# QJ/GAC

# 企 业 技 术 标 准

# Diagnostic Specification

（Revision: 2.3）

2013-06-10 发布 2013-06-20 实施

广汽集团汽车工程研究院 发布

# Contents

**Version History**

| Version | Editor | Date | Description of changes |
|---|---|---|---|
| 0.5 | - | 2008-09-12 | First Draft (reviewed) |
| 0.6 | Huanglifang | 2009-01-12 | Changed diagnostic protocol from KWP2000 to UDS |
| 0.7 | Huanglifang | 2009-01-19 | Textual changes after the change from KWP2000 to UDS |
| 0.8 | Huanglifang | 2009-01-20 | Added some detailed information |
| 0.85 | Huanglifang | 2009-01-30 | Added chapters about the Fault Memory |
| 0.90 | Huanglifang | 2009-02-13 | Corrected minor errors in interfaces, added DTC priority |
| 0.91 | Huanglifang | 2009-03-16 | - (5.3) Added the sentence "The functional request CAN identifier 7DF …"<br>- (6.2.1) Renamed P2 to P2server and P2Ext to P2*server<br>- (6.2.3) Renamed S2timer to S3server<br>- (6.2.3) Added the sentence "When receiving or transmitting any diagnostic messages, including $3E service, the timer will reset."<br>- (6.4.3) Added the sentence "A DTC can just be clearead by tester with service $14."<br>- (7.3.3) Added the sentence "A DTC can just be clearead by tester with service $14."<br>-(6.5.1.1) Added the DTCFormat ISO15031-6DTCFormat to the protocol service $19 01 |
| 0.92 | Huanglifang | 2009-03-31 | - (6.1.2) Enabled service ReadDTCInformation – ReportSupportedDtc by using functional addressing<br>- (6.1.4) Modifed Session state model by leaving programming session |
| 0.93 | Huanglifang | 2009-04-03 | - (6.6.3) Introduced value of 5 as a maximum number of data identifiers to be read with a single request.<br>- (6.8.1) (6.8.2) (6.8.3) Introduced security access level 2 |
| 0.94 | Huanglifang | 2009-04-17 | - (7.3.7) Changed naming of DTC status bis 1 and 6 from "… this monitoring cycle" to "… this operation cycle"<br>- (5.5.2) Added a bullet regarding falling back to inactive, if diagnostic request is completed on default session. |

| | | | |
|---|---|---|---|
| | | | - (5.5.2) Added two figures to describe BCAN ECU sleep handling. |
| 1.00 | Huanglifang | 2009-04-30 | - (Annex D) Added routine control identifier 0xFF02<br>- (6.2.3) Modified session state model by adding EcuReset services<br>- (6.8.3) Added sentence about waiting 10 seconds before acception first RequestSeed<br>- (6.10) Changed periodic messages of service $2A to response type-1<br>- (6.5.1) Added chapters 6.5.1.6 and 6.5.1.7 to provide fault memory services $19 12 and $19 13<br>- (2.) Removed reference to document [GAC8]<br>- (7.3.7) Added reference to examples of Annex in UDS specification<br>- (6.8.3) Modified SecurityState model by adding EcuReset ($10) and $10 01 to the diagram<br>- (6.5.1.1) Renamed status mask of positive response from DTCStatus to DTCStatusAvailablityMask<br>- (6.5.3) Added recommendations of services $19 04 and $19 06 to a single Snapshot/ExtendeDataRecord<br>- (7.6) Changed the restriction of variant coding string length to "... is not restricted".<br>- (Annex B) Corrected naming and description of data identifiers 0xF18C and 0xF18B<br>- (5.3) Added sentence regarding network layer error handling |
| 1.1 | Huanglifang | 2009-10-28 | - (7.3.7) Added sentence regarding recommendation of DTCStatusAvailabilityMask<br>- (Annex B) Replacement of Identifier 0xF183 of 0xF184 for bootloader fingerprint<br>- (5.4.2) Added hint on parameters for emission related systems<br>- (5.4.3) Added chapter regarding application layer error handling.<br>- (Annex B) Modified data identifiers: 0xF154, 0xF182, 0xF187, 0xF188, 0xF189, 0xF18C, 0xF190 and 0xF191 according to agreed proposal.<br>- (6.3.2) Modified two paragraphs regarding sending positive response message of DiagnosticSessionCon- |

| | | | |
|---|---|---|---|
| | | | trolService by specifying primitive.<br><br>- (6.5.2), (6,5,3) Added description about services 0x19 0x12 and 0x19 0x13<br><br>- (6.8.3) Removed typing error from "[…] the ECU and shall wait […]" to "[…] the ECU shall wait […]"<br><br>- (6.12.1.1) Added convention index M1/U1 to controlSate<br><br>- (6.15.1) Changed request message format<br><br>- (6.16.1) Predefined byte positions as a result of using just formatIdentifier 0x24<br><br>- (6.16.2) Removed dangling reference to an unknown chapter 0.<br><br>- (6.22.3) Corrected typing error, Added hint.<br><br>- (7.3.2) Changed typing error from "afty ritics" to "safty critics"<br><br>- (7.3.5) Removed content and added reference to [SS]<br><br>- (7.3.7) Consistent usage of "Operation cycle" instead of "Monitoring cycle"<br><br>- (7.6) Corrected reference within text to figure 13<br><br>- (6.5.1.1), (6.5.1.2), (6.5.1.5), (6.5.1.6), (6.5.1.7) Changed value range of DTCStatusAvailabilityMask in positive responses from "00-FF" to "01-FF" as value zero does not make sense. (Result of internal review)<br><br>- (6.5.1.3) Added footnote about usage of DTC Groups in the request of service 0x19 0x04. (Result of internal review)<br><br>- (6.5.1.3) Corrected typing error in Byte positive response ("StausOfDTC" → "StatusOfDTC") (Result of internal review)<br><br>- (6.5.1.6) Corrected naming of byte 3 in positive response from DTCStatusMask to DTCStatusAvailabilityMask (Result of internal review)<br><br>- (7.3.6) Removed predefined time to check cyclically "(at least every 2 seconds)", as some checks need to be executed more or less frequently (e.g. corrupt NVRAM) (Result of internal review)<br><br>- (7.3.7) Pending DTC: Removed the requirement about "The status shall only be updated if the test runs and completes". (Result of internal review) |

| | | | |
|---|---|---|---|
| | | | - (6.8.3) Complete rework on SecurityAccess implementation rules |
| 2.0 | Huanglifang | 2010-05-05 | - Updated Table 8 and Table 9; |
| 2.1 | Huanglifang | 2011-08-01 | - (5.3.1) Add sentence:"GAC just supports half-duplex. With half-duplex, point-to point communication between two nodes is only possible in one direction at one time."<br>-Annex B, Update some information of DID Table. |
| 2.2 | Lijitai | 2012-11-15 | - (6.8.1.1) Add a sentence 'The length of seed shall be 4 bytes'.<br>- (6.8.1.2) Add a sentence 'The length of key shall be 4 bytes'. |
| 2.3 | Lijitai | 2013-06-10 | - (2.) Remove [GAC8]GC internal document – CAN ID assignment<br>- (2.) Add QJGAC 1150.003-2010 PIN definition for diagnostic connector<br>- (2.) Add QJGAC 1150.004-2010 Diagnostic CAN ID Assignment rule<br>- (5.1.1) Update Figure-1<br>- (5.1.2) Update Table-1<br>- (5.3) The functional request CAN identifier 7DF shall be used for HCAN, too.<br>- (5.3.1.2) Parameters used for PCAN can be also used for HCAN.<br>- (5.4.2) Parameters used for PCAN can be also used for HCAN.<br>- (5.5.1) Remove 'For PCAN is no network management in use'.<br>- (5.5.3) Add this section.<br>- (6.8.1) Add a sentence 'If then, the service SecurityAccess-SendKey is received, the server shall respond with NRC 0x24'.<br>- (6.8.1.1) Remove NRC 0x35.<br>- (6.8.3) Update the figure about FAA Flag.<br>- Annex B: Add note:The table just list part of DID information.In case of conflicts, the information of <GAC Diagnostic parameters table> take priority over that of this table. |

# 1 Scope of this Document

This specification contains the recommended practice for all ECUs of GAC, which support diagnostics. It is intended to provide guidance and requirements to architects, system designers, application developers, and suppliers that are to deliver diagnostic implementations to GAC production vehicle programs.

This document focuses on CAN based systems only. It does not replace the existing normative documents regarding diagnostics, but it adds additional requirements and restrictions to the contents of the standards. In case of conflicts this document takes priority over the normative documents.The content of this specification is mandatory for all ECU supporting diagnostics. Any deviation requires the approval of GAC and shall be documented within the corresponding ECU diagnostic specification.

The international standards for Emission related systems apply. In case of conflict, the legal requirements take priority over this specification.

# 2 Related Documents

The following referenced documents are essential 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.

[ISO1] ISO 14229-1: Road vehicles — Unified diagnostic services (UDS) — Part 1:Specification and requirements (2005). ISO International Organization for Standardization. ISO TC 22/SC 3/WG 1

[ISO2] ISO/DIS 15765-2: Road vehicles – Diagnostics on CAN – Part 2: Network layer services (2000) ISO International Organization for Standardization. ISO/TC 22/SC 3/WG 1/TF 2 N

[ISO3] ISO/DIS 15765-3: Road vehicles – Diagnostics on CAN – Part 3: Implementation of Diagnostic Services (2005) ISO International Organization for Standardization. ISO/TC 22/SC 3/WG 1/TF 2 N (2005)

[ISO4] ISO 15765-4: Road vehicles – Diagnostics on Controller Area Network (CAN) – Part 4: Requirements for emissions-related systems

[ISO5] ISO 15031-5: Road vehicles – Communications between vehicle and external equipment for emission-related diagnostics – Part 5: Emissions-related diagnostic services

[ISO6] ISO 15031-7: Road vehicles -– Communication between vehicle and external equipment for emissions-related diagnostics – Part 7: Data link security document.

[ISO7] ISO 15031-3: Road vehicles – Communication between vehicle and external equipment for emission-related diagnostics – Part 3: Diagnostic connector and related electrical circuits, specification and use

QJGAC 1150.003-2010 PIN definition for diagnostic connector

QJGAC 1150.004-2010 Diagnostic CAN ID Assignment rule

[DSS] Diagnostic System Specification

# 3 Terms, Acronyms and Abbreviations

## 3.1 Terms

| Term | Definition |
|---|---|
| Client | The function that is part of the tester and that makes use of the diagnostic services. A tester normally makes use of other functions such as data base management, specific interpretation, human-machine interface. |
| Diagnostic Data | Data that is located in the memory of an electronic control unit which may be inspected and/or possibly modified by the tester (diagnostic data includes analogue inputs and outputs, digital inputs and outputs, intermediate values and various status information). |
| Diagnostic Service | An information exchange initiated by a client in order to require diagnostic information from a server or/and to modify its behavior for diagnostic purpose. |
| Diagnostic Session | The level of diagnostic functionality provided by the ECU. |
| Diagnostic Trouble Code | A numerical common identifier for a fault condition identified by the on-board diagnostic system. |
| Electronic Control Unit | An Electronic Control Unit contains at least one server. |
| Identifier | Identifiers are numeric logical abstractions of a data element residing in the ECU. Logical addressing of parameters promotes global references of data elements thereby allowing identifiers to retain their original defined functions independent of changes to actual physical memory addresses.<br>Whereas diagnostic services pertain to general functional requests for data, identifiers are used to uniquely identify data elements in the ECU (i.e. sensor data, fault data, configuration, software variables, etc.). Identifiers commonly point to memory locations in RAM, ROM or EEPROM, and are logically grouped together by type to permit further expansion without compromising Identifier grouping. |
| Negative Response Code | This hexadecimal value identifies the specific condition the ECU was not able to respond positively. |
| Pending | The pending status of a failure is defined as a Test Result having reported a “Failed” for this failure during the current and/or the last completed driving cycle. Once the test has reported a “Passed” condition for a complete operation cycle of this failure the pending status is reset. |
| Security | The security access method used in this document satisfies the requirements for tamper protection as specified in [ISO6]. |
| Server | A function that is part of an electronic control unit and that provides the diagnostic services. Within this document, wording differentiates between the server (i.e. the function) and the electronic control unit so that this specification remains independent from the implementation. |
| Sub-function | The first diagnostic service parameter, which allows the selection of a specific functionality for the specific service. |
| Tester | A 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 operators (e.g. a scan tool dedicated to garage mechanics or a test tool dedicated to assembly plant agents). The tester is also referenced as the client. |

## 3.2 Acronyms and Abbreviations

| Acronym/ Abbreviation | Meaning |
|---|---|
| CAN | Controller Area Network |
| BCAN | Low Speed CAN (ISO 11519) |
| CVT | Convention, refer to chapter 4. |
| DCAN/ Diagnostic CAN | Central diagnostic connection for vehicle diagnostics. Implemented as High Speed CAN (ISO 11898). |
| DTC | Diagnostic Trouble Code |
| EC | Electromagnetic Compatibility |
| ECU | Electronic Control Unit |
| EOL | End of Line |
| HCAN | Hybrid CAN |
| ID | Identifier |
| OBD | On Board Diagnostics. Emission related diagnostics. |
| PCAN | High Speed CAN (ISO 11898) |
| UDS | Unified Diagnostic Services (ISO 14229) |

# 4 Document Conventions

In this document, the following terminology applies:

- „Shall" expresses an obligatory/mandatory requirement.
- „Should" expresses a recommendation or an advice.
- „Must" expresses a legal or normative requirement.
- "Will" expresses a precautionary consideration or an additional / optional feature.
- „May" expresses a permitted practice / method, not to be considered as a requirement.

In this document, the following conventions (CVT) are used for service parameters:

M („mandatory"): The parameter has to be present in the telegram.

C („conditional"): The parameter may be present in the telegram, based on certain criteria (e.g. sub-function/parameters within the telegram).

U („User optional"): The parameter may or may not be present, depending on dynamic usage of the user.

Identifiers added in brackets "( )" are in hexadecimal notation unless otherwise stated.

# 5 Diagnostic Communication

Within this chapter the communication regarding network and diagnostic layer are described.

## 5.1 Vehicle Network

### 5.1.1 Vehicle Topology

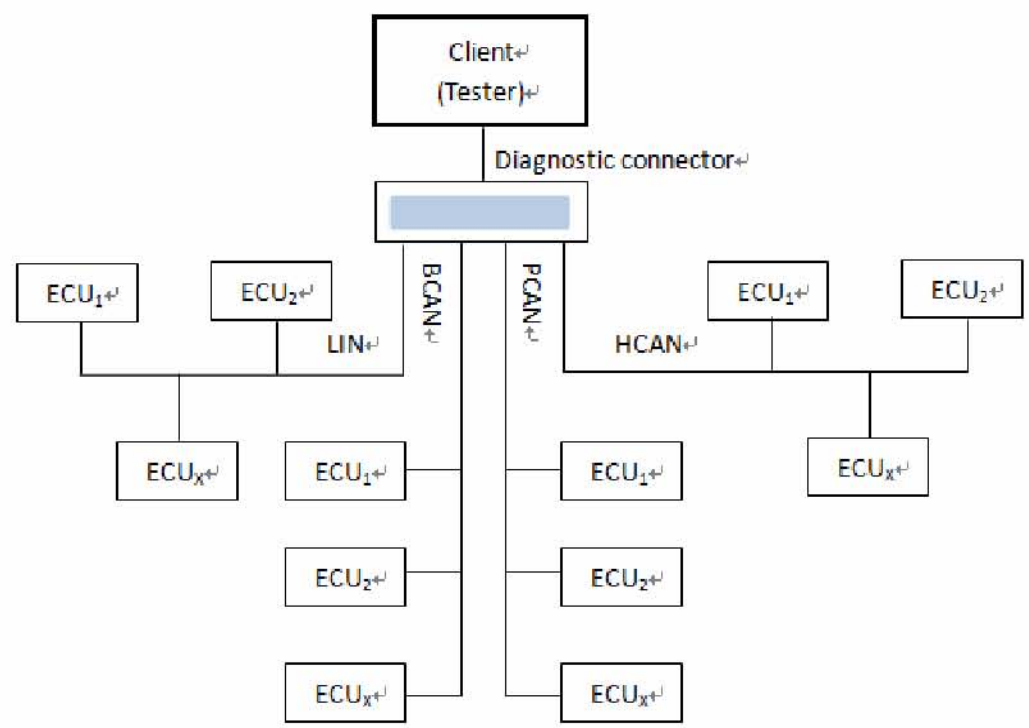



*Figure -1: Vehicle Topology (schematic)*

The Diagnostic Connector is the vehicle access of the external diagnostic test system. The PCAN has to fulfill all legal requirements for emission related diagnostics [ISO5].

### 5.1.2 Diagnostic Connector

The Diagnostic Connector is the vehicle access of the external diagnostic test system. The Diagnostic Connector is connected directly to the according CAN.

The pin assignment is according to [ISO7].

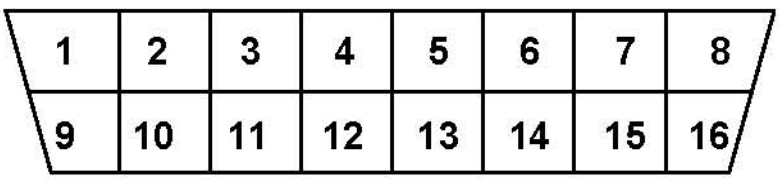



*Figure -2: OBD Connector Pinning*

| Pin | Description |
|---|---|
| 1 | CAN – H for HCAN |
| 2 | Reserved |
| 3 | CAN – H for BCAN |
| 4 | Chassis ground |
| 5 | Signal ground |
| 6 | CAN – H for PCAN |
| 7 | K Line |
| 8 | Reserved |
| 9 | CAN – L for HCAN |
| 10 | Reserved |
| 11 | CAN – L for BCAN |
| 12 | Reserved |
| 13 | Reserved |
| 14 | CAN – L for PCAN |
| 15 | LIN |
| 16 | Battery voltage (terminal 30) |

*Table -1: OBD Connector Pin Assignment*

For further details, please refer to [ISO7].

## 5.2 Addressing Method

### 5.2.1 Enhanced Diagnostics

Both physical and functional addressing shall be supported by the ECU.

Physical addressing is a peer-to-peer connection between the diagnostic tester and an ECU (1:1). It is used for data exchange. There is always a response on a physical request.

Functional addressing is a broadcast connection between the diagnostic tester and a group of ECU (1:n). According to the Transport Protocol specification [ISO2], functional requests are always single frame messages. In this specification, functional addressing is only used for setting and maintaining network wide states. The response of the ECU depends on the "suppress positive response message indication bit", even if the service is requested functionally.

Flow Control frame shall always be physical address

The communication schemes of physical address and function address shall meet the requirements as specified in [ISO1].

### 5.2.2 Emission related Diagnostics

OBD requires functional addressing. For details, please refer to the corresponding standard specification [ISO4].

## 5.3 CAN Network layer parameters

The addressing scheme and the CAN Identifier assignement are available in [GAC8]

The functional request CAN identifier 7DF shall be used for BCAN, PCAN and HCAN.

The network layer error handling shall meet the requirements as specified in [ISO2]

### 5.3.1 Parameters

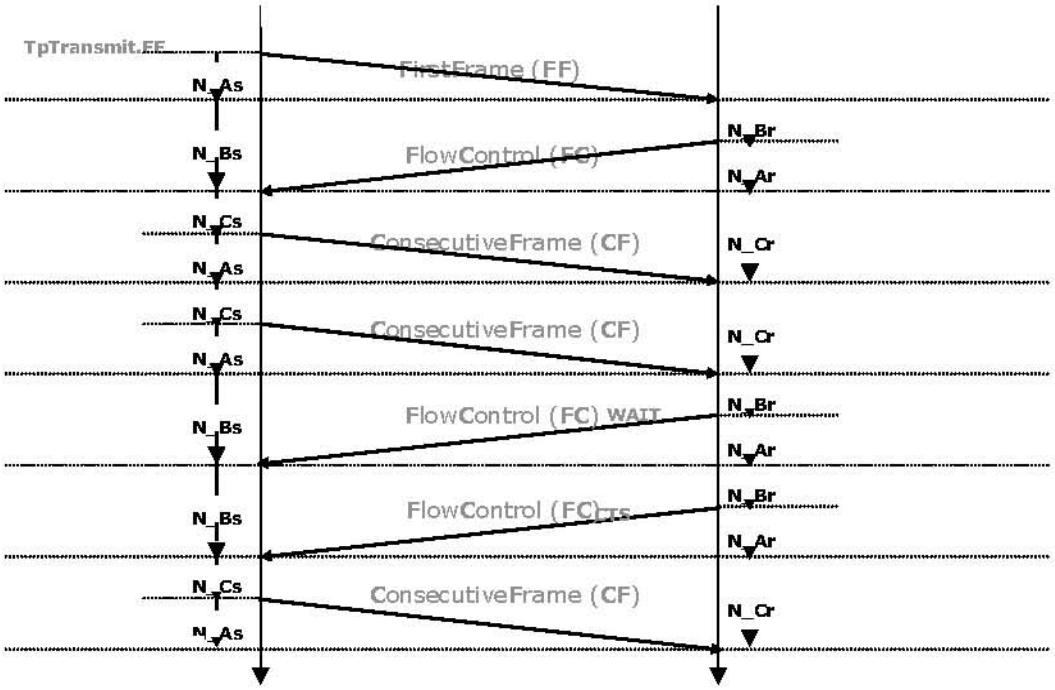



*Figure -3: Timing between tester and ECU*

The Maximum number of 'FC.Wait frame transmissions' (N_WFTmax) shall be set to zero (0). 'CAN frame data padding' shall be used; therefore the CAN DLC shall be set always to 8. It is recommended to pad the unused bytes with a specific value of 0xAA to avoid stuff bits.

All ECUs shall be able to receive subsequent diagnostic CAN messages.

GAC just supports half-duplex. With half-duplex, point-to point communication between two nodes is only possible in one direction at one time.

#### 5.3.1.1 Parameters for BCAN ECUs

| Parameter | Symbol | Application mode[1] | Bootloader mode[2] |
|---|---|---|---|
| Block Size | BS | 8 | 8 |
| Separation time | $ST_{min}$[3] | 20 | 0 |

| Parameter | Symbol | Timeout | Performance Requirement | Unit |
|---|---|---|---|---|
| Time until CAN message has to be transmitted | N_As/N_Ar | 70 | n/a | ms |
| Time for transmitting of the next flow control | N_Br | n/a | <70 | ms |
| Time for reception of the next flow control | N_Bs | 150 | n/a | ms |
| Time until transmission of the next consecutive frame | N_Cs | n/a | <70 | ms |
| Time until reception of the next consecutive frame | N_Cr | 150 | n/a | ms |

*Table -2: Requirements of network layer parameters for BCAN*

#### 5.3.1.2 Parameters for PCAN and HCAN ECUs

| Parameter | Symbol | Application mode | Bootloader mode |
|---|---|---|---|
| Block Size | BS | 8 | 8 |
| Separation time | $ST_{min}$ | 20 | 0 |

| Parameter | Symbol | Timeout | Performance Requirement | Unit |
|---|---|---|---|---|
| Time until CAN message has to be transmitted | N_As/N_Ar | 70 | n/a | ms |
| Time for transmitting of the next flow control | N_Br | n/a | < 70 | ms |
| Time for reception of the next flow control | N_Bs | 150 | n/a | ms |
| Time until transmission of the next consecutive frame | N_Cs | n/a | <70 | ms |
| Time until reception of the next consecutive frame | N_Cr | 150 | n/a | ms |

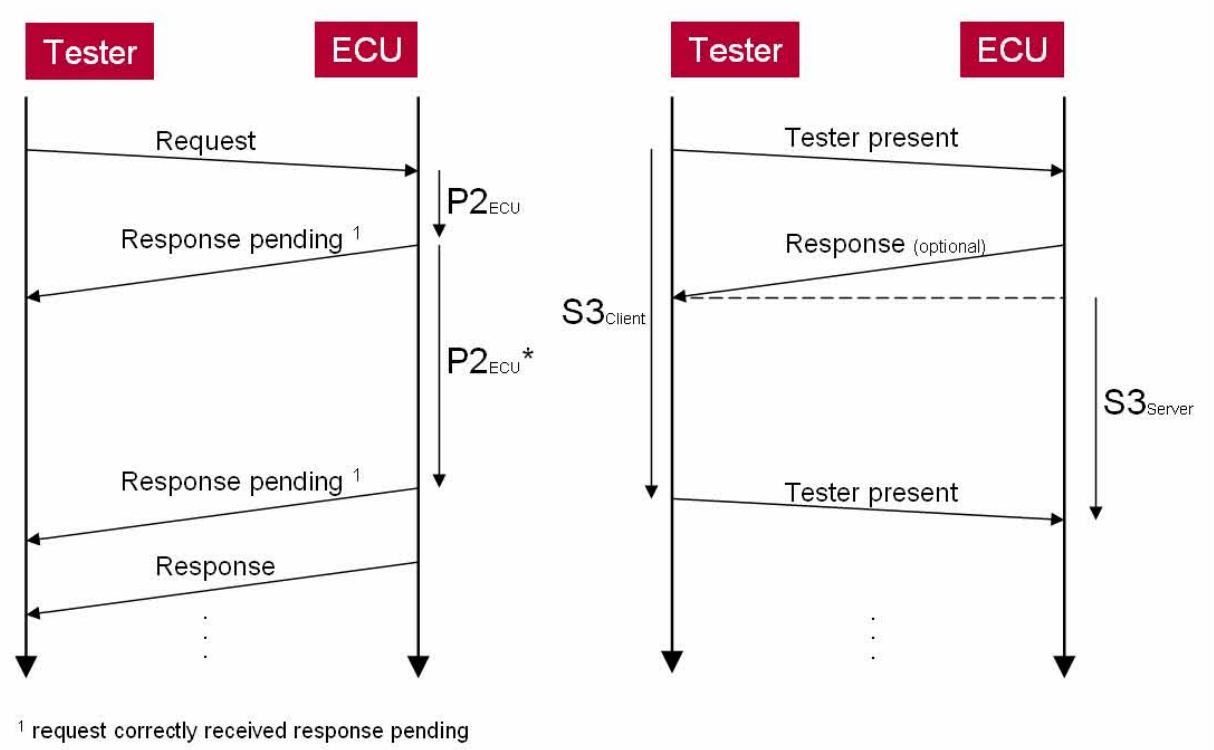
*Table -3: Requirements of network layer parameters for PCAN and HCAN*

[1] Application mode: The ECU is not running in the bootloader.

[2] Bootloader mode: The ECU is running in the bootloader, typically in the programming session.

[3] The $ST_{min}$ value unequal to zero shall prevent peak loads on the network and not be used to handle reception limitations in the ECU.

## 5.4 Diagnostic layer parameters




[1] request correctly received response pending

*Figure -4: Diagnostic layer timings between tester and ECU*

After the tester receives a diagnostic response message including Negative Response Code 0x78 (Request correctly received – response pending) the response timing is changed. Thus extended response timing ($P2_{Ecu}$*) is applicable.

### 5.4.1 BCAN

| Parameter | Symbol | Min | Max | Timeout | Unit |
|---|---|---|---|---|---|
| Time between client (tester) request and server (ECU) response | $P2_{server}$ | 0 | 50 | n/a | ms |
| | $P2_{client}$ | n/a | n/a | 150 | ms |
| Enhanced timeout for the client to wait after the reception of a negative response message with response code 78 hex | $P2^*_{server}$ | 0 | 2000 | n/a | ms |
| | $P2^*_{client}$ | n/a | n/a | 5000 | ms |
| Time between request and the next request from the tester – physical | $P3_{client\_phys}$ [4] | $P2_{Server}$ | n/a | n/a | ms |
| Time between consecutive request from the tester – functional | $P3_{client\_func}$ | $P2_{server\_max}$ | n/a | n/a | ms |

*Table -4: Requirements of diagnostic layer parameters for BCAN*

[4] This timing parameter applies to any physically addressed request message in case there is no response required to be transmitted by the server (suppressPosRspMsgIndicationBit = TRUE).

| Parameter | Symbol | Min | Nominal | Timeout | Unit |
|---|---|---|---|---|---|
| Session timeout; after timeout return to default-session | $S3_{server}$ | n/a | n/a | 5000 | ms |
| Time for transmitting next TesterPresent to keep non-default session | $S3_{client}$ | 0 | 2000 | 4000 | ms |

*Table -5: Requirements of session parameters for BCAN*

### 5.4.2 PCAN and HCAN

| Parameter | Symbol | Min | Max | Timeout | Unit |
|---|---|---|---|---|---|
| Time between client (tester) request and server (ECU) response | $P2_{server}$ | 0 | 50 | n/a | ms |
| | $P2_{client}$ | n/a | n/a | 150 | ms |
| Enhanced timeout for the client to wait after the reception of a negative response message with response code 78 hex | $P2^*_{server}$ | 0 | 2000 | n/a | ms |
| | $P2^*_{client}$ | n/a | n/a | 5000 | ms |
| Time between two consecutive physical requests in case no response is expected | $P3_{client}$[5] | $P2_{Server}$ | n/a | n/a | ms |
| Time between two consecutive functional requests from the tester in case no response is expected | $P3_{client}$ | $P2_{server\_max}$ | n/a | n/a | ms |

*Table -6: Requirements of diagnostic layer parameters for PCAN and HCAN*

| Parameter | Symbol | Min | Nominal | Timeout | Unit |
|---|---|---|---|---|---|
| Session timeout; after timeout return to default-session | $S3_{server}$ | n/a | n/a | 5000 | ms |
| Time for transmitting next TesterPresent to keep non-default session | $S3_{client}$ | 0 | 2000 | 4000 | ms |

*Table -7: Requirements of session parameters for PCAN and HCAN*

Hint:

For emission related systems (ISO-15765-4) the according timings could to be applied:

- ECU: $ST_{min}$=20, BS = 8
- Tester: $ST_{min}$=0, BS = 0

### 5.4.3 Application layer error handling

The application layer error handling shall meet the requirements as specified in 15765-3.

[5] This timing parameter applies to any physically addressed request message in case there is no response required to be transmitted by the server (suppressPosRspMsgIndicationBit = TRUE).

## 5.5 Availability of diagnostic server

As soon as an ECU is powered and communication is possible (e.g. sending/receiving CAN data is possible) it shall be diagnosable. An ECU must be at least diagnosable if ignition is on (clamp 15).

An ECU should also be diagnosable, if it is not connected to any system. Missing peripheral or CAN signals may be replaced by default values.

If an ECU is connected to more than one network, it shall be diagnosable only on one network. The network with the lowest amount of routing relations to the Diagnostic Connector shall be chosen prior to the network with the highest bandwidth[6].

### 5.5.1 ECUs connected to PCAN

All ECUs connected to the PCAN Bus start booting when ignition is set to “on” (clamp 15). After the boot time and CAN communication is possible the ECUs must be diagnosable.

### 5.5.2 ECUs connected to BCAN

BCAN ECUs are connected to permanent battery supply. Therefore, network management is required. The following requirements apply:

- While any diagnostic service is in progress, the network management Sleep Indication bit shall not be set to “Sleep”.
- If a diagnostic service is completed, the network management Sleep Indication bit may be set to “Sleep”, if possible.
- Receiving a diagnostic request the Sleep Indication bit shall be set actively to “awake” if it was previously set to “Sleep”.
- While non-default sessions are active, the network management Sleep Indication bit shall not be set to “Sleep”.
- After switching to the Default Diagnostic session the Sleep Indication bit may be set immediately to “Sleep”.
- Gateways shall not set Sleep Indication bit before a minimum period of S3server has expired after routing of a diagnostic message.

The following two figures give a sample on handling the Sleep Indication but in default and in extended session of the ECU.

[6] This requirement will be needed in the future if a gateway is used.

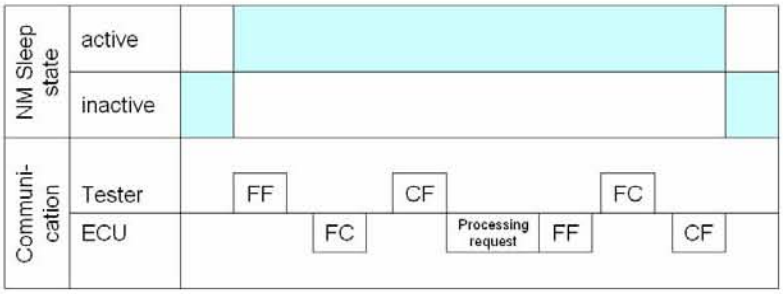



*Figure -5: Sleep Indication bit handling in default session*

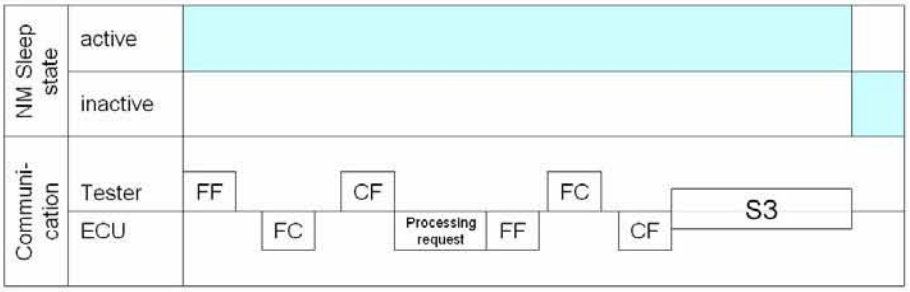



*Figure -6: Sleep Indication bit handling in extended/programming session*

### 5.5.3 ECUs connected to HCAN

All ECUs connected to the HCAN Bus start booting when ignition is set to “on” (clamp 15). After the boot time and CAN communication is possible the ECUs must be diagnosable.

## 5.6 Client Limitations

Multiple clients shall not be supported in the same network.

A client can have physical communication with up to **2** ECUs simultaneous.

A gateway will not be used.

For all diagnostic services using an addressAndLengthFormatIdentifier, the value 0x24 shall be used for it. This means a 4 byte value for the memoryAddress and a two byte value for the memorySize. Any deviation requires the approval of GAC.

The only exception is service RequestDownload (34). For this service the allowed value is restricted to 0x44.

# 6 Diagnostic Services

## 6.1 General

This chapter is considered to give rules regarding the implementation of diagnostic services.

### 6.1.1 Overview

The following table shows all of the UDS diagnostic services defined within this document and which are supported by GAC. The list has been sorted according to the Service Identifier (SID) assigned to each diagnostic service. The convention (Cvt) column specifies whether the service implementation is mandatory or user optional.

| SID (hex) | Service Name | Cvt |
|---|---|---|
| 10 | DiagnosticSessionControl | M |
| 11 | EcuReset | M |
| 14 | ClearDiagnosticInformation | M |
| 19 | ReadDTCInformation | M |
| 22 | ReadDataByIdentifier | M |
| 23 | ReadMemoryByAddress | U |
| 27 | SecurityAccess | U |
| 28 | CommunicationControl | M |
| 2A | ReadDataByPeriodicIdentifier | U |
| 2C | DynamicallyDefineDataIdentifier | U |
| 2E | WriteDataByIdentifier | U |
| 2F | InputOutputControlByIdentifier | U |
| 31 | RoutineControl | U |
| 34 | RequestDownload | U |
| 35 | RequestUpload | U |
| 36 | TransferData | U |
| 37 | RequestTransferExit | U |
| 3D | WriteMemoryByAddress | U |
| 3E | TesterPresent | M |
| 83 | AccessTimingParameter [7] | U |
| 85 | ControlDTCSetting | M |

*Table -8: Service Summary Table*

Note: The services which are marked “M” in the convention (Cvt) column are mandatory , but not all the subfunction and parameters of these sevices are mandatory for all the ECUs.The

[7] This diagnostic service is not supported by the transport protocol [ISO3]. It is just relevant, if used by bus system, different to CAN.

detailed ECU supported subfunction and parameters will be defined in the “Diagnostic parameter Table”.

All diagnostic services providing a “suppress positive response message indication” bit used by GAC must support the “suppress positive response message” functionality as specified in [ISO1].

### 6.1.2 Addressing Restrictions

The ECU shall support physical and functional addressing (for details, please refer to chapter 5.2.1).

The following table points out which services support which addressing.

| SID (hex) | Service Name | phys addr. | Func addr. |
|---|---|---|---|
| 10 | DiagnosticSessionControl | O | O 1) |
| 11 | EcuReset | O | O |
| 14 | ClearDiagnosticInformation | O | O |
| 19 | ReadDTCInformation | O | O |
| 22 | ReadDataByIdentifier | O | O |
| 23 | ReadMemoryByAddress | O | |
| 27 | SecurityAccess | O | |
| 28 | CommunicationControl | O | O |
| 2A | ReadDataByPeriodicIdentifier | O | |
| 2C | DynamicallyDefineDataIdentifier | O | |
| 2E | WriteDataByIdentifier | O | |
| 2F | InputOutputControlByIdentifier | O | |
| 31 | RoutineControl | O | |
| 34 | RequestDownload | O | |
| 35 | RequestUpload | O | |
| 36 | TransferData | O | |
| 37 | RequestTransferExit | O | |
| 3D | WriteMemoryByAddress | O | |
| 3E | TesterPresent | O | O |
| 83 | AccessTimingParameter | O | |
| 85 | ControlDTCSetting | O | O |

*Table -9: Service execution restrictions in diagnostic sessions*

**Explanation**

O = Addressing method is supported by this service

1) Depending on the session parameter. For default and extended session, the support of functional addressing is mandatory. (Client requirement)

### 6.1.3 Explanation of the sessions

#### 6.1.3.1 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 default-Session 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.

#### 6.1.3.2 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 (11 hex) service initiated by the client, a DiagnosticSessionControl (10 hex) service with sessionType equal to defaultSession, or a session layer timeout in the server.

In case the server runs in the boot software when it receives the DiagnosticSessionControl (10hex) service with sessionType equal to defaultSession or a session layer timeout occurs and 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.).

#### 6.1.3.3 ExtendedDiagnosticSession

This diagnosticSession can e.g. 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.

### 6.1.4 Session Restrictions

A diagnostic session enables a specific set of diagnostic services and/or functionality in the server. Consequently, all other services are not enabled and shall not be executed within this diagnostic session (refer to chapter 6.2

The following table shows the dependencies of a diagnostic service to the active diagnostic session. The support of the diagnostic sessions “default”, “extended” and “programming” are mandatory[8] for all GAC ECUs.

| SID (hex) | Service Name | default | extended | program. |
|---|---|---|---|---|
| 10 | DiagnosticSessionControl | O | O | O |
| 11 | EcuReset | O | O | O |
| 14 | ClearDiagnosticInformation | O | O | |
| 19 | ReadDTCInformation | O | O | |
| 22 | ReadDataByIdentifier | O 1) | O | O |
| 23 | ReadMemoryByAddress | O 2) | O | |
| 27 | SecurityAccess | | O | O |
| 28 | CommunicationControl | | O | |
| 2A | ReadDataByPeriodicIdentifier | | O | |
| 2C | DynamicallyDefineDataIdentifier | O | O | |
| 2E | WriteDataByIdentifier | O 1) | O | O |
| 2F | InputOutputControlByIdentifier | | O | |
| 31 | RoutineControl | O 3) | O | O |
| 34 | RequestDownload | | | O |
| 35 | RequestUpload | | | O |
| 36 | TransferData | | | O |
| 37 | RequestTransferExit | | | O |
| 3D | WriteMemoryByAddress | O 2) | O | O |
| 3E | TesterPresent | O | O | O |
| 83 | AccessTimingParameter | | O | |
| 85 | ControlDTCSetting | | O | |

*Table -10: Service execution restrictions in diagnostic sessions*

**Explanation**

O = Service may be executed in active session

1) Secured dataIdentifiers require a SecurityAccess service and therefore a non-default diagnostic session.

2) Secured memory areas require a SecurityAccess service and therefore a non-default diagnostic session.

3) Secured routines require a SecurityAccess service and therefore a non-default diagnostic

[8] If an ECU uses services to be run in the default session, and not in the extended or programming session, the session model does not need to be applied in the ECU.

session. A routine that requires to be stopped actively by the client also requires a non-default session.

Please note that a service will only be executed if the current session and the current security level allow the execution.

## 6.2 DiagnosticSessionControl (10)

This service is used by the client to enable different diagnostic sessions in the server(s). A diagnostic session enables a specific set of diagnostic services in the server(s).

### 6.2.1 Message Format

Request:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | RequestServiceIdentifier | M | 10 |
| #2 | Sub-Function = [ DiagnosticSessionType ] | M | 01 – 03 |

Positive Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | PositiveResponseServiceIdentifier | M | 50 |
| #2 | Sub-Function = [ DiagnosticSessionType ] | M | 01 – 03 |
| | P2server[9] [ ] = [ | | |
| #3 | byte#1 | M | 00-FF |
| #4 | byte#2 ] | M | 00-FF |
| | P2*server[10] [ ] = [ | | |
| #5 | byte#1 | M | 00-FF |
| #6 | byte#2 ] | M | 00-FF |

Negative Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | NegativeResponseServiceIdentifier | M | 7F |
| #2 | OriginalRequestServiceIdentifier | M | 10 |
| #3 | NegativeResponseCode | M | (see below) |

[9] Timing P2server value is provided in 1ms resolution.

[10] Timing P2*server value is provided in 10ms resolution.

Negative Response Codes:

| NRC (hex) | Error text |
|---|---|
| 12 | Subfunction not supported/Invalid format |
| 13 | Incorrect message length or invalid format |
| 22 | Conditions not correct or request sequence error |

### 6.2.2 Parameter Options

The DiagnosticSessionType parameter specifies which session shall be started. The explanation of the sessions can be found in 6.1.3.

| Option (hex) | Description |
|---|---|
| 01 | **DefaultSession** |
| 02 | **ProgrammingSession** |
| 03 | **ExtendedDiagnosticSession** |

*Table -11: Definition of SessionType values*

### 6.2.3 Implementation Rules

This service is used by the diagnostic tool to enable different types of diagnostic sessions in a server. In order to execute a diagnostic service the appropriate session has to be started first.

There shall be only one diagnostic session active at a time.

Normal/Default Session (01) shall be enabled automatically by the ECU if no diagnostic session has been requested at power up.

An ECU shall return to Normal/Default Session (01) after timeout of another diagnostic session.

An ECU shall be capable of providing all diagnostic functionality defined for the default diagnostic session under normal operating conditions.

The ECU shall first send a DiagnosticSessionControl Positive Response (50 xx) message before the new session becomes active in the ECU.

A DiagnosticSessionControl Positive Response (50 xx) message shall be returned by an ECU if the diagnostic tool requests a session that is already running. If the ECU has already received the same request message previously and performed the requested operation, the ECU shall continue to perform the current operation (i.e. it is not a change of the session).

An ECU shall remain in its current diagnostic session if it is not able to switch into the requested diagnostic session.

The Tester Present (3E) service shall be used to keep the diagnostic sessions active by retriggering S3server. Also any other service request keeps the diagnostic session alive.

A functional Tester Present (3E) request without response may be sent at any time, even regardless of any other service in progress.

When receiving or transmitting any diagnostic messages, including $3E service, the timer will reset.

Deactivating a session leads to reinitialisation of all activities, started already. The detailed reinitialisation activities shall meet the requirements as specified in [ISO1].

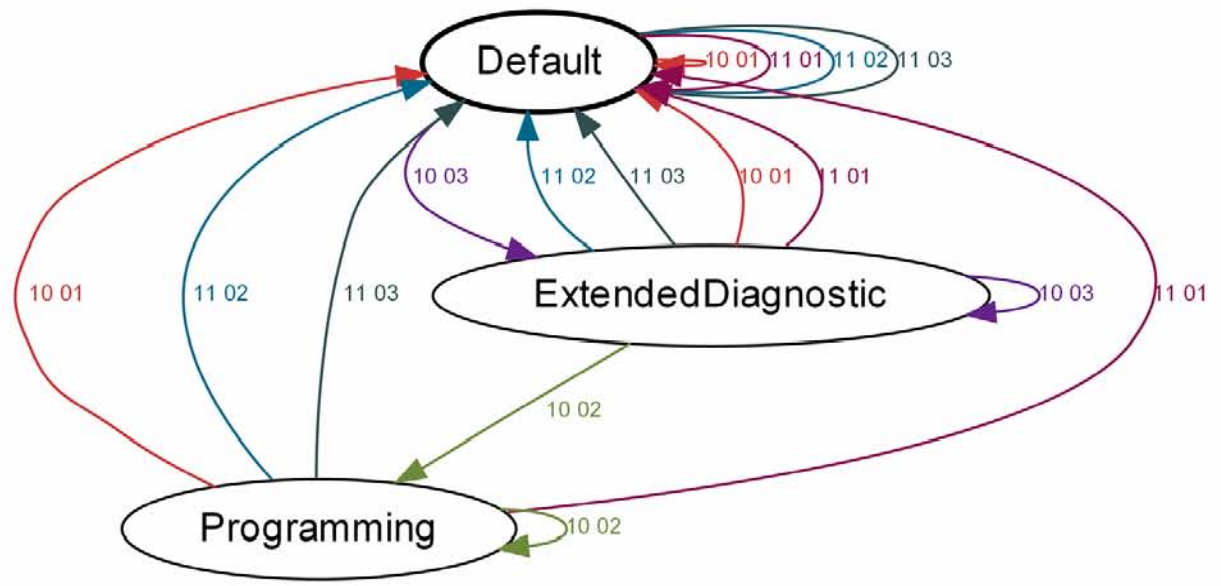



*Figure -7: Session transition diagram*

## 6.3 EcuReset (11)

This service requests the server to effectively perform an ECU reset based on the content of the ResetType parameter value.

### 6.3.1 MessageFormat

Request:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | RequestServiceIdentifier | M | 11 |
| #2 | Sub-Function = [<br>ResetType ] | M | 01-03 |

Positive Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | PositiveResponseServiceIdentifier | M | 51 |
| #2 | Sub-Function = [<br>ResetType ] | M | 01-03 |

Negative Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | NegativeResponseServiceIdentifier | M | 7F |
| #2 | OriginalRequestServiceIdentifier | M | 11 |
| #3 | NegativeResponseCode | M | (see below) |

Negative Response Codes:

| NRC (hex) | Error text |
|---|---|
| 12 | **subFunctionNotSupported**<br>Send if the sub-function parameter in the request message is not supported. |
| 13 | **incorrectMessageLengthOrInvalidFormat**<br>The length of the message is wrong. |
| 22 | **conditionsNotCorrect**<br>This code shall be returned if the criteria for the EcuReset request are not met. |
| 33 | **securityAccessDenied**<br>This code shall be sent if the requested reset is secured and the server is not in an unlocked state. |

### 6.3.2 Parameter Options

The ResetType parameter specifies which kind of reset shall be performed.

| Option (hex) | Description |
|---|---|
| 01 | **hardReset**<br>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. |
| 02 | **keyOffOnReset**<br>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. |
| 03 | **softReset**<br>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. |

*Table -12: Definition of ResetType values*

### 6.3.3 Implementation Rules

The positive response shall be sent before performing the ECU reset.

## 6.4 ClearDiagnosticInformation (14)

This service is used by the client to clear diagnostic information in the server's memory.

For details about the Fault Memory, please refer to chapter 7.3.

### 6.4.1 Message Format

Request:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | RequestServiceIdentifier | M | 14 |
| | GroupOfDTC [ ] = [ | | |
| #2 | groupOfDTCHighByte | M | 00-FF |
| #3 | groupOfDTCMiddleByte | M | 00-FF |
| #4 | groupOfDTCLowByte ] | M | 00-FF |

Positive Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | PositiveResponseServiceIdentifier | M | 54 |

Negative Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | NegativeResponseServiceIdentifier | M | 7F |
| #2 | OriginalRequestServiceIdentifier | M | 14 |
| #3 | NegativeResponseCode | M | (see below) |

Negative Response Codes:

| NRC (hex) | Error text |
|---|---|
| 13 | **incorrectMessageLengthOrInvalidFormat**<br>The length of the message is wrong |
| 22 | **conditionsNotCorrect**<br>This response code shall be used if internal conditions within the server prevent the clearing of DTC related information stored in the server. |
| 31 | **requestOutOfRange**<br>This return code shall be sent if the specified groupOfDTC parameter is not supported. |

### 6.4.2 Parameter Options

The groupOfDTC parameter is defined as follows:

| Option (hex) | Description |
|---|---|
| 000000 | Emissions-related systems |
| 100000 | Powertrain group |
| 400000 | Chassis group |
| 800000 | Body group |
| C00000 | Network communication group |
| FFFFFF | All Groups (all DTC's) |

### 6.4.3 Implementation Rules

The ECU behavior shall be consistent: The DTCs shall be cleared before the positive response is sent. This does not require that the non volatile memory is cleared, but any memory cache has to be.

With the parameter groupOfDTC it is also possible to clear only a specific group of DTCs (e.g. Powertrain) or a specific DTC.

A subsequent call to read out the fault memory shall not retrieve any DTC which has been set before the last execution of ClearDiagnosticInformation.

Even if no DTC was stored, the ECU shall return with a positive response.

There shall be no sequence dependency to any other service. Even if the fault memory was not read, it may be cleared.

The DTC and related information can be clearead by tester with service $14 or after a number of fault passed ignition or test cycle have occurred.

## 6.5 ReadDTCInformation (19)

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. The server shall return both emissions-related and non-emissions-related DTC information. For details about the Fault Memory, please refer to chapter 7.3.

This service allows the client to do the following (sub-functions in brackets)

- The client can retrieve the number of DTCs matching a client defined DTC status mask (0x19 0x01)

- The client can retrieve the list of all DTCs and corresponding statuses which match a client defined DTC status mask (0x19 0x02)

- The client can retrieve DTCSnapshot record(s) associated with a client defined DTC number and DTC snapshot record number ([FF hex] for all records). This functionality is typically used to read freeze frame data from emission related ECUs. (0x19 0x04)
- The client can retrieve the DTCExtendedData records associated with a client defined DTC number and DTCExtendedDatarecord number ([FF] for all records). This functionality is used to read environment data from chrono-stack. (0x19 0x06)
- The client can retrieve the status of all DTCs supported by the server (0x19 0x0A)
- The client can retrieve the number of emissions related OBD DTCs matching a client defined DTC status mask (0x19 0x12)
- The client can retrieve the list of all emissions related OBD DTCs and corresponding statuses which match a client defined DTC status mask (0x19 0x13)

### 6.5.1 Message Format

#### 6.5.1.1 ReadDTCInformation – ReportNumberOfDTCByStatusMask (0x19 0x01)

Request:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | RequestServiceIdentifier | M | 19 |
| #2 | Sub-Function = [<br>ReportNumberOfDTCByStatusMask ] | M | <br>01 |
| #3 | DTCStatusMask[11] | M | 00-FF |

Positive Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | PositiveResponseServiceIdentifier | M | 59 |
| #2 | ReportType = [<br>ReportNumberOfDTCByStatusMask ] | M | <br>01 |
| #3 | DTCStatusAvailabilityMask [11] | M | 01-FF |
| #4 | DTCFormatIdentifier = [<br>ISO15031-6DTCFormat<br>ISO14229-1DTCFormat ] | <br>M<br>M | <br>00<br>01 |
| #5<br>#6 | DTCCount [ ] = [<br>DTCCountHighByte<br>DTCCountLowByte ] | <br>M<br>M | <br>00-FF<br>00-FF |

Negative Response:

| Byte | Name | Cvt | Value |
|---|---|---|---|
| #1 | NegativeResponseServiceIdentifier | M | 7F |
| #2 | OriginalRequestServiceIdentifier | M | 19 |
| #3 | NegativeResponseCode | M | (see below) |

[11] See chapter 7.3.7.

Negative Response Codes:

| NRC (hex) | Error text |
|---|---|
| 12 | **subFunctionNotSupported**<br>This code is returned if the requested sub-function is not supported. |
| 13 | **incorrectMessageLengthOrInvalidFormat**<br>The length of the message is wrong. |
| 31 | **requestOutOfRange**<br>This code is returned if:<br>1. The client specified a DTCMaskRecord / DTCSeverityMaskRecord that was not recognized by the server.<br>2. The client specified an invalid DTCSnapshotRecordNumber /DTCExtendedDataRecordNumber. |

### 6.5.1.2 ReadDTCInformation – ReportDTCByStatusMask (0x19 0x02)

Request:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | RequestServiceIdentifier | M | 19 |
| #2 | Sub-Function = [<br>ReportDTCByStatusMask ] | M | <br>02 |
| #3 | DTCStatusMask[12] | M | 00-FF |

[12] For details see chapter 7.3.7.

Positive Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | PositiveResponseServiceIdentifier | M | 59 |
| #2 | ReportType = [<br>ReportDTCByStatusMask ] | M | <br>02 |
| #3 | DTCStatusAvailabilityMask | M | 01-FF |
| | DTCAndStatusRecord [ ] = [ | | |
| #4 | DTCHighByte#1 | C1 | 00-FF |
| #5 | DTCMiddleByte#1 | C1 | 00-FF |
| #6 | DTCLowByte#1 | C1 | 00-FF |
| #7 | statusOfDTC#1 | C1 | 00-FF |
| #8 | DTCHighByte#2 | C2 | 00-FF |
| #9 | DTCMiddleByte#2 | C2 | 00-FF |
| #10 | DTCLowByte#2 | C2 | 00-FF |
| #11 | statusOfDTC#2 | C2 | 00-FF |
| : | : | : | : |
| #n-3 | DTCHighByte#m | C2 | 00-FF |
| #n-2 | DTCMiddleByte#m | C2 | 00-FF |
| #n-1 | DTCLowByte#m | C2 | 00-FF |
| #n | statusOfDTC#m ] | C2 | 00-FF |
| C1: This parameter is only present if DTC information is available to be reported.<br>C2: This parameter is only present if more than one DTC is to be reported. | | | |

Negative Response:

| Byte | Name | Cvt | Value |
|---|---|---|---|
| #1 | NegativeResponseServiceIdentifier | M | 7F |
| #2 | OriginalRequestServiceIdentifier | M | 19 |
| #3 | NegativeResponseCode | M | (see 19 01) |

#### 6.5.1.3 ReadDTCInformation – ReportDTCSnapshotRecordByDTCNumber (0x19 0x04)

Request:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | RequestServiceIdentifier | M | 19 |
| #2 | Sub-Function = [<br>ReportDTCSnapshotRecordByDTCNumber ] | M | <br>04 |
| | DTCMaskRecord [ ] = [[13] | | |
| #3 | DTCHighByte | M | 00-FF |
| #4 | DTCMiddleByte | M | 00-FF |
| #5 | DTCLowByte ] | M | 00-FF |
| #6 | DTCSnapshotRecordNumber | M | 00-FF |

Positive Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | PositiveResponseServiceIdentifier | M | 59 |
| #2 | ReportType = [<br>ReportDTCSnapshotRecordByDTCNumber ] | M | <br>04 |
| | DTCAndStatusRecord [ ] = [ | | |
| #3 | DTCHighByte | M | 00-FF |
| #4 | DTCMiddleByte | M | 00-FF |
| #5 | DTCLowByte] | M | 00-FF |
| #6 | StatusOfDTC | M | 00-FF |
| #7 | DTCSnapshotRecordNumber #1 | C1 | 00-FE |
| #8 | DTCSnapshotRecordNumberOfIdentifiers #1 | C1 | 00-FF |
| | DTCSnapshotRecord[] #1 = [ | | |
| #9 | DataIdentifier#1 byte #1 (MSB) | C1 | 00-FF |
| #10 | DataIdentifier#1 byte #2 (LSB) | C1 | 00-FF |
| #11 | SnapshotData#1 byte #1 | C1 | 00-FF |
| : | : | : | : |
| #11+(p-1) | SnapshotData#1 byte #p | C1 | 00-FF |
| : | : | : | : |
| #r-(m-1)-2 | DataIdentifier#w byte #1 (MSB) | C2 | 00-FF |
| #r-(m-1)-1 | DataIdentifier#w byte #2 (LSB) | C2 | 00-FF |
| #r-(m-1) | SnapshotData#w byte #1 | C2 | 00-FF |
| : | : | : | : |
| #r | SnapshotData#w byte #m ] | C2 | 00-FF |

[13] The service does not support DTC Groups

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| : | : | : | : |
| #t | DTCSnapshotRecordNumber #x | C3 | 00-FE |
| #t+1 | DTCSnapshotRecordNumberOfIdentifiers #x | C3 | 00-FF |
| | DTCSnapshotRecord[] #x = [ | | |
| #t+2 | DataIdentifier#1 byte #1 (MSB) | C3 | 00-FF |
| #t+3 | DataIdentifier#1 byte #2 (LSB) | C3 | 00-FF |
| #t+4 | SnapshotData#1 byte #1 | C3 | 00-FF |
| : | : | : | : |
| #t+4+(p-1) | SnapshotData#1 byte #p | C3 | 00-FF |
| : | : | : | : |
| #n-(u-1)-2 | DataIdentifier#w byte #1 (MSB) | C4 | 00-FF |
| #n-(u-1)-1 | DataIdentifier#w byte #2 (LSB) | C4 | 00-FF |
| #n-(u-1) | SnapshotData#w byte #1 | C4 | 00-FF |
| : | : | : | : |
| #n | SnapshotData#w byte #u ] | C4 | 00-FF |

C1: 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 (DTCSnapshotRecordNumber unequal to FF hex in the request or only one record is available to be reported if DTCSnapshotRecordNumber is set to FF hex in the request).

C2/C4 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.

C3: The DTCSnapshotRecordNumber and the first dataIdentifier/snapshotData combination in the DTCSnapshotRecord parameter is only present if all records are requested to be reported (DTCSnapshotRecordNumber set to FF hex in the request) and more than one record is available to be reported.

Negative Response:

| Byte | Name | Cvt | Value |
|---|---|---|---|
| #1 | NegativeResponseServiceIdentifier | M | 7F |
| #2 | OriginalRequestServiceIdentifier | M | 19 |
| #3 | NegativeResponseCode | M | (see 19 01) |

#### 6.5.1.4 ReadDTCInformation – ReportDTCExtendedDataRecordsByDTCNumber (0x19 0x06)

Request:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | RequestServiceIdentifier | M | 19 |
| #2 | Sub-Function = [ | M | |
| | ReportDTCExtendedDataRecordByDTCNumber ] | | 06 |
| | DTCMaskRecord[] = [[14] | | |
| #3 | DTCHighByte | M | 00-FF |
| #4 | DTCMiddleByte | M | 00-FF |
| #5 | DTCLowByte ] | M | 00-FF |
| #6 | DTCExtendedDataRecordNumber | M | 00-FF |

Positive Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | PositiveResponseServiceIdentifier | M | 59 |
| | ReportType = [ | | |
| #2 | ReportDTCExtendedDataRecordByDTCNumber] | M | 06 |
| | DTCAndStatusRecord[] = [ | | |
| #3 | DTCHighByte | M | 00-FF |
| #4 | DTCMiddleByte | M | 00-FF |
| #5 | DTCLowByte | M | 00-FF |
| #6 | statusOfDTC ] | M | 00-FF |
| #7 | DTCExtendedDataRecordNumber #1 | C1 | 01-EF |
| | DTCExtendedDataRecord[] #1 = [ | | |
| #8 | ExtendedData #1 byte #1 | C1 | 00-FF |
| : | : | C1 | 00-FF |
| #8+(p-1) | ExtendedData #1 byte #p ] | C1 | 00-FF |
| : | : | : | : |
| #t | DTCExtendedDataRecordNumber #x | C2 | 01-EF |
| | DTCExtendedDataRecord[] #x = [ | | |
| #t+1 | ExtendedData #x byte #1 | C2 | 00-FF |
| : | : | C2 | 00-FF |
| #t+1+(q-1) | ExtendedData #x byte #q ] | C2 | 00-FF |
| C1: The DTCExtendedDataRecordNumber and the extendedData in the DTCExtendedDataRecord pa- | | | |

[14] The service does not support DTC Groups

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| rameter are only present if at least one DTCExtendedDataRecord is available to be reported (DTCExtendedDataRecordNumber unequal to FF hex in the request or only one record is available to be reported if DTCExtendedDataRecordNumber is set to FF hex in the request).<br>C2: The DTCExtendedDataRecordNumber and the extendedData in the DTCExtendedDataRecord parameter are only present if all records are requested to be reported (DTCExtendedDataRecordNumber set to FF hex in the request) and more than one record is available to be reported. | | | |

Negative Response:

| Byte | Name | Cvt | Value |
|---|---|---|---|
| #1 | NegativeResponseServiceIdentifier | M | 7F |
| #2 | OriginalRequestServiceIdentifier | M | 19 |
| #3 | NegativeResponseCode | M | (see 19 01) |

### 6.5.1.5 ReadDTCInformation – ReportSupportedDTCs (0x19 0x0A)

Request:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | RequestServiceIdentifier | M | 19 |
| #2 | Sub-Function = [<br>ReportSupportedDTC ] | M | <br>0A |

Positive Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | PositiveResponseServiceIdentifier | M | 59 |
| #2 | ReportType = [<br>ReportSupportedDTCs ] | M | <br>0A |
| #3 | DTCStatusAvailabilityMask | M | 01-FF |
| | DTCAndStatusRecord [ ] = [ | | |
| #4 | DTCHighByte#1 | C1 | 00-FF |
| #5 | DTCMiddleByte#1 | C1 | 00-FF |
| #6 | DTCLowByte#1 | C1 | 00-FF |
| #7 | statusOfDTC#1 | C1 | 00-FF |
| #8 | DTCHighByte#2 | C2 | 00-FF |
| #9 | DTCMiddleByte#2 | C2 | 00-FF |
| #10 | DTCLowByte#2 | C2 | 00-FF |
| #11 | statusOfDTC#2 | C2 | 00-FF |
| : | : | : | : |
| #n-3 | DTCHighByte#m | C2 | 00-FF |

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #n-2 | DTCMiddleByte#m | C2 | 00-FF |
| #n-1 | DTCLowByte#m | C2 | 00-FF |
| #n | statusOfDTC#m ] | C2 | 00-FF |
| C1: This parameter is only present if DTC information is available to be reported. | | | |
| C2: This parameter is only present if more than one DTC is to be reported. | | | |

Negative Response:

| Byte | Name | Cvt | Value |
|---|---|---|---|
| #1 | NegativeResponseServiceIdentifier | M | 7F |
| #2 | OriginalRequestServiceIdentifier | M | 19 |
| #3 | NegativeResponseCode | M | (see 19 01) |

#### 6.5.1.6 ReadDTCInformation – ReportNumberOfEmissionsRelatedOBDDTCByStatusMask (0x19 0x12)

Request:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | RequestServiceIdentifier | M | 19 |
| #2 | Sub-Function = [ ReportNumberOfEmissionsRelatedOBDDTCByStatusMask ] | M | 12 |
| #3 | DTCStatusMask[15] | M | 00-FF |

Positive Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | PositiveResponseServiceIdentifier | M | 59 |
| #2 | ReportType = [ ReportNumberOfEmissionsRelatedOBDDTCByStatusMask] | M | 12 |
| #3 | DTCStatusAvailabilityMask[11] | M | 01-FF |
| #4 | DTCFormatIdentifier = [ | | |
| | ISO15031-6DTCFormat | M | 00 |
| | ISO14229-1DTCFormat ] | M | 01 |
| #5 | DTCCount [ ] = [ | | |
| | DTCCountHighByte | M | 00-FF |
| #6 | DTCCountLowByte ] | M | 00-FF |

[15] See chapter 7.3.7.

Negative Response:

| Byte | Name | Cvt | Value |
|---|---|---|---|
| #1 | NegativeResponseServiceIdentifier | M | 7F |
| #2 | OriginalRequestServiceIdentifier | M | 19 |
| #3 | NegativeResponseCode | M | (see below) |

Negative Response Codes:

| NRC (hex) | Error text |
|---|---|
| 12 | **subFunctionNotSupported**<br>This code is returned if the requested sub-function is not supported. |
| 13 | **incorrectMessageLengthOrInvalidFormat**<br>The length of the message is wrong. |
| 31 | **requestOutOfRange**<br>This code is returned if:<br>1. The client specified a DTCMaskRecord / DTCSeverityMaskRecord that was not recognized by the server.<br>2. The client specified an invalid DTCSnapshotRecordNumber /DTCExtendedDataRecordNumber. |

#### 6.5.1.7 ReadDTCInformation – ReportEmissionsRelatedOBDDTCByStatusMask (0x19 0x13)

Request:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | RequestServiceIdentifier | M | 19 |
| #2 | Sub-Function = [<br>ReportEmissionsRelatedOBDDTCByStatusMask ] | M | <br>13 |
| #3 | DTCStatusMask[16] | M | 00-FF |

[16] For details see chapter 7.3.7.

Positive Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | PositiveResponseServiceIdentifier | M | 59 |
| #2 | ReportType = [<br>ReportEmissionsRelatedOBDDTCByStatusMask] | M | <br>13 |
| #3 | DTCStatusAvailabilityMask | M | 01-FF |
| | DTCAndStatusRecord [ ] = [ | | |
| #4 | DTCHighByte#1 | C1 | 00-FF |
| #5 | DTCMiddleByte#1 | C1 | 00-FF |
| #6 | DTCLowByte#1 | C1 | 00-FF |
| #7 | statusOfDTC#1 | C1 | 00-FF |
| #8 | DTCHighByte#2 | C2 | 00-FF |
| #9 | DTCMiddleByte#2 | C2 | 00-FF |
| #10 | DTCLowByte#2 | C2 | 00-FF |
| #11 | statusOfDTC#2 | C2 | 00-FF |
| : | : | : | : |
| #n-3 | DTCHighByte#m | C2 | 00-FF |
| #n-2 | DTCMiddleByte#m | C2 | 00-FF |
| #n-1 | DTCLowByte#m | C2 | 00-FF |
| #n | statusOfDTC#m ] | C2 | 00-FF |
| C1: This parameter is only present if DTC information is available to be reported.<br>C2: This parameter is only present if more than one DTC is to be reported. | | | |

Negative Response:

| Byte | Name | Cvt | Value |
|---|---|---|---|
| #1 | NegativeResponseServiceIdentifier | M | 7F |
| #2 | OriginalRequestServiceIdentifier | M | 19 |
| #3 | NegativeResponseCode | M | (see 19 01) |

### 6.5.2 Parameter Options

The sub-function parameters are used by this service to select one of the DTC report types as defined as follows:

| Option (hex) | Description |
|---|---|
| 01 | **reportNumberOfDTCByStatusMask**<br>This parameter specifies that the server shall transmit to the client the number of DTC's matching a client defined status mask. |
| 02 | **reportDTCByStatusMask**<br>This parameter specifies that the server shall transmit to the client a list of DTC's and |

| | |
|---|---|
| | corresponding statuses matching a client defined status mask. |
| 04 | **reportDTCSnapshotRecordByDTCNumber**<br>This parameter specifies that the server shall transmit to the client the DTCSnapshot record(s) associated with a client defined DTC number and DTCSnapshot record number (FF hex for all records). |
| 06 | **reportDTCExtendedDataRecordByDTCNumber**<br>This parameter specifies that the server shall transmit to the client the DTCExtended-Data record(s) associated with a client defined DTC number and DTCExtendedData record number (FF hex for all records, FE hex for all OBD records). |
| 0A | **reportSupportedDTC**<br>This parameter specifies that the server shall transmit to the client a list of all DTC's and corresponding statuses supported within the server. |
| 12 | **ReportNumberOfEmissionsRelatedOBDDTCByStatusMask**<br>This parameter specifies that the server shall transmit to the client the number of emissionsrelated OBD DTC's matching a client defined status mask. The number of OBD DTC's reported shall only be those which are required to be compatible with emissions-related legal requirements |
| 13 | **ReportEmissionsRelatedOBDDTCByStatusMask**<br>This parameter specifies that the server shall transmit to the client a list of emissions-related OBD DTC's and corresponding statuses matching a client defined status mask. The list of OBD DTC's reported shall only be those which are required to be compatible with emissions-related legal requirements. |

Explanation of some parameters

| Parameter | Description |
|---|---|
| **DTC Status Mask** | The DTC Status Mask contains 8 DTC status bits. Definitions can be found at 7.3.7.<br>This byte is used in the request message to allow an external test tool to request DTC information for the DTCs whose status matches the DTC Status Mask. A DTC's status matches the DTC Status Mask if any one of the DTC's actual status bits is set to '1' and the corresponding status bit in the DTC Status Mask is also set to '1'. (i.e., if the DTC Status Mask is bit-wise logically ANDed with the DTC's actual status and the result is non-zero, then a match has occurred). If the external test tool specifies a status mask that contains bits that the ECU does not support, then the ECU shall process the DTC information using only the bits that it does support. |
| **DTC Mask Record** | DTC Record either contains DTC High Byte, DTC Middle Byte and DTC Low Byte or a DTC according to SAEJ1939.<br>DTC High Byte, DTC Middle Byte and DTC Low Byte together represent a unique identification number for a specific diagnostic trouble code supported by an ECU.<br>A DTC within the ECU matches the DTC Mask Record if the two values are identical (exact match required). |
| **DTC Extended Data Record Number** | DTC Extended Data Record Number is a 1-byte value indicating the number of the specific DTC Extended Data record requested for an external test tool defined DTC Mask Record via the ReportDTCExtendedDataRecordByDTCNumber sub-function.<br>- A value of 01 – 8F requests the server to report the GAC specific stored DTCExtendedData records.<br>- A value of 90 – EF requests the server to report legislated OBD stored DTC ExtendedData records.<br>- A value of FE requests the server to report all legislated OBD stored DTC ExtendedData records in a single response message.<br>- A value of FF requests the server to report all stored DTCExtendedData records in a single response message. |
| **DTC Snapshot Record Number** | DTC Snapshot Record Number is a single byte value indicating the number of the specific DTC Snapshot Data Record either associated with one particular DTC or as global unique identification of a Snapshot Data Record within the individual ECU.<br>For ECUs that do not support multiple DTC Snapshot Data Records for a single DTC, the external test tool shall set this value to 0. For ECUs that do support multiple DTC Snapshot Data Records for a single DTC, the diagnostic shall set this to a value ranging from 1 to the maximum number supported by the ECU (which may range up to FE, depending on the ECU). A value of FF requests the ECU to report all stored DTC Snapshot data records at once. |

### 6.5.3 Implementation Rules

**General:**

This service should only be supported, if environment data is helpful for trouble-shooting.

**ReadDTCInformation – ReportNumberOfDTCByStatusMask (0x19 0x01):**

A client can retrieve a count of the number of DTCs matching a client defined status mask by sending a request for this service. 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. After the DTCStatusAvailabilityMask the positive response contains the DTCFormatIdentifier which reports information about the way of 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.

**ReadDTCInformation – ReportDTCByStatusMask (0x19 0x02):**

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. For example, this sub-function allows the client to request the server to report all DTCs that are "testFailed" OR "confirmed" OR "something else".

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. The server shall return not only the DTCStatusAvailabilityMask, but also 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 which contains unsupported bits by the server, then the server shall process the DTC information using only the bits that it does support. If no DTCs within the server match the masking criteria specified in the client's request, no DTC or status information shall be provided following the DTCStatusAvailabilityMask byte in the positive response message.

DTC status information shall be cleared when the client sends the service ClearDiagnosticInformation (0x14).

**ReadDTCInformation – ReportDTCSnapshotRecordByDTCNumber (0x19 0x04):**

With the sub-function reportDTCSnapshotRecordByDTCNumber it is possible for a client to retrieve captured DTCSnapshot record data for defined DTCMaskRecord in conjunction with a DTCSnapshot record number. The server shall search through its supported DTCs until it finds an exact match with the DTCMaskRecord specified by the client (containing the DTC number (high, middle, and low byte)). If the server finds an exact match 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.

If a failure has been identified for the client defined DTCMaskRecord and DTCSnapshotRecordNumber, the server shall return a single pre-defined DTCSnaphotRecord in response to the client's request. That means, that in this case the server returns the DTC number, the statusOfDtc and this DTCSnapshotRecord.

The DTCSnapshot record contains multiple data parameters. These parameters can be used to reconstruct the vehicle conditions (e.g. B+, RPM, time-stamp) at the time of the failure occurrence.

First of all the data reported in the DTCSnapshotRecord contains a dataIdentifier to identify the data that follows. This combination of dataIdentifier[17] and data can be repeated within the DTCSnapshotRecord. The usage of one or multiple dataIdentifiers in the DTCSnapshotRecord allows the storage of different types of vehicle conditions. Along with the DTCSnapshotRecord should be a parameter which indicates the number of record DataIdentifiers contained within each DTCSnapshotRecord.

Normally, the server shall report one DTCSnapshotRecord in a single response message. However, if the client has set the DTCSnapshotRecordNumber to FF hex, the server shall respond with all DTCSnapshot records stored for the client defined DTCMaskRecord in a single response message. Typically the existence of multiple SnapshotRecords for one DTC describes the multiple occurrence of this DTC, the contained data shows the vehicle conditions at time(s) for (each) occurrence.[18]

If the client requested to report all DTCSnapshot records by DTC number, the DTCAndStatusRecord is only included one time in the response message. With this service it is not possible to request a specified SnapshotRecordNumber from all DTCs – using a DTC Group number in the request is not possible.

If the DTCMaskRecord or DTCSnapshotRecordNumber parameters are invalid or not supported by the server, the server will send a negative response.

However, if the DTCMaskRecord and/or DTCSnapshotRecordNumber parameters in the request are valid and supported by the server, but have no DTCSnapshot data associated with it, 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) followed by the DTCSnapshotRecordNumber (echo of the requested record number).

It is recommended to to use only one DTCSnapshotRecord.

DTCSnapshot information shall be cleared when the client sends the service ClearDiagnosticInformation (0x14).

**ReadDTCInformation – ReportDTCExtendedDataRecordByDTCNumber (0x19 0x06)**

The sub-function ReportDTCExtendedDataRecordByDTCNumber causes the server to respond DTCExtendedData for a client defined DTCMaskRecord together with a DTCExtendedData record number. In this case the server searches through all supported DTCs for an exact match with the DTCMaskRecord from the request. The response shall contain a single pre-defined DTCExtendedData record.

---

[17] See also Service 22 ReadDataByIdentifier for the usage of dataIdentifiers in the ECU.

[18] For restrictions due to the (limited) number of SnapshotRecords per DTC and the handling for updating the content, see chapter 7.3.

The structure of data reported in the DTCExtendedDataRecord is defined by the DTCExtendedDataRecordNumber in a similar way to the definition of data within a record DataIdentifier. The response may include multiple DTCExtendedDataRecordNumbers and associated DTCExtendedDatarecords. A single DTC can store different type of DTCExtendedDataRecords.

Normally the server shall report every DTCExtendedData record in a single response message. However, if the client has set the DTCExtendedDataRecordNumber to FF, the server shall respond with all DTCExtendedData records stored for the client defined DTCMaskRecord in a single response message. If the DTCMaskRecord or DTCExtendedDataRecordNumber parameters are invalid or not supported, the server responds negatively.

It is recommended to to use only one DTCExtendedDataRecord.

The ExtendedData information can be cleared with service 0x14.

**ReadDTCInformation – ReportSupportedDTCs (0x19 0x0A)**

The sub-function reportSupportedDTCs causes the server to respond all DTCs which are supported by the server. This response contains DTCStatusAvailabilityMask, which provides an indication of DTC status bits that are supported by the server for masking purposes. Furthermore the response also contains the listOfDTCAndStatusRecord, which contains the DTC number and associated status for every diagnostic trouble code supported by the server.

**ReadDTCInformation – ReportNumberOfEmissionsRelatedOBDDTCByStatusMask (0x19 0x12)**

A client can retrieve a count of the number of emission related OBD DTCs matching a client defined status mask by sending a request for this service. 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. After the DTCStatusAvailabilityMask the positive response contains the DTCFormatIdentifier which reports information about the way of DTC formatting and encoding. The DTCFormatIdentifier is followed by the DTCCount parameter which is a 2-byte unsigned numeric number containing the number of emission related OBD DTCs available in the server's memory based on the status mask provided by the client.

**ReadDTCInformation – ReportEmissionsRelatedOBDDTCByStatusMask (0x19 0x13)**

The client can retrieve a list of emission related OBD DTCs, which satisfy a client defined status mask by sending a request with the sub-function byte set to reportEmissionRelatedOBDDTCByStatusMask. For example, this sub-function allows the client to request the server to report all emission related OBD DTCs that are "testFailed" OR "confirmed" OR "something else".

The evaluation of the status bits shall be done as as described in service 0x19 0x02.

## 6.6 ReadDataByIdentifier (22)

The request message requests data record values from the server identified by one or more DataIdentifier(s).

The server sends data record values via the positive response message. The format and definition of the RecordValues is defined by GAC. RecordValues shall include analogue input and output signals, digital input and output signals, internal data and system status information if supported by the ECU.

### 6.6.1 Message Format

Request:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | RequestServiceIdentifier | M | 22 |
| | DataIdentifier[ ] #1 = [ | | |
| #2 | byte#1 (MSB) | M | 00-FF |
| #3 | byte#2 ] | M | 00-FF |
| : | : | : | : |
| | DataIdentifier[ ] #m = [ | | |
| #n-1 | byte#1 (MSB) | U | 00-FF |
| #n | byte#2 ] | U | 00-FF |

Positive Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | PositiveResponseServiceIdentifier | M | 62 |
| | DataIdentifier [ ] #1 = [ | | |
| #2 | byte#1 (MSB) | M | 00-FF |
| #3 | byte#2] | M | 00-FF |
| | DataRecord[ ] #1 = [ | | |
| #4 | data#1 | M | 00-FF |
| : | : | : | : |
| #(k-1)+4 | data#k ] | U | 00-FF |
| : | : | : | : |
| #n-(o-1)-2 | DataIdentifier [ ] #m = [ | | |
| | byte#1 (MSB) | U | 00-FF |
| #n-(o-1)-1 | byte#2 ] | U | 00-FF |
| | DataRecord[ ] #m = [ | | |
| #n-(o-1) | data#1 | U | 00-FF |
| : | : | : | : |
| #n | data#o ] | U | 00-FF |

Negative Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | NegativeResponseServiceIdentifier | M | 7F |
| #2 | OriginalRequestServiceIdentifier | M | 22 |
| #3 | NegativeResponseCode | M | (see below) |

Negative Response Codes:

| NRC (hex) | Error text |
|---|---|
| 13 | **incorrectMessageLengthOrInvalidFormat**<br>This response code shall be sent if the length of the request message is invalid. |
| 22 | **conditionsNotCorrect**<br>This response code shall be sent if the operating conditions of the server are not met to perform the required action. |
| 31 | **requestOutOfRange**<br>This code shall be sent if<br>1. none of the requested dataIdentifier values are supported by the device.<br>2. the client exceeded the maximum number of dataIdentifiers allowed to be requested at a time. |
| 33 | **securityAccessDenied**<br>This code shall be sent if at least one of the dataIdentifiers is secured and the server is not in an unlocked state. |

### 6.6.2 Parameter Options

The DataIdentifier are defined in Annex B.

### 6.6.3 Implementation Rules

The maximum number of data identifiers to be read within a single request is limited to 5.

## 6.7 ReadMemoryByAddress (23)

This service requests data from the ECU. The location of the data and the size of the data block are identified by Memory Address and Memory Size parameters.

### 6.7.1 Message Format

Request:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | RequestServiceIdentifier | M | 23 |
| #2 | AddressAndLengthFormatIdentifier | M | 0x24 |
| | MemoryAddress [ ] = [ | | |
| #3 | byte#1 (MSB) | M | 00-FF |
| : | : | : | : |
| #6 | byte#4 ] | M | 00-FF |
| | MemorySize [ ] = [ | | |
| #7 | byte#1 (MSB) | M | 00-FF |
| #8 | byte#2 ] | M | 00-FF |

Positive Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | PositiveResponseServiceIdentifier | M | 63 |
| | DataRecord [ ] = [ | | |
| #2 | data#1 | M | 00-FF |
| : | : | : | : |
| #n | data#m ] | U | 00-FF |

Negative Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | NegativeResponseServiceIdentifier | M | 7F |
| #2 | OriginalRequestServiceIdentifier | M | 23 |
| #3 | NegativeResponseCode | M | (see below) |

Negative Response Codes:

| NRC (hex) | Error text |
|---|---|
| 13 | **incorrectMessageLengthOrInvalidFormat**<br>The length of the message is wrong. |
| 22 | **conditionsNotCorrect**<br>This response code shall be sent if the operating conditions of the server are not met to perform the required action. |
| 31 | **requestOutOfRange**<br>This response code shall be sent if:<br>1. any memory address within the interval [$MA, ($MA + $MS -$1)] is invalid,<br>2. any memory address within the interval [$MA, ($MA + $MS -$1)] is restricted,<br>3. the memorySize parameter value in the request message is greater than the maximum value supported by the server,<br>4. the specified addressAndLengthFormatIdentifier is not valid. |
| 33 | **SecurityAccessDenied**<br>This code shall be sent if any memory address within the interval [$MA, ($MA + $MS -$1)] is secure and the server is locked. |

### 6.7.2 Parameter Options

The MemoryAddress parameter length shall be 4Byte.

The address may represent a physical address location within the ECU, but addresses may also be mapped (e.g. to implement a memory selector in upper bits). However, if the address parameter does not correspond to the physical location, the mapping shall be documented within the ECU diagnostic specification.

The MemorySize parameter length shall be 2Byte.

### 6.7.3 Implementation Rules

An ECU shall perform appropriate security mechanisms for direct memory access.

## 6.8 SecurityAccess (27)

### 6.8.1 Message Format

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.

The client shall request the server to unlock by sending the service SecurityAccess-RequestSeed message. The server shall respond by sending a seed. The seed is the input parameter for the key calculation algorithm. It is used by the client to calculate the corresponding key value.

In a second step, the client shall request the key comparison by sending the calculated key to the server using the appropriate service SecurityAccess-SendKey. 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 If the two numbers do not match, this shall be considered as a false access attempt. If access is rejected for any other reason, it shall not be considered as a false access attempt. An invalid key requires the client to start over from the beginning with a SecurityAccess-RequestSeed message positive response message.

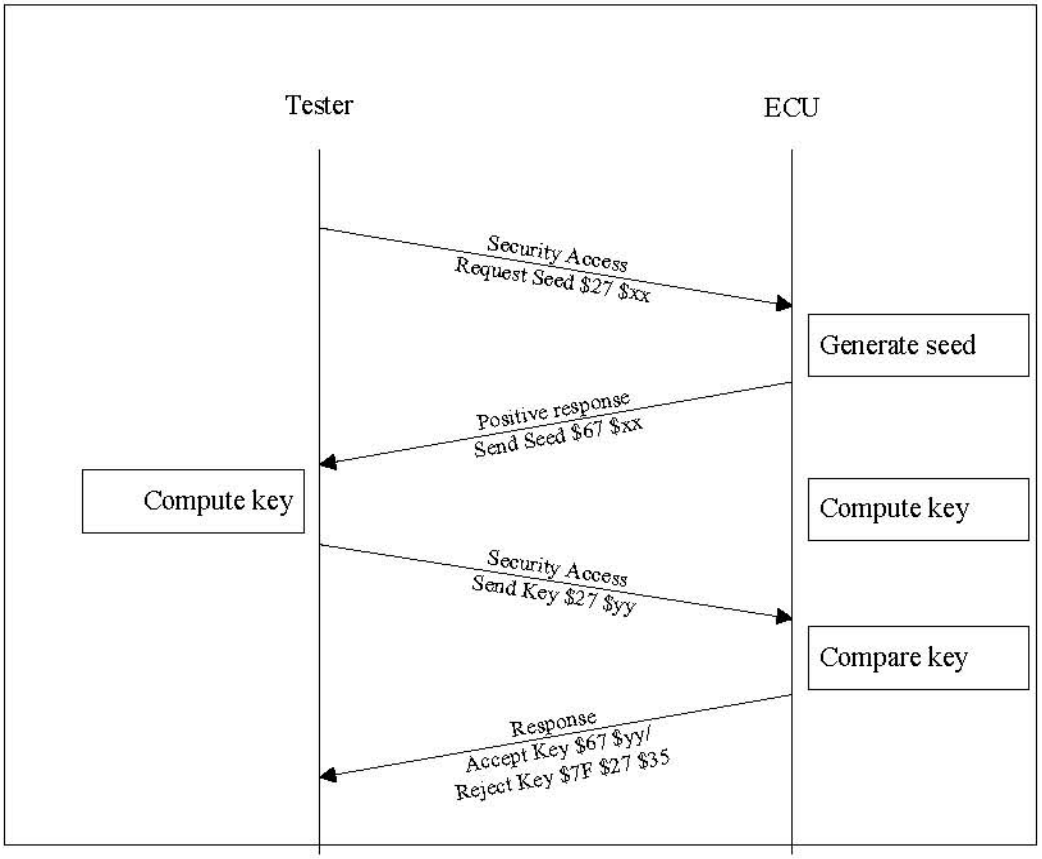



*Figure -8: SecurityAccess procedure*

If a server supports security, but 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). If then, the service SecurityAccess-SendKey is received, the server shall respond with NRC 0x24. The client shall use this method to determine if a server is locked by checking for a non-zero seed.

Level 2 has higher Priority than Level 1. A tester request is successful in unlocking the security level associated with requestSeed 03, then the secured functionality supported by the security level associated with requestSeed 03 and 01 shall be unlocked at that time.

#### 6.8.1.1 Request Seed

This service requests a seed from the server. Based on this seed, the client is able to calculate the corresponding key to be sent for unlocking the server.

The length of seed shall be 4 bytes.

Request:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | RequestServiceIdentifier | M | 27 |
| #2 | Sub-fucntion=[<br>SecurityAccessType = RequestSeed<br>] | M | 01, 03, 11 |

Positive Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | PositiveResponseServiceIdentifier | M | 67 |
| #2 | SecurityAccessType = RequestSeed | M | 01, 03, 11 |
| | SecuritySeed [ ] = [ | | |
| #3 | seed#1 (high byte) | M | 00-FF |
| : | : | : | : |
| #n | seed#m (low byte) ] | M | 00-FF |

Negative Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | NegativeResponseServiceIdentifier | M | 7F |
| #2 | OriginalRequestServiceIdentifier | M | 27 |
| #3 | NegativeResponseCode | M | (see below) |

Negative Response Codes:

| NRC (hex) | Error text |
|---|---|
| 12 | **subFunctionNotSupported**<br>Send if the sub-function parameter in the request message is not supported. |
| 13 | **incorrectMessageLengthOrInvalidFormat**<br>The length of the message is wrong. |
| 22 | **conditionsNotCorrect**<br>This code shall be returned if the criteria for the request SecurityAccess are not met. |
| ~~35~~ | ~~**invalidKey**~~<br>~~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.~~ |
| 37 | **requiredTimeDelayNotExpired**<br>Send if the delay timer is active and a request is transmitted. |

### 6.8.1.2 Send Key

This service sends a key calculated by the client to the server. 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.

The length of key shall be 4 bytes.

Request:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | RequestServiceIdentifier | M | 27 |
| #2 | Sub-fucntion=[<br>SecurityAccessType = SendKey<br>] | M | 02, 04, 12 |
| #3<br>:<br>#n | SecurityKey [ ] = [<br>key#1 (high byte)<br>:<br>key#m (low byte) ] | M<br>:<br>U | 00-FF<br>:<br>00-FF |

Positive Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | PositiveResponseServiceIdentifier | M | 67 |
| #2 | SecurityAccessType = SendKey | M | 02, 04, 12 |

Negative Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | NegativeResponseServiceIdentifier | M | 7F |
| #2 | OriginalRequestServiceIdentifier | M | 27 |

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #3 | NegativeResponseCode | M | (see below) |

Negative Response Codes:

| NRC (hex) | Error text |
|---|---|
| 12 | **subFunctionNotSupported**<br>Send if the sub-function parameter in the request message is not supported. |
| 13 | **incorrectMessageLengthOrInvalidFormat**<br>The length of the message is wrong. |
| 22 | **conditionsNotCorrect**<br>This code shall be returned if the criteria for the request SecurityAccess are not met.. |
| 24 | **requestSequenceError**<br>Send if the 'sendKey' sub-function is received without first receiving a 'requestSeed' request message. |
| 35 | **invalidKey**<br>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. |
| 36 | **exceededNumberOfAttempts**<br>Send if the delay timer is active due to exceeding the maximum number of allowed false access attempts. |

### 6.8.2 Parameter Options

The SecurityAccessType is defined as follows:

| Option (hex) | Description |
|---|---|
| 00 | (Reserved by ISO) |
| 01 | **requestSeed to reach security level: Unlocked (Level 1)**<br>RequestSeed with the level of security defined by the GAC. |
| 02 | **sendKey to reach security level: Unlocked (Level 1)**<br>SendKey with the level of security defined by GAC. |
| 03 | **requestSeed to reach security level: Unlocked (Level 2)**<br>RequestSeed with the level of security defined by the GAC. |
| 04 | **sendKey to reach security level: Unlocked (Level 2)**<br>SendKey with the level of security defined by GAC. |
| 11 | **requestSeed to reach security level: Unlocked (Flash)**<br>RequestSeed with the level of security defined by GAC. |
| 12 | **sendKey to reach security level: Unlocked (Flash)**<br>SendKey with the level of security defined by GAC. |

### 6.8.3 Implementation Rules

After PowerOn/Reset the ECU is in locked state. The security access failure counter is set to zero.

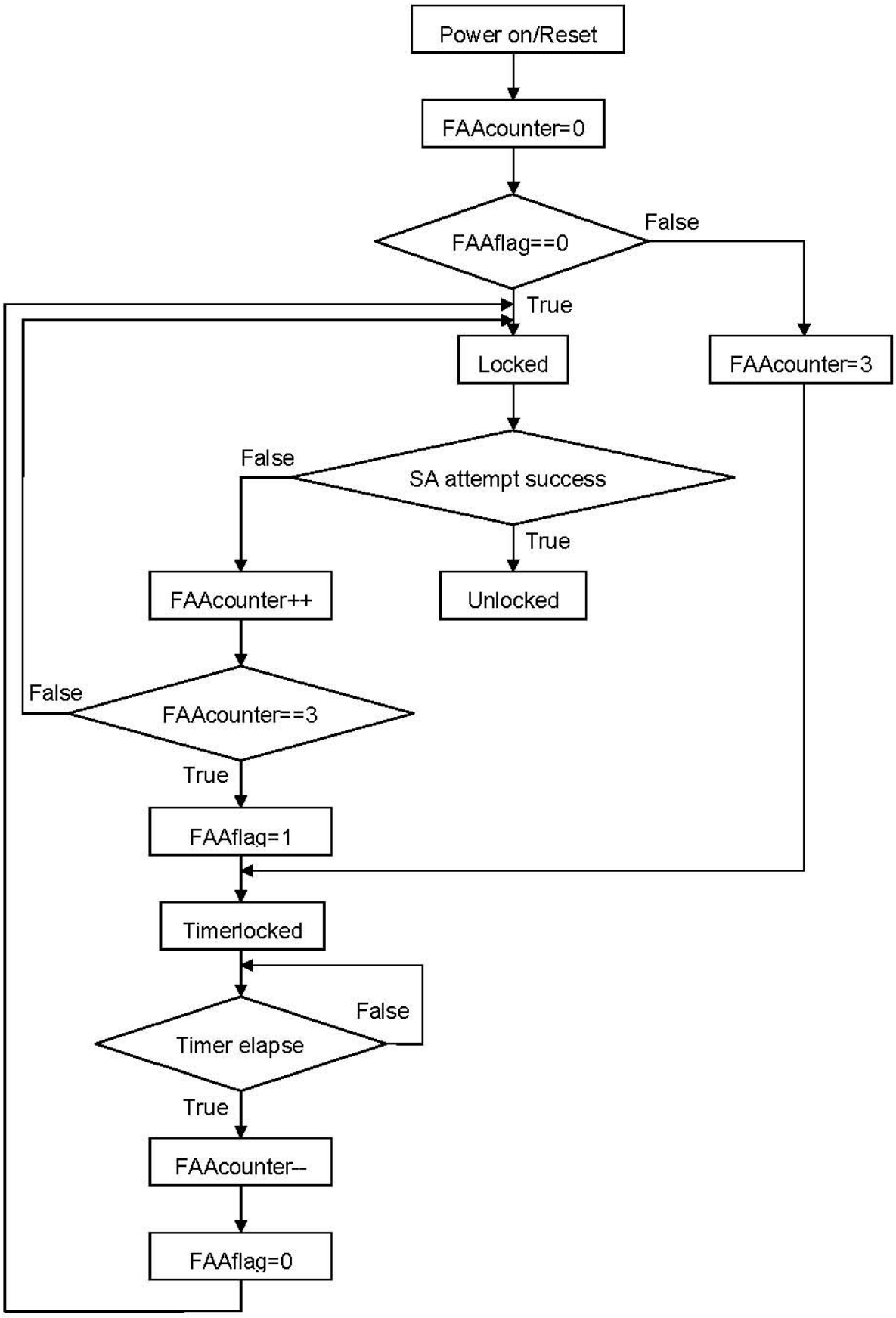
Depending on the ECU, the ECU shall wait 10 seconds before accepting the first Request-Seed message after EcuReset/PowerOn. The implementation of this behaviour shall be agreed by GAC.

For all other ECUs not implementing this behaviour, a flag called FAA-Flag is stored in the EEPROM of the ECU. On every PowerOn/Reset, the ECU checks for this flag and, if available it waits 10s before accepting the next “Request Seed” message.

After the third failure attempt of a “SecurityAccess sendKey request” (security access failure counter reaches the value of three) the ECU shall wait 10s before accepting the next “Request Seed” message. Any SecurityAccess request during this time will be rejected with the negative response code “Required time delay not expired” (37).

When the 10s wait time is elapsed the security access failure counter is decremented by one, any exisiting FAA flag is removed and another try is allowed. When during this try the security access failure counter is incremented again (due to an invalid key), the ECU shall wait again 10s before accepting another “Request Seed” message. When this try is valid, the security access failure counter is not changed, but the FAA Flag is removed.

The following figure shows this behaviour:


Power on/Reset
FAAcounter=0
FAAflag==0
False
True
Locked
FAAcounter=3
False
SA attempt success
True
FAAcounter++
Unlocked
False
FAAcounter==3
True
FAAflag=1
Timerlocked
False
Timer elapse
True
FAAcounter--
FAAflag=0

If the tester requests a seed, it has to send the corresponding key to the ECU. This sequence is mandatory. If the tester sends a consecutive “Request Seed”, the request is accepted and the same seed is returned, but the security access failure counter is incremented.

If the tester sends an invalid key, the request is rejected with negative response code “InvalidKey”, the sequence shall be reset (any current seed becomes invalid) and the security access failure counter is incremented.

GAC uses four states (three different levels): Locked, Unlocked (Level 1), Unlock (Level 2) and Unlocked (Flash).

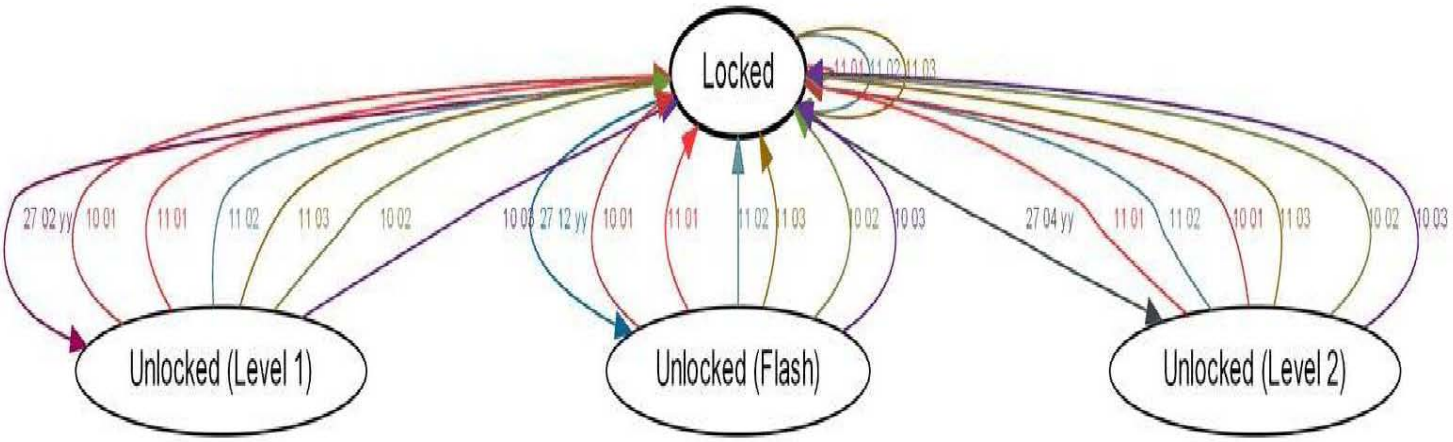



Figure -9: SecurityAccess

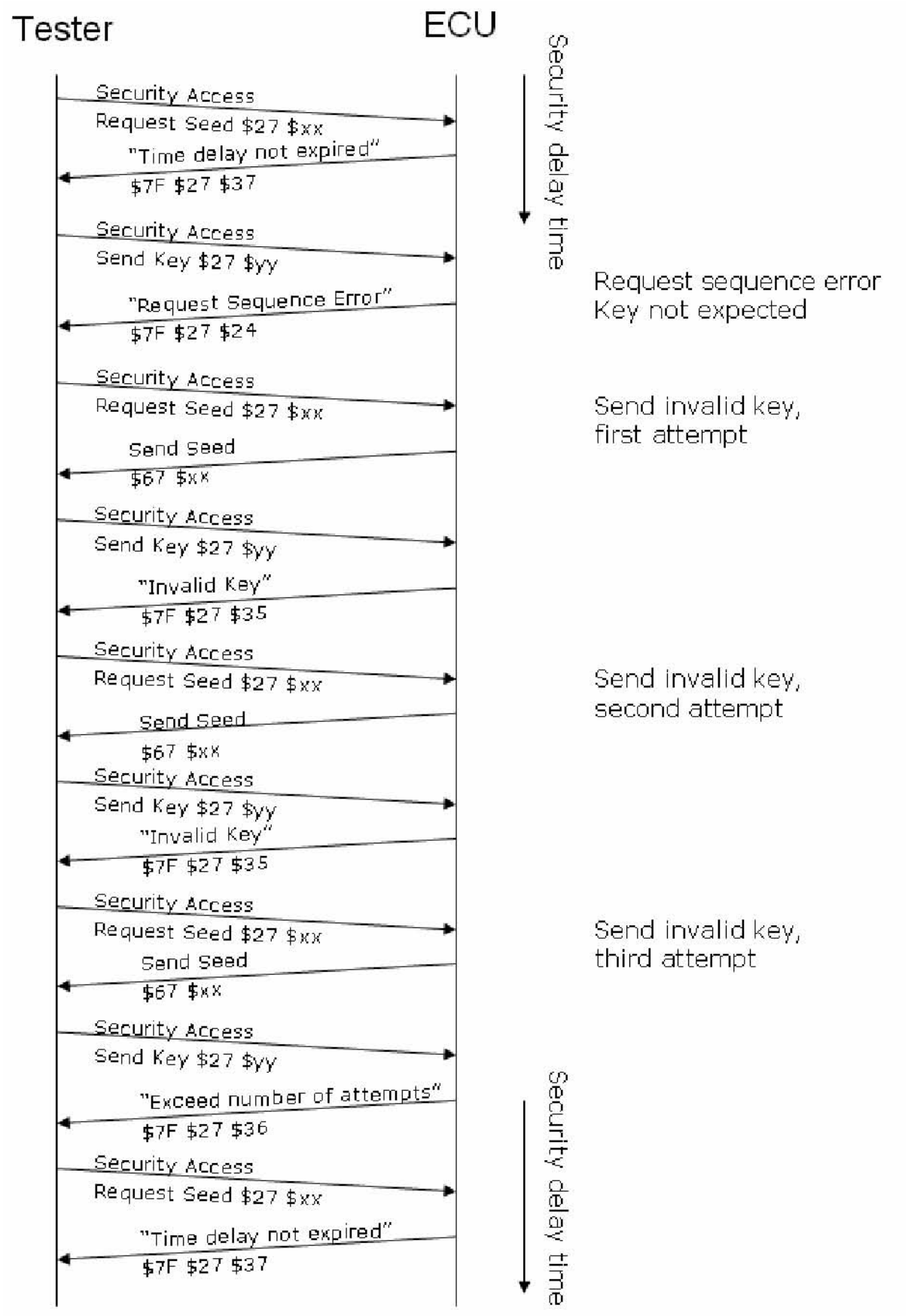



Figure -10: SecurityAccess NRC requirements

Secured data identifiers require an increased security level. Therefore, they are accessible only within a non-default diagnostic session.

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

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.

## 6.9 CommunicationControl (28)

The service is used to "switch on/off" the transmission and/or the reception of certain messages of (a) server(s).

### 6.9.1 Message Format

Request:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | RequestServiceIdentifier | M | 28 |
| #2 | Sub-Function = [ ControlType ] | M | 00-FF |
| #3 | CommunicationType | M | 00-FF |

Positive Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | PositiveResponseServiceIdentifier | M | 68 |
| #2 | Sub-Function = [ ControlType ] | M | 00-7F |

Negative Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | NegativeResponseServiceIdentifier | M | 7F |
| #2 | OriginalRequestServiceIdentifier | M | 28 |
| #3 | NegativeResponseCode | M | (see below) |

Negative Response Codes:

| NRC (hex) | Error text |
|---|---|
| 12 | **subFunctionNotSupported**<br>Send if the sub-function parameter in the request message is not supported |
| 13 | **incorrectMessageLengthOrInvalidFormat**<br>The length of the message is wrong |
| 22 | **conditionsNotCorrect**<br>Used when the server is in a critical normal mode activity and therefore cannot disable/enable the requested communication type. |
| 31 | **requestOutOfRange**<br>The server shall use this response code, if it detects an error in the communicationType parameter. |

Please Note:

According to [ISO3], the DisableMode parameter of the request message is not repeated within the response message.

### 6.9.2 Parameter Options

The following table specifies predefined values for the ControlType parameter.

| Option (hex) | Description |
|---|---|
| 00 | **enableRxAndTx**<br>This value indicates that the reception and transmission of messages shall be enabled for the specified communicationType. |
| 01 | **enableRxAndDisableTx**<br>This value indicates that the reception of messages shall be enabled and the transmission shall be disabled for the specified communicationType. |
| 02 | **disableRxAndEnableTx**<br>This value indicates that the reception of messages shall be disabled and the transmission shall be enabled for the specified communicationType. |
| 03 | **disableRxAndTx**<br>This value indicates that the reception and transmission of messages shall be disabled for the specified communicationType. |

The CommunicationType 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.

| Option (hex) | Description |
|---|---|
| 00 | (Reserved by ISO) |
| 01 | **normalCommunicationMessages**<br>This value references all application-related communication (inter-application signal exchange between multiple in-vehicle servers). |
| 02 | **networkManagementCommunicationMessages**<br>This value references all network management related communication. |
| 03 | **networkManagementCommunicationMessages and**<br>**normalCommunicationMessages**<br>This value references all network management and application-related communication. |

Note: If the ECU is communicating on multiple channels, this service will affect the corresponding message on all channels (not only those where the diagnostic request has been received)!

### 6.9.3 Implementation Rules

There are no special implementation rules for this service.

## 6.10 ReadDataByPeriodicIdentifier (2A)

This service allows the client to request the periodic transmission of data record values from the server identified by one or more periodicDataIdentifiers

The format and definition of the dataRecord is defined by GAC, and may include analog input and output signals, digital input and output signals, internal data, and system status information if supported by the server.

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.

There are two types of periodic data response messages defined to transmit the periodicDataIdentifier data to the client following the initial positive response message in [ISO1]. This specification handles only one kind of positive response.

### 6.10.1 Message Format

Request:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | RequestServiceIdentifier | M | 2A |
| #2 | Transmission Mode | M | 01, 02, 03, 04 |
| #3 | PeriodicDataIdentifier [ ] #1 | C | 00-FF |
| : | : | : | : |
| #m+2 | PeriodicDataIdentifier [ ] #m | U | 00-FF |
| 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. | | | |

Positive Response[19]:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | PositiveResponseServiceIdentifier | M | 6A |
| #2 | PeriodicDataIdentifier | M | 00-FF |
| | DataRecord [ ] = [ | | |
| #3 | data#1 | M | 00-FF |
| : | : | : | : |
| #k+1 | data#k ] | U | 00-FF |

[19] Note: This is a so called Positive Response Type #1. After receiving the service request, the ECU returns $6A as an initial positive response.

Negative Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | NegativeResponseServiceIdentifier | M | 7F |
| #2 | OriginalRequestServiceIdentifier | M | 2A |
| #3 | NegativeResponseCode | M | (see below) |

Negative Response Codes:

| NRC (hex) | Error text |
|---|---|
| 13 | **incorrectMessageLengthOrInvalidFormat**<br>This response code shall be sent if the length of the request message is invalid. |
| 22 | **conditionsNotCorrect**<br>This response code 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. |
| 31 | **requestOutOfRange**<br>This code shall be sent if<br>1. none of the requested periodicDataIdentifier values are supported by the device.<br>2. the client exceeded the maximum number of periodicDataIdentifiers allowed to be requested at a time.<br>3. The specified transmissionMode is not supported by the device. |
| 33 | **securityAccessDenied**<br>This code shall be sent if the periodicDataIdentifier is secured and the server is not in an unlocked state. |

### 6.10.2Parameter Options

The transmissionMode parameter is defined as follows:

| Option (hex) | Description |
|---|---|
| 00 | (Reserved by ISO) |
| 01 | **sendAtSlowRate**<br>This parameter specifies that the server shall transmit the requested dataRecord information at a slow rate in response to the request message. The repetition rate specified by the transmissionMode parameter slow is established by GAC, and pre-defined in the server. |
| 02 | **sendAtMediumRate**<br>This parameter specifies that the server shall transmit the requested dataRecord information at a medium rate in response to the request message. The repetition rate specified by the transmissionMode parameter medium is established by GAC, and pre-defined in the server. |
| 03 | **sendAtFastRate**<br>This parameter specifies that the server shall transmit the requested dataRecord information at a fast rate in response to the request message. The repetition rate specified by the transmissionMode parameter fast is established by GAC, and pre-defined in the server. |
| 04 | **stopSending**<br>The server stops transmitting positive response messages send periodically/repeatedly.<br>(stopSending shall be supported only if at least one of the other transmission modes (sendAtSlowRate, sendAtMediumRate, and/or sendAtFastRate) are supported.) |
| 05 – FF | (Reserved by ISO) |

The periodicDataIdentifier represents the low byte of a dataIdentifier out of the range 0xF2 0x00 – 0xF2 0xFF.

### 6.10.3Implementation Rules

The server may limit the number of periodicDataIdentifiers that can be simultaneously supported as agreed upon GAC 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.

This type of response message is mapped onto a USDT message, using the same response CAN identifier as used for any other USDT response message. The USDT message for a single periodicDataIdentifier shall not exceed the size of a single CAN frame.

## 6.11 DynamicallyDefineDataIdentifier (2C)

This service allows the client to dynamically define a data identifier in a server that can be read with the ReadDataByIdentifier service at a later time.

The intention of this service is to provide the client with the ability to group one or more data elements into a data superset that can be requested en masse via the ReadDataByIdentifier or ReadDataByPeriodicIdentifier service.

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 data elements to be grouped together can be referenced by a source data identifier.

The dynamically defined dataIdentifier will then contain a concatenation of the data parameter definitions.

Note: A dynamically defined identifier shall not refer to another dynamically defiend identifier to avoid server complexity and data inconsistency when the referenced identifier is cleared before its referent.

### 6.11.1 Message Format

#### 6.11.1.1 DynamicallyDefineDataIdentifier – DefineByIdentifier

Request:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | RequestServiceIdentifier | M | 2C |
| #2 | Sub-Function = [ DefineByIdentifier ] | M | 01 |
| | dynamicallyDefinedDataIdentifier[] = [ | | |
| #3 | byte#1 (MSB) | M | F2, F3 |
| #4 | byte#2 (LSB) ] | M | 00-FF |
| | sourceDataIdentifier[] #1 = [ | | |
| #5 | byte#1 (MSB) | M | 00-FF |
| #6 | byte#2 (LSB) ] | M | 00-FF |
| #7 | positionInSourceDataRecord #1 | M | 01-FF |
| #8 | memorySize #1 | M | 00-FF |
| : | : | : | : |
| | sourceDataIdentifier[] #m = [ | | |
| #n – 3 | byte#1 (MSB) | U | 00-FF |
| #n – 2 | byte#2 (LSB) ] | U | 00-FF |
| #n – 1 | positionInSourceDataRecord #m | U | 01-FF |
| N | memorySize #m | U | 00-FF |

Positive Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | PositiveResponseServiceIdentifier | M | 6C |
| #2 | DefinitionType = [ DefineByIdentifier ] | M | 01 |
| | dynamicallyDefinedDataIdentifier[20] [] = [ | | |
| #3 | byte#1 (MSB) | C | F2, F3 |
| #4 | byte#2 (LSB) ] | C | 00-FF |
| 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. | | | |

Negative Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | NegativeResponseServiceIdentifier | M | 7F |
| #2 | OriginalRequestServiceIdentifier | M | 2C |
| #3 | NegativeResponseCode | M | (see below) |

### 6.11.1.2 DynamicallyDefineDataIdentifier – ClearDynamicallyDefinedDataIdentifier

Request:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | RequestServiceIdentifier | M | 2C |
| #2 | Sub-Function = [ ClearDynamicallyDefinedDataIdentifier ] | M | 03 |
| | dynamicallyDefinedDataIdentifier[] = [ | | |
| #3 | byte#1 (MSB) | C | F2, F3 |
| #4 | byte#2 (LSB) ] | C | 00-FF |
| 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. | | | |

Positive Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | PositiveResponseServiceIdentifier | M | 6C |
| #2 | DefinitionType = [ ClearDynamicallyDefinedDataIdentifier ] | M | 03 |
| | dynamicallyDefinedDataIdentifier [] = [ | | |
| #3 | byte#1 (MSB) | C | F2, F3 |
| #4 | byte#2 (LSB) ] | C | 00-FF |
| 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. | | | |

[20] Testers can only read identifiers defined with MSB 0xF2 periodically.

Negative Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | NegativeResponseServiceIdentifier | M | 7F |
| #2 | OriginalRequestServiceIdentifier | M | 2C |
| #3 | NegativeResponseCode | M | (see below) |

Negative Response Codes:

| NRC (hex) | Error text |
|---|---|
| 12 | **subFunctionNotSupported**<br>This response code shall be sent if the sub-function parameter is not supported. |
| 13 | **incorrectMessageLengthOrInvalidFormat**<br>The length of the message is wrong |
| 22 | **conditionsNotCorrect**<br>This response code shall be sent if the operating conditions of the server are not met to perform the required action. |
| 31 | **requestOutOfRange**<br>This response code shall be sent if:<br>1. Any data identifier (dynamicallyDefinedDataIdentifier or any sourceDataIdentifier) in the request message is not supported/invalid.<br>2. The positionInSourceDataRecord was incorrect (less than 1, or greater than maximum allowed by server)<br>3. Any memory address in the request message is not supported in the server.<br>4. The specified memorySize was invalid.<br>5. The amount of data to be packed into the dynamic data identifier exceeds the maximum allowed by server.<br>6. The specified addressAndLengthFormatIdentifier is not valid. |
| 33 | **securityAccessDenied**<br>This code shall be sent if:<br>1. any data identifier (dynamicallyDefinedDataIdentifier or any sourceDataIdentifier) in the request message is secured and the server is not in an unlocked state.<br>2. any memory address in the request message is secured and the server is not in an unlocked state. |

### 6.11.2Parameter Options

The subfunction parameter is defined as follows:

| Option (hex) | Description |
|---|---|
| 01 | **defineByIdentifier**<br>This value shall be used to specify to the server that definition of the dynamic data identifier shall occur via a data identifier reference. |
| 03 | **clearDynamicallyDefinedDataIdentifier**<br>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 (0xF200-0xF3FF). 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. |

### 6.11.3 Implementation Rules

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.!

## 6.12 InputOutputControlByIdentifier (2F)

This service is used by the client to substitute a value for an input signal, internal ECU function and/or control an output (actuator) of an electronic system referenced by a DataIdentifier of the server. The user optional ControlOption parameter shall include all information required by the server's input signal, internal function and/or output signal. The server shall send a positive response message if the request message was successfully executed.

### 6.12.1 Message Format

#### 6.12.1.1 InputOutputControlByIdentifier

Request:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | RequestServiceIdentifier | M | 2F |
| #2 | DataIdentifier#1[ ] = [ byte#1 (MSB) | M | 00-FF |
| #3 | byte#2 (LSB) ] | M | 00-FF |
| #4<br>:<br>#4+(m-1) | ControlOptionRecord#1[ ] = [<br>controlState#1/inputOutputControlParameter<br>:<br>constrolState#m] | M1/U1:<br>C1 | 00-FF<br>:<br>00-FF |
| #4+m<br>:<br>#4+m+(r-1) | ControlEnableMaskRecord#1[ ] = [<br>controlMask#1<br>:<br>controlMask#r] | C2<br>:<br>C2 | 00-FF<br>:<br>00-FF |

M1/U1 Mandatory: ControlState#1 can be used as either an InputOutputControlParameter or as an additional controlState. If it is used as an InputOutputControlParameter then it shall be implemented as defined in Annex B.

C1: The presence of this parameter depends on the dataIdentifier#1 and the inputOutputControlParameter of controlOptionRecord#1 (if controlState#1 of controlOptionRecord#1 is used as an inputOutputControlParameter).

C2: The presence of this parameter depends on the dataIdentifier#1.

Positive Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | PositiveResponseServiceIdentifier | M | 6F |
| | DataIdentifier#1 = [ | | |
| #2 | byte#1 (MSB) | M | 00-FF |
| #3 | byte#2 (LSB) ] | M | 00-FF |
| | ControlStatusRecord#1[ ] = [ | | |
| #4 | controlState#1/inputOutputControlParameter | C1 | 00-FF |
| : | : | : | : |
| #4+(m-1) | controlState#m ] | C2 | 00-FF |

C1: The presence of this parameter depends on its usage in the request message. ControlState#1 is either used as an InputOutputControlParameter or as an additional controlState. If it is used as an InputOutputControlParameter then it shall be present in the response message and shall be the echo of the InputOutputControlParameter value given in the request message.

In all other cases it is user optional to be present (depends on the usage of a controlStatusRecord).

C2: The presence of this parameter depends on the dataIdentifier and the inputOutputControlParameter (if controlState#1 is used as an inputOutputControlParameter).

Negative Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | NegativeResponseServiceIdentifier | M | 7F |
| #2 | OriginalRequestServiceIdentifier | M | 2F |
| #3 | NegativeResponseCode | M | (see below) |

Negative Response Codes:

| NRC (hex) | Error text |
|---|---|
| 13 | **incorrectMessageLengthOrInvalidFormat**<br>The length of the message is wrong |
| 22 | **conditionsNotCorrect**<br>This code shall be returned if the criteria for the request InputOutputControl are not met. |
| 31 | **requestOutOfRange**<br>This code shall be returned if:<br>1. the requested dataIdentifier value is not supported by the device,<br>2. the dataIdentifier uses the controlState#1 parameter as an inputOutputControlParameter and the value contained in this parameter is invalid (see definition of inputOutputControlParameter),<br>3. the one or multiple of the applicable controlStates of the controlOptionRecord record are invalid. |
| 33 | **securityAccessDenied**<br>This code shall be returned if a client sends a request with a valid secure dataIdentifier and the server's security feature is currently active. |

### 6.12.2Parameter Options

Possible InputOutputControlParameters are:

| Option (Hex) | Description |
|---|---|
| 00 | **returnControlToECU**<br>This value shall indicate to the server that the client does no longer have control about the input signal, internal parameter or output signal referenced by the inputOutputLocalIdentifier.<br>Number of controlState bytes in request: 0<br>Number of controlState bytes in pos. response: 0 |
| 01 | **resetToDefault**<br>This value shall indicate to the server that it is requested to reset the input signal, internal parameter or output signal referenced by the inputOutputLocalIdentifier to its default state.<br>Number of controlState bytes in request: 0<br>Number of controlState bytes in pos. response: 0 |
| 02 | **freezeCurrentState**<br>This value shall indicate to the server that it is requested to freeze the current state of the input signal, internal parameter or output signal referenced by the inputOutputLocalIdentifier.<br>Number of controlState bytes in request: 0<br>Number of controlState bytes in pos. response: 0 |
| 03 | **shortTermAdjustment**<br>This value shall indicate to the server that it is requested to adjust the input signal, internal parameter or output signal referenced by the inputOutputLocalIdentifier 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).<br>Number of controlState bytes in request: depends on the dataIdentifier<br>Number of controlState bytes in pos. response: depends on the dataIdentifier |
| 04 – FF | (Reserved by ISO) |

### 6.12.3Implementation rules

There are no special implementation rules for this service.

## 6.13 RoutineControl (31)

This service is used by the client to start/stop a routine and request routine results in the server's memory. The routine is identified by a 2-byte RoutineIdentifier.

### 6.13.1 Message Format

Request:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | RequestServiceIdentifier | M | 31 |
| #2 | RoutineControlType | M | 00-FF |
| | RoutineIdentifier [ ] = [ | | |
| #3 | byte#1 (MSB) | M | 00-FF |
| #4 | byte#2 ] | M | 00-FF |
| | RoutineControlOptionRecord [ ] = [ | | |
| #5 | routineControlOption#1 | C/U | 00-FF |
| : | : | : | : |
| #n | routineControlOption#m ] | C/U | 00-FF |
| C: This parameter is user optional to be present for sub-function parameter startRoutine and stopRoutine. | | | |

Positive Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | PositiveResponseServiceIdentifier | M | 71 |
| #2 | RoutineControlType | M | 00-7F |
| | RoutineIdentifier [ ] = [ | | |
| #3 | byte#1 (MSB) | M | 00-FF |
| #4 | byte#2 ] | M | 00-FF |
| | RoutineStatusrecord [ ] = [ | | |
| #5 | routineStatus#1 | U | 00-FF |
| : | : | : | : |
| #n | routineStatus#m ] | U | 00-FF |

Negative Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | NegativeResponseServiceIdentifier | M | 7F |
| #2 | OriginalRequestServiceIdentifier | M | 31 |
| #3 | NegativeResponseCode | M | (see below) |

Negative Response Codes:

| NRC (hex) | Error text |
|---|---|
| 12 | **subFunctionNotSupported**<br>This code is returned if the requested sub-function is not supported. |
| 13 | **incorrectMessageLengthOrInvalidFormat**<br>The length of the message is wrong |
| 22 | **conditionsNotCorrect**<br>This code shall be returned if the criteria for the request RoutineControl are not met. |
| 24 | **requestSequenceError**<br>This code shall be returned if the 'stopRoutine' or 'requestRoutineResults' sub-function is received without first receiving a 'startRoutine' for the requested routineIdentifier. |
| 31 | **requestOutOfRange**<br>This code shall be returned if:<br>1. The server does not support the requested routineIdentifier,<br>2. The user optional routineControlOptionRecord contains invalid data for the requested routineIdentifier. |
| 33 | **securityAccessDenied**<br>This code shall be sent if this code shall be returned if a client sends a request with a valid secure routineIdentifier and the server's security feature is currently active. |
| 72 | **generalProgrammingFailure**<br>This return code shall be sent if the server detects an error when performing a routine, which accesses server internal memory. An example is when the routine erases or programms a certain memory location in the permanent memory device (e.g. Flash Memory) and the access to that memory location fails. |

### 6.13.2Parameter Options

The following table specifies predefined values for the RoutineControlType parameter.

| Option (hex) | Description |
|---|---|
| 01 | **startRoutine**<br>This parameter specifies that the server shall start the routine specified by the routineIdentifier. |
| 02 | **stopRoutine**<br>This parameter specifies that the server shall stop the routine specified by the routineIdentifier. |
| 03 | **requestRoutineResults**<br>This parameter specifies that the server shall return result values of the routine specified by the routineIdentifier. |

### 6.13.3Implementation Rules

There are no special implementation rules for this service.

## 6.14 WriteDataByIdentifier (2E)

The WriteDataByIdentifier service is used by the client to write record data values to a server. The data record is identified by a DataIdentifier. Possible use-cases for this service are:

Clear non-volatile memory

Reset learned values

Set option content

Set Vehicle Identification Number (VIN)

### 6.14.1 Message Format

Request:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | RequestServiceIdentifier | M | 2E |
| | DataIdentifier [ ] = [ | | |
| #2 | byte#1 (MSB) | M | 00-FF |
| #3 | byte#2 ] | M | 00-FF |
| | DataRecord [ ] = [ | | |
| #4 | data#1 | M | 00-FF |
| : | : | : | : |
| #m+3 | data#m ] | U | 00-FF |

Positive Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | PositiveResponseServiceIdentifier | M | 6E |
| | DataIdentifier [ ] = [ | | |
| #2 | byte#1 (MSB) | M | 00-FF |
| #3 | byte#2 ] | M | 00-FF |

Negative Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | NegativeResponseServiceIdentifier | M | 7F |
| #2 | OriginalRequestServiceIdentifier | M | 2E |
| #3 | NegativeResponseCode | M | (see below) |

Negative Response Codes:

| NRC (hex) | Error text |
|---|---|

| NRC (hex) | Error text |
|---|---|
| 13 | **incorrectMessageLengthOrInvalidFormat**<br>The length of the message is wrong. |
| 22 | **conditionsNotCorrect**<br>This response code shall be sent if the operating conditions of the server are not met to perform the required action. |
| 31 | **requestOutOfRange**<br>This response code shall be sent if<br>1. The dataIdentifier in the request message is not supported in the server or the dataIdentifier is supported for read only purpose (via ReadDataByIdentifier service).<br>2. Any data transmitted in the request message after the dataIdentifier is invalid (if applicable to the node). |
| 33 | **securityAccessDenied**<br>This code shall be sent if the dataIdentifier, which reference a specific address, is secured and the server is not in an unlocked state. |
| 72 | **generalProgrammingFailure**<br>This return code shall be sent if the server detects an error when writing to a memory location. |

### 6.14.2Parameter Options

Predefined DataIdentifiers are defined as specified in Annex B.

### 6.14.3Implementation Rules

It is the system supplier's responsibility that the server conditions are met when performing any action requested by this service.

## 6.15 RequestDownload (34)

This 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.

### 6.15.1 Message Format

Request:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | RequestServiceIdentifier | M | 34 |
| #2 | DataFormatIdentifier | M | 00-FF |
| #3 | addressAndLengthFormatIdentifier | M | 44 |
| | MemoryAddress [ ] = [ | | |
| #4 | byte#1 (MSB) | M | 00-FF |
| #5 | byte#2 | M | 00-FF |
| #6 | byte#3 | M | 00-FF |
| #7 | byte#4 (LSB)] | M | 00-FF |
| | MemorySize [ ] = [ | | |
| #8 | byte#1 (MSB) | M | 00-FF |
| #9 | byte#2 | M | 00-FF |
| #10 | byte#3 | M | 00-FF |
| #11 | byte#4 (LSB) ] | M | 00-FF |

Positive Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | PositiveResponseServiceIdentifier | M | 74 |
| #2 | LengthFormatIdentifier | M | 00-F0 |
| | MaxNumberOfBlockLength [ | | |
| #3 | byte#1 (MSB) | M | 00-FF |
| : | : | : | : |
| #n | byte#m ] | M | 00-FF |

Negative Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | NegativeResponseServiceIdentifier | M | 7F |
| #2 | OriginalRequestServiceIdentifier | M | 34 |
| #3 | NegativeResponseCode | M | (see below) |

Negative Response Codes:

| NRC (hex) | Error text |
|---|---|
| 13 | **incorrectMessageLengthOrInvalidFormat**<br>The length of the message is wrong |
| 22 | **conditionsNotCorrect**<br>This return code shall be sent 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. |
| 31 | **requestOutOfRange**<br>This return code shall be sent if<br>1. The specified dataFormatIdentifier is not valid.<br>2. The specified addressAndLengthFormatIdentifier is not valid.<br>3. The specified memoryAddress/memorySize is not valid. |
| 33 | **securityAccessDenied**<br>This return code shall be sent if the server is secure (for server's that support the SecurityAccess service) when a request for this service has been received. |
| 70 | **uploadDownloadNotAccepted**<br>This response code indicates that an attempt to download to a server's memory cannot be accomplished due to some fault conditions. |

### 6.15.2 Parameter Options

The MemoryAddress parameter length shall be 4Byte.

The address may represent a physical address location within the ECU, but addresses may also be mapped (e.g. to implement a memory selector in upper bits). However, if the address parameter does not correspond to the physical location, the mapping shall be documented within the ECU diagnostic specification.

The DataFormatIdentifier consists of 2 nibbles, the left one containing the compression method, the right one containing the encryption method. If no compression or encryption applies, the corresponding parameter shall be set to 0.

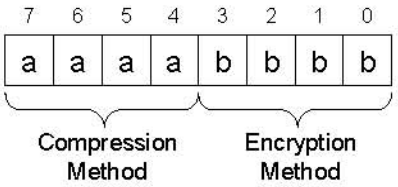



*Figure -11: Structure of DataFormatIdentifier*

At least a DataFormatIdentifier of 00 shall be supported.

The addressAndLengthFormatIdentifier is a one byte value with each nibble coded separately

- bit 7 – 4: Length (number of bytes) of the memorySize parameter
- bit 3 – 0: Length (number of bytes) of the memoryAddress parameter

The parameter memoryAddress is the starting address of the server memory where the data is to be written to.

The memorySize parameter shall be used by the server to compare the uncompressed memory size with the total amount of data transferred during the TransferData service. The number of bytes used for this size is defined by the high nibble (bit 7 – 4) of the addressFormatIdentifier.

The LengthFormatIdentifier is a one byte value with each nibble coded separately:

- bit 7-4: Length (number of bytes) of the maxNumberOfBlockLength parameter
- bit 3-0: reserved by document to be set to 0 hex

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 hex.

The parameter maxnumberOfBlockLength is used by the requestDownload positive response message to inform the client how many data bytes shall be included in each Transferata 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 to the server.

### 6.15.3Implementation Rules

There are no special implementation rules for this service.

## 6.16 RequestUpload (35)

This service is used by the client to initiate a data transfer from the serve 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.

### 6.16.1 Message Format

Request:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | RequestServiceIdentifier | M | 35 |
| #2 | DataFormatIdentifier | M | 00-FF |
| #3 | AddressAndLengthFormatIdentifier | M | 24 |
| | MemoryAddress [ ] = [ | | |
| #4 | byte#1 (MSB) | M | 00-FF |
| : | : | : | : |
| #7 | byte#m ] | C1 | 00-FF |
| | MemorySize [ ] = [ | | |
| #8 | byte#1 (MSB) | M | 00-FF |
| : | : | : | : |
| #9 | byte#k ] | C2 | 00-FF |
| C1 = The presence of this parameter depends on address length information parameter of the addressAndLengthFormatIdentifier<br>C2 = The presence of this parameter depends on the memory size length information of the addressAndLengthFormatIdentifier. | | | |

Positive Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | PositiveResponseServiceIdentifier | M | 75 |
| #2 | LengthFormatIdentifier | M | 00-F0 |
| | MaxNumberOfBlockLength [ | | |
| #3 | byte#1 (MSB) | M | 00-FF |
| : | : | : | : |
| #n | byte#m ] | M | 00-FF |

Negative Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | NegativeResponseServiceIdentifier | M | 7F |
| #2 | OriginalRequestServiceIdentifier | M | 35 |
| #3 | NegativeResponseCode | M | (see below) |

Negative Response Codes:

| NRC (hex) | Error text |
|---|---|
| 13 | **incorrectMessageLengthOrInvalidFormat**<br>The length of the message is wrong |
| 22 | **conditionsNotCorrect**<br>This return code shall be sent 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. |
| 31 | **requestOutOfRange**<br>This return code shall be sent if<br>1. The specified dataFormatIdentifier is not valid.<br>2. The specified addressAndLengthFormatIdentifier is not valid.<br>3. The specified memoryAddress/memorySize is not valid. |
| 33 | **securityAccessDenied**<br>This return code shall be sent if the server is secure (for server's that support the SecurityAccess service) when a request for this service has been received. |
| 70 | **uploadDownloadNotAccepted**<br>This response code indicates that an attempt to download to a server's memory cannot be accomplished due to some fault conditions. |

### 6.16.2 Paramter Options:

The MemoryAddress parameter length shall be 4Byte.

The address may represent a physical address location within the ECU, but addresses may also be mapped (e.g. to implement a memory selector in upper bits). However, if the address parameter does not correspond to the physical location, the mapping shall be documented within the ECU diagnostic specification.

The DataFormatIdentifier consists of 2 nibbles, the left one containing the compression method, the right one containing the encryption method. If no compression or encryption applies, the corresponding parameter shall be set to 0.

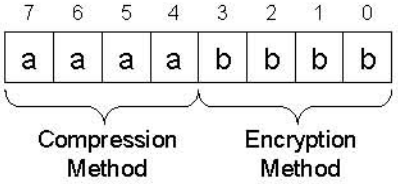



*Figure -12: Structure of DataFormatIdentifier*

The addressAndLengthFormatIdentifier is a one byte value with each nibble coded separately

- bit 7 – 4: Length (number of bytes) of the memorySize parameter
- bit 3 – 0: Length (number of bytes) of the memoryAddress parameter

The parameter memoryAddress is the starting address of the server memory where the data is to be written to.

The memorySize parameter shall be used by the server to compare the uncompressed memory size with the total amount of data transferred during the TransferData service. The number of bytes used for this size is defined by the high nibble (bit 7 – 4) of the addressFormatIdentifier.

### 6.16.3Implementation Rules:

There are no special implementation rules for this service.

## 6.17 TransferData (36)

This 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 within this document is fixed to download only.

### 6.17.1 Message Format

Request:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | RequestServiceIdentifier | M | 36 |
| #2 | BlockSequenceCounter | M | 00-FF |
| #3<br>:<br>#n | TransferRequestParameterRecord [ ] = [<br>transferRequestParameter#1<br>:<br>transferRequestParameter#m ] | C<br>:<br>U | 00-FF<br>:<br>00-FF |
| C = Conditional: this parameter is mandatory if a download is in progress. | | | |

Positive Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | PositiveResponseServiceIdentifier | M | 76 |
| #2 | BlockSequenceCounter | M | 00-FF |
| #3<br>:<br>#n | TransferResponseParameterRecord [ ] = [<br>transferResponseParameter#1<br>:<br>transferResponseParameter#m ] | C<br>:<br>U | 00-FF<br>:<br>00-FF |
| C = Conditional: this parameter is mandatory if an upload is in progress. | | | |

Negative Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | NegativeResponseServiceIdentifier | M | 7F |
| #2 | OriginalRequestServiceIdentifier | M | 36 |
| #3 | NegativeResponseCode | M | (see below) |

Negative Response Codes:

| NRC (hex) | Error text |
|---|---|
| 13 | **incorrectMessageLengthOrInvalidFormat**<br>The length of the message is wrong (e.g., message length does not meet requirements of maxNumberOfBlockLength parameter returned in the positive response to request-Download). |
| 24 | **requestSequenceError**<br>The server shall use this response code<br>1. If the RequestDownload or RequestUpload service is not active when a request for this service is received.<br>2. 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 Request-Downlod or RequestUpload service.<br>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. |
| 31 | **requestOutOfRange**<br>This return code shall be sent if the transferRequestParameterRecord contains additional control parameters (e.g. additional address information) and this control information is invalid. |
| 71 | **transferDataSuspended**<br>This return code shall be sent if<br>1. The response code indicates that a data transfer operation was halted due to some fault.<br>2. The download module length does not meet the requirements of the memorySize parameter sent in the request message of the requestDownload service. |
| 72 | **generalProgrammingFailure**<br>This return code shall be sent 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. |
| 73 | **wrongBlockSequenceCounter**<br>This return code shall be sent if the server detects an error in the sequence of the block-SequenceCounter.<br>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. |
| 92-93 | **voltageTooHigh / voltageTooLow**<br>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). |

### 6.17.2Parameter Options

The BlockSequenceCounter allows the diagnostic server to detect and handle transmission errors. It shall start at 1. Its value is incremented by 1 for each subsequent request of this service. When the block number 255 (0xFF) is transmitted, it shall continue with block number 0.

### 6.17.3Implementation Rules

There are no special implementation rules for this service.

## 6.18 RequestTransferExit (37)

This service is used by the client to terminate a data transfer between client and server.

### 6.18.1 Message Format

Request:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | RequestServiceIdentifier | M | 37 |

Positive Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | PositiveResponseServiceIdentifier | M | 77 |

Negative Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | NegativeResponseServiceIdentifier | M | 7F |
| #2 | OriginalRequestServiceIdentifier | M | 37 |
| #3 | NegativeResponseCode | M | |

Negative Response Codes:

| NRC (hex) | Error text |
|---|---|
| 13 | **incorrectMessageLengthOrInvalidFormat**<br>The length of the message is wrong. |
| 24 | **requestSequenceError**<br>This return code shall be sent if:<br>1. The programming process is not completed when a request for this service is received.<br>2. The RequestDownload or RequestUpload service is not active. |

### 6.18.2 Parameter Options

There are no parameters for this service.

### 6.18.3 Implementation Rules

There are no special implementation rules for this service.

## 6.19 WriteMemoryByAddress (3D)

This service is used to write record values into the server's memory. The location of the data and the size of the data block are identified by Memory Address and Memory Size parameters. As AddressAndLengthFormatIdentifier the value 0x24 shall be used.

### 6.19.1 Message Format

Request:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | RequestServiceIdentifier | M | 3D |
| #2 | AddressAndLengthFormatIdentifier | M | 24 |
| | MemoryAddress [ ] = [ | | |
| #3 | byte#1 (MSB) | M | 00-FF |
| #4 | byte#2 | M | 00-FF |
| #5 | byte#3 | M | 00-FF |
| #6 | byte#4 ] | M | 00-FF |
| | MemorySize [ ] = [ | | |
| #7 | byte#1 (MSB) | M | 00-FF |
| #8 | byte#2 ] | M | 00-FF |
| | Datarecord [ ] = [ | | |
| #n-(r-1) | data#1 | M | 00-FF |
| : | : | : | : |
| #n | data#r ] | U | 00-FF |

Positive Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | PositiveResponseServiceIdentifier | M | 7D |
| #2 | AddressAndLengthFormatIdentifier | M | 24 |
| | MemoryAddress [ ] = [ | | |
| #3 | byte#1 (MSB) | M | 00-FF |
| #4 | byte#2 | M | 00-FF |
| #5 | byte#3 | M | 00-FF |
| #6 | byte#4 ] | M | 00-FF |
| | MemorySize [ ] = [ | | |
| #7 | byte#1 (MSB) | M | 00-FF |
| #8 | byte#2 ] | M | 00-FF |

Negative Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | NegativeResponseServiceIdentifier | M | 7F |
| #2 | OriginalRequestServiceIdentifier | M | 3D |
| #3 | NegativeResponseCode | M | (see below) |

Negative Response Codes:

| NRC (hex) | Error text |
|---|---|
| 13 | **incorrectMessageLengthOrInvalidFormat**<br>The length of the message is wrong. |
| 22 | **conditionsNotCorrect**<br>This response code shall be sent if the operating conditions of the server are not met to perform the required action. |
| 31 | **requestOutOfRange**<br>1. Any memory address within the interval [$MA, ($MA + $MS -$1)] is invalid.<br>2. Any memory address within the interval [$MA, ($MA + $MS -$1)] is restricted.<br>3. The memorySize parameter value in the request message is greater than the maximum value supported by the server.<br>4. The specified addressAndLengthFormatIdentifier is not valid. |
| 33 | **securityAccessDenied**<br>This code shall be sent if any memory address within the interval [$MA, ($MA + $MS -$1)] is secure and the server is locked. |
| 72 | **generalProgrammingFailure**<br>This return code shall be sent if the server detects an error when writing to a memory location. |

### 6.19.2Parameter Options

The MemoryAddress parameter length shall be 4Byte.

The address may represent a physical address location within the ECU, but addresses may also be mapped (e.g. to implement a memory selector in upper bits). However, if the address parameter does not correspond to the physical location, the mapping shall be documented within the ECU diagnostic specification.

The MemorySize parameter length shall be 2Byte.

### 6.19.3Implementation Rules

An ECU shall perform appropriate security mechanisms for direct memory access.

## 6.20 TesterPresent (3E)

This service shall be used to indicate to a server that the client is present. This service is required in the absence of other UDS services to prevent servers from automatically returning to normal operation and stop communication.

### 6.20.1 Message Format

It is recommended to send 3E 80 (SPRMIB = TRUE) functionally.

Request:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | RequestServiceIdentifier | M | 3E |
| #2 | ZeroSubFunction | M | 00 |

Positive Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | PositiveResponseServiceIdentifier | M | 7E |
| #2 | ZeroSubFunction | M | 00 |

Negative Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | NegativeResponseServiceIdentifier | M | 7F |
| #2 | OriginalRequestServiceIdentifier | M | 3E |
| #3 | NegativeResponseCode | M | (see below) |

Negative Response Codes:

| NRC (hex) | Error text |
|---|---|
| 12 | **subFunctionNotSupported**<br>Send if the sub-function parameter in the request message is not supported. |
| 13 | **incorrectMessageLengthOrInvalidFormat**<br>The length of the message is wrong. |

### 6.20.2Parameter Options

The ZeroSubFunction is defined as follows:

| Option (hex) | Description |
|---|---|
| 00 | **zeroSubFunction**<br>This parameter value is used to indicate that no sub-function value beside the suppressPosRspMsgIndicationBit is supported by this service. |
| 01-7F | (Reserved by ISO) |

### 6.20.3Implementation Rules

There are no special implementation rules for this service.

## 6.21 AccessTimingParameter (83)

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

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.

NOTE: Currently it is not defined which parameters of this service are supported. The service was introduced to fulfill future requirements.

NOTE: This service is not supported by the diagnostic protocol on CAN [ISO3].

### 6.21.1 Message Format

Request:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | RequestServiceIdentifier | M | 83 |
| #2 | TimingParameterAccessType | M | 01, 02, 03, 04 |
| | TimingParameterRequestRecord [ | | |
| #3 | byte #1 | C | 00-FF |
| : | : | : | : |
| #n | byte #m ] | C | 00-FF |
| C: The TimingParameterRequestRecord is only present if timingParameterAccessType = setTimingParametersToGivenValues. The structure and content of the TimingParameterRequestRecord is data link layer dependend. | | | |

Positive Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | PositiveResponseServiceIdentifier | M | C3 |
| #2 | DTCSettingType | M | 01, 02, 03, 04 |
| | TimingParameterResponseRecord [ | | |
| #3 | byte #1 | C | 00-FF |
| : | : | : | : |
| #n | byte #m ] | C | 00-FF |
| C: The TimingParameterResponseRecord is only present if timingParameterAccessType = readExtendedTimingParameterSet or readCurrentlyActiveTimingParameters. The structure and content of the TimingParameterResponseRecord is data link layer dependend. | | | |

Negative Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | NegativeResponseServiceIdentifier | M | 7F |
| #2 | OriginalRequestServiceIdentifier | M | 83 |
| #3 | NegativeResponseCode | M | (see below) |

Negative Response Codes:

| NRC (hex) | Error text |
|---|---|
| 12 | **subFunctionNotSupported**<br>Send if selected Timing Parameter Access Type is not supported. |
| 13 | **incorrectMessageLengthOrInvalidFormat**<br>The length of the message is wrong. |
| 22 | **conditionsNotCorrect**<br>This code shall be returned if the criteria for the request AccessTimingParameter are not met. |
| 31 | **requestOutOfRange**<br>This code shall be sent if the TimingParameterRequestRecord contains invalid timing parameter values. |

## 6.21.2Parameter Options

The parameter timingParameterAccessType is defined as follows:

| Option (hex) | Description |
|---|---|
| 01 | **readExtendedTimingParameterSet**<br>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.<br>If the read access to the timing parameter set is successful, the server shall send an AccessTimingParameter response primitive with the positive response parameters.<br>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.<br>This sub-function is used to provide an extra set of timing parameters for the currently active diagnostic session.<br>With the timingParameterAccessType = setTimingParametersToGivenValues only this set (read by timingParameterAccessType = readExtendedTimingParameterSet) of timing parameters can be set. |
| 02 | **setTimingParametersToDefaultValues**<br>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).<br>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. |
| 03 | **readCurrentlyActiveTimingParameters** |

| Option (hex) | Description |
|---|---|
| | Upon receiving an AccessTimingParameter indication primitive with timingParameterAccessType = readCurrentlyActiveTimingParameters, the server shall read the currently used timing parameters.<br><br>If the read access to the timing parameters is successful, the server shall send an AccessTimingParameter response primitive with the positive response parameters.<br><br>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. |
| 04 | **setTimingParametersToGivenValues**<br><br>Upon receiving an AccessTimingParameter indication primitive with timingParameterAccessType = setTimingParametersToGivenValues, the server shall check if the timing parameters can be changed under the present conditions.<br><br>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).<br><br>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.<br><br>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. |

### 6.21.3 Implementation Rules:

Note: The following rules apply only for non-CAN busses.

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 parmeter after the transmission/reception of the request message.

## 6.22 ControlDTCSetting (85)

This service shall be used by a diagnostic test tool to enable and disable the setting of Diagnostic Trouble Codes, DTCs, in the ECU(s).

The ControlDTCSetting service shall be used by a client to stop or resume the setting of diagnostic trouble codes in the server(s).

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

### 6.22.1 Message Format

Request:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | RequestServiceIdentifier | M | 85 |
| #2 | Sub-function=[<br>DTCSettingType] | M | 00-7F |

Positive Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | PositiveResponseServiceIdentifier | M | C5 |
| #2 | DTCSettingType | M | 00-7F |

Negative Response:

| Byte | Name | Cvt | Value (hex) |
|---|---|---|---|
| #1 | NegativeResponseServiceIdentifier | M | 7F |
| #2 | OriginalRequestServiceIdentifier | M | 85 |
| #3 | NegativeResponseCode | M | (see below) |

Negative Response Codes:

| NRC (hex) | Error text |
|---|---|
| 12 | **subFunctionNotSupported**<br>Send if the sub-function parameter in the request message is not supported |
| 13 | **incorrectMessageLengthOrInvalidFormat**<br>The length of the message is wrong. |
| 22 | **conditionsNotCorrect**<br>Used when the server is in a critical normal mode activity and therefore cannot perform the requested DTC control functionality. |
| 31 | **requestOutOfRange**<br>The server shall use this response code, if it detects an error in the DTCSettingControlOptionRecord. |

### 6.22.2Parameter Options

The DtcSettingType parameter is defined as follows:

| Option (hex) | Description |
|---|---|
| 01 | **on**<br>The server(s) shall resume the setting of diagnostic trouble codes according to normal operating conditions |
| 02 | **off**<br>The server(s) shall stop the setting of diagnostic trouble codes. |

### 6.22.3Implementation Rules

The setting of DTCs in the ECU shall be switched off directly after the request ControlDTCSetting with setting mode 2 (switch off) is received (Option: Suppress positive response message indication bit is set) or after successful transmission of the positive response (Option: Suppress positive response message indication bit is not set).

The setting of DTCs in the ECU shall be switched on again, when one of the following events occurs:

- The request ControlDTCSetting with setting mode 1 (switch on) is received (Option: Suppress positive response message indication bit is set) or after successful transmission of the positive response (Option: Suppress positive response message indication bit is not set).
- The ECU power supply voltage is down (reset or ECU power down event).
- The diagnostic session changes from Extended Diagnostic Session to any other Diagnostic Session.

Hint: The services ReadDTCInformation – Report DTC by status mask (19 02), ReadDtcInforma-tion – Report supported DTC (19 0A) and ClearDiagnosticInformation (14) shall not be affected by this service.

# 7 General Diagnostic Requirements

All requirements which are described in this chapter must be supported by all ECUs supporting diagnostics.

## 7.1 Prerequisites

Electrical control units which support diagnostic functionality shall include a self check and check of the peripherals (sensors, actuators, switches, wiring) and the network connection.

The electrical control units must be testable and configurable without connecting to sensors or actuators. In case of a malfunction the ECU shall support substitute values to ensure the overall system continues to work.

The diagnostic requirements of development, manufacturing and service are to be considered.

## 7.2 ECU Identification

The support of identification parameters are ECU specific and have to be specified per ECU.

For reading the ECU identification, the diagnostic service ReadDataByIdentifier (22) shall be used. If the ECU supports writing an ECU identification block, the service WriteDataByIdentifier (2E) shall be used. The identification data block shall be symmetric for reading and writing, and the data identifier representing the identification data block shall be the same for reading and writing.

The ECU shall support to read out all relevant information regarding internal and peripheral data values by an external test device. These values are typically non-constant and changing over the time.

Considering the audience of diagnostics, the requirements of development, manufacturing and service are different. A superset of the relevant information shall be available to the external test devices.

Dynamic data reading shall return the data values at the time of the read request.

Examples for dynamic data:

- Input parameters (sensors, switches, keys, controls…)
- Output data
- Actuator data
- Data supplied by other ECU via the network
- Snapshot data stored as fault memory additional information

If possible, the format and the interpretation rule for diagnostic data shall be identical to the data of network communication (CAN matrix) and CCP/XCP (A2L).

The service ReadDataByIdentifier (22) shall be used to read out dynamic data.

In special cases, the ECU developers may access dynamic data also with the service ReadMemoryByAddress (23). However, this service shall not be used in production.

## 7.3 Fault Memory

### 7.3.1 General Requirements

The diagnostic failure detection shall cover a complete checking of the ECU functions and of the peripheral components (sensors, actuators, switches and wiring). Additionally, all network error situations shall be monitored. There shall be no restrictions on error detection due to ECU states.

Any detected failure is to be recorded within the fault memory by setting the corresponding DTC (Diagnostic Trouble Code).

The error detection shall continue to work while the diagnostic communication with an external test device is active. There shall be no restrictions of error detection in any diagnostic session (besides ECU programming). The error detection is beyond scope of the FaultMemory.

### 7.3.2 Requirements for Emission-related ECUs

For emission-related ECU, the legislative OBD requirements (as specified in [ISO2]) must be covered in addition to the requirements of this specification.

It is not meant to have two separate implementations, one for emission-related diagnostics and one for enhanced diagnostics. However, it must be ensured that the enhanced safty critics implementation covers all legal requirements. In case of conflict, the legal requirements for emission-related diagnostics take priority over this specification.

### 7.3.3 Fault Management

The purpose of managing diagnostic trouble codes is to detect the source of the defect or failure. It is essential, that the ECU reports the exact (and irreducible) source for the defect so that the service technician is able to identify and replace (only) the faulty part(s). The ECU documentation shall contain an extensive documentation about each supported DTC. The documentation shall comprise the information to localize the defect and a repair instruction.

There shall be exact one DTC for one defect or failure. If a DTC is no longer supported within the ECU, this DTC shall not be reused to describe any other defect or failure.

All DTCs shall be stored in non-volatile memory. Before storing, the defect detection shall debounce the fault symptoms.

The DTCs shall be reported to the external test device if requested by the corresponding diagnostic services. If there is a need for additional failure information, the fault memory shall manage this information in addition to the DTCs. This extended failure information shall comprise data about the failure detection criteria and the fault path. For each occurrence of a DTC, the extended information shall be updated.

A DTC can just be clearead by tester with service $14.

The AgingCriteriaFulfilled property becomes true if a specified number of ‘test passed’ events were signaled (e.g. 40 engine warm-ups without another detected malfunction).

The FaultRecordOverwritten property is set to ‘true’ after a ClearDiagnosticInformation service was requested and before the next request of any service.

The StartOfNextOperatingCycle property is set to ‘true’ with starting a new Operation cycle.

The WarningIndicatorOffCriteriaFullfilled criteria can be used to set the status of the WarningIndicator to 'off'. In the same way the WarningIndicatorOnCriteriaFullfilled criteria can be used to set the status of the WarningIndicator to 'on'. It is also possible that the warning indicator is requested or not.

The DtcConfirmationCriteriaFullFilled checks if the number of Trips with status 'TestFailed' is greater than the ConfirmedLevel.

### 7.3.4 Layout of DTCs

The DTCs of GAC should have the following properties:

| Property | DataType | Description |
|---|---|---|
| Error Text | String | An understandable error message. This message will be displayed on the tester's display. |
| Set condition | String | Specifies the environment conditions when this trouble appears. |
| Reset condition | String | Specifies the conditions under which this DTC gets deleted from the Fault Memory. |
| Corrective action | String | This gives the user a choice to have an approximate direction where the error appeared. |
| Special instruction | String | This field gives the user a choice to explain the corrective action more exactly. |
| DTCPriority | Unsigned Integer | Value (0-255) defining the priority of this DTC, where 0 is the maximum value. The following priority should be applied:<br>0 invalid value, not available<br>1 Error: This fault has a heavy impact on the car availability. The car is not ready to start.<br>2 Error: This fault requires a direct visit of the repair shop.<br>3 Error: This fault does NOT require a direct visit of the repair shop, but can be done within a service.<br>4 Error: This fault leads to a need for action before the car can be moved; maybe the availability of the car is limited.<br>5-15 (reserved)<br>16 Message: This fault has influence on the abrasion level and is therefore relevant for the service (e.g. fill level/ abrasion critical, aging etc).<br>17 Message: This fault has influence on the comfort, but does not influence the availability of the car.<br>18 General message<br>19 Message for development<br>20 – 255 to be defined |
| SelfHealingLimit | Unsigned Integer | Value (0-255) defining the limit when SelfHealing is applied. |
| Error Name | String | A value which is used during code generation. Must fulfill variable definition standards in C. |

Example DTC:

| Property | Example value |
|---|---|
| Error Text | Temperature too hot. |
| Set condition | This DTC appears if the temperature in the air intake > 50 °C. |
| Reset condition | Off after 3 driving cycle without fault. |
| Corrective action | Check wiring |
| Special instruction | The wiring at point A is broken. |
| DTCPriority | 2 (Error, which requires a direct visit of the repair shop.) |
| SelfHealingLimit | 3 |
| Error Name | TEMPERATURE_TOO_HOT |

### 7.3.5 DTC priorization

In general, ECUs are not designed to store extensive DTC status information in EEPROM for each supported DTC at the same time. Thus, a kind of DTC overwritten rule is required.

DTC overwritten rule is specified in [SS]

### 7.3.6 Failure Detection

For failure detection, there are two opposite targets: On the one side, failure detection shall be fast to allow a quick repair check in production and service. On the other side, failure detection shall be safe avoiding any false failure recognition. In practice, a good compromise shall be taken.

Failure detection shall be implemented using fault paths. The detection implementation is isolated in independent routines, which are processed independently from the current operation thread of the ECU. Fault paths checks shall be executed cyclically. The complete ECU functionality shall be covered.

Some tests can only run if the ECU is in a specific state, i.e. the test input parameters have some predefined values. Consequently, some fault path checks require specific test entry conditions. If these entry conditions are not met, the fault path check cannot be executed. The readiness flag of the DTC status specifies if a fault path for this DTC is already processed in the current driving cycle.

Before recording a fault symptom as an error, it shall be filtered and debounced to avoid false failures. An error shall only be set when multiple measures indicate a failure.

This is a typical structure of a fault path:

- Check the test conditions: Ensure that the entry conditions are met so that the fault check may be executed.
- Inspect the fault symptoms: Detect and debounce any potential failure.
- Update the DTC states: Set or reset the current state of DTCs (e.g. to active, but also to no more active)

A fault path checking shall take place as soon as the system is in a steady state. After startup, the ECU should be steady after a period of at most 5 seconds. A normal voltage jitter while starting the engine shall not lead to set any DTC.

Only the root cause DTC shall be kept. If some failures are detected due to low or over voltage, only the DTC representing the low or over voltage shall be recorded.

If a fault path requires special test prerequisites (e.g. a special operation mode with extended diagnostic capabilities), it shall be possible to run the fault path by starting a remote routine by an external test device. If it is possible to meet the preconditions of a fault path by an automatic activation of actuators, the complete test shall be implemented as a remote routine which can be started by the external test device.

The process of failure detection happens completely outside of the Fault Memory. The result at the end of the failure detection process is then put into the Fault Memory.

### 7.3.7 DTC Status

The fault memory manages DTCs including its status information. This section defines the DTC Status Information Byte representing storage state of an individual DTC. The following table defines the meaning of each individual bit included in the DTC status byte.

| Status Byte | Description |
|---|---|
| 0 | **Test failed**<br>0x00 – false<br>0x01 – true<br>This bit shall indicate the result of the most recently performed test. A logical '1' shall indicate that the DTC test failed at time of request. A logical '0' shall indicate that the DTC test pass at time of request.<br>Reset to 0 after a call of ClearDiagnosticInformation. |
| 1 | **Test failed this operation cycle**<br>0x00 – false<br>0x01 – true<br>This bit shall indicate that a diagnostic test has reported a Test Failed result at any time during the current operation cycle.<br>Reset to 0 when a new operation cycle is initiated or after a call to Clear Diagnostic Information.<br>Once this bit is set to 1, it shall remain a 1 until a new operation cycle is started. |
| 2 | **Pending DTC**<br>0x00 – false<br>0x01 – true<br>This bit shall indicate whether or not a diagnostic test has reported a Test Failed 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 Pending DTC bit and the Test Failed This Operation Cycle bit are the same. The difference is that the Test Failed This Operation Cycle is cleared at the end of the current operation cycle and the Pending DTC bit is not cleared until an operation cycle has completed where the test has passed at least once and never failed.<br>Reset to 0 after a call of ClearDiagnosticInformation. |
| 3 | **Confirmed DTC**<br>0x00 – false<br>0x01 – true<br>This bit shall indicate whether a malfunction was detected enough times to warrant that the DTC is stored in long-term memory (Pending DTC has been set = '1' one or more times for emission relevant electronic control units). This information can be used by the external test tool to request additional diagnostic information such as Extended Data Records or Snapshot Records.<br>A Confirmed DTC does not indicate that the malfunction is present at the time of the request (Pending DTC or Test Failed can be used to determine if a malfunction is present at the time of the request.).<br>Reset to logical '0' after a call to Clear Diagnostic Information or after self-healing criteria has been satisfied or after discarding this DTC due to Fault Memory overflow. |
| 4 | **Test not completed since last clear** |

| Status Byte | Description |
|---|---|
| | 0x00 – false<br>0x01 – true<br>This bit shall indicate whether a DTC test has run to completion since the last time a call was made to Clear Diagnostic Information.<br>One (1) shall indicate that the DTC test has not run to completion.<br>If the test runs and passes or fails (Test Failed This Operation Cycle = 1) then the bit shall be set to a Zero (0) and latched.<br>Reset to One (1) after a call to Clear Diagnostic Information. |
| 5 | **Test failed since last clear**<br>0x00 – false<br>0x01 – true<br>This bit shall indicate whether a DTC test has ever returned a Test Failed = 1 result since the last time a call was made to Clear Diagnostic Information (latched Test Failed This Operation Cycle =1).<br>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. Reset to Zero (0) after a call to Clear Diagnostic Information. In contradiction to the Confirmed DTC this bit is not reset by self-healing criteria or when it was overwritten due to an overflow of the fault memory. |
| 6 | **Test not completed this operation cycle**<br>0x00 – false<br>0x01 – true<br>This bit shall indicate whether a DTC test has ever run and completed during the current operation cycle.<br>One (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.<br>Reset to 1 after a call to Clear Diagnostic Information. |
| 7 | **Warning indicator requested**<br>0x00 – false<br>0x01 – true<br>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 given system or particular DTC, this status shall default to a logic"0" state.<br>The bit is set to 1 when the performed test results 'failed'.<br>Reset to a logical '0' after a call to Clear Diagnostic Information. |

*Table -13 Definition of DTC Status Byte*

Current DTC：bit0 = 1, bit3 = 1;

When ECU detect the malfunction and meet DTC's set condition, Bit0,Bit3 will set to 1,at the same time the information of DTC will memory in the Volatile Memory and Non-Volatile memory.

History DTC：bit0 = 0, bit3 = 1。

It is recommended to use a value of 0x39 as a DTCStatusAvailabilityMask

For further information and examples see Annex D of [ISO1].

## 7.4 Dynamic Data

The ECU shall support to read current values with the service ReadDataByIdentifier (22).

In contrast to dynamic data, parameters are constant. However, the ECU shall support to read out and write all relevant parameters by an external test device.

## 7.5 Stored Data

The ECU shall support to read out and write all relevant Stored Data by an external test device. Typically, this information is stored in non-volatile memory.

Any Stored Data shall be readable and optionally writable by an external test device. For reading, the service ReadDataByIdentifier (22) shall be used and for writing, the service WriteDataByIdentifier (2E) shall be used. The data block shall be symmetric for reading and writing, and the local identifier representing the data block shall be the same for reading and writing.

The Serial Number is considered to be Stored Data. It shall be implemented for each ECU using the services specified above. Reading out the Serial Number shall already be implemented in the first sample ECU.

## 7.6 ECU Configuration (Variant Coding)

The target of ECU configuration is to reduce the variants of ECUs.

Any configuration can be implemented either by parameterization or coding, or by ECU programming. A coding approach has advantages for timing reasons (programming and restarting takes quite a long time) and for variant considerations (coding element contents are easier to handle than a number of data files). Consequently, a coding approach should be the preferred one, besides there are strong arguments for another solution.

This decision about implementing the ECU configuration has to be done in agreement with GAC.

Any ECU configuration parameter, measurement values and variant-building elements shall be readable and writable by an external test device. For reading, the service ReadDataByIdentifier (22) shall be used and for writing, the service WriteDataByIdentifier (2E) shall be used. The configuration block shall be symmetric for reading and writing, and the local identifier representing the configuration block shall be the same for reading and writing.

The configuration data shall be accepted immediately and stored in nonvolatile memory.

When delivered, the ECU shall have a well-defined and reasonable default configuration. The default configuration shall minimize the effort of reconfiguration in manufacturing or service. It shall be documented within the ECU diagnostic specification.

If the active configuration is not compliant to the ECU and its peripherals, an error shall be stored to point out this configuration error. An invalid ECU configuration shall not result in any destruction of vehicle components.

The maximum length of the variant coding string is not restricted. Only a few bits are valid variant coding strings. It is important that there is an external database (SQL, Excel, …) which stores the valid variant codings of an ECU. The supplier and the diagnostic development need to get consensus on the valid variant coding strings.

Every bit in the variant coding has a special meaning. On release of an ECU variant, the variant coding string should contain a valid default string. As an example in Figure-13, the second left bit is responsible for the steering (0 means steering left, 1 means steering right).

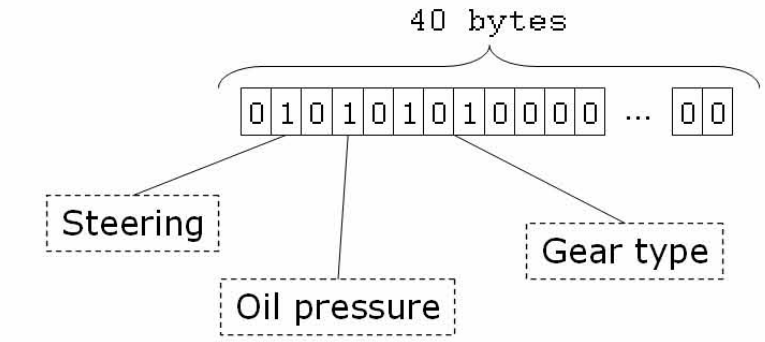



*Figure -13: Refinement of bits*

## 7.7 Simulation of Input Signals and Control of Output

An external test device shall be able to substitute values such as input signals and ECU internal function parameters. Additionally, it shall be able to control output values (actuator control) of the ECU. These functions shall be usable by the external test device without the need to download additional code into the ECU.

While the external test device is controlling the ECU functions, safety monitoring shall remain active for the outputs to avoid self destruction and personal injuries. In special cases, it may be possible to switch off the safety monitoring by an explicit diagnostic request. This feature has to be implemented in agreement with GAC.

If an adjustment is only designed for manufacturing purposes, the ECU shall implement an effective locking mechanism to ensure that these functions shall not be executed after end of line. These restricted functions and the locking mechanisms shall be documented in the ECU specific diagnostic specification.

For temporary adjustments, the service InputOuputControlByIdentifier (2F) shall be used for reading and controlling. If ECU data is modified permanently, the service WriteDataByIdentifier (2E) shall be used for writing and the service ReadDataByIdentifier (22) for reading.

It shall be possible to control every actuator using diagnostic services.

## 7.8 Remote Routines

Remote routines are intended to start predefined sequences within the server.

Examples for the use of remote routines:

Run an actuator in batch mode (e.g. test run for instrument cluster)

Perform initial calibration

Enable enhanced diagnostic functionality in parallel to normal operation

There are two different kinds of remote routines: Synchronous and asynchronous.

Synchronous routines are started when the diagnostic request is received and the positive response is sent after the routine finished its work. If the execution of the routine takes a while, the diagnostic server keeps the request active by sending a negative response with a response code 78. Consequently, the communication is blocked while the routine is executed.

Asynchronous routines are started when the diagnostic request is received and the positive response is sent to confirm the successful starting of the routine. The routine runs in parallel to the diagnostic communication. It is specific to the routine if it ends autonomously or if it has to be stopped by the external test device by sending a stop command. If such a routine returns any result data, this result data shall be transmitted with the final positive response.

Depending on the use-case of the routine, a synchronous or an asynchronous implementation is a better choice. It is up the specific routine which kind is selected. However, the kind of routine shall be documented within the ECU specific diagnostic specification. If such a routine creates any result data, it shall be picked up by executing the service Routine Control – requestRoutineResult (31 03).

For starting the routine, the service RoutineContro – startRoutinel (31 01) shall be used. If the ECU supports stopping the routine, the service RoutineControl – stopRoutine (31 02) shall be used.

There should be no sequence required for execution of remote routines (e.g. routine A has to be executed before routine B may be executed), besides there are functional requirements which require a sequence (e.g. for initialization purposes).

## 7.9 Memory Access

The diagnostic protocol allows to access memory via address, by reading and writing.

The direct memory access shall only be necessary (and allowed) in the development phase. The memory access has often a strong dependency to a specific software version, because the memory addresses of internal data can vary from software version to version. Therefore all data which is relevant for manufacturing and service shall be accessible by local identifiers

The direct memory access shall be implementer/supplier specific. It is recommend to secure these services (which require a security access) to avoid an unintentional access to the memory.

The ECU specific diagnostic specification shall contain a specification of the different memory areas and the corresponding security mechanisms for reading and writing.

For reading data from memory, the diagnostic service ReadMemoryByAddress (23) shall be used. For writing data to memory, the service WriteMemoryByAddress (3D) shall be used.

## 7.10 Security Access

The security access mechanism protects several diagnostic operations against unauthorized access.

If the ECU supports security access, it shall implement a reasonable algorithm for key generation. This algorithm shall be documented within a specific document which is accessible by only a small group of persons.

To ensure that the algorithm is kept secret, there should be a special algorithm during development time which is replaced just before the ECU gets released.

For implementing the security access, the services SecurityAccess-RequestSeed and SecurityAccess-SendKey shall be used.

## 7.11 Dynamically Define Identifiers

Dynamically defined data identifiers are used to get values from different identifiers with a single request.

Therefore dynamically defined identifiers are created by using service DynamicallyDefineDataIdentifier (2C) with subfunction DefineByIdentifier (0x01).

The server shall maintain the dynamically defined data record until the power down of the server or the ECU was resetted.

The reserver identifier range for dynamically defined identifiers is 0xF200-0xF3FF.

## 7.12 Periodic Identifiers

Periodic identifiers are one or more 1Byte values that identify data record(s) maintained by the server. The periodicDataIdentifier represents the low byte of a dataIdentifier out of the dataIdentifier range reserved for this service (F2xx hex), e.g. the periodicDataIdentifier E7 hex used in this service is the dataIdentifier F2E7 hex.

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.

The data of a periodicDataIdentifier is transmitted periodically (with updated data) at a rate determined by the transmissionMode parameter of the request.

The Number of periodicDataIdentifiers that can be supported simultaneously is limited to a maximum of 5. Exceeding this 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.

# Annex A. List of Negative Response Codes

This is the definition of negative response code values defines all negative response codes used within this specification. Each diagnostic service specifies applicable negative response codes but the diagnostic service implementation in the server may also utilize additional and applicable negative response codes specified in this annex. The only negative response code affecting the communication is response code 78 (requestCorrectlyReceived-ResponsePending).

The negative response codes are divided in three ranges:

00: PositiveResponse parameter value for server internal implementation.

01 – 7F: Communication related negative response codes

80 – FF: 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 utilized whenever response code 22 (conditionsNotCorrect) is listed as valid in order to report more specifically why the requested action can not be taken.

| NRC (hex) | Name/Definition |
|---|---|
| 00 | **positiveResponse**<br>This response code shall not be used in a negative response message. This positiveResponse parameter value is reserved for server internal implementation. |
| 01 – 0F | (Reserved by ISO) |
| 10 | **generalReject**<br>This response code indicates that the requested action has been rejected by the server.<br>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 response code be a general replacement for the response codes defined in this document. |
| 11 | **serviceNotSupported**<br>This response code indicates that the requested action will not be taken because the server does not support the requested service.<br>The server shall send this response code in case the client has sent a request message with a service identifier, which is either unknown or not supported by the server. 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. |
| 12 | **subFunctionNotSupported**<br>This response code indicates that the requested action will not be taken because the server does not support the service specific parameters of the request message.<br>The server shall send this response code 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. |
| 13 | **incorrectMessageLengthOrInvalidFormat**<br>This response code 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 |

| NRC (hex) | Name/Definition |
|---|---|
| | service or the format of the paramters do not match the prescribed format for the specified service. |
| 14 | **responseTooLong**<br>This response code shall be reported by the server if the response to be generated exceeds the maximum number of bytes available by the underlying network layer.<br>EXAMPLE This problem may occur when several DIDs at a time are requested and thecom-bination of all DIDs in the response exceeds the limit of the underlying transport protocol. |
| 15 – 20 | (Reserved by ISO) |
| 21 | **busyRepeatRequest**<br>This response code indicates that the server is temporarily too busy to perform the re-quested 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 de-layed by a timespecified in the respective implementation documents.<br>Example: In a multi-client environment the diagnostic request of one client might be blocked temporarily by a NRC $21 while a different client finishes a diagnostic task.<br>NOTE If the server is able to perform the diagnostic task but needs additional time to finish the task and prepare the response, the NRC $78 shall be used instead of NRC $21.<br>This response code 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. |
| 22 | **conditionsNotCorrect**<br>This response code indicates that the requested action will not be taken because the server prerequisite conditions are not met. |
| 23 | (Reserved by ISO) |
| 24 | **requestSequenceError**<br>This response code 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.<br>EXAMPLE A successful SecurityAccess service specifies a sequence of requestSeed and sendKey as susb-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 24 hex - . requestSequenceError. |
| 25 | **noResponseFromSubnetComponent**<br>This response code 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.<br>The noResponseFromSubnetComponent negative response shall be implemented by gate-ways 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.<br>This response code 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. |
| 26 | **failurePreventsExecutionOfRequestedAction**<br>This response code 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, Con- |

| NRC (hex) | Name/Definition |
|---|---|
| | firmed or TestFailedSinceLastClear set to 1), has occurred and that this failure condition prevents the server from performing the requested action.<br>This NRC can, for example, direct the technician to read DTCs in order to identify and fix the problem.<br>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.<br>This response code 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. |
| 27 – 30 | (Reserved by ISO) |
| 31 | **requestOutOfRange**<br>This response code 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.<br>This response code shall be implemented for all services, which allow the client to read data, write data or adjust functions by data in the server. |
| 32 | (Reserved by ISO) |
| 33 | **securityAccessDenied**<br>This response code indicates that the requested action will not be taken because the server's security strategy has not been satisfied by the client.<br>The server shall send this response code if one of the following cases occur:<br>- the test conditions of the server are not met,<br>- the required message sequence e.g. DiagnosticSessionControl, securityAccess is not met,<br>- the client has sent a request message which requires an unlocked server.<br>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 isrequired and is not yet granted to perform the required service. |
| 34 | (Reserved by ISO) |
| 35 | **invalidKey**<br>This response code 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. |
| 36 | **exceedNumberOfAttempts**<br>This response code 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. |
| 37 | **requiredTimeDelayNotExpired**<br>This response code 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. |
| 38 – 4F | **reservedByExtendedDataLinkSecurityDocument** |

| NRC (hex) | Name/Definition |
|---|---|
| | This range of values is reserved by ISO 15764 Extended data link security. |
| 50 – 6F | (Reserved by ISO) |
| 70 | **uploadDownloadNotAccepted**<br>This response code indicates that an attempt to upload/download to a server's memory cannot be accomplished due to some fault conditions. |
| 71 | **transferDataSuspended**<br>This response code indicates that a data transfer operation was halted due to some fault. The active transferData sequence shall be aborted. |
| 72 | **generalProgrammingFailure**<br>This response code indicates that the server detected an error when erasing or programming a memory location in the permanent memory device (e.g. Flash Memory). |
| 73 | **wrongBlockSequenceCounter**<br>This response code 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. |
| 74 – 77 | (Reserved by ISO) |
| 78 | **requestCorrectlyReceived-ResponsePending**<br>This response code 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.<br>The negative response message with this response code may be repeated by the server until the requested service is completed and the final response message is sent. This response code might impact the application layer timing parameter values. The detailed specification shall be included in the data link specific implementation document.<br>This response code 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.<br>When this response code is used, the server shall always send a final response (positive or negative) independent of the suppressPosRspMsgIndicationBit value.<br>A typical example where this response code 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.<br>This response code 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. |
| 79 – 7D | (Reserved by ISO) |
| 7E | **subFunctionNotSupportedInActiveSession**<br>This response code indicates that the requested action will not be taken because the server does not support the requested sub-function in the session currently active.<br>This response code 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. |

| NRC (hex) | Name/Definition |
|---|---|
| 7F | **serviceNotSupportedInActiveSession**<br>This response code indicates that the requested action will not be taken because the server does not support the requested service in the session currently active. This response code shall only be used when the requested service is known to be supported in another session, otherwise response code SNS (serviceNotSupported) shall be used.<br>This response code 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. |
| 80 | (Reserved by ISO) |
| 81 | **rpmTooHigh**<br>This response code 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). |
| 82 | **rpmTooLow**<br>This response code 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). |
| 83 | **engineIsRunning**<br>This 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. |
| 84 | **engineIsNotRunning**<br>This 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. |
| 85 | **engineRunTimeTooLow**<br>This response code 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). |
| 86 | **temperatureTooHigh**<br>This response code indicates that the requested action will not be taken because the server prerequisite condition for temperature is not met (current temperature is above a preprogrammed maximum threshold). |
| 87 | **temperatureTooLow**<br>This response code indicates that the requested action will not be taken because the server prerequisite condition for temperature is not met (current temperature is below a preprogrammed minimum threshold). |
| 88 | **vehicleSpeedTooHigh**<br>This response code 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). |
| 89 | **vehicleSpeedTooLow**<br>This response code 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). |
| 8A | **throttle/PedalTooHigh**<br>This response code 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- |

| NRC (hex) | Name/Definition |
|---|---|
| | programmed maximum threshold). |
| 8B | **throttle/PedalTooLow**<br>This response code 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). |
| 8C | **transmissionRangeNotInNeutral**<br>This response code 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). |
| 8D | **transmissionRangeNotInGear**<br>This response code 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). |
| 8E | (Reserved by ISO) |
| 8F | **brakeSwitch(es)NotClosed (Brake Pedal not pressed or not applied)**<br>For saftey reasons, this is required for certain tests before it begins, and must be maintained for the entire duration of the test. |
| 90 | **shifterLeverNotInPark**<br>For saftey reasons, this is required for certain tests before it begins, and must be maintained for the entire duration of the test. |
| 91 | **torqueConverterClutchLocked**<br>This response code 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 preprogrammed limit or locked). |
| 92 | **voltageTooHigh**<br>This response code 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). |
| 93 | **voltageTooLow**<br>This response code 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). |
| 94 – FE | **reservedForSpecificConditionsNotCorrect**<br>This range of values is reserved by this document for future definition. |
| FF | (Reserved by ISO) |

# Annex B. Data Identifiers

The DataIdentifiers shall be used as follows:

| Option (hex) | Description |
|---|---|
| 0100-EFFF | **VehicleManufacturer Specific Rcord data identifiers** |
| F000-F00F | **NetworkConfigurationData** |
| F010-F0FF | **IOControl data identifiers** |
| F100 | **activeDiagnosticInformationDataIdentifier**<br>This Data Identifier provides the UDS Protocol Version.<br>diagnosticInformation [2Byte BCD] = [<br>UDS Protocol Version ] |
| F10D | **diagnosticSpecificInformationDataIdentifier**<br>This data identifier and the associated data record reflect the diagnostic specification versions according to which the ECU diagnostic software was developed.<br>diagnosticSpecificInformation [3Byte] = [<br>GAC Diagnostic Specification Version<br>ECU Reprogramming requirements – Specification version<br>Diagnostic Performance Requirements Standard ] |
| F150 | **hardwareVersionInformationDataIdentifier**<br>This Data Identifier is used to reply a manufacturer specific hardware version number.<br>hardwareVersion[3Byte] = [<br>Patch Level<br>Week<br>Year ] |
| F151 | **softwareVersionInformationDataIdentifier**<br>This Data Identifier is used to reply a manufacturer specific software version number.<br>softwareVersion[3Byte] = [<br>Patch Level<br>Week<br>Year ] |
| F154 | **hardwareSupplierIdentificationDataIdentifier**<br>The hardware supplier identification determines the manufacturer of an ECU's hard-ware. (list with hex-values and name of the suppliers)<br>hardwareSupplierIdentification[15 Byte ASCII] |
| F17F | **hardwareVersionInformationDataIdentifier**<br>This Data Identifier is used to reply a manufacturer specific hardware version number.<br>HardwareVersion [17 Byte] = [<br>Part number<br>The first letter of Hardware<br>Dot |

| Option (hex) | Description |
|---|---|
| | version ] |
| F180 | **bootSoftwareIdentification**<br>This Data Identifier is used to reply a manufacturer specific bootSoftware version number.<br>bootSoftwareVersion [17 Byte] = [<br>Part number<br>The first letter of Hardware<br>Dot<br>version ] |
| F181 | **Application SoftwareIdentification**<br>This Data Identifier is used to reply a manufacturer specific Application Software version number.<br>Application SoftwareVersion [17 Byte] = [<br>Part number<br>The first letter of Hardware<br>Dot<br>version ] |
| F182 | **applicationDataIdentificationDataIdentifier**<br>This value shall be used to reference the GAC 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 as following:<br>ApplicationDataIdentificationDataIdentifier [5 Byte ASCII] |
| F183 | **bootSoftwareFingerprintDataIdentifier** |
| F184 | **applicationSoftwareFingerprintDataIdentifier**<br>This value shall be used to reference the GAC specific ECU application software fingerprint identification record as following:<br>- Tool Suplier Identification[1 Byte Unsigned]<br>- Programming date [3 Byte BCD] = { Year, Month, Day }<br>- Tester Serial Number [6 Byte] |
| F185 | **applicationDataFingerprintDataIdentifier** |
| F186 | **activeDiagnosticSessionDataIdentifier**<br>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.<br>activeDiagnosticSessionDataIdentifier[1Byte] = [<br>0x01 = Default Session<br>0x02 = Programming Session<br>0x03 = ExtendedSession ] |
| F187 | **vehicleManufacturerSparePartNumberDataIdentifier**<br>This value shall be used to reference the spare part number of GAC. Record data content and format shall be as following:<br>VehicleManufacturerSparePartNumberDataIdentifier [14 Byte ASCII] |

| Option (hex) | Description |
|---|---|
| F188 | **vehicleManufacturerECUSoftwareNumberDataIdentifier**<br>This value shall be used to reference the ECU (server) software number of GAC. Record data content and format shall be as following:<br>VehicleManufacturerECUSoftwareNumber [3 Byte BCD]={Year, Month, Day} |
| F189 | **vehicleManufacturerECUSoftwareVersionNumberDataIdentifier**<br>This Data Identifier is used to reply a manufacturer specific software version number.<br>softwareVersion [17 Byte] = [<br>Part number<br>The first letter of Hardware<br>Dot<br>version ] |
| F18A | **systemSupplierIdentifierDataIdentifier** |
| F18B | **ECUManufacturingDateDataIdentifier**<br>This value shall be used to reference the ECU (server) manufacturing date. The according format shall be:<br>[3 Byte BCD] = { Year, Month, Day } |
| F18C | **EcuSerialNumberDataIdentifier**<br>This value shall be used to reference the ECU (server) serial number. Record data contentand format shall be server specific.<br>- Product Serial Number[3 byte]<br>- Production Line Number[2 byte]<br>- Production date [3 Byte] = { Year, Month, Day } |
| F18D | **supportedFunctionalUnitsDataIdentifier** |
| F18E | **GACKitAssemblyPartNumberDataIdentifier** |
| F190 | **VINDataIdentifier**<br>This value shall be used to reference the VIN number. Record data content and format shall be as following;<br>VINDataIdentifier [17Byte ASCII] |
| F191 | **vehicleManufacturerECUHardwareNumberDataIdentifier**<br>This value shall be used by reading services to reference GAC specific ECU (server) hardware number. Record data content and format shall be as following:<br>VehicleManufacturerECUHardwareNumberDataIdentifier[30 Byte ASCII] |
| F192 | **systemSupplierECUHardwareNumberDataIdentifier**<br>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. |
| F193 | **systemSupplier ECU Hardware Version** |
| F194 | **systemSupplierECUSoftwareNumberDataIdentifier** |
| F195 | **systemSupplierECUSoftwareVersionNumberDataIdentifier** |
| F196 | **exhaustRegulationOrTypeApprovalNumberDataIdentifier** |
| F197 | **systemNameOrEngineTypeDataIdentifier** |
| F198 | **repairShopCodeOrTesterSerialNumberDataIdentifier** |

| Option (hex) | Description |
|---|---|
| F199 | **programmingDateDataIdentifier** |
| F19A | **calibrationRepairShopCodeOrCalibrationEquipmentSerialNumberDataIdentifier** |
| F19B | **calibrationDateDataIdentifier**<br>This Data Identifier provides information about the date of the last calibration.<br>calibrationDate[6Byte] = [<br>Year [2Byte]<br>Month [2Byte]<br>Day [2Byte]] |
| F19C | **calibrationEquipmentSoftwareNumberDataIdentifier** |
| F19D | **ECUInstallationDateDataIdentifier** |
| F19E | **ODXFileDataIdentifier** |
| F1B2 | **Calibration SoftwareIdentification**<br>This Data Identifier is used to reply a manufacturer specific Calibration Software version number.<br>Calibration Software Identification [17 Byte] = [<br>14 bytes:Part number<br>1 byte:The first letter of Hardware<br>1 byte:Dot<br>1 byte:version ] |

Note:The table just list part of DID information.In case of conflicts, the information of <GAC Diagnostic parameters table> take priority over that of this table.

# Annex C. ScalingByte Parameter definitions

The parameter scalingByte (SBYT) consists of one byte (high and low nibble). The scaling-Byte 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.

High Nibble:

| Encoding of High Nibble (Hex) | Description |
|---|---|
| 0 | **unsignedNumeric (1 to 4 bytes)**<br>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 65536 steps, etc. |
| 1 | **signedNumeric (1 to 4 bytes)**<br>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 65536 steps, etc. |
| 2 | **bitMappedReportedWithOutMask**<br>Bit mapped encoding uses individual bits or small groups of bits to represent status. For every bit which represents status, a corresponding mask bit is required as part of the parameter definition. The mask indicates the validity of the bit for particular applications. This type of bit mapped parameter does not contain additional bytes to report the validity mask. |
| 3 | **bitMappedReportedWithMask**<br>Bit mapped encoding uses individual bits or small groups of bits to represent status. For every bit which represents status, a corresponding mask bit is required as part of the parameter definition. The mask indicates the validity of the bit for particular applications. This type of bit mapped parameter contains one validity mask byte for each status byte representing data. |
| 4 | **BinaryCodedDecimal**<br>Conventional Binary Coded Decimal encoding is used to represent two numeric digits per byte.<br>The upper nibble is used to represent the most significant digit (0 – 9), and the lower nibble the least significant digit (0 -9). |
| 5 | **stateEncodedVariable (1 byte)**<br>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 (1) byte. |
| 6 | **ASCII (1 to 15 bytes for each scalingByte)**<br>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. |

| Encoding of High Nibble (Hex) | Description |
|---|---|
| 7 | **signedFloatingPoint**<br>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. |
| 8 | **packet**<br>Packets contain multiple data values, usually related, each with unique scaling. Scaling information is not included for the individual values. |
| 9 | **formula**<br>A formula is used to calculate a value from the raw data. Formula Identifiers are specified in the table definig the formulaIdentifier encoding |
| A | **unit/format**<br>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.<br>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 positive response. |
| B | **stateAndConnectionType (1 byte)**<br>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. |
| C – F | (Reserved by ISO) |

Low Nibble:

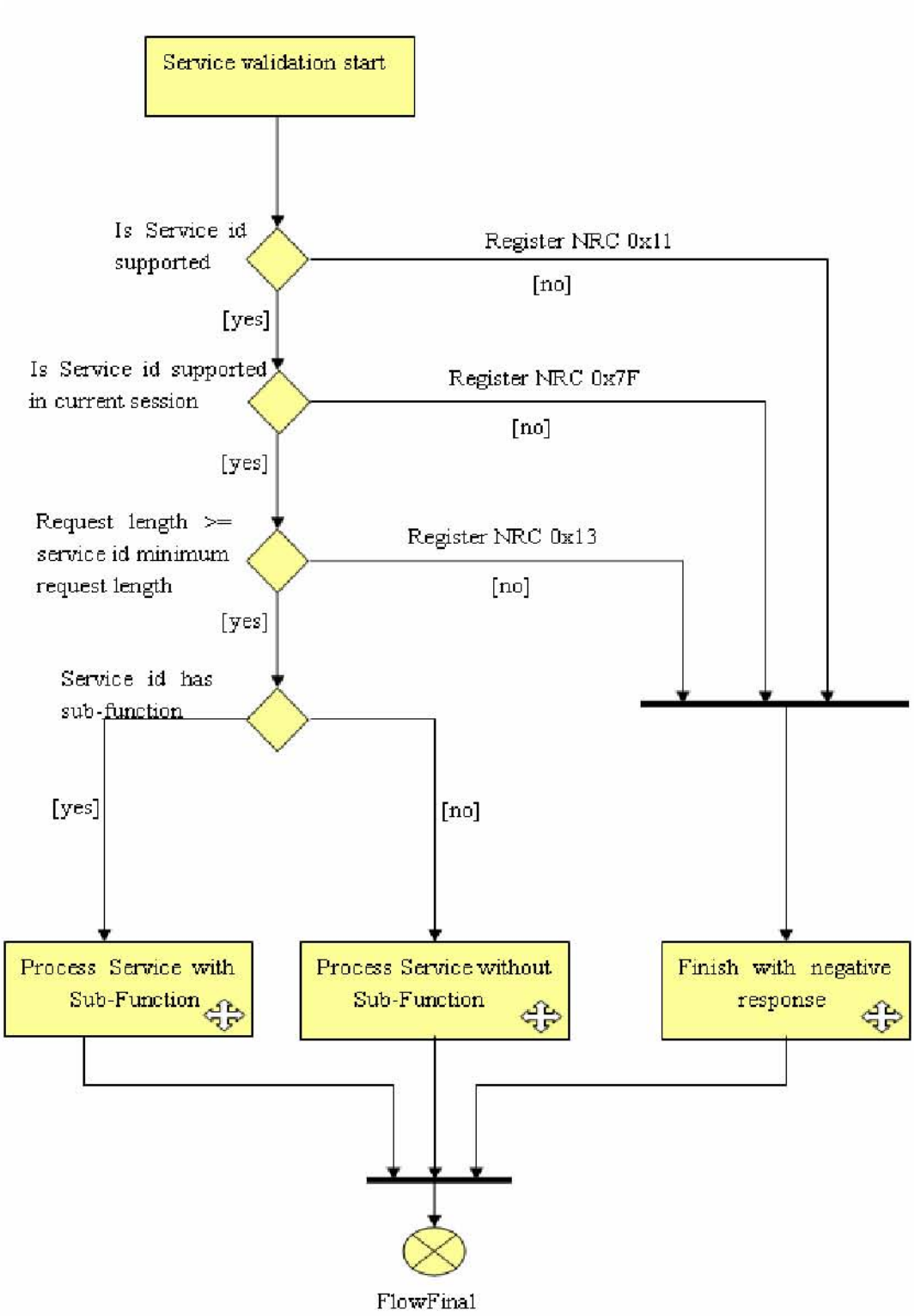
| Encoding of Low Nibble (Hex) | Description |
|---|---|
| 0 – F | **numberOfBytesOfParameter**<br>This range of values specifies the number of data bytes in a data stream referenced by a parameter identifier. The length of a parameter is defined by the scaling byte(s), which is always preceded by a 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.<br>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.<br>Note: For the scalingByte with high nibble encoded as formula or unit/format this value is $0. |

# Annex D. Routine Identifier

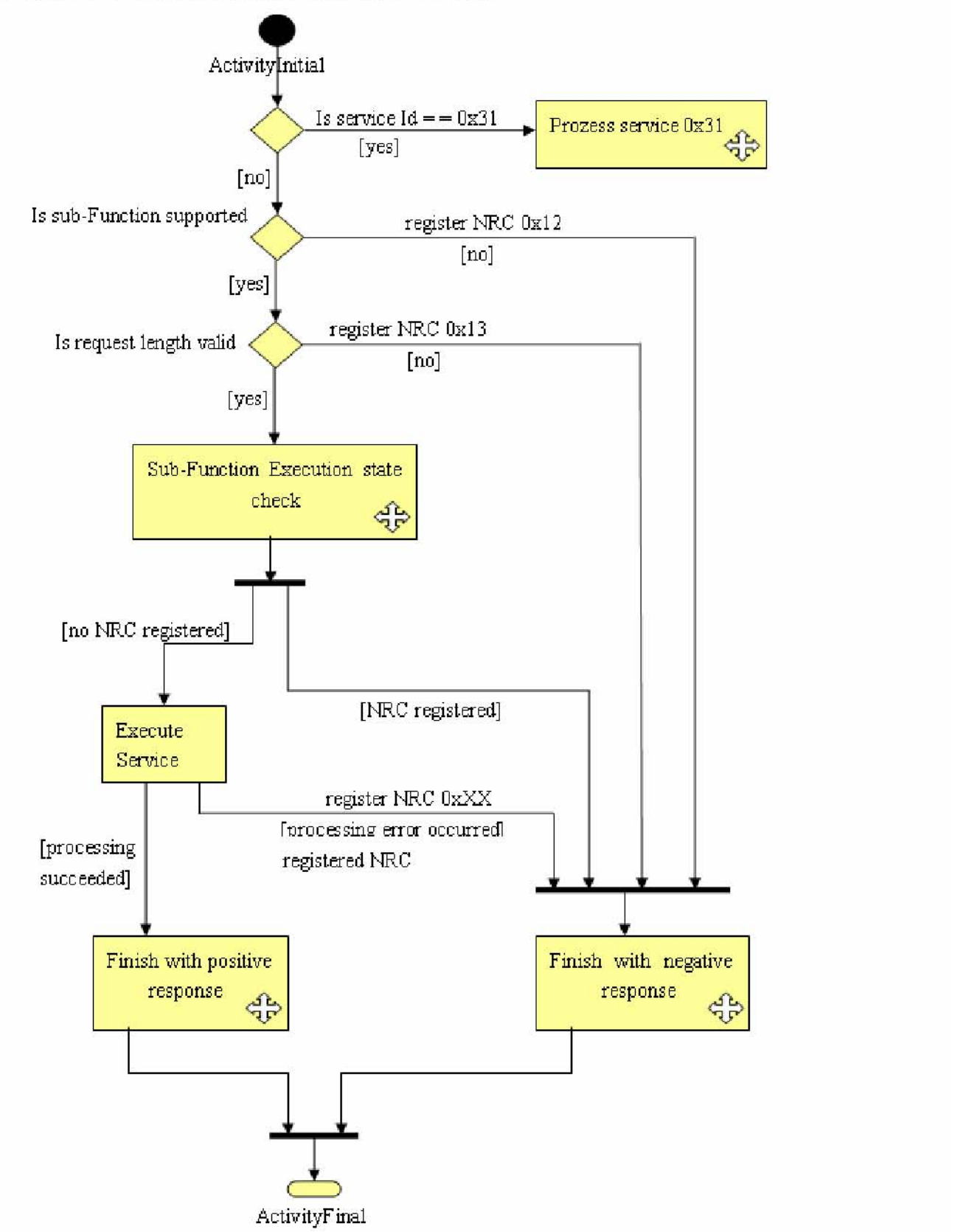
| Option (Hex) | Description |
|---|---|
| 0203 | **CheckProgramPrecondition**<br>This value shall be used to check the preconditions for reprogramming of the ECU. |
| FF02 | **EraseMirrorMemoryDTCs**<br>This value shall be used to erase the servers mirror memory DTCs. |

# Annex E. NRC flow diagrams

## Diagram: Service dispatcher



## Diagram: Process Service with Sub-function



## Diagram: Process Service 0x31

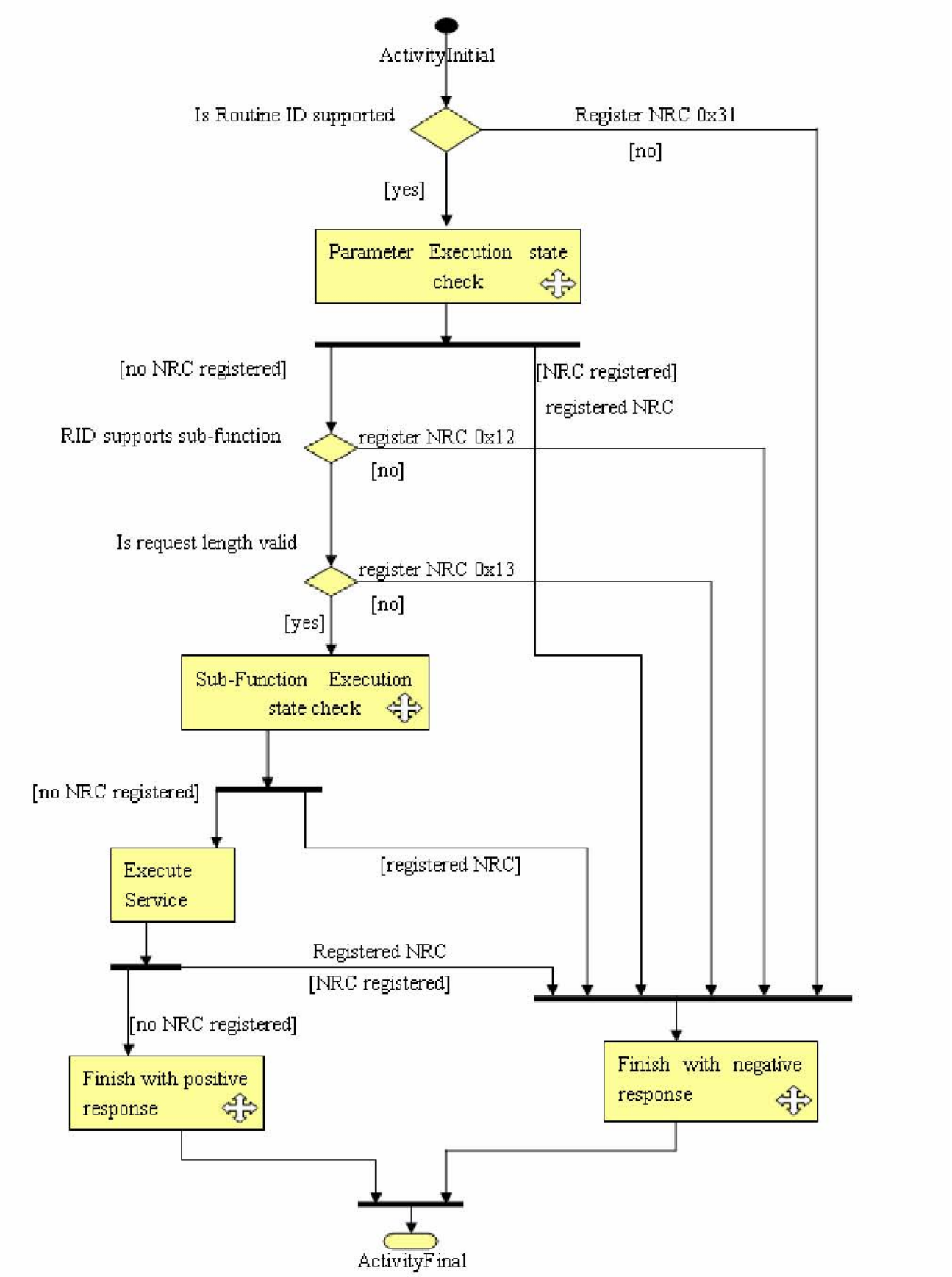
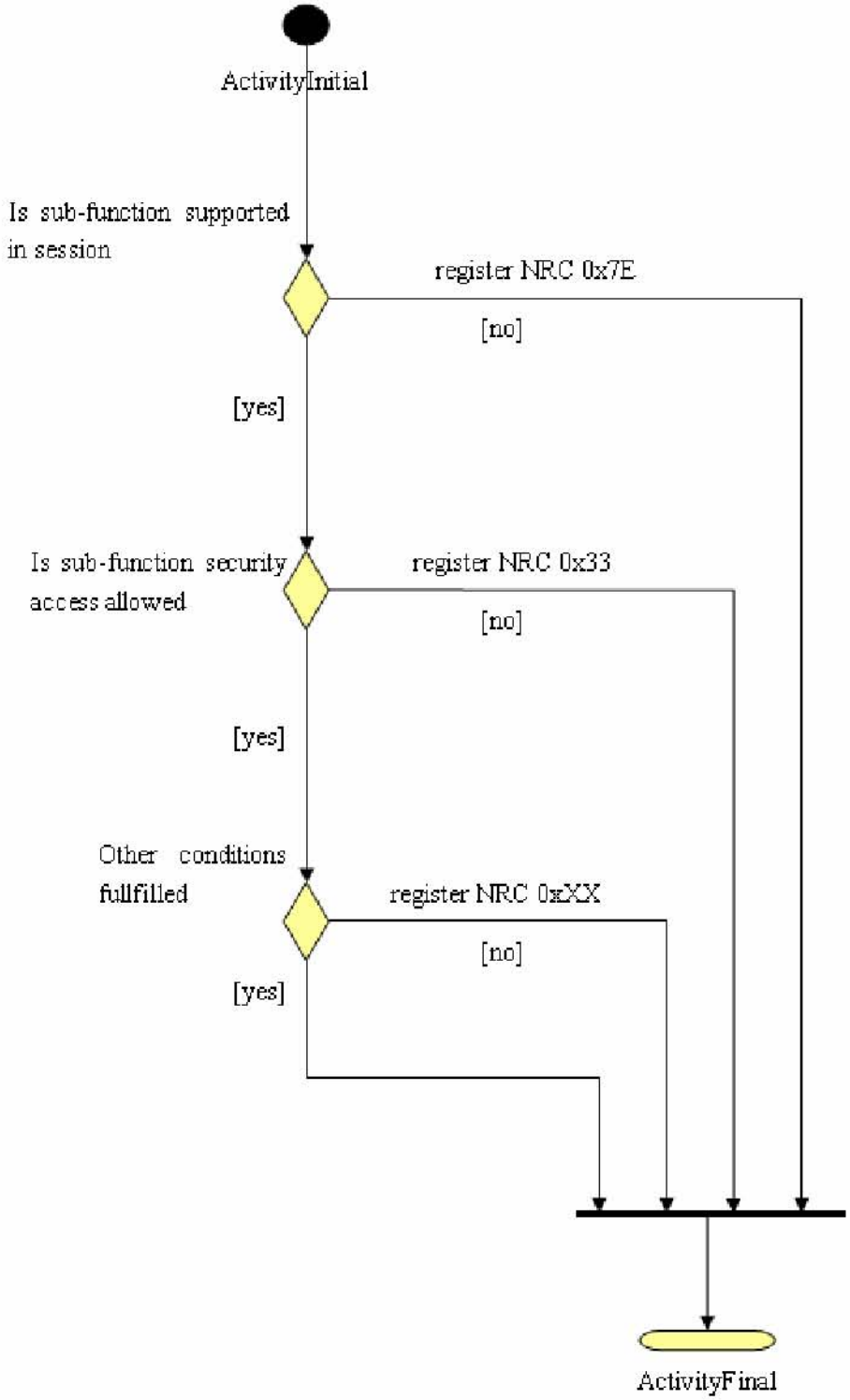
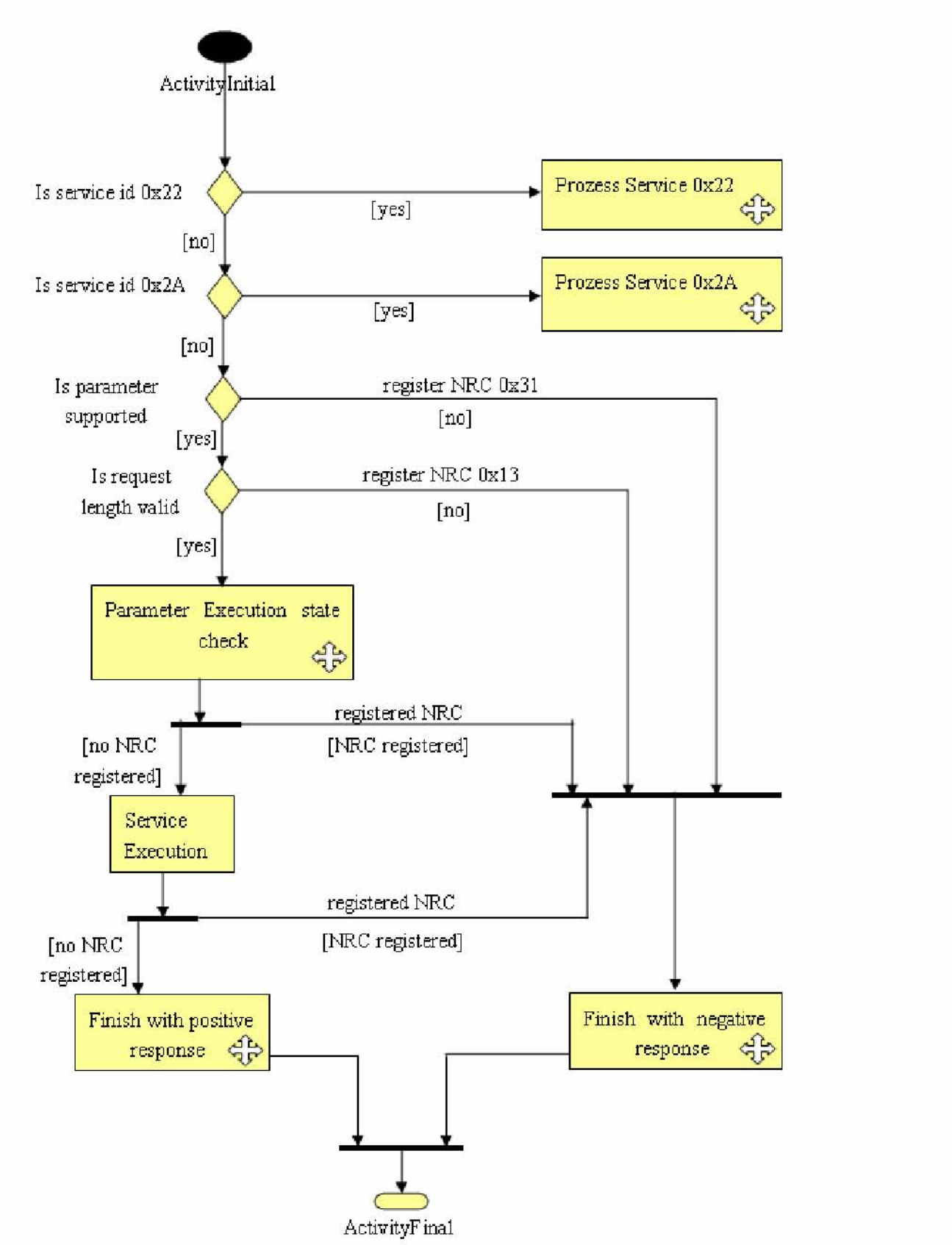


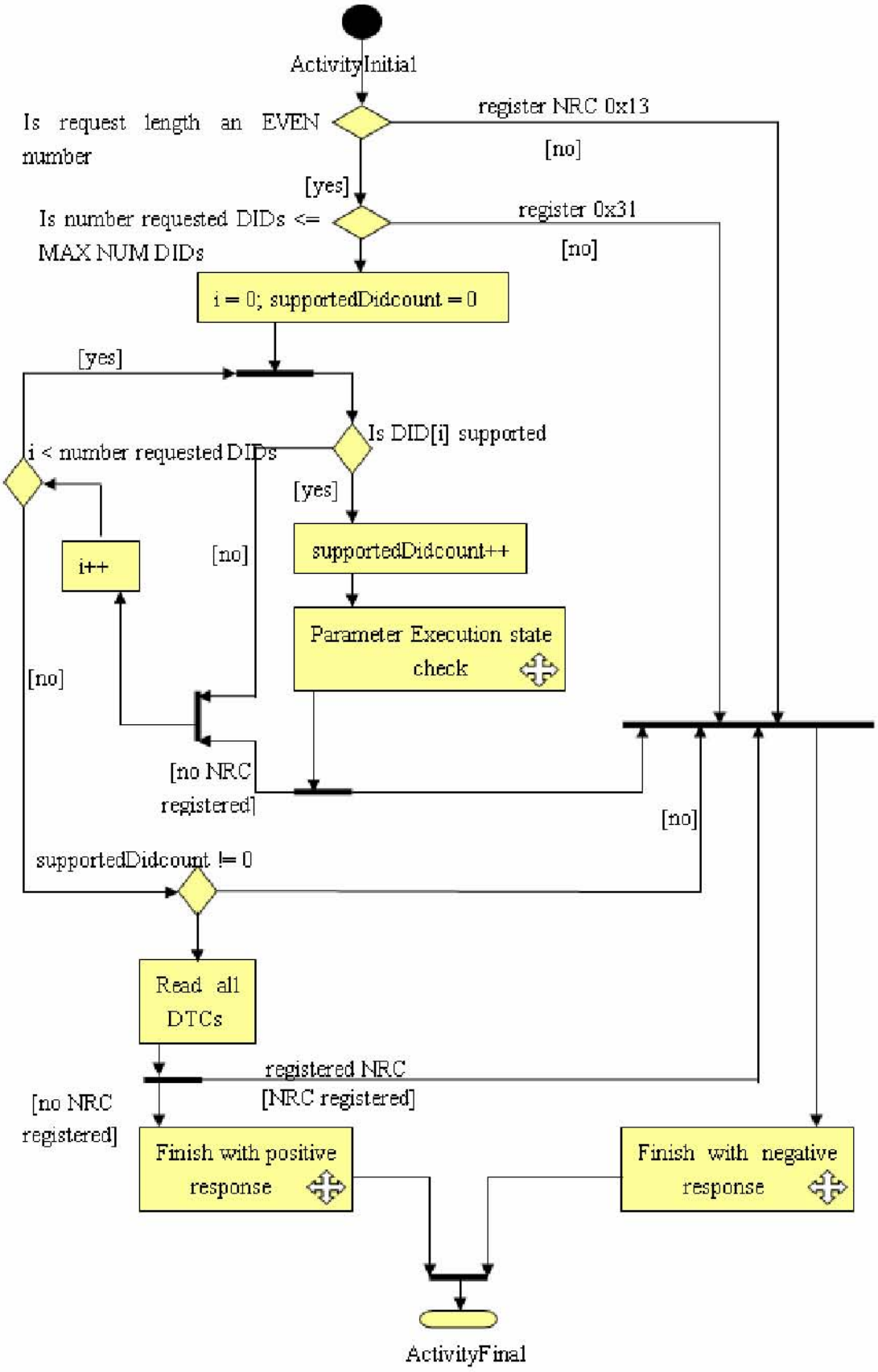
**Diagram: Check sub-function execution state**



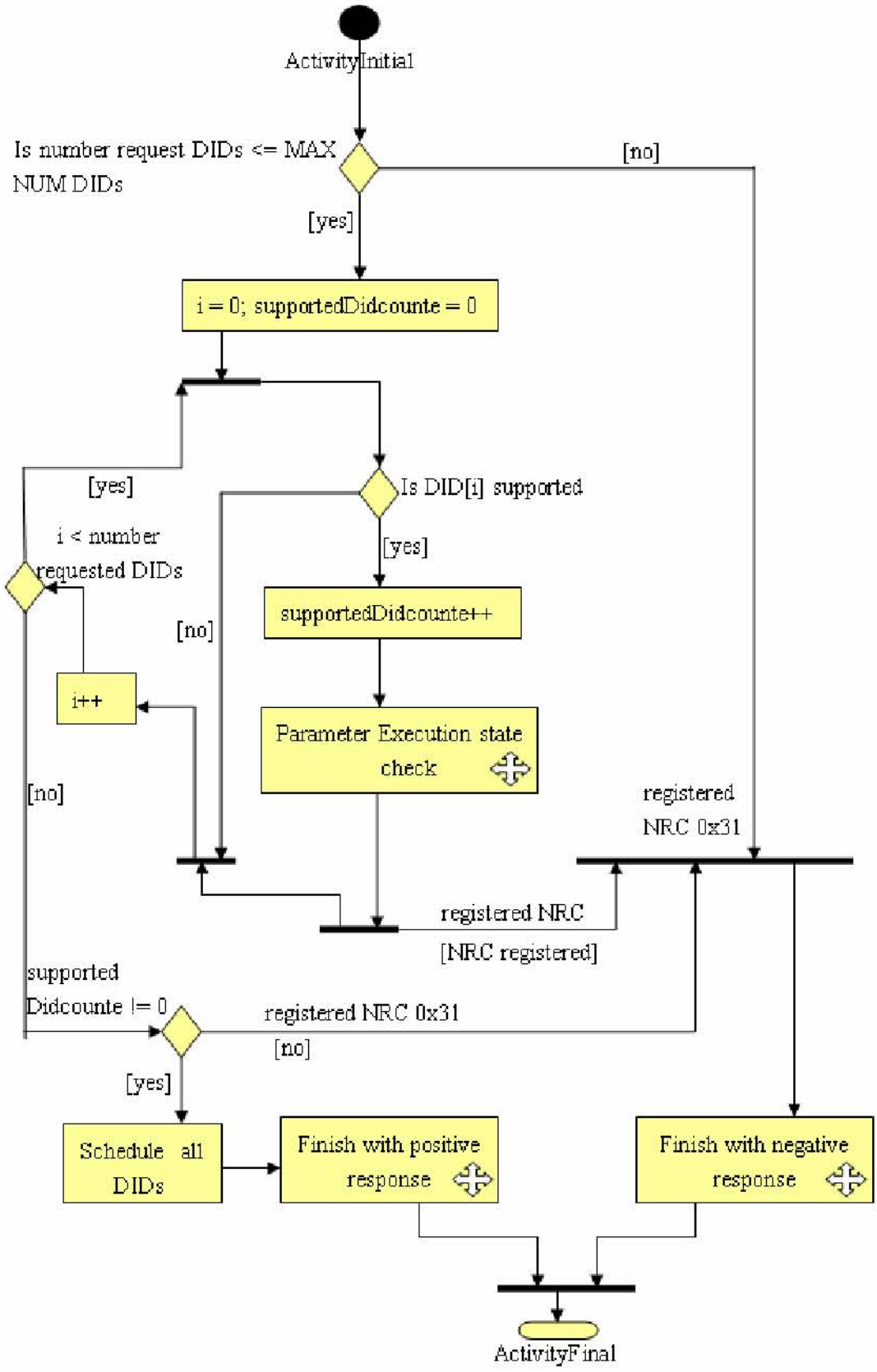
## Diagram: Process Service without Sub-function



## Diagram: Process Service 0x22



## Diagram: Process Service 0x2A



## Diagram: Parameter Execution State Check

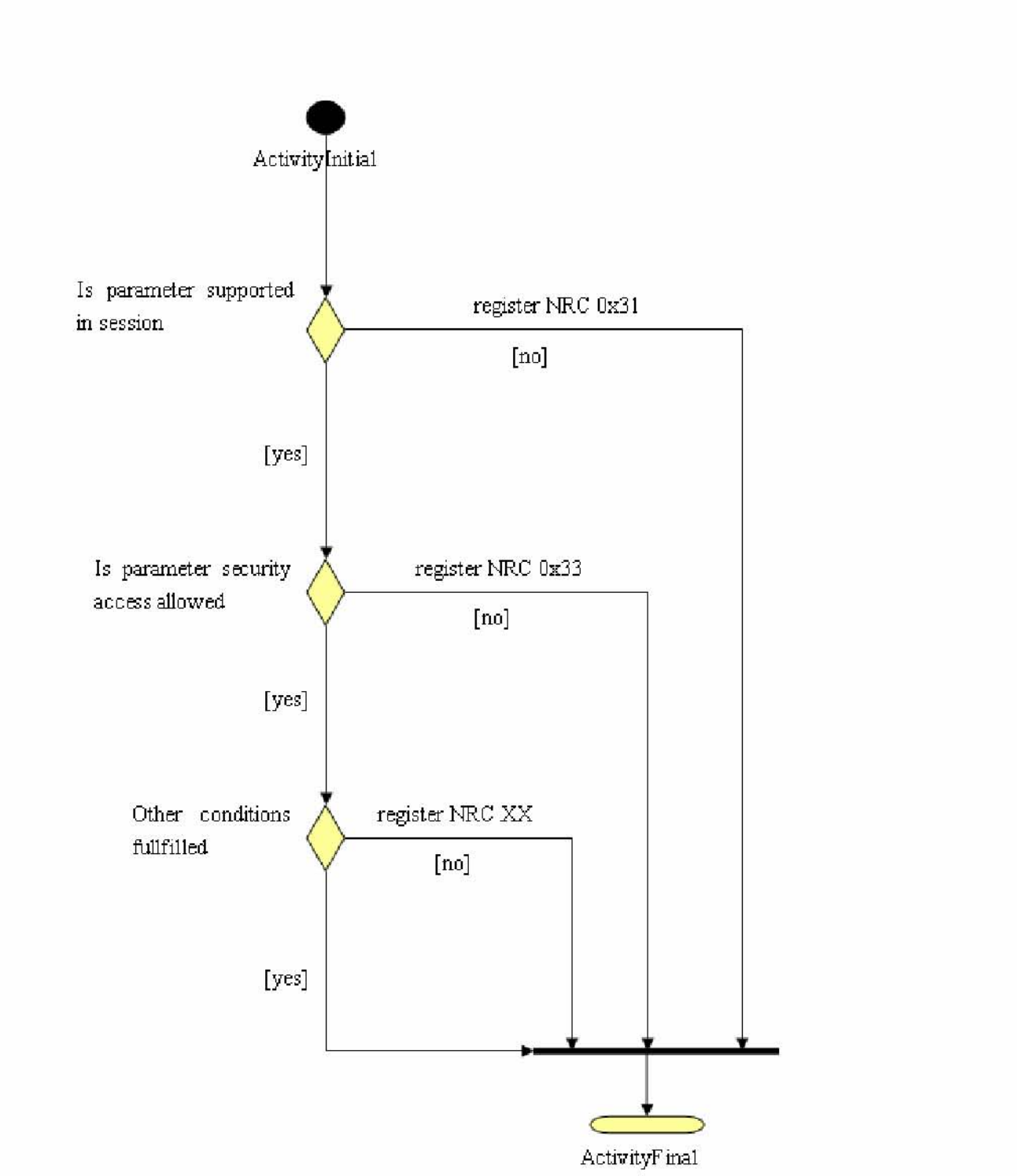


**Diagram: Finish with negative response**

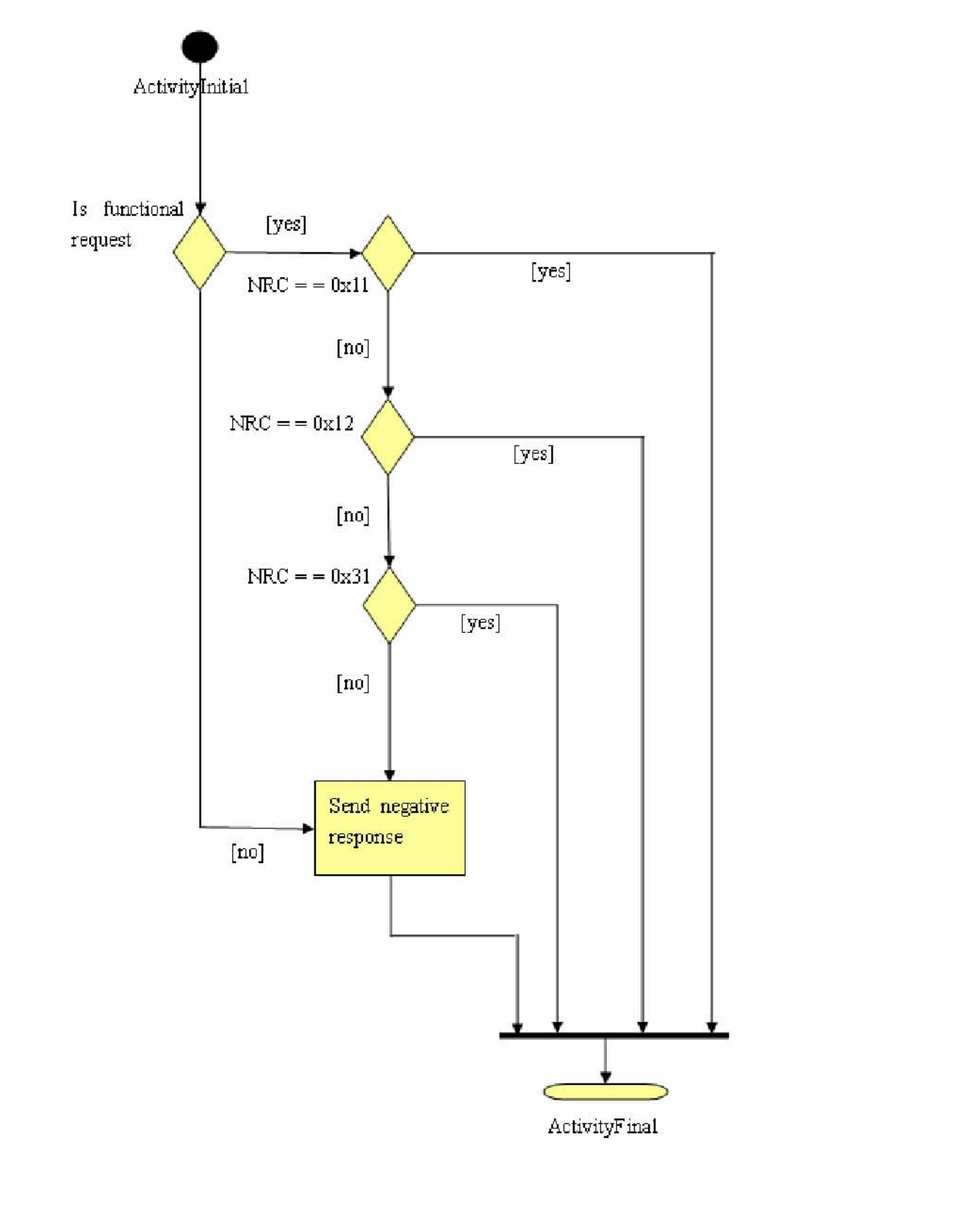


**Diagram: Finish with positive response**