Comments
0.9995	1	7E0	Phys. Request	8	04 31 01 FF 00 55 55 55	RC message-SF
0,0001	1	7E8	Response	8	03 7F 31 78 AA AA AA AA	NR w/ NRC78-SF
1,0004	1	7DF	Func. Request	8	02 3E 80 55 55 55 55 55	TP message-SF
1,9995	1	7E8	Response	8	03 7F 31 78 AA AA AA AA	NR w/ NRC78-SF
0,0005	1	7DF	Func. Request	8	02 3E 80 55 55 55 55 55	TP message-SF
2,0001	1	7DF	Func. Request	8	02 3E 80 55 55 55 55 55	TP message-SF
1,0002	1	7E8	Response	8	04 71 01 FF 00 AA AA AA	RC message-SF
0,9998	1	7DF	Func. Request	8	02 3E 80 55 55 55 55 55	TP message-SF

Table 452 — RequestDownload — Module #1

Relative Time	Ch. #	CAN ID	Client Request/Server Response	DLC	PCI and frame data bytes	Comments
1,9989	1	7E0	Phys. Request	8	10 09 34 00 33 00 19 68	RD message-FF
0,0001	1	7E8	Response	8	30 00 00 AA AA AA AA AA	FlowControl
0,0010	1	7DF	Func. Request	8	02 3E 80 55 55 55 55 55	TP message-SF
0,0001	1	7E0	Phys. Request	8	21 00 01 FF 55 55 55 55	RD message-CF
0,0012	1	7E8	Response	8	04 74 20 00 FF AA AA AA	RD message-SF
1,9987	1	7DF	Func. Request	8	02 3E 80 55 55 55 55 55	TP message-SF

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 453 — TransferData — Module #1 (block #1)

Relative Time	Ch. #	CAN ID	Client Request/Server Response	DLC	PCI and frame data bytes	Comments
0.9996	1	7E0	Phys. Request	8	10 FF 36 01 02 03 04 05	TD message-FF)
0,0001	1	7E8	Response	8	30 00 00 AA AA AA AA AA	FlowControl
0,0012	1	7E0	Phys. Request	8	21 06 07 08 09 0A 0B 0C	TD message-CF
0,0010	1	7E0	Phys. Request	8	22 0D 0E 0F 10 11 12 13	TD message-CF
0,0010	1	7E0	Phys. Request	8	23 14 15 16 17 18 19 1A	TD message-CF
:	:	:	:	:	:	:
0,0010	1	7E0	Phys. Request	8	23 F4 F5 F6 F7 F8 F9 FA	TD message-CF
0,0009	1	7E0	Phys. Request	8	24 FB FC FD FE 55 55 55	TD message-CF
0,0011	1	7E8	Response	8	02 76 01 AA AA AA AA AA	TD message-SF
0,9630	1	7DF	Func. Request	8	02 3E 80 55 55 55 55 55	TP message-SF

Table 454 — TransferData — Module #1 (block #2)

Relative Time	Ch. #	CAN ID	Client Request/Server Response	DLC	PCI and frame data bytes	Comments
1,9994	1	7E0	Phys. Request	8	10 FF 36 02 02 03 04 05	TD message (FF)
0,0001	1	7E8	Response	8	30 00 00 AA AA AA AA AA	FlowControl
0,0012	1	7E0	Phys. Request	8	21 06 07 08 09 0A 0B 0C	TD message (CF)
0,0010	1	7E0	Phys. Request	8	22 0D 0E 0F 10 11 12 13	TD message (CF)
0,0010	1	7E0	Phys. Request	8	23 14 15 16 17 18 19 1A	TD message (CF)
:	:	:	:	:	:	:
0,0010	1	7E0	Phys. Request	8	23 F4 F5 F6 F7 F8 F9 FA	TD message (CF)
0,0009	1	7E0	Phys. Request	8	24 FB FC FD FE 55 55 55	TD message (CF)
0,0011	1	7E8	Response	8	02 76 02 AA AA AA AA AA	TD message
1,9633	1	7DF	Func. Request	8	02 3E 80 55 55 55 55 55	TP message

Table 455 — TransferData — Module #1 (block #3)

Relative Time	Ch. #	CAN ID	Client Request/Server Response	DLC	PCI and frame data bytes	Comments
0,9991	1	7E0	Phys. Request	8	07 36 03 02 03 04 05 06	TD message-SF
0,0011	1	7E8	Response	8	02 76 03 AA AA AA AA AA	TD message-SF
0,9998	1	7DF	Func. Request	8	02 3E 80 55 55 55 55 55	TP message-SF

Table 456 — RequestTransferExit — Module #1

Relative Time	Ch. #	CAN ID	Client Request/Server Response	DLC	PCI and frame data bytes	Comments
1,9999	1	7E0	Phys. Request	8	01 37 55 55 55 55 55 55	RTE message-SF
0,0002	1	7DF	Func. Request	8	02 3E 80 55 55 55 55 55	TP message-SF
0,0009	1	7E8	Response	8	01 77 AA AA AA AA AA AA	RTE message-SF
1,9992	1	7DF	Func. Request	8	02 3E 80 55 55 55 55 55	TP message-SF
2,0001	1	7DF	Func. Request	8	02 3E 80 55 55 55 55 55	TP message-SF

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 457 — RequestDownload — Module #2

Relative Time	Ch. #	CAN ID	Client Request/Server Response	DLC	PCI and frame data bytes	Comments
1,9995	1	7E0	Phys. Request	8	10 09 34 00 33 00 1B 67	RD message-FF
0,0001	1	7E8	Response	8	30 00 00 AA AA AA AA AA	FlowControl
0,0004	1	7DF	Func. Request	8	02 3E 80 55 55 55 55 55	TP message-SF
0,0007	1	7E0	Phys. Request	8	21 00 01 FF 55 55 55 55	RD message-CF
0,0012	1	7E8	Response	8	04 74 20 00 FF AA AA AA	RD message-SF
1,9982	1	7DF	Func. Request	8	02 3E 80 55 55 55 55 55	TP message-SF

Table 458 — TransferData — Module #2 (block #1)

Relative Time	Ch. #	CAN ID	Client Request/Server Response	DLC	PCI and frame data bytes	Comments
1,0002	1	7E0	Phys. Request	8	10 FF 36 01 02 03 04 05	TD message-FF
0,0001	1	7E8	Response	8	30 00 00 AA AA AA AA AA	FlowControl
0,0012	1	7E0	Phys. Request	8	21 06 07 08 09 0A 0B 0C	TD message-CF
0,0010	1	7E0	Phys. Request	8	22 0D 0E 0F 10 11 12 13	TD message-CF
0,0010	1	7E0	Phys. Request	8	23 14 15 16 17 18 19 1A	TD message-CF
:	:	:	:	:	:	:
0,0010	1	7E0	Phys. Request	8	23 F4 F5 F6 F7 F8 F9 FA	TD message-CF
0,0009	1	7E0	Phys. Request	8	24 FB FC FD FE 55 55 55	TD message-CF
0,0011	1	7E8	Response	8	02 76 01 AA AA AA AA AA	TD message-SF
1,9626	1	7DF	Func. Request	8	02 3E 80 55 55 55 55 55	TP message-SF

Table 459 — TransferData — Module #2 (block #2)

Relative Time	Ch. #	CAN ID	Client Request/Server Response	DLC	PCI and frame data bytes	Comments
1,9994	1	7E0	Phys. Request	8	10 FF 36 02 02 03 04 05	TD message-FF
0,0001	1	7E8	Response	8	30 00 00 AA AA AA AA AA	FlowControl
0,0012	1	7E0	Phys. Request	8	21 06 07 08 09 0A 0B 0C	TD message-CF
0,0010	1	7E0	Phys. Request	8	22 0D 0E 0F 10 11 12 13	TD message-CF
0,0010	1	7E0	Phys. Request	8	23 14 15 16 17 18 19 1A	TD message-CF
:	:	:	:	:	:	:
0,0010	1	7E0	Phys. Request	8	23 F4 F5 F6 F7 F8 F9 FA	TD message-CF
0,0009	1	7E0	Phys. Request	8	24 FB FC FD FE 55 55 55	TD message-CF
0,0011	1	7E8	Response	8	02 76 02 AA AA AA AA AA	TD message-SF
1,9633	1	7DF	Func. Request	8	02 3E 80 55 55 55 55 55	TP message-SF

Table 460 — TransferData — Module #2 (block #3)

Relative Time	Ch. #	CAN ID	Client Request/Server Response	DLC	PCI and frame data bytes	Comments
0,9996	1	7E0	Phys. Request	8	07 36 03 02 03 04 05 06	TD message-FF
0,0011	1	7E8	Response	8	02 76 03 AA AA AA AA AA	TD message-SF
0,9993	1	7DF	Func. Request	8	02 3E 80 55 55 55 55 55	TP message-SF
2,0001	1	7DF	Func. Request	8	02 3E 80 55 55 55 55 55	TP message-SF

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table 461 — RequestTransferExit — Module #2

Relative Time	Ch. #	CAN ID	Client Request/Server Response	DLC	PCI and frame data bytes	Comments
0,0002	1	7E0	Phys. Request	8	01 37 55 55 55 55 55 55	RTE message-SF
0,0011	1	7E8	Response	8	01 77 AA AA AA AA AA AA	RTE message-SF
1,9987	1	7DF	Func. Request	8	02 3E 80 55 55 55 55 55	TP message-SF
2,0001	1	7DF	Func. Request	8	02 3E 80 55 55 55 55 55	TP message-SF

Table 462 — RoutineControl(validate application)

Relative Time	Ch. #	CAN ID	Client Request/Server Response	DLC	PCI and frame data bytes	Comments
1,0012	1	7E0	Phys. Request	8	04 31 01 FF 01 55 55 55	RC message-SF
0,0001	1	7E8	Response	8	03 7F 31 78 AA AA AA AA	NR w/ NRC78-SF
0,9987	1	7DF	Func. Request	8	02 3E 80 55 55 55 55 55	TP message-SF
2,0001	1	7DF	Func. Request	8	02 3E 80 55 55 55 55 55	TP message-SF
0,0011	1	7E8	Response	8	03 7F 31 78 AA AA AA AA	NR w/ NRC78-SF
1,9990	1	7DF	Func. Request	8	02 3E 80 55 55 55 55 55	TP message-SF
1,0019	1	7E8	Response	8	04 71 01 FF 01 AA AA AA	RC message-SF
0,9982	1	7DF	Func. Request	8	02 3E 80 55 55 55 55 55	TP message-SF
2,0001	1	7DF	Func. Request	8	02 3E 80 55 55 55 55 55	TP message-SF

Table 463 — WriteDataByIdentifier — dataIdentifier = VIN

Relative Time	Ch. #	CAN ID	Client Request/Server Response	DLC	PCI and frame data bytes	Comments
0,0004	1	7E0	Phys. Request	8	10 14 2E F1 90 57 41 4C	WDBI message-FF
0,0001	1	7E8	Response	8	30 00 00 AA AA AA AA AA	FlowControl
0,0012	1	7E0	Phys. Request	8	21 54 4F 4E 53 2D 57 45	WDBI message-CF
0,0010	1	7E0	Phys. Request	8	22 42 2E 43 4F 4D 20 20	WDBI message-CF
0,0011	1	7E8	Response	8	03 6E F1 90 AA AA AA AA	WDBI message-SF
1,9961	1	7DF	Func. Request	8	02 3E 80 55 55 55 55 55	TP message-SF
2,0001	1	7DF	Func. Request	8	02 3E 80 55 55 55 55 55	TP message-SF

15.4.4 Programming phase #1 — Post-Programming step

See Table 464.

Table 464 — ECUReset — hardReset

Relative Time	Ch. #	CAN ID	Client Request/Server Response	DLC	PCI and frame data bytes	Comments
0,3946	1	7DF	Func. Request	8	02 11 01 55 55 55 55 55	ER message-SF
0,0011	1	7E8	Response	8	02 51 01 AA AA AA AA AA	ER message-SF
0,0001	1	7E9	Response	8	02 51 01 AA AA AA AA AA	ER message-SF

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Annex A
(normative)

Global parameter definitions

A.1 Negative response codes

Table A.1 defines all negative response codes used within this standard. Each diagnostic service specifies applicable negative response codes. The diagnostic service implementation in the server may also utilise additional and applicable negative response codes specified in this as defined by the vehicle manufacturer.

The negative response code range 0x00 – 0xFF is divided into three ranges:

— 0x00: positiveResponse parameter value for server internal implementation,

— 0x01 – 0x7F: communication related negative response codes,

— 0x80 – 0xFF: negative response codes for specific conditions that are not correct at the point in time the request is received by the server. These response codes may be utilised whenever response code 0x22 (conditionsNotCorrect) is listed as valid in order to report more specifically why the requested action can not be taken.

Table A.1 — Negative Response Code (NRC) definition and values

Byte value	Negative Response Code (NRC) definition	Mnemonic
0x00	**positiveResponse** This NRC shall not be used in a negative response message. This positiveResponse parameter value is reserved for server internal implementation. Refer to 7.5.5.	PR
0x01 – 0x0F	**ISOSAEReserved** This range of values is reserved by this document for future definition.	ISOSAERESRVD
0x10	**generalReject** This NRC indicates that the requested action has been rejected by the server. The generalReject response code shall only be implemented in the server if none of the negative response codes defined in this document meet the needs of the implementation. At no means shall this NRC be a general replacement for the response codes defined in this document.	GR
0x11	**serviceNotSupported** This NRC indicates that the requested action will not be taken because the server does not support the requested service. The server shall send this NRC in case the client has sent a request message with a service identifier which is unknown, not supported by the server, or is specified as a response service identifier. Therefore this negative response code is not shown in the list of negative response codes to be supported for a diagnostic service, because this negative response code is not applicable for supported services.	SNS

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table A.1 — *(continued)*

Byte value	Negative Response Code (NRC) definition	Mnemonic
0x12	**sub-functionNotSupported** This NRC indicates that the requested action will not be taken because the server does not support the service specific parameters of the request message. The server shall send this NRC in case the client has sent a request message with a known and supported service identifier but with "sub-function“ which is either unknown or not supported.	SFNS
0x13	**incorrectMessageLengthOrInvalidFormat** This NRC indicates that the requested action will not be taken because the length of the received request message does not match the prescribed length for the specified service or the format of the paramters do not match the prescribed format for the specified service.	IMLOIF
0x14	**responseTooLong** This NRC shall be reported by the server if the response to be generated exceeds the maximum number of bytes available by the underlying network layer. This could occur if the response message exceeds the maximum size allowed by the underlying transport protocol or if the response message exceeds the server buffer size allocated for that purpose. EXAMPLE This problem may occur when several DIDs at a time are requested and the combination of all DIDs in the response exceeds the limit of the underlying transport protocol.	RTL
0x15 – 0x20	**ISOSAEReserved** This range of values is reserved by this document for future definition.	ISOSAERESRVD
0x21	**busyRepeatRequest** This NRC indicates that the server is temporarily too busy to perform the requested operation. In this circumstance the client shall perform repetition of the "identical request message" or "another request message". The repetition of the request shall be delayed by a time specified in the respective implementation documents. EXAMPLE In a multi-client environment the diagnostic request of one client might be blocked temporarily by a NRC 0x21 while a different client finishes a diagnostic task. If the server is able to perform the diagnostic task but needs additional time to finish the task and prepare the response, the NRC 0x78 shall be used instead of NRC 0x21. This NRC is in general supported by each diagnostic service, as not otherwise stated in the data link specific implementation document, therefore it is not listed in the list of applicable response codes of the diagnostic services.	BRR
0x22	**conditionsNotCorrect** This NRC indicates that the requested action will not be taken because the server prerequisite conditions are not met.	CNC
0x23	**ISOSAEReserved** This range of values is reserved by this document for future definition.	ISOSAERESRVD

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table A.1 — *(continued)*

Byte value	Negative Response Code (NRC) definition	Mnemonic
0x24	**requestSequenceError** This NRC indicates that the requested action will not be taken because the server expects a different sequence of request messages or message as sent by the client. This may occur when sequence sensitive requests are issued in the wrong order. EXAMPLE A successful SecurityAccess service specifies a sequence of requestSeed and sendKey as sub-fuctions in the request messages. If the sequence is sent different by the client the server shall send a negative response message with the negative response code 0x24 requestSequenceError.	RSE
0x25	**noResponseFromSubnetComponent** This NRC indicates that the server has received the request but the requested action could not be performed by the server as a subnet component which is necessary to supply the requested information did not respond within the specified time. The noResponseFromSubnetComponent negative response shall be implemented by gateways in electronic systems which contain electronic subnet components and which do not directly respond to the client's request. The gateway may receive the request for the subnet component and then request the necessary information from the subnet component. If the subnet component fails to respond, the server shall use this negative response to inform the client about the failure of the subnet component. This NRC is in general supported by each diagnostic service, as not otherwise stated in the data link specific implementation document, therefore it is not listed in the list of applicable response codes of the diagnostic services.	NRFSC
0x26	**FailurePreventsExecutionOfRequestedAction** This NRC indicates that the requested action will not be taken because a failure condition, identified by a DTC (with at least one DTC status bit for TestFailed, Pending, Confirmed or TestFailedSinceLastClear set to 1), has occurred and that this failure condition prevents the server from performing the requested action. This NRC can, for example, direct the technician to read DTCs in order to identify and fix the problem. NOTE This implies that diagnostic services used to access DTCs shall not implement this NRC as an external test tool may check for the above NRC and automatically request DTCs whenever the above NRC has been received. This NRC is in general supported by each diagnostic service (except the services mentioned above), as not otherwise stated in the data link specific implementation document, therefore it is not listed in the list of applicable response codes of the diagnostic services.	FPEORA
0x27 – 0x30	**ISOSAEReserved** This range of values is reserved by this document for future definition.	ISOSAERESRVD
0x31	**requestOutOfRange** This NRC indicates that the requested action will not be taken because the server has detected that the request message contains a parameter which attempts to substitute a value beyond its range of authority (e.g. attempting to substitute a data byte of 111 when the data is only defined to 100), or which attempts to access a dataIdentifier/routineIdentifer that is not supported or not supported in active session. This NRC shall be implemented for all services, which allow the client to read data, write data or adjust functions by data in the server.	ROOR

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table A.1 — *(continued)*

Byte value	Negative Response Code (NRC) definition	Mnemonic
0x32	**ISOSAEReserved** This range of values is reserved by this document for future definition.	ISOSAERESRVD
0x33	**securityAccessDenied** This NRC indicates that the requested action will not be taken because the server's security strategy has not been satisfied by the client. The server shall send this NRC if one of the following cases occur: — the test conditions of the server are not met, — the required message sequence e.g. DiagnosticSessionControl, securityAccess is not met, — the client has sent a request message which requires an unlocked server. Beside the mandatory use of this negative response code as specified in the applicable services within this standard, this negative response code can also be used for any case where security is required and is not yet granted to perform the required service.	SAD
0x34	**ISOSAEReserved** This range of values is reserved by this document for future definition.	ISOSAERESRVD
0x35	**invalidKey** This NRC indicates that the server has not given security access because the key sent by the client did not match with the key in the server's memory. This counts as an attempt to gain security. The server shall remain locked and increment ist internal securityAccessFailed counter.	IK
0x36	**exceedNumberOfAttempts** This NRC indicates that the requested action will not be taken because the client has unsuccessfully attempted to gain security access more times than the server's security strategy will allow.	ENOA
0x37	**requiredTimeDelayNotExpired** This NRC indicates that the requested action will not be taken because the client's latest attempt to gain security access was initiated before the server's required timeout period had elapsed.	RTDNE
0x38 – 0x4F	**reservedByExtendedDataLinkSecurityDocument** This range of values is reserved by extended data link security.	RBEDLSD
0x50 – 0x6F	**ISOSAEReserved** This range of values is reserved by this document for future definition.	ISOSAERESRVD
0x70	**uploadDownloadNotAccepted** This NRC indicates that an attempt to upload/download to a server's memory cannot be accomplished due to some fault conditions.	UDNA
0x71	**transferDataSuspended** This NRC indicates that a data transfer operation was halted due to some fault. The active transferData sequence shall be aborted.	TDS
0x72	**generalProgrammingFailure** This NRC indicates that the server detected an error when erasing or programming a memory location in the permanent memory device (e.g. Flash Memory).	GPF

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table A.1 — *(continued)*

Byte value	Negative Response Code (NRC) definition	Mnemonic
0x73	**wrongBlockSequenceCounter** This NRC indicates that the server detected an error in the sequence of blockSequenceCounter values. Note that 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
0x74 – 0x77	**ISOSAEReserved** This range of values is reserved by this document for future definition.	ISOSAERESRVD
0x78	**requestCorrectlyReceived-ResponsePending** This NRC indicates that the request message was received correctly, and that all parameters in the request message were valid, but the action to be performed is not yet completed and the server is not yet ready to receive another request. As soon as the requested service has been completed, the server shall send a positive response message or negative response message with a response code different from this. The negative response message with this NRC may be repeated by the server until the requested service is completed and the final response message is sent. This NRC might impact the application layer timing parameter values. The detailed specification shall be included in the data link specific implementation document. This NRC shall only be used in a negative response message if the server will not be able to receive further request messages from the client while completing the requested diagnostic service. When this NRC is used, the server shall always send a final response (positive or negative) independent of the suppressPosRspMsgIndicationBit value or the suppress requirement for responses with NRCs SNS, SFNS, SNSIAS, SFNSIAS and ROOR on functionally addressed requests. A typical example where this NRC may be used is when the client has sent a request message, which includes data to be programmed or erased in flash memory of the server. If the programming/erasing routine (usually executed out of RAM) is not able to support serial communication while writing to the flash memory the server shall send a negative response message with this response code. This NRC is in general supported by each diagnostic service, as not otherwise stated in the data link specific implementation document, therefore it is not listed in the list of applicable response codes of the diagnostic services.	RCRRP
0x79 – 0x7D	**ISOSAEReserved** This range of values is reserved by this document for future definition.	ISOSAERESRVD
0x7E	**sub-functionNotSupportedInActiveSession** This NRC indicates that the requested action will not be taken because the server does not support the requested sub-function in the session currently active. This NRC shall only be used when the requested sub-function is known to be supported in another session, otherwise response code SFNS (sub-functionNotSupported) shall be used (e.g., servers executing the boot software generally do not know which subfunctions are supported in the application (and vice versa) and therefore may need to respond with NRC 0x12 instead). This NRC shall be supported by each diagnostic service with a sub-function parameter, if not otherwise stated in the data link specific implementation document, therefore it is not listed in the list of applicable response codes of the diagnostic services.	SFNSIAS

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table A.1 — *(continued)*

Byte value	Negative Response Code (NRC) definition	Mnemonic
0x7F	**serviceNotSupportedInActiveSession** This NRC indicates that the requested action will not be taken because the server does not support the requested service in the session currently active. This NRC shall only be used when the requested service is known to be supported in another session, otherwise response code SNS (serviceNotSupported) shall be used (e.g., servers executing the boot software generally do not know which services are supported in the application (and vice versa) and therefore may need to respond with NRC 0x11 instead). This NRC is in general supported by each diagnostic service, as not otherwise stated in the data link specific implementation document, therefore it is not listed in the list of applicable response codes of the diagnostic services.	SNSIAS
0x80	**ISOSAEReserved** This range of values is reserved by this document for future definition.	ISOSAERESRVD
0x81	**rpmTooHigh** This NRC indicates that the requested action will not be taken because the server prerequisite condition for RPM is not met (current RPM is above a pre-programmed maximum threshold).	RPMTH
0x82	**rpmTooLow** This NRC indicates that the requested action will not be taken because the server prerequisite condition for RPM is not met (current RPM is below a pre-programmed minimum threshold).	RPMTL
0x83	**engineIsRunning** This NRC is required for those actuator tests which cannot be actuated while the Engine is running. This is different from RPM too high negative response, and needs to be allowed.	EIR
0x84	**engineIsNotRunning** This NRC is required for those actuator tests which cannot be actuated unless the Engine is running. This is different from RPM too low negative response, and needs to be allowed.	EINR
0x85	**engineRunTimeTooLow** This NRC indicates that the requested action will not be taken because the server prerequisite condition for engine run time is not met (current engine run time is below a pre-programmed limit).	ERTTL
0x86	**temperatureTooHigh** This NRC indicates that the requested action will not be taken because the server prerequisite condition for temperature is not met (current temperature is above a pre-programmed maximum threshold).	TEMPTH
0x87	**temperatureTooLow** This NRC indicates that the requested action will not be taken because the server prerequisite condition for temperature is not met (current temperature is below a pre-programmed minimum threshold).	TEMPTL
0x88	**vehicleSpeedTooHigh** This NRC indicates that the requested action will not be taken because the server prerequisite condition for vehicle speed is not met (current VS is above a pre-programmed maximum threshold).	VSTH

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table A.1 — *(continued)*

Byte value	Negative Response Code (NRC) definition	Mnemonic
0x89	**vehicleSpeedTooLow** This NRC indicates that the requested action will not be taken because the server prerequisite condition for vehicle speed is not met (current VS is below a pre-programmed minimum threshold).	VSTL
0x8A	**throttle/PedalTooHigh** This NRC indicates that the requested action will not be taken because the server prerequisite condition for throttle/pedal position is not met (current TP/APP is above a pre-programmed maximum threshold).	TPTH
0x8B	**throttle/PedalTooLow** This NRC indicates that the requested action will not be taken because the server prerequisite condition for throttle/pedal position is not met (current TP/APP is below a pre-programmed minimum threshold).	TPTL
0x8C	**transmissionRangeNotInNeutral** This NRC indicates that the requested action will not be taken because the server prerequisite condition for being in neutral is not met (current transmission range is not in neutral).	TRNIN
0x8D	**transmissionRangeNotInGear** This NRC indicates that the requested action will not be taken because the server prerequisite condition for being in gear is not met (current transmission range is not in gear).	TRNIG
0x8E	**ISOSAEReserved** This range of values is reserved by this document for future definition.	ISOSAERESRVD
0x8F	**brakeSwitch(es)NotClosed (Brake Pedal not pressed or not applied)** This NRC indicates that for safety reasons, this is required for certain tests before it begins, and must be maintained for the entire duration of the test.	BSNC
0x90	**shifterLeverNotInPark** This NRC indicates that for safety reasons, this is required for certain tests before it begins, and must be maintained for the entire duration of the test.	SLNIP
0x91	**torqueConverterClutchLocked** This NRC indicates that the requested action will not be taken because the server prerequisite condition for torque converter clutch is not met (current TCC status above a pre-programmed limit or locked).	TCCL
0x92	**voltageTooHigh** This NRC indicates that the requested action will not be taken because the server prerequisite condition for voltage at the primary pin of the server (ECU) is not met (current voltage is above a pre-programmed maximum threshold).	VTH
0x93	**voltageTooLow** This NRC indicates that the requested action will not be taken because the server prerequisite condition for voltage at the primary pin of the server (ECU) is not met (current voltage is below a pre-programmed maximum threshold).	VTL
0x94 – 0xEF	**reservedForSpecificConditionsNotCorrect** This range of values is reserved by this document for future definition.	RFSCNC
0xF0 – 0xFE	**vehicleManufacturerSpecificConditionsNotCorrect** This range of values is reserved for vehicle manufacturer specific condition not correct scenarios.	VMSCNC

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table A.1 — *(continued)*

Byte value	Negative Response Code (NRC) definition	Mnemonic
0xFF	**ISOSAEReserved** This range of values is reserved by this document for future definition.	ISOSAERESRVD

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Annex B
(normative)

Diagnostic and communication management functional unit data-parameter definitions

B.1 communicationType parameter definition

The communicationType is a 1-byte value. The bit-encoded low nibble of this byte represents the communicationTypes, which can be controlled via the CommunicationControl (0x28) service. For example, a communicationType with a bit combination (Bits 1-0) of "11b" is valid and disables both "normalCommunicationMessages" and "networkManagementCommunicationMessages" messages.
The high nibble of the communicationType 1-byte value defines which of the subnets connected to the receiving node shall be disabled / enabled when an appropriate CommunicationControl service is received.

Table B.1 defines the communicationType and subnetNumber byte.

Table B.1 — Definition of communicationType and subnetNumber byte

Encoding of bit	Value	Description	Cvt	Mnemonic
0 – 1	0x0	**ISOSAEReserved**	M	
	0x1	**normalCommunicationMessages** This value references all application-related communication (inter-application signal exchange between multiple in-vehicle servers).	U	NCM
	0x2	**networkManagementCommunicationMessages** This value references all network management related communication.	U	NWMCM
	0x3	**networkManagementCommunicationMessages and normalCommunicationMessages** This value references all network management and application-related communication.	U	NWMCM-NCM
2 – 3	0x0 – 0x3	**ISOSAEReserved**	M	ISOSAERESRVD
4 – 7	0x0	**Disable / Enable specified communicationType** See encoding of bit 0-1. In the receiving node including communication to all connected networks. This only disables the node's communication into the connected networks but not the communication of other nodes on the networks (i.e., receiving node is not responsible to disable communication in each node of the network).	U	DISENSCT
	0x1 – 0xE	**Disable / Enable specific subnet identified by subnet number**	U	DISENSSIVSN
	0xF	**Disable/Enable network which request is received on (Receiving node (server))**	U	DENWRIRO

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

B.2 eventWindowTime parameter definition

Table B.2 defines the eventWindowTime parameter values.

Table B.2 — Definition of eventWindowTime parameter values

Byte Value	Description	Cvt	Mnemonic
0x00 – 0x01	**ISOSAEReserved** This value is reserved by the document	M	ISOSAERESRVD
0x02	**infiniteTimeToResponse** This value specifies that the event window shall stay active for an infinite amount of time (e.g. open window until power off).	U	ITTR
0x03 – 0x7F	**vehicleManufacturerSpecific** This range of values is reserved for vehicle manufacturer specific use. The resolution of the eventWindowTime parameter is left vehicle manufacturer discretionary.	U	VMS
0x80 – 0xFF	**ISOSAEReserved** This range of values is reserved by this document for future definition.	M	ISOSAERESRVD

B.3 linkControlModeIdentifier parameter definition

Table B.3 defines the linkControlModeIdentifier values.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table B.3 — Definition of linkControlModeIdentifier values

Byte Value	Description	Cvt	Mnemonic
0x00	**ISOSAEReserved** This value is reserved by this document for future definition.	M	ISOSAERESRVD
0x01	**PC9600Baud** This value specifies the standard PC baudrate of 9.6 KBaud.	U	PC9600
0x02	**PC19200Baud** This value specifies the standard PC baudrate of 19.2 KBaud.	U	PC19200
0x03	**PC38400Baud** This value specifies the standard PC baudrate of 38.4 KBaud.	U	PC38400
0x04	**PC57600Baud** This value specifies the standard PC baudrate of 57.6 KBaud.	U	PC57600
0x05	**PC115200Baud** This value specifies the standard PC baudrate of 115.2 KBaud.	U	PC115200
0x06 – 0x0F	**ISOSAEReserved** This range of values is reserved by this document for future definition.	M	ISOSAERESRVD
0x10	**CAN125000Baud** This value specifies the standard CAN baudrate of 125 KBaud.	U	CAN125000
0x11	**CAN250000Baud** This value specifies the standard CAN baudrate of 250 KBaud.	U	CAN250000
0x12	**CAN500000Baud** This value specifies the standard CAN baudrate of 500 KBaud.	U	CAN500000
0x13	**CAN1000000Baud** This value specifies the standard CAN baudrate of 1 MBaud.	U	CAN1000000
0x14 – 0x1F	**ISOSAEReserved** This range of values is reserved by this document for future definition.	M	ISOSAERESRVD
0x20	**ProgrammingSetup** This value specifies the programming setup of a network, which can be parameterized depend on the vehicle network requirements.	U	PROGSU
0x21- 0xFF	**ISOSAEReserved** This range of values is reserved by this document for future definition.	M	ISOSAERESRVD

B.4 nodeIdentificationNumber parameter definition

The nodeIdentificationNumber is a 2-byte value which represents a unique identification number of a node somewhere connected to a network in the vehicle where the same node can be connected to different networks in different car lines (e.g a LIN node with an unique node address is connected to network A in one model while the same node is connected to network B in a different model). Therefore the nodeIdentificationNumber provides a mechanism where the associated master node, which the remote node is connected to, transitions the relevant network into a certain diagnostic mode (e.g. disables normal communication on a LIN network). Only the associated master node, which has detected the connection of the related node, identified by the nodeIdentificationNumber, shall perform the requested communicationControl service.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

NOTE This parameter is only available if the controlType value is set to 0x04 or 0x05. Individual parameters will be defined by the vehicle manufacturer.

Table B.4 defines the nodeIdentificationNumber values.

Table B.4 — Definition of nodeIdentificationNumber values

Byte Value	Description	Cvt	Mnemonic
0x0000	**ISOSAEReserved** This value is reserved by this document for future definition.	M	ISOSAERESRVD
0x0001 – 0xFFFF	**nodeIdentificationNumber** These values identify a node connected on a bus system somewhere in the vehicle. Only in case of a valid number the receiving ECU shall carry out the request CommunicationControl function.	U	NIN

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Annex C
(normative)

Data transmission functional unit data-parameter definitions

C.1 DID parameter definitions

The parameter dataIdentifier (DID) logically represents an object (e.g., Air Inlet Door Position) or collection of objects. This parameter shall be available in the server's memory. The dataIdentifier value shall either exist in fixed memory or temporarily stored in RAM if defined dynamically by the service dynamicallyDefineDataIdentifier. In general, a dataIdentifier is capable of being utilized in many diagnostic service requests including 0x22 (readDataByIdentifier), 0x2E (writeDataByIdentifier), and 0x2F (inputOutputControlByIdentifier). A dataIdentifier is also used in various diagnostic service responses (e.g., positive response to service 0x19 subfunction readDTCSnapshotRecordByDTCNumber).

IMPORTANT — Regardless of which service a dataIdentifier is used with, it shall consistently represent the same thing (i.e., a given object with a given size / meaning / etc.) on a given ECU.

The only case this does not apply to is the dynamically defined dataIdentifiers, as they are not predefined in the ECU, but are defined by the client using service 0x2C (dynamicallyDefineDataIdentifier). DataIdentifier values are defined in Table C.1.

Table C.1 — DID data-parameter definitions

Byte Value	Description	Cvt	Mnemonic
0x0000 – 0x00FF	**ISOSAEReserved** This range of values shall be reserved by this document for future definition.	M	ISOSAERESRVD
0x0100 – 0xA5FF	**VehicleManufacturerSpecific** This range of values shall be used to reference vehicle manufacturer specific record data identifiers and input/output identifiers within the server.	U	VMS
0xA600 – 0xA7FF	**ReservedForLegislativeUse** This range of values is reserved for future legislative requirements.	M	RFLU
0xA800 – 0xACFF	**VehicleManufacturerSpecific** This range of values shall be used to reference vehicle manufacturer specific record data identifiers and input/output identifiers within the server.	U	VMS
0xAD00 – 0xAFFF	**ReservedForLegislativeUse** This range of values is reserved for future legislative requirements.	M	RFLU
0xB000 – 0xB1FF	**VehicleManufacturerSpecific** This range of values shall be used to reference vehicle manufacturer specific record data identifiers and input/output identifiers within the server.	U	VMS
0xB200 – 0xBFFF	**ReservedForLegislativeUse** This range of values is reserved for future legislative requirements.	M	RFLU

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table C.1 — *(continued)*

Byte Value	Description	Cvt	Mnemonic
0xC000 – 0xC2FF	**VehicleManufacturerSpecific** This range of values shall be used to reference vehicle manufacturer specific record data identifiers and input/output identifiers within the server.	U	VMS
0xC300 – 0xCEFF	**ReservedForLegislativeUse** This range of values is reserved for future legislative requirements.	M	RFLU
0xCF00 – 0xEFFF	**VehicleManufacturerSpecific** This range of values shall be used to reference vehicle manufacturer specific record data identifiers and input/output identifiers within the server.	U	VMS
0xF000 – 0xF00F	**networkConfigurationDataForTractorTrailerApplicationData-Identifier** This value shall be used to request the remote addresses of all trailer systems independent of their functionality.	U	NCDFTTADID
0xF010 – 0xF0FF	**vehicleManufacturerSpecific** This range of values shall be used to reference vehicle manufacturer specific record data identifiers and input/output identifiers within the server.	U	VMS
0xF100 – 0xF17F	**identificationOptionVehicleManufacturerSpecificDataIdentifier** This range of values shall be used for vehicle manufacturer specific server/vehicle identification options.	U	IDOPTVMSDID
0xF180	**BootSoftwareIdentificationDataIdentifier** This value shall be used to reference the vehicle manufacturer specific ECU boot software identification record. The first data byte of the record data shall be the numberOfModules that are reported. Following the numberOfModules the boot software identification(s) are reported. The format of the boot software identification structure shall be ECU specific and defined by the vehicle manufacturer.	U	BSIDID
0xF181	**applicationSoftwareIdentificationDataIdentifier** This value shall be used to reference the vehicle manufacturer specific ECU application software number(s). The first data byte of the record data shall be the numberOfModules that are reported. Following the numberOfModules the application software identification(s) are reported. The format of the application software identification structure shall be ECU specific and defined by the vehicle manufacturer.	U	ASIDID
0xF182	**applicationDataIdentificationDataIdentifier** This value shall be used to reference the vehicle manufacturer specific ECU application data identification record. The first data byte of the record data shall be the numberOfModules that are reported. Following the numberOfModules the application data identification(s) are reported. The format of the application data identification structure shall be ECU specific and defined by the vehicle manufacturer.	U	ADIDID
0xF183	**bootSoftwareFingerprintDataIdentifier** This value shall be used to reference the vehicle manufacturer specific ECU boot software fingerprint identification record. Record data content and format shall be ECU specific and defined by the vehicle manufacturer.	U	BSFPDID

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table C.1 — *(continued)*

Byte Value	Description	Cvt	Mnemonic
0xF184	**applicationSoftwareFingerprintDataIdentifier** This value shall be used to reference the vehicle manufacturer specific ECU application software fingerprint identification record. Record data content and format shall be ECU specific and defined by the vehicle manufacturer.	U	ASFPDID
0xF185	**applicationDataFingerprintDataIdentifier** This value shall be used to reference the vehicle manufacturer specific ECU application data fingerprint identification record. Record data content and format shall be ECU specific and defined by the vehicle manufacturer.	U	ADFPDID
0xF186	**ActiveDiagnosticSessionDataIdentifier** This value shall be used to report the active diagnostic session in the server. The values are defined by the diagnosticSessionType subfunction parameter in the DiagnosticSessionControl service.	U	ADSDID
0xF187	**vehicleManufacturerSparePartNumberDataIdentifier** This value shall be used to reference the vehicle manufacturer spare part number. Record data content and format shall be server specific and defined by the vehicle manufacturer.	U	VMSPNDID
0xF188	**vehicleManufacturerECUSoftwareNumberDataIdentifier** This value shall be used to reference the vehicle manufacturer ECU (server) software number. Record data content and format shall be server specific and defined by the vehicle manufacturer.	U	VMECUSNDID
0xF189	**vehicleManufacturerECUSoftwareVersionNumberDataIdentifier** This value shall be used to reference the vehicle manufacturer ECU (server) software version number. Record data content and format shall be server specific and defined by the vehicle manufacturer.	U	VMECUSVNDID
0xF18A	**systemSupplierIdentifierDataIdentifier** This value shall be used to reference the system supplier name and address information. Record data content and format shall be server specific and defined by the system supplier.	U	SSIDDID
0xF18B	**ECUManufacturingDateDataIdentifier** This value shall be used to reference the ECU (server) manufacturing date. Record data content and format shall be unsigned numeric, ASCII or BCD, and shall be ordered as Year, Month, Day.	U	ECUMDDID
0xF18C	**ECUSerialNumberDataIdentifier** This value shall be used to reference the ECU (server) serial number. Record data content and format shall be server specific.	U	ECUSNDID
0xF18D	**supportedFunctionalUnitsDataIdentifier** This value shall be used to request the functional units implemented in a server.	U	SFUDID
0xF18E	**VehicleManufacturerKitAssemblyPartNumberDataIdentifier** This value shall be used to reference the vehicle manufacturer order number for a kit (assembled parts bought as a whole for production e.g. cockpit), when the spare part number designates only the server (e.g. for aftersales). The record data content and format shall be server specific and defined by the vehicle manufacturer.	U	VMKAPNDID

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table C.1 — *(continued)*

Byte Value	Description	Cvt	Mnemonic
0xF18F	**ISOSAEReservedStandardized** This range of values shall be reserved by this document for future definition of standardized server/vehicleIdentification options.	M	ISOSAERESRVD
0xF190	**VINDataIdentifier** This value shall be used to reference the VIN number. Record data content and format shall be specified by the vehicle manufacturer.	U	VINDID
0xF191	**vehicleManufacturerECUHardwareNumberDataIdentifier** This value shall be used by reading services to reference the vehicle manufacturer specific ECU (server) hardware number. Record data content and format shall be server specific and defined by vehicle manufacturer.	U	VMECUHNDID
0xF192	**systemSupplierECUHardwareNumberDataIdentifier** This value shall be used to reference the system supplier specific ECU (server) hardware number. Record data content and format shall be server specific and defined by the system supplier.	U	SSECUHWNDID
0xF193	**systemSupplierECUHardwareVersionNumberDataIdentifier** This value shall be used to reference the system supplier specific ECU (server) hardware version number. Record data content and format shall be server specific and defined by the system supplier.	U	SSECUHWVNDID
0xF194	**systemSupplierECUSoftwareNumberDataIdentifier** This value shall be used to reference the system supplier specific ECU (server) software number. Record data content and format shall be server specific and defined by the system supplier.	U	SSECUSWNDID
0xF195	**systemSupplierECUSoftwareVersionNumberDataIdentifier** This value shall be used to reference the system supplier specific ECU (server) software version number. Record data content and format shall be server specific and defined by the system supplier.	U	SSECUSWVNDID
0xF196	**exhaustRegulationOrTypeApprovalNumberDataIdentifier** This value shall be used to reference the exhaust regulation or type approval number (valid for those systems which require type approval). Record data content and format shall be server specific and defined by the vehicle manufacturer. Refer to the relevant legislation for any applicable requirements.	U	EROTANDID
0xF197	**systemNameOrEngineTypeDataIdentifier** This value shall be used to reference the system name or engine type. Record data content and format shall be server specific and defined by the vehicle manufacturer.	U	SNOETDID
0xF198	**repairShopCodeOrTesterSerialNumberDataIdentifier** This value shall be used to reference the repair shop code or tester (client) serial number (e.g., to indicate the most recent service client used re-program server memory). Record data content and format shall be server specific and defined by the vehicle manufacturer.	U	RSCOTSNDID
0xF199	**programmingDateDataIdentifier** This value shall be used to reference the date when the server was last programmed. Record data content and format shall be unsigned numeric, ASCII or BCD, and shall be ordered as Year, Month, Day.	U	PDDID

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table C.1 — *(continued)*

Byte Value	Description	Cvt	Mnemonic
0xF19A	**calibrationRepairShopCodeOrCalibrationEquipmentSerialNumber DataIdentifier** This value shall be used to reference the repair shop code or client serial number (e.g., to indicate the most recent service used by the client to re-calibrate the server). Record data content and format shall be server specific and defined by the vehicle manufacturer.	U	CRSCOCESNDID
0xF19B	**calibrationDateDataIdentifier** This value shall be used to reference the date when the server was last calibrated. Record data content and format shall be unsigned numeric, ASCII or BCD, and shall be ordered as Year, Month, Day.	U	CDDID
0xF19C	**calibrationEquipmentSoftwareNumberDataIdentifier** This value shall be used to reference software version within the client used to calibrate the server. Record data content and format shall be server specific and defined by the vehicle manufacturer.	U	CESWNDID
0xF19D	**ECUInstallationDateDataIdentifier** This value shall be used to reference the date when the ECU (server) was installed in the vehicle. Record data content and format shall be either unsigned numeric, ASCII or BCD, and shall be ordered as Year, Month, Day.	U	EIDDID
0xF19E	**ODXFileDataIdentifier** This value shall be used to reference the ODX (Open Diagnostic Data Exchange) file of the server to be used to interprete and scale the server data.	U	ODXFDID
0xF19F	**EntityDataIdentifier** This value shall be used to reference the entity data identifier for a secured data transmission.	U	EDID
0xF1A0 – 0xF1EF	**identificationOptionVehicleManufacturerSpecific** This range of values shall be used for vehicle manufacturer specific server/vehicle identification options.	U	IDOPTVMS
0xF1F0 – 0xF1FF	**identificationOptionSystemSupplierSpecific** This range of values shall be used for system supplier specific server/vehicle system identification options.	U	IDOPTSSS
0xF200 – 0xF2FF	**periodicDataIdentifier** This range of values shall be used to reference periodic record data identifiers. Those can either be statically or dynamically defined.	U	PDID
0xF300 – 0xF3FF	**DynamicallyDefinedDataIdentifier** This range of values shall be used for dynamicallyDefinedDataIdentifiers.	U	DDDDI
0xF400 – 0xF4FF	**OBDDataIdentifier** This range of values is reserved for OBD/EOBD PIDs as defined in ISO 15031-5.	M	OBDDID
0xF500 – 0xF5FF	**OBDDataIdentifier** This range of values is reserved to represent future defined OBD/EOBD PIDs.	M	OBDDID

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table C.1 — *(continued)*

Byte Value	Description	Cvt	Mnemonic
0xF600 – 0xF6FF	**OBDMonitorDataIdentifier** This range of values is reserved for OBD/EOBD on-board monitoring result values as defined in ISO 15031-5.	M	OBDMDID
0xF700 – 0xF7FF	**OBDMonitorDataIdentifier** This range of values is reserved to represent future defined OBD/EOBD on-board monitoring result values.	M	OBDMDID
0xF800 – 0xF8FF	**OBDInfoTypeDataIdentifier** This range of values is reserved for OBD/EOBD info type values as defined in ISO 15031-5.	M	OBDINFTYPDID
0xF900 – 0xF9FF	**TachographDataIdentifier** This range of values is reserved for Tachograph DIDs as defined in ISO 16844-7.	M	TACHODID
0xFA00 – 0xFA0F	**AirbagDeploymentDataIdentifier** This range of values is reserved for end of life activation of on-board pyrotechnic devices as defined in ISO 26021-2.	M	ADDID
0xFA10	**NumberOfEDRDevices** This value shall be used to report the number of EDR devices capable of reporting EDR data.	U	NOEDRD
0xFA11	**EDRIdentification** This value shall be used to report EDR identification data.	U	EDRI
0xFA12	**EDRDeviceAddressInformation** This value shall be used to report EDR device address information according to the format defined in ISO 26021-2 for dataIdentifier 0xFA02.	U	EDRDAI
0xFA13 – 0xFA18	**EDREntries** This range shall be be used to report individual EDR entries. Each DID shall represent a single EDR entry with 0xFA13 representing the latest EDR entry.	U	EDRES
0xFA19 – 0xFAFF	**SafetySystemDataIdentifier** This range of values is reserved to represent safety system related DIDs.	M	SSDID
0xFB00 – 0xFCFF	**ReservedForLegislativeUse** This range of values is reserved for future legislative requirements.	M	RFLU
0xFD00 – 0xFEFF	**SystemSupplierSpecific** This range of values shall be used to reference system supplier specific record data identifiers and input/output identifiers within the server.	U	SSS
0xFF00	**UDSVersionDataIdentifier** This value shall be used to reference the UDS version implemented in the server. See Table C.11 for the scaling of this DID.	U	UDSVDID
0xFF01 – 0xFFFF	**ISOSAEReserved** This range of values shall be reserved by this document for future definition.	M	ISOSAERESRVD

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

C.2 scalingByte parameter definitions

The parameter scalingByte (SBYT) consists of one byte (high and low nibble). The scalingByte high nibble defines the data type, which is used to represent the dataIdentifier (DID). The scalingByte low nibble defines the number of bytes used to represent the parameter in a datastream.

Table C.2 defines the scalingByte (High Nibble) parameter.

Table C.2 — scalingByte (High Nibble) parameter definitions

Encoding of High Nibble	Description of Data Type	Cvt	Mnemonic
0x0	**unSignedNumeric (1 to 4 bytes)** This encoding uses a common binary weighting scheme to represent a value by mean of discrete incremental steps. One byte affords 256 steps; two bytes yields 65 536 steps, etc.	U	USN
0x1	**signedNumeric (1 to 4 bytes)** This encoding uses a two's complement binary weighting scheme to represent a value by mean of discrete incremental steps. One byte affords 256 steps; two bytes yields 65 536 steps, etc.	U	SN
0x2	**bitMappedReportedWithOutMask** Bit mapped encoding uses individual bits or small groups of bits to represent status. A validity mask is used to indicate the validity of each bit for particular applications. BitMappedReportedWithOutMask encoding signifies that a validity mask is not part of the parameter definition itself. A separate scalingByteExtension (see C.3.1) is required to report the validity mask.	U	BMRWOM
0x3	**bitMappedReportedWithMask** Bit mapped encoding uses individual bits or small groups of bits to represent status. BitMappedReportedWithMask encoding signifies that a validity mask in included as part of the parameter definition itself. For every bit which represents status, a corresponding mask bit is required as part of the parameter definition. The mask indicates the validity of each bit for particular applications. This type of bit mapped parameter contains one validity mask byte for each status byte representing data. Since the validity mask is part of the parameter definition, a separate scalingByteExtension is not required.	U	BMRWM
0x4	**BinaryCodedDecimal** Conventional Binary Coded Decimal encoding is used to represent two numeric digits per byte. The upper nibble is used to represent the most significant digit (0 - 9), and the lower nibble the least significant digit (0 -9).	U	BCD
0x5	**stateEncodedVariable (1 byte)** This encoding uses a binary weighting scheme to represent up to 256 distinct states. An example is a parameter, which represents the status of the Ignition Switch. Codes "00", "01", "02" and "03" may indicate ignition off, locked, run, and start, respectively. The representation is always limited to one byte.	U	SEV
0x6	**ASCII (1 to 15 bytes for each scalingByte)** Conventional ASCII encoding is used to represent up to 128 standard characters with the MSB = logic '0'. An additional 128 custom characters may be represented with the MSB = logic '1'.	U	ASCII
0x7	**signedFloatingPoint** Floating point encoding is used for data that needs to be represented in floating point or scientific notation. Standard IEEE formats shall be used according to ANSI/IEEE Std 754-1985.	U	SFP

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table C.2 — *(continued)*

Encoding of High Nibble	Description of Data Type	Cvt	Mnemonic
0x8	**packet** Packets contain multiple data values, usually related, each with unique scaling. Scaling information is not included for the individual values. See C.3.1.	U	P
0x9	**formula** A formula is used to calculate a value from the raw data. Formula Identifiers are specified in the table defining the formulaIdentifier encoding. See C.3.2.	U	F
0xA	**unit/format** The units and formats are used to present the data in a more user-friendly format. Unit and Format Identifiers are specified in the table defining the formulaIdentifier encoding. NOTE If combined units and/or formats are used, e.g. mV, then one scalingByte (and scalingData) for each unit/format shall be included in the readScalingDataByIdentifier postive response. See C.3.3.	U	U
0xB	**stateAndConnectionType (1 byte)** This encoding is used especially for input and output signals. The information encoded in the data byte specifies the high level physical layout, electrical levels and functional state. It is recommended to use this option for digital input and output parameters. See C.3.4.	U	SACT
0xC – 0xF	**ISOSAEReserved** Reserved by this document for future definition.	M	ISOSAERESRVD

Table C.3 defines the scalingByte (Low Nibble) parameter.

Table C.3 — scalingByte (Low Nibble) parameter definition

Encoding of Low Nibble	Description of Data Type	Cvt	Mnemonic
0x0 – 0xF	**numberOfBytesOfParameter** This range of values specifies the number of data bytes in a data stream referenced by a parameter identifier. The length of a parameter is defined by the scaling byte(s), which is always preceded by a parameter identifier (one or multiple bytes). If multiple scaling bytes follow a parameter identifier the length of the data referenced by the parameter identifier is the summation of the content of the low nibbles in the scaling bytes. e.g. VIN is identified by a single byte parameter identifier and followed by two scaling bytes. The length is calculated up to 17 data bytes. The content of the two low nibbles may have any combination of values that add up to 17 data bytes. NOTE For the scalingByte with high nibble encoded as formula or unit/format this value is 0x0.	U	NROBOP

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

C.3 scalingByteExtension parameter definitions

C.3.1 scalingByteExtension for scalingByte high nibble of bitMappedReportedWithOutMask

The parameter scalingByteExtension (SBYE) is only supported for scalingByte parameters with the high nibble encoded as formula, unit/format, or bitMappedReportedWithOutMask.

A scalingByte with high nibble encoded as bitMappedReportedWithOutMask shall be followed by scalingByteExtension bytes representing the validity mask for the bit mapped dataIdentifier. Each byte shall indicate which bits of the corresponding dataIdentifier byte are supported for the current a pplication.

Table C.4 defines the scalingByteExtension for bitMappedReportedWithOutMask.

Table C.4 — scalingByteExtension for bitMappedReportedWithOutMask

Byte Value	Description	Cvt
#1	dataIdentifier dataRecord#1 validity mask	M
:	:	C1
#p	dataIdentifier dataRecord#p validity mask	C1
C1: The presence of this parameter depends on the size of the dataIdentifier the information is being requested for. The validity mask shall have as many bytes as the dataIdentifier has dataRecords.		

C.3.2 scalingByteExtension for scalingByte high nibble of formula

The parameter scalingByteExtension (SBYE) is only supported for scalingByte parameters with the high nibble encoded as formula, unit/format, or bitMappedReportedWithOutMask.

A scalingByte with high nibble encoded as formula shall be followed by scalingByteExtension bytes defining the formula. The scalingByteExtension consists a of one byte formulaIdentifier and constants as described in the table below.

Table C.5 defines the scalingByteExtension Bytes for formula.

Table C.5 — scalingByteExtension Bytes for formula

Byte Value	Description	Cvt
#1	formulaIdentifier (refer to table defining the formulaIdentifier encoding for details)	M
#2	C0 high byte	M
#3	C0 low byte	M
#4	C1 high byte	U
#5	C1 low byte	U
:	:	U
#2n+2	Cn high byte	U
#2n+3	Cn low byte	U

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table C.6 defines the formulaIdentifier encoding.

Table C.6 — formulaIdentifier encoding

Byte Value	Description	Cvt
0x00	y = C0 * x + C1	U
0x01	y = C0 * (x + C1)	U
0x02	y = C0 / (x + C1) + C2	U
0x03	y = x / C0 + C1	U
0x04	y = (x + C0) / C1	U
0x05	y = (x + C0) / C1 + C2	U
0x06	y = C0 * x	U
0x07	y = x / C0	U
0x08	y = x + C0	U
0x09	y = x * C0 / C1	U
0x0A – 0x7F	ISO/SAE reserved	M
0x80 – 0xFF	Vehicle manufacturer specific	U

Formulas are defined using variables (y, x, etc.) and constants (C0, C1, C2, etc.). The variable y is the calculated value. The other variables, in consecutive order, are part of the data stream referenced by a dataIdentifier. Each constant is expressed as a two byte real number defined in Table C.7. The two byte real numbers (C = M * 10E) contain a 12 bit signed (2's complement) mantissa (M) and a 4 bit signed (2's complement) exponent (E). The mantissa can hold values within the range –2 048 to +2 047, and the exponent can scale the number by 10^{-8} to 10^{7}. The exponent is encoded in the high nibble of the high byte of the two byte real number. The mantissa is encoded in the low nibble of the high byte and the complete low byte of the two byte real number.

Table C.7 — Two byte real number format

High Byte								Low Byte							
High Nibble				Low Nibble				High Nibble				Low Nibble			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Exponent				Mantissa											

C.3.3 scalingByteExtension for scalingByte high nibble of unit / format

The parameter scalingByteExtension (SBYE) is only supported for scalingByte parameters with the high nibble encoded as formula, unit/format, or bitMappedReportedWithOutMask.

A scalingByte with high nibble encoded as unit / format shall be followed by a single scalingByteExtension byte defining the unit / format. The one byte scalingByteExtension is defined in Table C.8. If combined units and/or formats are used, e.g. mV, then one scalingByte (and scalingByteExtension) shall be included for each unit / format.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table C.8 — Unit / format scalingByteExtension encoding

ScalingByteExtension Byte#1	Name	Symbol	Description	Cvt
0x00	No unit, no prefix	---	---	U
0x01	Meter	m	Length	U
0x02	Foot	ft	Length	U
0x03	Inch	in	Length	U
0x04	Yard	yd	Length	U
0x05	mile (English)	mi	length	U
0x06	Gram	g	mass	U
0x07	ton (metric)	t	mass	U
0x08	Second	s	time	U
0x09	Minute	min	time	U
0x0A	Hour	h	time	U
0x0B	Day	d	time	U
0x0C	year	y	time	U
0x0D	ampere	A	current	U
0x0E	volt	V	voltage	U
0x0F	coulomb	C	electric charge	U
0x10	ohm	W	resistance	U
0x11	farad	F	capacitance	U
0x12	henry	H	inductance	U
0x13	siemens	S	electric conductance	U
0x14	weber	Wb	magnetic flux	U
0x15	tesla	T	magnetic flux density	U
0x16	kelvin	K	thermodynamic temperature	U
0x17	Celsius	°C	thermodynamic temperature	U
0x18	Fahrenheit	°F	thermodynamic temperature	U
0x19	candela	cd	luminous intensity	U
0x1A	radian	rad	plane angle	U
0x1B	degree	°	plane angle	U
0x1C	hertz	Hz	frequency	U
0x1D	joule	J	energy	U
0x1E	Newton	N	force	U
0x1F	kilopond	kp	force	U
0x20	pound force	lbf	force	U
0x21	watt	W	power	U
0x22	horse power (metric)	hk	power	U
0x23	horse power (UK and US)	hp	power	U

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table C.8 — *(continued)*

ScalingByteExtension Byte#1	Name	Symbol	Description	Cvt
0x24	Pascal	Pa	pressure	U
0x25	bar	bar	pressure	U
0x26	atmosphere	atm	pressure	U
0x27	pound force per square inch	psi	pressure	U
0x28	becqerel	Bq	radioactivity	U
0x29	lumen	lm	light flux	U
0x2A	lux	lx	illuminance	U
0x2B	liter	l	volume	U
0x2C	gallon (British)	---	volume	U
0x2D	gallon (US liq)	---	volume	U
0x2E	cubic inch	cu in	volume	U
0x2F	meter per second	m/s	speed	U
0x30	kilometer per hour	km/h	speed	U
0x31	mile per hour	mph	speed	U
0x32	revolutions per second	rps	angular velocity	U
0x33	revolutions per minute	rpm	angular velocity	U
0x34	counts	---	---	U
0x35	percent	%	---	U
0x36	milligram per stroke	mg/stroke	mass per engine stroke	U
0x37	meter per square second	m/sP^{2}P	acceleration	U
0x38	Newton meter	Nm	moment (e.g. torsion moment)	U
0x39	liter per minute	l/min	flow	U
0x3A	Watt per square meter W/m2 Intensity	W/mP^2	Intensity	U
0x3B	Bar per second	bar/s	Pressure change	U
0x3C	Radians per second	rad/s	Angular velocity	U
0x3D	Radians per square second	rad/sP^{2}P	Angular acceleration	U
0x3E	Kilogram per square meter	kg/mP^{2}P	---	U
0x3F	---	---	Reserved by document	M
0x40	exa (prefix)	E	1018	U
0x41	peta (prefix)	P	1015	U
0x42	tera (prefix)	T	1012	U
0x43	giga (prefix)	G	109	U
0x44	mega (prefix)	M	106	U
0x45	kilo (prefix)	k	103	U
0x46	hecto (prefix)	h	102	U
0x47	deca (prefix)	da	10	U

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table C.8 — *(continued)*

ScalingByteExtension Byte#1	Name	Symbol	Description	Cvt
0x48	deci (prefix)	d	10-1	U
0x49	centi (prefix)	c	10-2	U
0x4A	milli (prefix)	m	10-3	U
0x4B	micro (prefix)	m	10-6	U
0x4C	nano (prefix)	n	10-9	U
0x4D	pico (prefix)	p	10-12	U
0x4E	femto (prefix)	f	10-15	U
0x4F	atto (prefix)	a	10-18	U
0x50	Date1	-	Year-Month-Day	U
0x51	Date2	-	Day/Month/Year	U
0x52	Date3	-	Month/Day/Year	U
0x53	week	W	calendar week	U
0x54	Time1	---	UTC Hour/Minute/Second	U
0x55	Time2	---	Hour/Minute/Second	U
0x56	DateAndTime1	---	Second/Minute/Hour/Day/Month/Year	U
0x57	DateAndTime2	---	Second/Minute/Hour/Day/Month/Year/Local minute offset/Local hour offset	U
0x58	DateAndTime3	---	Second/Minute/Hour/Month/Day/Year	U
0x59	DateAndTime4	---	Second/Minute/Hour/Month/Day/Year/Local minute offset/Local hour offset	U
0x5A – 0xFF	---	---	ISO/SAE reserved	M

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

C.3.4 scalingByteExtension for scalingByte high nibble of stateAndConnectionType

A scalingByte with high nibble encoded as stateAndConnectionType shall be followed by a single scalingByteExtension byte defining the stateAndConnectionType. The one byte scalingByteExtension is defined in Table C.9. The stateAndConnectionType encoding is used specially for input and output signals. Encoded in the scalingByteExtension data byte is information about the physical layout, electrical levels and functional state.

Table C.9 — Encoding of scalingByte High Nibble of stateAndConnectionType

Encoding of bits	Value	Used with input signals	Used with output signals
0x0 – 0x2	0	State: Not Active	State: Not Activated
	1	State: Active, function 1	State: Active, function 1
	2	State: Error detected	State: Plausibility error detected
	3	State: Not available	State: Not available
	4	State: Active, function 2 (only in combination with 3 states)	State: Active, function 2 (only in combination with 3 states)
	5 – 7	Reserved	Reserved
0x3 – 0x4	0	Signal at low level (ground)	Signal at low level (ground)
	1	Signal at middle level (between ground and +)	Signal at middle level (between ground and +)
	2	Signal at high level (+)	Signal at high level (+)
	3	Reserved by document	Reserved by document
0x5	0	Input signal	Not defined
	1	Not defined	Output signal
0x6 – 0x7	0	Internal signal or via CAN not exclusively available in ECU connector	Internal signal or via CAN no exclusively available in ECU connector
	1	Pull-down resistor input type (2 states)	Low side switch (2 states)
	2	Pull-up resistor input type (2 states)	High side switch (2 states)
	3	Pull-up and pull-down resistor input type (3 states)	Low side and high side switch (3 states)

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

C.4 transmissionMode parameter definitions

Table C.10 defines the transmissionMode parameter.

Table C.10 — transmissionMode parameter definitions

Byte Value	Description	Cvt	Mnemonic
0x00	**ISOSAEReserved** This value shall be reserved by this document for future definition.	M	ISOSAERESRVD
0x01	**sendAtSlowRate** This parameter specifies that the server shall transmit the requested dataRecord information at a slow rate in response to the request message (where the # of responses to be sent = maximumNumberOfResponsesToSend). The repetition rate specified by the transmissionMode parameter slow is vehicle manufacturer specific, and pre-defined in the server.	U	SASR
0x02	**sendAtMediumRate** This parameter specifies that the server shall transmit the requested dataRecord information at a medium rate in response to the request message (where the # of responses to be sent = maximumNumberOfResponsesToSend). The repetition rate specified by the transmissionMode parameter medium is vehicle manufacturer specific, and pre-defined in the server.	U	SAMR
0x03	**sendAtFastRate** This parameter specifies that the server shall transmit the requested dataRecord information at a fast rate in response to the request message (where the # of responses to be sent = maximumNumberOfResponsesToSend). The repetition rate specified by the transmissionMode parameter fast is vehicle manufacturer specific, and pre-defined in the server.	U	SAFR
0x04	**stopSending** The server stops transmitting positive response messages send periodically/repeatedly. Note that maximumNumberOfResponsesToSend parameter should be set to 0x01 if transmissionMode = stopSending (otherwise, server operation could be undefined).	M	SS
0x05 – 0xFF	**ISOSAEReserved** This value shall be reserved by this document for future definition.	M	ISOSAERESRVD

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

C.5 Coding of UDS version number

Table C.11 defines the coding of UDS version number DID 0xFF00 – 4 bytes unsigned value. The specification release version of this document is: 2.0.0.0.

Table C.11 — Coding of UDS version number DID 0xFF00 – 4 bytes unsigned value

Byte 1 (MSB)	Byte 2	Byte 3	Byte 4 (LSB)
Major (0..255)	Minor (0..255)	Revision (0..255)	0

Table C.12 defines two examples for V1.0.0.0 and V2.0.0.0.

Table C.12 — DID 0xFF00 values for 1st and 2nd edition of ISO 14229-1

Byte 1 (MSB)	Byte 2	Byte 3	Byte 4 (LSB)
Version 1.0.0.0			
1	0	0	0
Version 2.0.0.0			
2	0	0	0

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Annex D
(normative)

Stored data transmission functional unit data-parameter definitions

D.1 groupOfDTC parameter definition

Table D.1 provides group of DTC definitions.

Table D.1 — Definition of groupOfDTC and range of DTC numbers

Byte Value	Description	Cvt	Mnemonic
0x000000 – 0x0000FF	This range of values is reserved for future legislative requirements.	M	RFLU
to be determined by vehicle manufacturer	Powertrain Group: engine and transmission	U	PG
	Powertrain DTCs	U	PDTC_
	Chassis Group	U	CG
	Chassis DTCs	U	CDTC_
	Body Group	U	BG
	Body DTCs	U	BDTC_
	Network Communication Group	U	NCG
	Network Communication DTCs	U	NCDTC_
0xFFFF00 – 0xFFFFFE	The lower byte shall always be the FunctionalGroupIdentifier as defined in Table D.15. For example, a value of 0xFFFF33 shall equal the Emissions Group and a value of 0xFFFFD0 shall equal the Safety Group.	M	ISOSAERESRVD
0xFFFFFF	All Groups (all DTCs)	M	AG

D.2 DTCStatusMask and statusOfDTC bit definitions

D.2.1 Convention and definition

This subclause defines the mapping of the DTCStatusMask / statusOfDTC parameters used with the ReadDTCInformation service. Every server shall adhere to the convention for storing bit-packed DTC status information as defined in the table below. Actual usage of the bit-fields shall be defined in the implementation standards.

The status of the TestFailed bit shall not directly be linked to the failsafe behaviour associated with the monitor status. That means for triggering of the failsafe behaviour which is associated with the status of a certain monitor a separate set of status bits needs to be maintained. The vehicle manufacturer shall define if and how any synchronization mechanism between DTC status and failsafe relevant monitor status is applied and implemented.

The following is a list of definitions used for the description of the DTC status bit definitions.

— **Test:** A test is an on-board diagnostic software algorithm that determines the malfunction status of a component or system typically within a single operation cycle. Some tests run only once during an operation cycle. Other tests can run every program loop, sampling as often as every few milliseconds. The end result of a test represents a completely matured / qualified condition (i.e., passed or failed). That

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

means a test which needs a failing condition over a specific time or evaluation of additional plausibility checks before a component is considered to be failing will return a “Failed” condition only after all maturation criteria have been fulfilled. Each DTC is associated with a test representing a detectable fault symptom.

— **Test Sample:** A test sample represents the 'pass' or 'fail' result from a single instance of a DTC test execution when the test run criteria are met. This represents a single sample and therefore not generally a fully matured / qualified condition. For an ECU supporting the DTC Fault Detection Counter a test sample representing a fail will increase the DTC Fault Detection Counter by a specific amount and a test sample representing a pass will decrease the DTC Fault Detection Counter by a specific amount.

— **Complete:** Complete is an indication that a test was able to determine whether a malfunction exists or does not exist for the current operation cycle (complete does not imply failed).

— **Test results:** While a test runs or after it has completed it may indicate one the following results to the internal failure handler:

  — **PreFailed:** This status may be used by tests in ECUs to indicate that the test is currently maturing a failure condition. One use case for this information is in manufacturing to speed up failure detection for optimised workflow while maintaining fault tolerance in the field.

  — **Failed:** This status is available after a test has run to its completion and indicates a completely matured failing condition.

  — **Passed:** This status is available after a test has run to its completion and indicates that the system or component is not failing.

— **Failure:** A failure is the inability of a component or system to meet its intended function. A failure has occurred when fault conditions have been detected for a sufficient period of time, implying that a test returned a “Failed” result. The terms "failure" and "malfunction" are interchangeable.

— **Monitor:** A monitor consists of one or more tests used to determine the proper functioning of a component or system.

— **Monitoring cycle:** A monitoring cycle is the time in which a monitor runs to its completeness. This is a manufacturer defined set of conditions during which the tests of a monitor can run. A monitoring cycle may be executed several times during an operation cycle or once over several operation cycles.

— **Operation Cycle:** An operation cycle defines the start and end conditions for monitors to run. During an operation cycle several monitoring cycles may have completed (regardless of their test results). An ECU may support several operation cycles. For body and chassis ECUs it is up to the manufacturer to define an operation cycle (e.g. time between powering up and powering down the ECU or between ignition on and ignition off). For powertrain ECUs, there are additional criteria defining an operation cycle. Emissions-related powertrain ECUs use an engine-running or engine-off time period to define an operation cycle which is referred to as driving cycle. If a reset condition for a DTC status bit is associated with the beginning of the operation cycle, it might also be considered the end of the previous cycle (i.e., it is not always possible to distinguish the beginning versus the end of each operation cycle).

NOTE For emissions-related monitors the criteria for the beginning and the end of an operation cycle are defined by legislation.

— **Pending:** The pending status of a failure is defined as a test having reported a “Failed” result for this test during the current operation cycle or during the last completed operation cycle. Once the test has reported a “Passed” condition for a complete operation cycle of this failure the pending status is reset.

— **Confirmation Threshold:** The confirmed status of a failure is defined as a test having reported 'Failed' for this test for a given number of operation cycles where the test has run to completion. Typically for non-OBD use cases the threshold for operation cycles is defined as one. For OBD use cases this threshold is typically greater than one. Implementations may use a Trip Counter (see Figure D.9) as a trigger for

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

 © ISO 2013 – All rights reserved

changing the confirmed status from 0 to 1. The Trip Counter counts the number of operation cycles (driving cycles) where a malfunction occurred. If the counter reaches the threshold (e.g., 2 driving cycles) the confirmed bit changes from 0 to 1.

— **Aging Threshold:** The aging of a DTC is defined as a test having reported no 'Failed' result for a given number of vehicle manufacturer or regulation defined operation cylces and it is vehicle manufacturer specific if the respective cycle triggers incrementing the aging counter depending on whether the test has run to completion or not during the cycle. Implementations may use an aging counter see Figure D.11) as a trigger for changing the confirmed status from 1 to 0 and erasing the DTC information from non-volatile memory. The aging counter counts the number of cycles (e.g., warm-up cycles) meeting the previously mentioned criteria. If this counter reaches the threshold (e.g., 40 warm-up cycles) the confirmed bit changes from 1 to 0.

— **Driving cycle:** A specific type of operation cycle used for emissions-related ECUs. Refer to “OperationCycle” for further details. In emissions-related ECUs only one operation cycle shall be supported, which is identical to the driving cycle as defined by legislation.

— **Monitor Level Enable Conditions:** The criteria / conditions for when a monitor is allowed to run and report a test result.

— **DTC Status Update Condition:** A condition where all DTC status bits are allowed to be updated by the monitor (e.g., controlDTCSetting DTCSettingType does not equal 'off"). This generic condition applies to all DTC status bit transitions (i.e., if this condition is false none of the transitions depicted in Figure D.1 – Figure D.8 shall be allowed except reset of the status bits triggered by the reception of a clearDiagnosticInformation command (see 9.9.1 and 11.2.1)).

— **DTC Storage Condition:** A condition defined by the vehicle manufacturer indicating whether the relevant DTC status bits and the related DTC data (e.g., DTC Extended or Snapshot data) that is capable of being updated is updated and stored in non-volatile memory.

D.2.2 Pseudocode data dictionary

The pseudocode data dictionary defines variables used in the pseudocode definition for each statusOfDTC bit.

Table D.2 defines the Pseudocode data dictionary.

Table D.2 — Pseudocode data dictionary

Variable	Description
initializationFlag_TF initializationFlag_TFTOC initializationFlag_PDTC initializationFlag_CDTC initializationFlag_TNCSLC initializationFlag_TFSLC initializationFlag_TNCTOC initializationFlag_WIR	Flags used within the following pseudocode to ensure that the DTC status bit initialization operations are only performed once. At a minimum, it is expected that the flags are defaulted to a value of FALSE prior to the first power-up of the ECU. The variables shall remain latched at TRUE until ECU software is reset or any other such vehicle manufacturer specific reset is performed. FALSE = initialization not performed TRUE = initialization performed
lastOperationCycle	Storage variable used to record the most recently completed operation cycle. A value shall be assigned to the variable during the respective initialization phase of operation given in the following pseudocode.
currentOperationCycle	Storage variable used to record the current operation cycle. Updated continuously outside the scope of the DTC status bit logic.
failedOperationCycle	Storage variable used to record the most recently failed operation cycle. A value shall be assigned to the variable during the respective initialization phase of operation given in the following pseudocode.
confirmStage	Storage variable used to record the stage of operation of the confirmedDTC status bit pseudocode.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

D.2.3 DTC status bit definitions

Table D.3 defines the DTC status bit '0' testFailed.

Table D.3 — DTC status bit 0 testFailed definitions

Bit	Description	Cvt	Mnemonic
0	**testFailed**	U	TF
	This bit shall indicate the result of the most recently performed test. A logical ‘1’ shall indicate that the last test failed meaning that the failure is completely matured. Reset to logical '0' if the result of the most recently performed test returns a “pass” result meaning that all de-mature criteria have been fulfilled. Additional reset conditions may be defined by the vehicle manufacturer / implementation		
	Bit state after a successfull ClearDiagnosticInformation service	**logical ‘0’**	
	Reset to logical '0' if a call has been made to ClearDiagnosticInformation.		
	Bit state definition: '0' = most recent result from DTC test indicated no failure detected. '1' = most recent result from DTC test indicated a matured failing result.		

```
#   Pseudocode Operation
1   IF (initializationFlag_TF == FALSE)
2      Set initializationFlag_TF = TRUE
3      Set testFailed = 0
4   IF ((most recent test result == PASSED) OR
        (ClearDiagnosticInformation requested == TRUE) OR
        (vehicle manufacturer/implementation reset conditions satisfied))
5      Set testFailed = 0
6   ELSE IF (most recent test result == FAILED)
7      Set testFailed = 1
```

Figure D.1 defines the DTC status bit '0' testFailed logic.

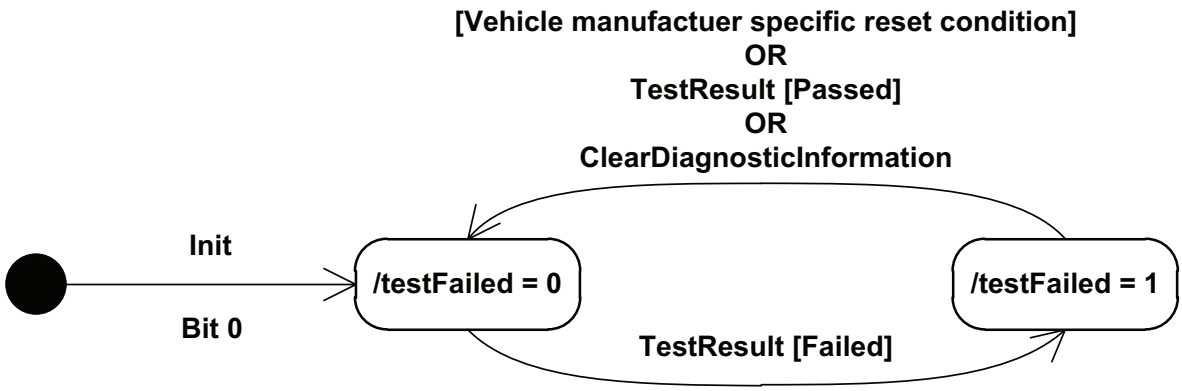


Figure D.1 — DTC status bit 0 testFailed logic

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table D.4 defines the DTC status bit '1' testFailedThisOperationCycle.

Table D.4 — DTC status bit 1 testFailedThisOperationCycle definitions

Bit	Description	Cvt	Mnemonic
1	**testFailedThisOperationCycle**	U	TFTOC
	This bit shall indicate whether or not a diagnostic test has reported a testFailed result at any time during the current operation cycle (or that a testFailed result has been reported during the current operation cycle and after the last time a call was made to ClearDiagnosticInformation). Reset to logical '0' when a new operation cycle is initiated or after a call to ClearDiagnosticInformation. If this bit is set to logical '1', it shall remain a '1' until a new operation cycle is started.		
	Bit state after a successful ClearDiagnosticInformation service	**logical '0'**	
	Reset to a logical '0' after a call to ClearDiagnosticInformation.		
	Bit state definition: '0' = testFailed: result has not been reported during the current operation cycle or after a call was made to ClearDiagnosticInformation during the current operation cycle. '1' = testFailed: result was reported at least once during the current operation cycle.		

```
#   Pseudocode Operation
1   IF (initializationFlag_TFTOC == FALSE)
2      Set initializationFlag_TFTOC = TRUE
3      Set testFailedThisOperationCycle = 0
4      Set lastOperationCycle = currentOperationCycle
5   IF ((currentOperationCycle != lastOperationCycle) OR
           (ClearDiagnosticInformation requested == TRUE))
6      Set lastOperationCycle = currentOperationCycle
7      Set testFailedThisOperationCycle = 0
8   ELSE IF (most recent test result == FAILED)
9      Set testFailedThisOperationCycle = 1
```

Figure D.2 defines the DTC status bit '1' testFailedThisOperationCycle logic.

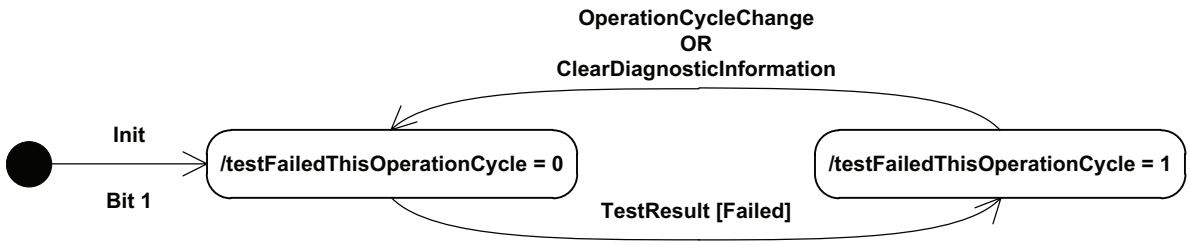


Figure D.2 — DTC status bit 1 testFailedThisOperationCycle logic

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table D.5 defines the DTC status bit '2' pendingDTC.

Table D.5 — DTC status bit 2 pendingDTC definitions

Bit	Description	Cvt	Mnemonic
2	**pendingDTC**	U	PDTC
	This bit shall indicate whether or not a diagnostic test has reported a testFailed result at any time during the current or last completed operation cycle. The status shall only be updated if the test runs and completes. The criteria to set the pendingDTC bit and the TestFailedThisOperationCycle bit are the same. The difference is that the testFailedThisOperationCycle is cleared at the beginning of each operation cycle and the pendingDTC bit is not cleared until an operation cycle has completed where the test has passed at least once and never failed. If the test did not complete during the current operation cycle, the status bit shall not be changed. For example, if a monitor stops running after a confirmed DTC is set, the pendingDTC must remain set = '1'. For an OBD DTC, a pending DTC is required to be stored after a malfunction is detected during the first driving cycle.		
	Bit state after a successful ClearDiagnosticInformation service	**logical '0'**	
	Reset to a logical '0' after a call to ClearDiagnosticInformation.		
	Bit state definition: '0' = This bit shall be set to 0 after completing an operation cycle during which the test completed and a malfunction was not detected or upon a call to the ClearDiagnosticInformation service. '1' = This bit shall be set to 1 and latched if a malfunction is detected during the current operation cycle.		

```
#   Pseudocode Operation
1   IF (initializationFlag_PDTC == FALSE)
2      Set initializationFlag_PDTC = TRUE
3      Set pendingDTC = 0
4   IF (ClearDiagnosticInformation requested == TRUE)
5      Set pendingDTC = 0
6   ELSE IF (most recent test result == FAILED)
7      Set pendingDTC = 1
8   ELSE IF ((currentOperationCycle == stop) AND
            (testNotCompletedThisOperationCycle == 0) AND
            (testFailedThisOperationCycle == 0))
9      Set pendingDTC = 0
```

Figure D.3 defines the DTC status bit '2' pendingDTC logic.

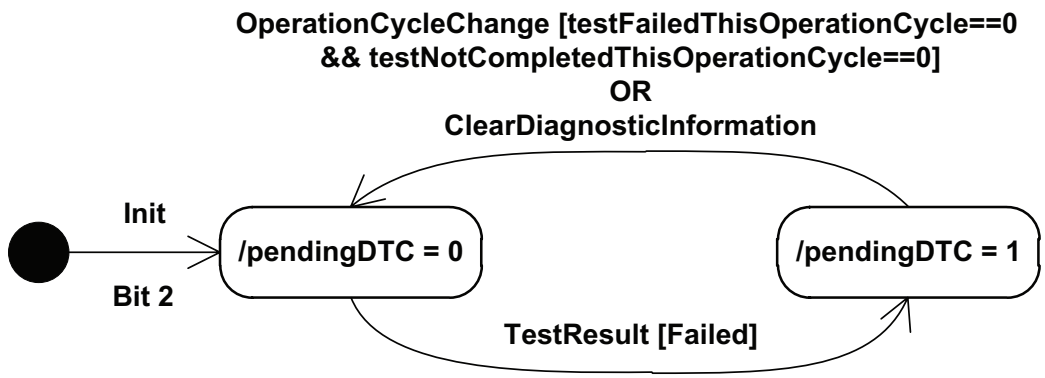


Figure D.3 — DTC status bit 2 pendingDTC logic

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table D.6 defines the DTC status bit '3' confirmedDTC.

Table D.6 — DTC status bit 3 confirmedDTC definitions

Bit	Description	Cvt	Mnemonic
3	**confirmedDTC**	M	CDTC
	This bit shall indicate whether a malfunction was detected enough times to warrant that the DTC is desired to be stored in long-term memory. A confirmedDTC does not always indicate that the malfunction is present at the time of the request. (testFailed can be used to determine if a malfunction is present at the time of the request). Reset to logical '0' after a call to ClearDiagnosticInformation or after aging threshold has been satisfied (e.g., 40 engine warm-ups without another detected malfunction). Furthermore this bit is reset when the fault record associated with this DTC is overwritten by a newer DTC based upon vehicle manufacturer specific fault memory overflow requirements. DTC confirmation threshold and aging threshold are defined by the vehicle manufacturer or mandated by On Board Diagnostic regulations.		
	Bit state after a successfull ClearDiagnosticInformation service	**logical ‘0’**	
	Reset to a logical ‘0’ after a call to ClearDiagnosticInformation.		
	Bit state definition '0' = DTC has never been confirmed since the last call to ClearDiagnosticInformation or after the aging criteria have been satisfied for the DTC (or DTC has been erased due to fault memory overflow). '1' = DTC confirmed at least once since the last call to ClearDiagnosticInformation and aging criteria have not yet been satisfied.		

```
#   Pseudocode Operation
1   IF (initializationFlag_CDTC == FALSE)
2      Set initializationFlag_CDTC = TRUE
3      Set confirmedDTC = 0
4      Set confirmStage = INITIAL_MONITOR
5   IF (confirmStage == INITIAL_MONITOR)
6      IF (confirmation threshold == TRUE)
7         Set confirmedDTC = 1
8         Reset aging status
9         Set confirmStage = AGING_MONITOR
10     ELSE
11        Set confirmedDTC = 0
12  IF (confirmStage == AGING_MONITOR)
13     IF ((ClearDiagnosticInformation requested == TRUE) OR
           (aging threshold satisfied == TRUE))
14        Set confirmedDTC = 0
15        Set confirmStage = INITIAL_MONITOR
16     ELSE IF (most recent test result == FAILED)
17        Reset aging status
18     ELSE
19        Update aging status as appropriate
```

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Figure D.4 defines the DTC status bit '3' confirmedDTC logic.

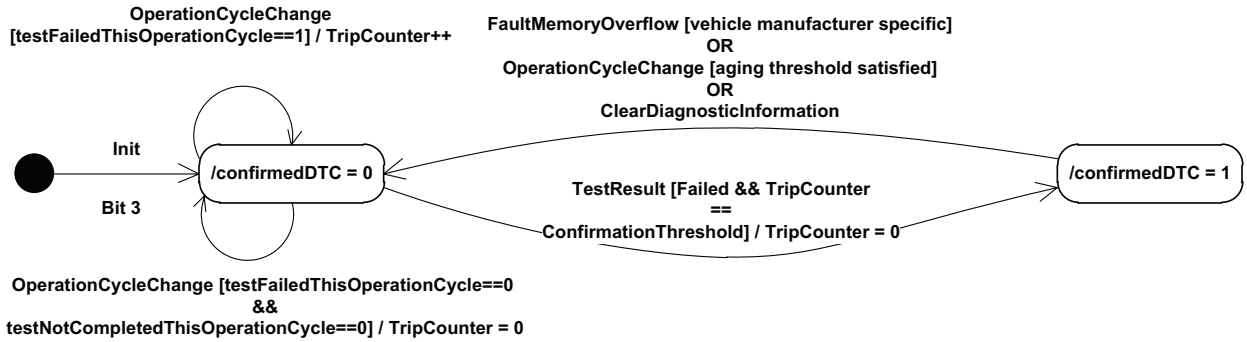


Figure D.4 — DTC status bit 3 confirmedDTC logic

Table D.7 defines the DTC status bit '4' testNotCompletedSinceLastClear.

Table D.7 — DTC status bit 4 testNotCompletedSinceLastClear definitions

Bit	Description	Cvt	Mnemonic
4	**testNotCompletedSinceLastClear**	U	TNCSLC
	This bit shall indicate whether a DTC test has ever run and completed since the last time a call was made to ClearDiagnosticInformation. One ('1') shall indicate that the DTC test has not run to completion. If the test runs and passes or if the test runs and fails (e.g. testFailedThisOperationCycle = '1') then the bit shall be set to a '0' (and latched).		
	Bit state after a successfull ClearDiagnosticInformation service	**logical '1'**	
	Reset to a logical '1' after a call to ClearDiagnosticInformation.		
	Bit state definition '0' = DTC test has returned either a passed or failed test result at least one time since the last time diagnostic information was cleared. '1' = DTC test has not run to completion since the last time diagnostic information was cleared.		

```
#   Pseudocode Operation
1   IF (initializationFlag_TNCSLC == FALSE)
2      Set initializationFlag_TNCSLC = TRUE
3      Set testNotCompletedSinceLastClear = 1
4   IF (ClearDiagnosticInformation requested = TRUE)
5      Set testNotCompletedSinceLastClear = 1
6   ELSE IF ((most recent test result = PASSED) OR (most recent test result =
       FAILED))
7      Set testNotCompletedSinceLastClear = 0
```

Figure D.5 defines the DTC status bit '4' testNotCompletedSinceLastClear logic.

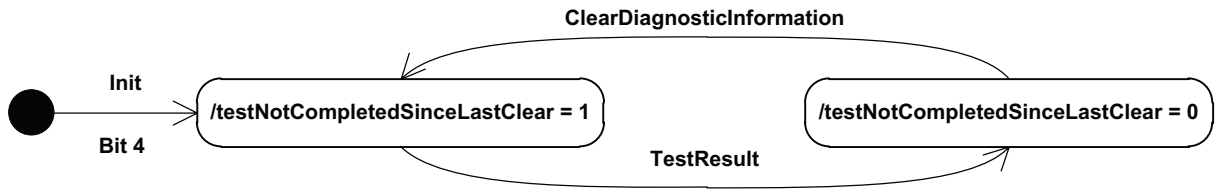


Figure D.5 — DTC status bit 4 testNotCompletedSinceLastClear logic

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table D.8 defines the DTC status bit '5' testFailedSinceLastClear.

Table D.8 — DTC status bit 5 testFailedSinceLastClear definitions

Bit	Description	Cvt	Mnemonic
5	**testFailedSinceLastClear**	U	TFSLC
	This bit shall indicate whether a DTC test has completed with a failed result since the last time a call was made to ClearDiagnosticInformation (i.e., this is a latched testFailedThisOperationCycle = '1'). Zero ('0') shall indicate that the test has not run or that the DTC test ran and passed (but never failed). If the test runs and fails then the bit shall remain latched at a '1'. It is the responsibility of the vehicle manufacturer to specify whether or not this bit is reset by aging-criteria or reset due to an overflow of the fault memory.		
	Bit state after a successfull ClearDiagnosticInformation service	**logical ‘0’**	
	Reset to a logical ‘0’ after a call to ClearDiagnosticInformation.		
	Bit state definition '0' = DTC test has not indicated a failed result since the last time diagnostic information was cleared. It is the responsibility of the vehicle manufacturer if this bit shall also be reset to zero ('0') in case aging threshold is fulfilled or an overflow of the fault memory occurs. '1' = DTC test returned a failed result at least once since the last time diagnostic information was cleared.		

```
#   Pseudocode Operation
1   IF (initializationFlag_TFSLC == FALSE)
2      Set initializationFlag_TFSLC = TRUE
3      Set testFailedSinceLastClear = 0
4   IF (ClearDiagnosticInformation requested == TRUE)
       /* optional:  OR (aging threshold satisfied == TRUE)
       /* optional:  OR (overflow criteria satisfied == TRUE)
5      Set testFailedSinceLastClear = 0
6   ELSE IF (most recent test result == FAILED)
7      Set testFailedSinceLastClear = 1
```

Figure D.6 defines the DTC status bit '5' testFailedSinceLastClear logic.

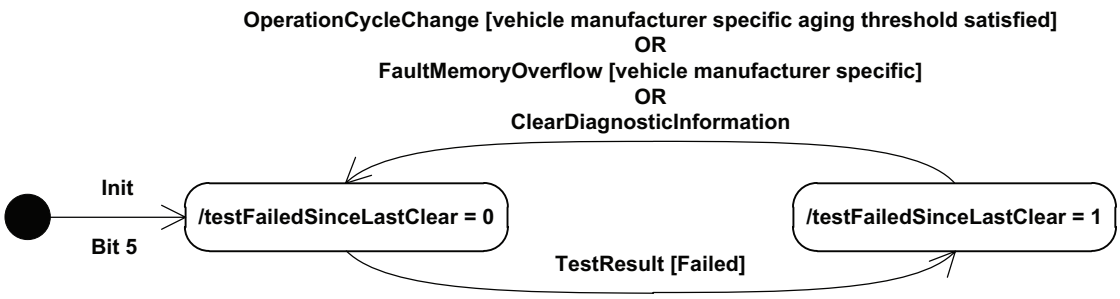


Figure D.6 — DTC status bit 5 testFailedSinceLastClear logic

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table D.9 defines the DTC status bit '6' testNotCompletedThisOperationCycle.

Table D.9 — DTC status bit 6 testNotCompletedThisOperationCycle definitions

Bit	Description	Cvt	Mnemonic
6	**testNotCompletedThisOperationCycle**	U	TNCTOC
	This bit shall indicate whether a DTC test has ever run and completed during the current operation cycle (or completed during the current operation cycle after the last time a call was made to ClearDiagnosticInformation). A logical '1' shall indicate that the DTC test has not run to completion during the current operation cycle. If the test runs and passes or fails then the bit shall be set (and latched) to '0' until a new operation cycle is started.		
	Bit state after a successfull ClearDiagnosticInformation service	**logical '1'**	
	Reset to a logical '1' after a call to ClearDiagnosticInformation.		
	Bit state definition '0' = DTC test has returned either a passed or testFailedThisOperationCycle = '1' result during the current drive cycle (or since the last time diagnostic information was cleared during the current operation cycle). '1' = DTC test has not run to completion this operation cycle (or since the last time diagnostic information was cleared this operation cycle).		

```
#   Pseudocode Operation
1   IF (initializationFlag_TNCTOC == FALSE)
2      Set initializationFlag_TNCTOC = TRUE
3      Set testNotCompletedThisOperationCycle = 1
4      Set lastOperationCycle = currentOperationCycle
5   IF (ClearDiagnosticInformation requested == TRUE)
6      Set testNotCompletedThisOperationCycle = 1
7   ELSE IF (currentOperationCycle != lastOperationCycle)
8      Set lastOperationCycle = currentOperationCycle
9      Set testNotCompletedThisOperationCycle = 1
10  ELSE IF ((most recent test result == PASSED) OR
       (most recent test result == FAILED))
11     Set testNotCompletedThisOperationCycle = 0
```

Figure D.7 defines the DTC status bit '6' testNotCompletedThisOperationCycle logic.

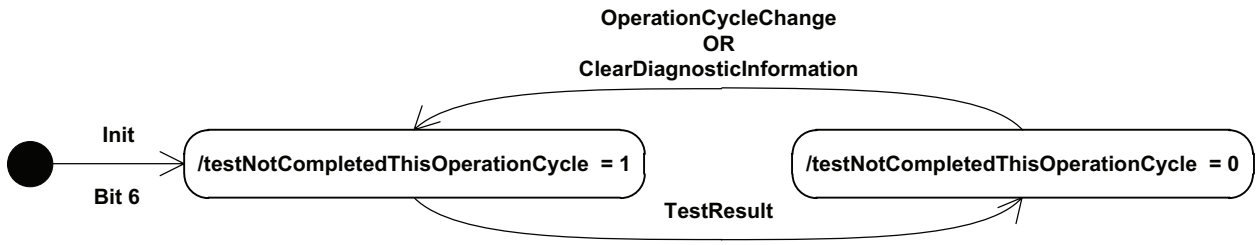


Figure D.7 — DTC status bit 6 testNotCompletedThisOperationCycle logic

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table D.10 defines the DTC status bit '7' WarningIndicator requested.

Table D.10 — DTC status bit 7 WarningIndicator requested definitions

Bit	Description	Cvt	Mnemonic
7	**warningIndicatorRequested**	U	WIR
	This bit shall report the status of any warning indicators associated with a particular DTC. Warning outputs may consist of indicator lamp(s), displayed text information, etc. If no warning indicators exist for a particular DTC, this status shall default to a logic '0' state. Conditions for activating the warning indicator shall be defined by the vehicle manufacturer / implementation, but if the warning indicator is on for a given DTC, then confirmedDTC shall also be set to '1' (with the exception described below).		
	Bit state after a successfull ClearDiagnosticInformation service	**logical ‘0’**	
	Reset to a logical ‘0’ after a call to ClearDiagnosticInformation. Some ECUs may latch the failsafe strategy associated with a particular confirmed fault for the current operation cycle. If the warning indicator is still requested due to this latched failsafe following a call to ClearDiagnosticInformation, this bit shall not be cleared to a logical '0'. Rather, this bit shall remain set to logical '1' until the failsafe strategy is no longer active (e.g., test completes and passes). Additional reset conditions shall be defined by the vehicle manufacturer / implementation.		
	Bit state definition '0' = Server is not requesting warningIndicator to be active '1' = Server is requesting warningIndicator to be active		

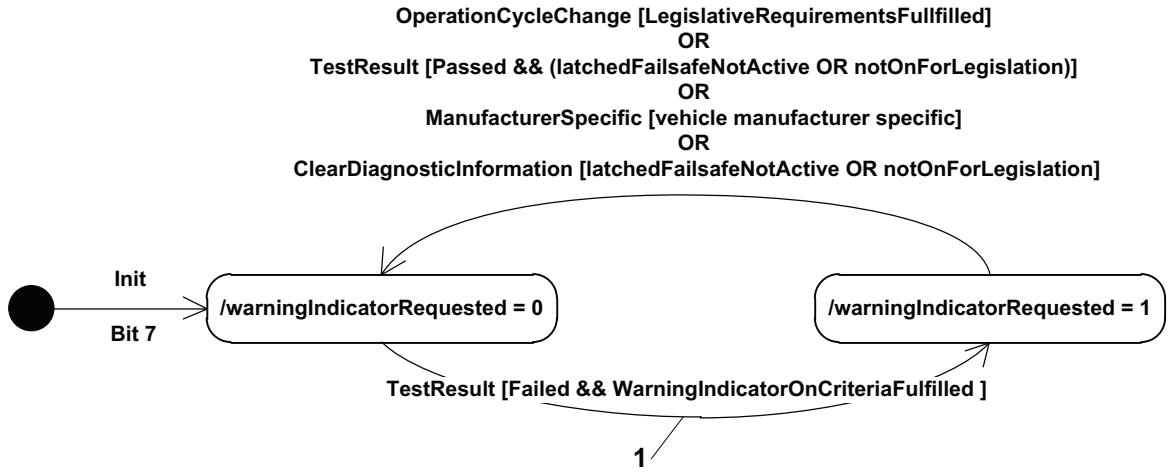
```
#   Pseudocode Operation
1   IF (initializationFlag_WIR == FALSE)
2      Set initializationFlag_WIR = TRUE
3      Set warningIndicatorRequested = 0
4   IF (((ClearDiagnosticInformation requested == TRUE) OR (TestResult == Passed)
       OR (vehicle manufacturer or implementation-specific warning indicator
           disable criteria are satisfied))
       AND ((warning indicator not requested on due to latched failsafe for
           particular DTC) OR (warning indicator not requested on by legislation)))
5      Set warningIndicatorRequested = 0
6   ELSE IF (((TestResult == Failed) AND (warning indicator exists for the
             particular DTC)
       AND ((confirmedDTC == 1)
       OR  (vehicle manufacturer or implementation-specific warning indicator
       enable criteria are satisfied)))
7      Set warningIndicatorRequested = 1
```

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Figure D.8 defines the DTC status bit '7' WarningIndicator requested logic.



Key

1 WarningIndicatorOnCriteriaFulfilled = warning indicator exists for particular DTC **AND** (confirmedDTC = 1 **OR** vehicle manufacturer or implementation-specific warning indicator enable criteria are satisfied)

Figure D.8 — DTC status bit 7 WarningIndicator requested logic

D.2.4 Example for operation of DTC Status Bits

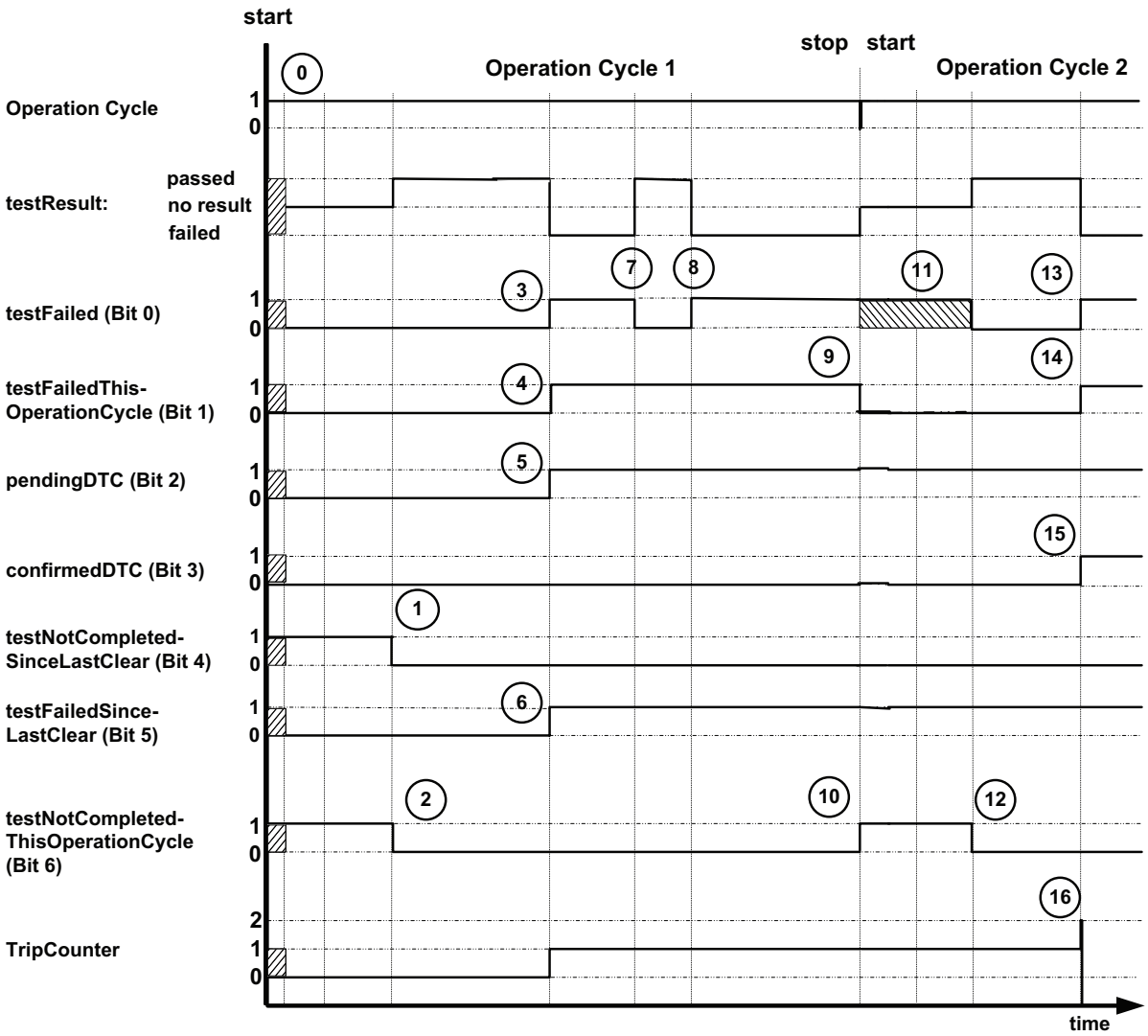
This example provides an overview on the operation of the DTC status bits in a two operation cycle emissions-related OBD DTC. The figure shows the handling for a two operation cycle emissions-related OBD DTC. The handling can also be applied to non-emissions-related OBD DTCs and is shown here for general informational purpose.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Figure D.9 defines an example of a two operation cycle emissions-related OBD DTC.



Key

▨	Undefined bit state
▧	Manufacturer specific bit state
0	ClearDiagnosticInformation received → initialization of DTC status byte
1, 2	the related diagnostic monitor reported a sufficient number of passed test samples fulfilling the DTC pass criteria → testNotCompleted bits (4 and 6) change from 1 to 0, indicating the monitor has run to completion and the DTC readiness has been reached since last clear and for operation cycle 1
3,4,5,6	the related diagnostic monitor reported a sufficient number of failed test samples fulfilling DTC failed criteria → testFailed, testFailedThisMonitoringCycle, pendingDTC and testFailedSinceLastClear bits change from 0 to 1 indicating a malfunction has been detected but the malfunction has not been confirmed over 2 operation cycles
7	the related diagnostic monitor reported a sufficient number of passed test samples fulfilling DTC passed criteria → testFailed bit changes from 1 to 0 indicating the malfunction is currently not active
8	the related diagnostic monitor reported a sufficient number of failed test samples fulfilling DTC failed criteria → testFailed bit changes from 0 to 1 indicating a malfunction has been detected repeatedly in operation cycle 1

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

9, 10 operation cycle 1 ends and operation cycle 2 starts, testFailedthisOperationCycle changes from 1 to 0 and testNotCompleteThisOperationCycle change from 0 to 1; it is manufacturer specific if this reset is executed at the very end of the operation cycle or at the immediately after starting the new cycle

11 After a new operation cycle has started (it is manufacturer specific whether the testFailed status is retained through the transition from operation cycle 1 to operation cycle 2) the related diagnostic monitor reported a sufficient number of passed results fulfilling DTC passed criteria → testFailed bit transistions to 0

12 after a new operation cycle has started the related diagnostic monitor reported a sufficient number of passed test samples fulfilling DTC passed criteria → testNotCompleteThisOperationCycle bit changes from 1 to 0, indicating the monitor has run to completion at least once during the new operation cycle

13, 14 the related diagnostic monitor reported a sufficient number of failed test samples fulfilling DTC failed criteria → testFailed, testFailedThisMonitoringCycle bits change from 0 to 1 indicating a malfunction has been detected during the new operation cycle

15 the confirmedDTC bit changes from 0 to 1 indicating that the related malfunction detected during the last operation cycle is still present

16 TripCounter spikes to '2' at the time DTC status changes to confirmedDTC and then immediately resets to '0' according to Figure D.4.

Figure D.9 — Example of a two operation cycle emissions-related OBD DTC

SHW3SZH 2013-09-29 NormMaster

D.3 DTC severity and class definition

D.3.1 DTC severity and class byte definition

This subclause defines the mapping of the DTCSeverityMask / DTCSeverity parameters used with the ReadDTCInformation service. Every server shall adhere to the convention for storing bit-packed DTC severity information as defined in Table D.11.

The DTCSeverityMask / DTCSeverity byte contains DTC severity and DTC class information. The DTCSeverityMask / DTCSeverity byte is reported in a 1-byte value as defined in Table D.11. The optional upper 3 bits (bit 7-5) of the 1-byte value are used to represent the DTC severity information. If not supported by the server those bits shall be set to "0". The mandatory lower 5 bits (bit 4-0) of the 1-byte value are used to represent the DTC class information.

Table D.11 — DTCSeverityMask / DTCSeverity byte definition

DTCSeverity byte							
Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
DTC severity information (optional)			DTC class information				

D.3.2 DTC severity bit definition

The DTC severity bit definition defines bit states to report the recommended action to be taken by the system (e.g. vehicle) operator. Table D.12 defines DTC severity status bits.

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

 © ISO 2013 – All rights reserved

Table D.12 — DTC severity bit definitions (bit 7-5)

Bit	Description	Cvt	Mnemonic
5	**maintenanceOnly** 0 = no maintenanceOnly severity 1 = maintenanceOnly severity This value indicates that the failure requests maintenance only.	M	MO
6	**checkAtNextHalt** 0 = do not checkAtNextHalt 1 = checkAtNextHalt This value indicates to the failure that a check of the vehicle is required at next halt.	M	CHKANH
7	**checkImmediately** 0 = do not checkImmediately 1 = checkImmediately This value indicates to the failure that an immediate check of the vehicle is required.	M	CHKI

D.3.3 DTC class definition

The DTC class definitions apply to OBD systems which comply with the WWH-OBD GTR. Class A, B1, B2 or C are attributes of an emissions-related DTC. These attributes characterise the impact of a malfunction on emissions or on the OBD system's monitoring capability according to the requirements of the WWH-OBD GTR.

NOTE The DTC class information contained within a diagnostic request is allowed to have more than one bit set to 1 in order to request information for multiple DTC classes. The DTC class information contained within a diagnostic response shall only ever have a single bit set to 1. Table D.13 defines the GTR DTC Class definition (bit 4-0).

Table D.13 — GTR DTC Class definition (bit 4-0)

Bit Value	Description	Cvt	Mnemonic
0	**DTCClass_0** DTCClass 0 is unclassified. This class shall be used if DTCSeverity is included in the response message but no DTC class information is reported e.g. legacy DTCs as defined in SAE J2012-DA and ISO°14229-1. Bit = 0: DTCClass_0 is disabled for the reported DTC. Bit = 1: DTCClass_0 is enabled for the reported DTC.	M	DTCCLASS_0
1	**DTCClass_1** DTCClass_1 matches the GTR module B Class A definition. A malfunction shall be identified as Class A when the relevant OBD threshold limits (OTLs) are assumed to be exceeded. It is accepted that the emissions may not be above the OTLs when this class of malfunction occurs. Bit = 0: DTCClass_1 is disabled for the reported DTC. Bit = 1: DTCClass_1 is enabled for the reported DTC.	M	DTCCLASS_1

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table D.13 — *(continued)*

Bit Value	Description	Cvt	Mnemonic
2	**DTCClass_2** DTCClass_2 matches the GTR module B Class B1 definition. A malfunction shall be identified as Class B1 where circumstances exist that have the potential to lead to emissions being above the OTLs but for which the exact influence on emission cannot be estimated and thus the actual emissions according to circumstances may be above or below the OTLs. Class B1 malfunctions shall include malfunctions that restrict the ability of the OBD system to carry out monitoring of Class A or B1 malfunctions. Bit = 0: DTCClass_2 is disabled for the reported DTC. Bit = 1: DTCClass_2 is enabled for the reported DTC.	M	DTCCLASS_2
3	**DTCClass_3** DTCClass_3 matches the GTR module B Class B2 definition. A malfunction shall be identified as Class B2 when circumstances exist that are assumed to influence emissions but not to a level that exceeds the OTL. Malfunctions that restrict the ability of the OBD system to carry out monitoring of Class B2 malfunctions of shall be classified into Class B1 or B2. Bit = 0: DTCClass_3 is disabled for the reported DTC. Bit = 1: DTCClass_3 is enabled for the reported DTC.	M	DTCCLASS_3
4	**DTCClass_4** DTCClass_4 matches the GTR module B Class C definition. A malfunction shall be identified as Class C when circumstances exist that, if monitored, are assumed to influence emissions but to a level that would not exceed the regulated emission limits. Malfunctions that restrict the ability of the OBD system to carry out monitoring of Class C malfunctions shall be classified into Class B1 or B2. Bit = 0: DTCClass_4 is disabled for the reported DTC. Bit = 1: DTCClass_4 is enabled for the reported DTC.	M	DTCCLASS_4

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

D.4 DTCFormatIdentifier definition

This parameter value defines the format of a DTC reported by the server. A given server shall support only one DTCFormatIdentifier.

Table D.14 — Definition of DTCFormatIdentifier (DTCFID_)

Byte Value	Description	Cvt	Mnemonic
0x00	**SAE_J2012-DA_DTCFormat_00** This parameter value identifies the DTC format reported by the server as defined in ISO 15031-6 specification.	M	J2012-DADTCF00
0x01	**ISO_14229-1_DTCFormat** This parameter value identifies the DTC format reported by the server as defined in this table by the parameter DTCAndStatusRecord.	M	14229-1DTCF
0x02	**SAE_J1939-73_DTCFormat** This parameter value identifies the DTC format reported by the server as defined in SAE J1939-73.	M	J1939-73DTCF
0x03	**ISO_11992-4_DTCFormat** This parameter value identifies the DTC format reported by the server as defined in ISO 11992-4 specification.	M	11992-4DTCF
0x04	**SAE_J2012-DA_DTCFormat_04** This parameter value identifies the DTC format reported by the server as defined in ISO 27145-2 specification.	M	J2012-DADTCF04
0x05 - 0xFF	**ISO/SAE reserved**	M	ISOSAERESRVD
	This value is reserved by this document for future definition.		

D.5 FunctionalGroupIdentifier definition

The FunctionalGroupIdentifier specifies different functional system groups. The identifier is used to distinguish commands sent by the test equipment between different functional system groups within an electrical architecture which consists of many different servers. If a server 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 emissions I/M test should not be failed because another functional system group (e.g., safety system group) has DTC information stored.

The FunctionalGroupIdentifier specifies a functional system group for the purpose of:

— Requesting the Unified Diagnostic Services version number to identify the protocol,

— Requesting DTC status information from a vehicle, and

— Clearing DTC information in the vehicle.

The main purpose is to be able to report/clear DTC information specific to a functional system group. An ECU may be part of several functional system groups e.g. emissions system, brake system, etc. In case DTCs are reported for the brake system during an emissions inspection & maintenance (I/M) test the vehicle shall not fail the emissions I/M test because the ECU, which is part of the emissions functional system, also reports brake functional system DTCs.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table D.15 defines the FunctionalGroupIdentifiers.

Table D.15 — Definition of FunctionalGroupIdentifiers (FGID_)

Byte Value	Description	Cvt	Mnemonic
0x00 - 0x32	**ISO/SAE reserved** This range of values is reserved by this document for future definition.	M	ISOSAERESRVD
0x33	**Emissions-system group** This value identifies the Emissions system in a server.	M	EMSYSGRP
0x34 - 0xCF	**ISO/SAE reserved** This range of values is reserved by this document for future definition.	M	ISOSAERESRVD
0xD0	**Safety-system group** This value identifies the Safety system in a server.	M	SAFESYSGRP
0xD1 - 0xDF	**Legislative system group** This range of values is reserved for legislative required group identifiers by this document for future definition.	M	LEGSYSGRP
0xE0 - 0xFD	**ISO/SAE reserved** This range of values is reserved by this document for future definition.	M	ISOSAERESRVD
0xFE	**VOBD system** This value identifies the VOBD system device. Depending on the VOBD strategy which is implemented, only a gateway, a dedicated VOBD ECU or any other ECU which has the VOBD function implemented (e.g. engine controller) may respond.	M	VOBDSYSGRP
0xFF	**All functional system groups** This value identifies all functional system groups as listed in this table in a server.	M	ALLFCTSYSGRP

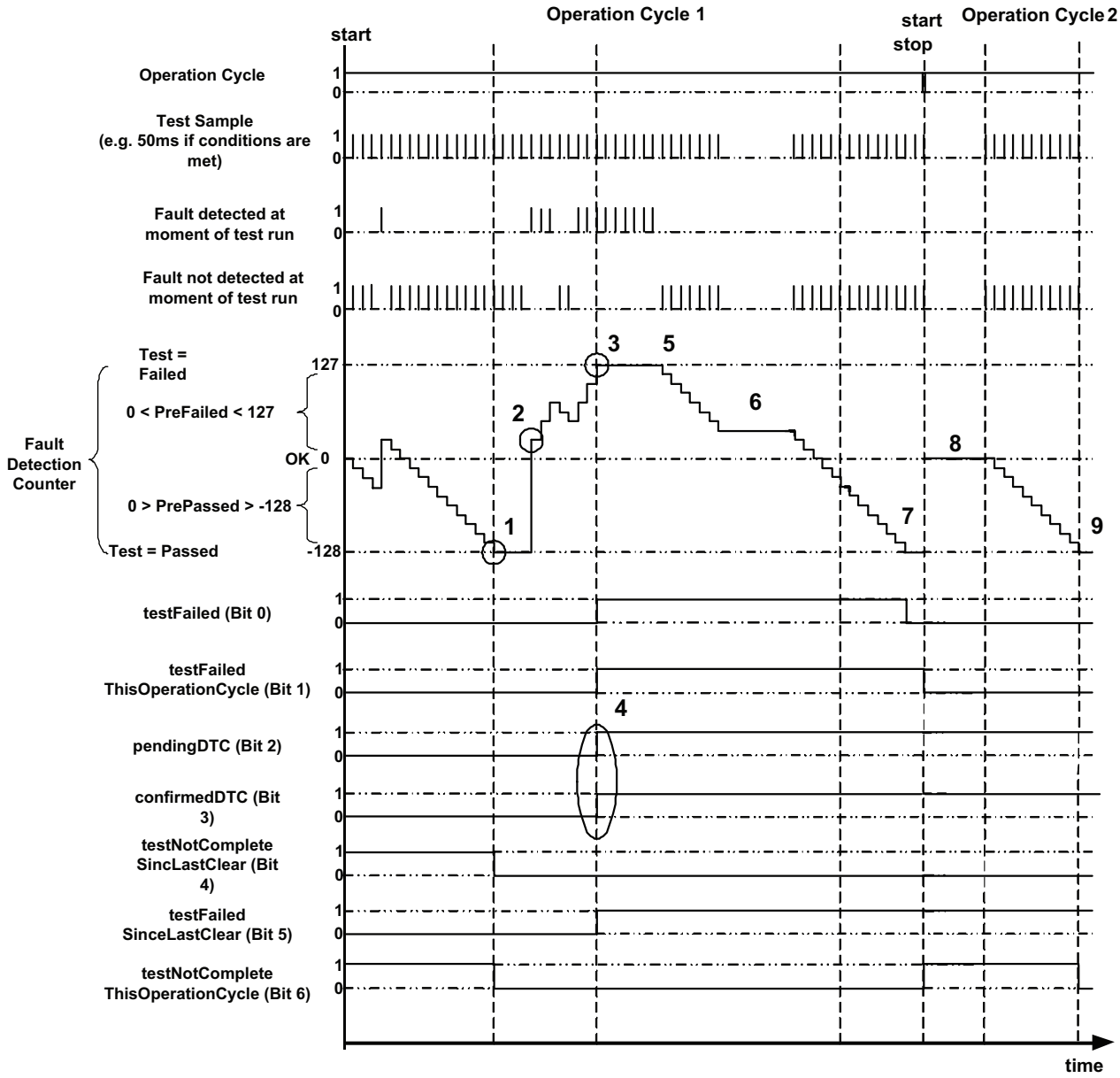
SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

D.6 DTCFaultDetectionCounter operation implementation example

The DTC fault detection counter operation for non-emissions related servers is shown in Figure D.10.



Key

1 Test completes when fault detection counter reaches minimum (-128) or maximum (127) and consequently the testNotCompleteSinceLastClear and testNotCompleteThisOperationCycle bits change from 1 to 0.

2 If one test sample of a test returns a failed result it always causes the fault detection counter to increment above 0 (ensures that the the fail detection time following a test complete with pass is not doubled)

3 The fault detection counter reaches its maximum (127) indicating a fault condition has fully matured; the test has reported a failed result consequently the testFailed, testFailedThisOperationCycle and testFailedSinceLastClear bits change from 0 to 1.

4 The ConfirmedDTC bit is set (change from 0 to 1) at the same time as the pendingDTC bit because this example is for a non emissions-related server/ECU with a confirmation threshold of 1.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

5 It is manufacturer specific if one test sample of a test returns a passed result it always causes the fault detection counter to decrement starting at 0 (ensures that the the passed detection time following a test complete with failed is not doubled).

6 The monitor(s) related to the test are not run because the monitor level enable condtions are not fulfilled, and therefore test sample results are generated. It is manufacturer specific whether or not the fault detection counter is reset to 0 when the monitor enable condition is again satisfied.

7 The counter reaches again its minimum (-128) in the current operation cycle and consequently the testFailed bit changes from 1 to 0.

8 After a new operation cycle has started the monitor(s) related to the test are not enabled yet; therefore the DTC status bits do not change except the bits which are linked to the start of the operation cycle. These bit are reset at the latest when the new operation cycle has started.

9 The counter reaches its minimum (-128) after a new operation cycle has started and consequently the testNotCompleteThisOperationCycle bit changes from 1 to 0.

Figure D.10 — Example of DTCFaultDetectionCounter operation for non-emissions related server

D.7 DTCAgingCounter example

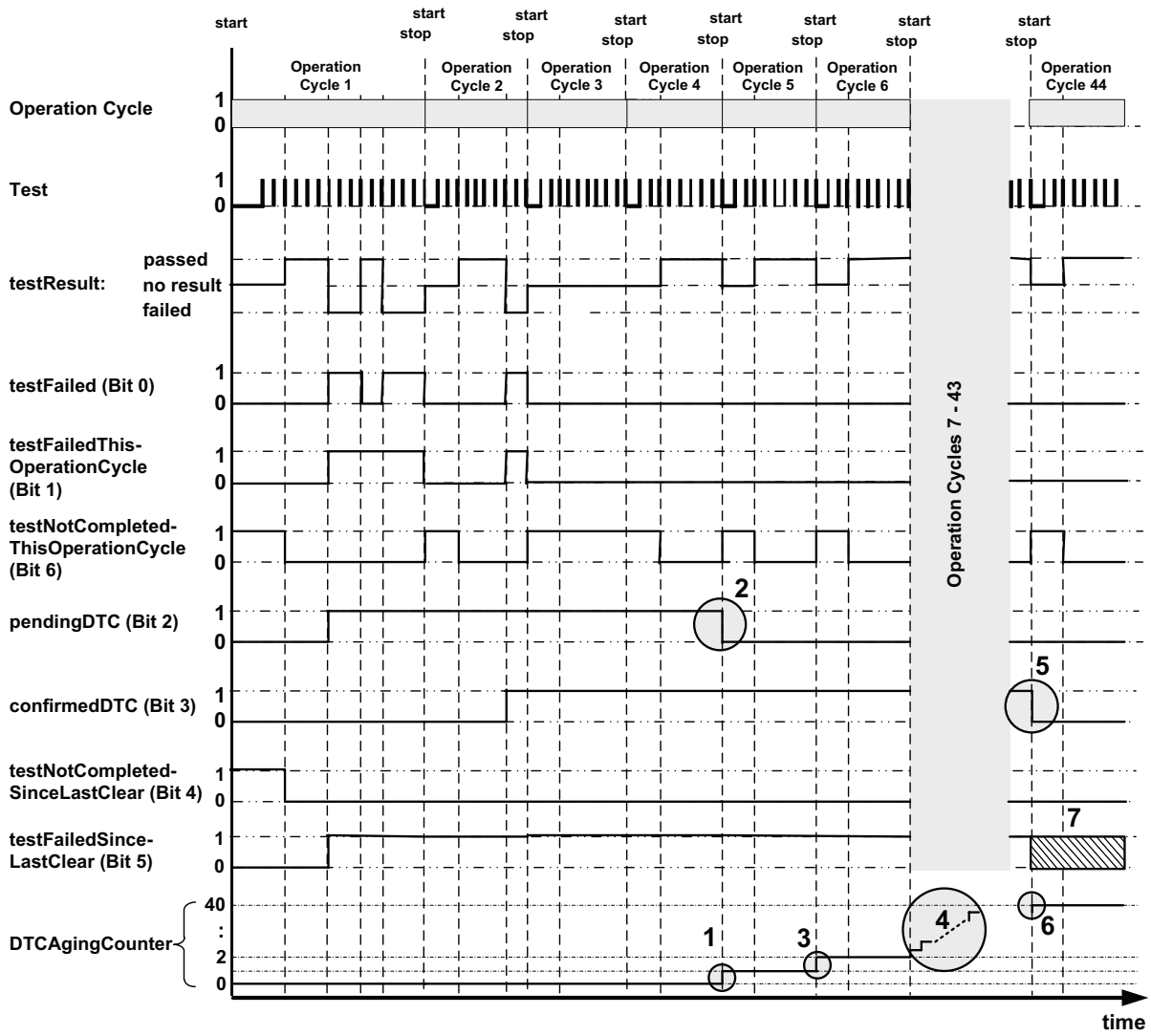
This example provides an overview on the operation of a DTCAgingCounter which counts number of driving cycles since the fault was latest failed.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Figure D.11 defines the DTCAgingCounter example.



Key

1 DTCAgingCounter is incremented after completing an operation cycle in which test did not fail

2 pendingDTC is set to zero after an operation cycle in which the test completed and did not fail. In case an ECU does not support a power down sequence (i.e. is immediately shut off when the ignition is turned off) it will be unable to detect the end of the operation cycle. Therefore it is also valid to set the pendingDTC bit to zero at the beginning of the next operation cycle

3 DTCAgingCounter is incremented after completing an operation cycle in which test did not fail

4 DTCAgingCounter continues to increment because test is not failing during these operation cycles

5 confirmedDTC is set to zero when aging criteria is fully satisfied (e.g., DTCAgingCounter reaches a specific value)

6 DTCAgingCounter reaches a maximum value (e.g., 40) at which time the confirmedDTC bit is cleared

7 It is the responsibility of the vehicle manufacturer to specify whether or not testFailedSinceLastClear bit is reset by aging-criteria or due to an overflow of the fault memory

Figure D.11 — DTCAgingCounter example

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Annex E
(normative)

Input output control functional unit data-parameter definitions

E.1 InputOutputControlParameter definitions

Table E.1 defines the inputOutputControlParameter.

Table E.1 — inputOutputControlParameter definitions

Byte Value	Description	Cvt	Mnemonic
0x00	**returnControlToECU** This value shall indicate to the server that the client does no longer have control about the input signal(s), internal parameter(s) and/or output signal(s) referenced by the dataIdentifier. Details of controlState bytes in request: 0 bytes Details of controlState bytes in positive response: Equal to the size and format of the dataIdentifier's dataRecord	U	RCTECU
0x01	**resetToDefault** This value shall indicate to the server that it is requested to reset the input signal(s), internal parameter(s) and/or output signal(s) referenced by the dataIdentifier to its default state. Details of controlState bytes in request: 0 bytes Details of controlState bytes in positive. response: Equal to the size and format of the dataIdentifier's dataRecord	U	RTD
0x02	**freezeCurrentState** This value shall indicate to the server that it is requested to freeze the current state of the input signal(s), internal parameter(s) and/or output signal referenced by the dataIdentifier. Details of controlState bytes in request: 0 bytes Details of controlState bytes in positive. response: Equal to the size and format of the dataIdentifier's dataRecord	U	FCS
0x03	**shortTermAdjustment** This value shall indicate to the server that it is requested to adjust the input signal(s), internal parameter(s) and/or controlled output signal(s) referenced by the dataIdentifier in RAM to the value(s) included in the controlOption parameter(s) (e.g., set Idle Air Control Valve to a specific step number, set pulse width of valve to a specific value/duty cycle). Details of controlState bytes in request: Equal to the size and format of the dataIdentifier's dataRecord Details of controlState bytes in pos. response: Equal to the size and format of the dataIdentifier's dataRecord	U	STA
0x04 – 0xFF	**ISOSAEReserved** This value is reserved by this document for future definition.	M	ISOSAERESRVD

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Annex F
(normative)

Routine functional unit data-parameter definitions

F.1 RoutineIdentifier (RID) definition

Table F.1 defines the routineIdentifier.

Table F.1 — routineIdentifier definition

Byte Value	Description	Cvt	Mnemonic
0x0000 – 0x00FF	**ISOSAEReserved** This value shall be reserved by this document for future definition.	M	ISOSAERESRVD
0x0100 - 0x01FF	**TachographTestIds** This range of values is reserved to represent Tachograph test result values.	U	TACHORI_
0x0200 - 0xDFFF	**vehicleManufacturerSpecific** This range of values is reserved for vehicle manufacturer specific use.	U	VMS_
0xE000 0xE1FF	**OBDTestIds** This range of values is reserved to represent OBD/EOBD test result values.	U	OBDRI_
0xE200	**DeployLoopRoutineID** This value shall be used to initiate the deployment of the previously selected ignition loop.	U	DLRI_
0xE201 – 0xE2FF	**SafetySystemRoutineIDs** This range of values shall be reserved by this document for future definition of routines implemented by safety related systems.	M	SASRI_
0xE300 - 0xEFFF	**ISOSAEReserved** This value shall be reserved by this document for future definition.	M	ISOSAERESRVD
0xF000 - 0xFEFF	**systemSupplierSpecific** This range of values is reserved for system supplier specific use.	U	SSS_
0xFF00	**eraseMemory** This value shall be used to start the server's memory erase routine. The Control option and status record format shall be ECU specific and defined by the vehicle manufacturer.	U	EM_
0xFF01	**checkProgrammingDependencies** This value shall be used to check the server's memory programming dependencies. The Control option and status record format shall be ECU specific and defined by the vehicle manufacturer.	U	CPD_
0xFF02	**eraseMirrorMemoryDTCs** This value shall be used to erase the server's mirror memory DTCs.	U	EMMDTC_
0xFF03 - 0xFFFF	**ISOSAEReserved** This value shall be reserved by this document for future definition.	M	ISOSAERESRVD

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Annex G
(normative)

Upload and download functional unit data-parameter

G.1 Definition of modeOfOperation values

The RequestFileTransfer request message contains the modeOfOperation parameter. The values are defined in Table G.1.

Table G.1 — Definition of modeOfOperation values

Byte Value	Description	Cvt	Mnemonic
0x00	**ISO/SAE reserved** This value is reserved by this document for future definition.	M	ISOSAERESRVD
0x01	**AddFile** This value shall be used to add the file (download) defined in the filePathAndName parameter.	U	ADDFILE
0x02	**DeleteFile** This value shall be used to delete the file defined in the filePathAndName parameter.	U	DELFILE
0x03	**ReplaceFile** This value shall be used to replace the file (download) defined in the filePathAndName parameter. If the file is not stored at the location the file shall be added.	U	REPLFILE
0x04	**ReadFile** This value shall be used to read the file (upload) at the location defined by the filePathAndName parameter.	U	RDFILE
0x05	**ReadDir** This value shall be used to read the directory defined in the filePathAndName parameter. This value implies that the request does not include a fileName.	U	RDDIR
0x06 - 0xFF	**ISO/SAE reserved** This value is reserved by this document for future definition.	M	ISOSAERESRVD

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Annex H
(informative)

Examples for addressAndLengthFormatIdentifier parameter values

H.1 addressAndLengthFormatIdentifier example values

Table H.1 contains examples of combinations of values for the high and low nibble of the addressAndLengthFormatIdentifier. The following needs to be considered:

— Values, which are either marked as "not applicable" for the "manageable memorySize" or the "memoryAddress range", are not allowed to be used and have to be rejected by the server via a negative response message.

— Values with an applicable "manageable memorySize" and "memoryAddress range" are allowed for this parameter.

Table H.1 — addressAndLengthFormatIdentifier example

Byte Value	Description			
	bit 7-4 (high nibble) number of memorySize bytes		bit 3-0 (low nibble) number of memoryAddress bytes	
	bytes used for memorySize parameter	manageable size	bytes used for memoryAddress parameter	addressable memory
0x00	not applicable	not applicable	not applicable	not applicable
0x01	not applicable	not applicable	1	256 Byte - 1
0x02	not applicable	not applicable	2	64 KB - 1
0x03	not applicable	not applicable	3	16 MB - 1
0x04	not applicable	not applicable	4	4 GB - 1
0x05	not applicable	not applicable	5	1,024 GB - 1
0x06 – 0x0F	:	:	:	:
0x10	1	256 Byte	not applicable	not applicable
0x11	1	256 Byte	1	256 Byte – 1
0x12	1	256 Byte	2	64 KB – 1
0x13	1	256 Byte	3	16 MB – 1
0x14	1	256 Byte	4	4 GB – 1
0x15	1	256 Byte	5	1,024 GB – 1
0x16 – 0x1F	:	:	:	:
0x20	2	64 KB	not applicable	not applicable
0x21	2	64 KB	1	256 Byte – 1
0x22	2	64 KB	2	64 KB – 1
0x23	2	64 KB	3	16 MB – 1
0x24	2	64 KB	4	4 GB – 1

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table H.1 — *(continued)*

Byte Value	Description			
	bit 7-4 (high nibble) number of memorySize bytes		bit 3-0 (low nibble) number of memoryAddress bytes	
	bytes used for memorySize parameter	manageable size	bytes used for memoryAddress parameter	addressable memory
0x25	2	64 KB	5	1,024 GB – 1
0x26 – 0x2F	:	:	:	:
0x30	3	16 MB	not applicable	not applicable
0x31	3	16 MB	1	256 Byte – 1
0x32	3	16 MB	2	64 KB – 1
0x33	3	16 MB	3	16 MB – 1
0x34	3	16 MB	4	4 GB – 1
0x35	3	16 MB	5	1,024 GB – 1
0x36 – 0x3F	:	:	:	:
0x40	4	4 GB	not applicable	not applicable
0x41	4	4 GB	1	256 Byte – 1
0x42	4	4 GB	2	64 KB – 1
0x43	4	4 GB	3	16 MB – 1
0x44	4	4 GB	4	4 GB – 1
0x45	4	4 GB	5	1,024 GB - 1
0x46 -0xFF	:	:	:	:

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Annex I
(normative)

Security access state chart

I.1 General

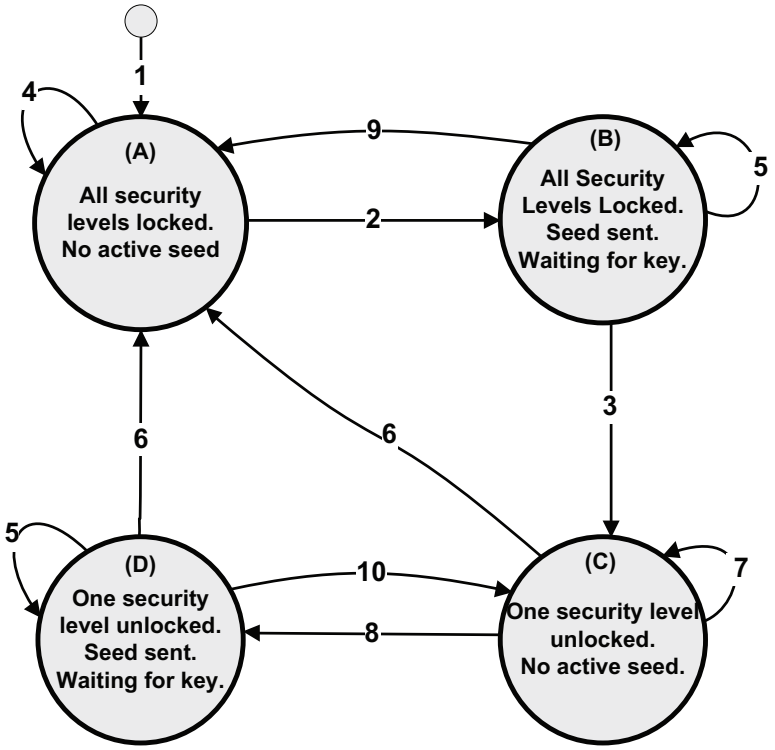
The purpose of this annex is to describe the SecurityAccess service handling in an ECU, based on a state chart with state transition conditions and action definitions. The following is the base for the definition:

— Usage of the disjunctive normal form in order to have single transitions defined between and within the states.

— Definition of “disjunctive normal form”: “A statement is in disjunctive normal form if it is a disjunction (sequence of ORs) consisting of one or more disjuncts, each of which is a conjunction (AND) of one or more literals”

I.2 Disjunctive normal form based state transition definitions

Figure I.1 graphically depicts the state chart for the SecurityAccess handling. The given numbers reference state transition conditions and actions to be performed on the transition.



Key

(A) All security levels locked. No active seed.
(B) All Security Levels Locked. Seed sent. Waiting for key.
(C) One security level unlocked. No active seed.
(D) One security level unlocked. Seed sent. Waiting for key.
1 .. 10 see Table I.2

Figure I.1 — SecurityAccess state chart

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

The state chart takes into account that the general session handling is done at the proper place within the session management layer (see ISO 14229-2) and therefore does not need to be considered in the state-chart.

The state transition definitions make use of some parameters that can be set according to vehicle manufacturer specific requirements. The support of Delay_Timer and Att_Cnt parameters is optional and decided by the vehicle manufacturer. In general, for longer seed/key lengths (e.g., 16 bytes and beyond) the support of these parameters is no longer as important.

Table I.1 defines the state transitions – parameters.

Table I.1 — State transitions – parameters

Name	Description
Delay_Timer	If supported, this value represents the required minimum time between security access attempts. In addition, it is vehicle manufacturer specific whether this delay timer will be invoked upon every power on / start up. The standard use case will have a fixed value for the delay time, but it is also possible for a customer-specific use case to have a variable value (e.g., the value depends on the number of false access attempts in case they are stored in non-volatile memory). NOTE A server may choose to implement a separate timer for each security level or utilize a single timer for all levels.
Att_Cnt	If supported, this value represents the number of false security access attempts before a delay time (Start_Delay) is inserted. When implemented, the counter is required for each individual security level.
Static_Seed	This represents a boolean value where true indicates that a seed is stored and re-used in a positive response to a seed request under certain conditions according to Table I.2. A value of false indicates that a random seed is used every time a new seed request is received. If Delay_Timer and Att_Cnt are not supported, a random seed shall always be used.
xx	This represents the last requestSeed securityAccessType received by the server.
yy	This represents the current sendKey securityAccessType received by the server.

Legend:

AND, OR	logical operation
Italic	optional, customer specific
“==”	equal (comparison operator)
"="	assignment operator
“<>”	un-equal
“<”	less than
“>”	greater than
“+”	mathematical addition
“-”	mathematical subtraction
"++"	increment operator (variable++ is the same as variable = variable + 1)

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table I.2 includes the complete set of state transition definitions.

Table I.2 — State transitions – disjunctive normal form representation

No.	Operation		Condition	Action
1			Start / restart of ECU Application (e.g., ECU reset, power cycle, key cycle, sleep → wake transition, etc.).	Initialize Att_Cnt (if applicable). Start Delay_Timer[a] (if required on start up).
2		AND	SecurityAccess requestSeed received. Message length OK[b]. *Optional pre-conditions fulfilled*. Delay expired (if applicable).	If Static_Seed == True then generate and store Seed for the requested securityAccessType (if not previously generated and stored during the current ECU operating cycle). If Static_Seed = False, then generate new Seed. Save sub-function: xx = securityAccess Type Transmit SecurityAccess positive response on requestSeed request with Seed as active for the requested securityAccessType.
3		AND	SecurityAccess sendKey received sub-function: yy == xx+1[c]. Message length OK. Key OK.	Att_Cnt = 0 for subfunction xx (if applicable). Store Att_Cnt in non-volatile memory (if applicable). Unlock security level for subfunction xx. If Static_Seed = True then clear generated seed for subfunction xx. Transmit SecurityAccess positive response on sendKey request.
4	OR	AND	SecurityAccess requestSeed received. Message Length NOK.	Transmit negative response NRC 0x13.
			SecurityAccess sendKey received.	Transmit negative response NRC 0x24.
		AND	*SecurityAccess requestSeed received.* *Message length OK.* *Optional pre-conditions NOT fulfilled*[d].	Transmit negative response NRC 0x22.
		AND	SecurityAccess requestSeed received. Message length OK. Delay NOT expired (if applicable). *Optional pre-conditions fulfilled.*	Transmit negative response NRC 0x37.
			SecurityAccess request results in a general negative response code (e.g., minimum length, sub-function supported) according to the general negative response handling (see section 7.5).	Transmit negative response code as defined in section 7.5.
5	OR	AND	SecurityAccess requestSeed received. Static_Seed == False.	Generate new seed and transmit SecurityAccess positive response with the new seed for the requested securityAccessType. Save sub-function: xx = securityAccess Type
		AND	SecurityAccess requestSeed received. Static_Seed == True. requested securityAccessType has an active stored seed.	Transmit SecurityAccess positive response with the active stored seed for the requested securityAccessType. Save sub-function: xx = securityAccess Type
		AND	SecurityAccess requestSeed received. Static_Seed == True. requested securityAccessType has no active seed stored (different securityAccessType than before).	Generate and store Seed for the requested securityAccessType (if not previously generated and stored during the current ECU operating cycle). Save sub-function: xx = securityAccess Type Transmit SecurityAccess positive response on requestSeed request with Seed as active for the requested securityAccessType.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table I.2 — *(continued)*

No.	Operation		Condition	Action
6			DiagnosticSessionControl accepted or session timeout occurs.	Start appropriate diagnostic session. Lock ECU.
7	OR		SecurityAccess sendKey received.	Transmit negative response NRC 0x24.
		AND	SecurityAccess requestSeed received.	Transmit SecurityAccess positive response with zero seed[e].
			Requested level is unlocked.	
			SecurityAccess request results in a general negative response code (e.g., minimum length, sub-function supported) according to the general negative response handling (see section 7.5).	Transmit negative response code as defined in section 7.5.
8		AND	SecurityAccess requestSeed received.	Generate and store Seed for the requested securityAccessType (if not previously generated and stored during the current ECU operating cycle). Save sub-function: xx = securityAccess Type Transmit SecurityAccess positive response on requestSeed request with Seed as active for the requested securityAccessType.
			Requested level is NOT unlocked.	
			Message length OK.	
			Optional pre-conditions fulfilled.	
			Delay expired (if applicable).	
9	OR	AND	SecurityAccess sendKey received.	Att_Cnt++ for sub-function xx (if applicable)[f]. *Store Att_Cnt in non-volatile memory (if applicable).* Transmit negative response NRC 0x35.
			sub-function: yy == xx+1.	
			Message length OK.	
			Key NOK.	
			(Att_Cnt+1) < Att_Cnt_Limit (if applicable).	
		AND	SecurityAccess sendKey received.	Att_Cnt++ for sub-function xx (if applicable). *Start Delay_Timer for sub-function xx (if applicable).* *Store Att_Cnt in non-volatile memory (if applicable).* Transmit negative response NRC 0x36.
			sub-function: yy == xx+1.	
			Message length OK.	
			Key NOK.	
			(Att_Cnt+1) >= Att_Cnt_Limit.	
		AND	SecurityAccess sendKey received.	Transmit negative response NRC 0x24. Att_Cnt++ for sub-function xx (if applicable). *Store Att_Cnt in non-volatile memory.*
			sub-function: yy <> xx+1.	
		AND	SecurityAccess sendKey received.	Transmit negative response NRC 0x13. Att_Cnt++ for sub-function xx (if applicable). *Store Att_Cnt in non-volatile memory.*
			sub-function: yy == xx+1.	
			Message length NOK.	
		AND	DiagnosticSessionControl accepted or session timeout occurs.	Start appropriate diagnostic session.
			SecurityAccess request results in a general negative response code (e.g., minimum length, sub-function supported) according to the general negative response handling (see section 7.5).	Transmit negative response code as defined in section 7.5.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table I.2 — *(continued)*

No.	Operation		Condition	Action
10	OR	AND	SecurityAccess requestSeed received.	Transmit SecurityAccess positive response with zero seed[g].
			Requested level is unlocked.	
		AND	SecurityAccess sendKey received.	Att_Cnt = 0 for subfunction xx (if applicable). *Store Att_Cnt in non-volatile memory (if applicable).* Lock currently unlocked security level. Unlock security level for subfunction xx. If Static_Seed = True then clear generated seed for subfunction xx.Transmit SecurityAccess positive response on sendKey request.
			sub-function: yy == xx+1[h].	
			Message length OK.	
			Key OK.	
		AND	SecurityAccess sendKey received.	Att_Cnt++ for sub-function xx (if applicable)[i]. *Store Att_Cnt in non-volatile memory (if applicable).* Transmit negative response NRC 0x35.
			sub-function: yy == xx+1.	
			Message length OK.	
			Key NOK.	
			(Att_Cnt+1) < Att_Cnt_Limit (if applicable).	
		AND	SecurityAccess sendKey received.	Att_Cnt++ for sub-function xx (if applicable). *Start Delay_Timer for sub-function xx (if applicable).* *Store Att_Cnt in non-volatile memory (if applicable).* Transmit negative response NRC 0x36.
			sub-function: yy == xx+1.	
			Message length OK.	
			Key NOK.	
			(Att_Cnt+1) >= Att_Cnt_Limit.	
		AND	SecurityAccess sendKey received.	Transmit negative response NRC 0x24. Att_Cnt++ for sub-function xx (if applicable)[j].
			sub-function: yy <> xx+1.	
		AND	SecurityAccess sendKey received.	Transmit negative response NRC 0x13. Att_Cnt++ for sub-function xx (if applicable).
			sub-function: yy == xx+1.	
			Message length NOK.	
			SecurityAccess request results in a general negative response code (e.g., minimum length, sub-function supported) according to the general negative response handling (see section 7.5).	Transmit negative response code as defined in section 7.5.

[a] The default use case will have a fixed value for the delay time, but it is also possible for a customer-specific use case that the value depends on the number of false access attempts in case they are stored in non-volatile memory.

[b] The exact length check can only be done after the evaluation of the sub-function, because the length depends on the sub-function (i.e., length of requestSeed is different from length of sendKey message). The check for the minimum length is done during the general service evaluation process.

[c] The sendKey sub-function (yy) must be of the expected securityAccessType (the active stored seed is for the corresponding requestSeed securityAccessType, i.e. sendKey securityAccessType – 1).

[d] Customer specific precondition can be checked (e.g. fingerprint written in this driving cycle, engine not running, and vehicle not moving).

[e] Once a given security level is unlocked, it shall remain unlocked even after a seed request is received for a different security level until either a new security level is completely unlocked or the security access is exited for other reasons (e.g., DiagnosticSessionControl accepted or session timeout occurs).

[f] The counter for false access attempts will be increased with every valid formatted, but invalid key value and will be set to zero in case a valid key is received. It may be a customer-specific use case to store this counter in non-volatile memory to be able to decide after reset if a delay has to be started or not (and possibly the delay time even depends on the value of this counter). In case a valid formatted key was received the stored seed shall be discarded.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

[g] Once a given security level is unlocked, it shall remain unlocked even after a seed request is received for a different security level until either a new security is completely unlocked or the security access is exited for other reasons (e.g., diagnosticSessionControl received).

[h] The sendKey sub-function must be of the expected access type (active stored seed is for sendKey accessType – 1).

[i] The counter for false access attempts will be increased with every valid formatted, but invalid key value and will be set to zero in case a valid key is received. It may be a customer-specific use case to store this counter in non-volatile memory to be able to decide after reset if a delay has to be started or not (and possibly the delay time even depends on the value of this counter). In case a valid formatted key was received the stored seed shall be discarded.

[j] The counter for false access attempts will be increased with every valid formatted, but invalid key value and will be set to zero in case a valid key is received. It may be a customer-specific use case to store this counter in non-volatile memory to be able to decide after reset if a delay has to be started or not (and possibly the delay time even depends on the value of this counter). In case a valid formatted key was received the stored seed shall be discarded.

NOTE It has to be considered that when defining the state transitions via multiple conjunctions which are OR-ed together and each conjunction has an action applied that only one of the conjunctions of a disjunction becomes true at a time and forces a state transition in order to only execute one of the actions for a certain state transition defined (e.g. only single negative response to be transmitted).

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Annex J
(informative)

Recommended implementation for multiple client environments

J.1 Introduction

This annex is intended to address the increasing number of use cases where the diagnostic vehicle topology is extended by adding one or more onboard diagnostic clients to the basic diagnostic topology with a single diagnostic client (external test equipment) and multiple servers (ECUs in vehicle).

This document and the normative references herein do not limit the number of diagnostic communication channels that a server can support. The design of such a server-implementation for multi-client handling, needs to take into account that there are specifications and restrictions which force certain diagnostic clients to be served with a higher priority than others, e.g. to fulfil existing legislative OBD requirements. In this case the vehicle system design needs to ensure that parallel client requests can be handled by the respective server(s).

An example for such a scenario would be an internal data logger which is connected to a server in parallel to a OBD scan tool externally connected to the diagnostic connector.

Either the overall vehicle design accounts for this parallel handling of client requests (e.g. gateway arbiter mechanism) or the individual servers have to implement new strategies to assign the available resources to different clients. In the server either the protocol implementation or the available resources are unique and can only be accessed by one client at a time.

This annex describes the implementation on server level only. It is the vehicle manufacturer's responsibility to select a mechanism which fits its individual needs best.

J.2 Implementation specific limitations

A unique Address Information must be assigned to each communication participant to allow the detection of different clients, which then can be used to limit the functionality or to assign priorities.

If the vehicle manufacturer's design does not use unique address information for certain peer protocol entities, the implementation described in this annex does not apply. In this case the vehicle design needs to ensure that the chosen approach for handling of multiple clients fulfils the legislative requirements.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

J.3 Use cases relevant for system design

Figure J.1 shows an example of a vehicle topology where multiple clients exist.

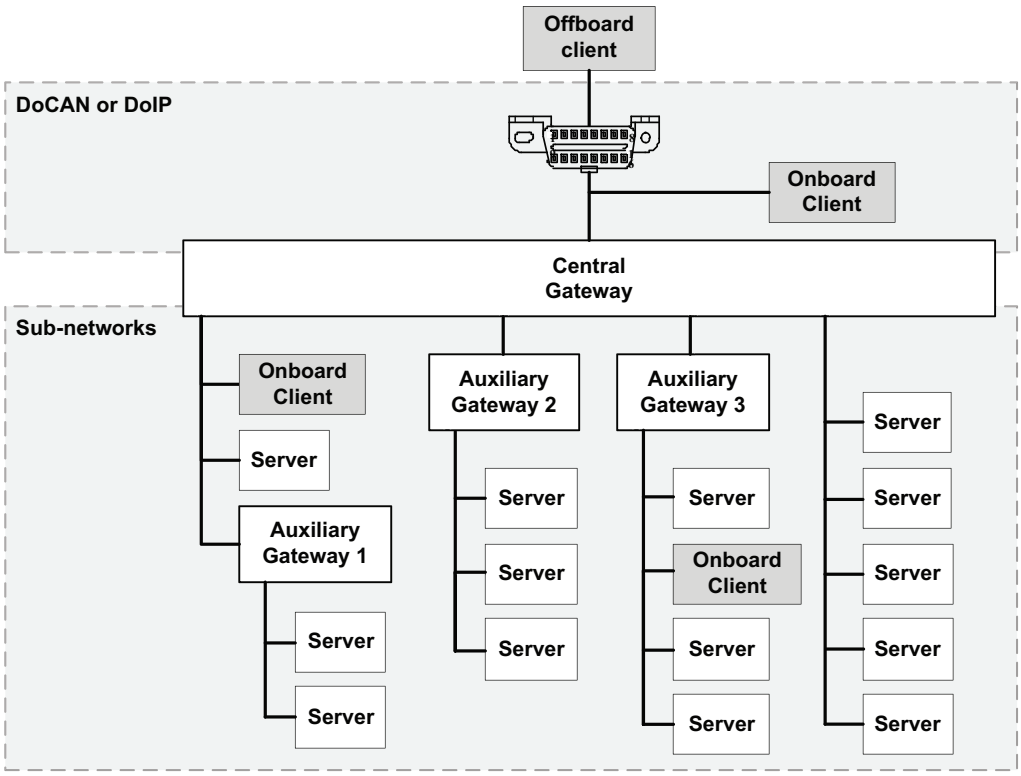


Figure J.1 — Example vehicle topology with onboard clients

The implementation described in this document is intended to fulfil the use cases summarized in Figure J.1. All use case scenarios marked with an 'N/A' in the table below are not described as part of this standard. It is highly recommended to avoid such scenarios. The implementation and design rules specified in this Annex are not intended to support OBD communication requirements beyond the scenarios defined in Table J.1.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Table J.1 — Use case (UC) matrix of multiple client scenarios to be addressed

Test equipment in use / Additional test equipment	Off-board clients (vehicle external test equipment)		On-board clients (vehicle internal test equipment)		
	OBD scan (tool) test equipment	**OEM service test equipment**	**On-board client 1**	**On-board client 2**	**On-board client n**
OBD scan (tool) test equipment	Not existent	N/A	A (UC 1)	A (UC 1)	A (UC 1)
OEM service test equipment	N/A	Not existent	X (UC 2)	X (UC 2)	X (UC 2)
On-board client 1	T (UC 3)	X (UC 4)	Not existent	X (UC 5)	X (UC 5)
On-board client 2	T (UC 3)	X (UC 4)	X (UC5)	Not existent	X (UC 5)
…	…	…	…	…	…
On-board client n	T (UC3)	X (UC 4)	X (UC5)	X (UC5)	Not existent
T: Test equipment in use has higher pirority than additional test tool A: Additional test equipment has higher priority than test tool in use X: vehicle manufacturer specific (equal or different priority)					

When referring to the term 'test equipment in use' it needs to be differentiated between the server perspective and the client perspective as follows:

— from a server perspective a test tool is considered in use if a request is currently processed or a non-default session is active

— from a client perspective a test tool is considered in use if an expected response has not yet been received, P3 client is not expired yet or a non-default session is active

When referring to the term 'additional test equipment' the following definition applies:

— a test equipment in this context is considered 'additional' if another tool is in use (refer to definition of test tool in use)

When referring to the term 'OBD scan (tool) test equipment' the following definition applies:

— On-Board Diagnostic (OBD) regulations require passenger cars and light, medium and heavy duty trucks to support communication of a minimum set of diagnostic information with off-board test equipment according to SAE J1978 / ISO 15031-4. A vehicle is considered non-compliant if the communication with the test equipment (e.g., handheld scan tools, PC based diagnostic computers, etc.) cannot be conducted as defined by the appropriate standards.

When referring to the term 'OEM service test equipment' the following definition applies:

— An OEM specified test equipment which fulfils the OEM requirements and utilizes proprietary address information.
The OEM Service test equipment may utilize standardized parts, i.e. SAE J2534 to communicate between the application and the Service tool hardware, but the communication to the vehicle utilizes OEM proprietary information.

When referring to the term 'on-board client' the following definition applies:

— An ECU which may include at least a diagnostic client part but also may include a diagnostic server part. The client part has functionality to sent diagnostic service requests to other servers in the vehicle. An example may be a telematics gateway which can be integrated into an ECU which also includes other functionality. The telematics gateway will act as a server if an OEM Service tool is connected to the

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

diagnostic connector and requests data from the telematic gateway, but the telematic gateway itself also acts as a client requesting data from the other servers in the vehicle.

J.4 Use Case Evaluation:

Table J.2 is intended to guide the decision what kind of concept the system designer should select.

Table J.2 — Evaluation of multiple client use cases

Use Case #	Pseudo parallel concept	Priority concept
UC 1, UC3	**Pro:** — both type of test equipment can be handled without stopping a protocol — all clients will be informed by the server (worst case NRC 0x21, usually positive response) — if OBD scan (tool) test equipment client is permanently connected (3rd party tools), OBD and non-OBD requests can be handled in parallel	**Pro:** — processing based on dedicated priority assumption: OBD has higher prio than onboard client — no resource management needed — dedicated timing behaviour for OBD responses, due to the fact the ongoing non-OBD response will be stopped — just one single buffer required
	Con: — resource management needed — non-OBD request might be rejected or not even responded to — OBD II: if the physical OBD CAN IDs are used for UDS concept is not working	**Con:** — client (on-board test equipment) request can only processed when OBD request is not currently processed — enable application to 'kill' ongoing requests from other clients — If permanently connected it depends on the request frequency whether the onboard client is still able to collect data on-board or not.
UC2, UC4, UC5	**Pro:** — client requests are handled based on arrival time without stopping a protocol if default session is active and protocol parameters are identical — client is informed by NRC 0x21 when server is busy processing a different request or being in non-default session requested by a different client	**Pro:** — processing based on dedicated priority — allows to prioritize between different clients
	Con: — parallel handling just possible if clients do not request a non-default session — client shall always request default session when done with data retrieval	**Con:** — low prio client not informed about the fact that it won't be served — detection just via timeout ($P2_{max}$ timeout) — enable application to 'kill' ongoing requests from other clients

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

J.5 Multiple client server level implementation

J.5.1 Definition of diagnostic protocol

In this context a diagnostic communication protocol is a compilation of specific parameter values depending on the Address Information (e.g. protocol buffer size, session timings, supported services, security levels).

A protocol is identified by a communication path established between peer protocol entities. Each peer protocol entity has exactly one unique physical address, and 0..n functional addresses (for the server(s)) identified by the respective N_AI.

NOTE 1 That means one single address cannot be used for different protocols.

NOTE 2: As defined in this part of ISO 14229 there is only one diagnostic session state and one security level state active at a time in one specific ECU and shared over all active protocols.

J.5.2 Assumptions

A protocol can either have exclusive protocol resources or multiple protocols can share one protocol resource.

The OBD scan (tool) test equipment client address has either the highest priority or an exclusive protocol resource is assigned to this address ensuring that the legislative requirements can be fulfilled.

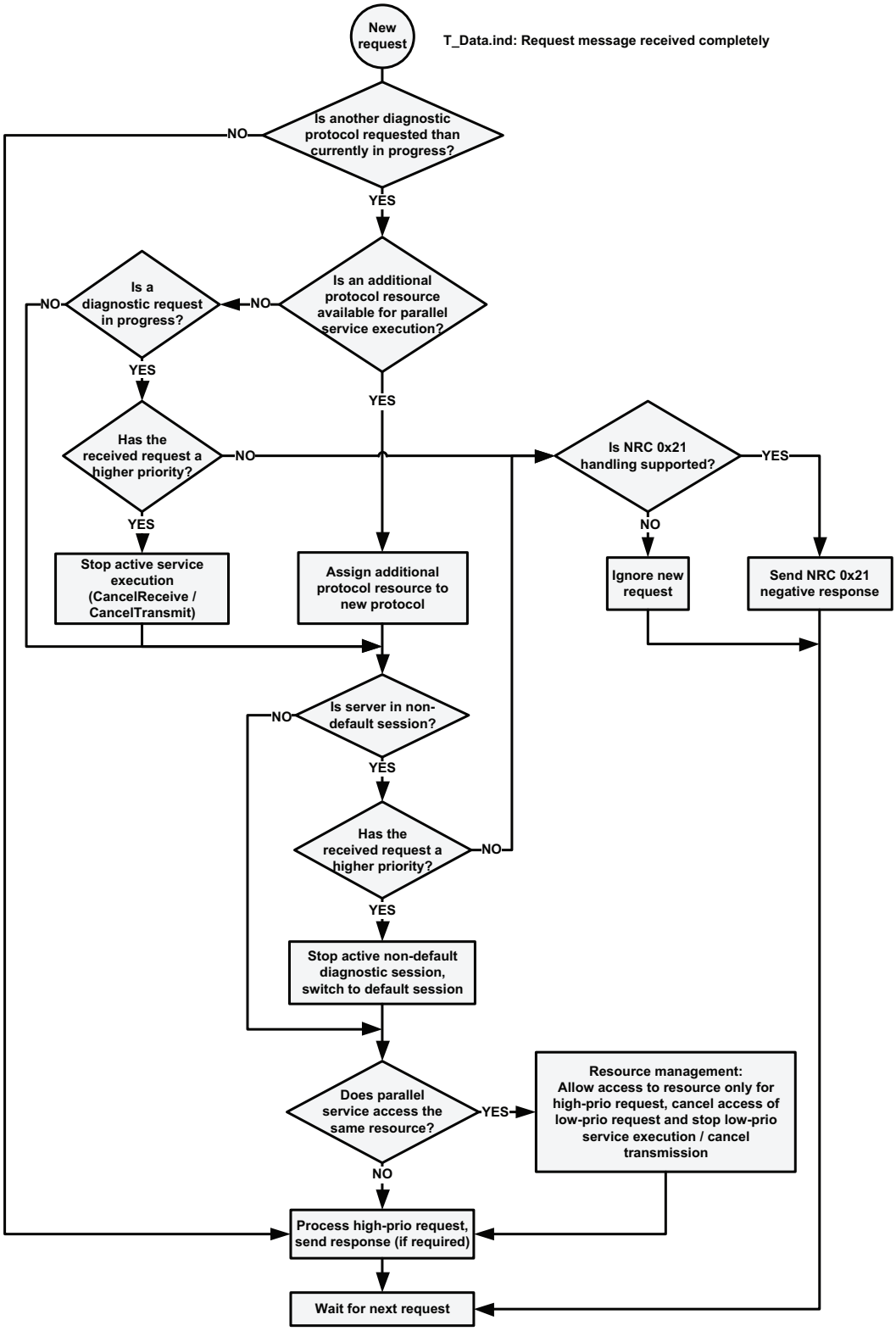
J.5.3 Multiple client handling flow

If a server implements multiple client handling on server level the implementation shall adhere to the flow chart depicted in Figure J.2.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved



Key

1 Reason for overflow: The temporary receive-buffer for the 2nd request is limited to one frame.

Figure J.2 — Multiple client handling flow

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Bibliography

[1] ISO 4092:1988/Cor.1:1991, *Road vehicles — Diagnostic systems for motor vehicles — Vocabulary — Technical Corrigendum 1*

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

[3] ISO/TR 8509:1987, *Information processing systems — Open Systems Interconnection — Service conventions*

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

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

[6] ISO 14229-3, *Road vehicles — Unified diagnostic services (UDS) — Part 3: Unified diagnostic services on CAN implementation (UDSonCAN)*

[7] ISO 14229-4, *Road vehicles — Unified diagnostic services (UDS) — Part 4: Unified diagnostic services on FlexRay implementation (UDSonFR)*

[8] ISO 14229-5, *Road vehicles — Unified diagnostic services (UDS) — Part 5: Unified diagnostic services on Internet Protocol implementation (UDSonIP)*[1)]

[9] ISO 14229-6, *Road vehicles — Unified diagnostic services (UDS) — Part 6: Unified diagnostic services on K-Line implementation (UDSonK-Line)*

[10] ISO 14229-7, *Road vehicles — Unified diagnostic services (UDS) — Part 7: Unified diagnostic services on Local Interconnect Network implementation (UDSonLIN)*[2)]

[11] ISO 15031-2, *Road vehicles — Communication between vehicle and external equipment for emissions-related diagnostics — Part 2: Guidance on terms, definitions, abbreviations and acronyms*

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

[13] ISO 15765-4, *Road vehicles — Diagnostic communication over Controller Area Network (DoCAN) — Part 4: Requirements for emissions-related systems*

[14] ISO 22901-1, *Road vehicles — Open diagnostic data exchange (ODX) — Part 1: Data model specification*

[15] ISO 26021-2, *Road vehicles — End-of-life activation of on-board pyrotechnic devices — Part 2: Communication requirements*

[16] ISO 27145-2, *Road vehicles — Implementation of World-Wide Harmonized On-Board Diagnostics (WWH-OBD) communication requirements — Part 2: Common data dictionary*

---

1) To be published.

2) Under preparation.

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

[17] ISO 27145-3, *Road vehicles — Implementation of World-Wide Harmonized On-Board Diagnostics (WWH-OBD) communication requirements — Part 3: Common message dictionary*

[18] SAE J1939:2011, *Serial Control and Communications Heavy Duty Vehicle Network — Top Level Document*

[19] SAE J1939-73:2010, *Application Layer — Diagnostics*

[20] ANSI/IEEE Std 754-1985, *IEEE Standard for Binary Floating Point Arithmetic*

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

© ISO 2013 – All rights reserved

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18
SHW3SZH 2013-09-29 NormMaster

SHW3SZH 2013-09-29 NormMaster

Normen-Download-Beuth-Hoser & Mende KG-KdNr.95670-LfNr.6222688001-2013-05-24 13:18

ICS 43.180

Price based on 392 pages

© ISO 2013 – All rights reserved