SSF 14230 Road Vehicles - Diagnostic Systems

Keyword Protocol 2000 - Part 3 - Application Layer

Swedish Implementation Standard

Document: SSF 14230-3

Status: Issue 2

Date: February 1, 2000

**This document is based on the International Standard ISO 14230 Keyword Protocol 2000 and has been further developed to meet Swedish automotive manufacturer's requirements by the Swedish Vehicle Diagnostics Task Force.
It is based on mutual agreement between the following companies:**

- **Saab Automobile AB**
- **SCANIA AB**
- **Volvo Car Corp.**
- **Volvo Bus Corp.**
- **Mecel AB**

Document updates and issue history

This document can be revised and appear in several versions. The document will be classified in order to allow identification of updates and versions.

A. Document status classification

The document is assigned the status *Outline*, *Draft* or *Issue*.
It will have the *Outline* status during the initial phase when parts of the document are not yet written.
The *Draft* status is entered when a complete document is ready, which can be submitted for reviews. The draft is not approved. The draft status can appear between issues, and will in that case be indicated together with the new issue number E.g. *Draft Issue 2*.
An *Issue* is established when the document is reviewed, corrected and approved.

B. Version number and history procedure

Each issue is given a number and a date. A history record shall be kept over all issues.
Document in Outline and Draft status may also have a history record.

C. History

Issue #	Date	Comment
1	May 12, 1998	
2	February 1, 2000	Mayor revision to make SSF 14230-3 compatible with ISO 16844-6. StopDiagnosticSession service replaced by StartDiagnosticSession, parameter diagnosticSession = 81 (standardSession).

Table of Content

Introduction

This document is based on the International Standard *ISO 14230-3 Road vehicles - Diagnostic systems - Keyword Protocol 2000 - Part 3: Application layer*.

The *SSF 14230-3 Keyword Protocol 2000 - Part 3 - Application Layer, Swedish Implementation Standard* has been established in order to define common requirements for the implementation of diagnostic services for diagnostic systems. The figure below shows the "Hierarchy of Documents" and indicates the level of commonality between the documents.

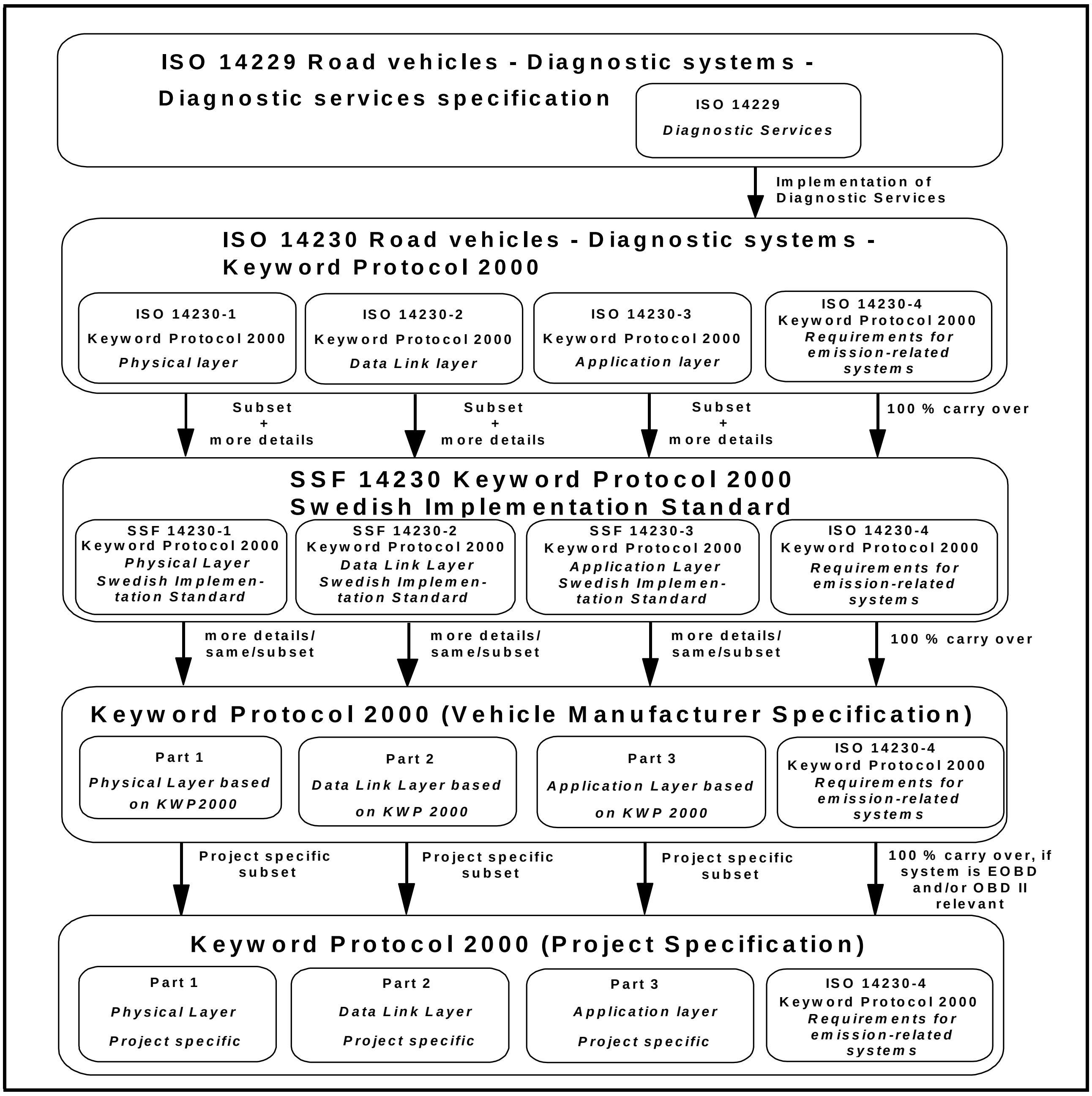

ISO 14229 Road vehicles - Diagnostic systems - Diagnostic services specification
ISO 14229 Diagnostic Services
Implementation of Diagnostic Services
ISO 14230 Road vehicles - Diagnostic systems - Keyword Protocol 2000
ISO 14230-1 Keyword Protocol 2000 Physical layer
ISO 14230-2 Keyword Protocol 2000 Data Link layer
ISO 14230-3 Keyword Protocol 2000 Application layer
ISO 14230-4 Keyword Protocol 2000 Requirements for emission-related systems
Subset + more details
Subset + more details
Subset + more details
100 % carry over
SSF 14230 Keyword Protocol 2000 Swedish Implementation Standard
SSF 14230-1 Keyword Protocol 2000 Physical Layer Swedish Implementation Standard
SSF 14230-2 Keyword Protocol 2000 Data Link Layer Swedish Implementation Standard
SSF 14230-3 Keyword Protocol 2000 Application Layer Swedish Implementation Standard
ISO 14230-4 Keyword Protocol 2000 Requirements for emission-related systems
more details/same/subset
more details/same/subset
more details/same/subset
100 % carry over
Keyword Protocol 2000 (Vehicle Manufacturer Specification)
Part 1 Physical Layer based on KWP2000
Part 2 Data Link Layer based on KWP 2000
Part 3 Application Layer based on KWP 2000
ISO 14230-4 Keyword Protocol 2000 Requirements for emission-related systems
Project specific subset
Project specific subset
Project specific subset
100 % carry over, if system is EOBD and/or OBD II relevant
Keyword Protocol 2000 (Project Specification)
Part 1 Physical Layer Project specific
Part 2 Data Link Layer Project specific
Part 3 Application layer Project specific
ISO 14230-4 Keyword Protocol 2000 Requirements for emission-related systems

Figure 0.1 - Hierarchy of Documents

- The ISO 14229 document specifies common requirements for the implementation of diagnostic services.
- The ISO 14230 documents specify the requirements of the "Physical Layer, Data Link Layer and Application Layer" in three (3) different documents.

- The SSF 14230 documents are based on the International Standard ISO 14230 and therefore fully compatible. The Keyword Protocol 2000 - Physical Layer, Data Link Layer and Application Layer Swedish Implementation Standard documents are a subset with additional detail based on mutual agreement between all companies listed on the cover sheet of the documents.
- The Keyword Protocol 2000 Vehicle Manufacturer Specification documents are based on SSF 14230 documents. The Part 1 Physical Layer, Part 2 Data Link Layer, and the Part 3 Application Layer documents shall specify *more details* or *the same* or *a subset* of the SSF 14230 documents.
- The Keyword Protocol 2000 Project Specification document (e.g. Engine Management) is based on the vehicle manufacturer specification document. The documents of the system supplier specifies system and customer specific Part 1 Physical Layer, Part 2 Data Link Layer and Part 3 Application Layer details (e.g. datastreams, diagnostic trouble codes, input/output controls, etc.).

The SSF 14230 Swedish Implementation Standard is based on the Open Systems Interconnection (O.S.I.) Basic Reference Model in accordance with ISO/IEC 7498 and ISO/IEC 10731 which structures communication systems into seven layers. When mapped on this model, the services used by a diagnostic tester and an Electronic Control Unit (ECU) are broken into:

- Diagnostic services (layer 7),
- Communication services (layers 1 to 6)

Mapping of the Diagnostic Services and Keyword Protocol 2000 onto the OSI model is shown below. SSF 14230 consists of the following parts, under the general title *Road Vehicles - Diagnostic Systems - Keyword Protocol 2000***:**

- Part 1: Physical Layer
- Part 2: Data Link Layer
- Part 3: Application Layer

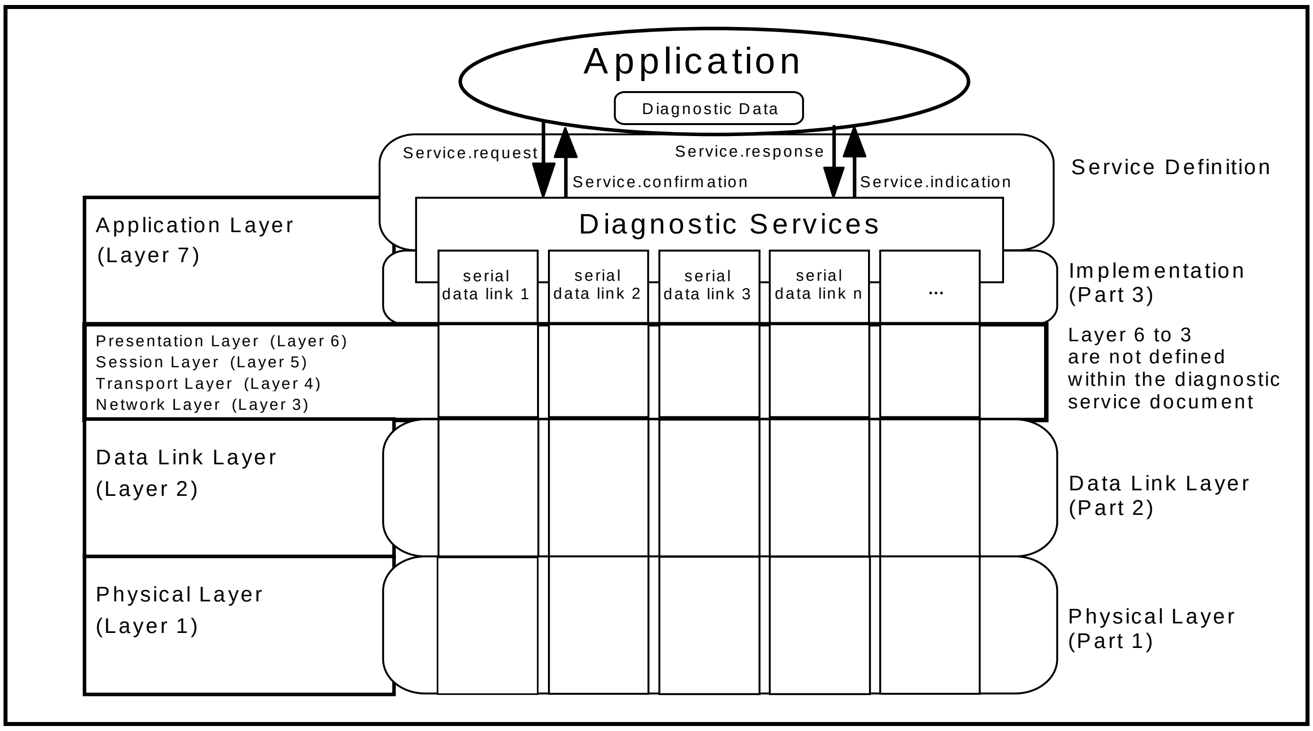


Figure 0.2 - Mapping of the Diagnostic Services and Keyword Protocol 2000 on the OSI Model

1 Scope

SSF 14230 specifies the requirements for the Keyword Protocol 2000 data link by which one or several on-vehicle Electronic Control Units are connected to an off-board tester in order to perform diagnostic functions.

SSF 14230 specifies additional detail to some sections of ISO 14230.

It consists of three parts:

- Part 1: Physical Layer
- Part 2: Data Link Layer
- Part 3: Application Layer

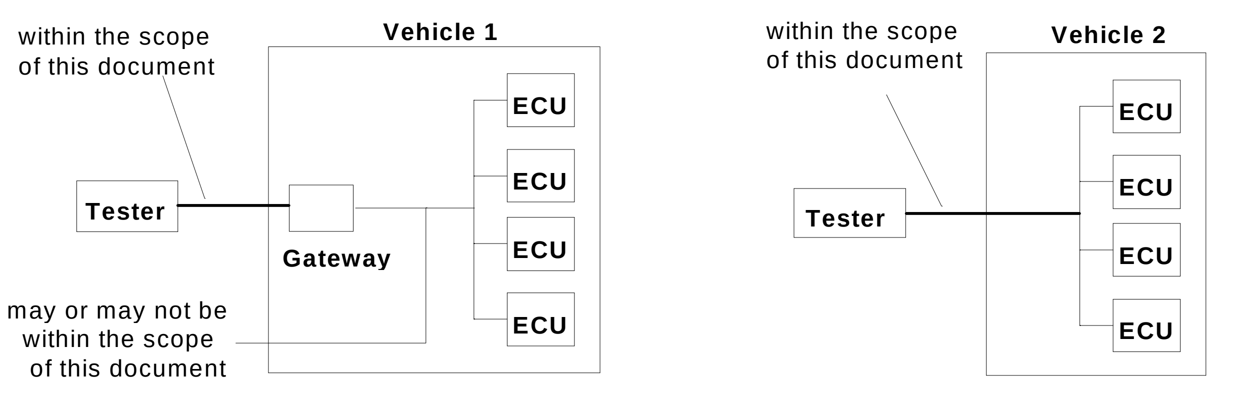
This part of SSF 14230 specifies requirements on the implementation of the Diagnostic Services specified in ISO 14229, including:

- byte encoding and hexadecimal values for the service identifiers,
- byte encoding for the parameters of the diagnostic service request and respons messages,
- hexadecimal values for the standard parameters.

The vehicle environment to which this standard applies may consist of:

- a single tester which may be temporarily connected to the on-vehicle diagnostic data link and
- several on-vehicle Electronic Control Units connected directly or indirectly.

See figure 1.1 below.



In vehicle 1, the ECUs are connected by an internal data link and indirectly connected to the diagnostic data link through a gateway. This document applies to the diagnostic communications over the diagnostic data link; the diagnostic communications over the internal data link may conform to this document or to another protocol.

In vehicle 2, the ECUs are directly connected to the diagnostic data link.

Figure 1.1 - Vehicle diagnostic architectures

2 Normative reference

2.1 ISO standards

The following standards contain provisions which, through reference in this text, constitute provisions of this document. All standards are subject to revision, and parties to agreement based on this document are encouraged to investigate the possibility of applying the most recent editions of the standards listed below. Members of ISO maintain registers of currently valid International Standards.

ISO/IEC 7498	Information technology - Open Systems Interconnection - Basic Reference Model
ISO/IEC 10731	Information technology - Open Systems Interconnection - Basic Reference Model - Conventions for the definition of OSI services
ISO 14229	Road vehicles - Diagnostic systems - Diagnostic services specification
ISO 14230-1	Road vehicles - Diagnostic systems - Keyword Protocol 2000 - Part 1: Physical layer
ISO 14230-2	Road vehicles - Diagnostic systems - Keyword Protocol 2000 - Part 2: Data link layer
ISO 14230-3	Road vehicles - Diagnostic systems - Keyword Protocol 2000 - Part 3: Application layer
ISO 14230-4	Road Vehicles - Diagnostic systems - Keyword Protocol 2000 - Part 4: Requirements for emission-related systems

2.2 Other standards

SSF 14230-1	Road Vehicles - Diagnostic Systems, Keyword Protocol 2000 - Part 1 - Physical Layer, Swedish Implementation Standard
SSF 14230-2	Road Vehicles - Diagnostic Systems, Keyword Protocol 2000 -Part 2 - Data Link Layer, Swedish Implementation Standard
ANSI/IEEE Std 754-1985	
SAE J1930	Electrical/Electronic Systems Diagnostic Terms, Definitions, Abbreviations & Acronyms
SAE J1979	E/E Diagnostic Test Modes

3 Definitions and abbreviations

3.1 Terms defined in other standards

3.1.1 ISO definitions

This document makes use of terms defined in ISO 14229.

3.1.2 SAE definitions

This document makes use of terms defined in SAE J1930.

3.2 Terms defined by this document

3.2.1 Service Identifier value convention table

The following chart indicates the different ranges of service identifier values, which are defined in SSF 14230, SAE J1979, by vehicle manufacturers or by system suppliers.

Service Identifier Hex Value	Service type (bit 6)	Where defined
00 - 0F	Request	SAE J1979
10 - 1F	Request (bit 6 = 0)	SSF 14230-3
20 - 2F		
30 - 3E		
3F	Not Applicable	Reserved by document
40 - 4F	Response	SAE J1979
50 - 5F	Positive Response to Services ($10 - $3E) (bit 6 = 1)	SSF 14230-3
60 - 6F		
70 - 7E		
7F	Negative Response	SSF 14230-3
80	Request 'ESC' - Code	ISO 14230-3
81 - 8F	Request (bit 6 = 0)	SSF 14230-2
90 - 9F	Request (bit 6 = 0)	Reserved for future exp. as needed
A0 - B9	Request (bit 6 = 0)	Defined by vehicle manufacturer
BA - BF	Request (bit 6 = 0)	Defined by system supplier
C0	Positive Resp. 'ESC' - Code	ISO 14230-3
C1 - CF	Positive Response (bit 6 = 1)	SSF 14230-2
D0 - DF	Positive Response (bit 6 = 1)	Reserved for future exp. as needed
E0 - F9	Positive Response (bit 6 = 1)	Refined by vehicle manufacturer
FA - FF	Positive Response (bit 6 = 1)	Refined by system supplier

Table 3.2.1 - Service Identifier value convention table

Service identifier values "$00 - $0F" and "$40 - $4F" are reserved to be defined in SAE J1979 which currently only includes functionally addressed services.
There is a one-to-one correspondence between request messages and positive response messages, with "bit 6" of the service identifier hex value indicating the service type.

3.2.2 Definitions used in this document

- The **"default timing"** shall always reference the timing parameter values as initiated by the keybytes sent by the server (ECU) at the start of the communication. The timing parameter values and possible keybytes are specified in SSF 14230-2.

- The term **“system supplier”** refers to the supplier with the product responsibility for the system or component. It is not necessarily the same as the system/component manufacturer.

4 Conventions

This document is guided by the OSI Service Conventions discussed in ISO/IEC 10731 as they apply to the diagnostic services. These conventions define the interactions between the service user and the service provider. Information is passed between the service user and the service provider by service primitives, which may convey parameters.

4.1 Service description convention

This section defines the layout used to describe the diagnostic services. It includes:

- Parameter Definition
- Message Data Bytes
- Message Description
- Message Flow Example
- Implementation Example

4.1.1 Parameter definition

This section defines the use and the values of parameters used by the service.

4.1.2 Message data bytes

The definition of each message includes a table which lists the parameters of its primitives: request/indication ("Req/Ind"), response/confirmation ("Rsp/Cnf") for positive or negative result. All have the same structure. The first table (refer to section 4.1.2.1) describes the request message, the second table (refer to section 4.1.2.2) the positive response message and the third table (refer to section 4.1.2.3) the negative response message. Thus, only a positive or a negative response message may be used; both are listed in separate tables because the list of parameters differ between positive and negative response messages.

The header bytes and the checksum content of each message table is defined in SSF 14230-2.

Table content description:

- Under the <**Service Name> Request Message** are listed the parameters specific to the service request/indication.
- Under the <**Service Name> Positive Response Message** are listed the parameters specific to the service response/confirmation in case the requested service was successful.
- Under the <**Service Name> Negative Response Message** are listed the parameters specific to the service response/confirmation in case the requested service has failed or could not be completed in time.

For a given primitive, the presence of each parameter (or parameter value) is described by one of the following convention (Cvt) values:

- **M:** mandatory;
- **U:** user option; the parameter may or may not be supplied, depending on dynamic usage by the user;
- **C:** conditional; the presence of the parameter depends upon other parameters within the service.

4.1.2.1 Request message

The following table shows a general service table structure and its syntax.

Type	Parameter Name	Cvt	Hex Value	Mnemonic
Header Bytes	**Format Byte** **Target Byte** **Source Byte** **Length Byte**	**M** **M** **M)** **C1)**	**xx** **xx** **xx** **xx**	**FMT** **TGT** **SRC** **LEN**
<ServiceId>	<Service Name> Request Service Identifier	M	xx	SN
<Parameter Type> : <Parameter Type>	<List of parameters> = [<Parameter Name> : <Parameter Name>]	C2)	xx=[xx : xx]	PN
CS	**Checksum Byte**	**M**	**xx**	**CS**

Table 4.1.2.1 - Request message

C1) Condition 1: The header byte "Length" depends on the content of the "Format Byte" which is specified in SSF 14230-2.

C2) Condition 2: These parameters may be either mandatory (M) or user optional (U), depending on the individual message.

4.1.2.2 Positive response message

A positive response message shall be sent by the server if it is able to completely perform the requested actions.

Type	Parameter Name	Cvt	Hex Value	Mnemonic
Header Bytes	**Format Byte** **Target Byte** **Source Byte** **Length Byte**	**M** **M** **M** **C1)**	**xx** **xx** **xx** **xx**	**FMT** **TGT** **SRC** **LEN**
<ServiceId>	<Service Name> Positive Response Service Identifier	M	xx	SNPR
<Parameter Type> : <Parameter Type>	<List of parameters> = [<Parameter Name> : <Parameter Name>]	C2)	xx=[xx : xx]	PN
CS	**Checksum Byte**	**M**	**xx**	**CS**

Table 4.1.2.2 - Positive response message

Conditions: see 4.1.2.1

4.1.2.3 Negative response message

A negative response message shall be sent by the server if it is not able to completely perform the requested actions.

Type	Parameter Name	Cvt	Hex Value	Mnemonic
Header Bytes	**Format Byte** **Target Byte** **Source Byte** **Length Byte**	**M** **M** **M** **C1)**	**xx** **xx** **xx** **xx**	**FMT** **TGT** **SRC** **LEN**
<ServiceId>	negativeResponse Service Identifier	M	7F	NR
<ServiceId>	<Service Name> Request Service Identifier	M	xx	SN
<Parameter Type>	responseCode= [{ section 4.4 }]	M	xx	RC_...
CS	**Checksum Byte**	**M**	**xx**	**CS**

Table 4.1.2.3 - Negative response message

Conditions: see 4.1.2.1

4.1.3 Message description

This section provides a description of the actions performed by the client and the server which are specific to the KWP 2000 data link.
The response condition is service specific and defined separately for each service.

4.1.4 Message flow examples

This section provides message flow descriptions presented in a table format. Time relates to the table in a ***top to bottom*** sequence. The table consists of three columns:

- **column 1:** includes the relevant inter-message timing which is specified in SSF 14230-2. The message shall be started within the relevant inter-message timing.
- **column 2:** includes all requests sent by the client to the server
- **column 3:** includes all responses sent by the server to the client

For simplification all messages are described without any identifiers and/or data values. Details of messages are always specified in the section: **Message data bytes.**

time	client (Tester)	server (ECU)
P3	<Service Name> Request[...]	
P2		<Service Name> PositiveResponse[...]

Table 4.1.4 - Message flow example of physical addressed service

Note: Above message flow example is not documented for each service. Only services, which call for more detailed message flow description shall have their own message flow section.

4.1.5 Implementation example

4.1.5.1 Message flow Conditions

This section specifies the conditions to perform the service shown in the section 4.1.5.2 - Message flow.

4.1.5.2 Message flow

This section specifies the message flow of an example which is related to the service specified in above main section.

STEP#1 service name(...)

time	Client (tester) Request Message	Hex
P3	**service name.ReqSId**[data byte#2 **:** data byte#m]	**xx**

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**service name.PosRspSId**[	**xx**	**negativeResponse Service Identifier**	**7F**
	recordValue#1	xx	**service name.ReqSId**[	**xx**
	:	**:**	responseCode { refer to section 4.4 }]	xx
	recordValue#m]	xx		
	return(main)		**return(responseCode)**	

4.2 Functional unit table

The intention of specifying functional unit tables is to group similar Keyword Protocol 2000 services into a **functional** unit. The definition of each functional unit includes a table which lists its services.

Functional Unit	Description
Diagnostic Management	This functional unit includes Keyword Protocol 2000 services which are used to realise diagnostic management functions between the client (tester) and the server (ECU).
Data Transmission	This functional unit includes Keyword Protocol 2000 services which are used to realise data transmission functions between the client (tester) and the server (ECU).
Stored Data Transmission	This functional unit includes Keyword Protocol 2000 services which are used to realise stored data transmission functions between the client (tester) and the server (ECU).
Input / Output Control	This functional unit includes Keyword Protocol 2000 services which are used to realise input / output control functions between the client (tester) and the server (ECU).
Remote Activation of Routine	This functional unit includes Keyword Protocol 2000 services which are used to realise remote activation of routine functions between the client (tester) and the server (ECU).
Upload / Download	This functional unit includes Keyword Protocol 2000 services which are used to realise upload / download functions between the client (tester) and the server (ECU).

Table 4.2 - Keyword Protocol 2000 functional units

4.3 Service Identifier value summary table

The purpose of the **"Service Identifier value summary table"** is to provide an overview of all services specified/referenced in this document. The table is designed to assist system designers at a very early stage in the development of a new system with self diagnostic capabilities and serial interface to select the appropriate diagnostic services for the system to be developed without studying the entire document.

The table indicates which diagnostic session enables which set of diagnostic services.

- The 1st column lists all implemented services from ISO 14229.
- The 2nd column includes the section number in this document where the service is further defined.
- The 3rd column assigns the **Service Identifier values** for request messages.
- The 4th column specifies the services of the **"standardSession (SS)** which may be implemented in each server (ECU) if the electronic system supports the functionality of these services.
- The 5th column specifies the services of the **"adjustmentSession" (AS)** which may be implemented to allow for adjustment of input/output signals of the server (ECU).
- The 6th column specifies the services of the **"programmingSession" (PS)** which may be implemented to allow for programming of memory (e.g. flash), variant coding, parameters, etc. in the server (ECU).
- The 7th column specifies the services of the **"developmentSession" (DS)** which may be implemented during the development of the server (ECU).
- The 8th column specifies the services of the **"vehicleManufacturerSpecificSession" (VMSS)** which is to be defined by the vehicle manufacturer
- The 9th column specifies the services of the **"systemSupplierSpecificSession" (SSSS)** which is to be defined by the system supplier.

Diagnostic Services			**Diagnostic Sessions**					
Diagnostic Service Name	**Section No.**	**SId Hex Value**	**SS**	**AS**	**PS**	**DS**	**VMSS**	**SSSS**
startCommunication	**1)**	**81**	■	■	■	■	■	■
stopCommunication	**1)**	**82**	■					
accessTimingParameters	**1)**	**83**	□	□	□	□		
SAE J1979 Diag. Test Modes	**2)**	**00-0F**	□ ◆	□		□		
testerPresent	**6.4**	**3E**	■	■	■	■	■	■
startDiagnosticSession	**6.1**	**10**	□	■	■	■	■	■
stopDiagnosticSession	6.2	20						
securityAccess	**6.3**	**27**	□	□	□	□		
ecuReset	**6.5**	**11**	□	□	□	□		
readEcuIdentification	**6.6**	**1A**	■	■	■	■		
readDataByLocalIdentifier	**7.1**	**21**	□	□	□	□		
readDataByCommonIdentifier	**7.2**	**22**	□	□	□	□		
readMemoryByAddress	**7.3**	**23**			□	□		
dynamicallyDefineLocalIdentifier	**7.4**	**2C**	□	□		□		
writeDataByLocalIdentifier	**7.5**	**3B**		□	□	□		
writeDataByCommonIdentifier	**7.6**	**2E**		□	□	□		
writeMemoryByAddress	**7.7**	**3D**			□	□		
setDataRates	7.8	26						
stopRepeatedDataTransmission	7.9	25						
readDiagnosticTroubleCodes	8.1	13						
readDiagnosticTroubleCodesByStatus	**8.2**	**18**	■	□		□		
readStatusOfDiagnosticTroubleCodes	**8.3**	**17**	□	□		□		
readFreezeFrameData	**8.4**	**12**	□	□		□		
clearDiagnosticInformation	**8.5**	**14**	■	□		□		
inputOutputControlByLocalIdentifier	**9.1**	**30**	❍	□		□		
inputOutputControlByCommonIdentifier	**9.2**	**2F**	❍	□		□		
startRoutineByLocalIdentifier	**10.1**	**31**		□	□	□		
startRoutineByAddress	**10.2**	**38**			□	□		
stopRoutineByLocalIdentifier	**10.3**	**32**		□	□	□		
stopRoutineByAddress	**10.4**	**39**			□	□		
requestRoutineResultsByLocalIdentifier	**10.5**	**33**		□	□	□		
requestRoutineResultsByAddress	**10.6**	**3A**			□	□		
requestDownload	**11.1**	**34**			□	□		

Diagnostic Services			**Diagnostic Sessions**					
Diagnostic Service Name	**Section No.**	**SId Hex Value**	**SS**	**AS**	**PS**	**DS**	**VMSS**	**SSSS**
requestUpload	**11.2**	**35**			☐	☐		
transferData	**11.3**	**36**			☐	☐		
requestTransferExit	**11.4**	**37**			☐	☐		

Table 4.3 - Service Identifier value summary table

■ This symbol indicates that the service is mandatory in this diagnostic session.
◆ This symbol indicates that the service is mandatory in this diagnostic session if the server (ECU) is OBDII compliant.
☐ This symbol indicates that the service may be available in this diagnostic session. Selection of this service is defined by the vehicle manufacturer.
❍ This sysbol indicates that the service may be available in this diagnostic session. The service shall only be used to read (report) values. It is not allowed to use the service to control (adjust) values in this diagnostic session.
No symbol indicates that this service is not allowed in this diagnostic session (except in the last two columns, which are to be defined by the vehicle manufacturer / system supplier).

1) Communication services are specified in the document SSF 14230-2:1997 Keyword Protocol 2000 -Part 2 - Data Link Layer, Swedish Implementation Standard
2) OBDII emission related diagnostic services are speicified in SAE J1979. The service identifiers in SAE J1979 are co-ordinated with values specified in this document. The range is listed in table 4.3 for information only.

4.4 Response Code value summary table

The following table lists and assigns hex values for all response codes used in KWP 2000.
SSF 14230-3 has included each negative response code definition to reduce the number of documents which are referenced. The abbreviation "**RC_"** is used in combination with each "Mnemonic" to clearly identify the name of the response code.

Note: The use of (a) negative response message(s) by the server (ECU) shall be in case the server (ECU) can not respond with a positive response message on a client (tester) request message. In such case the server (ECU) shall send one of the response codes listed below as specified in figure 4.4.1.

The figure below specifies the server (ECU) behavior on a client (tester) request message. This figure shows the logic as specified in the description of the response codes and to be implemented in the server (ECU) and client (tester) as appropriate.

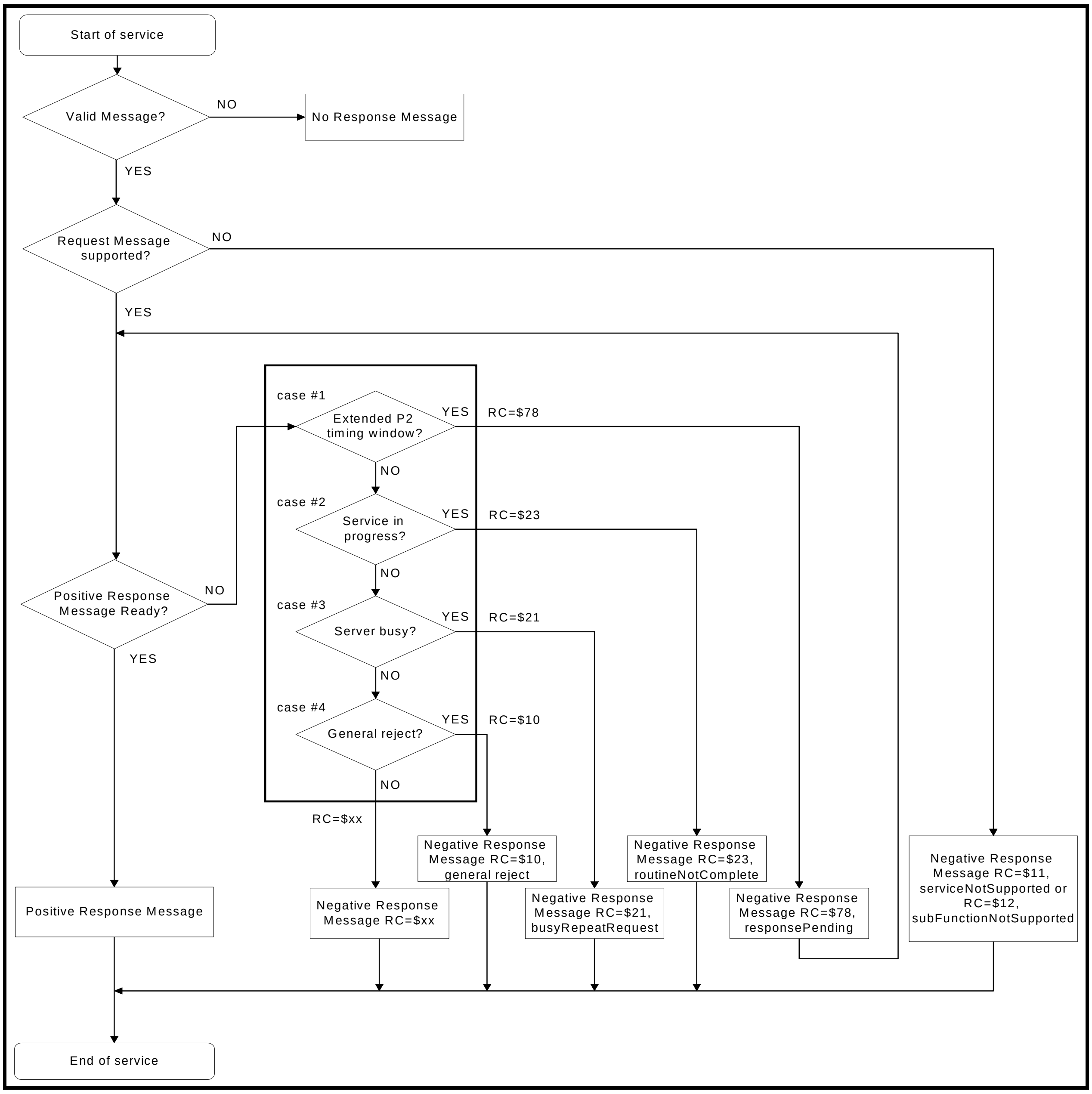

Start of service
Valid Message?
NO
No Response Message
YES
Request Message supported?
NO
YES
case #1
Extended P2 timing window?
YES
RC=$78
NO
case #2
Service in progress?
YES
RC=$23
NO
case #3
Server busy?
YES
RC=$21
NO
case #4
General reject?
YES
RC=$10
NO
Positive Response Message Ready?
NO
YES
RC=$xx
Negative Response Message RC=$10, general reject
Negative Response Message RC=$23, routineNotComplete
Negative Response Message RC=$11, serviceNotSupported or RC=$12, subFunctionNotSupported
Positive Response Message
Negative Response Message RC=$xx
Negative Response Message RC=$21, busyRepeatRequest
Negative Response Message RC=$78, responsePending
End of service

Figure 4.4.1 - Server (ECU) positive and negative response message behavior

Hex Value	Definition of Response Code	Mnemonic
00	**reservedByDocument** This value shall not be used as a response code.	**RBD**

Table 4.4.0 - Reserved response code

Hex Value	Definition of Response Code	Mnemonic
10	**generalReject** The service is rejected but the server (ECU) does not specify the reason of the rejection. **The communication timing is not affected by this response code!**	**GR**
Example: The server (ECU) shall send this response code if no other response code is available which properly indicates the rejection. It is up to the scan tool manufacturer to define the reaction of the client (tester) in case this response code is sent by the server (ECU). If a repetition of the request message is performed the number of repetitions shall be limited by the scan tool manufacturer to a certain value to prevent deadlock states.		

Table 4.4.1 - Definition of response code 10 - generalReject

Hex Value	Definition of Response Code	Mnemonic
11	**serviceNotSupported** This response code indicates that the requested action will not be taken because the server (ECU) does not support the requested service. **The communication timing is not affected by this response code!**	**SNS**
Example: The server (ECU) shall send this response code in case the client (tester) has sent a request message with a service identifier which is either unknown (not a KWP 2000 Service Identifier) or not supported by the server (ECU).		

Table 4.4.2 - Definition of response code 11 - serviceNotSupported

Hex Value	Definition of Response Code	Mnemonic
12	**subFunctionNotSupported-invalidFormat** This response code indicates that the requested action will not be taken because the server (ECU) does not support the arguments of the request message or the format of the argument bytes do not match the prescribed format for the specified service. **The communication timing is not affected by this response code!**	**SFNS-IF**
Example: The server (ECU) shall send this response code in case the client (tester) has sent a request message with a known and supported service identifier but with "sub parameters“ which are either unknown or not supported or have an invalid format. It is recommended that the client (tester) shall not repeat the identical request message.		

Table 4.4.3 - Definition of response code 12 - subFunctionNotSupported-invalidFormat

Hex Value	Definition of Response Code	Mnemonic
21	**busy-repeatRequest** This response code indicates that the server (ECU) is temporarily too busy to perform the requested operation and the requested service will not be started. In this circumstance repetition of the "identical request message" or "another request message" shall be performed by the client (tester). This response code shall be returned for example while a server (ECU) is in the process of clearing stored DTC(s) information or fetching information. **The communication timing is not affected by this response code!**	**B-RR**
Example: The server (ECU) shall send this response code in case the client (tester) has sent a request message at a time when the server (ECU) is busy with internal processing not related to the current request message. It is recommended that the client (tester) shall repeat the request message within the P3 timing window.		

Table 4.4.4 - Definition of response code 21 - busy-RepeatRequest

Hex Value	Definition of Response Code	Mnemonic
22	**conditionsNotCorrectOrRequestSequenceError** This response code indicates that the requested action will not be taken because the server (ECU) prerequisite conditions are not met. This request may occur when sequence sensitive requests are issued in the wrong order. **The communication timing is not affected by this response code!**	**CNCORSE**
Example: The server (ECU) shall send this response code in case the client (tester) has sent a known and supported request message at a time where the server (ECU) has expected another request message because of a predefined sequence of services. A typical example of occurrence is the securityAccess service which requires a sequence of messages as specified in the message description of this service.		

Table 4.4.5 - Definition of response code 22 - conditionsNotCorrectOrRequestSequenceError

Hex Value	Definition of Response Code	Mnemonic
23	**routineNotCompleteOrServiceInProgress** This response code indicates that the request message was properly received by the server (ECU) and the routine (service), which has been initiated by the request message is already in process, but not yet completed. The server (ECU) knows in advance that the processing time is greater than the P2 timing window. The successful execution and completion of the request message will not be indicated by a positive response message. **In case the client (tester) repeats the request message the server (ECU) shall "not reinitiate the task" if the initial task has not been completed!** **The communication timing is not affected by this response code!**	**RNC**
Example: The server (ECU) shall send this response code in case the client (tester) has sent a clearDiagnosticInformation request message and if the server (ECU) is still clearing the diagnostic information and will not be finished before the timing parameter value of P2max is reached. In this particular case the server (ECU) shall send the negative response message with this response code. In case the client (tester) repeats the identical request message, the server (ECU) shall not restart the clearDiagnosticInformation routine. It is recommended to sent the next request message close to the timing parameter value of P3max. This increases the amount of time available to the server (ECU) to finish the routine. See also message flow examples in section 5.3.1.2 and section 5.3.2.2.		

Table 4.4.6 - Definition of response code 23 - routineNotComplete

Hex Value	Definition of Response Code	Mnemonic
31	**requestOutOfRange** This response code indicates that the requested action will not be taken because the server (ECU) detects the request message contains a data byte(s) which attempt(s) to substitute (a) value(s) beyond its range of authority (e.g. attempting to substitute a data byte of 111 when the data is only defined to 100). **The communication timing is not affected by this response code!**	**ROOR**
Example: The server (ECU) shall send this response code in case the client (tester) has sent a request message including data bytes to adjust a variant which does not exist (invalid) in the server (ECU). This response code shall be implemented for all services which allow the client (tester) to write data or adjust functions by data in the server (ECU). The communication timing is not affected!		

Table 4.4.7 - Definition of response code 31 - requestOutOfRange

Hex Value	Definition of Response Code	Mnemonic
33	**securityAccessDenied-securityAccessRequested** This response code indicates that the requested action will not be taken because the server's (ECU's) security strategy has not been satisfied by the client (tester). **The communication timing is not affected by this response code!**	**SAD-SAR**
Example: The server (ECU) shall send this response code if one of the following cases occur: • the test conditions of the server (ECU) are not met • the required message sequence e.g. startDiagnosticSession, securityAccess is not met the client (tester) has sent a request message which requires an unlocked server (ECU)		

Table 4.4.8 - Definition of response code 33 - securityAccessDenied-securityAccessRequested

Hex Value	Definition of Response Code	Mnemonic
35	**invalidKey** This response code indicates that security access has not been given by the server (ECU) because the key sent by the client (tester) did not match with the key in the server's (ECU's) memory. This counts as an attempt to gain security. The server (ECU) shall remain locked! **The communication timing is not affected by this response code!**	**IK**
Example: The server (ECU) shall send this response code in case the client (tester) has sent a securityAccess request message with the sendKey and Key parameter where the key value does not match the key value stored in the server's (ECU's) memory. The server (ECU) shall increment its internal securityAccessFailed counter.		

Table 4.4.9 - Definition of response code 35 - invalidKey

Hex Value	Definition of Response Code	Mnemonic
36	**exceedNumberOfAttempts** This response code indicates that the requested action will not be taken because the client (tester) has unsuccessfully attempted to gain security access more times than the server's (ECU's) security strategy will allow. Refer to message description of the securityAccess service definition. **The communication timing is not affected by this response code!**	**ENOA**
Example: The server (ECU) shall send this response code in case the client (tester) has sent a securityAccess request message with the sendKey and Key parameter where the key value does not match the key value stored in the server's (ECU's) memory and the number of attempts (securityAccessFailed counter value) have reached the server's (ECU's) securityAccessFailed calibration value.		

Table 4.4.10 - Definition of response code 36 - exceedNumberOfAttempts

Hex Value	Definition of Response Code	Mnemonic
37	**requiredTimeDelayNotExpired** This response code indicates that the requested action will not be taken because the client's (tester's) latest attempt to gain security access was initiated before the server's (ECU's) required timeout period had elapsed. **The communication timing is not affected by this response code!**	**RTDNE**
Example: An invalid Key requires the client (tester) to start over from the beginning with a securityAccess service. If the security protection algorithm is not passed after "X" failed attempts ("X" = specified in the server's (ECU's) memory), all additional attempts are rejected for at least "Y" seconds ("Y" = specified in the server's (ECU's) memory). The "Y" second timer shall begin with the "X" failed attempt or upon a power/ignition cycle or reset of the server (ECU) to ensure a security lockout after all power interruptions.		

Table 4.4.11 - Definition of response code 37 - requiredTimeDelayNotExpired

Hex Value	Definition of Response Code	Mnemonic
40	**downloadNotAccepted** This response code indicates that an attempt to download to a server's (ECU's) memory cannot be accomplished due to some fault conditions. **The communication timing is not affected by this response code!**	**DNA**
Example: The server (ECU) shall send this response code in case the client (tester) has sent a requestDownload request message which the server (ECU) can not accept and the failure can not be described with another negative response code.		

Table 4.4.12 - Definition of response code 40 - downloadNotAccepted

Hex Value	Definition of Response Code	Mnemonic
41	**improperDownloadType** This response code indicates that an attempt to download to a server's (ECU's) memory cannot be accomplished because the server (ECU) does not support the type of download being attempted. **The communication timing is not affected by this response code!**	**IDT**
Example: The server (ECU) shall send this response code in case the client (tester) has sent a requestDownload request message with transferRequestParameters which are unknown to the server (ECU) and therefore will not be supported.		

Table 4.4.13 - Definition of response code 41 - improperDownloadType

Hex Value	Definition of Response Code	Mnemonic
42	**canNotDownloadToSpecifiedAddress** This response code indicates that an attempt to download to a server's (ECU's) memory cannot be accomplished because the server (ECU) does not recognize the target address for the download as being available. **The communication timing is not affected by this response code!**	**CNDTSA**
Example: The server (ECU) shall send this response code in case the client (tester) has sent a requestDownload request message with transferRequestParameters which includes a memory address which can not be downloaded to.		

Table 4.4.14 - Definition of response code 42 - canNotDownloadToSpecifiedAddress

Hex Value	Definition of Response Code	Mnemonic
43	**canNotDownloadNumberOfBytesRequested** This response code indicates that an attempt to download to a server's (ECU's) memory cannot be accomplished because the server (ECU) does not recognize the number of bytes for the download as being available. **The communication timing is not affected by this response code!**	**CNDNOBR**
Example: The server (ECU) shall send this response code in case the client (tester) has sent a requestDownload request message with transferRequestParameters which includes a value in the uncompressedMemorySize parameter which does not fit with the number of bytes expected for this download by the server (ECU).		

Table 4.4.15 - Definition of response code 43 - canNotDownloadNumberOfBytesRequested

Hex Value	Definition of Response Code	Mnemonic
50	**uploadNotAccepted** This response code indicates that an attempt to upload from a server's (ECU's) memory cannot be accomplished due to some fault conditions. **The communication timing is not affected by this response code!**	**UNA**
Example: The server (ECU) shall send this response code in case the client (tester) has sent an requestUpload request message which the server (ECU) can not accept and the failure can not be described with another negative response code.		

Table 4.4.16 - Definition of response code 50 - uploadNotAccepted

Hex Value	Definition of Response Code	Mnemonic
51	**improperUploadType** This response code indicates that an attempt to upload from a server's (ECU's) memory cannot be accomplished because the server (ECU) does not support the type of upload being attempted. **The communication timing is not affected by this response code!**	**IUT**
Example: The server (ECU) shall send this response code in case the client (tester) has sent an requestUpload request message with transferRequestParameters which are unknown to the server (ECU) and therefore will not be supported.		

Table 4.4.17 - Definition of response code 51 - improperUploadType

Hex Value	Definition of Response Code	Mnemonic
52	**canNotUploadFromSpecifiedAddress** This response code indicates that an attempt to upload from a server's (ECU's) memory cannot be accomplished because the server (ECU) does not recognize the target address for the upload as being available. **The communication timing is not affected by this response code!**	**CNUFSA**
Example: The server (ECU) shall send this response code in case the client (tester) has sent a requestUpload request message with transferRequestParameters which includes a memory address which can not be uploaded from.		

Table 4.4.18 - Definition of response code 52 - canNotUploadFromSpecifiedAddress

Hex Value	Definition of Response Code	Mnemonic
53	**canNotUploadNumberOfBytesRequested** This response code indicates that an attempt to upload from a server's (ECU's) memory cannot be accomplished because the server (ECU) does not recognize the number of bytes for the upload as being available. **The communication timing is not affected by this response code!**	**CNUNOBR**
Example: The server (ECU) shall send this response code in case the client (tester) has sent a requestUpload request message with transferRequestParameters which includes a value in the uncompressedMemorySize parameter which does not fit with the number of bytes expected for this upload by the server (ECU).		

Table 4.4.19 - Definition of response code 53 - canNotUploadNumberOfBytesRequested

Hex Value	Definition of Response Code	Mnemonic
71	**transferSuspended** This response code indicates that a data transfer operation was halted due to some fault. **The communication timing is not affected by this response code!**	**TS**
Example: The server (ECU) shall send this response code in case the client (tester) has sent a request message with incorrect data. In such case this response code shall be sent if no other response code matches the situation..		

Table 4.4.20 - Definition of response code 71 - transferSuspended

Hex Value	Definition of Response Code	Mnemonic
72	**transferAborted** This response code indicates that a data transfer operation was halted due to some fault, but will not be completed later. **The communication timing is not affected by this response code!**	**TA**
Example: The server (ECU) shall send this response code in case the client (tester) has sent a request message with data which require the server (ECU) to abort the data transfer.		

Table 4.4.21 - Definition of response code 72 - transferAborted

Hex Value	Definition of Response Code	Mnemonic
74	**illegalAddressInBlockTransfer** This response code indicates that the starting address included in a request message is either out of range , protected, the wrong type of memory for the receiving data, or cannot be written to for some reason. **The communication timing is not affected by this response code!**	**IAIBT**
Example: The server (ECU) shall send this response code in case the client (tester) has sent a request message with an memoryAddress which is in conflict with the memory addresses supported by the server (ECU).		

Table 4.4.22 - Definition of response code 74 - illegalAddressInBlockTransfer

Hex Value	Definition of Response Code	Mnemonic
75	**illegalByteCountInBlockTransfer** This response code indicates that the number of data bytes included in the request message is either more than the request message can accommodate, requires more memory than is available at the requested starting address, or cannot be handled by the software. **The communication timing is not affected by this response code!**	**IBCIBT**
Example: The server (ECU) shall send this response code in case the client (tester) has sent a request message with an amount of data which does not fit with the number of bytes expected for this data transfer.		

Table 4.4.23 - Definition of response code 75 - illegalByteCountInBlockTransfer

Hex Value	Definition of Response Code	Mnemonic
76	**illegalBlockTransferType** This response code indicates that the transferRequestParameter(s) included in the transferData request message is (are) not valid for this application. **The communication timing is not affected by this response code!**	**IBTT**
Example: The server (ECU) shall send this response code in case the client (tester) has sent a transferData request message with transferRequestParameter(s) which include invalid/illegal parameter values for the requested data transfer.		

Table 4.4.24 - Definition of response code 76 - illegalBlockTransferType

Hex Value	Definition of Response Code	Mnemonic
77	**blockTransferDataChecksumError** This response code indicates that the data checksum calculated for the transferData messages does not agree with the expected value. **The communication timing is not affected by this response code!**	**BTDCE**
Example: The server (ECU) shall send this response code in case the client (tester) has completed the data transfer to the server (ECU) and the server (ECU) has computed a data checksum from the data transfer which is different than the one internally stored.		

Table 4.4.25 - Definition of response code 77 - blockTransferDataChecksumError

Hex Value	Definition of Response Code	Mnemonic
78	**reqCorrectlyRcvd-RspPending (requestCorrectlyReceived-ResponsePending)** This response code indicates that the request message was received correctly, and that any parameters in the request message were valid, but the action to be performed may not be completed yet. This response code can be used to indicate that the request message was properly received and does not need to be re-transmitted, but the server (ECU) is not yet ready to receive another request. **This response code shall only be used in a negative response message if the server (ECU) will not be able to receive further request messages from the client (tester) within the P3 timing window. This may be the case if the server (ECU) does data processing or executes a routine which does not allow any attention to serial communication.** The following description specifies the communication timing method: This response code shall manipulate the P2max timing parameter value in the server (ECU) and the client (tester). The P2max timing parameter is set to the value [in ms] of the P3max timing parameter. In addition, the client (tester) shall disable the testerPresent service. As soon as the server (ECU) has completed the task (routine) initiated by the request message it shall send either a positive or negative response message (negative response message with a response code other than $78) based on the last request message received. When the client (tester) has received the positive response message which has been preceded by the negative response message(s) with this response code, the client (tester) and the server (ECU) shall reset the P2 timing parameter to the previous P2 timing value. In addition, the client (tester) shall re-enable the testerPresent service. **The client (tester) shall not repeat the request message after the reception of a negative response message with this response code!** ***The communication timing is affected if this response code is used (refer to section 4.4.1 in KWP 2000 Part 2: Data Link Layer Recommended Practice)!***	**RCR-RP**
Example: A typical example where this response code may be used is when the client (tester) has sent a request message which includes data to be programmed or erased in flash memory of the server (ECU). If the programming/erasing routine (usually executed out of RAM) routine is not able to support serial communication while writing to the flash memory the server (ECU) shall send a negative response message with this response code. As soon as the data are programmed or erased in flash memory the server (ECU) shall send a positive response message (or negative response message but not RC_78).		

Table 4.4.26 - Definition of response code 78 - reqCorrectlyRcvd-RspPending (requestCorrectlyReceived-ResponsePending)

Hex Value	Definition of Response Code	Mnemonic
79	**incorrectByteCountDuringBlockTransfer** This response code indicates that the number of data bytes that was expected to be sent was not the same as the number of data bytes received. **The communication timing is not affected by this response code!**	**IBCDBT**
Example: The server (ECU) shall send this response code in case the client (tester) has sent the requestTransferExit request message which indicates to the server (ECU) that the client (tester) will not send any further data bytes but the number of data bytes expected by the server (ECU) does not match the number of data bytes sent by the client (tester).		

Table 4.4.27 - Definition of response code 79 - incorrectByteCountDuringBlockTransfer

Hex Value	Definition of Response Code	Mnemonic
80	**serviceNotSupportedInActiveDiagnosticSession** This response code indicates that the requested action will not be taken because the server (ECU) does not support the requested service in the diagnostic session currently active. **The communication timing is not affected by this response code!**	**SNSIADS**
Example: The server (ECU) shall send this response code in case the client (tester) has sent a request message (requestDownload) which e.g. is only supported in the programming session but the server (ECU) is in a diagnostic session which does not support this service.		

Table 4.4.28 - Definition of response code 80 - serviceNotSupportedInActiveDiagnosticMode

Hex Value	Definition of Response Code	Mnemonic
81 - 8F	**reservedByDocument** This range of values shall be reserved for diagnostic service implementation related future response code definitions.	**RBD**
90 - F9	**vehicleManufacturerSpecific** This range of values is reserved for vehicle manufacturer specific use.	**VMS**
FA - FE	**systemSupplierSpecific** This range of values is reserved for system supplier specific use.	**SSS**
FF	**reservedByDocument** This value shall not be used as a response code.	**RBD**

Table 4.4.29 - Reserved ranges of response codes

5 General implementation rules

5.1 Parameter definitions

The following rules in regard to parameter definitions shall apply:

- The subsequent sections define the services of each functional unit. In these sections, the service structure makes reference to parameters, in order to describe the allowable values for such parameters. Parameters which are specific to a functional unit are described in the corresponding section.
- This document lists and defines response codes and values. Negative response codes are specified in section 4.4. Other response codes may be reserved either for future definition by this document or for the system designer's specific use.
- This document specifies the parameters which shall be used within each KWP 2000 service.
- If a parameter value or a record value consists of more than one byte the most significant byte shall always be transmitted first, followed by bytes of decreasing significance.
- The sequence of parameters within a service shall not be changed during an implementation.
- This document specifies the parameter memoryAddress based on a three (3) byte address (High Byte, Middle Byte and Low Byte). Additional bytes of specifying the memoryAddress (e.g. memory type identifier, larger address range) may be implemented and is the responsibility of the vehicle manufacturer. This applies to all services which use the memoryAddress parameter.

5.2 Functional and physical addressed service requests

Two (2) different addressing methods are specified in KWP 2000 to send a service request to a server(s).

- Functional addressing with a three or four byte header is used by the client if it doesn't know the physical address of the server that shall respond to a service request or if more than one server can respond to the request.
- Physical addressing with a three or four byte header shall always be a dedicated message to one server. The header of the service request message indicates (target address) which server shall respond to the service request message.
- Functional and Physical addressing methods are specified in detail in the document SSF 14230-2.
- The data link shall always be initialized prior to sending any of the KWP 2000 services.

5.3 Message flow examples of physical/functional addressed services

5.3.1 Physical addressed services

5.3.1.1 Physical addressed service with positive/negative response message

The table below shows a typical service request followed by a positive response message and a service request followed by a negative response message.

time	client (Tester)	server (ECU)
P3	<Service Name> Request[...]	
P2		<Service Name> PositiveResponse[...]
P3	<Service Name> Request[...]	
P2		<Service Name> NegativeResponse[RC]
P3	<Service Name> Other Request[...]	
P2		<Service Name> Other PositiveResponse[...]

Table 5.3.1.1 - Message flow example of physical addressed service

5.3.1.2 Physical addressed service with periodic transmissions

The table below shows a message flow which describes a physical addressed service request with multiple positive response messages { PT = PeriodicTransmissions }.

time	client (Tester)	server (ECU)
P3	<startDiagnosticSession> Request[PT]	
P2		<startDiagnosticSession> PositiveResponse#1[PT]
P2		:
P2		<startDiagnosticSession> PositiveResponse#k[...]
P3*	<Service Name> Request A[...]	
P2		<Service Name> PositiveResponse A#1[...]
P2		:
P2		<Service Name> PositiveResponse A#m[...]
P3*	<Service Name> Request B[...]	
P2		<Service Name> PositiveResponse B#1[...]
P2		:
P2		<Service Name> PositiveResponse B#n[...]
P3*	<stopDiagnosticSession> Request	
P2	***OR***	<stopDiagnosticSession> PositiveResponse
P3*	<startDiagnosticSession> Request[...]	***OR***
P2		<startDiagnosticSession> PositiveResponse[...]
P3	<Service Name> Request C[...]	
P2		<Service Name> PositiveResponse C[...]

Table 5.3.1.2 - Message flow example of physical addressed service with periodic transmissions

P3* **: The values of the "*P3**" timing parameters shall be less than the value of "P2min" in order to allow the client (tester) to send a new request message (refer to SSF 14230-2).**

Above message flow describes a physical addressed service request with the periodic transmission mode enabled. The request message is followed by multiple positive response messages from the physically addressed server. The periodically transmitted response messages are terminated by the client with a stopDiagnosticSession request message send to the server which has been initiated within the **"P3*"** timing window.

5.3.1.3 Physical addressed service and negative response message(s) with "routineNotComplete ($23)" and "busy-RepeatRequest ($21)"

The table below shows a message flow which describes a physical addressed service request followed by a negative response message with the response code set to "routineNotComplete ($23)".

time	client (Tester)	server (ECU)
P3	<Service Name> Request[...]	**{ server has started routine! }**
P2		<Service Name> NegativeResponse[routineNotComplete ($23)]
P3	<Service Name> Request[...]	
P2		<Service Name> NegativeResponse[busy-RepeatRequest ($21)]
	:	:
P3	<Service Name> Request[...]	
P2		<Service Name> NegativeResponse[busy-RepeatRequest ($21)]
P3	<Service Name> Request[...]	**{ server has stopped routine! }**
P2		<Service Name> PositiveResponse[...]
		OR
P2		<Service Name> NegativeResponse[RC ≠ busy-RepeatRequest ($21)]

Table 5.3.1.3 - Physical addressing and negative response with "routineNotComplete ($23)" and "busy-RepeatRequest ($21)"

Above message flow example is based on a request message from the client which cause the server to respond with a negative response message including the negative response code "routineNotComplete ($23)". This response code indicates that the request message was properly received by the server and the routine/task/function (initiated by the request message) is in process, but not yet completed. If the client repeats or sends another request message the server shall "not reinitiate the task" (in case of the same request message) if the initial task has not been completed. The server shall send another negative response

message with the response code "busy-RepeatRequest ($21)". The negative response message shall be sent on each request message as long as the server has not yet completed the initial routine/task/function. If the server has finished the routine/task/function it shall respond with either a positive response message or a negative response message with a response code not equal to "busy-RepeatRequest ($21)".
The communication timing is not affected!

Applications which may require above message flow are:
- clearDiagnosticInformation,
- execution of routines
- securityAccess

5.3.1.4 Physical addressed service with "server can not send a positive response within required timing"

5.3.1.4.1 Physical addressed service and negative response message with "reqCorrectlyRcvd-RspPending ($78) within Normal or Extended Timing"

The table below shows a message flow which describes a physical addressed service request followed by a negative response message with the response code set to "requestCorrectlyReceived-ResponsePending".

time	client (Tester)	server (ECU)
P3	<Service Name> Request[...]	
P2		<Service Name> NegRsp#1[reqCorrectlyRcvd-RspPending ($78)]
		:
P2*		<Service Name> NegRsp#n[reqCorrectlyRcvd-RspPending ($78)]
P2*		<Service Name> PositiveResponse[...]
P3	<Service Name> Other Request[...]	
P2		<Service Name> Other PositiveResponse[...]

Table 5.3.1.4.1 - Example of physical addressing and negative response with "reqCorrectlyRcvd-RspPending ($78)"

P2* **: P2min to P3max (refer to SSF 14230-2)**

Above message flow example is based on a request message from the client which cause the server to respond with one or multiple negative response message including the negative response code "requestCorrectlyReceived-ResponsePending ($78)".
This response code indicates that the request message was received correctly, and that any parameters in the request message were valid, but the action to be performed may not be completed yet. This response code can be used to indicate that the request message was properly received and does not need to be retransmitted, but the server is not yet ready to receive another request.
The negative response message may be sent once or multiple times within a service if required by the server!
During the period of (a) negative response message(s) the testerPresent service shall be disabled in the client!
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 within the P3 timing window. This may be the case if the server does data processing or executes a routine which does not allow any attention to serial communication.

Note: The communication timing method is specified in SSF 14230-2.

Applications which may require above message flow are:
- clearDiagnosticInformation
- execution of routines
- transferData request messages including up to 255 data bytes
- Flash EEPROM or EEPROM memory programming

5.3.1.4.2 Physical addressed service with "server can not send a positive response within Default Timing"

The table below shows a message flow which describes a physical addressed service request followed by a positive response message sent with previously modified timing.

time	client (Tester)	server (ECU)
P3	startDiagnosticSession.Request[...]	
P2		startDiagnosticSession.PosRsp[...]
P3	accessTimingParameters.Request[readLimits]	
P2		accessTimingParameters.PosRsp[readLimits, P2-P4]
P3	accessTimingParameters.Request[setValues, P2-P4]	
P2	{ e.g. P2max = $F2 (18850 ms) }	accessTimingParameters.PosRsp[setValues]
	{ Modified Timing is active! }	**{ Modified Timing is active! }**
P3	<Service Name> Request[...]	
P2		<Service Name> PositiveResponse[...]
P3	<stopDiagnosticSession> Request	
P2	***OR***	<stopDiagnosticSession> PositiveResponse
P3	<startDiagnosticSession> Request[...]	***OR***
P2		<startDiagnosticSession> PositiveResponse[...]
	{ Default Timing is active! }	**{ Default Timing is active! }**
P3	<Service Name> Request[...]	
P2		<Service Name> PositiveResponse C[...]

Table 5.3.1.4.2 - Physical addressing and modified timing with accessTimingParameters service

P3, P2 **: New timing parameter values are active!**

Above message flow example describes the method of modifying the timing parameter values in the server and the client by using the accessTimingParameters service as specified in the SSF 14230-2.
This method shall be used in case a server does not support a negative response message with the response code "requestCorrectlyReceived-ResponsePending ($78)".

5.3.2 Functional addressed services

5.3.2.1 Functional addressed service with positive/negative response message

The table below shows a message flow example which describes a functional addressed service request followed by response messages of multiple servers (ECU#1, ECU#2 ... ECU#n).

time	client (Tester)	server (ECU)
P3	<Service Name> Request[...] { functional }	
P2		<Service Name> PositiveResponse[...] {ECU#1}
P2		<Service Name> NegativeResponse[...] {ECU#2}
:		**:**
P2		<Service Name> PositiveResponse[...] {ECU#n}
P3	<Service Name> Other Request[...]{ functional }	
P2		<Service Name> Other PositiveResponse[...] {ECU#1}
P2		<Service Name> Other NegativeResponse[...] {ECU#2}
:		**:**
P2		<Service Name> Other PositiveResponse[...] {ECU#n}

Table 5.3.2.1 - Message flow example of functional addressed service

Above message flow example is based on a functional request message sent by the client. A number of servers has been initialized previously which send positive and negative response messages.

5.3.2.2 Functional addressed service and negative response message(s) with "routineNotComplete ($23)" and "busy-RepeatRequest ($21)"

The table below shows a message flow which describes a functional addressed service request followed by a negative response message with the response code set to "routineNotComplete ($23)" from one (1) of three (3) server. All other servers send positive response messages.

time	client (Tester)	server (ECU)
P3	<Service Name> Request[...] { functional }	**{ server has started routine! }**
P2		<Service Name> NegRsp[routineNotComplete ($23)] {ECU#1}
P2		<Service Name> PositiveResponse[...] {ECU#2}
P2		<Service Name> PositiveResponse[...] {ECU#3}
P3	<Service Name> Request[...] { functional }	
P2		<Service Name> NegRsp[busy-RepeatRequest ($21)] {ECU#1}
P2		<Service Name> PositiveResponse[...] {ECU#2}
P2		<Service Name> PositiveResponse[...] {ECU#3}
P3	<Service Name> Request[...] { functional }	**{ server has stopped routine! }**
P2		<Service Name> PositiveResponse[...] {ECU#1}
P2		<Service Name> PositiveResponse[...] {ECU#2}
P2		<Service Name> PositiveResponse[...] {ECU#3}

Table 5.3.2.2 - Functional addressing and negative response with "busy-RepeatRequest ($21)"

Above message flow example is based on a functional request message from the client which cause one server (ECU#1) to respond with a negative response message including the negative response code "routineNotComplete ($23)". The other servers (ECU#2, ECU#3) send a positive response message.
The response code indicates that the request message was properly received by the server and the routine/task/function (initiated by the request message) is in process, but not yet completed. If the client repeats or sends another functional request message the server shall "not reinitiate the task" (in case of the same request message) if the initial task has not been completed. The server shall send another negative response message with the response code "busy-RepeatRequest ($21)". The negative response message shall be sent on each request message as long as the server has not yet completed the initial routine/task/function. If the server (ECU#1) has finished the routine/task/function, it shall respond with either a positive response message or a negative response message with a response code not equal to "busy-RepeatRequest ($21)".
The communication timing is not affected!

Applications which may require above message flow are:
- clearDiagnosticInformation,
- execution of routines
- securityAccess

5.3.2.3 Functional addressed service and negative response message with " reqCorrectlyRcvd-RspPending ($78)"

The table below shows a message flow example which describes a functional addressed request message followed by a negative response message with the response code set to "requestCorrectlyReceived-ResponsePending" from one server (ECU#1) and positive response messages from the other servers (ECU#2, ECU#3).

time	client (Tester)	server (ECU)
P3	<Service Name> Req[...]	
P2	{ Functional }	<Service Name> NegRsp#1[reqCorrectlyRcvd-RspPendg ($78)] {ECU#1}
P2*		<Service Name> PositiveResponse[...] {ECU#2}
P2*		<Service Name> PositiveResponse[...] {ECU#3}
P2*		<Service Name> NegRsp#2[reqCorrectlyRcvd-RspPendg ($78)] {ECU#1}
P2*		**:**
P2*		<Service Name> NegRsp#n[reqCorrectlyRcvd-RspPendg ($78)] {ECU#1}
P2*		<Service Name> PositiveResponse[...] {ECU#1}
P3	<Service Name> Req[...] { Functional }	
P2		<Service Name> PositiveResponse[...] {ECU#1}
P2		<Service Name> PositiveResponse[...] {ECU#2}
P2		<Service Name> PositiveResponse[...] {ECU#3}

Table 5.3.2.3 - Example of functional addressing and negative response with "reqCorrectlyRcvd-RspPending ($78)"

P2* **: P2min to P3max (refer to SSF 14230-2).**

Above message flow example is based on a request message from the client which cause the server to respond with one or multiple negative response message including the negative response code "requestCorrectlyReceived-ResponsePending". This response code indicates that the request message was received correctly, and that any parameters in the request message were valid, but the action to be performed may not be completed yet. This response code can be used to indicate that the request message was properly received and does not need to be retransmitted, but the server is not yet ready to receive another request. The negative response message may be sent once or multiple times within a service if required by the server !
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 within the P3 timing window. This may be the case if the server does data processing or executes a routine which does not allow any attention to serial communication.

Note: The communication timing method is specified in SSF 14230-2.

Applications which may require above message flow are:
- clearDiagnosticInformation
- execution of routines
- transferData request messages including up to 255 data bytes
- Flash EEPROM or EEPROM memory programming

5.4 Data Scaling

5.4.1 Data Scaling Purpose

The purpose of data scaling is to enable the client (tester) to convert "raw data" which are retrieved from the server (ECU) into engineering units or ECU identification data (e.g. ECU Hardware Part Number: 90357412). The scaling data are contained in Data Scaling Tables. Those always have the same structure and are an extension of the scaling definition described in the SAE J2190 document. The use of Data Scaling Tables is vehicle manufacturer specific.

5.4.2 Data Scaling Table Structure

The table below specifies the general structure of a data scaling table.

Type	Description	Cvt	Hex value
	scalingTableRecordValue=[		
Offset	scalingOffset#1	M	xx
Identifier	parameterIdentifier#1.1	M	xx
Identifier	parameterIdentifier#1.2	C1	xx
DataValue	scalingByte#1.1	M	xx
DataValue	scalingByteExtention#1.1.1	C2	xx
:	:	:	:
DataValue	scalingByteExtention#1.1.p	C2	xx
:	:	:	:
DataValue	scalingByte#1.m	C1	xx
DataValue	scalingByteExtention#1.m.1	C2	xx
:	:	:	:
DataValue	scalingByteExtention#1.m.p	C2	xx
:	:	:	:
Offset	scalingOffset#n	C1	xx
Identifier	parameterIdentifier#n.1	M	xx
Identifier	parameterIdentifier#n.2	C1	xx
DataValue	scalingByte#n.1	M	xx
DataValue	scalingByteExtention#n.1.1	C2	xx
:	:	:	:
DataValue	scalingByteExtention#n.1.p	C2	xx
:	:	:	:
DataValue	scalingByte#n.m	C1	xx
DataValue	scalingByteExtention#n.m.1	C2	xx
:	:	:	:
DataValue	scalingByteExtention#n.m.p	C2	xx
OffsetEOT	scalingOffset#n+1 {End of Table}]	M	FF

Table 5.4.2.1 - General data scaling table structure

C1 = condition: parameter only exists if it requires more than 1 byte.
C2 = condition: parameter only exists in combination with some special scalingBytes.

Above scaling table specifies the general structure of scaling tables referenced by this document. Each parameter scaling structure starts with a scalingOffset, one or multiple parameterIdentifier bytes (#1 to #2) and ends with one or multiple scaling bytes (#1 to #m). Scaling byte extention (#1 to #p) may be added to the scaling byte to give more information about the described raw data.

5.4.3 ScalingOffset

The ***scalingOffset*** is of type offset and is used to build a linked list of scaling structures referenced by a single ***parameterIdentifier***. The ScalingOffset = $FF indicates an EOT (End of Table).

5.4.4 Parameter Identifier

The ***parameterIdentifier*** is used to identify a specific parameter by its unique number which may consist of a single byte or multiple bytes. Each parameterIdentifier may only appear once in a Data Scaling Table. E.g. in case an ECU consists of two (2) micro controllers, each with its own software identification number, it is not possible to use the identificationOption “systemSupplierECUSoftwareNumber” twice (one for each micro controller). Instead these numbers have to be combined into a single parameter. The scaling table can then be used to define the format of this parameter. A message flow example with this configuration is described in section 6.6.5.2.

The following are parameterIdentifiers specified in this document for which data scaling can be used:

• identificationOption	(refer to section 6.6.3)
• recordLocalIdentifier	(refer to section 7.1.3)
• recordCommonIdentifier	(refer to section 7.2.3)
• inputOutputLocalIdentifier	(refer to section 9.1.3)
• inputOutputCommonIdentifier	(refer to section 9.2.3)

5.4.5 ScalingByte

The ***scalingByte*** consists of one byte (high and low nibble). The high nibble defines the type of information which is used to represent the parameter. The scaling byte low nibble defines the number of bytes used to represent the parameter in a datastream.

It is recommended to place vehicleManufacturerSpecific scaling bytes and scaling bytes not defined in this document after other scalingBytes for the same parameter identification. The user of the table have to ignor unknown scaling bytes and the following data for the identified parameter as they may include scalingByteExtension.

5.4.5.1 ScalingByte High Nibble

Encoding of High Nibble (Hex)	Description of Data Type	Cvt	Mnemonic
0	**unSignedNumeric (1 to 4 bytes)** This encoding uses a common binary weighting scheme to represent a value by mean of discrete incremental steps. One byte affords 256 steps; two bytes yields 65536 steps, etc.	**U**	**USN**
1	**signedNumeric (1 to 4 bytes)** This encoding uses a two's complement binary weighting scheme to represent a value by mean of discrete incremental steps. One byte affords 256 steps; two bytes yields 65536 steps, etc.	**U**	**SN**
2	**bitMappedReportedWithOutMask** 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 a second byte which contains the validity mask.	**U**	**BMR WOM**
3	**bitMappedReportedWithMask** 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 two bytes; one representing status and one containing the validity mask.	**U**	**BMR WM**
4	**binaryCodedDecimal** Conventional Binary Coded Decimal encoding is used to represent two numeric digits per byte. The upper nibble is used to represent the most significant digit (0 - 9), and the lower nibble the least significant digit (0 -9).	**U**	**BCD**
5	**stateEncodedVariable (1 byte)** This encoding uses a binary weighting scheme to represent up to 256 distinct states. An example is a parameter which represents the status of the Ignition Switch. Codes "00", "01", "02" and "03" may indicate ignition off, locked, run, and start, respectively. The representation is always limited to one (1) byte.	**U**	**SEV**
6	**ASCII (1 to 15 bytes for each scalingByte)** Conventional ASCII encoding is used to represent up to 128 standard characters with the MSB = logic 0. An additional 128 custom characters may be represented with the MSB = logic 1.	**U**	**ASCII**

Encoding of High Nibble (Hex)	Description of Data Type	Cvt	Mnemonic
7	**signedFloatingPoint** Floating point encoding is used for data that needs to be represented in floating point or scientific notation. Standard IEEE formats shall be used according to ANSI/IEEE Std 754 - 1985.	**U**	**SFP**
8	**packet** Packets contain multiple data values, usually related, each with unique scaling. Scaling information is not included for the individual values.	**U**	**P**
9	**formula** A formula is used to calculate a value from the raw data. Formula Identifiers are specified in section 5.4.6.1.	**U**	**F**
A	**unit/format** The units and formats are used to present the data in a more user-friendly format. Unit and Format Identifiers are specified in section 5.4.6.2. 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 data scaling table.	**U**	**U**
B	**unSignedNumericWithIndication (1 to 4 bytes)** This encoding uses a format according to SAE J1939/71 where some of the highest values are used to represent pre-defined indications.	**U**	**USNWI**
C - E	**vehicleManufacturerSpecific** This range of values is reserved for vehicle manufacturer specific use.		
F	**reservedByDocument** Reserved by this document for future definition.	**M**	**RBD**

Table 5.4.5.1 - Encoding of High Nibble of ScalingByte

5.4.5.2 ScalingByte Low Nibble

Encoding of Low Nibble (Hex)	Description of Low Nibble	Cvt	Mnemonic
0 - F	**numberOfBytesOfParameter** This range of values specifies the number of data bytes in a data stream referenced by a parameter identifier. The length of a parameter is defined by the scaling byte(s) which is always preceded by a parameter identifier (one or multiple bytes). If multiple scaling bytes follow a parameter identifier the length of the data referenced by the parameter identifier is the summation of the content of the low nibbles in the scaling bytes. e.g. VIN is identified by a single byte parameter identifier and followed by two scaling bytes. The length is calculated up to 17 data bytes. The content of the two low nibbles may have any combination of values which add up to 17 data bytes. Note: For the ScalingByte with High Nibble encoded as formula or unit/format this value is $0.	**U**	**NROBOP**

Table 5.4.5.2 - Encoding of Low Nibble of ScalingByte

5.4.6 Scaling Byte Extention

The ***scalingByteExtention*** is one or more bytes following some of the the scalingBytes. The scalingByteExtention gives additional information to the scaling byte.

5.4.6.1 Formula Identifiers

A scalingByte with High Nibble encoded as formula shall be followed by scalingByteExtention bytes defining the formula. The scalingByteExtention consists of one byte formulaIdentifier and corresponding constants as defined in table 5.4.6.1.1.
The formulaIdentifier and the constants (C0, C1, C2,...) shall be placed in the scaling table in the following order:

formulaIdentifier
C0 high byte
C0 low byte
C1 high byte

C1 low byte
etc.

FormulaIdentifier (Hex)	Description	Cvt
00	y = C0 * x + C1	**U**
01	y = C0 * (x + C1)	**U**
02	y = C0 / (x + C1) + C2	**U**
03	y = x/C0 + C1	**U**
04	y = (x + C0) / C1	**U**
05	y = (x + C0) / C1 + C2	**U**
06	y = C0 * x	**U**
07	y = x / C0	**U**
08	y = x + C0	**U**
09	y = x * C0 / C1	**U**
0A - 7F	Reserved by document	**M**
80 - FF	Vehicle manufacturer specific	**U**

Table 5.4.6.1.1 - Encoding of formula identifiers

Formulas are defined using variables (y, x, ..) and constants (C0, C1, C2,...).
The variable y is the calculated value. The other variables, in consecutive order, are part of the data stream referenced by a parameter identifier.
Each constant is expressed as a two byte real number defined in table 6 (the two byte form is very compact and provides three digit accuracy), C = M * 10E.
Two byte real numbers contain a 12 bit signed mantissa (M) and a 4 bit signed exponent (E). The mantissa can hold values within the range -2047 to 2047, and the exponent can scale the number by 10-7 to 107.
The exponent is encoded in the high nibble of the high byte of the two byte real number. The mantissa is encoded in the low nibble of the high byte and in the low byte of the two byte real number.

High Byte								Low Byte							
High Nibble				Low Nibble				High Nibble				Low Nibble			
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Sign 0 = + 1 = –	Exponent			Sign 0 = + 1 = –	Mantissa										

Table 5.4.6.1.2 - Definition of two byte real number used in formulas

For more details see example in 5.4.6.3.

5.4.6.2 Unit and Format Identifiers

The following table contains unit and format identifiers defined by this document.

Id No. (Hex)	Name	Symbol	Description	Cvt
00	No unit, no prefix			U
01	meter	m	length	U
02	foot	ft	length	U
03	inch	in	length	U
04	yard	yd	length	U
05	mile (English)	mi	length	U
06	gram	g	mass	U
07	ton (metric)	t	mass	U
08	second	s	time	U
09	minute	min	time	U
0A	hour	h	time	U
0B	day	d	time	U
0C	year	y	time	U

Id No. (Hex)	Name	Symbol	Description	Cvt
0D	ampere	A	current	U
0E	volt	V	voltage	U
0F	coulomb	C	electric charge	U
10	ohm	W	resistance	U
11	farad	F	capacitance	U
12	henry	H	inductance	U
13	siemens	S	electric conductance	U
14	weber	Wb	magnetic flux	U
15	tesla	T	magnetic flux density	U
16	kelvin	K	thermodynamic temperature	U
17	Celsius	°C	thermodynamic temperature	U
18	Fahrenheit	F	thermodynamic temperature	U
19	candela	cd	luminous intensity	U
1A	radian	rad	plane angle	U
1B	degree	°	plane angle	U
1C	hertz	Hz	frequency	U
1D	joule	J	energy	U
1E	Newton	N	force	U
1F	kilopond	kp	force	U
20	pound force	lbf	force	U
21	watt	W	power	U
22	horse power (metric)	hk	power	U
23	horse power (UK and US)	hp	power	U
24	Pascal	Pa	pressure	U
25	bar	bar	pressure	U
26	atmosphere	atm	pressure	U
27	pound force per square inch	psi	pressure	U
28	becqerel	Bq	radioactivity	U
29	lumen	lm	light flux	U
2A	lux	lx	illuminance	U
2B	liter	l	volume	U
2C	gallon (British)	-	volume	U
2D	gallon (US liq)	-	volume	U
2E	cubic inch	cu in	volume	U
2F	meter per second	m/s	speed	U
30	kilometer per hour	km/h	speed	U
31	mile per hour	mph	speed	U
32	revolutions per second	rps	angular velocity	U
33	revolutions per minute	rpm	angular velocity	U
34	counts	-	-	U
35	percent	%	-	U
36	milligram per stroke	mg/stroke	mass per engine stroke	U
37	meter per square second	m/s^2	acceleration	U
38	Newton meter	Nm	moment (e.g. torsion moment)	U
39-3F	-	-	Reserved by document	M
40	exa (prefix)	E	10^{18}	U
41	peta (prefix)	P	10^{15}	U
42	tera (prefix)	T	10^{12}	U
43	giga (prefix)	G	10^{9}	U
44	mega (prefix)	M	10^{6}	U
45	kilo (prefix)	k	10^{3}	U
46	hecto (prefix)	h	10^{2}	U
47	deca (prefix)	da	10	U
48	deci (prefix)	d	10^{-1}	U

Id No. (Hex)	Name	Symbol	Description	Cvt
49	centi (prefix)	c	10^{-2}	U
4A	milli (prefix)	m	10^{-3}	U
4B	micro (prefix)	m	10^{-6}	U
4C	nano (prefix)	n	10^{-9}	U
4D	pico (prefix)	p	10^{-12}	U
4E	femto (prefix)	f	10^{-15}	U
4F	atto (prefix)	a	10^{-18}	U
50	Date1	-	Year-Month-Day	U
51	Date2	-	Day/Month/Year	U
52	Date3	-	Month/Day/Year	U
53	week	W	calendar week	U
54	Time1	-	UTC Hour/Minute/Second	U
55	Time2	-	Hour/Minute/Second	U
56	DateAndTime1	-	Second/Minute/Hour/Day/Month/Year	
57	DateAndTime2	-	Second/Minute/Hour/Day/Month/Year/ Local minute offset/Local hour offset	U
58-7F	-	-	Reserved by document	M
80-FF	-	-	Vehicle manufacturer specific	U

Table 5.4.6.2 - Encoding of unit and format identifiers

5.4.6.3 Example of formula and unit identifiers

This section shows how the formula and unit/format identifiers can be used for defining the format of a data variable in an ECU. The data variable used in this example is:

Vehicle Speed = 0,75 * x + 30 km/h

where 'x' is the actual data stored in the ECU data table and is identified by the local identifier $10.

Local Identifier Scaling Table	Hex
....	...
ScalingOffset { pointer position + $0B }	**0B**
vehicle speed local Id	10
ScalingByte#1 { unsigned numeric, 1 data byte in the data stream }	01
ScalingByte#2 { formula, 0 data bytes in the data stream }	90
ScalingByte#3 { formula Id, C0*x+C1 }	00
ScalingByte#4 { C0 high byte }	A0
ScalingByte#5 { C0 low byte }	4B
ScalingByte#6 { C1 high byte }	00
ScalingByte#7 { C1 low byte }	1E
ScalingByte#8 { unit, 0 data bytes in the data stream }	A0
ScalingByte#9 { unit Id, km/h }	30
ScalingOffset { pointer position + $03 }	**03**
....	...

6 Diagnostic Management functional unit

The services provided by this functional unit are described in table 6:

Service name	Description
startDiagnosticSession	The client requests to start a diagnostic session with a server(s).
stopDiagnosticSession	**This service is not part of SSF 14230-3 and shall therefore not be implemented!**
securityAccess	The client requests to unlock a secured server.
testerPresent	The client indicates to the server(s) that it is still present.
ecuReset	The client forces the server(s) to perform a reset.
readEcuIdentification	The client requests identification data from the server(s).

Table 6 - Diagnostic Management functional unit

6.1 StartDiagnosticSession service

6.1.1 Message description

The service startDiagnosticSession is used to enable different diagnostic sessions in the server. A diagnostic session enables a specific set of diagnostic services according to table 4.3.
A diagnostic session shall only be started if communication has been established between the client and the server. For more details on how to start communication refer to SSF 14230-2.

- After startCommunication shall the standardSession automatically be enabled in the server. The standardSession shall support at least the sevices that are marked as mandatory services in table 4.3 “Service Identifier value summary table”:
  "startCommunication service"
  "stopCommunication service"
  "testerPresent service"
  "readECUIdentification service"
  "readDiagnosticTroubleCodesByStatus service"
  “clearDiagnosticInformation service”
- If a diagnostic session, which is already running, is requested by the client the server shall send a positive response message.
- Whenever a new diagnostic session is requested by the client, the server shall first send a startDiagnosticSession positive response message before the new session becomes active in the server. If the server sends a negative response message with the startDiagnosticSession request service identifier the current session shall continue.
- There shall be only one session active at a time. A diagnostic session enables a specific set of KWP 2000 services which shall be defined by the vehicle manufacturer. A session can enable vehicle manufacturer specific services which are not part of this document.
- If a startDiagnosticSession with the diagnosticSession parameter set to PeriodicTransmissions in the request message is received by a server this shall enable ***Periodic Transmission*** in the server. This shall not change the set of diagnostic services or the security level made available by the previous session. To disable Periodic Transmissions the client must issue a startDiagnosticSession request with the parameter diagnosticSession = 81 (standardSession)(or a stopCommunication request). This will automatically return the server to the standardSession with default timing and default security level.
- The default timing parameters shall be active after a successful startDiagnosticSession with the diagnosticSession parameter set to standardSession in the request message if another session was previously active.
- Some diagnostic sessions in a server (ECU) may require a successful security access service in order to support/perform secured functions. In such case the following sequence of services shall be required:
  ① startDiagnosticSession(diagnosticSession) service
  ② securityAccess(...) service
  Before a positive response has been given to the securityAccess service request, the server (ECU) shall only make available the mandatory services according to table 4.3 in the new session.

NOTE ISO 14230-3 specifies a StopDiagnosticSession service. It is recommended not to use that service. Instead the StartDiagnosticSession service request with the parameter diagnosticSession = 81 (standardSession) shall be used. This will stop any other diagnostic session in the server and return the server to the standard diagnostic session in an identical manner as the result of a StopDiagnosticSession service call.

6.1.2 Message data bytes

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**startDiagnosticSession Request Service Id**	**M**	**10**	**STDS**
#2	diagnosticSession=[refer to table 6.1.3]	M	xx	**DCS_...**

Table 6.1.2.1 - StartDiagnosticSession Request Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**startDiagnosticSession Positive Response Service Id**	**M**	**50**	**STDSPR**
#2	diagnosticSession=[refer to table 6.1.3]	M	xx	**DCS_...**

Table 6.1.2.2 - StartDiagnosticSession Positive Response Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**negativeResponse Service Id**	**M**	**7F**	**NR**
#2	**startDiagnosticSession Request Service Id**	**M**	**10**	**STDS**
#3	responseCode=[KWP2000ResponseCode { section 4.4 }]	M	xx=[00-FF]	**RC_...**

Table 6.1.2.3 - StartDiagnosticSession Negative Response Message

6.1.3 Parameter definition

- The parameter ***diagnosticSession (DCS_)*** is used by the startDiagnosticSession service to select the specific behavior of the server(s). The following diagnostic sessions are specified in this document:

Hex	Description	Cvt	Mnemonic
00 - 80	**reservedByDocument** This range of values is reserved by this document for future definition.	**M**	**RBD**
81	**standardSession** The standardSession enables all Keyword Protocol 2000 services specified in **Table 4.3 column 4 "SS"**. The standardSession shall be active after the initialisation has been successfully completed between client (tester) and server (ECU). No startDiagnosticSession request with the parameter diagnosticSession set to "standardSession" is required after initialisation to start this diagnostic session. The services enabled in this diagnostic session provide most of all functions required in a repair shop environment. The standardSession also enables all SAE J1979 related services (in SAE documents services are called "test modes") if the server (ECU) is OBD II compliant. The data content of SAE J1979 specified messages shall not be changed or modified or used differently than specified in the SAE J1979 Recommended Practice. If any other session than the standardSession has been active in the server and the standardSession is once again started, then the following implementation rules shall be followed: • If the server has sent a StartDiagnosticSession positive response message it shall have stopped the current diagnostic session, that is, perform the necessary action to return to a state in which it is able to restart the standard diagnostic session. Restoring the normal operating conditions of the server may include the reset of all the actuators controlled if they have been activated by the client during the diagnostic session being stopped and resuming all normal algorithms of the server. • If the server has sent a StartDiagnosticSession positive response message it shall have re-locked the server if it was unlocked by the client during the diagnostic session. • If the server sends a negative response message with the StartDiagnosticSession request service identifier the active session shall be continued. • If the server has sent a stopDiagnosticSession positive response message it shall have disabled Periodic Transmissions if Periodic Transmissions was enabled by the client in the previously active diagnostic session. • If a stopDiagnosticSession has been requested by the client and the StandardSession is already running the server shall send a positive response message and immediately reset all timing parameters to the default values The new timing becomes active after the successful completion of the positive response message of this service.	**U**	**SS**
82	**periodicTransmissions** This diagnostic session supports all Keyword Protocol 2000 services which were supported in the previous diagnostic session. In this session the server (ECU) shall send the response message periodically with current data (updated data) based on the client (tester) request message. Periodic transmissions communication is specified in SSF 14230-2.	**U**	**PT**
83 - 84	**reservedByDocument** This range of values is reserved by this document for future definition.	**U**	
85	**programmingSession** The programmingSession enables all Keyword Protocol 2000 services specified in **Table 4.3 column 6 "PS"**. These services support the memory programming of a server (ECU).	**U**	**PS**

Hex	Description	Cvt	Mnemonic
86	**developmentSession** The developmentSession enables all Keyword Protocol 2000 services specified in **Table 4.3 column 7 "DS"**. These services support the development of a server (ECU).	**U**	**DS**
87	**adjustmentSession** The adjustmentSession enables all Keyword Protocol 2000 services specified in **Table 4.3 column 5 "AS"**. These services additionally support adjustment of inputs and outputs of the server (ECU).	**U**	**AS**
88	**reservedByDocument** This value is reserved by this document for future definition.	**U**	
89 - F9	**vehicleManufacturerSpecificSession** This range of values is reserved for vehicle manufacturer specific use.	**U**	**VMSS**
FA - FE	**systemSupplierSpecificSession** This range of values is reserved for system supplier specific use.	**U**	**SSSS**
FF	**reservedByDocument** This value is reserved by this document for future definition.	**M**	**RBD**

Table 6.1.3 - Definition of diagnosticSession Values

6.1.4 Message flow example

See message flow examples of a Physically Addressed Service in section 5.3.1.1 and a Functionally Addressed Service in section 5.3.2.1.

6.1.5 Implementation example of startDiagnosticSession

6.1.5.1 Message flow conditions to enable "ProgrammingSession"

Conditions for this example is to enable the programmingSession ($85) in the server (ECU).

6.1.5.2 Message flow

STEP#1 startDiagnosticSession(DS_PS)

time	Client (tester) Request Message	Hex
P3	**startDiagnosticSession.ReqSId**[diagnosticSession = programmingSession]	**10** 85

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**startDiagnosticSession.PosRspSId**[diagnosticSession = ProgrammingSession]	**50** 85	**negativeResponse Service Identifier** **startDiagnosticSession.ReqSId[** responseCode { refer to section 4.4 }]	**7F** **10** xx

6.2 StopDiagnosticSession service

This service is not part of SSF 14230-3 and shall therefore not be implemented!

The purpose of this service is to disable the current diagnostic session in the server (ECU). It is not implemented since the same result can be obtained with a StartDiagnosticSession service request with the parameter diagnosticSession = 81 (standardSession). This will stop any diagnostic session in the server and return the server to the standard diagnostic session in an identical manner as the result of a StopDiagnosticSession service call.

6.3 SecurityAccess service

6.3.1 Message description

This service is intended to be used to implement data link security measures defined by the vehicle manufacturer or the system supplier, e.g. the method for Security Access procedure defined in SAE J2186.

The security system shall not prevent normal diagnostic or vehicle communications between the client and the servers. Proper "unlocking" of the server is a prerequisite to the client's ability to perform some of the more critical functions such as reading specific memory locations within the server, downloading information to specific locations, or downloading routines for execution by the controller. In other words, the only access to the server permitted while in a "locked" mode is through the server specific software. This permits the server specific software to protect itself from unauthorized intrusion.
If the Server cannot execute the service, or if access is denied, it shall issue a SecurityAccess response primitive with the Negative Response parameters.
The system designer has the flexibility to specify what the Server may actually do if errors occur during the service.

The procedure defined by this service includes the following steps:

- The client shall request the server to "unlock" itself by sending the service securityAccess request #1. The server shall respond by sending a "seed" using the service securityAccess positive response #1. The client shall respond by returning a "key" number back to the server using the service securityAccess request #2 (the algorithm for calculating the Key number shall be defined by the vehicle manufacturer or the system supplier). The server shall compare this "key" to one internally stored. If the two numbers match, then the server shall enable ("unlock") the client's access to specific KWP 2000 services and indicate that with the service securityAccess positive response #2.
- If a device supports security, but is already unlocked when a securityAccess request #1 is received, that server shall respond with a securityAccess positive response #1 service with a seed of "$00 00". A client shall use this method to determine if a server is locked by checking for a non-zero seed.

Some servers could support multiple levels of security, either for different functions controlled by the server, or to allow different capabilities to be exercised. These additional levels of security can be accessed by using the service securityAccess requests #1 and #2 with an accessMode value greater than the default value. The accessMode parameter for "requestSeed" shall always be an odd number, and the accessMode parameter for "sendKey" shall be the next even number.

In order to further increase the level of security the system designer may also optionally specify delays to be introduced between repeated usage of the securityAccess service. The following is an example of how this could be implemented:

- If upon two (2) attempts of a service securityAccess request #2 by the client, where the two keys do not match, then the server shall insert a ten (10) second time delay before allowing further attempts.
- The ten (10) second time delay shall also be required before the server responds to a service securityAccess request #1 from the client after server power-on.

Some diagnostic sessions in a server (ECU) require a successful security access service in order to support/perform secured functions. In such case the following sequence of services shall be required:

① startDiagnosticSession(diagnosticSession) service
② securityAccess(...) service

Before a positive response has been given to the securityAccess service request, the server (ECU) shall only make available the mandatory services according to table 4.3 in the new session.

6.3.2 Message data bytes

6.3.2.1 Message sequence step #1

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**securityAccess Request Service Id**	**M**	**27**	**SA**
#2	accessMode=[requestSeed ($01 default) vehicleManufacturerSpecific, systemSupplierSpecific]	M	xx=[odd no. 01 - 7F 81-F9, FB-FD]	**AM_...**

Table 6.3.2.1 - securityAccess Request Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**securityAccess Positive Response Service Id**	**M**	**67**	**SAPR**
#2	accessMode=[requestSeed ($01 default), vehicleManufacturerSpecific, systemSupplierSpecific]	M	xx=[odd no. 01 - 7F 81-F9, FB-FD]	**AM_...**
#3 : #n	seed#1 (High Byte) : seed#m (Low Byte)	C **:** C	xx : xx	**SEED**

Table 6.3.2.2 - securityAccess Positive Response Message

C = condition: accessMode = "requestSeed".

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**negativeResponse Service Id**	**M**	**7F**	**NR**
#2	**securityAccess Request Service Id**	**M**	**27**	**SA**
#3	responseCode=[KWP2000ResponseCode { section 4.4 }]	M	xx=[00-FF]	**RC_...**

Table 6.3.2.3 - SecurityAccess Negative Response Message

6.3.2.2 Message sequence step #2

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**securityAccess Request Service Id**	**M**	**27**	**SA**
#2	accessMode=[sendKey ($02 default) vehicleManufacturerSpecific, systemSupplierSpecific]	M	xx=[even no. 02 - 80 82-FA, FC-FE]	**AM_...**
#3 **:** #n	key#1 (High Byte) **:** key#m (Low Byte)	C **:** C	xx **:** xx	**KEY**

Table 6.3.2.4 - SecurityAccess Request Message

C = condition: accessMode = "sendKey".

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**securityAccess Positive Response Service Id**	**M**	**67**	**SAPR**
#2	accessMode=[sendKey ($02 default) vehicleManufacturerSpecific, systemSupplierSpecific]	M	xx=[even no. 02 - 80 82-FA, FC-FE]	**AM_...**
#3	securityAccessStatus=[securityAccessAllowed]	M	34	**SAA**

Table 6.3.2.5 - SecurityAccess Positive Response Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**negativeResponse Service Id**	**M**	**7F**	**NR**
#2	**securityAccess Request Service Id**	**M**	**27**	**SA**
#3	responseCode=[KWP2000ResponseCode { section 4.4 }]	M	xx=[00-FF]	**RC_...**

Table 6.3.2.6 - SecurityAccess Negative Response Message

6.3.3 Parameter definition

- The parameter ***accessMode (AM_)*** is used in the securityAccess service. It indicates to the server the step in progress for this service, the level of security the client wants to access and the format of seed and key. Values are defined in the table below for requestSeed and sendKey.

Hex	Description	Cvt	Mnemonic
00	**reservedByDocument** This value is reserved by this document.	**M**	**RBD**
01	**requestSeed** Request Seed with the default level of security defined by the vehicle manufacturer.	**M**	**RSD**
02	**sendKey** Send Key with the default level of security defined by the vehicle manufacturer.	**M**	**SK**
03, 05, 07 - 7F	**requestSeed** Request Seed with different levels of security defined by the vehicle manufacturer.	**U**	**RSD**
04, 06, 08 - 80	**sendKey** Send Key with different levels of security defined by the vehicle manufacturer.	**U**	**SK**
81 - FA	**vehicleManufacturerSpecific** This range of values is reserved for vehicle manufacturer specific use.	**U**	**VMS**
FB - FE	**systemSupplierSpecific** This range of values is reserved for system supplier specific use.	**U**	**SSS**
FF	**reservedByDocument** This value is reserved by this document.	**M**	**RBD**

Table 6.3.1.1 - Definition of accessMode values

- The values of the parameter ***seed*** are not defined in this document except for the values **"$00 ... 00" (all numbers are $00)** which indicates to the client (tester) that the server (ECU) is not locked and **"$FF ... FF" (all numbers are $FF)** which shall not be used because this value may occur if the server's (ECU's) memory has been erased.
- The values of the parameter ***key*** are not defined in this document.
- The parameter ***securityAccessStatus (SAS_)*** may be used to receive the status of the server security system. The value is defined in the table below.

Hex	Description	Mnemonic
34	**securityAccessAllowed**	**SAA**

Table 6.3.3.2 - Definition of securityAccessStatus value

6.3.4 Message flow example

time	client (Tester)	server (ECU)

P3 P2	securityAccess.Request#1[...]	 securityAccess.PositiveResponse#1[...]
P3 P2	securityAccess.Request#2[...]	 securityAccess.PositiveResponse#2[...]

Table 6.3.4.1 - Message flow example of securityAccess services

Note: In case of a securityAccess negative response it is the vehicle manufacturer's responsibility to decide how to proceed.

6.3.5 Implementation example of securityAccess

6.3.5.1 Message flow conditions for securityAccess

This section specifies the conditions to be fulfilled to successfully unlock the server (ECU) if in a "locked“ state
requestSeed = $01; sendKey = $02; seed = $3675; key = $C98B {e.g. 2's complement of seed value }

6.3.5.2 Message flow

6.3.5.2.1 Message flow if server (ECU) is in a "locked" state (seed ≠ $0000)

STEP#1 securityAccess(AM_RSD)

time	Client (tester) Request Message	Hex
P3	**securityAccess.ReqSId**[	**27**
	accessMode = requestSeed]	01

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**securityAccess.PosRspSId**[	**67**	**negativeResponse Service Identifier**	**7F**
	accessMode = requestSeed	01	**securityAccess.ReqSId**[	**27**
	seed (High Byte)	36	responseCode { refer to section 4.4 }]	xx
	seed (Low Byte)]	75		

STEP#2 securityAccess(AM_SK, KEY)

time	Client (tester) Request Message	Hex
P3	**securityAccess.ReqSId**[	**27**
	accessMode = sendKey	02
	key (High Byte) { 2's complement of seed }	C9
	key (Low Byte) { 2's complement of seed }]	8B

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**securityAccess.PosRspSId**[	**67**	**negativeResponse Service Identifier**	**7F**
	accessMode = sendKey	02	**securityAccess.ReqSId**[	**27**
	securityAccessAllowed]	34	responseCode { refer to section 4.4 }]	xx

6.3.5.2.2 Message flow if server (ECU) is in an "unlocked" state (seed = $0000)

STEP#1 securityAccess(AM_RSD)

time	Client (tester) Request Message	Hex
P3	**securityAccess.ReqSId**[	**27**
	accessMode = requestSeed]	01

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**securityAccess.PosRspSId**[	**67**	**negativeResponse Service Identifier**	**7F**
	accessMode = requestSeed	01	**securityAccess.ReqSId**[	**27**
	seed (High Byte)	00	responseCode { refer to section 4.4 }]	xx
	seed (Low Byte)]	00		

6.4 TesterPresent service

6.4.1 Message description

This service shall be used to indicate to a server that the client is present. This service is required in the absence of other KWP 2000 services to prevent servers from automatically returning to normal operation and stop communication.

The following rules shall be followed:

- The presence of this service shall keep communication active between client and server.
- The presence of this service shall indicate that the system should remain in a diagnostic session of operation.
- The server shall always send a response to the request message.

6.4.2 Message Data Bytes

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**testerPresent Request Service Id**	**M**	**3E**	**TP**

Table 6.4.2.1: testerPresent Request Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**testerPresent Positive Response Service Id**	**M**	**7E**	**TPPR**

Table 6.4.2.2: testerPresent Positive Response Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**negativeResponse Service Id**	**M**	**7F**	**NR**
#2	**testerPresent Request Service Id**	**M**	**3E**	**TP**
#3	responseCode=[KWP2000ResponseCode { section 4.4 }]	M	xx=[00-FF]	**RC_...**

Table 6.4.2.3 - testerPresent Negative Response Message

6.4.3 Parameter definition

No parameters are defined for this service.

6.4.4 Message flow example

time	client (Tester)	server (ECU)
P3	testerPresent.Request	
P2		testerPresent.PositiveResponse

Table 6.4.4 - Message flow example of testerPresent services

6.4.5 Implementation example of testerPresent

6.4.5.1 Message flow conditions for testerPresent

This section specifies the conditions to be fulfilled at the client (tester) before it sends a testerPresent request message to the server (ECU).
Condition client (tester): if (timer(P3timeout) == Hex(P3max) - Hex(01)) then testerPresent.req;

6.4.5.2 Message flow

STEP#1 testerPresent()

time	Client (tester) Request Message	Hex
P3	**testerPresent.ReqSId**	**3E**

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**testerPresent.PosRspSId**	**7E**	**negativeResponse Service Identifier**	**7F**
			testerPresent.ReqSId[	**3E**
			responseCode { refer to section 4.4 }]	xx

6.5 EcuReset service

6.5.1 Message description

This service requests the server to effectively perform an ECU reset based on the content of the resetMode parameter value.
The positive response message shall be sent before the resetMode is executed.

6.5.2 Message data bytes

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**ecuReset Request Service Id**	**M**	**11**	**ER**
#2	resetMode= [refer to table 6.5.3.1]	M	xx	**RM_...**

Table 6.5.2.1 - ecuReset Request Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**ecuReset Positive Response Service Id**	**M**	**51**	**ERPR**
#2	resetStatus#1 { refer to table 6.5.3.2 }	U	xx	**RS_...**
:	:	:	:	
#n	resetStatus#m { refer to table 6.5.3.2 }	U	xx	

Table 6.5.2.2 - ecuReset Positive Response Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**negativeResponse Service Id**	**M**	**7F**	**NR**
#2	**ecuReset Request Service Id**	**M**	**11**	**ER**
#3	responseCode=[KWP2000ResponseCode { section 4.4 }]	M	xx=[00-FF]	**RC_...**

Table 6.5.2.3 - ecuReset Negative Response Message

6.5.3 Parameter definition

- The parameter ***resetMode (RM_)*** is used by the ecuReset request message to describe how the server has to perform the reset. Values are defined in the table below:

Hex	Description	Cvt	Mnemonic
00	**reservedByDocument** This value is reserved by this document.	**M**	**RBD**
01	**powerOn** This value identifies the power-on reset cycle which shall simulate the reset cycle performed when power is connected to an un-powered server (ECU). When the server (ECU) performs the reset the client (tester) shall be prepared to re-establish communication.	**M**	**PO**
02	**reservedByDocument** This value is reserved by this document.	**M**	**RBD**
03	**keyOn** This value identifies the power-on reset cycle which shall simulate the reset cycle performed at the starter key OFF to DRIVING POSITION switching (ignition OFF/ON cycle). When the server (ECU) performs the reset the client (tester) shall be prepared to re-establish communication.	**U**	**KO**
04 - 7F	**reservedByDocument** This range of values is reserved by this document for future definition.	**M**	**RBD**
80	**sendResetStatus** This value shall indicate to the server (ECU) that it shall send the resetStatus parameter in the positive response message.	**U**	**RS**
81 - F9	**vehicleManufacturerSpecific** This range of values is reserved for vehicle manufacturer specific use.	**U**	**VMS**
FA - FE	**systemSupplierSpecific** This range of values is reserved for system supplier specific use.	**U**	**SSS**
FF	**reservedByDocument** This value is reserved by this document.	**M**	**RBD**

Table 6.5.3.1 - Definition of resetMode values

- The parameter ***resetStatus (RS_)*** is used by the ecuReset positive response message to provide information about the status of the reset(s). Format and length of this parameter are vehicle manufacturer specific.

  A possible implementation would be to report the number of resets performed by the server(s) after the last full power down / power up cycle (key off/on).

Hex	Description	Mnemonic
xx ... xx	**resetStatus** This parameter shall report resetStatus information. It is the vehicle manufacturer's responsibility to define the type and format of the data value(s).	**RS_...**

Table 6.5.3.2 - Definition of resetStatus value

6.5.4 Message flow example

See message flow examples of a Physically Addressed Service in section 5.3.1.1 and a Functionally Addressed Service in section 5.3.2.1.

6.5.5 Implementation example of ecuReset

6.5.5.1 Message flow conditions ecuReset

This section specifies the conditions to be fulfilled to successfully perform an ecuReset service in the server (ECU). Condition of server (ECU): ignition = on, system shall not be in an operational mode (e.g. if the system is an engine management, engine shall be off);

Note: The server (ECU) shall send an ecuReset positive response message before the server (ECU) performs the resetMode. The server (ECU) will not send any further response messages based on request messages which may be sent by the client (tester). The client (tester) shall re-establish communication with the server (ECU) by sending a new initialisation sequence after a timeout (ECU power on self test) to be specified by the system supplier. If the resetMode is set to *powerOn* the client (tester) shall behave the same as having received a stopCommunication positive response message!

6.5.5.2 Message flow

STEP#1 ecuReset(RM_PO)

time	Client (tester) Request Message	Hex
P3	**ecuReset.ReqSId**[	**11**
	resetMode = powerOn]	01

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**ecuReset.PosRspSId**	**51**	**negativeResponse Service Identifier**	**7F**
			ecuReset.ReqSId[	**11**
			responseCode { refer to section 4.4 }]	xx

6.6 ReadEcuIdentification service

6.6.1 Message description

The readECUIdentification request message requests identification data from the server (ECU). The type of identification data requested by the client (tester) shall be identified by the identificationOption parameter. The server (ECU) sends an identification data record included in the readEcuIdentification positive response message. The format of the identification data shall be according to a Data Scaling table as specified in section 5.4. The first parameter of the positive response message shall always be the repetition of the identificationOption of the request message. If the identificationOption parameter is set to ECUIdentificationDataTable or ECUIdentificationScalingTable then the positive response message may include multiple type of data based on vehicle manufacturer definitions. The identification data included in the positive response message shall be of the types specified in table - 6.6.3.

The service ***writeDataByCommonIdentifier***, defined in section 7.6, can be used to write identificationRecordValues in the servers memory.

6.6.2 Message data bytes

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**readEcuIdentification Request Service Id**	**M**	**1A**	**REI**
#2	identificationOption=[refer to table 6.6.3]	M	80-9F	**IO_**

Table 6.6.2.1 - readEcuIdentification Request Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**readEcuIdentification Positive Response Service Id**	**M**	**5A**	**REIPR**
#2	identificationOption=[refer to table 6.6.3]	M	80-9F	**IO_...**
#3 : #n	identificationRecordValue#1 : identificationRecordValue #m	M : U	xx : xx	**IRV_...** : **IRV_...**

Table 6.6.2.2 - ReadEcuIdentification Positive Response Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**negativeResponse Service Id**	**M**	**7F**	**NR**
#2	**readEcuIdentification Request Service Id**	**M**	**1A**	**REI**
#3	responseCode=[KWP2000ResponseCode { section 4.4 }]	M	xx=[00-FF]	**RC_...**

Table 6.6.2.3 - ReadEcuIdentification Negative Response Mssage

6.6.3 Parameter definition

- The parameter ***identificationOption (IO_)*** is used by the readEcuIdentification request message to describe the kind of identification data requested by the client (tester). Either the parameter ECUIdentificationDataTable ($80) or at least one of the parameters in the range of $82 -$9F shall be implemented in the server (ECU) for identification purposes. Values are defined in the table below. The parameter *identificationOption* is included as part of the *recordCommonIdentifer* (see table 7.2.3) range of values.
  The identificationOption table consists of six (6) columns and multiple lines.
  - The **1st column (Hex)** includes the "Hex Value" assigned to the identificationOption specified in the 2nd column.
  - The **2nd column (Description)** specifies the identificationOption.
  - The **3rd column (Source of Identification)** includes the source of the identification data for this identificationOption.
  - The **4th column (Programmable)** specifies if the identificationOption is programmable.
  - The **5th column (Cvt)** is the convention column which specifies whether the identificationOption is "M" (mandatory) or "U" (User Optional).
  - The **6th column (Mnemonic)** specifies the mnemonic of this identificationOption.

Hex	Description	Source of Id.	Prog.	Cvt	Mnemonic

80	**ECUIdentificationDataTable** This value shall cause the server (ECU) to report the ECU identification data table in the readECUIdentification positive response message. The information contained in the ECUIdentificationTable shall be identified by the identificationOption numbers '$82 - $BF'. Data (paramter values) shall be sent in the same order as it is reported in the ECU identification scaling table.	system supplier	**N/A**	**U**	**ECUIDT**
81	**ECUIdentificationScalingTable** This value shall cause the server (ECU) to report the ECU identification scaling table in the readECUIdentification positive response message. The data content of the ECUIdentificationScalingTable depends on the identification data (length and format) being implemented in the ECU Identification Data Table which shall be reported if the identificationOption parameter is set to ECUIdentificationDataTable. The scaling data shall be of the format specified in section 5.4 Data Scaling.	system supplier	**YES**	**U**	**ECUIST**
82 - 85	**reservedByDocument** This range of values is reserved by this document for future definition.			**U**	**RBD**
86	**vehicleManufacturerSpecific** This range of values is reserved for vehicle manufacturer specific use.	vehicle manuf.		**U**	**VMS_**
87	**vehicleManufacturerSparePartNumber** This value shall cause the server (ECU) to report the vehicle manufacturer ECU spare part number in the readECUIdentification positive response message. The data content shall be ECU specific and be the responsibility of the vehicle manufacturer.	vehicle manuf.	**YES**	**U**	**VMSPN**
88	**vehicleManufacturerECUSoftwareNumber** This value shall cause the server (ECU) to report the vehicle manufacturer ECU software number in the readECUIdentification positive response message. The data content shall be ECU specific and be the responsibility of the vehicle manufacturer.	vehicle manuf.	**YES**	**U**	**VMECUSN**
89	**vehicleManufacturerECUSoftwareVersionNumber** This value shall cause the server (ECU) to report the vehicle manufacturer ECU software version number in the readECUIdentification positive response message. The data content shall be ECU specific and be the responsibility of the vehicle manufacturer.	vehicle manuf.	**YES**	**U**	**VMECUSVN**
8A	**systemSupplier** This value shall cause the server (ECU) to report the system supplier's name and address information in the readECUIdentification positive response message.	system supplier	**NO**	**U**	**SS**
8B	**ECUManufacturingDate** This value shall cause the server (ECU) to report the ECU manufacturing date in the readECUIdentification positive response message.	system supplier	**NO**	**U**	**ECUMD**
8C	**ECUSerialNumber** This value shall cause the server (ECU) to report the ECU serial number in the readECUIdentification positive response message.	system supplier	**NO**	**U**	**ECUSN**
8D - 8F	**systemSupplierSpecific** This range of values is reserved for system supplier specific use.	system supplier	**NO**	**U**	**SSS_**
90	**VIN (Vehicle Identification Number)** This value shall cause the server (ECU) to report the VIN in the readECUIdentification positive response message. The data content shall be specified by the vehicle manufacturer. If the VIN is not programmed in the ECU's memory those locations shall be padded with either '$00' or '$FF'.	vehicle manuf.	**YES**	**U**	**VIN**
91	**vehicleManufacturerECUHardwareNumber** This value shall cause the server (ECU) to report the vehicle manufacturer ECU hardware number (e.g. Part Number of ECU) in the readECUIdentification positive response message. The data content shall be ECU specific and be the responsibility of the vehicle manufacturer.	vehicle manuf.	**YES**	**U**	**VMECUHN**

92	**systemSupplierECUHardwareNumber** This value shall cause the server (ECU) to report the system supplier ECU hardware number in the readECUIdentification positive response message. The data content shall be ECU specific and be the responsibility of the system supplier.	system supplier	**NO**	**M**	**SSECUHN**
93	**systemSupplierECUHardwareVersionNumber** This value shall cause the server (ECU) to report the system supplier ECU hardware version number in the readECUIdentification positive response message. The data content shall be ECU specific and be the responsibility of the system supplier.	system supplier	**NO**	**U**	**SSECUHVN**
94	**systemSupplierECUSoftwareNumber** This value shall cause the server (ECU) to report the system supplier ECU software number in the readECUIdentification positive response message. The data content shall be ECU specific and be the responsibility of the system supplier.	system supplier	**NO**	**M**	**SSECUSN**
95	**systemSupplierECUSoftwareVersionNumber** This value shall cause the server (ECU) to report the system supplier ECU software version number in the readECUIdentification positive response message. The data content shall be ECU specific and be the responsibility of the system supplier.	system supplier	**NO**	**U**	**SSECUSVN**
96	**exhaustRegulationOrTypeApprovalNumber** This value shall cause the server (ECU) to report the exhaust regulation or type approval number (valid for those systems which require type approval) in the readECUIdentification positive response message. The data content shall be ECU specific and be the responsibility of the vehicle manufacturer.	vehicle manuf.	**NO**	**U**	**EROTAN**
97	**systemNameOrEngineType** This value shall cause the server (ECU) to report the system name and/or engine type in the readECUIdentification positive response message. The data content shall be ECU specific and be the responsibility of the vehicle manufacturer.	vehicle manuf.	**YES**	**U**	**SNOET**
98	**repairShopCodeOrTesterSerialNumber** This value shall cause the server (ECU) either to report the repair shop code or to report the tester's serial number if the ECU's memory has been previously re-programmed by the service tester.	repair shop	**YES**	**U**	**RSCOTSN**
99	**programmingDate** This value shall cause the server (ECU) to report the programming date when the software/parameter(s) was programmed into the server (ECU). Format and scaling shall be one of the following: unsigned numeric, ASCII, BCD. The sequence of the date in the record shall be: Year, Month, Day.	prog. equipm.	**YES**	**U**	**PD**
9A	**calibrationRepairShopCodeOrCalibrationEquipmentSerialNumber** This value shall cause the server (ECU) either to report the repair shop code or to report the tester's serial number if the ECU has been previously re-calibrated by the service tester.	repair shop	**YES**	**U**	**CRSCOCESN**
9B	**calibrationDate** This value shall cause the server (ECU) to report the date when the server (ECU) was calibrated. Format and scaling shall be one of the following: unsigned numeric, ASCII, BCD. The sequence of the date in the record shall be: Year, Month, Day.	prog. equipm.	**YES**	**U**	**CD**
9C	**calibrationEquipmentSoftwareNumber** This value shall cause the server (ECU) to report the calibration equipment software number.	prog. equipm.	**YES**	**U**	**CESN**
9D	**ECUInstallationDate** This value shall cause the server (ECU) to report the date when the server (ECU) was installed in the vehicle. Format and scaling shall be one of the following: unsigned numeric, ASCII, BCD. The sequence of the date in the record shall be: Year, Month, Day.	vehicle manuf.	**YES**	**U**	**ECUID**
9E - AF	**vehicleManufacturerSpecific** This range of values is reserved for vehicle manufacturer specific use.	vehicle manuf.		**U**	**VMS_**
B0 - BF	**systemSupplierSpecific** This range of values is reserved for system supplier specific use.	system supplier		**U**	**SSS**

Table 6.6.3: Definition of identificationOption values

N/A = Not Applicable

- The parameter ***identificationRecordValue (IRV_)*** is used by the readEcuIdentification positive response message to provide the identification data and/or scaling data to the client (tester). The content of the identification record is not defined in this document and is vehicle manufacturer specific.

6.6.4 Message flow example

See message flow examples of a Physically Addressed Service in section 5.3.1.1 and a Functionally Addressed Service in section 5.3.2.1.

6.6.5 Implementation example of readECUIdentification

6.6.5.1 Message flow conditions for readECUIdentification

This section specifies the conditions to be fulfilled to perform a readECUIdentification service. The client (tester) may request identification data at any time independent of the status of the server (ECU).
The table below lists an example of ECU identification data.

- The 1st column (ECU Identification Option Description) describes the ECU identification option.
- The 2nd column (ECU Identification Data) includes the ECU identification data as they shall be displayed in the client (tester).

ECU Identification Option Description	**ECU Identification Data (as displayed)**
VIN (Vehicle Identification Number)	W0L000043MB541326
vehicleManufacturerHardwareNumber	90254861 GD
systemSupplierECUHardwareNumber	10433
systemSupplierECUSoftwareNumber	uP1 33456 uP2 53129
systemSupplierECUSoftwareVersionNr	uP1 01 uP2 03
exhaustRegulationOrTypeApprovalNumber	B94001
systemNameOrEngineType	X20XEV
repairShopCodeOrTesterSerialNumber	6359984
programmingDate (YYYYMMDD)	19940911

Table 6.6.5.1 - Example of ECU identification data

- Message flow **STEP#1** shows a readECUIdentification request message which is sent to the server (ECU) with the identificationOption set to ECUIdentificationScalingTable. The server (ECU) shall send a positive response message if it supports the identificationOption. The identificationOption is specified in section 6.6.3.
- Message flow **STEP#2** shows a readECUIdentification request message which is sent to the server (ECU) with the identificationOption set to ECUIdentificationDataTable. The server (ECU) shall send a positive response message if it supports the identificationOption.
- Message flow **STEP#3** shows a readECUIdentification request message which is sent to the server (ECU) with the identificationOption set to VIN. The server (ECU) shall send a positive response message if it supports the identificationOption.

6.6.5.2 Message flow

STEP#1 readECUIdentification(IO_ECUIST)

time	Client (tester) Request Message	Hex
P3	**readECUIdentification.ReqSId**[	**1A**
	identificationOption = ECUIdentificationScalingTable]	81

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**readECUIdentification.PosRspSId**[	**5A**	**negativeResponse Service Identifier**	**7F**
	identificationOption =		**readECUIdentification.ReqSId**[	**1A**
	ECUIdentificationScalingTable	81	responseCode { refer to section 4.4 }]	xx
	ScalingOffset { pointer position + $04 }	**04**		
	VIN	90		
	ScalingByte#1 { ASCII, 15 data bytes }	6F		
	ScalingByte#2 { ASCII, 2 data bytes }	62		
	ScalingOffset { pointer position + $03 }	**03**		
	vehicleManufacturerHardwareNumber	91		
	ScalingByte#1 { ASCII, 11 data bytes }	6B		
	ScalingOffset { pointer position + $03 }	**03**		
	systemSupplierECUHardwareNumber	92		
	ScalingByte#1 { unsigned numeric, 2 data bytes }	02		
	ScalingOffset { pointer position + $06 }	**06**		
	systemSupplierECUSoftwareNumber	94		
	ScalingByte#1 { ASCII, 4 data bytes } { µP 1 }	64		
	ScalingByte#2 { unsigned numeric, 2 data bytes }	02		
	ScalingByte#3 { ASCII, 6 data bytes } { µP 2 }	66		
	ScalingByte#4 { unsigned numeric, 2 data bytes }	02		
	ScalingOffset { pointer position + $03 }	**03**		
	systemSupplierECUSoftwareVersionNumber	95		
	ScalingByte#1 { ASCII, 14 data bytes }	6E		
	ScalingOffset { pointer position + $03 }	**03**		
	exhaustRegulationOrTypeApprovalNumber	96		
	ScalingByte#1 { ASCII, 6 data bytes }	66		
	ScalingOffset { pointer position + $03 }	**03**		
	SystemNameOrEngineType	97		
	ScalingByte#1 { ASCII, 6 data bytes }	66		
	ScalingOffset { pointer position + $03 }	**03**		
	repairShopCodeOrTesterSerialNumber	98		
	ScalingByte#1 { unsigned numeric, 3 data bytes }	03		
	ScalingOffset { pointer position + $03 }	**03**		
	ProgrammingDate (YYYYMMDD)	99		
	ScalingByte#1 { BCD, 4 data bytes }	44		
	ScalingOffset { End of Table }]	**FF**		

STEP#2 readECUIdentification(IO_ECUIDT)

time	Client (tester) Request Message	Hex
P3	**readECUIdentification.ReqSId**[	**1A**
	identificationOption = ECUIdentificationDataTable]	80

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**readECUIdentification.PosRspSId**[	**5A**	**negativeResponse Service Identifier**	**7F**
	identificationOption = ECUIdentificationDataTable	80	**readECUIdentification.ReqSId**[	**1A**
	VIN { W }	57	responseCode { refer to section 4.4 }]	xx
	VIN { 0 }	30		
	VIN { L }	4C		
	VIN { 0 }	30		
	VIN { 0 }	30		
	VIN { 0 }	30		
	VIN { 0 }	30		
	VIN { 4 }	34		
	VIN { 3 }	33		
	VIN { M }	4D		
	VIN { B }	42		
	VIN { 5 }	35		
	VIN { 4 }	34		
	VIN { 1 }	31		
	VIN { 3 }	33		
	VIN { 2 }	32		
	VIN { 6 }	36		
	Vehicle Manufacturer Hardware Number { 9 }	39		
	Vehicle Manufacturer Hardware Number { 0 }	30		
	Vehicle Manufacturer Hardware Number { 2 }	32		
	Vehicle Manufacturer Hardware Number { 5 }	35		
	Vehicle Manufacturer Hardware Number { 4 }	34		
	Vehicle Manufacturer Hardware Number { 8 }	38		
	Vehicle Manufacturer Hardware Number { 6 }	36		
	Vehicle Manufacturer Hardware Number { 1 }	31		
	Vehicle Manufacturer Hardware Number { }	20		
	Vehicle Manufacturer Hardware Number { G }	47		
	Vehicle Manufacturer Hardware Number { D }	44		
	System Supplier ECU Hardware Number { 10433 }	28		
	System Supplier ECU Hardware Number	C1		
	System Supplier ECU S/W Number µP 1 { U }	55		
	System Supplier ECU S/W Number µP 1 { P }	50		
	System Supplier ECU S/W Number µP 1 { 1 }	31		
	System Supplier ECU S/W Number µP 1 { }	20		
	System Supplier ECU S/W Number µP 1 { 33456 }	82		
	System Supplier ECU S/W Number µP 1	B0		
	System Supplier ECU S/W Number µP 2 { }	20		
	System Supplier ECU S/W Number µP 2 { }	20		
	System Supplier ECU S/W Number µP 2 { U }	55		
	System Supplier ECU S/W Number µP 2 { P }	50		
	System Supplier ECU S/W Number µP 2 { 2 }	02		
	System Supplier ECU S/W Number µP 2 { }	20		
	System Supplier ECU S/W Number µP 2 { 53129 }	CF		
	System Supplier ECU S/W Number µP 2	89		
	System Supplier ECU S/W Version No µP 1 { U }	55		
	System Supplier ECU S/W Version No µP 1 { P }	50		
	System Supplier ECU S/W Version No µP 1 { 1 }	31		
	System Supplier ECU S/W Version No µP 1 { }	20		
	System Supplier ECU S/W Version No µP 1 { 0 }	00		
	System Supplier ECU S/W Version No µP 1 { 1 }	01		
	System Supplier ECU S/W Version No µP 1 { }	20		
	System Supplier ECU S/W Version No µP 2 { }	20		
	System Supplier ECU S/W Version No µP 2 { U }	55		
	System Supplier ECU S/W Version No µP 2 { P }	50		
	System Supplier ECU S/W Version No µP 2 { 2 }	02		
	System Supplier ECU S/W Version No µP 2 { }	20		
	System Supplier ECU S/W Version No µP 2 { 0 }	00		
	System Supplier ECU S/W Version No µP 2 { 3 }	03		
	Exhaust Regulat. / Type Approval Number { B }	42		
	Exhaust Regulat. / Type Approval Number { 9 }	39		
	Exhaust Regulat. / Type Approval Number { 4 }	34		
	Exhaust Regulat. / Type Approval Number { 0 }	30		
	Exhaust Regulat. / Type Approval Number { 0 }	30		
	Exhaust Regulat. / Type Approval Number { 1 }	31		

	Engine Name { X }	58		
	Engine Name { 2 }	32		
	Engine Name { 0 }	30		
	Engine Name { X }	58		
	Engine Name { E }	45		
	Engine Name { V }	56		
	Repair Shop Code/Tester Serial No. { 6359984 }	61		
	Repair Shop Code/Tester Serial No.	06		
	Repair Shop Code/Tester Serial No.	60		
	Programming Date { 19940911 }	19		
	Programming Date	94		
	Programming Date	09		
	Programming Date	11		

STEP#3 readECUIdentification(IO_VIN)

time	Client (tester) Request Message	Hex
P3	**readECUIdentification.ReqSId**[	**1A**
	identificationOption = VIN]	90

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**readECUIdentification.PosRspSId**[	**5A**	**negativeResponse Service Identifier**	**7F**
	identificationOption = VIN	90	**readECUIdentification.ReqSId**[	**1A**
	VIN { W }	57	responseCode { refer to section 4.4 }]	xx
	VIN { 0 }	30		
	VIN { L }	4C		
	VIN { 0 }	30		
	VIN { 0 }	30		
	VIN { 0 }	30		
	VIN { 0 }	30		
	VIN { 4 }	34		
	VIN { 3 }	33		
	VIN { M }	4D		
	VIN { B }	42		
	VIN { 5 }	35		
	VIN { 4 }	34		
	VIN { 1 }	31		
	VIN { 3 }	33		
	VIN { 2 }	32		
	VIN { 6 }]	36		

7 Data Transmission functional unit

The services provided by this functional unit are described in table 7:

Service name	Description
ReadDataByLocalIdentifier	The client requests the transmission of the current value of a record with access by recordLocalIdentifier.
ReadDataByCommonIdentfier	The client requests the transmission of the current value of a record with access by recordCommonIdentifier.
ReadMemoryByAddress	The client requests the transmission of a memory area.
DynamicallyDefineLocalIdentifier	The client requests to dynamically define local identifiers that may subsequently be accessed by recordLocalIdentifier.
WriteDataByLocalIdentifier	The client requests to write a record accessed by recordLocalIdentifier.
WriteDataByCommonIdentifier	The client requests to write a record accessed by recordCommonIdentifier.
WriteMemoryByAddress	The client requests to overwrite a memory area.
SetDataRates	**This service is not part of SSF 14230-3 and shall therefore not be implemented!**
StopRepeatedDataTransmission	**This service is not part of SSF 14230-3 and shall therefore not be implemented!**

Table 7 - Data Transmission functional unit

7.1 ReadDataByLocalIdentifier service

7.1.1 Message description

The readDataByLocalIdentifier request message requests data record values from the server identified by a recordLocalIdentifier. The server sends data record values via the readDataByLocalIdentifier positive response message. The format and definition of the recordValues shall be vehicle manufacturer specific. RecordValues can include analog input and output signals, digital input and output signals, internal data and system status information if supported by the ECU.

7.1.2 Message data bytes

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**readDataByLocalIdentifier Request Service Id**	**M**	**21**	**RDBLI**
#2	recordLocalIdentifier=[refer to table 7.1.3]	M	xx	**RLI_...**

Table 7.1.2.1 - readDataByLocalIdentifier Request Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**readDataByLocalIdentifier Positive Response Service Id**	**M**	**61**	**RDBLIPR**
#2	recordLocalIdentifier=[refer to table 7.1.3]	M	xx	**RLI_...**
#3	recordValue#1	M	xx	**RV_...**
:	:	:	:	:
#n	recordValue#m	U	xx	**RV_...**

Table 7.1.2.2 - readDataByLocalIdentifier Positive Response Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**negativeResponse Service Id**	**M**	**7F**	**NR**
#2	**readDataByLocalIdentifier Request Service Id**	**M**	**21**	**RDBLI**
#3	responseCode=[KWP2000ResponseCode { section 4.4 }]	M	xx=[00-FF]	**RC_...**

Table 7.1.2.3 - readDataByLocalIdentifier Negative Response Message

7.1.3 Parameter definition

- The parameter ***recordLocalIdentifier (RLI_)*** in the readDataByLocalIdentifier request message identifies a server specific local data record. This parameter shall be available in the server's memory. The recordLocalIdentifier value shall either exist in fixed memory or temporarily be stored in RAM e.g. if it is defined dynamically by the service dynamicallyDefineLocalIdentifier.
  RecordLocalIdentifer values defined by this document are shown in table 7.1.3 below.
  The recordLocalIdentifier table consists of four (4) columns and multiple lines.
  - The **1st column (Hex)** includes the “Hex Value” assigned to the recordLocalIdentifier specified in the 2nd column.
  - The **2nd column (Description)** specifies the recordLocalIdentifier.
  - The **3rd column (Cvt)** is the convention column which specifies whether the recordLocalIdentifier is “M” (mandatory) or “U” (User Optional).
  - The **4th column (Mnemonic)** specifies the mnemonic of this recordLocalIdentifier.

Hex	Description	Cvt	Mnemonic
00	**reservedByDocument** This value shall be reserved by this document for future definition.	**M**	**RBD**
01	**localIdentifierScalingTable** This value shall cause the server (ECU) to report the Local Identifier scaling table in the readDataByLocalIdentifier positive response message. The data content of the localIdentifierScalingTable depends on the data (length and format) being implemented in the ECU. The scaling data shall be of the format specified in section 5.4 Data Scaling.	**U**	**LIST**
02 - EF	**localIdentifier** This range of values shall be reserved to reference data records with fixed identifiers implemented in the server (ECU).	**U**	**RLI_...**
F0 - F9	**dynamicallyDefinedLocalIdentifier** This range of values shall be reserved to identify dynamically defined data records in the server (ECU).	**U**	**DDDLI_...**

Hex	Description	Cvt	Mnemonic
FA - FE	**systemSupplierSpecific** This range of values is reserved for system supplier specific use.	**U**	**SSS**
FF	**reservedByDocument** This value shall be reserved by this document for future definition.	**M**	**RBD**

Table 7.1.3 - Definition of recordLocalIdentifier values

- The parameter ***recordValue (RV_)*** is used by the readDataByLocalIdentifier positive response message to provide the data record identified by the recordLocalIdentifier to the client (tester). The user optional recordValues shall include ECU specific input, internal and output data.

7.1.4 Message flow example

See message flow examples of a Physically Addressed Service in section 5.3.1.1 and a Functionally Addressed Service in section 5.3.2.1.

7.1.5 Implementation example of readDataByLocalIdentifier

7.1.5.1 Message flow conditions for readDataByLocalIdentifier

The service readDataByLocalIdentifier is supported without any restriction by the server (ECU).
The recordLocalIdentifier example includes a datastream with multiple input signals, internal ECU parameters and output signals. Each parameter is abbreviated with "RV_" in the beginning and followed by the abbreviation specified in SAE J1930 like "B+" (Battery Positive Voltage) or "TPS" (Throttle Position Sensor).

7.1.5.2 Message flow

7.1.5.2.1 Message flow with record#10

STEP#1 readDataByLocalIdentifier(RLI_RECORD#10)

time	Client (tester) Request Message	Hex
P3	**readDataByLocalIdentifier.ReqSId**[recordLocalIdentifier=record#10]	**21** 0A

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex

P2	**readDataByLocalIdentifier.PosRspSId**[
	recordLocalIdentifier=record#10
	RV_B+ { Battery Positive Voltage }
	RV_ECT { Engine Coolant Temperature }
	RV_TPS { Throttle Position Sensor }
	RV_O2SB1 { Oxygen Sensor Bank 1 }
	RV_**...**
	:
	RV_**...**]
		61
		0A
		8C
		A6
		66
		A0
		xx
		:
		xx
			negativeResponse Service Identifier
			readDataByLocalIdentifier.ReqSId[
			responseCode { refer to section 4.4 }]
				7F
				21
				xx

	return(main)		**return(responseCode)**	

7.2 ReadDataByCommonIdentifier service

7.2.1 Message description

The readDataByCommonIdentifier request message requests data record values from the server(s) identified by a common recordCommonIdentifier. The server(s) send data record values via the readDataByCommonIdentifier positive response message. The format and definition of the recordValues shall be vehicle manufacturer specific. RecordValues can include analog input and output signals, digital input and output signals, internal data and system status information if supported by the ECU(s).

7.2.2 Message data bytes

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**readDataByCommonIdentifier Request Service Id**	**M**	**22**	**RDBCI**
#2	recordCommonIdentifier (High Byte) =[refer to table 7.2.3]	M	xx	**RCI_...**
#3	recordCommonIdentifier (Low Byte) =[refer to table 7.2.3]	M	xx	

Table 7.2.2.1 - readDataByCommonIdentifier Request Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**readDataByCommonIdentifier Positive Response Service Id**	**M**	**62**	**RDBCIPR**
#2	recordCommonIdentifier (High Byte) =[refer to table 7.2.3]	M	xx	**RCI_...**
#3	recordCommonIdentifier (Low Byte) =[refer to table 7.2.3]	M	xx	
#4	recordValue#1	M	xx	**RV_...**
:	:	:	:	:
#n	recordValue#m	U	xx	**RV_...**

Table 7.2.2.2 - readDataByCommonIdentifier Positive Response Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**negativeResponse Service Id**	**M**	**7F**	**NR**
#2	**readDataByCommonIdentifier Request Service Id**	**M**	**22**	**RDBCI**
#3	responseCode=[KWP2000ResponseCode { section 4.4 }]	M	xx=[00-FF]	**RC_...**

Table 7.2.2.3 - readDataByCommonIdentifier Negative Respons Message

7.2.3 Parameter definition

- The parameter ***recordCommonIdentifier (RCI_)*** in the readDataByCommonIdentifier service identifies a data record which is supported by multiple servers.
  RecordCommonIdentifer values defined by this document are shown in table 7.2.3 below.
  The recordCommonIdentifier table consists of four (4) columns and multiple lines.
  - The **1st column (Hex)** includes the "Hex Value" assigned to the recordCommonIdentifier specified in the 2nd column.
  - The **2nd column (Description)** specifies the recordCommonIdentifier.
  - The **3rd column (Cvt)** is the convention column which specifies whether the recordCommonIdentifier is "M" (mandatory) or "U" (User Optional).
  - The **4th column (Mnemonic)** specifies the mnemonic of this recordCommonIdentifier.

Hex	Description	Cvt	Mnemonic
0000	**reservedByDocument** This value shall be reserved by this document for future definition.	**M**	**RBD**
0001	**commonIdentifierScalingTable** This value shall cause the server (ECU) to report the Common Identifier scaling table in the readDataByCommonIdentifier positive response message. The data content of the commonIdentifierScalingTable depends on the data (length and format) being implemented in the ECU. The scaling data shall be of the format specified in section 5.4 Data Scaling.	**U**	**CIST**
0002 - 007F	**commonIdentifier** This range of values shall be reserved to reference data records with fixed identifiers implemented in the server (ECU).	**U**	**RCI_...**
0080 - 00BF	**identificationOption** This range of values shall be reserved for server (ECU) identification option information (refer to section 6.6.1).	**M/U**	**IO_...**
00C0 - EFFF	**commonIdentifier** This range of values shall be reserved to reference data records with fixed identifiers implemented in the server (ECU).	**U**	**RCI_...**
F000 - FFFE	**systemSupplierSpecific** This range of values is reserved for system supplier specific use.	**U**	**SSS**
FFFF	**reservedByDocument** This value shall be reserved by this document for future definition.	**M**	**RBD**

Table 7.2.3 - Definition of recordCommonIdentifier values

- The parameter ***recordValue (RV_)*** is used by the readDataByCommonIdentifier positive response message to provide the data record identified by the recordCommonIdentifier to the client (tester). The user optional recordValues shall include ECU specific input, internal and output data.

7.2.4 Message flow example

See message flow example of Physically Addressed Service in section 5.3.1.1 and Functionally Addressed Service in section 5.3.2.1.

7.2.5 Implementation example of readDataByCommonIdentifier

7.2.5.1 Message flow conditions for readDataByCommonIdentifier

The service readDataByCommonIdentifier is supported without any restriction by the server (ECU).
This example requests the parameter "Battery Positive Voltage" (B+) by referencing the data by a recordCommonIdentifier. The identifier may be built from the SAE J1979 specification. Each PID in the SAE J1979 consists of a single byte PID. In this example the SAE J1979 PID is preceded by a high byte which includes '$00'.
Each parameter is abbreviated with "RV_" in the beginning and followed by the abbreviation specified in SAE J1930 like "B+" (Battery Positive Voltage).

7.2.5.2 Message flow

STEP#1 readDataByCommonIdentifier(RCI_B+)

time	Client (tester) Request Message	Hex
P3	**readDataByCommonIdentifier.ReqSId**[recordCommonIdentifier { High Byte } = Battery Positive Voltage recordCommonIdentifier { Low Byte } = Battery Positive Voltage]	**22** 00 10

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex

P2	**readDataByCommonIdentifier.PosRspSId**[
	recordCommonIdentifier { High Byte }{ B+ }
	recordCommonIdentifier { Low Byte } { B+ }
	RV_B+ { B+ = 13.8 V (resolution: 100mV)}]
		62
		00
		10
		8A
			negativeResponse Service Identifier
			readDataByCommonIdentifier.ReqSId[
			responseCode { refer to section 4.4 }]
				7F
				22
				xx

	return(main)		**return(responseCode)**	

7.3 ReadMemoryByAddress service

7.3.1 Message description

The readMemoryByAddress request message requests memory data from the server identified by the parameters memoryAddress and memorySize.
The server sends data record values via the readMemoryByAddress positive response message. The format and definition of the recordValues shall be vehicle manufacturer specific. RecordValues can include analog input and output signals, digital input and output signals, internal data and system status information if supported by the ECU.

7.3.2 Message data bytes

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**readMemoryByAddress Request**	**M**	**23**	**RMBA**
#2	memoryAddress (High Byte)	M	xx	**MA_...**
#3	memoryAddress (Middle Byte)	M	xx	
#4	memoryAddress (Low Byte)	M	xx	
#5	memorySize	M	xx	**MS_...**

Table 7.3.2.1 - ReadMemoryByAddress Request Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**readMemoryByAddress Positive Response**	**M**	**63**	**RMBAPR**
#2	recordValue#1	M	xx	**RV_...**
:	**:**	**:**	**:**	**:**
#n	recordValue#m	U	xx	**RV_...**

Table 7.3.2.2 - ReadMemoryByAddress Positive Response Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**negativeResponse Service Id**	**M**	**7F**	**NR**
#2	**readMemoryByAddressRequest Service Id**	**M**	**23**	**RMBA**
#3	responseCode=[KWP2000ResponseCode { section 4.4 }]	M	xx=[00-FF]	**RC_...**

Table 7.3.2.3 - ReadMemoryByAddress Negative Response Message

7.3.3 Parameter definition

- The parameter ***memoryAddress (MA_)*** in the readMemoryByAddress request message identifies the start address in the server's memory. If the server supports a "16 bit" wide address range the high byte of the memoryAddress shall be used as a memoryIdentifier. The memoryIdentifier shall be vehicle manufacturer/system supplier/project specific.
- The parameter ***memorySize (MS_)*** specifies the number of bytes to be read starting at a specified memory address in the server's memory.
- The parameter ***recordValue (RV_)*** is used by the readMemoryByAddress positive response message to provide the data record identified by the Memory Address to the client (tester). The user optional recordValues shall include ECU specific input, internal and output data.

7.3.4 Message flow example

See message flow example of Physically Addressed Service in section 5.3.1.1.

7.3.5 Implementation example of readMemoryByAddress

7.3.5.1 Test specification readMemoryByAddress

The service in this example is not limited by any restriction of the server (ECU). The client (tester) reads three (3) data bytes from the server's (ECU's) external RAM cells starting at memory address $204813.

7.3.5.2 Message flow

STEP#1 readMemoryByAddress(MA_..., MS_...)

time	Client (tester) Request Message	Hex
P3	**readMemoryByAddress.ReqSId**[	**23**
	memoryAddress { High Byte }	20
	memoryAddress{ Middle Byte }	48
	memoryAddress{ Low Byte }	13
	memorySize]	03

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**readMemoryByAddress.PosRspSId**[	**63**	**negativeResponse Service Identifier**	**7F**
	externalRAMCell#1	00	**readMemoryByAddress.ReqSId**[	**23**
	externalRAMCell#2	46	responseCode { refer to section 4.4 }]	xx
	externalRAMCell#3]	FB		
	return(main)		**return(responseCode)**	

7.4 DynamicallyDefineLocalIdentifier service

7.4.1 Message description

This message is used by the client (tester) to dynamically define the content of a local identifier accessible with the readDataByLocalIdentifier service.
The request message may consist of multiple definitions of different definitionModes (other than clearDynamicallyDefinedLocalIdentifier).
The server shall send a positive response message after it has stored the definition of the dynamicallyDefinedLocalIdentifier record.

If the definitionMode parameter is set to clearDynamicallyDefinedLocalIdentifier this service is used by the client to clear a dynamicallyDefinedLocalIdentifier. This request message shall free up the server's memory which is used to create a dynamicallyDefinedLocalIdentifier data record.
The server shall send a positive response message after it has erased the dynamically defined local Identifier record.

Note: It shall not be possible to modify an already existing dynamically defined data record referenced by a local identifier. It shall be possible to define multiple dynamicallyDefinedLocalIdentifiers.

7.4.2 Message data bytes

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**dynamicallyDefineLocalIdentifier Request Service Id**	**M**	**2C**	**DDLI**
#2	dynamicallyDefinedLocalIdentifier	M	xx	**DDDLI_...**
#3	definitionMode#1	M	xx	**DM_..**
#4	positionInDynamicallyDefinedLocalIdentifier	C1	xx	**PIDDDLI_...**
#5	memorySize	C1	xx	**MS_..**
 #6 #7 ... #i+5	dataRecord#1 [Data#1 Data#2 ... Data#i]	C1	 xx xx ... xx	
#i+6	definitionMode#2	C2	xx	**DM_..**
#i+7	positionInDynamicallyDefinedLocalIdentifier	C2	xx	**PIDDDLI_...**
#i+8	memorySize	C2	xx	**MS_..**
 #i+9 #i+10 ... #i+j+8	dataRecord#2 [Data#1 Data#2 ... Data#j]	C2	 xx xx ... xx	
:	**:**	**:**	**:**	**:**
#n-k	definitionMode#m	C2	xx	**DM_..**
#n-k+1	positionInDynamicallyDefinedLocalIdentifier	C2	xx	**PIDDDLI_...**
#n-k+2	memorySize	C2	xx	**MS_..**
 #n-k+3 #n-k+4 ... #n	dataRecord#m [Data#1 Data#2 ... Data#k]	C2	 xx xx ... xx	

Table 7.4.2.1 - DynamicallyDefineLocalIdentifier Request Message

C1 = condition: parameter is not present if definitionMode is set to clearDynamicallyDefinedLocalIdentifier
C2 = condition: parameter is only present if dynamicallyDefinedLocalIdentifier consists of more than one dataRecord.

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**dynamicallyDefineLocalIdentifier Positive Response SId**	**M**	**6C**	**DDLIPR**

#2	dynamicallyDefinedLocalIdentifier	M	xx	**DDDLI_...**

Table 7.4.2.2 - DynamicallyDefineLocalIdentifier Positive Response Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**negativeResponse Service Id**	**M**	**7F**	**NR**
#2	**dynamicallyDefineLocalIdentifierRequest Service Id**	**M**	**2C**	**DDLI**
#3	responseCode=[KWP2000ResponseCode { section 4.4 }]	M	xx=[00-FF]	**RC_...**

Table 7.4.2.3 - DynamicallyDefineLocalIdentifier Negative Response Message

7.4.3 Parameter definition

- The parameter ***dynamicallyDefinedLocalIdentifier (DDDLI_)*** (range of values as specified in table 7.1.3: $F0 - $F9) in the dynamicallyDefineLocalIdentifier request message identifies a new data record which has been defined by the client (tester) in the request message. The new data record shall be read with the readDataByLocalIdentifier service using the dynamicallyDefinedLocalIdentifier ($F0 - $F9) as a recordLocalIdentifier parameter value.
- The parameter ***definitionMode (DNM_)*** in the dynamicallyDefineLocalIdentifier request message specifies the method of how the data record is defined. Values are defined in the table below:

Hex	Description	Cvt	Mnemonic
00	**reservedByDocument** This value is reserved by this document for future definition.	**M**	**RBD**
01	**defineByLocalIdentifier** The purpose of local identifiers is to reference a unique parameter of a server (ECU).	**U**	**DBLI**
02	**defineByCommonIdentifier** The purpose of common identifiers is to reference a unique parameter with the same value known by multiple servers (ECUs).	**U**	**DBCI**
03	**defineByMemoryAddress** The purpose of a memory address as an identifier is to reference a parameter in the memory of a server (ECU).This definitionMode may be used if local identifiers are not supported by the server (ECU).	**U**	**DBMA**
04	**clearDynamicallyDefinedLocalIdentifier** The purpose of this definition mode is to clear a dynamicallyDefinedLocalIdentifier data record which previously has been defined in the server (ECU).	**U**	**CDDDLI**
05 - 7F	**reservedByDocument** This range of values is reserved by this document for future definition.	**M**	**RBD**
80	**vehicleManufacturerSpecific** This value is reserved for vehicle manufacturer specific use.	**U**	**VMS**
81	**defineByInputOutputLocalIdentifier** The purpose of input/output local identifiers is to reference a unique input/output of a server (ECU).	**U**	**DBLI**
82	**defineByInputOutputCommonIdentifier** The purpose of input/output common identifiers is to reference an input/output with the same reference known by multiple servers (ECUs).	**U**	**DBCI**
83 - F9	**vehicleManufacturerSpecific** This range of values is reserved for vehicle manufacturer specific use.	**U**	**VMS**
FA - FE	**systemSupplierSpecific** This range of values is reserved for system supplier specific use.	**U**	**SSS**
FF	**reservedByDocument** This value is reserved by this document for future definition.	**M**	**RBD**

Table 7.4.3 - Definition of definitionMode values

- The parameter ***positionInDynamicallyDefinedLocalIdentifier (PIDDDLI_)*** in the dynamicallyDefineLocal-Identifier request message identifies the **data record position in the dynamically defined data record** (first possible PIDDDLI=1). The dynamically defined data record is defined as the readDataByLocalIdentifier positive response message using the dynamicallyDefinedLocalIdentifier ($F0 - $F9) as a recordLocalIdentifier parameter value.

Note: The parameter *positionInDynamicallyDefinedLocalIdentifier* shall specify the consecutive order number of the data record (1,2,3, ...). It shall not specify the absolute byte position of the data record in the dynamically defined data record.

- The parameter ***memorySize (MS_)*** in the dynamicallyDefineLocalIdentifier request message identifies the number of bytes of data identified in the data record.
- The definition of the parameter ***dataRecord*** depends on the definitionMode:

definitionMode = defineByLocalIdentifier

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
	dataRecord[			
#1	recordLocalIdentifier	M	xx	**RLI_...**
#2	positionInRecordLocalIdentifier]	M	xx	**PIRLI_...**

- The parameter ***recordLocalIdentifier (RLI_)*** is defined in section 7.1.3.
- The parameter ***positionInRecordLocalIdentifier (PIRLI_)*** in the dynamicallyDefineLocalIdentifier request message identifies the **data record position in the readDataByLocalIdentifier positive response message (recordLocalIdentifier parameter value NOT $F0 - $F9)** (first possible PIRLI=1, the byte after the ServiceId and RLI).

definitionMode = defineByCommonIdentifier

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
	dataRecord[			
#1	recordCommonIdentifier (High Byte)	M	xx	**RCI_...**
#2	recordCommonIdentifier (Low Byte)	M	xx	
#3	positionInRecordCommonIdentifier]	M	xx	**PIRCI_...**

- The parameter ***recordCommonIdentifier (RCI_)*** is defined in section 7.2.3.
- The parameter ***positionInRecordCommonIdentifier (PIRCI_)*** in the dynamicallyDefineLocalIdentifier request message identifies the **data record position in the readDataByCommonIdentifier positive response message** (first possible PIRLI=1, the byte after the ServiceId and RCI).

definitionMode = defineByMemoryAddress

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
	dataRecord[			
#1	memoryAddress(High Byte)	M	xx	**MA_...**
#2	memoryAddress(middle Byte)	M	xx	
#3	memoryAddress(Low Byte)]	M	xx	

- The parameter ***memoryAddress (MA_)*** in the dynamicallyDefineLocalIdentifier request message identifies the start address of a data record in the server's memory. If the server (ECU) supports a "16 bit" wide address range the high byte of the memoryAddress shall be included as the memoryIdentifier. The definition of this value is vehicle manufacturer or system supplier specific.

definitionMode = defineByInputOutputLocalIdentifier

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
	dataRecord[			
#1	inputOutputLocalIdentifier	M	xx	**IOLI_...**
#2	inputOutputControlParameter	M	xx	**IOCP_...**
#3	positionInInputOutputLocalIdentifier]	M	xx	**PIIOLI_...**

- The parameter ***inputOutputLocalIdentifier (IOLI_)*** is defined in section 9.1.3.
- The parameter ***inputOutputControlParameter (IOCP_)*** is defined in section 9.1.3. Only the values $01, $02 and $0A-$FF can be used.
- The parameter ***positionInInputOutputLocalIdentifier (PIIOLI_)*** in the dynamicallyDefineLocalIdentifier request message identifies the **data record position in the inputOutputControlByLocalIdentifier positive response message** (first possible PIIOLI=2, the byte after the ServiceId, IOLI and IOCP).

definitionMode = defineByInputOutputCommonIdentifier

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic

	dataRecord[			
#1	inputOutputCommonIdentifier (High Byte)	M	xx	**IOCI_...**
#2	inputOutputCommonIdentifier (Low Byte)	M	xx	
#3	inputOutputControlParameter	M	xx	**IOCP_...**
#4	positionInInputOutputCommonIdentifier]	M	xx	**PIIOCI_...**

- The parameter ***inputOutputCommonIdentifier (IOCI_)*** is defined in section 9.2.3.
- The parameter ***inputOutputControlParameter (IOCP_)*** is defined in section 9.2.3. Only the values $01, $02 and $0A-$FF can be used.
- The parameter ***positionInInputOutputLocalIdentifier (PIIOLI_)*** in the dynamicallyDefineLocalIdentifier request message identifies the **data record position in the inputOutputControlByCommonIdentifier positive response message** (first possible PIIOCI=3, the byte after the ServiceId, IOCI and IOCP).

definitionMode = clearDynamicallyDefinedLocalIdentifier
- The parameter dataRecord is not used when definitionMode is set to clearDynamicallyDefinedLocalIdentifier

Note: If the definitionMode parameter is set to clearDynamicallyDefinedLocalIdentifier the DynamicallyDefineLocalIdentifier request message can only consist of this single definitionMode.

7.4.4 Message flow examples

The following message flow example shows a dynamicallyDefineLocalIdentifier service including the definitionMode parameter defineByLocalIdentifier. The dynamically defined data record is then read by a readDataByLocalIdentifier service.

Time	client (Tester)	server (ECU)
P3	dynamicallyDefineLocalId.Request[...]	
P2		dynamicallyDefineLocalId.PositiveResponse [...]
P3	readDataByLocalId.Request[...]	
P2		readDataByLocalId.PositiveResponse[...]

Table 7.4.4 - Message flow example of dynamicallyDefineLocalIdentifier service followed by a readDataByLocalIdentifier service

7.4.5 Implementation example of dynamicallyDefineLocalIdentifier

Note: In the following implementation examples the request messages only consists of multiple definitions of **the same** definitionMode. The service definition however allows multiple definitions of **different** definitionModes.

7.4.5.1 Message flow conditions for dynamicallyDefineLocalIdentifier (defineByLocalIdentifier)

The service in this example is not limited by any restriction of the server (ECU). The client (tester) supports a feature of allowing the service technician to select display parameters which he needs for a specific test (e.g. engine warm up test).

• Engine Coolant Temperature	(RV_ECT)
• Engine Speed	(RV_ES)
• Throttle Position Voltage	(RV_TP)

This table specifies a readDataByLocalIdentifier message referenced by the recordLocalIdentifier (RLI = $01) which is used to dynamically create a new data record which includes only a few small data records out of the entire message.

Header	Service Id	RLI	PIRLI $01 - $06	PIRLI: $07 RV_ECT	PIRLI $08 - $0E	PIRLI: $0F -$11 RV_ES	PIRLI $12 - $14	PIRLI: $15 RV_TP	CS
$xx - $xx	$61	$01	$xx - $xx	$8C	$xx - $xx	$0060FE	$xx - $xx	$30	$xx

Table 7.4.5.1.1 - Data record referenced by RLI = $01 of the readDataByLocalIdentifier message

The client (tester) shall either dynamically create the information specified in the table 7.4.5.1.2 (based on technician parameter selection) or the information may be preprogrammed in the memory of the client (tester). The table information is required to create the dynamicallyDefineLocalIdentifier request message(s).

Definition of the dynamicallyDefinedLocalIdentifier ($F0) with the definitionMode defineByLocalIdentifier (DNM_DBLI)					
Parameter Name (RV_)	**definition Mode (DNM_)**	**positionInDynamically DefinedLocalIdentifier (PIDDDLI)**	**memory Size (MS_)**	**recordLocal Identifier (RLI_)**	**positionInRecord LocalIdentifier (PIRLI)**
Engine Coolant Temperature {RV_ECT}	$01	$01	1 byte	$01	$07
Engine Speed {RV_ES}	$01	$02	3 bytes	$01	$0F
Throttle Position Voltage {RV_TP}	$01	$03	1 byte	$01	$15

Table 7.4.5.1.2 - Data to create the dynamicallyDefineLocalIdentifier request message(s)

Within KWP 2000 the dynamicallyDefineLocalIdentifier service is used to assign a dynamically defined localIdentifier "$F0" to above three (3) parameters which are accessed by a readDataByLocalIdentifier service. The client (tester) defines the value "$F0" (refer to table 7.1.1) for this localIdentifier which identifies a new record consisting of the above three (3) parameters selected by the service technician.

7.4.5.2 Message flow

7.4.5.2.1 Message flow with single request message

The service technician has selected the record values **"Engine Coolant Temperature" (ECT), "Engine Speed" (ES)** and **"Throttle Position" (TP)** in the client (tester) as the parameters to be dynamically defined with the dynamicallyDefineLocalIdentifier service in the server's (ECU's) memory.

STEP#1 dynamicallyDefineLocalIdentifier(..., RLI_ECT, ..., RLI_ES, ..., RLI_TP)

time	Client (tester) Request Message	Hex
P3	**dynamicallyDefineLocalIdentifier.ReqSId**[	**2C**
	dynamicallyDefinedLocalIdentifier (DDDLI)	F0
	definitionMode=defineByLocalIdentifier (DBLI)	01
	positionInDynamicallyDefinedLocalIdentifier (PIDDDLI)	01
	memorySize (MS)	01
	recordLocalIdentifier=(RLI of datastream which includes EngineCoolantTemperature)	01
	positionInRecordLocalIdentifier (PIRLI)	07
	definitionMode=defineByLocalIdentifier (DBLI)	01
	positionInDynamicallyDefinedLocalIdentifier (PIDDDLI)	02
	memorySize (MS)	03
	recordLocalIdentifier=(RLI of datastream which includes EngineSpeed)	01
	positionInRecordLocalIdentifier (PIRLI)	0F
	definitionMode= defineByLocalIdentifier (DBLI)	01
	positionInDynamicallyDefinedLocalIdentifier (PIDDDLI)	03
	memorySize (MS)	01
	recordLocalIdentifier=(RLI of datastream which includes ThrottlePosition)	01
	positionInRecordLocalIdentifier (PIRLI)]	15

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**dynamicallyDefineLocalIdentifier.PosRspSId**[	**6C**	**negativeResponse Service Identifier**	**7F**
	dynamicallyDefinedLocalIdentifier]	F0	**dynamicallyDefineLocalIdentifier.ReqSId**[	**2C**
			responseCode { refer to section 4.4 }]	xx
	goto STEP#2		**return(responseCode)**	

7.4.5.2.2 Message flow readDataByLocalIdentifier with RLI = $F0

After completion of the dynamicallyDefineLocalIdentifier (RLI_F0) the new record shall be read by the client (tester). The client (tester) sends a readDataByLocalIdentifier request message with the recordLocalIdentifier set to '$F0' to read the values of the three (3) parameters defined previously.

STEP#2 readDataByLocalIdentifier(RLI_F0)

time	Client (tester) Request Message	Hex
P3	**readDataByLocalIdentifier.ReqSId**[	**21**
	recordLocalIdentifier = dynamicallyDefinedRecordLocalIdentfier(**RLI_F0**)]	F0

time	Server (ECU) Positive Response Message	PIDDDLI	Hex	Server (ECU) Negative Response Message	Hex

P2	**readDataByLocalIdentifier.PosRspSId**[
	RLI_F0 { recordLocalIdentifier }
	RV_ECT { Engine Coolant Temperature }
	RV_ES { Engine Speed }
	RV_ES { Engine Speed }
	RV_ES { Engine Speed }
	RV_TP { Throttle Position }]
		
		01
		02
		03
		04
		05
			61
			F0
			8C
			00
			60
			FE
			30
				negativeResponse Service Identifier
				readDataByLocalIdentifier.ReqSId[
				responseCode { refer to section 4.4 }]
					7F
					21
					xx

	return(main)			**return(responseCode)**	

7.4.5.3 Message flow conditions for dynamicallyDefineLocalIdentifier (defineByCommonIdentifier)

The service in this example is not limited by any restriction of the server (ECU). The client (tester) supports a feature of allowing the service technician to select display parameters which he needs for a specific test (e.g. O2 Sensor Bank 1 test).

• O2 Sensor Bank 1	(RV_O2SB1)
• O2 Integrator Bank 1	(RV_O2IB1)
• Engine Speed	(RV_ES)

This table specifies three (3) readDataByCommonIdentifier messages referenced by three (3) different recordCommonIdentifiers which are used to dynamically create a new data record.

Header	ServiceIdentifier (SID)	recordCommonId (RCI_)	PIRLI: $01	CS
$xx - $xx	$62	$0108	$8C **(RV_O2SB1)**	$xx
$xx - $xx	$62	$0105	$22 **(RV_O2IB1)**	$xx
$xx - $xx	$62	$0102	$0060FE **(RV_ES)**	$xx

Table 7.4.5.3.1 - Data record referenced by three different RCI of three different readDataByCommonIdentifier messages

The client (tester) shall either dynamically create the information specified in the table 7.4.5.1.2 (based on technician parameter selection) or the information may be preprogrammed in the memory of the client (tester). The table information is required to create the dynamicallyDefineLocalIdentifier request message(s).

Definition of the dynamicallyDefinedLocalIdentifier ($F1) with the definitionMode defineByLocalIdentifier (DNM_DBLI)					
Parameter Name (RV_)	**definition Mode (DNM_)**	**positionInDynamically DefinedLocalIdentifier (PIDDDLI)**	**memory Size (MS_)**	**recordCom-monIdenti-fier (RCI_)**	**positionInRecord Common-Identifier(PIRLI)**
O2 Sensor Bank 1{RV_O2SB1}	$02	$01	1 byte	$0108	$01
O2 Integrator Bank 1{RV_O2IB1}	$02	$02	1 byte	$0105	$01
Engine Speed {RV_ES}	$02	$03	3 bytes	$0102	$01

Table 7.4.5.3.2 - Data to create the dynamicallyDefineLocalIdentifier request message(s)

Within KWP 2000 the dynamicallyDefineLocalIdentifier service is used to assign a dynamically defined localIdentifier "$F1" to above three (3) parameters which are accessed by a readDataByLocalIdentifier service.
The client (tester) defines the value "$F1" (refer to table 7.1.1) for this localIdentifier which identifies a new record consisting of the above three (3) parameters selected by the service technician.

7.4.5.4 Message flow

7.4.5.4.1 Message flow with single request message

The service technician has selected the record values **"O2 Sensor Bank 1" (O2SB1), "O2 Integrator Bank 1" (O2IB1)** and **"Engine Speed" (ES)** in the client (tester) as the parameters to be dynamically defined with the dynamicallyDefineLocalIdentifier service in the server's (ECU's) memory.

STEP#1 dynamicallyDefineLocalIdentifier(..., RCI_O2SB1, ..., RCI_O2IB1, ..., RCI_ES)

time	Client (tester) Request Message	Hex
P3	**dynamicallyDefineLocalIdentifier.ReqSId**[dynamicallyDefinedLocalIdentifier (DDDLI)	**2C** F1

	definitionMode=defineByCommonIdentifier (DBCI)
	positionInDynamicallyDefinedLocalIdentifier (PIDDDLI)
	memorySize (MS)
	recordCommonIdentifier=O2 Sensor Bank 1 (RCI) {High Byte}
	recordCommonIdentifier=O2 Sensor Bank 1 (RCI) {Low Byte}
	positionInRecordCommonIdentifier (PIRCI)
		02
		01
		01
		01
		08
		01

	definitionMode=defineByCommonIdentifier (DBCI)
	positionInDynamicallyDefinedLocalIdentifier (PIDDDLI)
	memorySize (MS)
	recordCommonIdentifier=O2 Integrator Bank 1 (RCI) {High Byte}
	recordCommonIdentifier=O2 Integrator Bank 1 (RCI) {Low Byte}
	positionInRecordCommonIdentifier (PIRCI)
		02
		02
		01
		01
		05
		01

	definitionMode=defineByCommonIdentifier (DBCI)
	positionInDynamicallyDefinedLocalIdentifier (PIDDDLI)
	memorySize (MS)
	recordCommonIdentifier=Engine Speed (RCI) {High Byte}
	recordCommonIdentifier= Engine Speed (RCI) {Low Byte}
	positionInRecordCommonIdentifier (PIRCI)]
		02
		03
		03
		01
		02
		01

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**dynamicallyDefineLocalIdentifier.PosRspSId**[dynamicallyDefinedLocalIdentifier]	**6C** F1	**negativeResponse Service Identifier** **dynamicallyDefineLocalIdentifier.ReqSId**[responseCode { refer to section 4.4 }]	**7F** **2C** xx
	goto STEP#2		**return(responseCode)**	

7.4.5.4.2 Message flow readDataByLocalIdentifier with RLI = $F1

After completion of the dynamicallyDefineLocalIdentifier (RLI_F1) the new record shall be read by the client (tester). The client (tester) sends a readDataByLocalIdentifier request message with the recordLocalIdentifier set to '$F1' to read the values of the three (3) parameters defined previously.

STEP#2 readDataByLocalIdentifier(RLI_F1)

time	Client (tester) Request Message	Hex
P3	**readDataByLocalIdentifier.ReqSId**[recordLocalIdentifier = dynamicallyDefinedRecordLocalIdentfier(**RLI_F1**)]	**21** F1

time	Server (ECU) Positive Response Message	PIDDDLI	Hex	Server (ECU) Negative Response Message	Hex

P2	**readDataByLocalIdentifier.PosRspSId**[
	RLI_F1 { recordLocalIdentifier }
	RV_O2SB1 { O2 Sensor Bank 1 }
	RV_O2IB1 { O2 Integrator Bank 1 }
	RV_ES { Engine Speed }
	RV_ES { Engine Speed }
	RV_ES { Engine Speed }]
		
		01
		02
		03
		04
		05
			61
			F1
			8C
			22
			00
			60
			FE
				negativeResponse Service Identifier
				readDataByLocalIdentifier.ReqSId[
				responseCode { refer to section 4.4 }]
					7F
					21
					xx

	return(main)			**return(responseCode)**	

7.4.5.5 Message flow conditions for dynamicallyDefineLocalIdentifier (defineByMemoryAddress)

The service in this example is not limited by any restriction of the server (ECU). The client (tester) supports a feature of allowing the service technician to select display parameters which he needs for a specific test (e.g. RAM Content test).

This table specifies two (2) readDataByMemoryAddress messages referenced by two (2) different memoryAddress parameter values which are used to dynamically create a new data record.

Header	ServiceIdentifier (SID)	recordValue (RV_)	checksum (CS)
$xx - $xx	$63	$45 {MA_ = $01DD22}	$xx
$xx - $xx	$63	$A7 {MA_ = $01AA33}	$xx

Table 7.4.5.5.1 - Data record referenced by two different memory addresses of two different readDataByMemoryAddress messages

The client (tester) shall either dynamically create the information specified in the table 7.4.5.1.2 (based on technician parameter selection) or the information may be preprogrammed in the memory of the client (tester). The table information is required to create the dynamicallyDefineLocalIdentifier request message(s).

Definition of the dynamicallyDefinedLocalIdentifier ($F2) with the definitionMode defineByLocalIdentifier (DNM_DBLI)				
Parameter Name (RV_)	**definition Mode (DNM_)**	**positionInDynamically DefinedLocalIdentifier (PIDDDLI)**	**memorySize (MS_)**	**memoryAddress (MA_)**
RAM Variable#1{RV_RAMVAR1}	$03	$01	1 byte	$01DD22
RAM Variable#2{RV_RAMVAR2}	$03	$02	1 byte	$01AA33

Table 7.4.5.5.2 - Data to create the dynamicallyDefineLocalIdentifier request message(s)

Within KWP 2000 the dynamicallyDefineLocalIdentifier service is used to assign a dynamically defined localIdentifier "$F2" to above two (2) parameters which are accessed by a readDataByLocalIdentifier service. The client (tester) defines the value "$F2" (refer to table 7.1.1) for this localIdentifier which identifies a new record consisting of the above two (2) parameters selected by the service technician.

7.4.5.6 Message flow

7.4.5.6.1 Message flow with single request message

The service technician has selected the memoryAddress values **"RAM Variable#1" (RAMV1), " RAM Variable#2" (RAMV2)** in the client (tester) as the parameters to be dynamically defined with the dynamicallyDefineLocalIdentifier service in the server's (ECU's) memory.

STEP#1 dynamicallyDefineLocalIdentifier(..., MA_ RAMV1, ..., MA_ RAMV1)

time	Client (tester) Request Message	Hex
P3	**dynamicallyDefineLocalIdentifier.ReqSId**[	**2C**
	dynamicallyDefinedLocalIdentifier (DDDLI)	F2
	definitionMode=defineByMemoryAddress (DBMA)	03
	positionInDynamicallyDefinedLocalIdentifier (PIDDDLI)	01
	memorySize (MS)	01
	memoryAddress=RAM Variable#1 (MA) {High Byte}	01
	memoryAddress=RAM Variable#1 (MA) {Middle Byte}	DD
	memoryAddress=RAM Variable#1 (MA) {Low Byte}	22
	definitionMode=defineByMemoryAddress (DBMA)	03
	positionInDynamicallyDefinedLocalIdentifier (PIDDDLI)	02
	memorySize (MS)	01
	memoryAddress=RAM Variable#2 (MA) {High Byte}	01
	memoryAddress=RAM Variable#2 (MA) {Middle Byte}	AA
	memoryAddress=RAM Variable#2 (MA) {Low Byte}]	33

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**dynamicallyDefineLocalIdentifier.PosRspSId**[	**6C**	**negativeResponse Service Identifier**	**7F**
	dynamicallyDefinedLocalIdentifier]	F2	**dynamicallyDefineLocalIdentifier.ReqSId**[	**2C**
			responseCode { refer to section 4.4 }]	xx
	goto STEP#2		**return(responseCode)**	

7.4.5.6.2 Message flow readDataByLocalIdentifier with RLI = $F2

After completion of the dynamicallyDefineLocalIdentifier (RLI_F2) the new record shall be read by the client (tester). The client (tester) sends a readDataByLocalIdentifier request message with the recordLocalIdentifier set to '$F2' to read the values of the two (2) RAM variables defined previously.

STEP#2 readDataByLocalIdentifier(RLI_F2)

time	Client (tester) Request Message	Hex
P3	**readDataByLocalIdentifier.ReqSId**[	**21**
	recordLocalIdentifier = dynamicallyDefinedRecordLocalIdentfier(**RLI_F2**)]	F2

time	Server (ECU) Positive Response Message	PIDDDLI	Hex	Server (ECU) Negative Response Message	Hex
P2	**readDataByLocalIdentifier.PosRspSId**[		**61**	**negativeResponse Service Identifier**	**7F**
	RLI_F2 { recordLocalIdentifier }		F2	**readDataByLocalIdentifier.ReqSId**[	**21**
	RV_RAMVAR1 { RAM Variable#1 }	01	45	responseCode { refer to section 4.4 }]	xx
	RV_RAMVAR2 { RAM Variable#2 }]	02	A7		
	goto STEP#3			**return(responseCode)**	

7.4.5.6.3 Message flow of deleting the dynamically defined recordLocalIdentifier

The client (tester) shall erase the dynamically defined recordLocalIdentifier. After successful completion of this service the client (tester) may create a new record of parameters referenced by a dynamicallyDefinedLocalIdentifier.
This example deletes the dynamically defined recordLocalIdentifier "$F2".

STEP#3 dynamicallyDefineLocalIdentifier(RLI_F2, CDDDLI)

time	Client (tester) Request Message	Hex
P3	**dynamicallyDefineLocalIdentifier.ReqSId**[	**2C**
	dynamicallyDefinedLocalIdentifier (**RLI_F2**)	F2
	definitionMode= clearDynamicallyDefinedLocalIdentifier (**CDDDLI**)]	04

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**dynamicallyDefineLocalIdentifier.PosRspSId**	**6C**	**negativeResponse Service Identifier**	**7F**
	[	F2	**dynamicallyDefineLocalIdentifier.ReqSId**[	**2C**
	dynamicallyDefinedLocalIdentifier]		responseCode { refer to section 4.4 }]	xx
	return(main)		**return(responseCode)**	

7.5 WriteDataByLocalIdentifier service

7.5.1 Message description

The writeDataByLocalIdentifier service is used by the client to write recordValues (data values) to a server. The data are identified by a recordLocalIdentifier. It is the vehicle manufacturer's responsibility that the server conditions are met when performing this service. Possible uses for this service are:

- Clear non-volatile memory
- Reset learned values
- Set option content
- Change calibration values

7.5.2 Message data bytes

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**writeDataByLocalIdentifier Request Service Id**	**M**	**3B**	**WDBLI**
#2	recordLocalIdentifier=[refer to table 7.1.3]	M	xx	**RLI_...**
#3 : #n	recordValue#1 : recordValue#m	M : M	xx : xx	**RV_...** **:** **RV_...**

Table 7.5.2.1 - WriteDataByLocalIdentifier Request Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**writeDataByLocalIdentifier Positive Response Service Id**	**M**	**7B**	**WDBLIPR**
#2	recordLocalIdentifier=[refer to table 7.1.3]	M	xx	**RLI_...**

Table 7.5.2.2 - WriteDataByLocalIdentifier Positive Response Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**negativeResponse Service Id**	**M**	**7F**	**NR**
#2	**writeDataByLocalIdentifier Request Service Id**	**M**	**3B**	**WDBLI**
#3	responseCode=[KWP2000ResponseCode { section 4.4 },]	M	xx=[00-FF]	**RC_...**

Table 7.5.2.3 - WriteDataByLocalIdentifier Negative Response Message

7.5.3 Parameter definition

- The parameter ***recordLocalIdentifier (RLI_)*** is defined in section 7.1.3.
- The parameter ***recordValue*** is defined in section 7.1.3.

7.5.4 Message flow example

See message flow examples of a Physically Addressed Service in section 5.3.1.1 and a Functionally Addressed Service in section 5.3.2.1.

7.5.5 Implementation example of “set Language code”

7.5.5.1 Message flow conditions for “set Language code”

The language code indicates which language shall be used for presentation of text messages on the servers display. The client (tester) shall write the language code data to the server (ECU) by using the writeDataByLocalIdentifier service. The server (ECU) shall send a positive response message after it has successfully programmed the language code data in EEPROM or FLASH memory.

7.5.5.2 Message flow

STEP#1 writeDataByLocalIdentifier(RLI_VIN, RV_VIN#1, ... , RV_VIN#17)

time	Client (tester) Request Message	Hex
P3	**writeDataByLocalIdentifier.ReqSId**[recordLocalIdentifier = Language Code]	**3B** 10

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**writeDataByLocalIdentifier.PosRspSId**[recordLocalIdentifier = Language Code]	**7B** 10	**negativeResponse Service Identifier** **writeDataByLocalIdentifier.ReqSId**[responseCode { refer to section 4.4 }]	**7F** **3B** xx
	return(main)		**return(responseCode)**	

7.6 WriteDataByCommonIdentifier service

The purpose of common identifiers is to reference a unique parameter which is understood in the same way by all servers (ECUs) supporting the identifier within a vehicle or at the vehicle manufacturer. In addition, some values of the parameter ***identificationOption*** shall be referenced by a ***recordCommonIdentifier*** by this service.

7.6.1 Message description

The writeDataByCommonIdentifier service is used by the client to write recordValues (data values) to a server. The data are identified by a recordCommonIdentifier. It is the vehicle manufacturer's responsibility that the server conditions are met when performing this service.
Possible uses for this service are:

- Clear non-volatile memory
- Reset learned values
- Set Vehicle Identification Number (VIN)
- Set option content

7.6.2 Message data bytes

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**writeDataByCommonIdentifier Request Service Id**	**M**	**2E**	**WDBCI**
#2	recordCommonIdentifier (High Byte) [refer to table 7.2.3]	M	xx	**RCI_...**
#3	recordCommonIdentifier (Low Byte)	M	xx	
#4	recordValue#1	M	xx	**RV_...**
:	**:**	**:**	**:**	**:**
#n	recordValue#m	M	xx	**RV_...**

Table 7.6.2.1 - WriteDataByCommonIdentifier Request Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**writeDataByCommonIdentifier Positive Response Service Id**	**M**	**6E**	**WDBCIPR**
#2	recordCommonIdentifier (High Byte) [refer to table 7.2.3]	M	xx	**RCI_...**
#3	recordCommonIdentifier (Low Byte)	M	xx	

Table 7.6.2.2 - WriteDataByCommonIdentifier Positive Response Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**negativeResponse Service Id**	**M**	**7F**	**NR**
#2	**writeDataByCommonIdentifier Request Service Id**	**M**	**2E**	**WDBCI**
#3	responseCode=[KWP2000ResponseCode { section 4.4 }]	M	xx=[00-FF]	**RC_...**

Table 7.6.2.3 - WriteDataByCommonIdentifier Negative Response Message

7.6.3 Parameter definition

- The parameter ***recordCommonIdentifier*** is defined in section 7.2.3.
- The parameter ***recordValue*** is defined in section 7.2.3.

7.6.4 Message flow example

See message flow examples of a Physically Addressed Service in section 5.3.1.1 and a Functionally Addressed Service in section 5.3.2.1.

7.6.5 Implementation example of writeDataByCommonIdentifier

7.6.5.1 Message flow conditions for "writeDataByCommonIdentifier"

The client (tester) shall command one or multiple server(s) (ECU(s)) to reset the learned values in non volatile memory by using the writeDataByCommonIdentifier service. The server(s) (ECU(s)) shall send a positive response message after it (they) has (have) successfully reset the non volatile memory.

7.6.5.2 Message flow

STEP#1 writeDataByCommonIdentifier(RCI_RLV)

time	Client (tester) Request Message	Hex
P3	**writeDataByCommonIdentifier.ReqSId**[	**2E**
	recordCommonIdentifier = resetLearnedValues (High Byte)	10
	recordCommonIdentifier = resetLearnedValues (Low Byte)]	01

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**writeDataByCommonIdentifier.PosRspSId**[	**6E**	**negativeResponse Service Identifier**	**7F**
	recordCommonIdentifier (High Byte)	10	**writeDataByCommonIdentifier.ReqSId**[	**2E**
	recordCommonIdentifier (Low Byte)]	01	responseCode { refer to section 4.4 }]	xx
	return(main)		**return(responseCode)**	

7.7 WriteMemoryByAddress service

7.7.1 Message description

The writeMemoryByAddress service is used by the client to write recordValues (data values) to a server. The data are identified by the server's memoryAddress and memorySize. It is the vehicle manufacturer's responsibility that the server conditions are met when performing this service.
Possible uses for this service are:

- Clear non-volatile memory
- Reset learned values
- Change calibration values

7.7.2 Message data bytes

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**writeMemoryByAddress Request Service Id**	**M**	**3D**	**WMBA**
#2	memoryAddress (High Byte)	M	xx	**MA_...**
#3	memoryAddress (Middle Byte)	M	xx	
#4	memoryAddress (Low Byte)	M	xx	
#5	memorySize	M	xx	**MS_...**
#6	recordValue#1	M	xx	**RV_...**
:	:	:	:	:
#n	recordValue#m	U	xx	**RV_...**

Table 7.7.2.1 - WriteMemoryByAddress Request Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**writeMemoryByAddress Positive Response Service Id**	**M**	**7D**	**WMBAPR**
#2	memoryAddress (High Byte)	M	xx	**MA_...**
#3	memoryAddress (Middle Byte)	M	xx	
#4	memoryAddress (Low Byte)	M	xx	

Table 7.7.2.2 - WriteMemoryByAddress Positive Response Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**negativeResponse Service Id**	**M**	**7F**	**NR**
#2	**writeMemoryByAddress Request Service Id**	**M**	**3D**	**WMBA**
#3	responseCode=[KWP2000ResponseCode { section 4.4 }]	M	xx=[00-FF]	**RC_...**

Table 7.7.2.3 - WriteMemoryByAddress Negative Response Message

7.7.3 Parameter definition

- The parameter ***memoryAddress (MA_)*** is defined in section 7.3.3.
- The parameter ***memorySize (MS_)*** is defined in section 7.3.3.
- The parameter ***recordValue (RV_)*** is defined in section 7.3.3.

7.7.4 Message flow example

See message flow example of Physical Addressed Service in section 5.3.1.1.

7.7.5 Implementation example of writeMemoryByAddress

7.7.5.1 Message flow conditions for “writeMemoryByAddress”

The service in this example is not limited by any restriction of the server (ECU). The client (tester) writes seven (7) data bytes to the server's (ECU's) serial EEPROM at the memory address $30FF13.

7.7.5.2 Message flow

STEP#1 writeMemoryByAddress(MA_..., MS_..., RV_#1 ... RV_#07)

time	Client (tester) Request Message	Hex
P3	**writeMemoryByAddress.ReqSId**[	**3D**
	memoryAddress { High Byte }	30
	memoryAddress { Middle Byte }	FF
	memoryAddress { Low Byte }	13
	memorySize	07
	recordValue#1	11
	recordValue#2	22
	recordValue#3	33
	recordValue#4	44
	recordValue#5	55
	recordValue#6	66
	recordValue#7]	77

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**writeMemoryByAddress.PosRspSId**	**7D**	**negativeResponse Service Identifier**	**7F**
	memoryAddress { High Byte }	30	**writeMemoryByAddress.ReqSId[**	**3D**
	memoryAddress { Middle Byte }	FF	responseCode { refer to section 4.4 }]	xx
	memoryAddress { Low Byte }	13		
	return(main)		**return(responseCode)**	

7.8 SetDataRates service

This service is not part of SSF 14230-3 and shall therefore not be implemented!
For support of repeated responses from the server, see description of ***Periodic Transmission*** in section 6.1 and in SSF 14230-2.

7.9 StopRepeatedDataTransmission service

This service is not part of SSF 14230-3 and shall therefore not be implemented!
For support of repeated responses from the server, see description of ***Periodic Transmission*** in section 6.1 and in SSF 14230-2.

8 Stored Data Transmission functional unit

The services provided by this functional unit are described in table 8:

Service name	Description
ReadDiagnosticTroubleCodes	**This service is not part of SSF 14230-3 and shall therefore not be implemented!**
ReadDiagnosticTroubleCodesBy-Status	The client requests from the server the transmission of both, number of DTC and values of the diagnostic trouble codes depending on their status.
ReadStatusOfDiagnosticTrouble-Codes	The client requests from the server the transmission of the number of DTC, values and status of the diagnostic trouble codes.
ReadFreezeFrameData	The client requests from the server the transmission of the value of a record stored in a freeze frame.
ClearDiagnosticInformation	The client requests from the server to clear all or a group of the diagnostic information stored.

Table 8 - Stored Data Transmission functional unit

8.1 ReadDiagnosticTroubleCodes service

This service is not part of SSF 14230-3 and shall therefore not be implemented!
The purpose of the service readDiagnosticTroubleCodes is to read diagnostic trouble codes from the server's (ECU's) memory. This service is not implemented because the same information, besides additional diagnostic trouble codes information (e.g. statusOfDTC), is contained in the service readDiagnosticTroubleCodesByStatus.

8.2 ReadDiagnosticTroubleCodesByStatus service

8.2.1 Message description

The readDiagnosticTroubleCodesByStatus service is used by the client (tester) to read diagnostic trouble codes by status from the server's (ECU's) memory. The server(s) (ECU(s)) shall report DTC(s) with status information, selected by the groupOfDTC parameter value sent by the client (tester).

The DTCs shall be reported by the server(s) (ECU(s)) in the same sequence as they have been identified/detected.
A DTC shall be reported by the server(s) (ECU(s)) only once in the same response message.

8.2.2 Message data bytes

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**readDiagnosticTroubleCodesByStatus Request SId**	**M**	**18**	**RDTCBS**
#2	statusOfDTCRequest	M	xx	**SODTCRQ_...**
#3 #4	groupOfDTC (High Byte) groupOfDTC (Low Byte)	M M	xx xx	**GODTC_...**

Table 8.2.2.1 - ReadDiagnosticTroubleCodesByStatus Request Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**readDiagnosticTroubleCodesByStatus Pos. Response SId**	**M**	**58**	**RDTCBSPR**
#2	numberOfDTC	M	xx	**NRODTC_...**
 #3 : : : : : #n	listOfDTCAndStatus=[DTC#1 (High Byte) DTC#1 (Low Byte) statusOfDTC#1 : DTC#m (High Byte) DTC#m (Low Byte) statusOfDTC#m]	C	 xx xx xx : xx xx xx	**LODTCAS_...** **DTC_** **SODTC_** **:** **DTC_** **SODTC_**

Table 8.2.2.2 - ReadDiagnosticTroubleCodesByStatus Positive Response Message

C = condition: listOfDTCAndStatus is only present if numberOfDTC is > "$00".

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**negativeResponse Service Id**	**M**	**7F**	**NR**
#2	**readDiagnosticTroubleCodesByStatus Request SId**	**M**	**18**	**RDTCBS**
#3	responseCode=[KWP2000ResponseCode { section 4.4 }]	M	xx=[00-FF]	**RC_...**

Table 8.2.2.3 - ReadDiagnosticTroubleCodesByStatus Negative Response Message

8.2.3 Parameter definition

8.2.3.1 Request parameter definition

- The parameter ***statusOfDTCRequest*** is used by the client (tester) in the request message to select diagnostic trouble codes (DTCs) to be included in the response message based on their status information. The parameter *statusOfDTCRequest* is also used by the client (tester) to request information about which status bits are supported for all trouble codes in a server (ECU).

Hex	Description	Cvt	Mnemonic
00 - 01	**reservedByDocument** This range of values is reserved by this document for future definition.	**M**	**RBD**
02	**requestStoredDTCAndStatus** This value is used by the client (tester) to indicate to the server(s) (ECU(s)) that the server(s) (ECU(s)) shall respond with all diagnostic trouble codes (DTCs) which have been validated and stored in non volatile memory . Status bit 5 = ‘1’ (if used).	**M**	**RSDTCAS**

Hex	Description	Cvt	Mnemonic
03	**requestAllDTCAndStatus** This value is used by the client (tester) to indicate to the server(s) (ECU(s)) that the server(s) (ECU(s)) shall respond with all supported diagnostic trouble codes (DTCs), regardless of their status.	U	RADTCAS
04 - 10	**reservedByDocument** This range of values is reserved by this document for future definition.	M	RBD
11	**requestPendingDTCAndStatus** This value is used by the client (tester) to indicate to the server(s) (ECU(s)) that the server(s) (ECU(s)) shall respond with all diagnostic trouble codes (DTCs) which have been pending present during this driving cycle. Status bit 1 = ‘1’ (if used).	U	RPDTCAS
12 - EF	**reservedByDocument** This range of values is reserved by this document for future definition.	M	RBD
F0 - F9	**vehicleManufacturerSpecific** This range of values is reserved for vehicle manufacturer specific use.	U	VMS
FA - FE	**systemSupplierSpecific** This range of values is reserved for system supplier specific use.	U	SSS
FF	**requestStatusBitsSupported** This value is used by the client (tester) to indicate to the server(s) (ECU(s)) that the server(s) (ECU(s)) shall respond with one diagnostic trouble code and a corresponding status byte where the content of the status byte shows which status bits are supported for all diagnostic trouble code in this server (ECU). If a specific status bit is set to ‘0’ (zero) it means that this status bit is not supported for all diagnostic trouble codes in this server (ECU). If a specific status bit is set to ‘1’ (one) it means that this status bit is supported for all diagnostic trouble codes in this server (ECU).	M	RSBS

Table 8.2.3.1.1 - Definition of statusOfDTCRequest values

- The parameter ***groupOfDTC (GODTC_)*** is used by the client (tester) to define a group of DTCs that the server (ECU) shall search for the requested DTCs. This parameter is used by the client (tester) primarily to select a specific DTC. It may also be used by the client (tester) to select a functional group of DTCs. In this case the format of this parameter is vehicle manufacturer specific. Standardised DTC groups and DTCs formats that can be used are specified in SAE J2012 for passenger cars and in SAE J1587 and SAE J1939 for heavy duty vehicles.

Hex	Description	Cvt	Mnemonic
0000 - FFFE	**specificDTC** This range of values is used by the client (tester) to select a specific DTC. The parameter value shall then be equal to the DTC. It may also be used to select a functional group of DTCs. In this case the format of this parameter is vehicle manufacturer specific	U	SDTC
FFFF	**allDTCs** This value is used by the client (tester) to indicate to the server(s) (ECU(s)) that the selected group of DTCs shall be all DTCs supported by the server(s).	M	ADTC

Table 8.2.3.1.2 - Definition of groupOfDTC values

8.2.3.2 Response parameter definition

- The parameter ***numberOfDTC (NRODTC_)*** is used by the readDiagnosticTroubleCodesByStatus and readStatusOfDiagnosticTroubleCodes services to indicate how many DTCs, of the group specified in the request, are reported by the server(s) ECU(s).

- The parameter ***listOfDTCAndStatus (LODTCAS_)*** is used by the readDiagnosticTroubleCodesByStatus **positive response message** to report diagnostic trouble codes (DTCs) and status information. The listOfDTCAndStatus parameter is of the type "record" and may include multiple DTCs with status information. If no DTC, with status information that match the value of the statusOfDTCRequest parameter at the time of the request, the server(s) (ECU(s)) shall not include the parameter listOfDTCAndStatus in the positive response message.

- The parameter ***DTC (DTC_)*** is used by the server (ECU) to report system failures by a two (2) byte Diagnostic Trouble Code number. The format of a DTC is specified by the vehicle manufacturer.
  Standardised DTCs and DTCs formats that can be used are specified in SAE J2012 for passenger cars and in SAE J1587 and SAE J1939 for heavy duty vehicles.

- The parameter ***statusOfDTC (SODTC_)*** is used by the server(s) (ECU(s)) in the positive response message to provide status information for each DTC. The value of this parameter shall always be the current status of the DTC at the time of the request.

Bit	Description of statusOfDTC
0	**pendingFaultPresent** This bit indicates that the fault indicated by this DTC is currently present in the server (ECU). **'0' = pending fault is not present** **'1' = pending fault is present** The pendingFaultPresent bit shall remain in the ‘1’ state until the fault is not present. When the driving cycle is finished or after a clearDiagnosticInformation service the pendingFaultPresent bit shall be set to '0'.
1	**pendingFaultState** This bit indicates that the pending fault indicated by this DTC has been present (bit 0 has been set) at least once during the current driving cycle. **'0' = pending fault has not been present during this driving cycle** **'1' = pending fault has been present during this driving cycle** The pendingFaultState bit shall be set to ‘1’ when the pendingFaultPresent bit (bit 0) changes from state ‘0’ to ‘1’ for the first time and shall remain in that state until the driving cycle is finished. When the driving cycle is finished or after a clearDiagnosticInformation service the pendingFaultState bit shall be set to '0'.
2	**testRunning** This bit indicates whether the diagnostic trouble code test conditions are met at the time of request. **'0' = test is not running** **'1' = test is running**
3	**testInhibit** This bit indicates that the diagnostic trouble code test conditions are not met because of another fault in the server (ECU). The pendingFaultPresent bit (bit 0) and validatedFaultPresent bit (bit 6) shall be set to ‘0’ and remain in that state as long as the test is inhibited. **'0' = test is not inhibited by other DTC** **'1' = test is inhibited by other DTC**
4	**testReadiness** This bit indicates that the diagnostic trouble code test has not yet been completed during the current driving cycle. **'0' = test has been completed** **'1' = test has not been completed** When the driving cycle is finished or after a clearDiagnosticInformation service the testReadiness bit shall be set to '1'.
5	**DTCStorageState** This bit indicates that the fault indicated by this DTC has been validated (bit 6 has been set) at least once and stored in non volatile memory since this DTC was last cleared. **‘0’ = DTC not validated and not stored in non volatile memory** **‘1’ = DTC validated and stored in non volatile memory.** After a clearDiagnosticInformation service the DTCStorageState bit shall be set to '0'.
6	**validatedFaultPresent** This bit indicates that the fault indicated by this DTC is currently validated in the server (ECU). **‘0’ = no validated fault present or fault has been validated as not present at time of request.** **‘1’ = validated fault present at time of request.** When the driving cycle is finished or after a clearDiagnosticInformation service the validatedFaultPresent bit shall be set to '0'.

Bit	Description of statusOfDTC
7	**validatedFaultState** This bit indicates that the fault indicated by this DTC has been validated (bit 6 has been set) at least once during the current driving cycle. **'0' = validated fault has not been present during this driving cycle** **'1' = validated fault has been present during this driving cycle** The validatedFaultState bit shall be set to ‘1’ when the validatedFaultPresent bit (bit 6) changes from state ‘0’ to ‘1’ for the first time and shall remain in that state until the driving cycle is finished. When the driving cycle is finished or after a clearDiagnosticInformation service the validatedFaultPresent bit shall be set to '0'.
NOTE	**Driving cycle starts when starter key is turned to driving position and stops when it is turned off.**

Table 8.2.3.2.1 - Definition of statusOfDTC values

The state diagram of the ***statusOfDTC*** shows the possible transitions between different statuses. Bits 2, 3 and 4 are not included in this diagram.

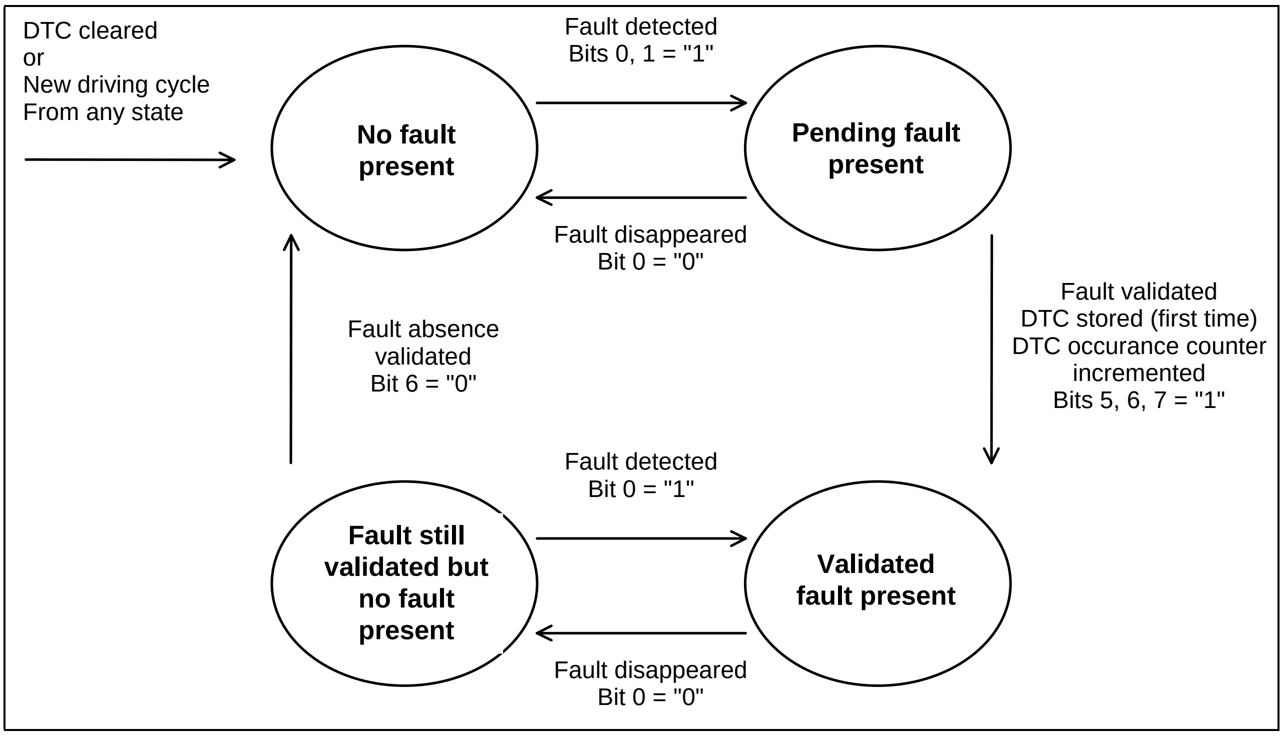


Figure 8.2.3.2.1 - State diagram for StatusOfDTC

The figure below shows a diagram which specifies an example of how the Diagnostic Trouble Code (DTC) status logic shall be implemented in a server (ECU). The diagram is splitted into thirteen (13) columns (steps). The step width is a measure of time. In this example, the criteria for validation of a fault is that it is “Pending Present” for two consecutive time steps. The criteria for a fault to be validated not present is that it is absent for two consecutive time steps. All bits are up-dated in the beginning of each time step (column).

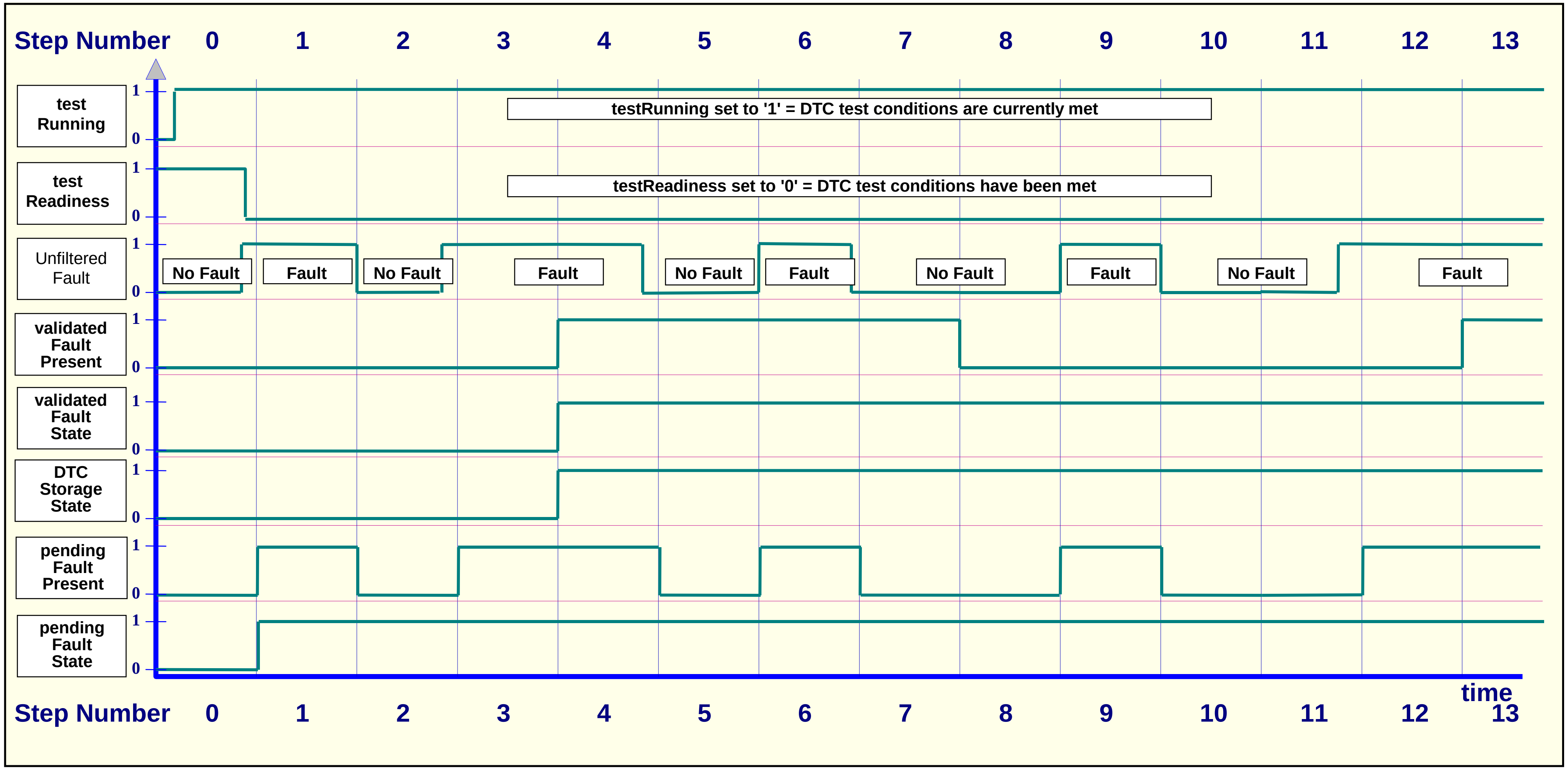


Figure 8.2.3.2.2 - Diagnostic Trouble Code Status

8.2.4 Message flow example

See message flow example of Physically Addressed Service in section 5.3.1.1 and Functionally Addressed Service in section 5.3.2.1.

8.2.5 Implementation example of readDiagnosticTroubleCodesByStatus

8.2.5.1 - Test specification readDiagnosticTroubleCodesByStatus

The following example describes a sequence of diagnostic trouble code occurence of a "Powertrain" system. In this example the parameter statusOfDTCRequest=$02 (requestStoredDTCAndStatus) and the paramter groupOFDTC=$0000 (representing the functional group of DTCs “Powertrain DTCs”)

The DTCs stored in the server's (ECU's) memory are specified below:

DTC#1: P0130: O2 Sensor Circuit Malfunction (Bank 1, Sensor 1)

statusOfDTC#1 = $A7:	testReadiness = '0b'	{ test has been completed }
	DTCStorageState = '1b'	{ DTC stored in non volatile memory }
	validatedFaultPresent = '0b'	{ fault not present at time of request}

DTC#2: P0120: Throttle Position Circuit Malfunction

statusOfDTC#2 = $E7:	testReadiness = '0b'	{ test has been completed }
	DTCStorageState = '1b'	{ DTC stored in non volatile memory }
	validatedFaultPresent = '1b'	{ fault present at time of request }

DTC#3: P0135: O2 Sensor Heater Circuit Malfunction (Bank 1, Sensor 1)

statusOfDTC#3 = $14:	testReadiness = '1b'	{ test has not been completed }
	DTCStorageState = '0b'	{ DTC not stored in non volatile memory }
	validatedFaultPresent = '0b'	{ fault not present at time of request }

8.2.5.2 - Message flow

8.2.5.2.1 - Message flow of readDiagnosticTroubleCodeByStatus (requestStoredDTCAndStatus)

Note: DTC#3 is not included in the positive response message because the DTCStorageState bit is still set to "DTC not stored in non volatile memory"!

STEP#1 readDiagnosticTroubleCodesByStatus(SODTC-RT, GODTC_PG)

time	Client (tester) Request Message	Hex
P3	**readDiagnosticTroubleCodeByStatus.ReqSId**[	**18**
	statusOfDTC { requestStoredDTCAndStatus }	02
	groupOfDTC { Powertrain High Byte }	00
	groupOfDTC { Powertrain Low Byte }]	00

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**readDTCByStatus.PosRspSId**[	**58**	**negativeResponse Service Identifier**	**7F**
	numberOfDTC	02	**readDTCByStatus.ReqSId**[	**18**
	DTC#1 { High Byte } {O2 Sensor Circuit Malf.}	01	responseCode { refer to section 4.4 }]	xx
	DTC#1 { Low Byte } {Bank 1, Sensor 1}	30		
	statusOfDTC#1	A7		
	DTC#2 { High Byte } {Throttle Position Malf.}	01		
	DTC#2 { Low Byte } {Throttle Position Malf.}	20		
	statusOfDTC#2	E7		
	return(main)		**return(responseCode)**	

8.2.5.2.2 Message flow of readDiagnosticTroubleCodeByStatus (requestAllDTCAndStatus)

The following 19 DTCs are supported by the server (ECU) as specified below:

DTC#1:	$0001:	ECU-specific DTC 1
DTC#2:	$0002:	ECU-specific DTC 2
DTC#3:	$0003:	ECU-specific DTC 3
DTC#4:	$0004:	ECU-specific DTC 4
DTC#5:	$0005:	ECU-specific DTC 5
DTC#6:	$0006:	ECU-specific DTC 6
DTC#7:	$0007:	ECU-specific DTC 7
DTC#8:	$0008:	ECU-specific DTC 8
DTC#9:	$0009:	ECU-specific DTC 9
DTC#10:	$000A:	ECU-specific DTC 10
DTC#11:	$000B:	ECU-specific DTC 11
DTC#12:	$000C:	ECU-specific DTC 12
DTC#13:	$000D:	ECU-specific DTC 13
DTC#14:	$000E:	ECU-specific DTC 14
DTC#15:	$000F:	ECU-specific DTC 15
DTC#16:	$0010:	ECU-specific DTC 16
DTC#17:	$0011:	ECU-specific DTC 17
DTC#18:	$0012:	ECU-specific DTC 18
DTC#19:	$0013:	ECU-specific DTC 19

statusOfDTC#1 - 19 = $xx

STEP#1 readDiagnosticTroubleCodesByStatus(SODTC-RT, GODTC_PG)

time	Client (tester) Request Message	Hex
P3	**readDiagnosticTroubleCodeByStatus.ReqSId**[	**18**
	statusOfDTC { requestAllDTCAndStatus}	03
	groupOfDTC { AllDTCs High Byte }	FF
	groupOfDTC { AllDTCs Low Byte }]	FF

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**readDTCByStatus.PosRspSId**[	**58**	**negativeResponse Service Identifier**	**7F**
	numberOfDTC	13	**readDTCByStatus.ReqSId**[	**18**
	DTC#1 { High Byte }	00	responseCode { refer to section 4.4 }]	xx
	DTC#1 { Low Byte }	01		
	statusOfDTC#1	**xx**		
	DTC#2 { High Byte }	00		
	DTC#2 { Low Byte }	02		
	statusOfDTC#2	xx		
	DTC#3 { High Byte }	00		
	DTC#3 { Low Byte }	03		
	statusOfDTC#3	**xx**		
	DTC#4 { High Byte }	00		
	DTC#4 { Low Byte }	04		
	statusOfDTC#4]	xx		
	:	:		
	:	:		
	DTC#19 { High Byte }	00		
	DTC#19 { Low Byte }	13		
	statusOfDTC#19]	xx		
	return(main)		**return(responseCode)**	

8.2.5.2.3 - Message flow of readDTCByStatus with server (ECU) transmit data segmentation

This message flow specifies two (2) different methods of implementations in the server (ECU) in case the number of DTCs to report to the client (tester) is greater than the transmit buffer in the server (ECU). In such case the server (ECU) shall use one of the following described methods:

Method #1: Data Segmentation

The client (tester) recognizes that the numberOfDTC parameter includes a greater value than the actual amount of data bytes (DTCs) included in the positive response message. The client (tester) receives multiple positive response messages with the same service identifier value and if a three (3) byte header is used it shall also look for the target address. The client (tester) shall concatenuate all positive response messages received which meet above described critera as one positive response message.

In this example the server (ECU) has a limited transmit buffer of fifteen (15) data bytes only. The server (ECU) supports sixteen (16) DTCs which add up to a total of 50 data bytes (16 * 3 + 2) if transmitted in one response message.

STEP#1 readDiagnosticTroubleCodesByStatus(SODTC-RT, GODTC_PG)

time	Client (tester) Request Message	Hex
P3	**readDiagnosticTroubleCodeByStatus.ReqSId**[	**18**
	statusOfDTC { requestAllDTCAndStatus }	03
	groupOfDTC { Powertrain High Byte }	00
	groupOfDTC { Powertrain Low Byte }]	00

time	Server (ECU) Positive Response Message	Hex		Server (ECU) Negative Response Message	Hex
P2	**readDTCByStatus.PosRspSId**[	**58**		**negativeResponse Service Identifier**	**7F**
	numberOfDTC	10		**readDTCByStatus.ReqSId**[	**18**
	DTC#1 { High Byte }	xx		responseCode { refer to section 4.4 }]	xx
	DTC#1 { Low Byte }	xx			
	statusOfDTC#1	xx			
	:	:			
	DTC#5 { High Byte }]	xx			
	goto(STEP#2)			**return(responseCode)**	

STEP#2 readDiagnosticTroubleCodesByStatusPositiveresponse(...)

time	Server (ECU) Positive Response Message	Hex
P2	**readDTCByStatus.PosRspSId**[	**58**
	DTC#5 { Low Byte }	xx
	statusOfDTC#5	xx
	:	:
	DTC#9 { High Byte }	xx
	DTC#9 { Low Byte }	xx
	statusOfDTC#9]	xx
	goto(STEP#3)	

Negative Response Message Not Possible!

STEP#3 readDiagnosticTroubleCodesByStatusPositiveresponse(...)

time	Server (ECU) Positive Response Message	Hex
P2	**readDTCByStatus.PosRspSId**[	**58**
	DTC#10 { High Byte }	xx
	DTC#10 { Low Byte }	xx
	statusOfDTC#10	xx
	:	:
	DTC#14 { High Byte }	xx
	DTC#14 { Low Byte }]	xx
	goto(STEP#4)	

Negative Response Message Not Possible!

STEP#4 readDiagnosticTroubleCodesByStatusPositiveresponse(...)

time	Server (ECU) Positive Response Message	Hex
P2	**readDTCByStatus.PosRspSId**[	**58**
	statusOfDTC#14	xx
	DTC#15 { High Byte }	xx
	DTC#15 { Low Byte }	xx
	statusOfDTC#15	xx
	DTC#16 { High Byte }	xx
	DTC#16 { Low Byte }	xx
	statusOfDTC#16]	xx
	return(main)	

Negative Response Message Not Possible!

Method #2: On line transmit buffer refill during transmission
The server (ECU) has more DTCs to report than the size of the transmit buffer. To avoid data segmentation the server (ECU) shall start sending the positive response message while adding more DTCs to the transmit buffer on the fly. This shall result in one (1) positive response message of the server (ECU).

STEP#1 readDiagnosticTroubleCodesByStatus(SODTC-RT, GODTC_PG)

time	Client (tester) Request Message	Hex
P3	**readDiagnosticTroubleCodeByStatus.ReqSId**[	**18**
	statusOfDTC { requestAllDTCAndStatus }	03
	groupOfDTC { Powertrain High Byte }	00
	groupOfDTC { Powertrain Low Byte }]	00

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**readDTCByStatus.PosRspSId**[	**58**	**negativeResponse Service Identifier**	**7F**
	numberOfDTC	10	**readDTCByStatus.ReqSId**[	**18**
	DTC#1 { High Byte }	xx	responseCode { refer to section 4.4 }]	xx
	DTC#1 { Low Byte }	xx		
	statusOfDTC#1	xx		
	:	**:**		
	DTC#16 { High Byte }	xx		
	DTC#16 { Low Byte }	xx		
	statusOfDTC#16]	xx		
	return(main)		**return(responseCode)**	

8.3 ReadStatusOfDiagnosticTroubleCodes service

8.3.1 Message description

The readStatusOfDiagnosticTroubleCodes service is used by the client to read stored diagnostic trouble codes with their associated status from the server's memory.
If the server(s) does not (do not) have any DTC with status information stored it (they) shall set the parameter numberOfDTC to noDTCStored ($00). This causes the server(s) not to include DTC(s) and status information in the parameter listOfDTC in the positive response message.
This service shall be used to report DTC location and symptom(s) (environmental parameter conditions when the DTC is identified) as systemSupplierData.
If the client (tester) requests the status of a DTC which is unknown to the server (ECU) the server (ECU) shall send a negative response message with the appropriate negative response code.

8.3.2 Message data bytes

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**readStatusOfDiagnosticTroubleCodes Request SId**	**M**	**17**	**RSODTC**
#2	groupOfDTC { High Byte }	M	xx	**GODTC_...**
#3	groupOfDTC { Low Byte }	M	xx	

Table 8.3.2.1 - ReadStatusOfDiagnosticTroubleCodes Request Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**readStatusOfDiagnosticTroubleCodes Positive Response SId**	**M**	**57**	**RSODTCPR**
#2	numberOfDTC	M	n	**NRODTC_**
	listOfDTCAndStatus=[			**LODTCAS_**
#3	DTC#1 { High Byte }	M	xx	**DTC_**
#4	DTC#1 { Low Byte }	M	xx	
#5	statusOfDTC	M	xx	**SODTC_**
#6	systemSupplierData#1.1	U	xx	**SSD**
:	:	:	:	:
:	systemSupplierData#1.m	:	:	:
:	DTC#2 { High Byte }	:	:	:
:	DTC#2 { Low Byte }	:	:	:
:	statusOfDTC	:	:	:
:	systemSupplierData#2.1	:	:	:
:	:	:	:	:
:	systemSupplierData#2.m]	:	:	:
:		:	:	:
:	...	:	:	:
:		:	:	:
:	DTC#n { High Byte }	:	:	:
:	DTC#n { Low Byte }	:	:	:
:	statusOfDTC	:	:	:
:	systemSupplierData#n.1	:	:	:
:	:	:	:	:
#j	systemSupplierData#n.m]	U	xx	**SSD**

Table 8.3.2.2 - ReadStatusOfDiagnosticTroubleCodes Positive Response Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**negativeResponse Service Id**	**M**	**7F**	**NA**
#2	**readStatusOfDiagnosticTroubleCodes Request SId**	**M**	**17**	**RSODTC**
#3	responseCode=[KWP2000ResponseCode { section 4.4 }]	M	xx=[00-FF]	**RC_...**

Table 8.3.2.3 - ReadStatusOfDiagnosticTroubleCodes Negative Response Message

8.3.3 Parameter definition

- The parameter ***groupOfDTC (GODTC_)*** is defined in section 8.2.3.

- The parameter ***numberOfDTC (NRODTC_)*** is defined in section 8.2.3.
- The parameter ***listOfDTCAndStatus (LODTCAS_)*** shall be defined by the system supplier and the vehicle manufacturer. This parameter shall report diagnostic trouble codes (DTCs) and status information for those DTCs which have been validated and stored in non volatile memory at the time of the request, independent of their status.
- The parameter ***DTC (DTC_)*** is defined in section 8.2.3.
- The parameter s***tatusOfDTC (SODTC_)*** is defined in section 8.2.3.
- The parameter ***systemSupplierData (SSD_)*** is used by the server (ECU) to report system specific information. Format of this parameter is system supplier and/or vehicle manufacturer specific. The number of systemSupplierData bytes must be the same for all DTCs within the same response message (it may vary between responses). Not used bytes shall be padded with "dummy" values.

  It is recommended that ***SystemSupplierData#n.1*** shall be the occurrence counter for the specific DTC. The occurrence counter contains the number of times a fault has gone from inactive to validated present.

8.3.4 Message flow example

See message flow example of Physically Addressed Service in section 5.3.1.1 and Functionally Addressed Service in section 5.3.2.1.

8.3.5 Implementation example of readStatusOfDiagnosticTroubleCodes

8.3.5.1 Test specification readStatusOfDiagnosticTroubleCodes

In this example the parameter groupOFDTC=$0000 (representing the specific DTC P0120 "Throttle Position Circuit Malfunction").

The server (ECU) has stored the DTC with the following statusOfDTC systemSupplierSpecific data:

DTC#n: P0120: Throttle Position Circuit Malfunction

statusOfDTC#n = $E7:	testReadiness = '0b'	{ test has been completed }
	DTCStorageState = '1b'	{ DTC stored in non volatile memory }
	validatedFaultPresent = '1b'	{ fault present at time of request }
systemSupplierData#n.1:	DTC Occurrence Counter	{ $07 = 7 }
systemSupplierData#n.2, n.3:	Environm. Cond.#1: Engine Speed	{ $2648 = 2450 r/min }
systemSupplierData#n.4:	Environm. Cond.#2: Vehicle Speed	{ $46 = 75 km/h }

8.3.5.2 Message flow

STEP#1 readStatusOfDiagnosticTroubleCodes(GODTC_DTC "P0120")

time	Client (tester) Request Message	Hex
P3	**readStatusOfDiagnosticTroubleCode.ReqSId**[	**17**
	groupOfDTC { Throttle Position Malf. High Byte }	01
	groupOfDTC { Throttle Position Malf. Low Byte }]	20

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**readStatusOfDTC.PosRspSId**[	**57**	**negativeResponse Service Identifier**	**7F**
	numberOfDTC	01	**readStatusOfDTC.ReqSId[**	**17**
	DTC#1 { High Byte } {Throttle Position Malf.}	01	responseCode { refer to section 4.4 }]	xx
	DTC#1 { Low Byte } {Throttle Position Malf.}	20		
	statusOfDTC	E2		
	systemSupplierData#1.1 { Occurrence Counter=7}	07		
	systemSupplierData#1.2 {Engine Speed=2450 r/min}	26		
	systemSupplierData#1.3 {Engine Speed=2450 r/min}	48		
	systemSupplierData#1.4 {Vehicle Speed=75 km/h}]	46		
	return(main)		**return(responseCode)**	

8.4 ReadFreezeFrameData service

8.4.1 Message description

Freeze Frames are specific data records stored in the server's memory. The content of the freeze frames is not defined by this standard, but typical usage of freeze frames is to store data upon detection of a system malfunction. Multiple freeze frames may be stored before and/or after the malfunction has been detected.

The readFreezeFrameData service is used by the client to read the freeze frame data stored in the server's memory. The parameter freezeFrameNumber identifies the respective freeze frame if one or multiple freeze frames are stored in the server's memory.

It is possible to use information from data scaling tables to interpret the content of a freeze frame. In such case it must be possible to identify what data is included in the freeze frame. The information about what kind of data can either be included directly in the freeze frame data or be retrieved from a freeze frame data structure table.

The following example, shown in the four figures below, illustrates how freeze frames can be created and stored in the server's memory: The first figure shows a typical scenario where malfunctions are detected and DTCs with corresponding freeze frame data are stored. The following three figures give different examples on how data can be stored in the server (ECU) memory. The amount of data stored in the server (ECU) memory is depending on how much information the client (tester) has to interpret the freeze frame data. The more information known in advance by the client (tester), the less data is needed to be stored in the server (ECU) memory.

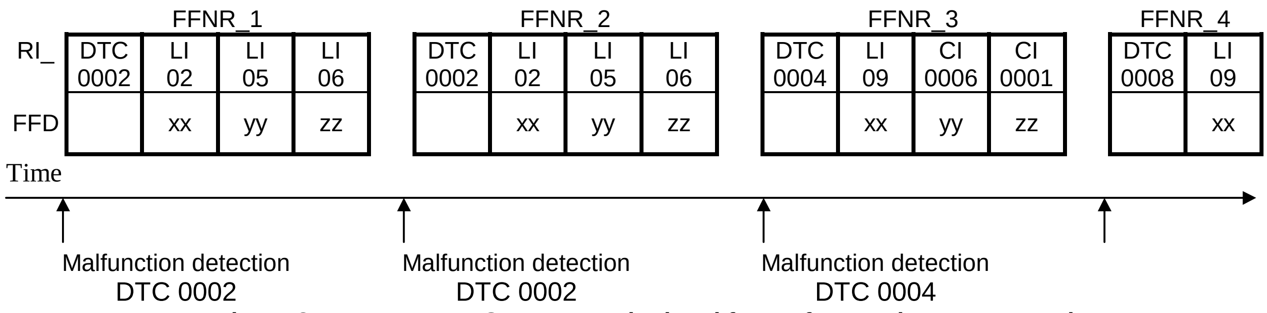


Figure 8.4.1.1 - How DTCs occur and related freeze frames data are created.

The following figure shows an example where only the stored data is included in the freeze frame data. There is no information included to identify what kind of data has been stored. One method to interpret this data is to retrieve information from a freeze frame data structure table.

FFNR_	DTC	Freeze Frame Data Record
0 (OBDII)	P0130	xx yyy zz ...
1	0002	xx yyy zz ...
2	0002	xx yyy zz ...
3	0004	xx yyy zz ...
4	0008	xx yyy zz ...

Figure 8.4.1.2 - Freeze frame data only stored in the server memory.

The following figure shows an example where the stored data is preceded by an identifier value to help interpret the stored data.

FFNR_	DTC	Freeze Frame Data Record
0 (OBDII)	P0130	xx yyy zz ...
1	0002	02 xx 05 yyy 06 zz ...
2	0002	02 xx 05 yyy 06 zz ...
3	0004	09 xx 0006 yyy 0001 zz ...
4	0008	09 xx ...

Figure 8.4.1.3 - Freeze frame data and identifier values stored in the server memory.

The following figure shows an example where the stored data is preceded by both the type and value of the identifier for the data.

FFNR_	DTC	Freeze Frame Data Record
0 (OBDII)	P0130	xx yyy zz ...
1	0002	LI 02 xx LI 05 yyy LI 06 zz ...
2	0002	LI 02 xx LI 05 yyy LI 06 zz ...
3	0004	LI 09 xx CI 0006 yyy CI 0001 zz ...
4	0008	LI 09 xx ...

Figure 8.4.1.4 - Freeze frame data, identifier values and type stored in the server memory.

The recordAccessMethodIdentifier in the request message identifies the access method used by the client (tester) to request freezeFrameData from the server's (ECU's) memory. In addition, the parameter recordIdentification shall always be used in conjunction with the recordAccessMethodIdentifier.
The server (ECU) shall send a positive response message including the freezeFrameNumber, freezeFrameData and the conditional parameters recordAccessMethodIdentifier and recordIdentification. If the freezeFrame defined by SAE J1979 (OBDII) is stored by the server (ECU) the freezeFrameNumber shall be of the value '$00'.

8.4.2 Message data bytes

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**readFreezeFrameData Request Service Id**	**M**	**12**	**RFFD**
#2	freezeFrameNumber [refer to table 8.4.3.1]	M	xx	**FFNR_...**
#3	recordAccessMethodIdentifier [refer to table 8.4.3.2]	M	xx	**RAMI_...**
#4 #5	recordIdentification { High byte } [refer to table 8.4.3.3] recordIdentification { Low byte } [refer to table 8.4.3.3]	C1/C2 C2	xx xx	**RI_...**

Table 8.4.2.1 - ReadFreezeFrameData Request Message

Conditions:
C1: present if recordIdentification size = 1 byte (refer to table 8.4.3.4)
C2: present if recordIdentification size = 2 byte (refer to table 8.4.3.4)
C1/C2: not present if recordAccessMethodIdentifier = requestAllData
C1/C2: not present if recordAccessMethodIdentifier = DTCThatCausedFreezeFrameStorage

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**readFreezeFrameData Positive Response Service Id**	**M**	**52**	**RFFDPR**
#2	freezeFrameNumber [refer to table 8.4.3.1]	M	xx	**FFNR_...**
#3 : #k	freezeFrameData#1 : freezeFrameData#m	C3 : C3	xx : xx	**FFD** **:** **FFD**
#k+1	recordAccessMethodIdentifier [refer to table 8.4.3.2]	M	xx	**RAMI_...**
#k+2 #k+3	recordIdentification { High byte } [refer to table 8.4.3.3] recordIdentification { Low byte } [refer to table 8.4.3.3]	C1/C2 C2	xx xx	**RI_...**

Table 8.4.2.2 - ReadFreezeFrameData Positive Response Message

Conditions:
C1: present if recordIdentification size = 1 byte (refer to table 8.4.3.4)
C2: present if recordIdentification size = 2 byte (refer to table 8.4.3.4)
C3: not present if recordAccessMethodIdentifier = DTCThatCausedFreezeFrameStorage

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**negativeResponse Service Id**	**M**	**7F**	**NR**
#2	**readFreezeFrameData Request Service Id**	**M**	**12**	**RFFD**
#3	responseCode=[KWP2000ResponseCode { section 4.4 }]	M	xx=[00-FF]	**RC_...**

Table 8.4.2.3 - ReadFreezeFrameData Negative Response Message

8.4.3 Parameter definition

- The parameter ***freezeFrameNumber (FFNR_)*** is used by the client to identify the freeze frame to be read from the server's memory.

Hex	Description	Cvt	Mnemonic
00	**OBDIIFreezeFrame** This value references the freeze frame defined in SAE J1979	U	OFFDR
01 - FE	**freezeFrame** This value references a specific freeze frame in the server's (ECU's) memory.	U	FFDR
FF	**allFreezeFrames** This value references all freeze frames in the server's (ECU's) memory. Each freeze frame shall be sent as a separate positive response message.	U	AFFDR

Table 8.4.3.1 - Definition of the freezeFrameNumber parameter

- The parameter ***recordAccessMethodIdentifier (RAMI_)*** is used by the client to define the access method which shall be used in the recordIdentification parameter to access a specific data within the freeze frame.

Hex	Description	Cvt	Mnemonic
00	**requestAllData** This value indicates to the server(s) (ECU(s)) that the client (tester) requests all data from a freeze frame stored within the server's (ECU's) memory.	U	RAD
01	**requestByRecordLocalIdentifier** This value indicates to the server that the client (tester) requests freeze frame data identified by a local identifier.	U	RBRLI
02	**requestByRecordCommonIdentifier** This value indicates to the server(s) (ECU(s)) that the client (tester) requests freeze frame data identified by a common identifier.	U	RBRCI
03	**requestByMemoryAddress** This parameter is not part of SSF 14230-3 and shall therefore not be implemented!	N/A	RBMA
04	**requestByDTC** This value indicates to the server (ECU) that the client (tester) requests freeze frames identified by a DTC.	U	RBDTC
05 - 7F	**reservedByDocument** This range of values is reserved by this document for future definition.		RBD
80	**DTCThatCausedFreezeFrameStorage** This value indicates to the server (ECU) that the client (tester) requests the DTC that caused the freeze frame storage.	U	DTCTC FFS
81	**requestByInputOutputLocalIdentifier** This value indicates to the server that the client (tester) requests freeze frame data identified by an InputOutputControlByLocalIdentifier.	U	RBIOLI
82	**requestByInputOutputCommonIdentifier** This value indicates to the server that the client (tester) requests freeze frame data identified by an InputOutputConntrolByCommonIdentifier.	U	RBIOCI
83	**requestFreezeFrameDataStructure** This value shall cause the server to report the Freeze Frame Data structure table identified by the DTC that will cause the storage of the corresponding freeze frame data. The Freeze Frame Data structure table shall be reported as FreezeFrameData in the positive response message. The Freeze Frame Data structure table is primarily intended to be used when the freeze frame data record contains freeze frame data only and no identifier value and no type information is stored in the freeze frame data record. Note: This parameter value is independent of any freeze frame number. The parameter freezeFrameNumber shall be set to allFreezeFrames when recordAccessMethodIdentifer = requestFreezeFrameDataStructure.	U	RFFDS
84 - F9	**vehicleManufacturerSpecific** This range of values is reserved for vehicle manufacturer specific use.	U	VMS
FA - FE	**systemSupplierSpecific** This range of values is reserved for system supplier specific use.	U	SSS
FF	**reservedByDocument** This range of values is reserved by this document for future definition.		RBD

Table 8.4.3.2 - Definition of recordAccessMethodIdentifier values

- The parameter ***recordIdentification (RI_)*** is used by the client to identify a record within the freeze frame referenced by the parameter freezeFrameNumber.

Hex	Description	Cvt	Mnemonic
01 - FE	**recordLocalIdentifier** This range of values identifies a server specific local data record within the freeze frame. Refer to table 7.1.3. “Definition of recordLocalIdentifier values”. **Note:** $00 and $FF are reservedByDocument.	**U**	**LI_...**
01 - FE	**inputOutputLocalIdentifier** This range of values identifies an input/output paramter which is local to a servers. Refer to table 9.1.3.1. “Definition of inputOutputLocalIdentifier values”. **Note:** $00 and $FF are reservedByDocument.	**U**	**IOLI_...**
0000	**DTC** This value is used in the readFreezeFramData positive response message to indicate that no freeze frame data is stored in server’s (ECU’s) memory that matches the parameters in the readFreezeFramData request message.	**U**	**DTC_...**
0001 - FFFE	**DTC** This range of values identifies a record which includes the data within a freeze frame referenced by a DTC (Diagnostic Trouble Code). Refer to table 8.2.3.1.2 “Definition of groupOfDTC values”.	**U**	**DTC_...**
FFFF	**DTC** This value is used in the readFreezeFramData positive response message when recordAccessMethodIdentifer = requestFreezeFrameDataStructure to indicate that the reported Freeze Frame Data structure table is used for all DTCs in this server (ECU).	**U**	**DTC_...**
0001 - FFFE	**recordCommonIdentifier** This range of values identifies a data record which is supported by multiple servers. Refer to table 7.2.3. “Definition of recordCommonIdentifier values”. **Note:** $0000 and $FFFF are reservedByDocument.	**U**	**CI_...**
0001 - FFFE	**inputOutputCommonIdentifier** This range of values identifies an input/output paramter which is supported by multiple servers. Refer to table 9.2.3.1. “Definition of inputOutputCommonIdentifier values”. **Note:** $0000 and $FFFF are reservedByDocument.	**U**	**IOCI_...**

Table 8.4.3.3 - Definition of recordIdentification values

The content of recordIdentification is related to the recordAccessMethodIdentifier according to the table below:

recordAccessMethodIdentifier	recordIdentification content	RI_... size	Cvt	Mnemonic
requestAllData	DTC (response message only)	**2 bytes**	**U**	**DTC_...**
requestByRecordLocalIdentifier	recordLocalIdentifier	**1 byte**	**U**	**RLI_...**
requestByRecordCommonIdentifier	recordCommonIdentifier	**2 bytes**	**U**	**RCI_...**
requestByDTC	DTC	**2 bytes**	**U**	**DTC_...**
DTCThatCausedFreezeFrameStorage	DTC (response message only)	**2 bytes**	**U**	**DTC_...**
requestByInputOutputLocalIdentifier	inputOutputLocalIdentifier	**1 byte**	**U**	**IOLI_...**
requestByInputOutputCommonIdentifier	inputOutputCommonIdentifier	**2 bytes**	**U**	**IOCI_...**
requestFreezeFrameDataStructure	DTC	**2 bytes**	**U**	**DTC_...**
vehicleManufacturerSpecific	vehicleManufacturerSpecific		**U**	**VMS_...**
systemSupplierSpecific	systemSupplierSpecific		**U**	**SSS_...**

Table 8.4.3.4 - Definition of recordIdentification content

- The parameter ***freezeFrameData*** is used by the readFreezeFrameData positive response message and consists of the data identified by the parameters freezeFrameNumber and, if present, recordIdentification.

  When recordAccessMethodIdentifer = requestFreezeFrameDataStructure, the freezeFrameData consists of the freeze frame data structure table for the stored freeze frame data as described in the table below:

Description	Cvt	Hex value
parameterIdentifierType#1 [refer to table 8.4.3.6]	M	xx
parameterIdentifier#1.1	M	xx
parameterIdentifier#1.2	C	xx
:	:	:
parameterIdentifierType#n	M	xx
parameterIdentifier#n.1	M	xx
parameterIdentifier#n.2	C	xx

Table 8.4.3.5 - Freeze frame data structure table

C = conditions: parameter present if parameterIdentifierType = recordCommonIdentifer or parameterIdentifierType = inputOutputCommonIdentifier

- The parameter ***parameterIdentifierType*** is used in the freeze frame data structure table to define what kind of parameter that is included in the freeze frame data.

Hex	Description	Cvt	Mnemonic
00	**reservedByDocument** This range of values is reserved by this document for future definition.		**RBD**
01	**recordLocalIdentifier** Refer to table 7.1.3. “Definition of recordLocalIdentifier values”.	**U**	**RLI**
02	**recordCommonIdentifier** Refer to table 7.2.3. “Definition of recordCommonIdentifier values”.	**U**	**RCI**
03 - 80	**reservedByDocument** This range of values is reserved by this document for future definition.		**RBD**
81	**inputOutputLocalIdentifier** Refer to table 9.1.3.1. “Definition of inputOutputLocalIdentifier values”.	**U**	**IOLI**
82	**inputOutputCommonIdentifier** Refer to table 9.2.3.1. “Definition of inputOutputCommonIdentifier values”.	**U**	**IOCI**
83 - FF	**reservedByDocument** This range of values is reserved by this document for future definition.		**RDB**

Table 8.4.3.6 - Definition of parameterIdentifierType values

- The parameter ***parameterIdentifier*** is used in the freeze frame data structure table to hold the value of the identifier defined by the parameterIdentifierType parameter.

8.4.4 Message flow example

See message flow example of Physical Addressed Service in section 5.3.1.1 and Functional Addressed Service in section 5.3.2.1.

8.4.5 Implementation example of readFreezeFrameData

8.4.5.1 Message flow conditions

Three (3) different implementation examples are specified in the Message Flow section 8.4.5.2. The cause of the freeze frame data storage in the server (ECU) in this example shall be by the occurrence of the DTC "P0130" O2 Sensor Circuit Malfunction (Bank 1, Sensor 1). This DTC causes the following parameters to be stored as freeze frame data:

Note: The parameters listed are specified as freeze frame data in the SAE J1979 E/E Diagnostic Test Modes. The single byte Parameter Identifier is treated as a "localIdentifier" as specified in section 8.4.2.

localIdentifier (Hex)	Parameter Description	Mnemonic
03	Fuel System Status	**FSS**
04	Calculated Load Value	**CLV**
05	Engine Coolant Temperature	**ECT**
06	Short Term Fuel Trim - Bank 1	**STFT**
07	Long Term Fuel Trim - Bank 1	**LTFT**
0A	Fuel Pressure (Gage)	**FP**

localIdentifier (Hex)	Parameter Description	Mnemonic
0B	Intake Manifold Absolute Pressure	**IMAP**
0C	Engine Speed	**ES**
0D	Vehicle Speed	**VS**

Table 8.4.6.1 - FreezeFrameData

8.4.5.2 Message flow

8.4.5.2.1 Message flow with recordAccessMethodIdentifier = DTCThatCausedFreezeFrameStorage

This example shows how the client (tester) requests the DTC which caused the freeze frame data to be stored. The positive response message includes the DTC value in the recordIdentification parameter.

STEP#1 readFreezeFrameData(FFNR_ZERO, RAMI_DTCTCFFS)

time	Client (tester) Request Message	Hex
P3	**readFreezeFrameData.ReqSId**[	**12**
	freezeFrameNumber	00
	recordAccessMethodId=DTCThatCausedFreezeFrameStorage]	80

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**readFreezeFrameData.PosRspSId**[	**52**	**negativeResponse Service Identifier**	**7F**
	freezeFrameNumber	00	**readFreezeFrameData.ReqSId**[	**12**
	RAMI=DTCThatCausedFreezeFrameStorage	80	responseCode { refer to section 4.4 }]	xx
	recordIdentification=P0130 { High Byte }	01		
	recordIdentification=P0130 { Low Byte }]	30		
	return(main)		**return(responseCode)**	

8.4.5.2.2 Message flow with recordAccessMethodIdentifier = RequestByDTC with "P0130"

This example shows how the client (tester) requests the freeze frame data with the recordAccessMethodIdentifier = requestByDTC and the DTC number P0130 included in the recordIdentification parameter. The positive response message includes the freeze frame stored by the server (ECU) at the time the DTC P0130 was stored in long term memory. If more than one freeze frame was stored for the given DTC each freeze frame is sent as a separate positive response message by the server (ECU).

STEP#1 readFreezeFrameData(FFNR_ALL, RAMI_RBDTC, RI_P0130)

time	Client (tester) Request Message	Hex
P3	**readFreezeFrameData.ReqSId**[	**12**
	freezeFrameNumber	FF
	recordAccessMethodId=requestByDTC	04
	recordIdentification=P0130 { High Byte }	01
	recordIdentification=P0130 { Low Byte }]	30

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**readFreezeFrameData.PosRspSId**[	**52**	**negativeResponse Service Identifier**	**7F**
	freezeFrameNumber	00	**readFreezeFrameData.ReqSId**[	**12**
	freezeFrameData#1 {PID#1 High Byte}	:	responseCode { refer to section 4.4 }]	xx
	freezeFrameData#2 {PID#1 Low Byte}	:		
	freezeFrameData#3 {PID#1 data#1}	:		
	:	:		
	freezeFrameData#2+k {PID#1 data#k}	:		
	:	:		
	:	:		
	freezeFrameData#n {PID#m High Byte}	:		
	freezeFrameData#n+1 {PID#m Low Byte}	:		
	freezeFrameData#n+2 {PID#m data#1}	:		
	:	:		
	freezeFrameData#n+k {PID#m data#k}	:		
	RAMI=requestByDTC	04		
	recordIdentification=P0130 { High Byte }	01		
	recordIdentification=P0130 { Low Byte }]	30		

	return(main)		**return(responseCode)**	

8.4.5.2.3 Message flow with recordAccessMethodIdentifier = requestByLocalIdentifier

The example listed below shows how the client (tester) requests the freeze frame parameter "Long Term Fuel Trim - Bank 1". The recordAccessMethodIdentifier is set to "requestByLocalIdentifier". The recordIdentification is set to the localIdentifier '$07' as specified in above table 8.4.6.1. The positive response message includes the content of the parameter requested by the recordIdentification value.

STEP#1 readFreezeFrameData(FFNR_ZERO, RAMI_RBLI, RI_)

time	Client (tester) Request Message	Hex
P3	**readFreezeFrameData.ReqSId**[	**12**
	freezeFrameNumber	00
	recordAccessMethodId=requestByLocalId	01
	recordIdentification=LongTermFuelTrim-Bank1]	07

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**readFreezeFrameData.PosRspSId**[	**52**	**negativeResponse Service Identifier**	**7F**
	freezeFrameNumber	00	**readFreezeFrameData.ReqSId[**	**12**
	FFD#1 {value of LongTermFuelTrim-Bank1}	8F	responseCode { refer to section 4.4 }]	xx
	recordAccessMethodId=requestByLocalId	01		
	recordIdentification=LongTermFuelTrim-Bank1]	07		
	return(main)		**return(responseCode)**	

8.4.5.2.4 Message flow with recordAccessMethodIdentifier = requestAllData

8.4.5.2.4.1 Message flow with recordAccessMethodIdentifier set to requestAllData for one specified freeze frame stored in the server (ECU)

The example listed below shows how the client (tester) requests all data from one specific freeze frame stored in the server's (ECU's) memory. In this example it is assumed that a DTC with number P0002 caused the freeze frame to be stored.

STEP#1 readFreezeFrameData(FFNR_ONE, RAMI_RAD)

time	Client (tester) Request Message	Hex
P3	**readFreezeFrameData.ReqSId**[	**12**
	freezeFrameNumber	01
	recordAccessMethodId=requestAllData]	00

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**readFreezeFrameData.PosRspSId**[	**52**	**negativeResponse Service Identifier**	**7F**
	freezeFrameNumber	01	**readFreezeFrameData.ReqSId[**	**12**
	freezeFrameData#1 {PID#1 High Byte}	:	responseCode { refer to section 4.4 }]	xx
	freezeFrameData#2 {PID#1 Low Byte}	:		
	freezeFrameData#3 {PID#1 data#1}	:		
	:	:		
	freezeFrameData#2+k {PID#1 data#k}	:		
	:	:		
	:	:		
	freezeFrameData#n {PID#m High Byte}	:		
	freezeFrameData#n+1 {PID#m Low Byte}	:		
	freezeFrameData#n+2 {PID#m data#1}	:		
	:	:		
	freezeFrameData#n+k {PID#m data#k}	:		
	RAMI=requestAllData	00		
	recordIdentification=P0002 { High Byte }	00		
	recordIdentification=P0002 { Low Byte }]	02		
	return(main)		**return(responseCode)**	

8.4.5.2.4.2 Message flow with recordAccessMethodIdentifier set to requestAllData for all freeze frames stored in the server (ECU)

The example listed below shows how the client (tester) requests all data from all freeze frames stored in the server's (ECU's) memory. Each freeze frame is sent as a separate positive response message by the server (ECU) based on the client (tester) request message.

STEP#1 readFreezeFrameData(FFNR_ALL, RAMI_RAD)

time	Client (tester) Request Message	Hex
P3	**readFreezeFrameData.ReqSId**[	**12**
	freezeFrameNumber	FF
	recordAccessMethodId=requestAllData]	00

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**readFreezeFrameData.PosRspSId**[	**52**	**negativeResponse Service Identifier**	**7F**
	freezeFrameNumber	00	**readFreezeFrameData.ReqSId[**	**12**
	freezeFrameData#1 {PID#1 High Byte}	:	responseCode { refer to section 4.4 }]	xx
	freezeFrameData#2 {PID#1 Low Byte}	:		
	freezeFrameData#3 {PID#1 data#1}	:		
	:	:		
	freezeFrameData#2+k {PID#1 data#k}	:		
	:	:		
	:	:		
	freezeFrameData#n {PID#m High Byte}	:		
	freezeFrameData#n+1 {PID#m Low Byte}	:		
	freezeFrameData#n+2 {PID#m data#1}	:		
	:	:		
	freezeFrameData#n+k {PID#m data#k}	:		
	RAMI=requestAllData	00		
	recordIdentification=P0130 { High Byte }	01		
	recordIdentification=P0130 { Low Byte }]	30		
	goto STEP#2		**return(responseCode)**	

:

STEP#n readFreezeFrameData positive response#n

time	Server (ECU) Positive Response Message	Hex
P2	**readFreezeFrameData.PosRspSId**[	**52**
	freezeFrameNumber	#n
	freezeFrameData#1 {PID#1 High Byte}	:
	freezeFrameData#2 {PID#1 Low Byte}	:
	freezeFrameData#3 {PID#1 data#1}	:
	:	:
	freezeFrameData#2+k {PID#1 data#k}	:
	:	:
	:	:
	freezeFrameData#n {PID#m High Byte}	:
	freezeFrameData#n+1 {PID#m Low Byte}	:
	freezeFrameData#n+2 {PID#m data#1}	:
	:	:
	freezeFrameData#n+k {PID#m data#k}	:
	RAMI=requestAllData	00
	recordIdentification=P1xxx { High Byte }	1x
	recordIdentification=P1xxx { Low Byte }]	xx
	return(main)	

Negative Response Message Not Possible!

8.4.5.2.4.3 Message flow with recordAccessMethodIdentifier set to requestAllData for all freeze frames with no freeze frames stored in the server (ECU)

The example listed below shows the server’s (ECU’s) positive response message in case the client (tester) requests all data from all freeze frames but none are stored in the server's (ECU's) memory at the time of the request. The recordIdentification parameter shall include ‘$00’ values to indicate that no freeze frames are stored in server’s (ECU’s) memory.

STEP#1 readFreezeFrameData(FFNR_ALL, RAMI_RAD)

time	Client (tester) Request Message	Hex
P3	**readFreezeFrameData.ReqSId**[	**12**
	freezeFrameNumber	FF
	recordAccessMethodId=requestAllData]	00

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**readFreezeFrameData.PosRspSId**[	**52**	**negativeResponse Service Identifier**	**7F**
	freezeFrameNumber	FF	**readFreezeFrameData.ReqSId[**	**12**
	RAMI=requestAllData	00	responseCode { refer to section 4.4 }]	xx
	recordIdentification=no freeze frames stored	00		
	recordIdentification=no freeze frames stored]	00		
	return(main)		**return(responseCode)**	

8.4.5.2.5 Message flow with recordAccessMethodIdentifier set to requestFreezeFrameDataStructure for DTC P0130

The example listed below shows the server's (ECU's) positive response message in case the client (tester) requests the data structure for a freeze frame stored related to DTC P0130.

STEP#1 readFreezeFrameData(FFNR_ALL, RAMI_RFFDS)

time	Client (tester) Request Message	Hex
P3	**readFreezeFrameData.ReqSId**[	**12**
	freezeFrameNumber	FF
	recordAccessMethodId= requestFreezeFrameDataStructure]	83
	recordIdentification=P0130 { High Byte }	01
	recordIdentification=P0130 { Low Byte }]	30

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**readFreezeFrameData.PosRspSId**[	**52**	**negativeResponse Service Identifier**	**7F**
	freezeFrameNumber	FF	**readFreezeFrameData.ReqSId[**	**12**
	parameterIdentifierType#1	01	responseCode { refer to section 4.4 }]	xx
	parameterIdentifier#1.1	03		
	parameterIdentifierType#2	01		
	parameterIdentifier#2.1	04		
	parameterIdentifierType#3	01		
	parameterIdentifier#3.1	05		
	parameterIdentifierType#4	01		
	parameterIdentifier#4.1	06		
	parameterIdentifierType#5	01		
	parameterIdentifier#5.1	07		
	parameterIdentifierType#6	01		
	parameterIdentifier#6.1	0A		
	parameterIdentifierType#7	01		
	parameterIdentifier#7.1	0B		
	parameterIdentifierType#8	01		
	parameterIdentifier#8.1	0C		
	parameterIdentifierType#9	01		
	parameterIdentifier#9.1	0D		
	RAMI= requestFreezeFrameDataStructure	83		
	recordIdentification=P0130 { High Byte }	01		
	recordIdentification=P0130 { Low Byte }]	30		
	return(main)		**return(responseCode)**	

8.5 ClearDiagnosticInformation service

8.5.1 Message description

The clearDiagnosticInformation service is used by the client (tester) to clear diagnostic information in the server's (ECU's) memory. The server (ECU) shall clear diagnostic information based on the content of the groupOfDiagnosticInformation paramter value selected by the client (tester).
If the server(s) (ECU(s)) does not (do not) have any DTC and/or diagnostic information stored the server(s) (ECU(s)) shall send a positive response message upon the reception of a clearDiagnosticInformation request message.
A successful completion of a clearDiagnosticInformation service shall reset all DTC related information in the server's (ECU's) memory. DTC related information is specified as: DTC number, statusOfDTC parameter, systemSupplierData and other DTC related data.

8.5.2 Message data bytes

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**clearDiagnosticInformation Request Service Id**	**M**	**14**	**CDI**

#2
#3
	groupOfDiagnosticInformation = { High Byte }
	groupOfDiagnosticInformation = { Low Byte }
		M
		M
			xx
			xx
				GODI_...

Table 8.5.2.1 - ClearDiagnosticInformation Request Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**clearDiagnosticInformation Positive Response SId**	**M**	**54**	**CDIPR**

#2
#3
	groupOfDiagnosticInformation = { High Byte }
	groupOfDiagnosticInformation = { Low Byte }
		M
		M
			xx
			xx
				GODI_...

Table 8.5.2.2 - ClearDiagnosticInformation Positive Response Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**negativeResponse Service Id**	**M**	**7F**	**NR**
#2	**clearDiagnosticInformation Request Service Id**	**M**	**14**	**CDI**
#3	responseCode=[KWP2000ResponseCode { section 4.4 }]	M	xx=[00-FF]	**RC_...**

Table 8.5.2.3 - ClearDiagnosticInformation Negative Response Message

8.5.3 Parameter definition

- The parameter ***groupOfDiagnosticInformation (GODI_)*** is used by the client (tester) to select a functional group of DTCs or to access a specific DTC. Format and usage of this parameter is the same as defined for the parameter ***groupOfDTC (GODTC_)*** defined in section 8.2.3.

8.5.4 Message flow example

See message flow example of Physically Addressed Service in section 5.3.1.1 and Functionally Addressed Service in section 5.3.2.1.

8.5.5 Implementation example of clearDiagnosticInformation

8.5.5.1 Message flow conditions for clearDiagnosticInformation

Two (2) different implementation examples are specified in this section. The groupOfDiagnosticInformation parameter in example #1 is set to "$0000" representing the functional group of DTCs “Powertrain DTCs”.
In example #2 this parameter is set to "$0120" representing the specific DTC P0120 "Throttle Position Circuit Malfunction".

The server (ECU) has stored two (2) DTCs:	"P0130" O2 Sensor Circuit Malfunction (Bank 1, Sensor 1);
	"P0120" Throttle Position Circuit Malfunction

8.5.5.2 Message flow

8.5.5.2.1 Message flow of clearDiagnosticInformation by function group

STEP#1 clearDiagnosticInformation(GODI_PG)

time	Client (tester) Request Message	Hex
P3	**clearDiagnosticInformation.ReqSId**[	**14**
	groupOfDiagnosticInformation = Powertrain { High Byte }	00
	groupOfDiagnosticInformation = Powertrain { Low Byte }]	00

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**clearDiagnosticInformation.PosRspSId**[	**54**	**negativeResponse Service Identifier**	**7F**
	groupOfDiagnosticInfo = Powertrain { High Byte }	00	**clearDiagnosticInformation.ReqSId[**	**14**
	groupOfDiagnosticInfo = Powertrain { Low Byte }]	00	responseCode { refer to section 4.4 }]	xx
	return(main)		**return(responseCode)**	

8.5.5.2.2 Message flow of clearDiagnosticInformation by DTC

STEP#1 clearDiagnosticInformation(GODI_DTC_P0120)

time	Client (tester) Request Message	Hex
P3	**clearDiagnosticInformation.ReqSId**[	**14**
	groupOfDiagnosticInformation = DTC_P0120 { High Byte }	01
	groupOfDiagnosticInformation = DTC_P0120 { Low Byte }]	20

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**clearDiagnosticInformation.PosRspSId**[	**54**	**negativeResponse Service Identifier**	**7F**
	groupOfDiagnosticInfo=DTC_P0120 { High Byte }	01	**clearDiagnosticInformation.ReqSId[**	**14**
	groupOfDiagnosticInfo=DTC_P0120 { Low Byte }]	20	responseCode { refer to section 4.4 }]	xx
	return(main)		**return(responseCode)**	

9 InputOutput Control functional unit

The services provided by this functional unit are described in table 9:

Service name	Description
InputOutputControlByLocalIdentifier	The client requests the control of an input/output specific to the server.
InputOutputControlByCommonIdentifier	The client requests the control of an input/output common to one or multiple server(s).

Table 9 - InputOutput Control functional unit

9.1 InputOutputControlByLocalIdentifier service

9.1.1 Message description

The inputOutputControlByLocalIdentifier 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 an inputOutputLocalIdentifier 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. It is user optional if the positive response message shall include controlStatus information which possibly is available during or after the control execution. The controlOption parameter can be implemented as a single ON/OFF parameter or as a more complex sequence of control parameters including a number of cycles, a duration, etc. if required.
The actions executed by this service may result in permanent changes, or they may be automatically reset by the ECU at certain conditions (e.g. when leaving diagnostic mode or when leaving the adjustmentSession).

9.1.2 Message Data Bytes

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**inputOutputControlByLocalIdentifier Request SId**	**M**	**30**	**IOCBLI**
#2	inputOutputLocalIdentifier	M	xx	**IOLI_...**
	controlOption=[			**CO_...**
#3	inputOutputControlParameter	M	xx	**IOCP_...**
#4	controlState#1	C	xx	**CS_...**
:	:	:	:	:
#n	controlState#m]	C	xx	**CS_...**

Table 9.1.2.1 - InputOutputControlByLocalIdentifier request message

C = condition: parameter depends on content of inputOutputControlParameter!

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**inputOutputControlByLocalIdentifier Positive Response SId**	**M**	**70**	**IOCBLIPR**
#2	inputOutputLocalIdentifier	M	xx	**IOLI_...**
	controlStatus=[			**CSS_...**
#3	inputOutputControlParameter	M	xx	**IOCP_...**
#4	controlState#1	C	xx	**CS_...**
:	:	:	:	:
#n	controlState#m]	C	xx	**CS_...**

Table 9.1.2.2 - InputOutputControlByLocalIdentifier positive response message

C = condition: parameter depends on content of inputOutputControlParameter!

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**negativeResponse Service Id**	**M**	**7F**	**NR**
#2	**inputOutputControlByLocalIdentifier Request SId**	**M**	**30**	**IOCBLI**
#3	responseCode=[KWP2000ResponseCode { section 4.4 }]	M	xx=[00-FF]	**RC_...**

Table 9.1.2.3 - InputOutputControlByLocalIdentifier negative response message

9.1.3 Parameter definition

- The parameter ***inputOutputLocalIdentifier (IOLI_)*** in the inputOutputControlByLocalIdentifier request service identifies an ECU local input signal, internal parameter or output signal.

Hex	Description	Cvt	Mnemonic
00	**reservedByDocument** This value shall be reserved by this document for future definition.	**M**	**RBD**
01 - F9	**inputOutputLocalIdentifier** This range of values shall be used by the vehicle manufacturer to reference input/output local identifiers in the server (ECU).	**U**	**IOLI_...**
FA - FE	**systemSupplierSpecific** This range of values is reserved for system supplier specific use.	**U**	**SSS**

FF	**reservedByDocument** This value shall be reserved by this document for future definition.	**M**	**RBD**

Table 9.1.3.1 - Definition of inputOutputLocalIdentifier values

- The ***controlOption*** parameters consist of an ***inputOutputControlParameter (IOCP_)*** and ***controlState (CS_)*** parameters in the inputOutputControlByLocalIdentifier service. The service shall be used to control different states of the server's (ECU's) local input signal(s), internal parameter(s) and output signal(s) as defined in the table below.
- The ***inputOutputControlParameter (IOCP_)*** parameter is used by the inputOutputControlByLocalIdentifier request message to describe how the server (ECU) has to control its inputs or outputs as specified in the table below.

Note: The usage of the inputOutputControlParameters specified in the following table depends on the inputOutputLocalIdentifier which shall be manipulated.

Hex	Description	Cvt	Mnemonic
00	**returnControlToECU** This value shall indicate to the server (ECU) that the client (tester) does no longer have control about the input signal, internal parameter or output signal referenced by the inputOutputLocalIdentifier.	**U**	**RCTECU**
01	**reportCurrentState** This value shall indicate to the server (ECU) that it is requested to report the current state of the input signal, internal parameter or output signal referenced by the inputOutputLocalIdentifier.	**U**	**RCS**
02	**reportIOConditions** This value shall indicate to the server (ECU) that it is requested to report the conditions of the input signal, internal parameter or output signal referenced by the inputOutputLocalIdentifier. The server (ECU) specific condition(s) are the responsibility of the vehicle manufacturer/system supplier.	**U**	**RIOC**
03	**reportIOScaling** This value shall indicate to the server (ECU) that it is requested to report the scaling of the input signal, internal parameter or output signal referenced by the inputOutputLocalIdentifier. The server (ECU) specific scaling is the responsibility of the vehicle manufacturer/system supplier. Scaling shall be reported according to the format defined in section 5.4, but only the scalingByte and ScalingByteExtention data for the requested inputOutputLocalIdentifer shall be included in the positive response message.	**U**	**RIOS**
04	**resetToDefault** This value shall indicate to the server (ECU) that it is requested to reset the input signal, internal parameter or output signal referenced by the inputOutputLocalIdentifier to its default state.	**U**	**RTD**
05	**freezeCurrentState** This value shall indicate to the server (ECU) that it is requested to freeze the current state of the input signal, internal parameter or output signal referenced by the inputOutputLocalIdentifier.	**U**	**FCS**
06	**executeControlState** This value shall indicate to the server (ECU) that it is requested to execute the controlState parameter(s). The controlState shall be of single or multiple parameters (bytes) if requested.	**U**	**ECO**
07	**shortTermAdjustment** This value shall indicate to the server (ECU) 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 controlState 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).	**U**	**STA**
08	**longTermAdjustment** This value shall indicate to the server (ECU) that it is requested to adjust the input signal, internal parameter or output signal referenced by the inputOutputLocalIdentifier in EEPROM/FLASH EEPROM to the value(s) included in the controlState parameter(s). (e.g. set Engine Idle Speed, set CO Adjustment parameter to a specific value).	**U**	**LTA**

09	**reportIOCalibrationParameters** This value shall indicate to the server (ECU) that it is requested to report the inputOutputCalibrationParameters of the input signal, internal parameter or output signal referenced by the inputOutputLocalIdentifier.	**U**	**RIOCP**
0A - F9	**vehicleManufacturerSpecific** This range of values is reserved for vehicle manufacturer specific use.	**U**	**VMS**
FA - FE	**systemSupplierSpecific** This range of values is reserved for system supplier specific use.	**U**	**SSS**
FF	**reservedByDocument** This value is reserved by this document for future definition.	**M**	**RBD**

Table 9.1.3.2 - Definition of inputOutputControlParameter values

- The ***controlState (CS_)*** parameter may consist of multiple bytes which include data (e.g. filter mask) depending on the inputOutputControl parameter used in the request message.
  This paramter shall be used to report scaling data (scalingByte and ScalingByteExtention data) in the positive response message if the paramter inputOutputControlPrameter = "$03" (reportIOScaling).
- The ***controlStatus (CSS_)*** parameters consist of an ***inputOutputControlParameter (IOCP_)*** and a ***controlState (CS_)*** parameters in the inputOutputControlByLocalIdentifier positive response message.

9.1.4 Message flow example

See message flow example of Physical Addressed Service in section 5.3.1.1.

9.1.5 Implementation example of "Desired Idle Adjustment"

9.1.5.1 "Desired Idle Adjustment" resetToDefault

9.1.5.1.1 Message flow conditions for resetToDefault

This section specifies the conditions of the resetToDefault function and the associated message flow of the "Desired Idle Adjustment" inputOutputLocalIdentifier ($32).
Test conditions: ignition=ON, engine at idle speed, engine at operating temperature, vehicle speed=0 [kph]
Conversion: Desired Idle Adjustment [R/MIN] = decimal(Hex) * 10 [r/min]

9.1.5.1.2 Message flow

STEP#1 inputOutputControlByLocalIdentifier("Desired Idle Adjustment, "IOCP_RTD)

time	Client (tester) Request Message	Hex

P3	**inputOutputControlByLocalId.Request**[
	inputOutputLocalId = "Desired Idle Adjustment"
	inputOutputControlParameter = resetToDefault]
		30
		32
		04

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex

P2	**inputOutputControlByLocalId.PosRsp**[
	inputOutputLocalId = "Desired Idle Adjustment"
	inputOutputControlParameter = resetToDefault
	750 r/min { default idle adjustment }]
		70
		32
		04
		4B
			negativeResponse Service Identifier
			inputOutputControlByLocalId.ReqSId[
			responseCode { refer to section 4.4 }]
				7F
				30
				xx

	return(main)		**return(responseCode)**	

9.1.5.2 "Desired Idle Adjustment" shortTermAdjustment

9.1.5.2.1 Message flow conditions for shortTermAdjustment

This section specifies the conditions of a shortTermAdjustment function and the associated message flow of the "Desired Idle Adjustment" inputOutputLocalIdentifier.
Test conditions: ignition=ON, engine at idle speed, engine at operating temperature, vehicle speed=0 [kph]
Conversion: Desired Idle Adjustment [r/min] = decimal(Hex) * 10 [r/min]

9.1.5.2.2 Message flow

STEP#1 inputOutputControlByLocalIdentifier("Desired Idle Adjustment", IOCP_RCS)

time	Client (tester) Request Message	Hex
P3	**inputOutputControlByLocalId.Request**[	**30**
	inputOutputLocalId = "Desired Idle Adjustment"	32
	inputOutputControlParameter = reportCurrentState]	01

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**inputOutputControlByLocalId.PosRsp**[	**70**	**negativeResponse Service Identifier**	**7F**
	inputOutputLocalId = "Desired Idle Adjustment"	32	**inputOutputControlByLocalId.ReqSId[**	**30**
	inputOutputControlParameter = reportCurrentState	01	responseCode { refer to section 4.4 }]	xx
	800 r/min]	50		
	goto STEP#2		**return(responseCode)**	

STEP#2 inputOutputControlByLocalIdentifier("Desired Idle Adjustment", IOCP_FCS)

time	Client (tester) Request Message	Hex
P3	**inputOutputControlByLocalId.Request**	**30**
	inputOutputLocalId = "Desired Idle Adjustment"	32
	inputOutputControlParameter = freezeCurrentState	05

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**inputOutputControlByLocalId.PosRsp**[	**70**	**negativeResponse Service Identifier**	**7F**
	inputOutputLocalId = "Desired Idle Adjustment"	32	**inputOutputControlByLocalId.ReqSId[**	**30**
	inputOutputControlParameter = freezeCurrentState	05	responseCode { refer to section 4.4 }]	xx
	800 r/min]	50		
	goto STEP#3		**return(responseCode)**	

STEP#3 inputOutputControlByLocalIdentifier("Desired Idle Adjustment", IOCP_STA, 1000 r/min)

time	Client (tester) Request Message	Hex
P3	**inputOutputControlByLocalId.Request**[	**30**
	inputOutputLocalId = "Desired Idle Adjustment"	32
	inputOutputControlParameter = shortTermAdjustment	07
	1000 r/min]	64

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**inputOutputControlByLocalId.PosRsp**[	**70**	**negativeResponse Service Identifier**	**7F**
	inputOutputLocalId = "Desired Idle Adjustment"	32	**inputOutputControlByLocalId.ReqSId[**	**30**
	inputOutputControlParameter = shortTermAdjustment	07	responseCode { refer to section 4.4 }]	xx
	820 r/min { current Engine Speed }]	52		
	goto STEP#4		**return(responseCode)**	

Note: The client (tester) has sent an inputOutputControlByLocalIdentifier request message as specified in STEP#3. The server (ECU) has sent an immediate positive response message which includes the controlState parameter "Engine Speed" with the value of "820 r/min". The engine requires a certain amount of time to adjust the idle speed to the requested value of "1000 r/min".

STEP#4 Repeat readDataByLocalIdentifier(RLI_01) Until ("Desired Idle Adjustment"==1000 r/min)

time	Client (tester) Request Message	Hex
P3	**readDataByLocalId.Request**[	**21**
	recordLocalIdentifier]	02

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**readDataByLocalId.PosRsp**[	**61**	**negativeResponse Service Identifier**	**7F**
	recordLocalIdentifier	02	**readDataByLocalId.ReqSId[**	**21**
	recordValue#1	xx	responseCode { refer to section 4.4 }]	xx
	:	:		
	recordValue#n]	xx		
	goto STEP#5		**return(responseCode)**	

STEP#5 inputOutputControlByLocalIdentifier("Desired Idle Adjustment", IOCP_RCTECU)

time	Client (tester) Request Message	Hex
P3	**inputOutputControlByLocalId.Request**[	**30**
	inputOutputLocalId = "Desired Idle Adjustment"	32
	inputOutputControlParameter = returnControlToECU]	00

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**inputOutputControlByLocalId.PosRsp**[	**70**	**negativeResponse Service Identifier**	**7F**
	inputOutputLocalId = "Desired Idle Adjustment"	32	**inputOutputControlByLocalId.ReqSId[**	**30**
	inputOutputControlParameter = returnControlToECU]	00	responseCode { refer to section 4.4 }]	xx
	return(main)		**return(responseCode)**	

9.1.5.3 "Desired Idle Speed Offset" longTermAdjustment

9.1.5.3.1 Message flow conditions for longTermAdjustment

This section specifies the conditions of a longTermAdjustment function and the associated message flow of the "Desired Idle Speed Offset" inputOutputLocalIdentifier ($33).
Test conditions: ignition=ON, engine at idle speed, engine at operating temperature, vehicle speed=0 [kph]
Conversion: Desired Idle Speed Offset [r/min] = decimal(Hex) * 1 [r/min]
Assumptions for this example: Current "Desired Idle Speed Offset" is 50 [r/min]. Current Idle Speed = 750 [r/min]. Basic Idle Speed = 700 [r/min].
After the server (ECU) has sent an InputOutputControlByLocalIdentifier positive response message to the client (tester), ignition shall be turned off to enable the ECU to write the new Desired Idle Speed Offset value into the Non Volatile Memory.

9.1.5.3.2 Message flow

STEP#1 inputOutputControlByLocalIdentifier("Desired Idle Speed Offset", IOCP_RIOCP)

time	Client (tester) Request Message	Hex
P3	**inputOutputControlByLocalId.Request**[	**30**
	inputOutputLocalId = "Desired Idle Speed Offset"	33
	inputOutputControlParameter = reportIOCalibrationParameters	09

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**inputOutputControlByLocalId.PosRsp**[	**70**	**negativeResponse Service Identifier**	**7F**
	inputOutputLocalId = "Desired Idle Speed Offset"	33	**inputOutputControlByLocalId.ReqSId[**	**30**
	inputOutputControlParameter = reportIOCalibrationPara	09	responseCode { refer to section 4.4 }]	xx
	minimumOffsetValue { 0 [r/min] }	00		
	maximumOffsetValue { 150 [r/min] }	96		
	offsetParameterResolution { 50 [r/min] }	32		
	defaultOffsetValue { 0 [r/min] }	00		
	actualOffsetValue] { 50 [r/min] }	32		
	goto STEP#2		**return(responseCode)**	

STEP#2 inputOutputControlByLocalIdentifier("Desired Idle Speed Offset", IOCP_LTA, 150 r/min)

time	Client (tester) Request Message	Hex
P3	**inputOutputControlByLocalId.Request**[	**30**
	inputOutputLocalId = "Desired Idle Speed Offset"	33
	inputOutputControlParameter = longTermAdjustment	08
	150 r/min]	96

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**inputOutputControlByLocalId.PosRsp**[	**70**	**negativeResponse Service Identifier**	**7F**
	inputOutputLocalId = "Desired Idle Speed Offset"	33	**inputOutputControlByLocalId.ReqSId[**	**30**
	inputOutputControlParameter = longTermAdjustment	08	responseCode { refer to section 4.4 }]	xx
	150 r/min]	96		
	goto STEP#3		**return(responseCode)**	

InputOutputLocalIdentifier: Desired Idle Adjustment ($32)
Basic Idle Speed now equals 850 [r/min].
Conversion: Desired Idle Adjustment [r/min] = decimal(Hex) * 10 [r/min]

STEP#3 inputOutputControlByLocalIdentifier("Desired Idle Adjustment", IOCP_RCS)

time	Client (tester) Request Message	Hex
P3	**inputOutputControlByLocalId.Request**[	**30**
	inputOutputLocalId = "Desired Idle Adjustment"	32
	inputOutputControlParameter = reportCurrentState]	01

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**inputOutputControlByLocalId.PosRsp**[	**70**	**negativeResponse Service Identifier**	**7F**
	inputOutputLocalId = "Desired Idle Adjustment"	32	**inputOutputControlByLocalId.ReqSId[**	**30**
	inputOutputControlParameter = reportCurrentState	01	responseCode { refer to section 4.4 }]	xx
	850 r/min]	55		
	return(main)		**return(responseCode)**	

9.2 InputOutputControlByCommonIdentifier service

9.2.1 Message description

The InputOutputControlByCommonIdentifier service is used by the client to substitute a value for an input signal, internal ECU function and/or an control output (actuator) of electronic systems referenced by an inputOutputCommonIdentifier 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(s) shall send a positive response message if the request message was successfully executed. It is user optional if the positive response message shall include controlStatus information which possibly is available during or after the control execution. The controlOption parameter can be implemented as a single ON/OFF parameter or as a more complex sequence of control parameters including a number of cycles, a duration, etc.
The actions executed by this service may result in permanent changes, or it may be automatically reset by the ECU at certain conditions (e.g. when leaving diagnostic mode).

9.2.2 Message data bytes

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**inputOutputControlByCommonIdentifier Request SId**	**M**	**2F**	**IOCBCID**
#2	inputOutputCommonIdentifier (High Byte)	M	xx	**IOCI_...**
#3	inputOutputCommonIdentifier (Low Byte)	M	xx	
	controlOption=[			**CO_...**
#4	inputOutputControlParameter	M	xx	**IOCP_...**
#5	controlState#1	C	xx	**CS_...**
:	**:**	**:**	**:**	**:**
#n	controlState#m]	C	xx	**CS_...**

Table 9.2.2.1 - InputOutputControlByCommonIdentifier Request Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**inputOutputControlByCommonIdentifier Positive Resp. SId**	**M**	**6F**	**IOCBCIDPR**
#2	inputOutputCommonIdentifier (High Byte)	M	xx	**IOCI_...**
#3	inputOutputCommonIdentifier (Low Byte)	M	xx	
	controlStatus=[			**CSS_...**
#4	inputOutputControlParameter	M	xx	**IOCP_...**
#5	controlState#1	C	xx	**CS_...**
:	**:**	**:**	**:**	**:**
#n	controlState#m]	C	xx	**CS_...**

Table 9.2.2.2 - InputOutputControlByCommonIdentifier Positive Response Message

C = condition: parameter depends on content of inputOutputControlParameter!

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**negativeResponse Service Id**	**M**	**7F**	**NACK**
#2	**inputOutputControlByCommonIdentifier Request SId**	**M**	**2F**	**IOCBCID**
#3	responseCode=[KWP2000ResponseCode { section 4.4 }]	M	xx=[00-FF]	**RC_...**

Table 9.2.2.3 - InputOutputControlByCommonIdentifier Negative Response Message

9.2.3 Parameter definition

- The parameter ***inputOutputCommonIdentifier*** in the inputOutputControlByCommonIdentifier request service identifies an input or output common to the ECU(s).

  The inputOutputCommonIdentifier table consists of four (4) columns and multiple lines.
  - The **1st column (Hex)** includes the "Hex Value" assigned to the inputOutputCommonIdentifier specified in the 2nd column.
  - The **2nd column (Description)** specifies the inputOutputCommonIdentifier.
  - The **3rd column (Cvt)** is the convention column which specifies whether the inputOutputCommonIdentifier is "M" (mandatory) or "U" (User Optional).
  - The **4th column (Mnemonic)** specifies the mnemonic of this inputOutputCommonIdentifier.

M = mandatory, **U** = user optional

Hex	Description	Cvt	Mnemonic
0000	**reservedByDocument** This value shall be reserved by this document for future definition.	**M**	**RBD**
0001 - EFFF	**inputOutputCommonIdentifier** This range of values shall be reserved to reference parameters implemented in the server (ECU) upon the vehicle manufacturer's request.	**U**	**IOCI_**
F000 - FFFE	**systemSupplierSpecific** This range of values is reserved for system supplier specific use.	**U**	**SSS**
FFFF	**reservedByDocument** This value shall be reserved by this document for future definition.	**M**	**RBD**

Table 9.2.3.1 - Definition of inputOutputCommonIdentifier values

- The ***controlOption*** parameters consist of an ***inputOutputControlParameter (IOCP_)*** and ***controlState (CS_)*** parameters in the inputOutputControlByCommonIdentifier service. The service shall be used to control different states of the server's (ECU's) local input signal(s), internal parameter(s) and output signal(s) as defined in the table below.
- The ***inputOutputControlParameter (IOCP_)*** parameter is used by the inputOutputControlByCommonIdentifier request message to describe how the server (ECU) has to control its inputs or outputs as specified in the table below.

Note: The usage of the inputOutputControlParameters specified in the following table depends on the inputOutputCommonIdentifier which shall be manipulated.

Hex	Description	Cvt	Mnemonic
00	**returnControlToECU** This value shall indicate to the server (ECU) that the client (tester) does no longer have control about the input signal, internal parameter or output signal referenced by the inputOutputCommonIdentifier.	**U**	**RCTECU**
01	**reportCurrentState** This value shall indicate to the server (ECU) that it is requested to report the current state of the input signal, internal parameter or output signal referenced by the inputOutputCommonIdentifier.	**U**	**RCS**
02	**reportIOConditions** This value shall indicate to the server (ECU) that it is requested to report the conditions of the input signal, internal parameter or output signal referenced by the inputOutputCommonIdentifier. The server (ECU) specific condition(s) are the responsibility of the vehicle manufacturer/system supplier.	**U**	**RIOC**
03	**reportIOScaling** This value shall indicate to the server (ECU) that it is requested to report the scaling of the input signal, internal parameter or output signal referenced by the inputOutputCommonIdentifier. The server (ECU) specific scaling is the responsibility of the vehicle manufacturer/system supplier. Scaling shall be reported according to the format defined in section 5.4, but only the scalingByte and ScalingByteExtention data for the requested inputOutputCommonIdentifer shall be included in the positive response message.	**U**	**RIOS**
04	**resetToDefault** This value shall indicate to the server (ECU) that it is requested to reset the input signal, internal parameter or output signal referenced by the inputOutputCommonIdentifier to its default state.	**U**	**RTD**
05	**freezeCurrentState** This value shall indicate to the server (ECU) that it is requested to freeze the current state of the input signal, internal parameter or output signal referenced by the inputOutputCommonIdentifier.	**U**	**FCS**
06	**executeControlState** This value shall indicate to the server (ECU) that it is requested to execute the controlState parameter(s). The controlState shall be of single or multiple parameters (bytes) if requested.	**U**	**ECO**
07	**shortTermAdjustment** This value shall indicate to the server (ECU) that it is requested to adjust the input signal, internal parameter or output signal referenced by the inputOutputCommonIdentifier in RAM to the value(s) included in the controlState 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).	**U**	**STA**

Hex	Description	Cvt	Mnemonic
08	**longTermAdjustment** This value shall indicate to the server (ECU) that it is requested to adjust the input signal, internal parameter or output signal referenced by the inputOutputCommonIdentifier in EEPROM/FLASH EEPROM to the value(s) included in the controlState parameter(s). (e.g. set Engine Idle Speed, set CO Adjustment parameter to a specific value).	**U**	**LTA**
09	**reportIOCalibrationParameters** This value shall indicate to the server (ECU) that it is requested to report the inputOutputCalibrationParameters of the input signal, internal parameter or output signal referenced by the inputOutputCommonIdentifier.	**U**	**RIOCP**
0A - F9	**vehicleManufacturerSpecific** This range of values is reserved for vehicle manufacturer specific use.	**U**	**VMS**
FA - FE	**systemSupplierSpecific** This range of values is reserved for system supplier specific use.	**U**	**SSS**
FF	**reservedByDocument** This value is reserved by this document for future definition.	**M**	**RBD**

Table 9.2.1.2 - Definition of inputOutputControlParameter values

- The ***controlState (CS_)*** parameter may consist of multiple bytes which include data (e.g. filter mask) depending on the inputOutputControl parameter used in the request message.
  This paramter shall be used to report scaling data (scalingByte and ScalingByteExtention data) in the positive response message if the paramter inputOutputControlPrameter = "$03" (reportIOScaling).
- The ***controlStatus (CSS_)*** parameters consist of an ***inputOutputControlParameter (IOCP_)*** and a ***controlState (CS_)*** parameter.

9.2.4 Message flow example

See message flow example of Physical Addressed Service in section 5.3.1.1 and Functional Addressed Service in section 5.3.2.1.

9.2.5 Implementation examples

Refer to the implementation examples of section 9.1.5.

10 Remote Activation Of Routine functional unit

The services provided by this functional unit are described in table 10:

Service name	Description
StartRoutineByLocalIdentifier	The client requests to start a routine in the ECU of the server.
StartRoutineByAddress	The client requests to start a routine in the ECU of the server.
StopRoutineByLocalIdentifier	The client requests to stop a routine in the ECU of the server.
StopRoutineByAddress	The client requests to stop a routine in the ECU of the server.
RequestRoutineResultsByLocalIdentifier	The client requests the results of a routine by the Routine Local Identifier.
RequestRoutineResultsByAddress	The client requests the results of a routine by the Routine Address.

Table 10 - Remote Activation Of Routine functional unit

This functional unit specifies the services of remote activation of routines as they shall be implemented in servers (ECUs) and client (tester). Figure 10.1 and 10.2 shows two (2) different methods of implementation. These methods shall be used as a guideline for implementation of routine services. There may be other methods of implementation possible.

Note: Each method may feature the service "requestRoutineResultsByLocalIdentifier/Address" after the routine has been stopped. The selection of method and the implementation is the responsibility of the vehicle manufacturer and system supplier.

The following is a brief description of the methods shown in figure 10.1 and 10.2:

- “Routine stopped by service”: This method is based on the assumption that after a routine has been started by the client (tester) in the server's (ECU's) memory the client (tester) shall be responsible to stop the routine.
  The ECU routine shall be started in the server's (ECU's) memory some time between the completion of the startRoutineByLocalIdentifier/Address request message and the completion of the first response message (if "positive" based on the server's conditions).
  The ECU routine shall be stopped in the server's (ECU's) memory some time after the completion of the stopRoutineByLocalIdentifier/Address request message and the completion of the first response message (if "positive" based on the server's conditions).
  The client (tester) may request routine results after the routine has been stopped.

- “Routine stopped by time”: This method is based on the assumption that after a routine has been started by the client (tester) in the server's (ECU's) memory that the server (ECU) shall be responsible to stop the routine.
  The ECU routine shall be started in the server's (ECU's) memory some time between the completion of the startRoutineByLocalIdentifier/Address request message and the completion of the first response message (if "positive" based on the server's conditions).
  The ECU routine shall be stopped any time as programmed or previously initialised in the server's (ECU's) memory.
  The client (tester) may request routine results after the routine has been stopped.

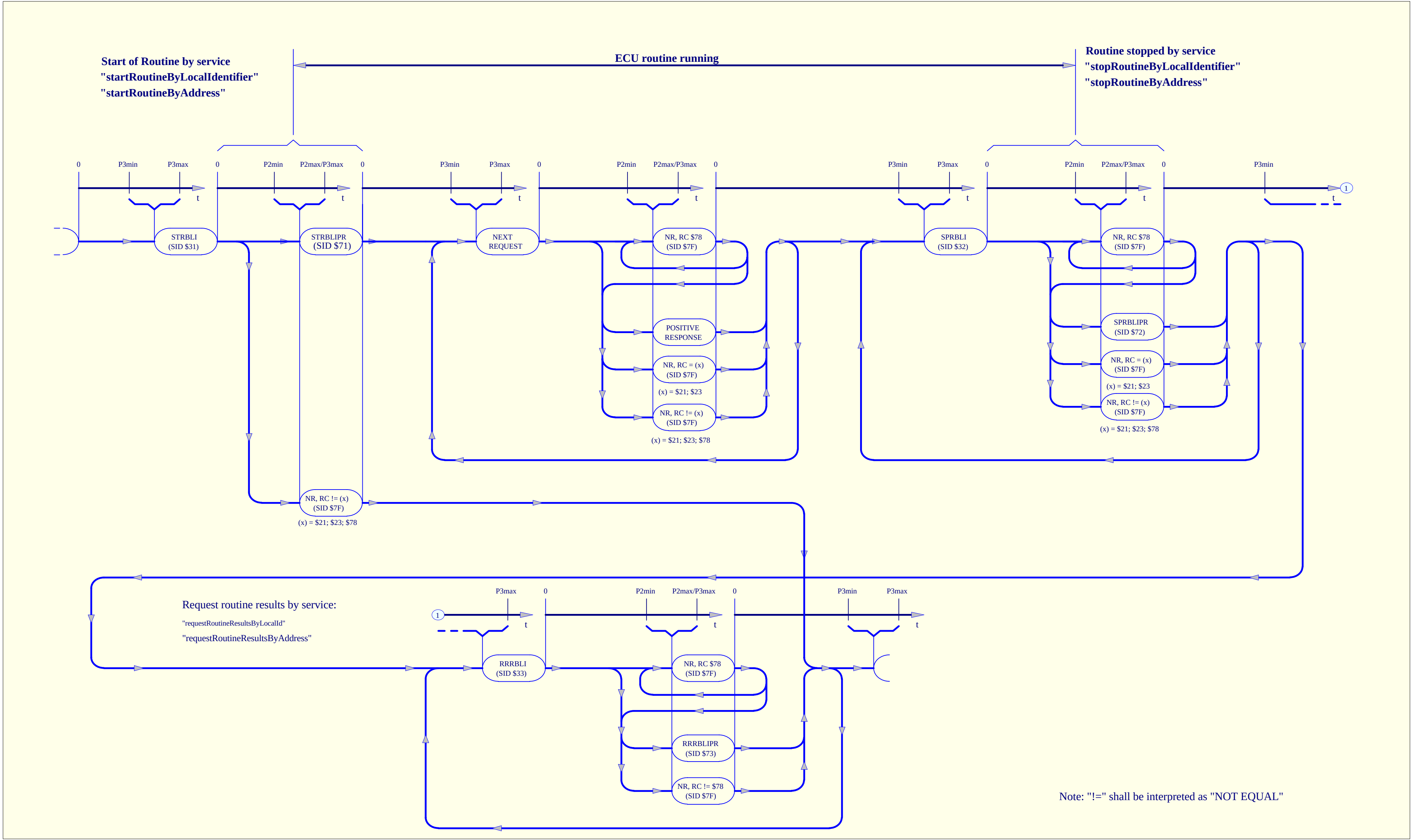


Figure 10.1 - Example of “Routine stopped by service”

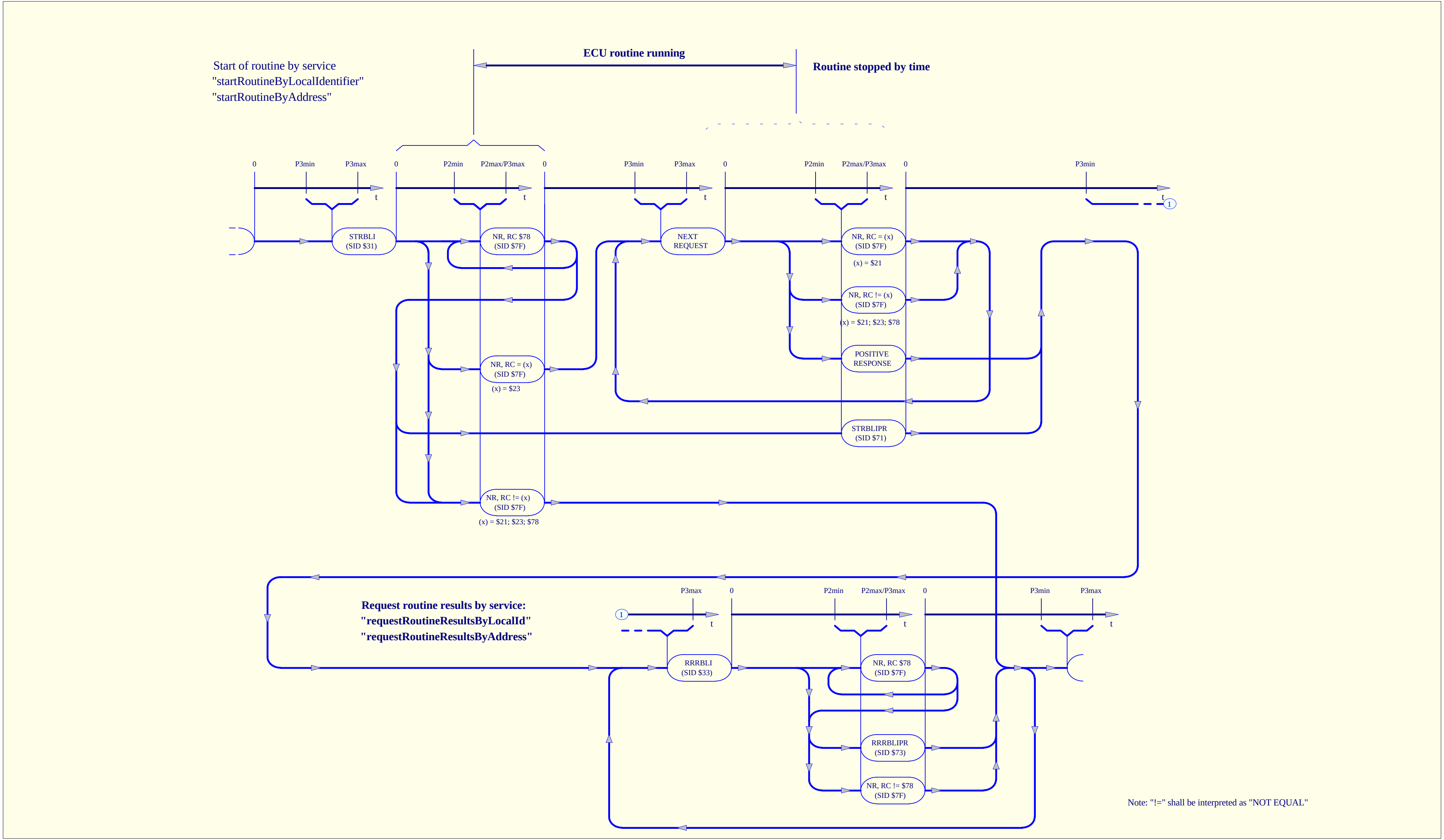


Figure 10.2 - Example of “Routine stopped by time”

10.1 StartRoutineByLocalIdentifier service

10.1.1 Message description

The startRoutineByLocalIdentifier service is used by the client to start a routine in the server's memory.
The ECU routine shall be started in the server's (ECU's) memory some time between the completion of the startRoutineByLocalIdentifier request message and the completion of the first response message if the response message is positive.
The routines could either be tests that run instead of normal operating code or could be routines that are enabled and executed with the normal operating code running. Particularly in the first case, it might be necessary to switch the server to a specific diagnostic session using the startDiagnosticSession service or to unlock the server using the securityAccess service prior to using the startRoutineByLocalIdentifier service.
The parameter routineEntryOption is user optional and. Examples of routineEntryOptions are: timeToRun, startUpVariables, etc.

10.1.2 Message data bytes

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**startRoutineByLocalIdentifier Request Service Id**	**M**	**31**	**STRBLI**
#2	routineLocalIdentifier	M	xx	**RELI_...**
#3 : #n	routineEntryOption#1 : routineEntryOption#m	U : U	xx : xx	**REYO_...** : **REYO_...**

Table 10.1.2.1 - StartRoutineByLocalIdentifier Request Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**startRoutineByLocalIdentifier Positive Response SId**	**M**	**71**	**STRBLIPR**
#2	routineLocalIdentifier	M	xx	**RELI_...**
#3 : #n	routineEntryStatus#1 : routineEntryStatus#m	U : U	xx : xx	**REYS_...** : **REYS_...**

Table 10.1.2.2 - StartRoutineByLocalIdentifier Positive Response Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**negativeResponse Service Id**	**M**	**7F**	**NR**
#2	**startRoutineByLocalIdentifier Request Service Id**	**M**	**31**	**STRBLI**
#3	responseCode=[KWP2000ResponseCode { section 4.4 }]	M	xx=[00-FF]	**RC_...**

Table 10.1.2.3 - StartRoutineByLocalIdentifier Negative Response Message

10.1.3 Parameter definition

The parameter ***routineLocalIdentifier (RELI_)*** in the start/stopRoutineByLocalIdentifier and requestRoutineResultsByLocalIdentifier request message identifies a server local routine.

Hex	Description	Cvt	Mnemonic
00	**reservedByDocument** This value shall be reserved by this document for future definition.	**M**	**RBD**
01 - F9	**routineLocalIdentifier** This range of values shall be used by the vehicle manufacturer to reference routines in the server (ECU).	**U**	**RELI_...**
FA - FE	**systemSupplierSpecific** This range of values is reserved for system supplier specific use.	**U**	**SSS**
FF	**reservedByDocument** This value shall be reserved by this document for future definition.	**M**	**RBD**

Table 10.1.1 - Definition of routineLocalIdentifier values

- The parameter ***routineEntryOption (REYO_)*** is used by the startRoutineByLocalIdentifier and startRoutineByAddress request message to specify the start conditions of the routine.

- The parameter ***routineEntryStatus (REYS_)*** is used by the startRoutineByLocalIdentifier and startRoutineByAddress positive response message to give to the client additional information about the status of the server following the start of the routine.

10.1.4 Message flow example

See message flow examples of a Physically Addressed Service in section 5.3.1.1 and a Functionally Addressed Service in section 5.3.2.1. A complete message flow example is shown in section 10.5.4.

10.1.5 Implementation example of "Input/output signal intermittent test routine"

10.1.5.1 Message flow conditions for "Input/output signal intermittent test routine" (IOSITR)

This section specifies the message flow conditions to start a routine in the server (ECU) to continuously test (as fast as possible) all input and output signals on intermittence while a technician would "wiggle" all wiring harness connectors of the system under test. The routineLocalIdentifier references this routine by the routineLocalIdentifier '$01'.
Test conditions: ignition = on, engine = off, vehicle speed = 0 [kph]

10.1.5.2 Message flow

STEP#1 startRoutineByLocalIdentifier(RELI_ IOSITR)

time	Client (tester) Request Message	Hex
P3	**startRoutineByLocalId.Request**[routineLocalId = inputOutputSignalIntermittentTestRoutine]	**31** 01

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**startRoutineByLocalId.PosRsp**[routineLocalId = inputOutputSignalIntermittent-TestRoutine]	**71** 01	**negativeResponse Service Identifier** **startRoutineByLocalId.ReqSId[** responseCode { refer to section 4.4 }]	**7F** **31** xx
	return(main)		**return(responseCode)**	

10.2 StartRoutineByAddress service

10.2.1 Message description

The startRoutineByAddress service is used by the client to start a routine in the server's memory.
The ECU routine shall be started in the server's (ECU's) memory some time between the completion of the startRoutineByAddress request message and the completion of the first response message if the response message is positive.
The routines could either be tests that run instead of normal operating code or could be routines that are enabled and executed with the normal operating code running. Particularly in the first case, it might be necessary to switch the server to a specific diagnostic session using the startDiagnosticSession service or to unlock the server using the securityAccess service prior to using the startRoutineByAddress service.
The parameter routineEntryOption is user optional. Examples of routineEntryOptions are: timeToRun, startUpVariables, etc.

10.2.2 Message data bytes

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**startRoutineByAddress Request Service Id**	**M**	**38**	**STRBA**
#2	routineAddress (High Byte)	M	xx	**RA_...**
#3	routineAddress (Middle Byte)	M	xx	
#4	routineAddress (Low Byte)	M	xx	
#5	routineEntryOption#1	U	xx	**REYO_...**
:	:	:	:	:
#n	routineEntryOption#m	U	xx	**REYO_...**

Table 10.2.2.1 - StartRoutineByAddress Request Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**startRoutineByAddress Positive Response Service Id**	**M**	**78**	**STRBAPR**
#2	routineAddress (High Byte)	M	xx	**RA_...**
#3	routineAddress (Middle Byte)	M	xx	
#4	routineAddress (Low Byte)	M	xx	
#5	routineEntryStatus#1	U	xx	**REYS_...**
:	:	:	:	:
#n	routineEntryStatus#m	U	xx	**REYS_...**

Table 10.2.2.2 - StartRoutineByAddress Positive Response Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**negativeResponse Service Id**	**M**	**7F**	**NR**
#2	**startRoutineByAddress Request Service Id**	**M**	**38**	**STRBA**
#3	responseCode=[KWP2000ResponseCode { section 4.4 }]	M	xx=[00-FF]	**RC_...**

Table 10.2.2.3 - StartRoutineByAddress Negative Response Message

10.2.3 Parameter definition

- The parameter ***routineAddress (RA_)*** in the start/stopRoutineByAddress and requestRoutine-ResultsByAddress request message identifies a server local routine.
- The parameter ***routineEntryOption (REYO_)*** is defined in section 10.1.3.
- The parameter ***routineEntryStatus (REYS_)*** is defined in section 10.1.3.

10.2.4 Message flow example

See message flow example of Physical Addressed Service in section 5.3.1.1.
A complete message flow example is shown in section 10.5.4.

10.2.5 Implementation example of "Input/output signal intermittent test routine"

10.2.5.1 Message flow conditions for "Input/output signal intermittent test routine" (IOSITR)

This section specifies the message flow conditions to start a routine at a specific address in the server's (ECU's) memory to continuously test (as fast as possible) all input and output signals on intermittence while a technician would "wiggle" all wiring harness connectors of the system under test. The routineAddress to access this routine shall be '$604720'.
Test conditions: ignition = on, engine = off, vehicle speed = 0 [kph]

10.2.5.2 Message flow

STEP#1 startRoutineByAddress(RA_...)

time	Client (tester) Request Message	Hex
P3	**startRoutineByAddress.Request**[	**38**
	routineAddress { High Byte }	60
	routineAddress { Middle Byte }	47
	routineAddress { Low Byte }]	20

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**startRoutineByAddress.PosRsp**[	**78**	**negativeResponse Service Identifier**	**7F**
	routineAddress{High Byte}	60	**startRoutineByAddress.ReqSId[**	**38**
	routineAddress { Middle Byte }	47	responseCode { refer to section 4.4 }]	xx
	routineAddress { Low Byte }]	20		
	return(main)		**return(responseCode)**	

10.3 StopRoutineByLocalIdentifier service

10.3.1 Message description

The stopRoutineByLocalIdentifier service is used by the client to stop a routine in the server's memory referenced by a routineLocalIdentifier. The parameter routineExitOption is user optional.
The ECU routine shall be stopped in the server's (ECU's) memory some time after the completion of the stopRoutineByLocalIdentifier/Address request message and the completion of the first response message if the response message is positive.
Examples of routineExitOption are: timeToExpireBeforeRoutineStops, variables, etc. The parameter routineExitStatus is user optional. Examples of routineExitStatus are: totalRunTime, results generated by the routine before stopped, etc.

10.3.2 Message data bytes

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**stopRoutineByLocalIdentifier Request Service Id**	**M**	**32**	**SPRBLI**
#2	routineLocalIdentifier	M	xx	**RELI_...**
#3 : #n	routineExitOption#1 : routineExitOption#m	U : U	xx : xx	**RETO_...** **:** **RETO_...**

Table 10.3.2.1 - StopRoutineByLocalIdentifier Request Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**stopRoutineByLocalIdentifier Positive Response**	**M**	**72**	**SPRBLIPR**
#2	routineLocalIdentifier	M	xx	**RELI_...**
#3 : #n	routineExitStatus#1 : routineExitStatus#m	U : U	xx : xx	**RETS_...**

Table 10.3.2.2 - StopRoutineByLocalIdentifier Positive Response Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**negativeResponse Service Id**	**M**	**7F**	**NR**
#2	**stopRoutineByLocalIdentifier Request Service Id**	**M**	**32**	**SPRBLI**
#3	responseCode=[KWP2000ResponseCode { section 4.4 }]	M	xx=[00-FF]	**RC_...**

Table 10.3.2.3 - StopRoutineByLocalIdentifier Negative Response Message

10.3.3 Parameter definition

- The parameter ***routineLocalIdentifier (RELI_)*** is defined in section 10.1.3.
- The parameter ***routineExitOption (RETO_)*** is used by the stopRoutineByLocalIdentifier and stopRoutineByAddress request message to specify the stop conditions of the routine.
- The parameter ***routineExitStatus (RETS_)*** is used by the stopRoutineByLocalIdentifier and stopRoutineByAddress positive response message to provide additional information to the client about the status of the server after the routine has been stopped. Values are defined in the table below:

Hex	Description	Cvt	Mnemonic
00-60	**reservedByDocument** This range of values is reserved by this document.	**M**	**RBD**
61	**normalExitWithResultsAvailable**	**U**	**NEWRA**
62	**normalExitWithoutResultsAvailable**	**U**	**NEWORA**
63	**abnormalExitWithResultsAvailable**	**U**	**AEWRA**
64	**abnormalExitWithoutResultsAvailable**	**U**	**AEWORA**
65 - 7F	**reservedByDocument** This range of values is reserved by this document.	**M**	**RBD**
80 - F9	**vehicleManufacturerSpecific** This range of values is reserved for vehicle manufacturer specific use.	**U**	**VMS**

Hex	Description	Cvt	Mnemonic
FA - FE	**systemSupplierSpecific** This range of values is reserved for system supplier specific use.	**U**	**SSS**
FF	**reservedByDocument** This value is reserved by this document.	**M**	**RBD**

Table 10.3.1 - Definition of routineExitStatus values

10.3.4 Message flow example

See message flow examples of a Physically Addressed Service in section 5.3.1.1 and a Functionally Addressed Service in section 5.3.2.1. A complete message flow example is shown in section 10.5.4.

10.3.5 Implementation example of "Input/output signal intermittent test routine"

10.3.5.1 Message flow conditions for "Input/output signal intermittent test routine" (IOSITR)

This section specifies the message flow conditions to stop a routine in the server (ECU) which has continuously tested (as fast as possible) all input and output signals on intermittence while a technician would have been "wiggled" all wiring harness connectors of the system under test. The routineLocalIdentifier references this routine by the routineLocalIdentifier '$01'. Test conditions: ignition = on, engine = off, vehicle speed = 0 [kph]

10.3.5.2 Message flow

STEP#1 stopRoutineByLocalIdentifier(RELI_IOSITR)

time	Client (tester) Request Message	Hex
P3	**stopRoutineByLocalId.Request**[routineLocalId = inputOutputSignalIntermittentTestRoutine]	**32** 01

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**stopRoutineByLocalId.PosRsp**[routineLocalId = inputOutputSignalIntermittent-TestRoutine]	**72** 01	**negativeResponse Service Identifier** **stopRoutineByLocalId.ReqSId[** responseCode { refer to section 4.4 }]	**7F** **32** xx
	return(main)		**return(responseCode)**	

10.4 StopRoutineByAddress service

10.4.1 Message description

The stopRoutineByAddress service is used by the client to stop a routine in the server's memory referenced by a memoryAddress. The parameter routineExitOption is user optional.
The ECU routine shall be stopped in the server's (ECU's) memory some time after the completion of the stopRoutineByAddress request message and the completion of the first response message if the response message is positive.
Examples of routineExitOption are: timeToExpireBeforeRoutineStops, variables, etc.
The parameter routineExitStatus is user optional. Examples of routineExitStatus are: totalRunTime, results generated by the routine before stopped, etc.

10.4.2 Message data bytes

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**stopRoutineByAddress Request Service Id**	**M**	**39**	**SPRBA**
#2	routineAddress (High Byte)	M	xx	**RA_...**
#3	routineAddress (Middle Byte)	M	xx	
#4	routineAddress (Low Byte)	M	xx	
#5	routineExitOption#1	U	xx	**RETO_...**
:	:	:	:	:
#n	routineExitOption#m	U	xx	**RETO_...**

Table 10.4.2.1 - StopRoutineByAddress Request Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**stopRoutineByAddress Positive Response Service Id**	**M**	**79**	**SPRBAPR**
#2	routineAddress (High Byte)	M	xx	**RA_...**
#3	routineAddress (Middle Byte)	M	xx	
#4	routineAddress (Low Byte)	M	xx	
#5	routineExitStatus#1	U	xx	**RETS_...**
:	:	:	:	:
#n	routineExitStatus#m	U	xx	**RETS_...**

Table 10.4.2.2 - StopRoutineByAddress Positive Response Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**negativeResponse Service Id**	**M**	**7F**	**NR**
#2	**stopRoutineByAddress Request Service Id**	**M**	**39**	**SPRBA**
#3	responseCode=[KWP2000ResponseCode { section 4.4 }]	M	xx=[00-FF]	**RC_...**

Table 10.4.2.3 - StopRoutineByAddress Negative Response Message

10.4.3 Parameter definition

- The parameter ***routineAddress (RA_)*** is defined in section 10.2.3.
- The parameter ***routineExitOption (RETO_)*** is defined in section 10.3.3.
- The parameter ***routineExitStatus (RETS_)*** is defined in section 10.3.3.

10.4.4 Message flow example

See message flow example of Physical Addressed Service in section 5.3.1.1. A complete message flow example is shown in section 10.5.4.

10.4.5 Implementation example of "Input/output signal intermittent test routine" (IOSITR)

10.4.5.1 Message flow conditions for "Input/output signal intermittent test routine"

This section specifies the message flow conditions to stop a routine at a specific address in the server's (ECU's) memory to continuously test (as fast as possible) all input and output signals on intermittence while a technician would "wiggle" all wiring harness connectors of the system under test. The routineAddress to access this routine is '$604720'.
Test conditions: ignition = on, engine = off, vehicle speed = 0 [kph]

10.4.5.2 Message flow

STEP#1 stopRoutineByAddress(RA_...)

time	Client (tester) Request Message	Hex
P3	**stopRoutineByAddress.Request**[	**39**
	routineAddress { High Byte }	60
	routineAddress { Middle Byte }	47
	routineAddress { Low Byte }]	20

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**stopRoutineByAddress.PosRsp**[	**79**	**negativeResponse Service Identifier**	**7F**
	routineAddress {High Byte}	60	**stopRoutineByAddress.ReqSId**[	**39**
	routineAddress { Middle Byte }	47	responseCode { refer to section 4.4 }]	xx
	routineAddress { Low Byte }]	20		
	return(main)		**return(responseCode)**	

10.5 RequestRoutineResultsByLocalIdentifier service

10.5.1 Message description

The requestRoutineResultsByLocalIdentifier service is used by the client to request results referenced by a routineLocalIdentifier and generated by the routine which was executed in the server's memory. The parameter routineResults is user optional. The requestRoutineResultsByLocalIdentifier service can be used based on the parameter routineExitStatus, which may have been received in the stopRoutineByLocalIdentifier or stopRoutineByAddress positive response message (routineExitStatus = normal/abnormalExitWithResults). Example of routineResults: data collected by the ECU which could not be transmitted during routine execution because of ECU performance limitations.

10.5.2 Message data bytes

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**requestRoutineResultsByLocalIdentifier Request Service Id**	**M**	**33**	**RRRBLI**
#2	routineLocalIdentifier	M	xx	**RELI_...**

Table 10.5.2.1 - RequestRoutineResultsByLocalIdentifier Request Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**requestRoutineResultsByLocalIdentifier Pos. Resp. Service Id**	**M**	**73**	**RRRBLIPR**
#2	routineLocalIdentifier	M	xx	**RELI_...**
#3 : #n	routineResults#1 : routineResults#m	M : U	xx : xx	**RRS_...** : **RRS_...**

Table 10.5.2.2 - RequestRoutineResultsByLocalIdentifier Positive Response Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**negativeResponse Service Id**	**M**	**7F**	**NR**
#2	**requestRoutineResultsByLocalIdentifier Request Service Id**	**M**	**33**	**RRRBLI**
#3	responseCode=[KWP2000ResponseCode { section 4.4 }]	M	xx=[00-FF]	**RC_...**

Table 10.5.2.3 - RequestRoutineResultsByLocalIdentifier Negative Response Message

10.5.3 Parameter definition

- The parameter ***routineLocalIdentifier (RELI_)*** is defined in section 10.1.3.
- The parameter ***routineResults (RRS_)*** is used by the requestRoutineResultsByLocalIdentifier and requestRoutineResultsByAddress positive response message to provide the results (e.g. exit status information) of the routine which has been stopped previously in the server.

10.5.4 Message flow example

See message flow examples of a Physically Addressed Service in section 5.3.1.1 and a Functionally Addressed Service in section 5.3.2.1. A complete message flow example is shown in the table below:

time	client (Tester)	server (ECU)
P3 P2	startRoutineByLocalId.Request[...]	 startRoutineByLocalId.PositiveResponse[...]
P3 P2	stopRoutineByLocalId.Request[...]	 stopRoutineByLocalId.PositiveResponse[...]
P3 P2	requestRoutineResultsByLocalId.Request[...]	 requestRoutineResultsByLocalId.PositiveResponse[...]

Table 10.5.4 - Message flow example of start/stopRoutineByLocalIdentifier service followed by a requestRoutineResultsByLocalIdentifier service

10.5.5 Implementation example of "Input/output signal intermittent test routine"

10.5.5.1 Message flow conditions for "Input/output signal intermittent test routine" (IOSITR)

This section specifies the message flow conditions to stop a routine in the server (ECU) which has continuously tested as fast as possible all input and output signals on intermittence while a technician would

have been "wiggled" at all wiring harness connectors of the system under test. The routineLocalIdentifier to reference this routine is '$01'.
Test conditions: ignition = on, engine = off, vehicle speed = 0 [kph]

10.5.5.2 Message flow

STEP#1 requestRoutineResultsByLocalIdentifier(RELI_IOSITR)

time	Client (tester) Request Message	Hex
P3	**requestRoutineResultsByLocalId.Request**[	**33**
	routineLocalId = inputOutputSignalIntermittentTestRoutine]	01

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**requestRoutineResultsByLocalId.PosRsp**[	**73**	**negativeResponse Service Identifier**	**7F**
	routineLocalId = inputOutputSignalIntermittent-TestRoutine	01	**requestRoutineResultsByLocalId.ReqSId[**	**33**
			responseCode { refer to section 4.4 }]	xx
	routineResult#1 = inputSignal#1	57		
	routineResult#2 = inputSignal#2	33		
	:	**:**		
	routineResult#m = inputSignal#m]	8F		
	return(main)		**return(responseCode)**	

10.6 RequestRoutineResultsByAddress service

10.6.1 Message description

The requestRoutineResultsByAddress service is used by the client to request results referenced by a memoryAddress and generated by the routine which was executed in the server's memory.
The parameter routineResults is user optional. The requestRoutineResultsByAddress service can be used based on the parameter routineExitStatus which may have been received in the stopRoutineByLocalIdentifier or stopRoutineByAddress positive response message (routineExitStatus = normal/abnormalExitWithResults).
Example of routineResults: data collected by the ECU which could not be transmitted during routine execution because of ECU performance limitations.

10.6.2 Message data bytes

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**requestRoutineResultsByAddress Request Service Id**	**M**	**3A**	**RRRBA**
#2 #3 #4	routineAddress (High Byte) routineAddress (Middle Byte) routineAddress (Low Byte)	M M M	xx xx xx	**RA_...**

Table 10.6.2.1 - RequestRoutineResultsByAddress Request Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**requestRoutineResultsByAddress Positive Response Service Id**	**M**	**7A**	**RRRBA**
#2 #3 #4	routineAddress (High Byte) routineAddress (Middle Byte) routineAddress (Low Byte)	M M M	xx xx xx	**RA_...**
#5 : #n	routineResults#1 : routineResults#m	M : U	xx : xx	**RRS_...** : **RRS_...**

Table 10.6.2.2 - RequestRoutineResultsByAddress Positive Response Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**Negative Response Service Id**	**M**	**7F**	**NR**
#2	**requestRoutineResultsByAddress Request Service Id**	**M**	**3A**	**RRRBA**
#3	responseCode=[KWP2000ResponseCode { section 4.4 }]	M	xx=[00-FF]	**RC_...**

Table 10.6.2.3 - RequestRoutineResultsByAddress Negative Response Message

10.6.3 Parameter definition

- The parameter ***routineAddress (RA_)*** is defined in section 10.2.3.
- The parameter ***routineResults (RRS_)*** is defined in section 10.5.3.

10.6.4 Message flow example

See message flow example of Physical Addressed Service in section 10.5.4.

Note: Above mentioned message flow example uses a localIdentifier instead of a memoryAddress.

10.6.5 Implementation example of "Input/output signal intermittent test routine"

10.6.5.1 Message flow conditions for "Input/output signal intermittent test routine" (IOSITR)

This section specifies the message flow conditions to stop a routine in the server (ECU) which has continuously tested (as fast as possible) all input and output signals on intermittence while a technician would have been "wiggled" all wiring harness connectors of the system under test. The routineAddress to access this routine is '$204720'.
Test conditions: ignition = on, engine = off, vehicle speed = 0 [kph]

10.6.5.2 Message flow

STEP#1 requestRoutineResultsByAddress(RA_...)

time	Client (tester) Request Message	Hex
P3	**requestRoutineResultsByAddress.Request**[	**3A**
	routineAddress { High Byte }	20
	routineAddress { Middle Byte }	47
	routineAddress { Low Byte }]	20

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**requestRoutineResultsByAddress.PosRsp**[	**7A**	**negativeResponse Service Identifier**	**7F**
	routineAddress { High Byte }	20	**requestRoutineResultsByAddress.ReqSId**[	**3A**
	routineAddress { Middle Byte }	47	responseCode { refer to section 4.4 }]	xx
	routineAddress { Low Byte }	20		
	routineResult#1 = inputSignal#1	57		
	routineResult#2 = inputSignal#2	33		
	:	**:**		
	routineResult#m = inputSignal#m]	8F		
	return(main)		**return(responseCode)**	

11 Upload Download functional unit

The services provided by this functional unit are described in table 11:

Service name	Description
RequestDownload	The client requests the negotiation of a data transfer from the client to the server.
RequestUpload	The client requests the negotiation of a data transfer from the server to the client.
TransferData	The client transmits data to the server (download) or requests data from the server (upload).
RequestTransferExit	The client requests the termination of a data transfer.

Table 11 - Upload Download functional unit

11.1 RequestDownload service

11.1.1 Message description

The requestDownload service is used by the client to initiate a data transfer from the client to the server (download). After the server has received the requestDownload request message the ECU shall take all necessary actions to receive data before it sends a positive response message.
The parameters transferRequestParameter and transferResponseParameter are user optional and shall be of any type and length defined within KWP 2000.
Example of transferRequestParameter: initialisation value(s) for data download.
Example of transferResponseParameter: maximum number of bytes per transferData message.

11.1.2 Message data bytes

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**requestDownload Request Service Id**	**M**	**34**	**RD**
	transferRequestParameter=[			**TRTP_...**
#2	memoryAddress { High Byte }	M	xx	**MA_...**
#3	memoryAddress { Middle Byte }	M	xx	**MA_...**
#4	memoryAddress { Low Byte }	M	xx	**MA_...**
#5	dataFormatIdentifier	M	xx	**DFI_...**
#6	unCompressedMemorySize { High Byte }	M	xx	**UCMS**
#7	unCompressedMemorySize { Middle Byte }	M	xx	**UCMS**
#8	unCompressedMemorySize { Low Byte }]	M	xx	**UCMS**

Table 11.1.2.1 - RequestDownload Request Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**requestDownload Positive Response Service Id**	**M**	**74**	**RDPR**
#2	transferResponseParameter=[maxNumberOfBlockLength]	M	xx	**TREP_...** **MNROBL**

Table 11.1.2.2 - RequestDownload Positive Response Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**negativeResponse Service Id**	**M**	**7F**	**NR**
#2	**requestDownload Request Service Id**	**M**	**34**	**RD**
#3	responseCode=[KWP2000ResponseCode { section 4.4 }]	M	xx=[00-FF]	**RC_...**

Table 11.1.2.3 - RequestDownload Negative Response Message

11.1.3 Parameter definition

- The ***transferRequestParameter (TRTP_)*** is used by the requestDownload request message. This parameter consists of several other parameters which are required by the server (ECU) to prepare for the data transfer from the client (tester) to the server (ECU). Each parameter is specified in detail:
- The parameter ***memoryAddress (MA_)*** is defined in section 7.3.3.
- The parameter ***dataFormatIdentifier (DFI_)*** is a one byte identifier. Each nibble shall be encoded separately. The high nibble specifies the "**compressionMethod**" **(CM_)** and the low nibble specifies the "**encryptingMethod**" **(EM_)**. The use of a data compression method saves overall data transfer time if large amount of data shall be transferred between client (tester) and server (ECU). Using an encoding method improves tamper protection. The values are specified in the table below:

Hex	Description of Compression and Encoding Method	Cvt	Mnemonic
0x	**unCompressed**	**M**	**UC**
1x - Fx	This range of values shall be reserved by the vehicle manufacturer / system supplier specific Compression Methods	**U**	**C_...**
x0	**unEncrypted**	**M**	**UE**
x1 - xF	vehicle manufacturer / system supplier specific Encrypting Methods	**U**	**E_...**

Table 11.1.1.1 - Definition of dataFormatIdentifier value

- The parameter ***unCompressedMemorySize (UCMS)*** is a three (3) byte value. This parameter shall be used by the server (ECU) to compare the uncompressed memory size with the total amount of data transferred during the requestTransferExit service. This increases the programming security.
- The ***transferResponseParameter (TREP_)*** "***maxNumberOfBlockLength (MNROBL)"*** is used by the requestDownload positive response message to inform the client (tester) how many data bytes shall be included in each transferData request message from the client (tester). This parameter allows the client (tester) to adapt to the receive buffer size of the server (ECU) before it starts transferring data to the server (ECU).

11.1.4 Message flow example

See message flow example of Physical Addressed Service in section 5.3.1.1.
A complete message flow example is shown in section 11.4.4.

11.1.5 Implementation example of "requestDownload"

Section **11.4.5.1 - Implementation example of "requestDownload, transferData, requestTransferExit"** includes a complete implementation example of requestDownload, followed by multiple transferData services and terminated by a requestTransferExit service.

11.2 RequestUpload service

11.2.1 Message description

The requestUpload service is used by the client to initiate a data transfer from the server to the client (upload). After the server has received the requestUpload request message the ECU shall take all necessary actions to send data before it sends a positive response message.
The parameters transferRequestParameter and transferResponseParameter are user optional and shall be of any type and length defined within KWP 2000.
Example of transferRequestParameter: initialisation value(s) for data upload.
Example of transferResponseParameter: maximum number of bytes per transferData message.

11.2.2 Message data bytes

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**requestUpload Request Service Id**	**M**	**35**	**RU**
	transferRequestParameter=[			**TRTP_...**
#2	memoryAddress { High Byte }	M	xx	**MA_...**
#3	memoryAddress { Middle Byte }	M	xx	**MA_...**
#4	memoryAddress { Low Byte }	M	xx	**MA_...**
#5	dataFormatIdentifier	M	xx	**DFI_...**
#6	unCompressedMemorySize { High Byte }	M	xx	**UCMS**
#7	unCompressedMemorySize { Middle Byte }	M	xx	**UCMS**
#8	unCompressedMemorySize { Low Byte }]	M	xx	**UCMS**

Table 11.2.2.1 - RequestUpload Request Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**requestUpload Positive Response Service Id**	**M**	**75**	**RUPR**
#2	transferResponseParameter=[maxNumberOfBlockLength]	M	xx	**TREP_...** **MNROBL**

Table 11.2.2.2 - RequestUpload Positive Response Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**negativeResponse Service Id**	**M**	**7F**	**NR**
#2	**requestUpload Request Service Id**	**M**	**35**	**RU**
#3	responseCode=[KWP2000ResponseCode { section 4.4 }]	M	xx=[00-FF]	**RC_...**

Table 11.2.2.3 - RequestUpload Negative Response Message

11.2.3 Parameter definition

- The ***transferRequestParameter (TRTP_)*** is used by the requestUpload request message. This parameter consists of several other parameters which are required by the server (ECU) to prepare for the data transfer from the server (ECU) to the client (tester). Each parameter is specified in detail:
- The parameter ***memoryAddress (MA_)*** is defined in section 7.3.3.
- The parameter ***dataFormatIdentifier (DFI_)*** is a one byte identifier. Each nibble shall be encoded separately. The high nibble specifies the "**compressionMethod**" **(CM_)** and the low nibble specifies the "**encryptingMethod**" **(EM_)**. The use of a data compression method saves overall data transfer time if large amount of data shall be transferred between server (ECU) and client (tester). Using an encoding method improves tamper protection. The values are specified in the table below:

Hex	Description of Compression and Encoding Method	Cvt	Mnemonic
0x	**unCompressed**	**M**	**UC**
1x - Fx	This range of values shall be reserved by the vehicle manufacturer / system supplier specific Compression Methods	**U**	**C_...**
x0	**unEncrypted**	**M**	**UE**
x1 - xF	vehicle manufacturer / system supplier specific Encrypting Methods	**U**	**E_...**

Table 11.2.1.1 - Definition of dataFormatIdentifier value

- The parameter ***uncompressedMemorySize (UCMS)*** is a three (3) byte value. This parameter shall be used by the server (ECU) to compare the uncompressed memory size with the total amount of data transferred during the requestTransferExit service.
- The ***transferResponseParameter (TREP_)*** "***maxNumberOfBlockLength (MNROBL)"*** is used by the requestUpload positive response message to inform the client (tester) how many data bytes will be included in each transferData positive response message from the server (ECU).

11.2.4 Message flow example

See message flow example of Physically Addressed Service in section 5.3.1.1. A complete message flow example is shown in section 11.4.4.

11.2.5 Implementation example of "requestUpload"

Section **11.4.5.2 - Implementation example of "requestUpload, transferData, requestTransferExit"** includes a complete implementation example of requestUpload, followed by multiple transferData services and terminated by a requestTransferExit service.

11.3 TransferData service

11.3.1 Message description

The transferData service is used by the client to transfer data either from the client to the server (download) or from the server to the client (upload). The data transfer direction is defined by the preceding requestDownload or requestUpload service.
The user optional parameters transferRequestParameter and transferResponseParameter in the transferData request and positive response messages shall be of any type and length defined within KWP 2000.
If the client initiated a requestDownload the data to be downloaded can be included in the parameter(s) transferRequestParameter in the transferData request message(s). The server may include the blockTransferComplete/NextBlock parameter as an transferResponseParameter in the transferData positive response message.
If the client initiated a requestUpload the data to be uploaded can be included in the parameter(s) transferResponseParameter in the transferData positive response message(s). The client may include the blockTransferComplete/NextBlock parameter as an transferRequestParameter in the transferData request message.
The transferData service shall only be terminated by the requestTransferExit service.
In order to increase tamper protection the transferRequestParameters and transferResponseParameters shall not include obvious absolute server (ECU) memoryAddress information !

11.3.2 Message data bytes

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**transferData Request Service Id**	**M**	**36**	**TD**
#2 : #n	transferRequestParameter#1 : transferRequestParameter#m	U : U	xx : xx	**TRTP_...** : **TRTP_...**

Table 11.3.2.1 - TransferData Request Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**transferData Positive Response Service Id**	**M**	**76**	**TDPR**
#2 : #n	transferResponseParameter#1 : transferResponseParameter#m	U : U	xx : xx	**TREP_...** : **TREP_...**

Table 11.3.2.2 - TransferData Positive Response Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**negativeResponse Service Id**	**M**	**7F**	**NR**
#2	**transferData Request Service Id**	**M**	**36**	**TD**
#3	responseCode=[KWP2000ResponseCode { section 4.4 }]	M	xx=[00-FF]	**RC_...**

Table 11.3.2.3 - TransferData Negative Response Message

11.3.3 Parameter definition

- The parameter ***transferRequestParameter (TRTP_)*** in the transferData request message shall contain parameter(s) which are required by the server to support the transfer of data. Format and length of this parameter(s) are vehicle manufacturer specific.
- The parameter ***transferResponseParameter (TREP_)*** in the transferData positive response message shall contain parameter(s) which are required by the client to support the transfer of data. Format and length of this parameter(s) are vehicle manufacturer specific.

Example of transferRequestParameter and transferResponseParameter:

- For a download, the parameter transferRequestParameter could include the data to be transferred and the parameter transferResponseParameter a checksum computed by the server.
- For an upload, the parameter transferRequestParameter parameter could include the address and number of bytes to retrieve data and the parameter transferResponseParameter the uploaded data.

11.3.4 Message flow example

time	client (Tester)	server (ECU)
P3	transferData.Request#1[...]	
P2		transferData.PositiveResponse#1[...]
P3	transferData.Request#2[...]	
P2		transferData.PositiveResponse#2[...]
:	**:**	**:**
P3	transferData.Request#n[...]	
P2		transferData.PositiveResponse#n[...]

Table 11.3.4 - Message flow example of transferData Service

11.3.5 Implementation example of "transferData"

Section **11.4.5.1 - Implementation example of "requestDownload, transferData, requestTransferExit"** includes a complete implementation example of requestDownload, followed by multiple transferData services and terminated by a requestTransferExit service.
Section **11.4.5.2 - Implementation example of "requestUpload, transferData, requestTransferExit"** includes a complete implementation example of requestUpload, followed by multiple transferData services and terminated by a requestTransferExit service.

11.4 RequestTransferExit service

11.4.1 Message description

This service is used by the client to terminate a data transfer between client and server. The user optional parameters transferRequestParameter and transferResponseParameter in the requestTransferExit request and positive response message shall be of any type and length defined within KWP 2000.

11.4.2 Message data bytes

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**requestTransferExit Request Service Id**	**M**	**37**	**RTE**
#2 : #n	transferRequestParameter#1 : transferRequestParameter#m	U : U	xx : xx	**TRTP_...** : **TRTP_...**

Table 11.4.2.1 - RequestTransferExit Request Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**requestTransferExit Positive Response Service Id**	**M**	**77**	**RTEPR**
#2 : #n	transferResponseParameter#1 : transferResponseParameter#m	U : U	xx : xx	**TREP_...** : **TREP_...**

Table 11.4.2.2 - RequestTransferExit Positive Response Message

Data Byte	Parameter Name	Cvt	Hex Value	Mnemonic
#1	**negativeResponse Service Id**	**M**	**7F**	**NR**
#2	**requestTransferExit Request Service Id**	**M**	**37**	**RTE**
#3	responseCode=[KWP2000ResponseCode { section 4.4 }]	M	xx=[00-FF]	**RC_...**

Table 11.4.2.3 - RequestTransferExit Negative Response Message

11.4.3 Parameter definition

- The parameter ***transferRequestParameter (TRTP_)*** shall contain parameter(s) which are required by the server to support the data transfer exit. Format and length of this parameter is vehicle manufacturer specific.
  Example of transferRequestParameter: checksum request of entire data transferred.
- The parameter ***transferResponseParameter (TREP_)*** in the requestTransferExit positive response message defines the status of the server's data transfer exit status. Format and length of additional parameters are vehicle manufacturer specific.
  Example of additional transferResponseParameter: checksum of entire data transferred.

11.4.4 Message flow example

time	client (Tester)	server (ECU)
P3	requestUpload.Request[...]	
P2		requestUpload.PositiveResponse[...]
P3	transferData.Request#1[...]	
P2		transferData.PositiveResponse#1[...]
P3	transferData.Request#2[...]	
P2		transferData.PositiveResponse#2[...]
:	:	:
P3	transferData.Request#n[...]	
P2		transferData.PositiveResponse#n[...]
P3	requestTransferExit.Request[...]	
P2		requestTransferExit.PositiveResponse[...]

Table 11.4.4 - Message flow example of requestUpload, multiple transferData services followed by a requestTransferExit service

11.4.5 Implementation example of "requestTransferExit"

11.4.5.1 Message flow conditions for "requestDownload, transferData, requestTransferExit"

11.4.5.1.1 Implementation example of "requestDownload, transferData, requestTransferExit"

This section specifies the conditions to transfer data (download) from the client (tester) to the server (ECU). The example consists of three (3) steps.
In the **1st** step the client (tester) and the server (ECU) execute a requestDownload service. With this service the following information is exchanged as parameters in the request and positive response message between client (tester) and the server (ECU):

Data Parameter Name	Data Parameter Value(s) (Hex)	Data Parameter Description
transferRequestParameter#1 - #3	$602000	memoryAddress (start) to download data to
transferRequestParameter#4	$11	dataFormatIdentifier (compressionMethod = $1x) (encryptingMethod = $x1)
transferRequestParameter#5 - #7	$00FFFF	uncompressedMemorySize = (64 Kbytes) This parameter value shall be used by the server (ECU) to compare to the actual number of bytes transferred during the execution of the requestTransferExit service.

Table 11.4.5.1.1 - Definition of transferRequestParameter values

Data Parameter Name	Data Parameter Value(s) (Hex)	Data Parameter Description
transferResponseParameter#1	$81	maximumNumberOfBlockLength: (serviceId + 128 ECU data bytes = 129 data bytes)

Table 11.4.5.1.2 - Definition of transferResponseParameter value

In the **2nd** step the client (tester) transfers 64 KBytes (number of transferData services with 128 data bytes can not be calculated because the compression method and its compression ratio is supplier specific) of data to the flash memory starting at memory address $602000 to the server (ECU).
In the **3rd** step the client (tester) terminates the data transfer to the server (ECU) with a requestTransferExit service.
Test conditions: ignition = on, engine = off, vehicle speed = 0 [kph]

11.4.5.1.2 Message flow

STEP#1 requestDownload(TRTP_MA, TRTP_DFI, TRTP_UCMS)

time	Client (tester) Request Message	Hex
P3	**requestDownload.Request**[	**34**
	transferRequestParameter#1 = memoryAddress { High Byte }	60
	transferRequestParameter#2= memoryAddress { Middle Byte }	20
	transferRequestParameter#3= memoryAddress { Low Byte }	00
	transferRequestParameter#4= dataFormatIdentifier { compressionMethod, encryptingMethod }	11
	transferRequestParameter#5= uncompressedMemorySize { High Byte }	00
	transferRequestParameter#6= uncompressedMemorySize { Middle Byte }	FF
	transferRequestParameter#7= uncompressedMemorySize { Low Byte }]	FF

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**requestDownload.PosRsp**[	**74**	**negativeResponse Service Identifier**	**7F**
	transferRspPara = maxNumberOfBlockLength]	81	**requestDownload.ReqSId**[	**34**
			responseCode { refer to section 4.4 }]	xx
	goto STEP#2		**return(responseCode)**	

STEP#2 transferData#1(TRTP_..., ... , TRTP_...)

time	Client (tester) Request Message	Hex
P3	**transferData.Request#1**[	**36**
	transferData#1 = dataByte#2	xx
	:	**:**
	transferData#128 = dataByte#129]	xx

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**transferData.PosRsp#1**	**76**	**negativeResponse Service Identifier**	**7F**
			transferData.ReqSId[	**36**
			responseCode { refer to section 4.4 }]	xx
	goto STEP#2 transferData#m()		**return(responseCode)**	

:

STEP#2 transferData#m(TRTP_..., ... , TRTP_...)

time	Client (tester) Request Message	Hex
P3	**transferData.Request#m**[	**36**
	transferData#1 = dataByte#2	xx
	:	**:**
	transferData#n-1 = dataByte#n]	xx

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**transferData.PosRsp#m**	**76**	**negativeResponse Service Identifier**	**7F**
			transferData.ReqSId[	**36**
			responseCode { refer to section 4.4 }]	xx
	goto STEP#3		**return(responseCode)**	

STEP#3 requestTransferExit()

time	Client (tester) Request Message	Hex
P3	**requestTransferExit.Request**	**37**

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**requestTransferExit.PosRsp**	**77**	**negativeResponse Service Identifier**	**7F**
			requestTransferExit.ReqSId[	**37**
			responseCode { refer to section 4.4 }]	xx
	return main()		**return(responseCode)**	

11.4.5.2 Message flow conditions for "requestUpload, transferData, requestTransferExit"

11.4.5.2.1 Implementation example of "requestUpload, transferData, requestTransferExit"

This section specifies the conditions to transfer data (upload) from a server (ECU) to the client (tester).
The example consists of three (3) steps. In the **1st** step the client (tester) and the server (ECU) execute a requestUpload service. With this service the following information is exchanged as parameters in the request and positive response message between client (tester) and the server (ECU):

Data Parameter Name	Data Parameter Value(s) (Hex)	Data Parameter Description
transferRequestParameter#1 - #3	$201000	memoryAddress (start) to upload data from
transferRequestParameter#4	$11	dataFormatIdentifier (compressionMethod = $1x) (encryptingMethod = $x1)
transferRequestParameter#5 - #7	$000F00	uncompressedMemorySize = (511 bytes) This parameter value shall indicate how many data bytes shall be transferred and shall be used by the server (ECU) to compare to the actual number of bytes transferred during execution of the requestTransferExit service.

Table 11.4.5.2.1 - Definition of transferRequestParameter values

Data Parameter Name	Data Parameter Value(s) (Hex)	Data Parameter Description

transferResponseParameter#1	$81	maximumNumberOfBlockLength: (serviceId + 128 ECU data bytes = 129 data bytes)

Table 11.4.5.2.2 - Definition of transferResponseParameter value

In the **2nd** step the client (tester) transfers 511 data bytes (3 transferData services with 129 (128 ECU data bytes + 1 serviceId data byte) data bytes and 1 transferData service with 128 (127 ECU data bytes + 1 serviceId data byte) data bytes from the external RAM starting at memory address $201000 in the server (ECU). In the **3rd** step the client (tester) terminates the data transfer to the server (ECU) with a requestTransferExit service.
Test conditions: ignition = on, engine = off, vehicle speed = 0 [kph]

11.4.5.2.2 Message flow

STEP#1 requestUpload(TRTP_MA, TRTP_DFI, TRTP_UCMS)

time	Client (tester) Request Message	Hex
P3	**requestUpload.Request**[	**35**
	transferRequestParameter#1 = memoryAddress { High Byte }	20
	transferRequestParameter#2 = memoryAddress { Middle Byte }	10
	transferRequestParameter#3 = memoryAddress { Low Byte }	00
	transferRequestParameter#4 = dataFormatIdentifier	11
	transferRequestParameter#5 = uncompressedMemorySize { High Byte }	00
	transferRequestParameter#6 = uncompressedMemorySize { Middle Byte }	0F
	transferRequestParameter#7 = uncompressedMemorySize { Low Byte }]	00

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**requestUpload.PosRsp**[	**75**	**negativeResponse Service Identifier**	**7F**
	transferRspPara=[maxNumberOfBlockLength]	81	**requestUpload.ReqSId**[	**35**
			responseCode { refer to section 4.4 }]	xx
	return(main)		**return(responseCode)**	

STEP#2 transferData#1-3()

time	Client (tester) Request Message	Hex
P3	**transferData.Request#1-3**	**36**

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**transferData.PosRsp#1-3**[	**76**	**negativeResponse Service Identifier**	**7F**
	transferData#1 = dataByte#2	xx	**transferData.ReqSId**[	**36**
	:	**:**	responseCode { refer to section 4.4 }]	xx
	transferData#128 = dataByte#129]	xx		
	goto STEP#2 transferData#4()		**return(responseCode)**	

STEP#2 transferData#4()

time	Client (tester) Request Message	Hex
P3	**transferData.Request#4**	**36**

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**transferData.PosRsp#4**[	**76**	**negativeResponse Service Identifier**	**7F**
	transferData#1 = dataByte#2	xx	**transferData.ReqSId**[	**36**
	:	**:**	responseCode { refer to section 4.4 }]	xx
	transferData#127 = dataByte#128]	xx		
	goto STEP#3		**return(responseCode)**	

STEP#3 requestTransferExit()

time	Client (tester) Request Message	Hex
P3	**requestTransferExit.Request**	**37**

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	**requestTransferExit.PosRsp**	**77**	**negativeResponse Service Identifier**	**7F**
			requestTransferExit.ReqSId[	**37**
			responseCode { refer to section 4.4 }]	xx
	return main()		**return(responseCode)**	