INTERNATIONAL STANDARD

ISO 14230-3

First edition
1999-03-15

Road vehicles — Diagnostic systems — Keyword Protocol 2000 —

Part 3:
Application layer

Véhicules routiers — Systèmes de diagnostic — Protocole «Keyword 2000» —

Partie 3: Couche application

Reference number
ISO 14230-3:1999(E)

Contents

Foreword

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

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

International Standard ISO 14230-1 was prepared by Technical Committee ISO/TC 22, *Road vehicles*, subcommittee SC 3, *Electrical and electronic equipment*.

ISO 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*

— *Part 4: Requirements for emissions-related systems*

Annex A of this part of ISO 14230 is for information only.

Introduction

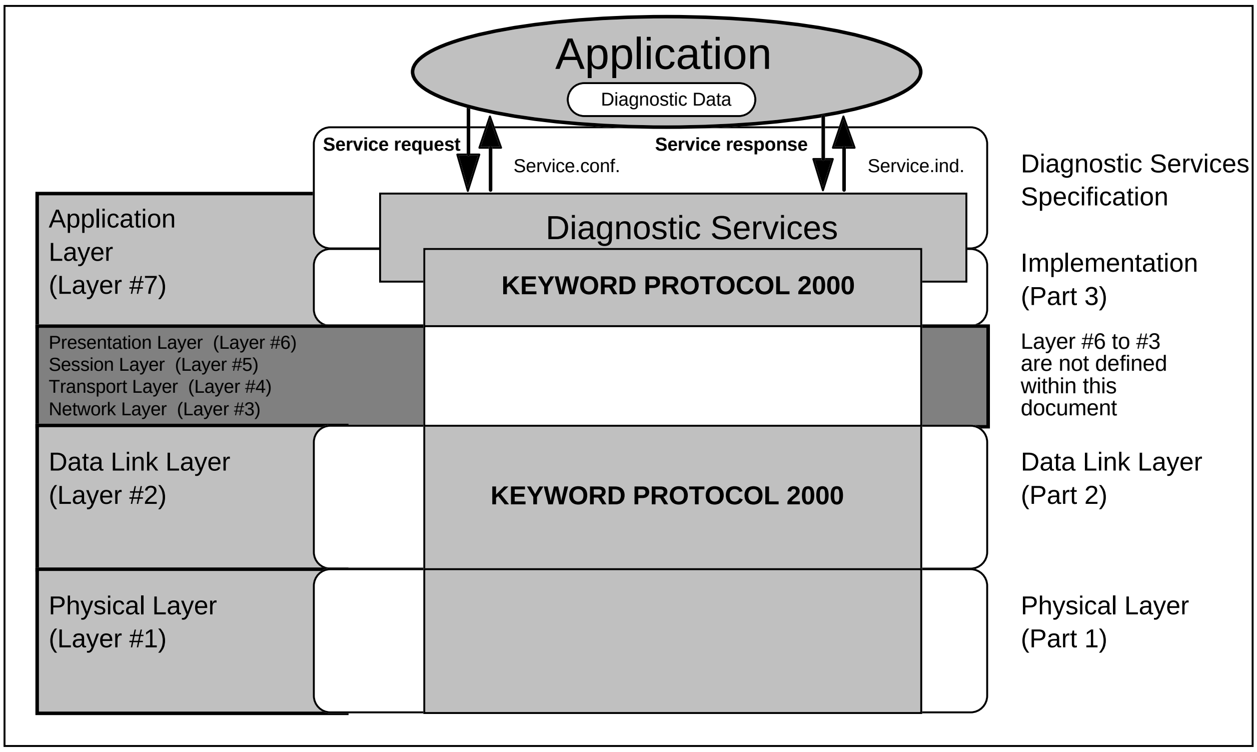
ISO 14230 has been established in order to define common requirements for diagnostic systems implemented on a serial data link.

To achieve this, it is based on the Open Systems Interconnection (OSI) Basic Reference Model in accordance with ISO 7498 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).

See figure 1.



Example of serial data links: KWP2000, VAN, CAN, J1850, etc.

Figure 1 — Mapping of Diagnostic Services and Keyword Protocol 2000 on OSI Model

Road vehicles — Diagnostic systems — Keyword Protocol 2000 —

Part 3:
Application layer

1 Scope

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

This part of ISO 14230 specifies the requirements of 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 requests and responses;

— hexadecimal values for the standard parameters.

The vehicle environment to which this part of ISO 14230 applies may consist of a single tester that may be temporarily connected to the on-vehicle diagnostic data link and several on-vehicle Electronic Control Units connected directly or indirectly (see figure 2).

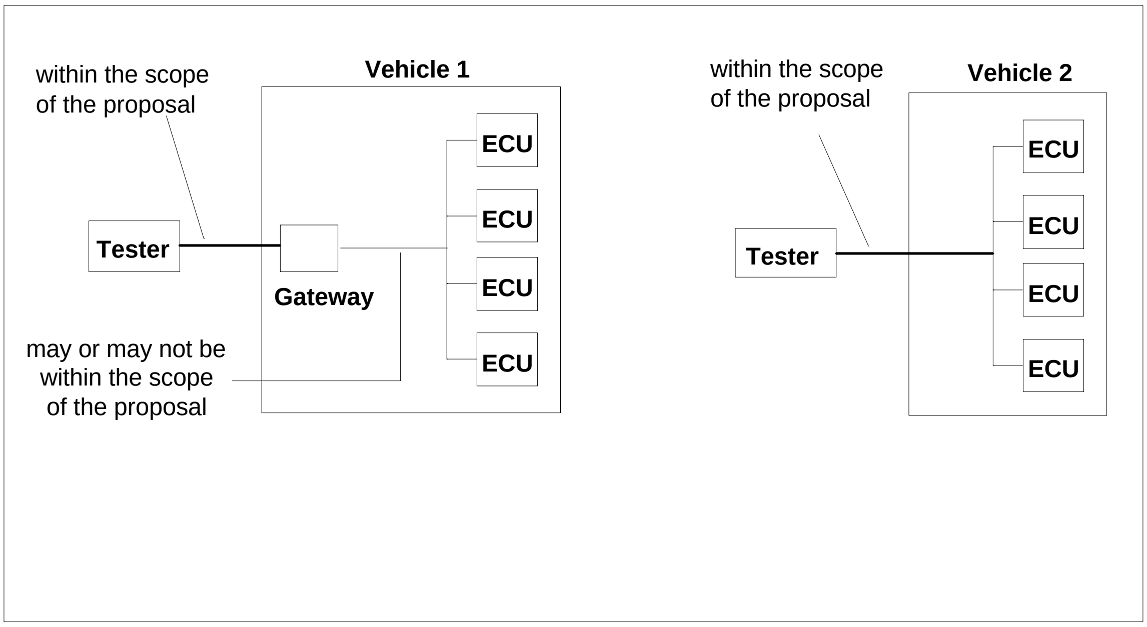

within the scope of the proposal
Vehicle 1
Tester
Gateway
ECU
ECU
ECU
ECU
may or may not be within the scope of the proposal
within the scope of the proposal
Vehicle 2
Tester
ECU
ECU
ECU
ECU

Figure 2 — Vehicle diagnostic architecture

2 Normative references

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 14229:—[1] , *Road vehicles — Diagnostic systems — Diagnostic services specification.*

ISO 14230-2:— [1] , *Road vehicles — Diagnostic systems — Keyword Protocol 2000 — Part 2 : Data link layer.*

SAE J 1930: 1995, *Electrical/electronic systems diagnostic — Terms, definitions, abbreviations and acronyms.*

SAE J 1979: 1997, *E/E diagnostic test modes— Terms, definitions, abbreviations and acronyms.*

3 Definitions

For the purposes of this part of ISO 14230, the definitions given in ISO 14229 and SAE J 1930 apply.

4 Conventions

4.1 General

4.1.1 This part of ISO 14230 is guided by the OSI service conventions (CVT; see ISO 8509) to the extent that they are applicable to the diagnostic services. These conventions define the interactions between the service use and the service provider by the supplier through service primitives which themselves may convey parameters.

4.1.2 Table 1 indicates the different ranges of service identifier values, which are defined in SAE J 1979, ISO 14230 or by the vehicle manufacturer.

Table 1 — Service Identifier value convention table

Service Identifier Hex Value	**Service type[1]** (bit 6)	**Where defined**
00 - 0F	Request	SAE J 1979
10 - 1F 20 - 2F 30 - 3E	Request (bit 6 = 0)	ISO 14230-3
3F	Not applicable	reserved
40 - 4F	Response	SAE J1979
50 - 5F 60 - 6F 70 - 7E	Positive Response to Services ($10 - $3E) (bit 6 = 1)	ISO 14230-3
7F	Negative Response	
80	Request 'ESC' - Code	
81 - 8F	Request (bit 6 = 0)	ISO 14230-2
90 - 9F	Request (bit 6 = 0)	reserved for future exp. as needed
A0 - BF	Request (bit 6 = 0)	defined by vehicle manufacturer
C0	Positive Resp. 'ESC' - Code	ISO 14230-3
C1 - CF	Positive Response (bit 6 = 1)	ISO 14230-2
D0 - DF	Positive Response (bit 6 = 1)	reserved for future exp. as needed
E0 - FF	Positive Response (bit 6 = 1)	defined by vehicle manufacturer

1) 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.

[1] To be published.

4.1.3 The table content consists of the following:

— 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.

4.1.4 For a given primitive, the presence of each parameter is described by one of the following 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;

— S: mandatory (unless specified otherwise) selection of a parameter from a parameter list.

4.2 Service description convention

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

— Parameter Definition;

— Message Data Bytes;

— Message Description;

— Message Flow Example.

4.2.1 Parameter definition

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

4.2.2 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. Table 2 describes the request message, table 3 the positive response message and table 4 the negative response message.

A positive response message shall be given by the server if it can carry out all the operations requested. It shall otherwise give a negative response.

The response messages are listed in separate tables because the list of parameters differs between positive and negative response messages.

Table 2 — Request message

Type	Parameter Name	CVT[1)]	Hex Value	Mnemonic
Header Bytes[2)]	Format Byte Target Byte Source Byte Length Byte	M C[3)] C[3)] C[4)]	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>]	C[5)]	xx=[xx : xx]	PN
CS[2)]	Checksum Byte	M	xx	CS

1) See 4.1.4
2) Defined in ISO 14230-2.
3) The header bytes "Target" and "Source" depend on the content of the "Format Byte" which is specified in ISO 14230-2 (KWP 2000 Part 2: Data Link Layer) document. Both either exist or do not exist in the header of each message.

4): The header byte "Length" depends on the content of the "Format Byte" which is specified in ISO DIS 14230-2.

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

Table 3 — Positive response message

Type	Parameter Name	CVT[1)]	Hex Value	Mnemonic
Header Bytes[2)]	Format Byte Target Byte Source Byte Length Byte	M C[3)] C[3)] C[4)]	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>]	C[5]	xx=[xx : xx]	PN
CS[2)]	Checksum Byte	M	xx	CS

1) See 4.1.4
2) Defined in ISO 14230-2.
3) The header bytes "Target" and "Source" depend on the content of the "Format Byte" which is specified in ISO 14230-2 (KWP 2000 Part 2: Data Link Layer) document. Both either exist or do not exist in the header of each message.

4): The header byte "Length" depends on the content of the "Format Byte" which is specified in ISO DIS 14230-2.

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

Table 4 — Negative response message

Type	Parameter Name	CVT[1]	Hex Value	Mnemonic
Header Bytes[2]	Format Byte Target Byte Source Byte Length Byte	M C[3] C[3] C[4]	xx xx xx xx	FMT TGT SRC LEN
<ServiceId>	NegativeResponse Service Identifier	M	xx	NACK
<ServiceId>	<Service Name> Request Service Identifier	M	xx	SN
<Parameter Type>	ResponseCode=[KWP2000ResponseCode, ManufacturerSpecific]	M	xx=[00-7F, 80-FF]	RC
CS[2]	Checksum Byte	M	xx	CS

1) See 4.1.4
2) Defined in ISO 14230-2.
3) The header bytes "Target" and "Source" depend on the content of the "Format Byte" which is specified in ISO 14230-2 (KWP 2000 Part 2: Data Link Layer) document. Both either exist or do not exist in the header of each message.

4): The header byte "Length" depends on the content of the "Format Byte" which is specified in ISO DIS 14230-2..

4.2.3 Message description

This section"Message description" provides a description of the actions performed by the client and the server which are specific to the KWP 2000 Protocol (see ISO 14230-2).

The response condition is service specific and defined separately for each service.

4.2.4 Message flow examples

This section provides message flow descriptions presented in a table format (see table 5). The table consists of three columns:

— column 1: includes the relevant inter-message timing which is specified in the document ISO/DIS 14230-2. The message shall be started within the relevant inter-message timing;

— column 2: includes all requests send by the client to the server;

— column 3: includes all responses send by the server to the client.

Sending of a message shall start during the period of time between appropriate messages.

Time relates to the table in a top to bottom sequence. The reading entry of the message flow table always starts with the first item in the time column "P3" (1st column) followed by a request message of the client (2nd column) The next entry is the timing parameter "P2" (1st column) for the server to send the positive or negative response message (3rd column).

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.

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

Table 5 — Message flow example of physical addressed service

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

4.3 Functional unit table

The intention of specifying functional unit tables is to group similar Keyword Protocol (KWP) 2000 services into a functional unit. The definition of each functional unit includes a table such as table 6 which lists its services.

Table 6 — Keyword Protocol 2000 functional units

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).

4.4 Service Identifier value summary table

The left column of table 7 lists all services of the Diagnostic Services Specification, the middle column assigns the KWP 2000 Implementation "Request Hex Value" and the right column assigns the KWP 2000 Implementation "Positive Response Hex Value". The positive response service identifier values are built from the request service identifier values by setting "bit 6 = 1".

4.5 Response Code value summary table

Table 8 lists and assigns hex values for all response codes used in KWP 2000. The definition of each response code is described in ISO 14229.

Table 7 — Service Identifier value summary table

Diagnostic Service Name	KWP 2000 Implementation	
	Request Hex Value	Response Hex Value
StartDiagnosticSession	10	50
ECUReset	11	51
ReadFreezeFrameData	12	52
ReadDiagnosticTroubleCodes	13	53
ClearDiagnosticInformation	14	54
ReadStatusOfDiagnosticTroubleCodes	17	57
ReadDiagnosticTroubleCodesByStatus	18	58
ReadECUIdentification	1A	5A
StopDiagnosticSession	20	60
ReadDataByLocalIdentifier	21	61
ReadDataByCommonIdentifier	22	62
ReadMemoryByAddress	23	63
SetDataRates	26	66
SecurityAccess	27	67
DynamicallyDefineLocalIdentifier	2C	6C
WriteDataByCommonIdentifier	2E	6E
InputOutputControlByCommonIdentifier	2F	6F
InputOutputControlByLocalIdentifier	30	70
StartRoutineByLocalIdentifier	31	71
StopRoutineByLocalIdentifier	32	72
RequestRoutineResultsByLocalIdentifier	33	73
RequestDownload	34	74
RequestUpload	35	75
TransferData	36	76
RequestTransferExit	37	77
StartRoutineByAddress	38	78
StopRoutineByAddress	39	79
RequestRoutineResultsByAddress	3A	7A
WriteDataByLocalIdentifier	3B	7B
WriteMemoryByAddress	3D	7D
TesterPresent	3E	7E
EscCode[1)]	80	C0
1) Does not form part of diagnostic services specification but only of KWP Protocol 2000.		

Table 8 —Response Code value summary table

Hex Value	Response Code
10	GeneralReject
11	ServiceNotSupported
12	SubFunctionNotSupported-invalidFormat
21	Busy-RepeatRequest
22	ConditionsNotCorrect or requestSequenceError
23	RoutineNotComplete
31	RequestOutOfRange
33	SecurityAccessDenied
35	InvalidKey
36	ExceedNumberOfAttempts
37	RequiredTimeDelayNotExpired
40	DownloadNotAccepted
41	ImproperDownloadType
42	Can'tDownloadToSpecifiedAddress
43	Can'tDownloadNumberOfBytesRequested
50	UploadNotAccepted
51	ImproperUploadType
52	Can'tUploadFromSpecifiedAddress
53	Can'tUploadNumberOfBytesRequested
71	TransferSuspended
72	TransferAborted
74	IllegalAddressInBlockTransfer
75	IllegalByteCountInBlockTransfer
76	IllegalBlockTransferType
77	BlockTransferDataChecksumError
78	ReqCorrectlyRcvd-RspPending
79	IncorrectByteCountDuringBlockTransfer
80 - FF	ManufacturerSpecificCodes
1) RequestCorrectlyReceived-ResponsePending	

4.6 Response handling

Figure 3 specifies the server behaviour on a client request message. It shows the logic as specified in the description of the response codes and to be implemented in the server and client as appropriate.

The use of (a) negative response message(s) by the server shall be in case the server can not respond with a positive response message on a client (tester) request message. In such case the server shall send one of the response codes listed as specified in figure 3.

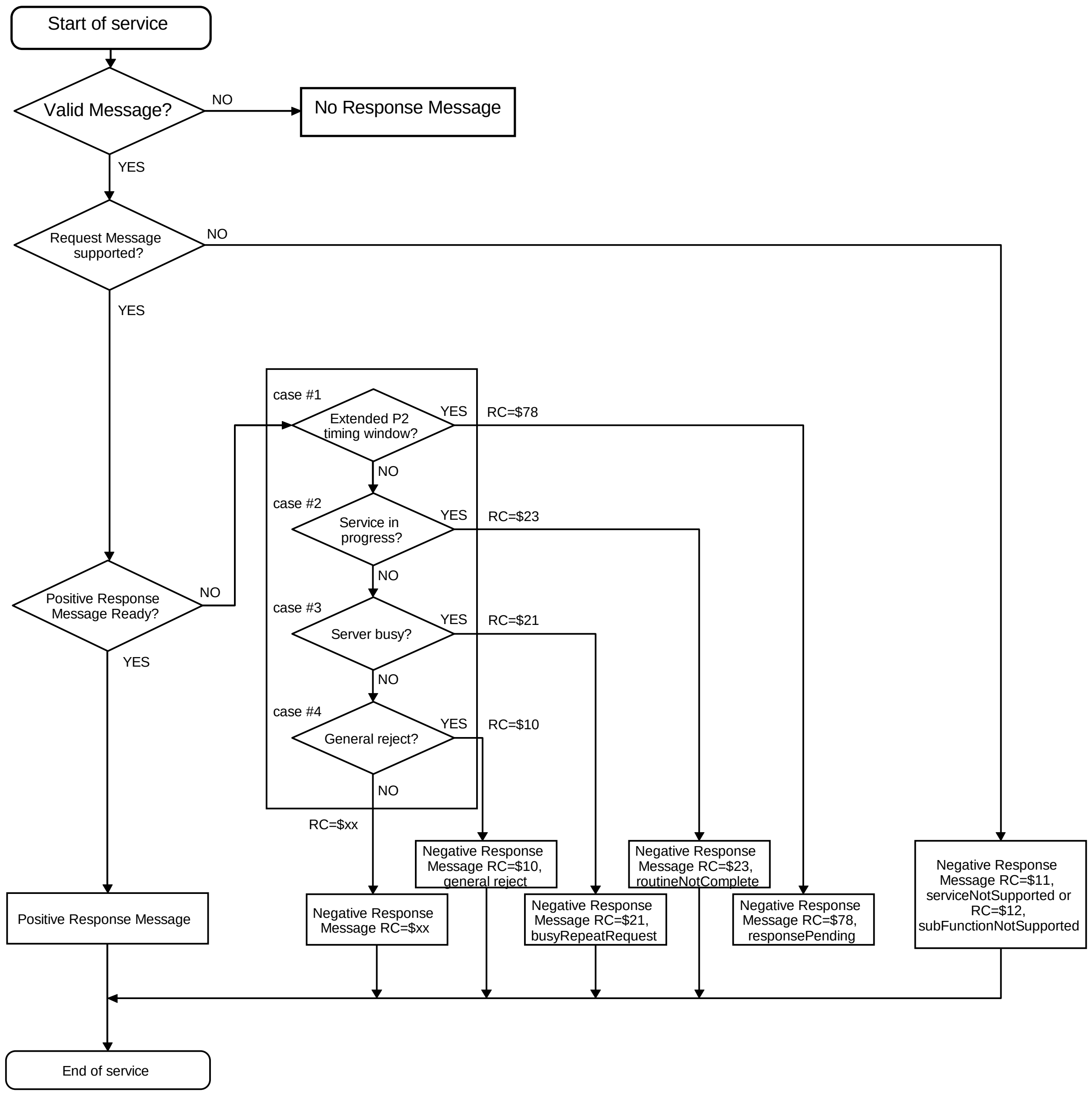

Start of service
Valid Message?
NO
No Response Message
YES
Request Message supported?
NO
YES
case #1
Extended P2 timing window?
YES
RC=$78
NO
case #2
Service in progress?
YES
RC=$23
NO
case #3
Server busy?
YES
RC=$21
NO
case #4
General reject?
YES
RC=$10
NO
RC=$xx
Positive Response Message Ready?
NO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

Figure 3 — Server positive and negative response message behaviour

5 General implementation rules

5.1 Parameter definitions

The following rules in regard to parameter definitions shall apply:

— clauses 6 to 12 define the services of each functional unit. In these clauses, the service structure makes reference to parameters, in order to describe the allowable values for such parameters. The parameters of general purpose are defined in ISO 14229. Parameters which are specific to a functional unit are described in the corresponding clause;

— this part of ISO 14230 lists and defines response codes and values. Negative response codes are specified in 4.4. Other response codes may be reserved either for future definition by this part of ISO 14230 or for the system designer's specific use;

— this part of ISO 14230 specifies the parameters which shall be used within each KWP 2000 service;

— the sequence of parameters within a service shall not be changed during an implementation;

— this part of ISO 14230 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 different addressing methods are specified in KWP 2000 to send a service request to a server(s).

Functional addressing with a three byte header is used by the client if it does not 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.

Functional addressing with a one byte header is not possible.

Physical addressing with a three 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.

Physical addressing with a one byte header is possible.

Only those server(s) which are initialized and in a diagnostic session which support the service request message shall send a response message.

Functional and Physical addressing methods are specified in detail in ISO 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

Table 9 shows a typical service request followed by a positive response message and a service request followed by a negative response message.

Table 9 — Message flow example of physical addressed service

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> Request[...]	
P2		<Service Name> PositiveResponse[...]

5.3.1.2 Physical addressed service with periodic transmissions

Table 10 shows a message flow which describes a physical addressed service request with multiple positive response messages.

Table 10 — Message Flow example of physical addressed service with periodic transmissions

time	client (tester)	server (ECU)
P3	<ReadDataByLocalIdentifier> Request[RLI,TXM]	
P2		<ReadDataByLocalIdentifier > PositiveResponse#1[RLI, ...]
P2		:
P2		<ReadDataByLocalIdentifier > PositiveResponse#k[RLI, ...]
P3*	<ReadDataByCommonIdentifier> Req.[RCI,TXM]	
P2		<ReadDataByCommonIdentifier> PosResp#1[RCI, ...]
P2		:
P2		<ReadDataByCommonIdentifier> PosResp#k[RCI, ...]
P3[1)]	<ReadMemoryByAddress> Request[MA,MS,TXM]	
P2		<ReadMemoryByAddress> PosResp#1[RECVAL]
P2		:
P2		<ReadMemoryByAddress> PosResp#1[RECVAL]
P3*	<Any Other Service Name> Request[...]	
P2		< Any Other Service Name > PositiveResponse[...]
P3	<ReadDataByLocalIdentifier> Request[RLI]	
P2		<ReadDataByLocalIdentifier> PositiveResponse[RLI,]
1) 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.		

The message flow in table 10 describes a physical addressed service request ReadDataByLocal/Common-Identifier or ReadMemoryByAddress 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 any request message not including the optional TransmissionMode parameter. This request message shall be send to the server within the "P3*" timing window.

5.3.1.3 Physical addressed service and negative response message(s) with "RoutineNotComplete" and "Busy-RepeatRequest"

Table 11 shows a message flow which describes a physical addressed service request followed by a negative response message with the response code set to "RoutineNotComplete".

Table 11 — Physical addressing and negative response with "RoutineNotComplete" and "Busy-RepeatRequest"

time	client (tester)	server (ECU)
P3	<Service Name> Request [...][1)]	<Service Name> NegativeResponse[RoutineNotComplete]
P2		
P3	<Service Name> Request[...]	<Service Name> NegativeResponse[Busy-RepeatRequest]
P2		:
	:	
P3	<Service Name> Request[...]	<Service Name> NegativeResponse[Busy-RepeatRequest]
P2		<Service Name> PositiveResponse[...]
P3	<Service Name> Request[...][2)]	OR
P2		Service Name> NegativeResponse[RC != Busy-RepeatRequest]
P2		
1) server has started routine. 2) server has completed routine.		

The message flow example in table 11 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". 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". 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".

The communication timing is not affected.

Applications which may require this message flow are as follows:

— ClearDiagnosticInformation;

— execution of routines;

— SecurityAccess.

5.3.1.4 Implementation example of "server can not send a positive response within required timing"

5.3.1.4.1 Example of physical addressed service and negative response message with "ReqCorrectlyRcvd-RspPending within Normal or Extended Timing"

Table 12 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".

Table 12 — Example of physical addressing and negative response with "ReqCorrectlyRcvd-RspPending"

time	client (ester)	server (ECU)
P3	<Service Name> Request[...]	
P2		<Service Name> NegRsp#1[ReqCorrectlyRcvd-RspPending]
P2[1)]		:
P2[1)]		<Service Name> NegRsp#n[ReqCorrectlyRcvd-RspPending]
P3	<Service Name> Request[...]	<Service Name> PositiveResponse[...]
P2		<Service Name> PositiveResponse[...]
1) P2min to P3max (refer to ISO 14230-2).		

The message flow example in table 12 is based on a request message from the client which causes the server to respond with one or multiple negative response message(s) 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 may 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 ISO 14230-2:—, clause 5.

Applications which may require above message flow are as follows:

— ClearDiagnosticInformation;

— execution of routines;

— TransferData request messages including up to 255 data bytes;

— Flash EEPROM or EEPROM memory programming.

5.3.1.4.2 Implementation of "Server can not send a positive response within Default Timing"

Table 13 shows a message flow which describes a physical addressed service request followed by a positive response message sent with previously modified timing.

Table 13 — Physical addressing and modified timing with AccessTimingParameter service

Time 1)	Client (tester)	Server (ECU)
P3	StartDiagnosticSession.Request[...]	
P2		StartDiagnosticSession.PosRsp[...]
P3	AccessTimingParameter.Request[ReadLimits]	
P2		AccessTimingParameter.PosRsp[ReadLimits, P2-P4]
P3	AccessTimingParameter.Request[setValues, P2-P4]	
P2	{ e.g. P2max = $F2 (18850 ms) }	
	{ Modified Timing is active! }	AccessTimingParameter.PosRsp[setValues] 2)
P3	<Service Name> Request[...]	
P2		<Service Name> PositiveResponse[...]
P3	<StopDiagnosticSession> Request[]	
P2	OR	<StopDiagnosticSession> PositiveResponse[]
P3	<StartDiagnosticSession> Request[...]	OR
P2		<StartDiagnosticSession> PositiveResponse[...] 3)
	{ Default Timing is active! }	
P3	<Service Name> Request[...]	
P2		<Service Name> PositiveResponse[...]
1) New timing parameter values are active.		
2) Modified Timing is active.		
3) Default Timing is active.		

The message flow example in table 13 describes the method of modifying the timing parameter values in the server and the client by using the AccessTimingParameter service as specified in ISO 14230-2.

This method shall be used in case a server does not support a negative response message with the response code ($78) "RequestCorrectlyReceived-ResponsePending".

5.3.2 Functional addressed services

5.3.2.1 Functional addressed service with positive/negative response message

Table 14 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-1, ECU#n).

Table 14 — Message flow example of functional addressed service

Time	client (tester)	server (ECU)
P3	<Service Name> Request[...]	
P2		<Service Name> PositiveResponse[...] {ECU#1}
P2		<Service Name> NegativeResponse[...] {ECU#2}
:		:
P2		<Service Name> PositiveResponse[...] {ECU#n-1}
P2		<Service Name> PositiveResponse[...] {ECU#n}
P3	<Service Name> Other Request[...]	
P2		<Service Name> Other PositiveRsponse[...] {ECU}

The message flow example in table 14 is based on a functional request message send by the client. An unknown number of servers has been initialized previously (e.g. fast initialization by wake up pattern) which send positive and negative response messages.

5.3.2.2 Functional addressed service and negative response messages with "RoutineNotComplete" and "Busy-RepeatRequest"

Table 15 shows a message flow which describes a functional addressed service request followed by a negative response message with the response code set to "RoutineNotComplete" from one server. All other servers send positive response messages.

Table 15 — Functional addressing and negative response with "Busy-RepeatRequest"

Time	client (tester)	server (ECU)
P3	<Service Name> Request[...]1), 2)	
P2		<Service Name> NegRsp[RoutineNotComplete] {ECU#1}
P2		<Service Name> PositiveResponse[...] {ECU#2}
P2		<Service Name> PositiveResponse[...] {ECU#3}
P3	<Service Name> Request[...]1) { Functional }	
P2		<Service Name> NegRsp[Busy-RepeatRequest] {ECU#1}
P2		<Service Name> PositiveResponse[...] {ECU#2}
P2		<Service Name> PositiveResponse[...] {ECU#3}
P3	<Service Name> Request[...]1), 3) { Functional}	
P2		<Service Name> PositiveResponse[...] {ECU#1}
P2		<Service Name> PositiveResponse[...] {ECU#2}
P2		<Service Name> PositiveResponse[...] {ECU#3}
1) Functional. 2) Server has started routine. 3) Server has completed routine.		

The message flow example in table 15 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" 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". The negative response message shall be sent on each functional 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".

The communication timing is not affected.

Applications which may require above message flow are as follows:

— ClearDiagnosticInformation;

— execution of routines;

— SecurityAccess.

5.3.2.3 Implementation example of functional addressed service and negative response message with "RequestCorrectlyReceived-ResponsePending"

Table 16 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).

Table 16 — Example of functional addressing and negative response with "ReqCorrectlyRcvd-RspPending"

Time	Client (tester)	Server (ECU)
P3	<Service Name> Req[...][1]	
P2		<Service Name> NegRsp#1[ReqCorrectlyRcvd-RspPendg] {ECU#1}
P2[2]		<Service Name> PositiveResponse[...] {ECU#2}
P2[2]		<Service Name> PositiveResponse[...] {ECU#3}
P2[2]		<Service Name> NegRsp#2[ReqCorrectlyRcvd-RspPendg] {ECU#1}
P2[2]		:
P2[2]	<Service Name> Req[...] { Functional }	<Service Name> NegRsp#n[ReqCorrectlyRcvd-RspPendg] {ECU#1}
P2[2]		<Service Name> PositiveResponse[...] {ECU#1}
P3		
P2		<Service Name> PositiveResponse[...] {ECU#1}
P2		<Service Name> PositiveResponse[...] {ECU#2}
P2		<Service Name> PositiveResponse[...] {ECU#3}

1) Functional.

2) P2min to P3max (refer to ISO 14230 KWP-2).

The message flow example in table 16 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.

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.

The communication timing method is specified in ISO 14230-2:—, clause 5.

Applications which may require above message flow are as follows:

— ClearDiagnosticInformation;

— execution of routines;

— TransferData request messages including up to 255 data bytes;

— Flash EEPROM or EEPROM memory programming.

6 Diagnostic Management functional unit

The services provided by this functional unit are described in table 17.

Table 17 — Diagnostic Management functional unit

Service name	Description
StartDiagnosticSession	The client requests to start a diagnostic session with a server(s).
StopDiagnosticSession	The client requests to stop the current diagnostic session.
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).

6.1 StartDiagnosticSession service

6.1.1 Parameter definition

The parameter **DiagnosticMode** is used by the StartDiagnosticSession service to select the specific behaviour of the server(s). No values are defined in this part of ISO 14230, but table 18 describes the range of values.

Table 18 — Definition of DiagnosticMode Values

Hex	Description
00 - 7F	**ReservedByDocument** This range of values is reserved by this standard for future definition.
80 - FF	**ManufacturerSpecific** This range of values is reserved for vehicle-manufacturer-specific use.

6.1.2 Message data bytes

Tables 19 to 21 describe the three possible messages for StartDiagnosticSession service.

Table 19 — StartDiagnosticSession Request Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**StartDiagnosticSession Request Service Id**	**M**	**10**	**SDS**
2	DiagnosticMode=[ReservedByDocument, ManufacturerSpecific]	M	xx=[00-7F, 80-FF]	**DIAGMODE**

Table 20 — StartDiagnosticSession Positive Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**StartDiagnosticSession Positive Response Service Id**	**S**	**50**	**SDSPR**
2	DiagnosticMode=[ReservedByDocument, ManufacturerSpecific]	U	xx=[00-7F, 80-FF]	**DIAGMODE**

Table 21 — StartDiagnosticSession Negative Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**NegativeResponse Service Id**	**S**	**7F**	**NACK**
2	**StartDiagnosticSession Request Service Id**	**S**	**10**	**SDS**
3	ResponseCode=[KWP2000ResponseCode, ManufacturerSpecific]	M	xx=[00-7F, 80-FF]	**RC**

6.1.3 Message description

The messages specified in 6.1.2 are used to enable different diagnostic modes in the server. Possible diagnostic modes are not defined in this part of ISO 14230.

A diagnostic session shall only be started if communication has been established between the client and the server. For more detail on how to start communication, refer to ISO 14230-2.

If no diagnostic session has been requested by the client after StartCommunication a default session shall be automatically enabled in the server. The default session shall support at least the following services:

— "StopCommunication service" (see ISO 14230-2);

— "TesterPresent service" (see ISO 14230-3).

The default timing parameters in the server shall be active while the default session is running.

If a diagnostic session has been requested by the client which is already running 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 may enable vehicle-manufacturer-specific services which are not part of this part of ISO 14230.

6.1.4 Message flow example

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

6.2 StopDiagnosticSession service

6.2.1 Parameter Definition

This service shall not use any parameter definition.

6.2.2 Message Data Bytes

Tables 22 to 24 describe the three possible messages for StopDiagnosticSession service.

Table 22 — StopDiagnosticSession Request Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**StopDiagnosticSession Request Service Id**	M	20	SPDS

Table 23 — StopDiagnosticSession Positive Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**StopDiagnosticSession Positive Response Service Id**	S	60	SPDSPR

Table 24 — StopDiagnosticSession Negative Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**NegativeResponse Service Id**	S	7F	NACK
2	**StopDiagnosticSession Request Service Id**	S	20	SPDS
3	ResponseCode=[KWP2000ResponseCode, ManufacturerSpecific]	M	xx=[00-7F, 80-FF]	RC

6.2.3 Message description

The messages specified in 6.2.2 are used to disable the current diagnostic mode in the server.

The following implementation rules shall be followed:

— a diagnostic session shall only be stopped if communication has been established between the client and the server and a diagnostic session is running;

— if no diagnostic session is running the default session is active (see StartDiagnosticSession). The default session cannot be disabled by a StopDiagnosticSession service;

— if the server has sent a StopDiagnosticSession positive response message the default timing parameters in the server shall be valid while the default session is running;

— if the server has sent a StopDiagnosticSession 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 restore its normal operating conditions. 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 StopDiagnosticSession positive response message it shall have re-locked the server if it was unlocked by the client during the diagnostic session;

— if a StopDiagnosticSession has been requested by the client and the default session is already running the server shall send a positive response message and immediately reset all timing parameters;

— the client shall send a StopDiagnosticSession request message before disabling communication via a StopCommunication service but only if a StartDiagnosticSession request message has been sent previously;

— if the server sends a negative response message with the StopDiagnosticSession request service identifier the active session shall be continued;

— a StopDiagnosticSession service shall also be used to disable vehicle-manufacturer-specific diagnostic sessions.

6.2.4 Message flow example

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

6.3 SecurityAccess service

6.3.1 Parameter definition

The parameter *AccessMode* 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 table 25 for RequestSeed and SendKey.

Table 25 — Definition of AccessMode values

Hex	Description
01	**RequestSeed** RequestSeed with the level of security defined by the vehicle manufacturer.
02	**SendKey** SendKey with the level of security defined by the vehicle manufacturer.
03, 05, 07 - 7F	**RequestSeed** RequestSeed with different levels of security defined by the vehicle manufacturer.
04, 06, 08 - 80	**SendKey** SendKey with different levels of security defined by the vehicle manufacturer.
81 - FF	**ManufacturerSpecific** This range of values is reserved for vehicle-manufacturer-specific use.

The values of the parameter ***Seed*** are not defined in this standard except for the value "$00 00" which indicates to the client that the server is not locked.

The values of the parameter ***Key*** are not defined in this part of ISO 14230.

The parameter ***SecurityAccessStatus*** may be used to receive the status of the server security system. The value is defined in table 26.

Table 26 — Definition of SecurityAccessStatus value

Hex	Description
34	SecurityAccessAllowed

6.3.2 Message data bytes

Tables 27 to 32 describe the different messages for SecurityAccess service.

Table 27 — SecurityAccess Request#1 Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**SecurityAccess Request#1 Service Id**	**M**	**27**	**SA#1**
2	AccessMode=[RequestSeed: ManufacturerSpecific	M	xx=[1) 01,03, :7F,81-FF]	**ACCMODE**
1) Shall be an odd number greater than $01 if additional levels of security are supported,($01 default),				

Table 28 — SecurityAccess Positive Response#1 Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**SecurityAccess Positive Response#1 Service Id**	**S**	**67**	**SA#1PR**
2	AccessMode=[RequestSeed: ManufacturerSpecific]	M	xx=[2) 01,03, :7F, 81-FF]	**ACCMODE**
3 : n - 1	seed#1 : seed#m	C 1) : C 1)	xx : xx	**SEED**
n	SecurityAccessStatus=[SecurityAccessAllowed]	U	34	**SACCSTAT**
1) The condition is AccessMode set to RequestSeed. 2) Shall be an odd number greater than $01 if additional levels of security are supported,($01 default).				

Table 29 — SecurityAccess Negative Response#1 Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**NegativeResponse#1 Service Id**	**S**	**7F**	**NACK**
2	**SecurityAccess Request Service Id**	**S**	**27**	**SA**
3	ResponseCode=[KWP2000ResponseCode, ManufacturerSpecific]	M	xx=[2) 00-7F, 80-FF]	**RC**

Table 30 — SecurityAccess Request#2 Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**SecurityAccess Request#2 Service Id**	**M**	**27**	**SA#2**
2	AccessMode=[SendKey: shall be an even number one greater than AccessMode in request#1 if additional levels of security are supported, ($02 default) ManufacturerSpecific]	M	xx=[02, 04, : 80, 82-FE]	**ACCMODE**
3 : n	key#1 : key#m	C 1) : C 1)	xx : xx	**KEY**
1) The condition is AccessMode set to SendKey. 2) Shall be an even number greater than $01 if additional levels of security are supported,($02 default).				

Table 31 — SecurityAccess Positive Response#2 Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**SecurityAccess Positive Response#2 Service Id**	**S**	**67**	**SA#2PR**
2	[SendKey: shall be an even number one greater than AccessMode in request#1 if additional levels of security are supported, ($02 default) ManufacturerSpecific]	M	xx=[[1)] 02, 04, : 80, 82-FE]	**ACCMODE**
3	SecurityAccessStatus=[SecurityAccessAllowed]	U	34	**SACCSTAT**
1) Shall be an even number greater than $01 if additional levels of security are supported,($02 default).				

Table 32 — SecurityAccess Negative Response#2 Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**NegativeResponse#2 Service Id**	**S**	**7F**	**NACK**
2	**SecurityAccess Request Service Id**	**S**	**27**	**SA**
3	ResponseCode=[KWP2000ResponseCode, ManufacturerSpecific]	M	xx=[00-7F, 80-FF]	**RC**

6.3.3 Message description

This mode is intended to be used to implement the data link security measures defined in SAE J 2186.

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 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 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 10 s time delay before allowing further attempts.

The 10 s time delay shall also be required before the server responds to a service SecurityAccess request#1 from the client after server power-on.

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.

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.

Servers that provide security shall support reject messages if a secure mode is requested while the server is locked.

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 second data byte of the "RequestSeed" shall always be an odd number, and the second data byte of the service to "SendKey" shall be the next even number.

6.3.4 Message flow example

Table 33 gives a message flow example of SecurityAccess services.

Table 33 — Message flow example of SecurityAccess services

time	client (tester)	server (ECU)
P3	SecurityAccess.Request#1[...]	
P2		SecurityAccess.PositiveResponse#1[...]
P3	SecurityAccess.Request#2[...]	
P2		SecurityAccess.PositiveResponse#2[...]

6.4 TesterPresent service

6.4.1 Parameter definition

The parameter ***ResponseRequired*** is used in the TesterPresent request message. It indicates to the server whether a response message is required or not. Values for this parameter are defined in table 34.

Table 34 — Definition of ResponseRequired values

Hex	Description
01	**yes** The server shall send a response on the request message.
02	**no** The server shall not send a response on the request message.
03 - 7F	**ReservedByDocument** This range of values is reserved by this document for future definition.
80 - FF	**vehicle ManufacturerSpecific** This range of values is reserved for vehicle-manufacturer-specific use.

6.4.2 Message data bytes

Tables 35 to 37 describe the different messages for TesterPresentservice.

Table 35 — TesterPresent Request Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**TesterPresent Request Service Id**	**M**	**3E**	**TP**
2	ResponseRequired=[Yes, No]	U	xx=[01, 02]	**RSPREQ**

Table 36 — TesterPresent Positive Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**TesterPresent Positive Response Service Id**	**S**	**7E**	**TPPR**

Table 37 — TesterPresent Negative Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**NegativeResponse Service Id**	**S**	**7F**	**NACK**
2	**TesterPresent Request Service Id**	**M**	**3E**	**TP**
3	ResponseCode=[KWP 2000ResponseCode, ManufacturerSpecific]	M	xx=[00-7F, 80-FF]	**RC**

6.4.3 Message description

This service shall be used to indicate to a server 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 ensure communication active is kept active between client and server;

— the presence of this service shall indicate that the system should remain in a diagnostic mode of operation;

— if the user optional parameter ResponseRequired is not included in the TesterPresent request message, the server shall send a TesterPresent positive response.

6.4.4 Message flow example

Table 38 gives a message flow example of TesterPresent services.

Table 38 — Message flow example of TesterPresent services

time	client (Tester)	server (ECU)
P3	TesterPresent.Request[Yes]	
P2		TesterPresent.PositiveResponse[]
P3	TesterPresent.Request[No]	
P2		{ no response from server }
P3	TesterPresent.Request[No]	
P2		{ no response from server }
P3	TesterPresent.Request[]	
P2		TesterPresent.PositiveResponse[]

6.5 ECUReset service

6.5.1 Parameter definition

The parameter ***ResetMode*** is used by the ECUReset request message to describe how the server has to perform the reset. Values are defined in table 39.

Table 39 — Definition of ResetMode values

Hex	Description
01	**PowerOn** This value identifies the PowerOn ResetMode which shall be a simulated PowerOn reset which most ECUs perform after ignition OFF/ON cycle. When the ECU performs the reset the client (tester) shall be prepared to re-establish communication.
02	**PowerOnWhileMaintainingCommunication** This value identifies the PowerOn ResetMode which shall be a simulated PowerOn reset which most ECUs perform after ignition OFF/ON cycle. When the ECU performs the reset the server (ECU) shall maintain the communication with the client (tester).
03 - 7F	**ReservedByDocument** This range of values is reserved by this document for future definition.
80 - FF	**ManufacturerSpecific** This range of values is reserved for vehicle-manufacturer-specific use.

The parameter ***ResetStatus*** 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; see table 40.

Table 40 — Definition of ResetStatus value

Hex	Description
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).

6.5.2 Message data bytes

Tables 41 to 43 describe the different messages for ECUReset service.

Table 41 — ECUReset Request Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**ECUReset Request Service Id**	**M**	**11**	**ECURST**
2	ResetMode=[PowerOn, PowerOnWhileMaintainingCommunication, ReservedByDocument, ManufacturerSpecific]	U	xx=[01, 02, 03 - 7F, 80 - FF]	**RSTOPT**

Table 42 — ECUReset Positive Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**ECUReset Positive Response Service Id**	**S**	**51**	**ECURSTPR**
2 : n	ResetStatus#1 : ResetStatus#m	U : U	xx : xx	**RSTSTAT**

Table 43 — ECUReset Negative Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**NegativeResponse Service Id**	**S**	**7F**	**NACK**
2	**ECUReset Request Service Id**	**M**	**11**	**ECURST**
3	ResponseCode=[KWP2000ResponseCode, ManufacturerSpecific]	M	xx=[00-7F, 80-FF]	**RC**

6.5.3 Message description

This service requests the server to perform an ECU reset effectively based on the content of the ResetMode parameter value. It is the vehicle manufacturer's responsibility to define whether the positive response message shall be sent before or after the ResetMode is executed.

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).

6.5.4 Message flow example

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

6.6 ReadECUIdentification service

6.6.1 Parameter definition

The parameter ***IdentificationOption*** is used by the ReadECUIdentification request message to describe the kind of identification data requested by the client. Values are defined in table 44.

Table 44 — Definition of IdentificationOption values

Hex	Description
00 - 7F	**ReservedByDocument** This range of values is reserved by this par ofISO 14230 for future definition.
80 - FF	**ManufacturerSpecific** This range of values is reserved for vehicle-manufacturer-specific use.

The parameter ***IdentificationRecordValue*** is used by the ReadECUIdentification positive response message to provide the identification data to the client. The content of the identification record is not defined in this part of ISO 14230 and is vehicle-manufacturer-specific.

6.6.2 Message data bytes

Tables 45 to 47 describe the different messages for ReadECUIdentificationService.

Table 45 — ReadECUIdentification Request Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**ReadECUIdentification Request Service Id**	**M**	**1A**	**RECUID**
2	IdentificationOption=[ReservedByDocument, ManufacturerSpecific]	U	xx=[00 - 7F, 80 - FF]	**IDOPT**

Table 46 — ReadECUIdentification Positive Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**ReadEcuIdentification Positive Response Service Id**	**S**	**5A**	**RECUIDPR**
2 n	IdentificationRecordValue=[ECUIdentificationParameter#1 : ECUIdentificationParameter#m]	M	xx=[xx : xx]	**IDRECVAL**

Table 47 — ReadECUIdentification Negative Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**NegativeResponse Service Id**	**S**	**7F**	**NACK**
2	**ReadECUdentification Request Service Id**	**M**	**1A**	**RECUID**
3	ResponseCode=[KWP2000ResponseCode, ManufacturerSpecific]	M	xx=[00-7F, 80-FF]	**RC**

6.6.3 Message description

The ReadECUIdentification request message requests identification data from the server. The type of identification data requested by the client shall be identified by the IdentificationOption parameter. The server sends an identification data record included in the ReadECUIdentification positive response message. The format and definition of the identification data record shall be vehicle-manufacturer-specific and is not part of this part of ISO 114230.

6.6.4 Message flow example

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

7 Data Transmission functional unit

The services provided by this functional unit are described in table 48.

Table 48 — Data Transmission functional unit

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 RecordLocal-Identifier.
WriteDataByCommonIdentifier	The client requests to write a record accessed by RecordCommon-Identifier.
WriteMemoryByAddress	The client requests to overwrite a memory area.
SetDataRates	The client changes the data rates for periodic transmissions.

7.1 ReadDataByLocalIdentifier service

7.1.1 Parameter definition

The parameter ***RecordLocalIdentifier*** 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 stored in RAM if defined dynamically by the service DynamicallyDefineLocalIdentifier.

The parameter ***TransmissionMode*** in the ReadDataByLocalIdentifier request message indicates how the server shall transmit the data record. Values are defined in table 49.

Table 49 — Definition of TransmissionMode values

Hex	Description
01	**single** The server transmits the positive response message only once independent of the value specified by the parameter MaximumNumberOfResponsesToSend.
02	**slow** The server transmits the positive response message periodically/repeatedly at the SlowRate. The value of the SlowRate is always predefined in the server and can be set by the client with the SetDataRates service. This parameter specifies the value of the period at which the server transmits the record value upon the next ReadDataByLocalIdentifier/GlobalIdentifier and ReadMemoryByAddress request message with the TransmissionMode parameter set to ***slow***. The repetition rate specified by the TransmissionMode parameter ***slow*** is vehicle-manufacturer-specific.
03	**medium** The server transmits the positive response message periodically/repeatedly at the MediumRate. The value of the MediumRate is always predefined in the server and can be set by the client with the SetDataRates service. This parameter specifies the value of the period at which the server transmits the record value upon the next ReadDataByLocalIdentifier/ CommonIdentifier and ReadMemoryByAddress request message with the TransmissionMode parameter set to ***Medium***. The repetition rate specified by the TransmissionMode parameter ***Medium*** is vehicle-manufacturer-specific.
04	**fast** The server transmits the positive response message periodically/repeatedly at the FastRate. The value of the FastRate is always predefined in the server and can be set by the client with the SetDataRates service. This parameter specifies the value of the period at which the server transmits the record value upon the next ReadDataByLocalIdentifier/GlobalIdentifier and ReadMemoryByAddress request message with the TransmissionMode parameter set to ***Fast***. The repetition rate specified by the TransmissionMode parameter ***Fast*** is vehicle-manufacturer-specific and means as fast as possible.
05	**stop** The server stops transmitting positive response messages send periodically/repeatedly.

The parameter ***MaximumNumberOfResponsesToSend*** in the ReadDataByLocalIdentifier request message shall be used to indicate the number of positive response messages to be sent by the server upon the request message. Values are defined in table 50.

Table 50 — Definition of MaximumNumberOfResponsesToSend values

Hex	Description
00	**InValid** This value shall not be used by this parameter.
01 - FF	**NumberOfResponsesToSend** This range of values shall be used to specify the number of responses the server shall send.

The parameter ***RecordValue*** is used by the ReadDataByLocalIdentifier positive response message to provide the data record identified by the RecordLocalIdentifier to the client. The content of the data record is not defined in this part of ISO 14230 and is vehicle-manufacturer-specific.

7.1.2 Message data bytes

Tables 51 to 53 describe the different messages for ReadDataByLocalIdentifier service.

Table 51 — ReadDataByLocalIdentifier Request Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**ReadDataByLocalIdentifier Request Service Id**	**M**	**21**	**RDBLID**
2	RecordLocalIdentifier	M	xx	**RLOCID**
3	TransmissionMode=[single, slow, medium, fast, stop]	U	xx=[01, 02, 03, 04, 05]	**TXM**
4	MaximumNumberOfResponsesToSend	U1)	xx	**M#ORTS**
1) condition: TransmissionMode parameter must be present				

Table 52 — ReadDataByLocalIdentifier Positive Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**ReadDataByLocalIdentifier Positive Response Service Id**	**S**	**61**	**RDBLIDPR**
2	RecordLocalIdentifier	M	xx	**RLOCID**
3 : n	RecordValue#1 : RecordValue#m	M : M	xx : xx	**RECVAL**

Table 53 — ReadDataByLocalIdentifier Negative Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**NegativeResponse Service Id**	**S**	**7F**	**NACK**
2	**ReadDataByLocalIdentifier Request Service Id**	**M**	**21**	**RDBLID**
3	ResponseCode=[KWP2000ResponseCode, ManufacturerSpecific]	M	xx=[00-7F, 80-FF]	**RC**

7.1.3 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 shall include analogue input and output signals, digital input and output signals, internal data and system status information if supported by the ECU.

If the server sends messages periodically and the client wants to stop the repeated positive response messages by a ReadDataByLocalIdentifier request message it shall send the request message after the P1 timing has expired and before the P2min timing becomes active. Refer to the message flow diagram and ISO 14230-2 more detail.

In addition, the user optional/conditional MaximumNumberOfResponsesToSend parameter indicates to the server how many repetitions of positive response messages are requested by the client. The timing is not affected by this parameter.

7.1.4 Message flow example

See message flow example of Physical Addressed Service in 5.3.1.1 and Physical Addressed Service with TransmissionMode parameter in 5.3.1.2.

The message flow in table 54 is an example of Physical Addressed Service with TransmissionMode (TXM) and MaximumNumberOfResponsesToSend (MAX#OFRSPTO-SEND) parameter present.

Table 54 — Message flow example of ReadDataByLocalIdentifier services with TransmissionMode and MaximumNumberOfResponsesToSend present

time	Client (tester)	Server (ECU)
P3	ReadDataByLocalId.Request[	
	TXM=slow...fast, MAX#OFRSPTOSEND=03]	
P2[1)]		ReadDataByLocalId.PositiveResponse#1[...]
P2[1)]		ReadDataByLocalId.PositiveResponse#2[...]
P2[1)]		ReadDataByLocalId.PositiveResponse#3[...]
	{ next request service }	
P3		
P2		{ next response service }
1) The P2 timing parameter may have been set previously by the parameters of the SetDataRates request message.		

7.2 ReadDataByCommonIdentifier service

7.2.1 Parameter definition

The parameter ***RecordCommonIdentifier*** in the ReadDataByCommonIdentifier service identifies a data record which is supported by multiple servers.

The parameter ***TransmissionMode*** is defined in 7.1.1.

The parameter ***MaximumNumberOfResponsesToSend*** is defined in 7.1.1.

The parameter ***RecordValue*** is defined in 7.1.1.

7.2.2 Message data bytes

Tables 55 to 57 describe the different messages for ReadDataByCommonIdentifier Service.

Table 55 — ReadDataByCommonIdentifier Request Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**ReadDataByCommonIdentifier Request Service Id**	**M**	**22**	**RDBCID**
2	RecordCommonIdentifier (High Byte)	M	xx	**RCIDHB**
3	RecordCommonIdentifier (Low Byte)	M	xx	**RCIDLB**
4	TransmissionMode=[single, slow, medium, fast, stop]	U	xx=[01, 02, 03, 04, 05]	**TXM**
5	MaximumNumberOfResponsesToSend	U1)	xx	**M#ORTS**
1) Condition: TransmissionMode parameter shall be present.				

Table 56 — ReadDataByCommonIdentifier Positive Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**ReadDataByCommonIdentifier Positive Response Service Id**	**S**	**62**	**RDBCIDPR**
2	RecordCommonIdentifier (High Byte)	M	xx	**RCIDHB**
3	RecordCommonIdentifier (Low Byte)	M	xx	**RCIDLB**
4 : n	RecordValue#1 : RecordValue#m	M : M	xx : xx	**RECVAL**

Table 57 — ReadDataByCommonIdentifier Negative Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**NegativeResponse Service Id**	**S**	**7F**	**NACK**
2	**ReadDataByCommonIdentifier Request Service Id**	**M**	**22**	**RDBCID**
3	ResponseCode=[KWP2000ResponseCode, ManufacturerSpecific]	M	xx=[00-7F, 80-FF]	**RC**

7.2.3 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 shall include analogue input and output signals, digital input and output signals, internal data and system status information if supported by the ECU(s).

If the server sends messages periodically and the client wants to stop the repeated positive response messages by a ReadDataByCommonIdentifier request message it shall send the request after the P1 timing has expired and before the P2min timing becomes active. Refer to the message flow diagram and ISO 14230-2 for more detail.

In addition, the user optional/conditional MaximumNumberOfResponsesToSend parameter indicates to the server how many repetitions of positive response messages are requested by the client. The timing is not affected by this parameter.

7.2.4 Message flow example

See message flow example of Physical Addressed Service in 5.3.1.1 or 5.3.1.2 and Functional Addressed Service in 5.3.2.1.

7.3 ReadMemoryByAddress service

7.3.1 Parameter definition

The parameter ***MemoryAddress*** in the ReadMemoryByAddress request message identifies the start address in the server's memory. If the server supports "16 bit" wide address range the high byte of the MemoryAddress shall be used as a MemoryIdentifier if required.

The parameter ***MemorySize*** specifies the number of bytes to be read starting at a specified memory address in the server's memory.

The parameter ***TransmissionMode*** is defined in 7.1.1.

The parameter ***MaximumNumberOfResposesToSend*** is defined in 7.1.1.

The parameter ***RecordValue*** is defined in 7.1.1.

7.3.2 Message data bytes

Tables 58 to 60 describe the different messages for ReadMemoryByAddress service.

Table 58 — ReadMemoryByAddress Request Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**ReadMemoryByAddress Request**	**M**	**23**	**RMBA**
2	MemoryAddress (High Byte)	M	xx	**MEMAHB**
3	MemoryAddress (Middle Byte)	M	xx	**MEMAMB**
4	MemoryAddress (Low Byte)	M	xx	**MEMALB**
5	MemorySize	M	xx	**MEMSIZE**
6	TransmissionMode=[single, slow, medium, fast, stop]	U	xx=[01, 02, 03, 04, 05]	**TXM**
7	MaximumNumberOfResponsesToSend	U[1)]	xx	**M#ORTS**
1) Condition: TransmissionMode parameter shall be present.				

Table 59 — ReadMemoryByAddress Positive Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**ReadMemoryByAddress Positive Response**	**S**	**63**	**RMBAPR**
2 : n	RecordValue#1 : RecordValue#m	M : M	xx : xx	**RECVAL**
n+1	MemoryAddress (High Byte)	U	xx	**MEMAHB**
n+2	MemoryAddress (Middle Byte)	U	xx	**MEMAMB**
n+3	MemoryAddress (Low Byte)	U	xx	**MEMALB**

Table 60 — ReadMemoryByAddress Negative Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**NegativeResponse Service Id**	**S**	**7F**	**NACK**
2	**ReadMemoryByAddressRequest Service Id**	**M**	**23**	**RMBA**
3	ResponseCode=[KWP2000ResponseCode, ManufacturerSpecific]	M	xx=[00-7F, 80-FF]	**RC**

7.3.3 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 shall include analogue input and output signals, digital input and output signals, internal data and system status information if supported by the ECU.

If the server sends positive response messages periodically and the client wants to stop the repeated messages by a ReadMemoryByAddress request message, it shall send the request after the P1 timing has expired and before the P2min timing becomes active. Refer to the message flow diagram and ISO 14230-2 for more detail.

In addition, the user optional/conditional MaximumNumberOfResponsesToSend parameter indicates to the server how many repetitions of positive response messages are requested by the client. The timing is not affected by this parameter.

7.3.4 Message flow example

See message flow examples of Physical Addressed Service in 5.3.1.1 or 5.3.1.2.

7.4 DynamicallyDefineLocalIdentifier service

7.4.1 Parameter definition

The parameter ***DynamicallyDefinedLocalIdentifier*** in the DynamicallyDefineLocalIdentifier request message identifies a data record which has been defined by the client in the request service.

The parameter ***DefinitionMode*** in the DynamicallyDefineLocalIdentifier request message specifies the method of how the data record is defined. Values are defined in table 61.

Table 61 — Definition of DefinitionMode values

Hex	Description
01	**DefineByLocalIdentifier**
02	**DefineByCommonIdentifier**
03	**DefineByMemoryAddress**
04	**ClearDynamicallyDefinedLocalIdentifier**
05 - 7F	**ReservedByDocument** This range of values is reserved by this document for future definition.
80 - FF	**ManufacturerSpecific** This range of values is reserved for vehicle-manufacturer-specific use.

The parameter ***PositionInDynamicallyDefinedLocalIdentifier*** in the Dynamically-DefineLocalIdentifier request message identifies the position of the data in the ReadDataByLocalIdentifier positive response message.

The parameter ***RecordLocalIdentifier*** is defined in 7.1.1.

The parameter ***RecordCommonIdentifier*** is defined in 7.1.2.

The parameter ***MemorySize*** in the DynamicallyDefineLocalIdentifier request message identifies the number of bytes of data identified by the RecordLocal/CommonIdentifier.

The parameter ***MemoryAddress*** in the DynamicallyDefineLocalIdentifier request message identifies the start address of a data record in the server's memory. If the server supports "16 bit" wide address range the high byte of the MemoryAddress shall be used as a MemoryIdentifier if required.

The parameter ***PositionInRecordLocalIdentifier*** in the DynamicallyDefineLocalIdentifier request message identifies the position in the data record identified by the RecordLocalIdentifier in the server's memory.

The parameter ***PositionInRecordCommonIdentifier*** in the DynamicallyDefineLocalIdentifier request message identifies the position in the data record identified by the RecordCommonIdentifier in the server's memory.

7.4.2 Message data bytes

Tables 62 to 65 describe the different RequestMessage definition mode for DynamicallyDefinedLocalIdentifier service.

Table 62 — DynamicallyDefineLocalIdentifier Request Message DefinitionMode=DefineByLocalIdentifier

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**DynamicallyDefineLocalIdentifier Request Service Id**	**M**	**2C**	**DLIDDY**
2	DynamicallyDefinedLocalIdentifier	M	xx	**DDLOCID**
3	DefinitionMode=[DefineByLocalIdentifier]#1	M	01	**DEFMODE**
4	PositionInDynamicallyDefinedLocalIdentifier#1	M	xx	**PIDYDLID**
5	MemorySize#1	M	xx	**MEMSIZE**
6	RecordLocalIdentifier#1	M	xx	**RLOCID**
7	PositionInRecordLocalIdentifier#1	M	xx	**PIRLOCID**
8	DefinitionMode=[DefineByLocalIdentifier]#2	M	01	**DEFMODE**
9	PositionInDynamicallyDefinedLocalIdentifier#2	M	xx	**PIDYDLID**
10	MemorySize#2	M	xx	**MEMSIZE**
11	RecordLocalIdentifier#2	M	xx	**RLOCID**
12	PositionInRecordLocalIdentifier#2	M	xx	**PIRLOCID**
:	**:**	**:**	**:**	**:**
n-4	DefinitionMode=[DefineByLocalIdentifier]#m	M	01	**DEFMODE**
#n-3	PositionInDynamicallyDefinedLocalIdentifier#m	M	xx	**PIDYDLID**
#n-2	MemorySize#m	M	xx	**MEMSIZE**
#n-1	RecordLocalIdentifier#m	M	xx	**RLOCID**
#n	PositionInRecordLocalIdentifier#m	M	xx	**PIRLOCID**

This message is used by the client to define dynamically a local identifier in the request message. If the DefinitionMode parameter is set to DefineByLocalIdentifier the following procedure shall be followed:

— the parameter PositionInDynamicallyDefinedLocalIdentifier shall specify the position in the record referenced by the parameter DynamicallyDefinedLocalIdentifier. The record specified may be a "1 byte" parameter (e.g. Engine Coolant Temperature Sensor) or a multiple byte record representing many parameters (e.g. analogue and discrete inputs);

— the parameter MemorySize shall specify the number of data bytes referenced by the parameter RecordLocalIdentifier;

— the parameter PositionInRecordLocalIdentifier shall specify the starting position in the record stored in the server's memory.

The request may consist of multiple definitions.

Table 63 — DynamicallyDefineLocalIdentifier Request Message DefinitionMode=DefineByCommonIdentifier

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**DynamicallyDefineLocalIdentifier Request Service Id**	**M**	**2C**	**DLIDDY**
2	DynamicallyDefinedLocalIdentifie	M	xx	**DDLOCID**
3	DefinitionMode=[DefineByCommonIdentifier]#1	M	02	**DEFMODE**
4	PositionInDynamicallyDefinedLocalIdentifier#1	M	xx	**PIDYDLID**
5	MemorySize#1	M	xx	**MEMSIZE**
6	RecordCommonIdentifier#1 (High Byte)	M	xx	**RCIDHB**
7	RecordCommonIdentifier#1 (Low Byte)	M	xx	**RCIDLB**
8	PositionInRecordCommonIdentifier#1	M	xx	**PIRLOCID**
9	DefinitionMode=[DefineByCommonIdentifier]#2	M	02	**DEFMODE**
10	PositionInDynamicallyDefinedLocalIdentifier#2	M	xx	**PIDYDLID**
11	MemorySize#2	M	xx	**MEMSIZE**
12	RecordCommonIdentifier#2 (High Byte)	M	xx	**RCIDHB**
13	RecordCommonIdentifier#2 (Low Byte)	M	xx	**RCIDLB**
14	PositionInRecordCommonIdentifier#2	M	xx	**PIRLOCID**
:	**:**	**:**	**:**	**:**
n-5	DefinitionMode=[DefineByCommonIdentifier]#m	M	02	**DEFMODE**
n-4	PositionInDynamicallyDefinedLocalIdentifier#m	M	xx	**PIDYDLID**
n-3	MemorySize#m	M	xx	**MEMSIZE**
n-2	RecordCommonIdentifier#m (High Byte)	M	xx	**RCIDHB**
n-1	RecordCommonIdentifier#m (Low Byte)	M	xx	**RCIDLB**
n	PositionInRecordCommonIdentifier#m	M	xx	**PIRLOCID**

This message is used by the client to define dynamically a local identifier in the request message. If the DefinitionMode parameter is set to DefineByCommonIdentifier the following procedure shall be followed:

— the parameter PositionInDynamicallyDefinedLocalIdentifier shall specify the position in the record referenced by the parameter DynamicallyDefinedLocalIdentifier. The record specified may be a "1 byte" parameter (e.g. Engine Coolant Temperature Sensor) or a multiple byte record representing many parameters (e.g. analogue and discrete inputs);

— the parameter MemorySize shall specify the number of data bytes referenced by the parameter RecordCommonIdentifier;

— the parameter PositionInRecordCommonIdentifier shall specify the starting position in the record stored in the server's memory.

The request may consist of multiple definitions.

Table 64 — DynamicallyDefineLocalIdentifier Request Message DefinitionMode=DefineByMemoryAddress

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**DynamicallyDefineLocalIdentifier Request Service Id**	**M**	**2C**	**DLIDDY**
2	DynamicallyDefinedLocalIdentifier	M	xx	**DDLOCID**
3	DefinitionMode=[DefineByMemoryAddress]#1	M	03	**DEFMODE**
4	PositionInDynamicallyDefinedLocalIdentifier#1	M	xx	**PIDYDLID**
5	MemorySize#1	M	xx	**MEMSIZE**
6	MemoryAddress#1 (High Byte)	M	xx	**MEMAHB**
7	MemoryAddress#1 (Middle Byte)	M	xx	**MEMAMB**
8	MemoryAddress#1 (Low Byte)	M	xx	**MEMALB**
9	DefinitionMode=[DefineByMemoryAddress]#2	M	03	**DEFMODE**
10	PositionInDynamicallyDefinedLocalIdentifier#2	M	xx	**PIDYDLID**
11	MemorySize#2	M	xx	**MEMSIZE**
12	MemoryAddress#2 (High Byte)	M	xx	**MEMAHB**
13	MemoryAddress#2 (Middle Byte)	M	xx	**MEMAMB**
14	MemoryAddress#2 (Low Byte)	M	xx	**MEMALB**
:	**:**	**:**	**:**	**:**
n-5	DefinitionMode=[DefineByMemoryAddress]#m	M	03	**DEFMODE**
n-4	PositionInDynamicallyDefinedLocalIdentifier#m	M	xx	**PIDYDLID**
n-3	MemorySize#m	M	xx	**MEMSIZE**
n-2	MemoryAddress#m (High Byte)	M	xx	**MEMAHB**
n-1	MemoryAddress#m (Middle Byte)	M	xx	**MEMAMB**
n	MemoryAddress#m (Low Byte)	M	xx	**MEMALB**

This message is used by the client to dynamically define a local identifier in the request message. If the DefinitionMode parameter is set to DefineByMemoryAddress the following procedure shall be followed:

— the parameter PositionInDynamicallyDefinedLocalIdentifier shall specify the position in the record referenced by the parameter DynamicallyDefinedLocalIdentifier. The record specified may be a "1 byte" parameter (e.g. Engine Coolant Temperature Sensor) or a multiple byte record representing many parameters (e.g. analogue and discrete inputs);

— the parameter MemorySize shall specify the number of data bytes starting at the specified MemoryAddress in the server's memory.

The request may consist of multiple definitions.

NOTE — The parameter PositionInRecordLocal/CommonIdentifier is not used in the request message if DefinitionMode is set to DefineByMemoryAddress.

Table 65 — DynamicallyDefineLocalIdentifier Request Message DefinitionMode=ClearDynamicallyDefinedLocalIdentifier

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**DynamicallyDefineLocalIdentifier Request Service Id**	**M**	**2C**	**DLIDDY**
2	DynamicallyDefinedLocalIdentifier	M	xx	**DDLOCID**
3	DefinitionMode=[ClearDynamicallyDefinedLocalIdentifier]	M	04	**DEFMODE**

This service is used by the client to clear a DynamicallyDefinedLocalIdentifier. The DefinitionMode parameter is set to ClearDynamicallyDefinedLocalIdentifier. This request message shall free up the server's memory which is used to create a DynamicallyDefinedLocalIdentifier data record.

Tables 66 and 67 describe the different response messages for DynamicallyDefineLocalIdentifier service.

Table 66 — DynamicallyDefineLocalIdentifier Positive Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**DynamicallyDefineLocalIdentifier Positive Response SId**	**M**	**6C**	**DLIDDYPR**
2	DynamicallyDefinedLocalIdentifier	M	xx	**DDLOCID**

Table 67 — DynamicallyDefineLocalIdentifier Negative Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**NegativeResponse Service Id**	**S**	**7F**	**NACK**
2	**DynamicallyDefineLocalIdentifierRequest Service Id**	**M**	**2C**	**DLIDDY**
3	ResponseCode=[KWP2000ResponseCode, ManufacturerSpecific]	M	xx=[00-7F, 80-FF]	**RC**

7.4.3 Message description

The server shall send a positive response message after it has erased the dynamically defined local Identifier record.

This service is used by the client to clear the data record in the server's memory by sending a DynamicallyDefineLocalIdentifier request message including the parameter DynamicallyDefined-LocalIdentifier to be cleared and the parameter DefinitionMode set to ClearDynamicallyDefined-LocalIdentifier.

7.4.4 Message flow examples

The message flow example in table 68 shows a DynamicallyDefineLocalIdentifier service which consists of a single request and positive response message to define the parameters referenced by the DynamicallyDefinedLocalIdentifier. The dynamically defined data record is then read by a ReadDataByLocalidentifier service.

Table 68 — Message flow example of single DynamicallyDefineLocalIdentifier service followed by a ReadDataByLocalIdentifier service

time	client (tester)	server (ECU)
P3	DynamicallyDefineLocalId.Request[...]	
P2		DynamicallyDefineLocalId.PositiveResponse[...]
P3	ReadDataByLocalId.Request[...]	
P2		ReadDataByLocalId.PositiveResponse[...]

The message flow example in table 69 shows a DynamicallyDefineLocalIdentifier service which consists of multiple requests and positive response messages to define the parameters referenced by the DynamicallyDefinedLocalIdentifier. The dynamically defined data record is then read by a ReadDataByLocalidentifier service.

Table 69 — Message flow example of multiple DynamicallyDefineLocalIdentifier services followed by a ReadDataByLocalIdentifier service

time	client (tester)	server (ECU)
P3	DynamicallyDefineLocalId.Request#1[...]	
P2		DynamicallyDefineLocalId.PositiveResponse#1[...]
P3	DynamicallyDefineLocalId.Request#2[...]	
P2		DynamicallyDefineLocalId.PositiveResponse#2[...]
:	:	:
P3	DynamicallyDefineLocalId.Request#n[...]	
P2		DynamicallyDefineLocalId.PositiveResponse#2[...]
P3	ReadDataByLocalId.Request[...]	
P2		ReadDataByLocalId.PositiveResponse[...]

The DynamicallyDefineLocalIdentifier service shall only be used with physical addressing.

7.5 WriteDataByLocalIdentifier service

7.5.1 Parameter definition

The parameter ***RecordLocalIdentifier*** is defined in 7.1.1.

The parameter ***RecordValue*** is defined in 7.1.1.

7.5.2 Message data bytes

Tables 70 to 72 describe the different messages for WriteDataByLocalIdentifier service.

Table 70 — WriteDataByLocalIdentifier Request Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**WriteDataByLocalIdentifier Request Service Id**	**M**	**3B**	**WDBLID**
2	RecordLocalIdentifier	M	xx	**RECLID**
3 : n	RecordValue#1 : RecordValue#m	M : M	xx : xx	**RECVAL**

Table 71 — WriteDataByLocalIdentifier Positive Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**WriteDataByLocalIdentifier Positive Response Service Id**	**S**	**7B**	**WDBLIDPR**
2	RecordLocalIdentifier	M	xx	**RECLID**

Table 72 — WriteDataByLocalIdentifier Negative Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**NegativeResponse Service Id**	**S**	**7F**	**NACK**
2	**WriteDataByLocalIdentifier Request Service Id**	**M**	**3B**	**WDBLID**
3	ResponseCode=[KWP2000ResponseCode, ManufacturerSpecific]	M	xx=[00-7F, 80-FF]	**RC**

7.5.3 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;

— set Vehicle Identification Number (VIN);

— change calibration values.

7.5.4 Message flow example

See message flow example of Physical Addressed Service in 5.3.1.1.

7.6 WriteDataByCommonIdentifier service

7.6.1 Parameter definition

The parameter ***RecordCommonIdentifier*** is defined in 7.2.1.

The parameter ***RecordValue*** is defined in 7.1.1.

7.6.2 Message data bytes

Tables 73 to 75 describe the different messages for WriteDataByCommonIdentifier service.

Table 73 — WriteDataByCommonIdentifier Request Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**WriteDataByCommonIdentifier Request Service Id**	**M**	**2E**	**WDBCID**
2	RecordCommonIdentifier (High Byte)	M	xx	**RECCIDHB**
3	RecordCommonIdentifier (Low Byte)	M	xx	**RECCIDLB**
4 : n	RecordValue#1 : RecordValue#m	M : M	xx : xx	**RECVAL**

Table 74 — WriteDataByCommonIdentifier Positive Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**WriteDataByCommonIdentifier Positive Response SId**	**S**	**6E**	**WDBCIDPR**
2	RecordCommonIdentifier (High Byte)	M	xx	**RECCIDHB**
3	RecordCommonIdentifier (Low Byte)	M	xx	**RECCIDLB**

Table 75 — WriteDataByCommonIdentifier Negative Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**NegativeResponse Service Id**	**S**	**7F**	**NACK**
2	**WriteDataByCommonIdentifier Request Service Id**	**M**	**2E**	**WDBCID**
3	ResponseCode=[KWP2000ResponseCode, ManufacturerSpecific]	M	xx=[00-7F, 80-FF]	**RC**

7.6.3 Message description

The WriteDataByCommonIdentifier service is used by the client to write RecordValues (data values) to multiple servers with a single request message. The data are identified by a RecordCommonIdentifier. It is the vehicle manufacturer's responsibility that the server's conditions are met when performing this service.

Possible uses for this service are:

— clear non-volatile memory;

— reset learned values;

— set option content;

— set Vehicle Identification Number (VIN).

7.6.4 Message flow example

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

7.7 WriteMemoryByAddress service

7.7.1 Parameter definition

The parameter ***MemoryAddress*** is defined in 7.3.1.

The parameter ***MemorySize*** is defined in 7.3.1.

The parameter ***RecordValue*** is defined in 7.1.1.

7.7.2 Message data bytes

Tables 76 to 78 describe the different messages for WriteMemoryByAddress service.

Table 76 — WriteMemoryByAddress Request Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**WriteMemoryByAddress Request Service Id**	**M**	**3D**	**WMBA**
2	MemoryAddress (High Byte)	M	xx	**MEMAHB**
3	MemoryAddress (Middle Byte)	M	xx	**MEMAMB**
4	MemoryAddress (Low Byte)	M	xx	**MEMALB**
5	MemorySize	M	xx	**MEMSIZE**
6 : n	RecordValue#1 : RecordValue#m	M : M	xx : xx	**RECVAL**

Table 77 — WriteMemoryByAddress Positive Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**WriteMemoryByAddress Positive Response**	**S**	**7D**	**WMBAPR**
2	MemoryAddress (High Byte)	U	xx	**MEMAHB**
3	MemoryAddress (Middle Byte)	U	xx	**MEMAMB**
4	MemoryAddress (Low Byte)	U	xx	**MEMALB**

Table 78 — WriteMemoryByAddress Negative Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**WriteMemoryByAddress Negative Response**	**S**	**7F**	**NACK**
2	**WriteMemoryByAddress Request Service Id**	**M**	**3D**	**WMBA**
3	ResponseCode=[KWP2000ResponseCode, ManufacturerSpecific]	M	xx=[00-7F, 80-FF]	**RC**

7.7.3 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;

— set option content;

— set Vehicle Identification Number (VIN);

— change calibration values.

7.7.4 Message flow example

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

7.8 SetDataRates service

7.8.1 Parameter definition

The parameters ***SlowRate, MediumRate*** and ***FastRate*** shall be used to specify the transmission speed of positive response messages to be sent by the server if the appropriate request message has specified the TransmissionMode parameter to one of the above rates. Table 79 lists possible values to be assigned to ***SlowRate, MediumRate*** and ***FastRate***.

Table 79 — Definition of SlowRate, MediumRate and FastRate values

Hex	Description
00	This value shall cause the server to send positive response messages as fast as possible based on the vehicle manufacturer's definition of SlowRate, MediumRate and FastRate.
01 - FE	This range of values shall be used by the vehicle manufacturer to specify/change the default rates of the parameters SlowRate, MediumRate and FastRate. It is the vehicle manufacturer's responsibility to define these values and their corresponding transmission rates of the parameters SlowRate, MediumRate and FastRate.
FF	This value shall not change the current rate of positive response messages send by the server.

7.8.2 Message data bytes

Tables 80 to 82 describe the different messages for SetDataRates service.

Table 80 — SetDataRates Request Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**SetDataRates Request Service Id**	**M**	**26**	**SDR**
2	SlowRate	M	xx	**SLRATE**
3	MediumRate	M	xx	**MDRATE**
4	FastRate	M	xx	**FTRATE**

Table 81 — SetDataRates Positive Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**SetDataRates Positive Response**	**S**	**66**	**SDRPR**

Table 82 — SetDataRates Negative Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**NegativeResponse Service Id**	**S**	**7F**	**NACK**
2	**SetDataRates Request Service Id**	**M**	**26**	**SDR**
3	ResponseCode=[KWP2000ResponseCode, ManufacturerSpecific]	M	xx=[00-7F, 80-FF]	**RC**

7.8.3 Message description

The SetDataRates service is used by the client to overwrite the default timing of periodic transmissions used by the server. This message only affects those services which make use of the parameter TransmissionMode in the ReadDataByLocalIdentifier, ReadDataByCommonIdentifier and ReadMemoryByAddress request messages. It is the vehicle manufacturer's responsibility to specify values and their corresponding periodic transmission rates.

Regardless of which rate is modified in the request, all rates shall have values as defined in table 78.

7.8.4 Message flow example

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

8 Stored Data Transmission functional unit

The services provided by this functional unit are described in table 83.

Table 83 — Stored Data Transmission functional unit

Service name	Description
ReadDiagnosticTroubleCodes	The client requests from the server the transmission of both, number of DTC and values of the diagnostic trouble codes.
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.

8.1 ReadDiagnosticTroubleCodes service

8.1.1 Parameter definition

The parameter ***GroupOfDTC*** is used by the ReadDTC, ReadDTCByStatus and ReadStatusOfDTC services to select to which functional group of DTC or to which specific DTC access is requested. Format and length of this parameter are vehicle-manufacturer-specific.

NOTE - Standardized DTCs and DTCs formats are specified in SAE J 2012 for passenger cars and in SAE J 1587 for heavy duty vehicles.

The parameter ***NumberOfDTC*** is used by the ReadNumberOfDTC positive response to indicate how many DTCs of the group specified in the request have been stored by the client(s) (see table 84). A specific standard value is NoDTCStored.

Table 84 — Definition of NumberOfDTC values

Hex	Description
00	**NoDTCStored** This value indicates to the server that the client(s) doesn't (don't) have any DTC stored.
01-FF	**NumberOfDTCStored** This range of values indicate to the server that the client(s) has (have) stored DTC(s).

The parameter ***ListOfDTC*** is used by the server to report DTC(s) stored by the client(s).

NOTE — Standardized DTCs and DTCs formats are specified in SAE J 2012 for passenger cars and in SAE J 1587 for heavy duty vehicles.

8.1.2 Message data bytes

Tables 85 to 87 describe the different messages for ReadDiagnosticTroubleCodes services.

Table 85 — ReadDiagnosticTroubleCodes Request Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**ReadDiagnosticTroubleCodes Request Service Id**	**M**	**13**	**RDDTC**
2 : n	GroupOfDTC#1 : GroupOfDTC#m	U : U	xx : xx	**GODTC**

Table 86 — ReadDiagnosticTroubleCodes Positive Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**ReadDiagnosticTroubleCodes Positive Resp. Service Id**	**M**	**53**	**RDDTCPR**
2	NumberOfDTC	M	xx	**#DTC**
 3 : : : : : : n	ListOfDTC=[DTC#1 : DTC#1 : : DTC#m : DTC#m]	C1)	 xx : xx : : xx : xx	**LSTOFDTC**
1) Condition: ListOfDTC is only present if NumberOfDTC is > "$00".				

Table 87 — ReadDiagnosticTroubleCodes Negative Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**NegativeResponse Service Id**	**S**	**7F**	**NACK**
2	**ReadDiagnosticTroubleCodes Request Service Id**	**M**	**13**	**RDDTC**
3	ResponseCode=[KWP2000ResponseCode, ManufacturerSpecific]	M	xx=[00-7F, 80-FF]	**RC**

8.1.3 Message description

The ReadDiagnosticTroubleCodes service is used by the client to read diagnostic trouble codes from the server's memory. If the user optional parameter GroupOfDTC is present in the request message then the server(s) shall only report DTC(s) based on the functional group selected by the client.

If the server(s) does (do) not have any DTC stored it (they) shall set the parameter NumberOfDTCStored to "$00". This causes the server(s) not to include DTC(s) in the parameter ListOfDTC in the positive response message.

8.1.4 Message flow example

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

8.2 ReadDiagnosticTroubleCodesByStatus service

8.2.1 Parameter definition

The parameter ***StatusOfDTC*** is used by ReadDiagnosticTroubleCodesByStatus and ReadStatusOfDiagnosticTroubleCodes service to inform the client about the status of each DTC stored in the server's memory. Format and length of this parameter are vehicle-manufacturer-specific.

NOTE — Standardized DTCs and DTCs formats are specified in SAE J 2012 for passenger cars and in SAE J 1587 for heavy duty vehicles.

The parameter ***GroupOfDTC*** is defined in 8.1.1.

The parameter ***NumberOfDTC*** is defined in 8.1.1.

The parameter ***ListOfDTCAndStatus*** is used by the server to report DTC(s) and their associated status.

8.2.2 Message data bytes

Tables 88 to 90 describe the different messages for ReadDiagnosticTroubleCodesByStatus.

Table 88 — ReadDiagnosticTroubleCodesByStatus Request Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**ReadDiagnosticTroubleCodesByStatus Request SId**	**M**	**18**	**RDDTCS**
2 : n	StatusOfDTC#1 : StatusOfDTC#m	M : U	xx : xx	**STATDTC**
n+1 : n+#m	GroupOfDTC#1 : GroupOfDTC#m	U : U	xx : xx	**GODTC**

Table 89 — ReadDiagnosticTroubleCodesByStatus Positive Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**ReadDiagnosticTroubleCodesByStatus Pos. Response SId**	**M**	**58**	**RDDTCSPR**
2	NumberOfDTC	M	xx	**#DTC**
 3 : : : : : : : : : : : : n	ListOfDTCAndStatus=[DTC#1 : DTC#1 StatusOfDTC#1 : StatusOfDTC#1 : : DTC#m : DTC#m StatusOfDTC#m : StatusOfDTC#m]	C[1]	 xx : xx xx : xx : : xx : xx xx : xx	**LSTDTCST**
1) Condition: ListOfDTCAndStatus is only present if NumberOfDTC is > "$00".				

Table 90 — ReadDiagnosticTroubleCodesByStatus Negative Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**NegativeResponse Service Id**	**S**	**7F**	**NACK**
2	**ReadDiagnosticTroubleCodesByStatus Request SId**	**M**	**18**	**RDDTC**
3	ResponseCode=[KWP2000ResponseCode, ManufacturerSpecific]	M	xx=[00-7F, 80-FF]	**RC**

8.2.3 Message description

The ReadDiagnosticTroubleCodesByStatus service is used by the client to read diagnostic trouble codes by status from the server's memory. If the user optional parameter GroupOfDTC is present in the request message then the server(s) shall only report DTC(s) with status information based on the functional group selected by the client.

If the server(s) does (do) not have any DTC with status information stored it (they) shall set the parameter NumberOfDTCStored to "$00". This causes the server(s) not to include DTC(s) and status information in the parameter ListOfDTC in the positive response message.

8.2.4 Message flow example

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

8.3 ReadStatusOfDiagnosticTroubleCodes service

8.3.1 Parameter definition

The parameter ***GroupOfDTC*** is defined in 8.1.1.

The parameter ***NumberOfDTC*** is defined in 8.1.1.

The parameter ***ListOfDTCAndStatus*** is defined in 8.2.1.

8.3.2 Message data bytes

Tables 91 to 93 describe the different messages for ReadStatusOfDiagnosticTroubleCodes.

Table 91 — ReadStatusOfDiagnosticTroubleCodes Request Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**ReadStatusOfDiagnosticTroubleCodes Request SId**	**M**	**17**	**RDSDTC**
2 **:** n	GroupOfDTC#1 **:** GroupOfDTC#m	U **:** U	xx **:** xx	**GODTC**

Table 92 — ReadStatusOfDiagnosticTroubleCodes Positive Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**ReadStatusOfDiagnosticTroubleCodes Pos. Resp. SId**	**M**	**57**	**RDSDTCPR**
2	NumberOfDTC	M	xx	**#DTC**
 3 : : : : : : : : : : : : n	ListOfDTCAndStatus=[DTC#1 : DTC#1 StatusOfDTC#1 : StatusOfDTC#1 : : DTC#m : DTC#m StatusOfDTC#m : StatusOfDTC#m]	C1)	 xx : xx xx : xx : : xx : xx xx : xx	**LSTDTCST**
1) Condition: ListOfDTCAndStatus is only present if NumberOfDTC is > "$00".				

Table 93 — ReadStatusOfDiagnosticTroubleCodes Negative Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**NegativeResponse Service Id**	**S**	**7F**	**NACK**
2	**ReadStatusOfDiagnosticTroubleCodes Request SId**	**M**	**17**	**RDDTC**
3	ResponseCode=[KWP2000ResponseCode, ManufacturerSpecific]	M	xx=[00-7F, 80-FF]	**RC**

8.3.3 Message description

The ReadStatusOfDiagnosticTroubleCodes service is used by the client to read diagnostic trouble codes with their associated status from the server's memory. If the user optional parameter GroupOfDTC is present in the request message then the server(s) shall only report DTC(s) with status information based on the functional group selected by the client.

If the server(s) does (do) not have any DTC with status information stored it (they) shall set the parameter NumberOfDTCStored to "$00". This causes the server(s) not to include DTC(s) and status information in the parameter ListOfDTC in the positive response message.

8.3.4 Message flow example

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

8.4 ReadFreezeFrameData service

8.4.1 Parameter definition

The parameter ***FreezeFrameNumber*** is used by the client to identify the freeze frame to be read from the server's memory.

The parameter ***RecordAccessMethodIdentifier*** is used by the client to define the access method which shall be used in the RecordIdentification parameter to access a specific freeze frame data record within the FreezeFrameData; see table 94.

Table 94 — Definition of RecordAccessMethodIdentifier values

Hex	Description
00	**RequestAllData** This value indicates to the server(s) that the client requests all freeze frame data stored within the server's memory.
01	**RequestByLocalIdentifier** This value indicates to the server(s) that the client requests freeze frame data identified by a local identifier.
02	**RequestByCommonIdentifier** This value indicates to the server(s) that the client requests freeze frame data identified by a common identifier.
03	**RequestByMemoryAddress** This value indicates to the server(s) that the client requests freeze frame data identified by a memory address. If the server supports "16 bit" wide address range the high byte of the MemoryAddress shall be used as a MemoryIdentifier if required.
04	**RequestByDTC** This value indicates to the server(s) that the client requests freeze frame data identified by a DTC.
05 - 7F	**ReservedByDocument** This range of values is reserved by this document for future definition.
80 - FF	**ManufacturerSpecific** This range of values is reserved for vehicle-manufacturer-specific use.

The parameter ***RecordIdentification*** is used by the client to identify the record within the freeze frame referenced by the parameter FreezeFrameNumber; see table 95.

Table 95 — Definition of RecordIdentification values

Hex	Description
00	**AllData** This value identifies the entire record which includes all data of a single freeze frame.
00 - FF	**LocalIdentifier** This range of values identifies a record which includes the data within a freeze frame referenced by a LocalIdentifier.
00 - FF 00 - FF	**CommonIdentifier (High Byte)** **CommonIdentifier (low Byte)** This range of values identifies a record which includes the data within a freeze frame referenced by a CommonIdentifier.
00 - FF 00 - FF 00 - FF	**MemoryAddress (High Byte)** **MemoryAddress (Middle Byte)** **MemoryAddress (Low Byte)** This range of values identifies a record which includes the data within a freeze frame referenced by a MemoryAddress.
00 - FF : 00 - FF	**DTC (Diagnostic Trouble Code)** This range of values identifies a record which includes the data within a freeze frame referenced by a DTC.

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.

8.4.2 Message data bytes

Tables 96 to 98 describe the different messages for ReadFreezeFrameData service.

Table 96 — ReadFreezeFrameData Request Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**ReadFreezeFrameData Request Service Id**	**M**	**12**	**RDFFD**
2	FreezeFrameNumber	M	xx	**FRZFR#**
3	RecordAccessMethodIdentifier	U	xx=[	**RECAMID**
	RequestAllData		00	
	RequestByLocalIdentifier		01	
	RequestByCommonIdentifier		02	
	RequestByMemoryAddress		03	
	RequestByDTC		04	
	ReservedByDocument		05 - 7F	
	ManufacturerSpecific[1)]]		80 - FF]	
4	RecordIdentification=[[1)]		xx=[	**RECID**
	AllData	U[2)]	00	
	LocalIdentifier	U[3)]	xx	
	CommonIdentifier (High Byte)	U[4)]	xx	
	MemoryAddress (High Byte)	U[5)]	xx	
	DTC	U[6)]	xx	
	ReservedByDocument	U[7)]	xx	
	ManufacturerSpecific	U[8)]	xx	
5	CommonIdentifier (Low Byte)	U[4)]	xx	
	MemoryAddress (Middle Byte)	U[5)]	xx	
	DTC	U[6)]	xx	
6	MemoryAddress (Low Byte)	U[5)]	xx	
	DTC	U[6)]	xx	
:	:		:	
n	DTC]	U[6)]	xx	
			]	

1) Parameters of the request shall only be present together.

2) Condition: RecordAccessMethodIdentifier = RequestAllData.

3) Condition: RecordAccessMethodIdentifier = RequestByLocalIdentifier.

4) Condition: RecordAccessMethodIdentifier = RequestByCommonIdentifier.

5) Condition RecordAccessMethodIdentifier = RequestByMemoryAddress.

6) Condition: RecordAccessMethodIdentifier = RequestByDTC.

7) Condition: RecordAccessMethodIdentifier = ReservedByDocument.

8) Condition: RecordAccessMethodIdentifier = ManufacturerSpecific.

Table 97 — ReadFreezeFrameData Positive Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**ReadFreezeFrameData Positive Response Service Id**	**S**	**52**	**RDFFDNR**
2	FreezeFrameNumber	M	xx	**FRZFR#**
3	FreezeFrameData	M	xx	**FRZFRD**
:	:		:	
k-2	FreezeFrameData		xx	
k-1	RecordAccessMethodIdentifier=[	U	xx=[	**RECAMID**
	RequestAllData		00	
	RequestByLocalIdentifier		01	
	RequestByCommonIdentifier		02	
	RequestByMemoryAddress		03	
	RequestByDTC		04	
	ReservedByDocument		05 - 7F	
	ManufacturerSpecific[1]]		80 - FF]	
k	RecordIdentification=[[1]		xx=[	**RECID**
	AllData	U[2]	00	
	LocalIdentifier	U[3]	xx	
	CommonIdentifier (High Byte)	U[4]	xx	
	MemoryAddress (High Byte)	U[5]	xx	
	DTC	U[6]	xx	
	ReservedByDocument	U[7]	xx	
	ManufacturerSpecific	U[8]	xx	
k+1	CommonIdentifier (Low Byte)	U[4]	xx	
	MemoryAddress (Middle Byte)	U[5]	xx	
	DTC	6)	xx	
k+2	MemoryAddress (Low Byte)	U[5]	xx	
	DTC	U[6]	xx	
:	:		:	
n	DTC]	U[6]	xx]	

1) Parameters of the request shall only be present together.

2) Condition: RecordAccessMethodIdentifier = RequestAllData.

3) Condition: RecordAccessMethodIdentifier = RequestByLocalIdentifier.

4) Condition: RecordAccessMethodIdentifier = RequestByCommonIdentifier.

5) Condition RecordAccessMethodIdentifier = RequestByMemoryAddress.

6) Condition: RecordAccessMethodIdentifier = RequestByDTC.

7) Condition: RecordAccessMethodIdentifier = ReservedByDocument.

8) Condition: RecordAccessMethodIdentifier = ManufacturerSpecific.

Table 98 — ReadFreezeFrameData Negative Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**NegativeResponse Service Id**	**S**	**7F**	**NACK**
2	**ReadFreezeFrameData Request Service Id**	**M**	**12**	**RDFFD**
3	ResponseCode=[	M	xx=[	**RC**
	KWP2000ResponseCode,		00-7F,	
	ManufacturerSpecific]		80-FF]	

8.4.3 Message description

The ReadFreezeFrameData service is used by the client to read FreezeFrameData 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.

The user optional RecordAccessMethodIdentifier in the request message identifies the access method used by the client to request FreezeFrameData from the server's memory. In addition, the user optional parameter RecordIdentifier shall always be used in conjunction with the RecordAccessMethodIdentifier. The content, type and size of this parameter depends on the value of the RecordAccessMethodIdentifier (refer to table 94).

The server shall send a positive response message including the FreezeFrameNumber, FreezeFrameData and the user optional/conditional parameters RecordAccessMethodIdentifier and RecordIdentifier, if both were present in the request.

Data content and data format of the FreezeFrameData shall be specified by the vehicle manufacturer.

Typical uses for this mode are to report data stored upon detection of a system malfunction. Multiple frames of data may be stored before and/or after the malfunction has been detected.

8.4.4 Message flow example

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

8.5 ClearDiagnosticInformation service

8.5.1 Parameter definition

The parameter ***GroupOfDiagnosticInformation*** is used by the ClearDiagnosticInformation service to select to which functional group of DTC or which specific DTC is selected. Format and length of this parameter are vehicle-manufacturer-specific.

NOTE — Standardized DTCs and DTCs formats are specified in SAE J 2012 for passenger cars and in SAE J 1587 for heavy duty vehicles.

8.5.2 Message data bytes

Tables 99 to 101 describe the different messages for ClearDiagnosticInformation service.

Table 99 — ClearDiagnosticInformation Request Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**ClearDiagnosticInformation Request Service Id**	**M**	**14**	**CLRDTC**
2 : n	GroupOfDiagnosticInformation#1 : GroupOfDiagnosticInformation#m	U : U	xx : xx	**GODIN**

Table 100 — ClearDiagnosticInformation Positive Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**ClearDiagnosticInformation Positive Response SId**	**S**	**54**	**CLRDTCPR**
2 : n	GroupOfDiagnosticInformation#1 : GroupOfDiagnosticInformation#m	U1) : U1)	xx : xx	**GODIN**
1) Condition: parameter shall be present if present in the request.				

Table 101 — ClearDiagnosticInformation Negative Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**NegativeResponse Service Id**	**S**	**7F**	**NACK**
2	**ClearDiagnosticInformation Request Service Id**	**M**	**14**	**CLRDTC**
3	ResponseCode=[KWP2000ResponseCode, ManufacturerSpecific]	M	xx=[00-7F, 80-FF]	**RC**

8.5.3 Message description

The ClearDiagnosticInformation service is used by the client to clear diagnostic information in the server's memory. If the user optional parameter GroupOfDiagnosticInformation is present in the request message, then the server(s) shall clear all diagnostic information based on the functional group selected by the client. If a specific DTC shall be cleared the client shall send the GroupOfDiagnosticInformation parameter including the DTC to clear.

If the server(s) do/does not have any DTC and/or diagnostic information stored the server(s) shall send a positive response message after an appropriate request message has been received.

8.5.4 Message flow example

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

9 InputOutput Control functional unit

The services provided by this functional unit are described in table 102.

Table 102 — InputOutput Control functional unit

Service name	Description
InputOutputControlByLocalIdentifier	The client requests the control of an input/output specific to the server.
InputOutputControlByCommonIdentifier	The client requests the control of a common input/output.

9.1 InputOutputControlByLocalIdentifier service

9.1.1 Parameter definition

The parameter ***InputOutputLocalIdentifier*** in the InputOutputControlByLocalIdentifier request service identifies an ECU local input signal, internal parameter or output signal.

The parameter ***ControlOption*** is used by the InputOutputControlByLocal/Common-Identifier request service to describe how the server has to control its inputs or outputs. It may consist of multiple bytes which include specific control information to manipulate the input or output signal identified by the InputOutputLocal/CommonIdentifier.

The parameter ***ControlStatus*** is used by the InputOutputControlByLocal/CommonIdentifier positive response service to give to the client information about the status (e.g. feedback) of the requested control and other related information.

9.1.2 Message data bytes

Tables 103 to 105 describe the different messages for InputOutputControlByLocalIdentifier service.

Table 103 — InputOutputControlByLocalIdentifier Request Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**InputOutputControlByLocalIdentifier Request SId**	**M**	**30**	**IOCBLID**
2	InputOutputLocalIdentifier	M	xx	**IOLID**
3 : n	ControlOption#1 : ControlOption#m	U : U	xx : xx	**CRTLOPT**

Table 104 — InputOutputControlByLocalIdentifier Positive Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**InputOutputControlByLocalIdentifier Positive Response SId**	**S**	**70**	**IOCBLIDPR**
2	InputOutputLocalIdentifier	M	xx	**IOLID**
3 : n	ControlStatus#1 : ControlStatus#m	U : U	xx : xx	**CRTLSTAT**

Table 105 — InputOutputControlByLocalIdentifier Negative Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**NegativeResponse Service Id**	**S**	**7F**	**NACK**
2	**InputOutputControlByLocalIdentifier Request SId**	**M**	**30**	**IOCBLID**
3	ResponseCode=[KWP2000ResponseCode, ManufacturerSpecific]	M	xx=[00-7F, 80-FF]	**RC**

9.1.3 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.

9.1.4 Message flow example

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

9.2 InputOutputControlByCommonIdentifier service

9.2.1 Parameter definition

The parameter ***InputOutputCommonIdentifier*** in the InputOutputControlByCommonIdentifier request service identifies an input or output common to the ECU(s).

The parameter ***ControlOption*** is defined in 9.1.1.

The parameter ***ControlStatus*** is defined in 9.1.1.

9.2.2 Message data bytes

Tables 106 to 108 describe the different messages for InputOutputControlByCommonIdentifier service.

Table 106 — InputOutputControlByCommonIdentifier Request Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**InputOutputControlByCommonIdentifier Request SId**	**M**	**2F**	**IOCBCID**
2	InputOutputCommonIdentifier (High Byte)	M	xx	**IOCIDHB**
3	InputOutputCommonIdentifier (Low Byte)	M	xx	**IOCIDLB**
4 : n	ControlOption#1 : ControlOption#m	U : U	xx : xx	**CRTLOPT**

Table 107 — InputOutputControlByCommonIdentifier Positive Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**InputOutputControlByCommonIdentifier Positive Resp. SId**	**S**	**6F**	**IOCBCIDPR**
2	InputOutputCommonIdentifier (High Byte)	M	xx	**IOCIDHB**
3	InputOutputCommonIdentifier (Low Byte)	M	xx	**IOCIDLB**
4 : n	ControlStatus#1 : ControlStatus#m	U : U	xx : xx	**CRTLSTAT**

Table 108 — InputOutputControlByCommonIdentifier Negative Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**NegativeResponse Service Id**	**S**	**7F**	**NACK**
2	**InputOutputControlByCommonIdentifier Request SId**	**M**	**2F**	**IOCBCID**
3	ResponseCode=[KWP2000ResponseCode, ManufacturerSpecific]	M	xx=[00-7F, 80-FF]	**RC**

9.2.3 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.

9.2.4 Message flow example

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

10 Remote Activation Of Routine functional unit

The services provided by this functional unit are described in table 109.

Table 109 — Remote Activation Of Routine functional unit

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.

10.1 StartRoutineByLocalIdentifier service

10.1.1 Parameter definition

The parameter ***RoutineLocalIdentifier*** in the Start/StopRoutineByLocalIdentifier and RequestRoutineResultsByLocalIdentifier request message identifies a server local routine.

The parameter ***RoutineEntryOption*** is used by the StartRoutineByLocalIdentifier and StartRoutineByAddress request message to specify the start conditions of the routine.

The parameter ***RoutineEntryStatus*** 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.

10.1.2 Message data bytes

Tables 110 to 112 describe the different messages for StartRoutineByLocalIdentifier service.

Table 110 — StartRoutineByLocalIdentifier Request Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**StartRoutineByLocalIdentifier Request Service Id**	**M**	**31**	**SRBLID**
2	RoutineLocalIdentifier	M	xx	**RLOCID**
3 : n	RoutineEntryOption#1 : RoutineEntryOption#m	U : U	xx : xx	**RENTOPT**

Table 111 — StartRoutineByLocalIdentifier Positive Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**StartRoutineByLocalIdentifier Positive Response SId**	**S**	**71**	**SRBLIDPR**
2	RoutineLocalIdentifier	M	xx	**RLOCID**
3 : n	RoutineEntryStatus#1 : RoutineEntryStatus#m	U : U	xx : xx	**RENTSTAT**

Table 112 — StartRoutineByLocalIdentifier Negative Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**NegativeResponse Service Id**	**S**	**7F**	**NACK**
2	**StartRoutineByLocalIdentifier Request Service Id**	**M**	**31**	**SRBLID**
3	ResponseCode=[KWP2000ResponseCode, ManufacturerSpecific]	M	xx=[00-7F, 80-FF]	**RC**

10.1.3 Message description

The StartRoutineByLocalIdentifier service is used by the client to start a routine in the server's memory.

The routine in the server shall be started after the positive response message has been sent. The routine shall be stopped until a StopRoutineByLocalIdentifier service is issued.

The routines could either be tests that run instead of normal operating code or could be routines that are enabled and executed with the normal operating code running. In particular in the first case, it might be necessary to switch the server in a specific diagnostic mode 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 shall be of any type and length defined within KWP 2000.

Examples of RoutineEntryOptions are: TimeToRun, StartUpVariables, etc.

10.1.4 Message flow example

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

A complete message flow example is shown in 10.5.4.

10.2 StartRoutineByAddress service

10.2.1 Parameter definition

The parameter ***RoutineAddress*** in the Start/StopRoutineByAddress and RequestRoutine-ResultsByAddress request message identifies a server local routine.

The parameter ***RoutineEntryOption*** is defined in 10.1.1.

The parameter ***RoutineEntryStatus*** is defined in 10.1.1.

10.2.2 Message data bytes

Tables 113 to 115 describe the different messages for StartRoutineByAddress service.

Table 113 — StartRoutineByAddress Request Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**StartRoutineByAddress Request Service Id**	**M**	**38**	**STRBAD**
2	RoutineAddress (High Byte)	M	xx	**MEMAHB**
3	RoutineAddress (Middle Byte)	M	xx	**MEMAMB**
4	RoutineAddress (Low Byte)	M	xx	**MEMALB**
5 : n	RoutineEntryOption#1 : RoutineEntryOption#m	U : U	xx : xx	**RENTOPT**

Table 114 — StartRoutineByAddress Positive Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**StartRoutineByAddress Positive Response Service Id**	**S**	**78**	**STRBADPR**
2	RoutineAddress (High Byte)	M	xx	**MEMAHB**
3	RoutineAddress (Middle Byte)	M	xx	**MEMAMB**
4	RoutineAddress (Low Byte)	M	xx	**MEMALB**
5 : n	RoutineEntryStatus#1 : RoutineEntryStatus#m	U : U	xx : xx	**RENTSTAT**

Table 115 — StartRoutineByAddress Negative Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**NegativeResponse Service Id**	**S**	**7F**	**NACK**
2	**StartRoutineByAddress Request Service Id**	**M**	**38**	**STRBAD**
3	ResponseCode=[KWP2000ResponseCode, ManufacturerSpecific]	M	xx=[00-7F, 80-FF]	**RC**

10.2.3 Message description

The StartRoutineByAddress service is used by the client to start a routine in the server's memory.

The routine in the server shall be started after the positive response message has been sent. The routine shall be stopped until a StopRoutineByAddress service is received.

The routines could either be tests that run instead of normal operating code or be routines that are enabled and executed with the normal operating code running. In particular in the first case, it might be necessary to switch the server in a specific diagnostic mode 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 and shall be of any type and length defined within KWP 2000.

Examples of RoutineEntryOptions are: TimeToRun, StartUpVariables, etc.

10.2.4 Message flow example

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

A complete message flow example is shown in 10.6.4.

10.3 StopRoutineByLocalIdentifier service

10.3.1 Parameter definition

The parameter ***RoutineLocalIdentifier*** is defined in 10.1.1.

The parameter ***RoutineExitOption*** is used by the StopRoutineByLocalIdentifier and StopRoutineByAddress request message to specify the stop conditions of the routine.

The parameter ***RoutineExitStatus*** 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 table 116.

Table 116 — Definition of RoutineExitStatus values

Hex	Description
00-60	**ReservedByDocument** This range of values is reserved by this document (refer to 4.4 for Response Codes).
61	**NormalExitWithResultsAvailable**
62	**NormalExitWithoutResultsAvailable**
63	**AbNormalExitWithResultsAvailable**
64	**AbNormalExitWithoutResultsAvailable**
65 - 7F	**ReservedByDocument** This range of values is reserved by this document (refer to 4.4 for Response Codes).
80 - FF	**ManufacturerSpecificCodes** This range of values is reserved for vehicle-manufacturer-specific use.

10.3.2 Message data bytes

Tables 117 to 119 describe the different messages for StopRoutineByLocalIdentifier service.

Table 117 — StopRoutineByLocalIdentifier Request Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**StopRoutineByLocalIdentifier Request Service Id**	**M**	**32**	**SPRBLID**
2	RoutineLocalIdentifier	M	xx	**RLOCID**
3 : n	RoutineExitOption#1 : RoutineExitOption#m	U : U	xx : xx	**REXITOPT**

Table 118 — StopRoutineByLocalIdentifier Positive Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**StopRoutineByLocalIdentifier Positive Response**	**S**	**72**	**SPRBLIDPR**
2	RoutineLocalIdentifier	M	xx	**RLOCID**
3 : n	RoutineExitStatus#1 : RoutineExitStatus#m	U : U	xx : xx	**REXITSTAT**

Table 119 — StopRoutineByLocalIdentifier Negative Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**NegativeResponse Service Id**	**S**	**7F**	**NACK**
2	**StopRoutineByLocalIdentifier Request Service Id**	**M**	**32**	**SPRBLID**
3	ResponseCode=[KWP2000ResponseCode, ManufacturerSpecific]	M	xx=[00-7F, 80-FF]	**RC**

10.3.3 Message description

The StopRoutineByLocalIdentifier service is used by the client to stop a routine in the server's memory referenced by a RoutineLocalIdentifier. The routine in the server shall be stopped after the positive response message has been sent.

The parameter RoutineExitOption is user optional and shall be of any type and length defined within KWP 2000.

Examples of RoutineExitOption are: TimeToExpireBeforeRoutineStops, variables, etc.

The parameter RoutineExitStatus is user optional and shall be of any type and length defined within KWP 2000.

Examples of RoutineExitStatus are: TotalRunTime, results generated by the routine before stopped, etc.

10.3.4 Message flow example

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

A complete message flow example is shown in 10.5.4.

10.4 StopRoutineByAddress service

10.4.1 Parameter definition

The parameter ***RoutineAddress*** is defined in 10.2.1.

The parameter ***RoutineExitOption*** is defined in 10.3.1.

The parameter ***RoutineExitStatus*** is defined in 10.3.1.

10.4.2 Message data bytes

Tables 120 to 122 describe the different messages for StopRoutineByAddress service.

Table 120 — StopRoutineByAddress Request Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**StopRoutineByAddress Request Service Id**	**M**	**39**	**SPRBAD**
2	RoutineAddress (High Byte)	M	xx	**MEMAHB**
3	RoutineAddress (Middle Byte)	M	xx	**MEMAMB**
4	RoutineAddress (Low Byte)	M	xx	**MEMALB**
5 : n	RoutineExitOption#1 : RoutineExitOption#m	U : U	xx : xx	**REXITOPT**

Table 121 — StopRoutineByAddress Positive Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**StopRoutineByAddress Positive Response Service Id**	**S**	**79**	**SPRBADPR**
2	RoutineAddress (High Byte)	M	xx	**MEMAHB**
3	RoutineAddress (Middle Byte)	M	xx	**MEMAMB**
4	RoutineAddress (Low Byte)	M	xx	**MEMALB**
5 : n	RoutineExitStatus#1 : RoutineExitStatus#m	U : U	xx : xx	**REXITSTAT**

Table 122 — StopRoutineByAddress Negative Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**NegativeResponse Service Id**	**S**	**7F**	**NACK**
2	**StopRoutineByAddress Request Service Id**	**M**	**39**	**SPRBAD**
3	ResponseCode=[KWP2000ResponseCode, ManufacturerSpecific]	M	xx=[00-7F, 80-FF]	**RC**

10.4.3 Message description

The StopRoutineByAddress service is used by the client to stop a routine in the server's memory referenced by a MemoryAddress. The routine in the server shall be stopped after the positive response message has been sent.

The parameter RoutineExitOption is user optional and shall be of any type and length defined within KWP 2000.

Examples of RoutineExitOption are: TimeToExpireBeforeRoutineStops, variables, etc.

The parameter RoutineExitStatus is user optional and shall be of any type and length defined within KWP 2000.

Examples of RoutineExitStatus are: TotalRunTime, results generated by the routine before stopped, etc.

10.4.4 Message flow example

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

A complete message flow example is shown in 10.6.4.

10.5 RequestRoutineResultsByLocalIdentifier service

10.5.1 Parameter definition

The parameter ***RoutineLocalIdentifier*** is defined in 10.1.1.

The parameter ***RoutineResults*** is used by the RequestRoutineResultsByLocalIdentifier and RequestRoutineResultsByAddress positive response message to provide the results (exit status information) of the routine which has been stopped previously in the server.

10.5.2 Message data bytes

Tables 123 to 125 describe the different messages for RequestRoutineResultsByLocalIdentifier service.

Table 123 — RequestRoutineResultsByLocalIdentifier Request Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**RequestRoutineResultsByLocalIdentifier Request SId**	**M**	**33**	**RRRBLID**
2	RoutineLocalIdentifier	M	xx	**RLOCID**

Table 124 — RequestRoutineResultsByLocalIdentifier Positive Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**RequestRoutineResultsByLocalIdentifier Pos. Resp. SId**	**S**	**73**	**RRRBLIDPR**
2	RoutineLocalIdentifier	M	xx	**RLOCID**
3 : n	RoutineResults#1 : RoutineResults#m	U : U	xx : xx	**RRESULT**

Table 125 — RequestRoutineResultsByLocalIdentifier Negative Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**NegativeResponse Service Id**	**S**	**7F**	**NACK**
2	**RequestRoutineResultsByLocalIdentifier Request SId**	**M**	**33**	**RRRBLID**
3	ResponseCode=[KWP2000ResponseCode, ManufacturerSpecific]	M	xx=[00-7F, 80-FF]	**RC**

10.5.3 Message description

The RequestRoutineResultsByLocalIdentifier service is used by the client to request results (e.g. exit status information) referenced by a RoutineLocalIdentifier and generated by the routine which was executed in the server's memory.

The parameter RoutineResults is user optional and shall be of any type and length defined within KWP 2000. Based on the RoutineResults which may have been received in the StopRoutineByLocalidentifier or StopRoutineByAddress positive response message (RoutineResults = normal/AbNormalExitWithResults)
the RequestRoutineResultsByLocalIdentifier service shall be used.

Example of RoutineResults - data collected by the ECU which could not be transmitted during routine execution because of ECU performance limitations.

10.5.4 Message flow example

Table 126 gives a message flow example of Start/StopRoutineByLocalIdentifier service.

Table 126 — Message flow example of Start/StopRoutineByLocalIdentifier service followed by a RequestRoutineResultsByLocalIdentifier service

time	client (tester)	server (ECU)
P3	StartRoutineByLocalId.Request[...]	
P2		StartRoutineByLocalId.PositiveResponse[...]
P3	StopRoutineByLocalId.Request[...]	
P2		StopRoutineByLocalId.PositiveResponse[...]
P3	RequestRoutineResultsByLocalId.Request[...]	
P2		RequestRoutineResultsByLocalId.PositiveResponse[...]

The RequestRoutineResultsByLocalIdentifier service shall only be used with physical addressing.

10.6 RequestRoutineResultsByAddress service

10.6.1 Parameter definition

The parameter ***RoutineAddress*** is defined in 10.2.1.

The parameter ***RoutineResults*** is defined in 10.5.1.

10.6.2 Message data bytes

Tables 127 to 129 describe the different messages for RequestRoutineResultsByAddress service.

Table 127 — RequestRoutineResultsByAddress Request Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**RequestRoutineResultsByAddress Request Service Id**	**M**	**3A**	**RRRBAD**
2	RoutineAddress (High Byte)	M	xx	**MEMAHB**
3	RoutineAddress (Middle Byte)	M	xx	**MEMAMB**
4	RoutineAddress (Low Byte)	M	xx	**MEMALB**

Table 128 — RequestRoutineResultsByAddress Positive Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**RequestRoutineResultsByAddress Positive Response**	**S**	**7A**	**RRRBAD**
2	RoutineAddress (High Byte)	M	xx	**MEMAHB**
3	RoutineAddress (Middle Byte)	M	xx	**MEMAMB**
4	RoutineAddress (Low Byte)	M	xx	**MEMALB**
5 : n	RoutineResults#1 : RoutineResults#m	U : U	xx : xx	**RRESULT**

Table 129 — RequestRoutineResultsByAddress Negative Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**Negative ResponseService Id**	**S**	**7F**	**NACK**
2	**RequestRoutineResultsByAddress Request Service Id**	**M**	**3A**	**RRRBAD**
3	ResponseCode=[KWP2000ResponseCode, ManufacturerSpecific]	M	xx=[00-7F, 80-FF]	**RC**

10.6.3 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 and shall be of any type and length defined within KWP 2000. Based on the RoutineResults which may have been received in the StopRoutineByLocalIdentifier or StopRoutineByAddress positive response message (RoutineResults = normal/AbNormalExitWithResults) the RequestRoutineResultsByAddress service shall be used.

Example of RoutineResults: data collected by the ECU which could not be transmitted during routine execution because of ECU performance limitations.

10.6.4 Message flow example

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

NOTE — The message flow example cited above uses a LocalIdentifier instead of a MemoryAddress.

11 Upload Download functional unit

The services provided by this functional unit are described in table 130.

Table 130 — Upload Download functional unit

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.

11.1 RequestDownload service

11.1.1 Parameter definition

The parameter ***TransferRequestParameter*** is used by all request messages of the Upload Download functional unit and shall include all the information necessary for the request messages. Format and length for this parameter is vehicle-manufacturer-specific.

The parameter ***TransferResponseParameter*** in the RequestDownload positive response message defines the status of the server if it is ready to receive data. One value is defined in table 131. Format and length of additional parameters are vehicle-manufacturer-specific.

Table 131 — Definition of TransferResponseParameter value

Hex	Description
00-43	**ReservedByDocument** This range of values is reserved by this document (refer to 4.4 for Response Codes).
44	**ReadyForDownload**
45-7F	**ReservedByDocument** This range of values is reserved by this document (refer to 4.4 for Response Codes).
80 - FF	**ManufacturerSpecificCodes** This range of values is reserved for vehicle-manufacturer-specific use.

11.1.2 Message data bytes

Tables 132 to 134 describe the different messages for RequestDownload service.

Table 132 — RequestDownload Request Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**RequestDownload Request Service Id**	**M**	**34**	**REQDWN**
2 : n	TransferRequestParameter#1 : TransferRequestParameter#m	U : U	xx : xx	**TRNFREQP**

Table 133 — RequestDownload Positive Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**RequestDownload Positive Response Service Id**	**M**	**74**	**REQDWNPR**
2 : n	TransferResponseParameter#1 : TransferResponseParameter#m	U : U	xx : xx	**TRNFRSPP**

Table 134 — RequestDownload Negative Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**NegativeResponse Service Id**	**S**	**7F**	**NACK**
2	**RequestDownload Request Service Id**	**M**	**34**	**REQDWN**
3	ResponseCode=[KWP2000ResponseCode, ManufacturerSpecific]	M	xx=[00-7F, 80-FF]	**RC**

11.1.3 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: initialization value(s) for data download. Example of TransferResponseParameter: maximum number of bytes per TransferData message.

11.1.4 Message flow example

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

A complete message flow example is shown in 11.4.4.

11.2 RequestUpload service

11.2.1 Parameter definition

The parameter ***TransferRequestParameter*** is defined in 11.1.1.

The parameter ***TransferResponseParameter*** in the RequestUpload positive response message defines the status of the server if it is ready to transmit data. One value is defined in table 135. Format and length of additional parameters are vehicle-manufacturer-specific.

Table 135 — Definition of TransferResponseParameter value

Hex	Description
00-53	**ReservedByDocument** This range of values is reserved by this document (refer to section 4.4 Response Codes).
54	**ReadyForUpload**
55-7F	**ReservedByDocument** This range of values is reserved by this document (refer to section 4.4 Response Codes).
80 - FF	**ManufacturerSpecificCodes** This range of values is reserved for vehicle-manufacturer-specific use.

11.2.2 Message data bytes

Tables 136 to 138 describe the different messages for RequestUpload service.

Table 136 — RequestUpload Request Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**RequestUpload Request Service Id**	**M**	**35**	**REQUP**
2 : n	TransferRequestParameter#1 : TransferRequestParameter#m	U : U	xx : xx	**TRNFREQP**

Table 137 — RequestUpload Positive Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**RequestUpload Positive Response Service Id**	**M**	**75**	**REQUPPR**
2 : n	TransferResponseParameter#1 : TransferResponseParameter#m	U : U	xx : xx	**TRNFRSPP**

Table 138 — RequestUpload Negative Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**NegativeResponse Service Id**	**S**	**7F**	**NACK**
2	**RequestUpload Request Service Id**	**M**	**35**	**REQUP**
3	ResponseCode=[KWP2000ResponseCode, ManufacturerSpecific]	M	xx=[00-7F, 80-FF]	**RC**

11.2.3 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: initialization value(s) for data upload.

Example of TransferResponseParameter: maximum number of bytes per TransferData message.

11.2.4 Message flow example

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

A complete message flow example is shown in 11.4.4.

11.3 TransferData service

11.3.1 Parameter definition

The parameter ***TransferRequestParameter*** 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. In addition, this part of ISO 14230 specifies one value which is described in table 139.

Table 139 — Definition of TransferRequestParameter value

Hex	Description
00-72	**ReservedByDocument** This range of values is reserved by this document (refer to 4.4 for Response Codes).
73	**BlockTransferComplete/NextBlock**
74-7F	**ReservedByDocument** This range of values is reserved by this document (refer to 4.4 for Response Codes).
80 - FF	**ManufacturerSpecificCodes** This range of values is reserved for vehicle-manufacturer-specific use.

The parameter ***TransferResponseParameter*** 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. In addition, this standard specifies one value which is described in table 140.

Table 140 — Definition of TransferResponseParameter value

Hex	Description
00-72	**ReservedByDocument** This range of values is reserved by this document (refer to 4.4 for Response Codes).
73	**BlockTransferComplete/NextBlock**
74-7F	**ReservedByDocument** This range of values is reserved by this document (refer to 4.4 for Response Codes).
80 - FF	**ManufacturerSpecificCodes** This range of values is reserved for vehicle-manufacturer-specific use.

11.3.2 Message data bytes

Tables 141 to 143 the different messages for TransferData service.

Table 141 — TransferData Request Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**TransferData Request Service Id**	**M**	**36**	**TRNDAT**
2 : n	TransferRequestParameter#1 : TransferRequestParameter#m	U : U	xx : xx	**TRNOPT**

Table 142 — TransferData Positive Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**TransferData Positive Response Service Id**	**M**	**76**	**TRNDATPR**
2 : n	TransferResponseParameter#1 : TransferResponseParameter#m	U : U	xx : xx	**TRNFRSPP**

Table 143 — TransferData Negative Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**NegativeResponse Service Id**	**S**	**7F**	**NACK**
2	**TransferData Request Service Id**	**M**	**36**	**TRNDAT**
3	ResponseCode=[KWP2000ResponseCode, ManufacturerSpecific]	M	xx=[00-7F, 80-FF]	**RC**

11.3.3 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.

If the client initiated a RequestDownload the data to be downloaded is 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 is 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 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.

For example, 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.

The TransferData service shall only be terminated by the RequestTransferExit service.

11.3.4 Message flow example

Table 144 gives a message flow example of TransferData Service.

Table 144 — Message flow example of TransferData service

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[...]

The TransferData service shall only be used with physical addressing.

11.4 RequestTransferExit service

11.4.1 Parameter definition

The parameter ***TransferRequestParameter*** is defined in 11.1.1.

The parameter ***TransferResponseParameter*** 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.

11.4.2 Message data bytes

Tables 145 to 147 describe the different messages for RequestTransferExit service.

Table 145 — RequestTransferExit Request Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**RequestTransferExit Request Service Id**	**M**	**37**	**RTRNEX**
2 : n	TransferRequestParameter#1 : TransferRequestParameter#m	U : U	xx : xx	**TRNFREQP**

Table 146 — RequestTransferExit Positive Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**RequestTransferExit Positive Response Service Id**	**M**	**77**	**RTRNEXPR**
2 : n	TransferResponseParameter#1 : TransferResponseParameter#m	U : U	xx : xx	**TRNFRSPP**

Table 147 — RequestTransferExit Negative Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**NegativeResponse Service Id**	**S**	**7F**	**NACK**
2	**RequestTransferExit Request Service Id**	**M**	**37**	**RTRNEX**
3	ResponseCode=[KWP2000ResponseCode, ManufacturerSpecific]	M	xx=[00-7F, 80-FF]	**RC**

11.4.3 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.

Example of TransferRequestParameter: checksum request of entire data transferred. Example of TransferResponseParameter: checksum of entire data transferred.

11.4.4 Message flow example

Table 148 gives a message flow example of flow example of RequestUpload service.

Table 148 — Message flow example of RequestUpload, multiple TransferData services followed by a RequestTransferExit service

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[...]

The RequestTransferExit service shall only be used with physical addressing.

12 Keyword Protocol 2000 extended service

12.1 EscapeCode service

12.1.1 Parameter definition

The parameter ***ManufacturerSpecific Service Identifier*** is defined as an extension of the service identifiers specified in this part of ISO 14230. The range of values is specified in table 149.

Table 149 — Definition of ManufacturerSpecific Service Identifier

Hex	Description
00 - FF	**ManufacturerSpecific** This range of values is reserved for vehicle-manufacturer-specific use.

The parameter ***RecordValue*** is defined in 7.1.1.

12.1.2 Message data bytes

Tables 150 to 152 describe the different messages for EscapeCode service.

Table 150 — EscapeCode Request Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**EscapeCode Request Service Id**	**M**	**80**	**ESCODE**
2	ManufacturerSpecific Service Id	M	xx	**MFSPSID**
3 : n	RecordValue#1 : RecordValue#m	U : U	xx : xx	**DATAVAL**

Table 151 — EscapeCode Positive Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**EscapeCode Positive Response Service Id**	**M**	**C0**	**ESCODEPR**
2	ManufacturerSpecific Service Id	M	xx	**MFSPSID**
3 : n	RecordValue#1 : RecordValue#m	U : U	xx : xx	**DATAVAL**

Table 152 — EscapeCode Negative Response Message

Data Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	**NegativeResponse Service Id**	**S**	**7F**	**NACK**
2	**EscapeCode Request Service Id**	**M**	**80**	**ESCODE**
3	ManufacturerSpecific Service Id	M	xx	**MFSPSID**
4	ResponseCode=[KWP2000ResponseCode, ManufacturerSpecific]	M	xx=[00-7F, 80-FF]	**RC**

12.1.3 Message description

The EscapeCode service is used by the client if none of the services specified in this part of ISO 14230 can provide the function required by the vehicle manufacturer.

12.1.4 Message flow example

Table 153 gives a message flow example of EscapeCode service.

Table 153 — Message flow example of EscapeCode service

time	client (tester)	server (ECU)
P3	EscapeCode.Request[...]	
P2		EscapeCode.PositiveResponse[...]
P3	EscapeCode.Request[...]	
P2		EscapeCode.NegativeResponse[...]
P3	EscapeCode.Request[...]	
P2		EscapeCode.PositiveResponse[...]

See also message flow examples of Physical Addressed Service in 5.3.1.1 and Functional Addressed Service in 5.3.2.1.

13 Application examples

This clause aims at showing with a real example how the Keyword Protocol 2000 services defined in this part of ISO 14230 may be used to establish diagnostic applications for diagnostic equipment and one or more on-board ECUs.

13.1 Description of on-vehicle ECUs

Tables 154 to 157 describe the configuration of the various ECUs on the considered vehicle for this example.

Table 154 — On-vehicle configuration

Description of On-Vehicle Configuration
Two ECUs connected: Engine Control Module & Transmission Control Module Multi Point Connection

Table 155 — ECU supported functions

List of communication functions supported by both ECUs
Fast Initialization Arbitration

Table 156 — Engine Control Module (ECM) features

ECM features
Re-programmable (Flash EPROM) Requires security login with Seed & Key

Table 157 — ECU supported services

ECU supported services	ECM	TCM	Security required
StartCommunication	YES	YES	NO
StopCommunication	YES	YES	NO
AccessCommunicationParameters (TPI: read, set)	YES	YES	NO
TesterPresent	YES	YES	NO
StartDiagnosticSession(RepairShop)	YES	YES	NO
StopDiagnosticSession	YES	YES	NO
ReadECUIdentification	YES	YES	NO
ReadDiagnosticTroubleCodes	YES	YES	NO
ClearDiagnosticInformation	YES	YES	NO
ReadDataByLocalIdentifer	YES	YES	NO
DynamicallyDefineLocalIdentifier	YES	YES	NO
SecurityAccess	YES	NO	NO
StartDiagnosticSession(Programming)	YES	NO	YES
RequestDownload	YES	NO	YES
TransferData	YES	NO	YES
StartRoutineByLocalIdentifier	YES	NO	YES
RequestTransferExit	YES	NO	YES

13.2 Functional initialization and functional addressed communication

Both ECUs shall be initialized with the fast initialization wake up pattern and StartCommunication request service.

A repair shop diagnostic session is started in both ECUs with a functional StartDiagnosticSession request service.

The client sends a functional ReadECUIdentification request service to retrieve all identification data of each ECU.

A functional ReadDiagnosticTroubleCodes request is used by the client to read the DTC's from the ECUs.

Table 158 shows this message flow.

Table 158 — Message flow of functional initialization and communication

Delay	Client (tester)	Server (ECU)
	Fast Initialization	
	StartComm.Req[] {TGT=$FE; Functional Init. }	
P2		StartComm.PosRsp[KB1,KB2] {ECM}1)
P2		StartComm.PosRsp[KB1,KB2] {TCM}1)
P3	StartDiagSess.Req[normalDiagSess] {TGT=$FE}	
P2		StartDiagSess.PosRsp[] {ECM}
P2		StartDiagSess.PosRsp[] {TCM}
P3	ReadECUId.Req[IdentificationOption] {TGT=$FE}	
P2		ReadECUId.PosRsp[IdentificationRecordValues] {TCM}
P2		ReadECUId.PosRsp[IdentificationRecordValues] {ECM}
P3	ReadDTC.Req[GroupOfDTC] {TGT=$FE}	
P2		ReadDTC.PosRsp[#DTC,DTC#1,DTC#2] {ECM}
P2		ReadDTC.PosRsp[#DTC,DTC#3] {TCM}
NOTE —In the example, both ECM and TCM ECUs respond to the StartCommunication request. It is also possible that only one of the ECU responds.		
1) Normal Timing is enabled by the keybytes of the ECM and TCM.		

13.3 Single and multiple response and termination of communication

The client reads specific data records identified by a RecordLocalIdentifier from each ECU individually by sending physical addressed ReadDataByLocalIdentifier requests.

Five data records, identified by another RecordLocalIdentifier, are requested from the ECM with the ReadDataByLocalIdentifier with the transmission mode parameter set to "fast" and maximum number of responses to send set to "5".

After above tests are finished the tester clears all diagnostic information by sending a functional ClearDiagnosticInformation request service.

Then the diagnostic session of the TCM is finished.

The communication with the TCM is stopped by a functional StopCommunication request service.

Table 159 shows this message flow.

Table 159 — Message flow of single and multiple response messages

time	client (tester)	server (ECU)
P3	ReadDataByLocId.Req[RecLocId=10] {TGT=ECM}	
P2		ReadDataByLocId.PosRsp[10,RecordValues] {ECM}
P3	ReadDataByLocId.Req[RecLocId=03] {TGT=TCM}	
P2		ReadDataByLocId.PosRsp[03,RecordValues] {TCM}
P3	ReadDataByLocId.Req[RecLocId=01,TXM=fast, MAX#ORTS=5] {TGT=ECM}	
P2		ReadDataByLocId.PosRsp[01,RecordValues] {ECM}
P2		ReadDataByLocId.PosRsp[01,RecordValues] {ECM}
P2		ReadDataByLocId.PosRsp[01,RecordValues] {ECM}
P2		ReadDataByLocId.PosRsp[01,RecordValues] {ECM}
P2		ReadDataByLocId.PosRsp[01,RecordValues] {ECM}
P3	ClearDiagInfo.Req[GroupOfDTC] {TGT=$FE}	
P2		ClearDiagnosticInformation.PosRsp[] {ECM}
P2		ClearDiagnosticInformation.PosRsp[] {TCM}
P3	StopDiagnosticSession.Req[] {TGT=TCM}	
P2		StopDiagnosticSession.PosRsp[] {TCM}
P3	StopCommunication.Req[] {TGT=TCM}	
P2		StopCommunication.PosRsp[] {TCM}

13.4 SecurityAccess, data transfer and modification of timing parameters

Now the tester starts preparing for reprogramming the ECM's memory by using the following services.

The tester sends a SecurityAccess request with the AccessMode set to RequestForSeed to the ECM. The ECM responds with the parameter Seed.

The tester manipulates the Seed into a Key value and sends it to the ECM with the SecurityAccess request with the AccessMode set to SendKey. The ECM denies the request by sending a negative response with the ResponseCode set to SecurityAccessDenied (tester has sent the wrong key). The ECM starts a 10 s timer which shall have expired before the next SecurityAccess is allowed.

Now it is the client's responsibility to keep the communication alive. Therefore the tester shall continuously send TesterPresent messages to the ECM with the parameter ResponseRequired set to YES. The number of TesterPresent services required depends on the timing parameter P3max. The tester stops sending TesterPresent requests as soon as the 10 s timer has expired. A new request is sent to the server.

Table 160 shows this message flow.

Table 160 — Message flow of SecurityAccess

time	Client (tester)	Server (ECU)
P3 P2	SecurityAcc.Req[RequestSeed] {TGT=ECM}	 SecurityAcc.PosRsp[ResponseWithSeed,Seed] {ECM}
P3 P2	SecurityAcc.Req[SendKey] {TGT=ECM}	SecurityAcc.NegRsp[SecurityAccessDenied] {ECM} 1)

P3
P2
P3
P2
:
P3
P2
	TesterPresent.Req#1[Yes] {TGT=ECM}
	
	TesterPresent.Req#2[Yes] {TGT=ECM}
	
	:
	TesterPresent.Req#n[Yes] {TGT=ECM}
	
		TesterPresent.PosRsp#1[] {ECM}
		
		TesterPresent.PosRsp#2[] {ECM}
		:
		
		TesterPresent.PosRsp#n[] {ECM}

1) Tester has sent the wrong key.10 s timer starts. 2) 10 s timer expires.		

The tester repeats the entire login procedure. It finally received a positive response with the parameter SecurityAccessAllowed. Now the ECM is ready to execute diagnostic sessions which require a security login.

The tester starts a programming session in the ECM which enables a new set of services which were not available in the RepairShop diagnostic mode. To speed up the upcoming data transfer, the tester changes the timing parameters based on the ECMs communication capabilities. Therefore it sends the AccessCommunication-Parameter request service with the TimingParameterIndex set to ReadLimits. The ECM responses with the internally stored timing parameters "P2 - P4" used for fast data transfer and memory re-programming.

The tester sends another AccessCommunicationParameter request service with the TimingParameterIndex set to setLimits. The timing becomes active with the next request of the tester.

Table 161 shows this message flow.

Table 161 — Message flow of SecurityAccess, StartDiagnosticSession and AccessCommunicationParameters

time	Client (tester)	Server (ECU)

P3
P2
P3
P2
	SecurityAcc.Req[RequestSeed] {TGT=ECM}
	
	SecurityAcc.Req[SendKey] {TGT=ECM}
	
		SecurityAcc.PosRsp[ResponseWithSeed,Seed] {ECM}
		
		SecurityAcc.PosRsp[SendKey,SecurityAccAllwd] {ECM}

P3 P2	StartDiagSess.Req[ProgSession] {TGT=ECM}	 StartDiagSess.PosRsp[] {ECM}
P3 P2	AccCommPara.Req[ReadLimits] {TGT=ECM}	 AccCommPara.PosRsp[ReadLimits,P2-P4] {ECM} 1)
P3 P2	AccCommPara.Req[setPara,P2 - P4] {TGT=ECM}	 AccCommPara.PosRsp[setPara] {ECM}
1) ECM sends fast programming P2...P4 timing values. 2) Fast programming timing P2...P4 is enabled.		

The tester sends a RequestDownload request with an TransferRequestParameter parameters (e.g. memory start address, memory size, etc.). The positive response message of the ECM indicates to the tester that the ECM has accepted the request and is ready for receiving download data.

The TransferData service is used to download a software routine from the tester to the ECM.

The tester now sends the StartRoutineByLocalIdentifier request message to start the routine which has been downloaded before. This routine re-organises the RAM of the ECM in order to free up additional memory space to store further downloaded data.

The tester uses multiple TransferData services to download all FLASH EEPROM data to be re-programmed.

When finished the tester terminates the data transfer by sending a RequestTransferExit request message. Now the tester stops the current diagnostic session and finally, the communication is terminated by the StopCommunication service.

Table 162 shows this message flow.

Table 162 — Message flow of TransferData and StopCommunication

time	Client (tester)	Server (ECU)
P3	RequestDownload.Req[TRNFREQP] {TGT=ECM}	
P2		RequestDownload.PosRsp[TRNFRSPP] {ECM}
P3	TransferData.Req[TRNFREQP,...] {TGT=ECM}	
P2		TransferData.PosRsp[TRNFRSPP] {ECM}
P3	StartRoutineByLocId.Req[RLOCID,RENTOPT...] {TGT=ECM}	
P2		StartRoutineByLocId.PosRsp[RLOCID,RENTSTAT] {ECM}
P3	TransferData.Req[TRNFREQP,...] {TGT=ECM}	
P2		TransferData.PosRsp[TRNFRSPP] {ECM}
P3	TransferData.Req[TRNFREQP,...] {TGT=ECM}	
P2		TransferData.PosRsp[TRNFRSPP] {ECM}
:	:	:
P3	TransferData.Req[TRNFREQP,...] {TGT=ECM}	
P2		TransferData.PosRsp[TRNFRSPP] {ECM}
P3	RequestTransferExit.Req[TRNFREQP] {TGT=ECM}	
P2		RequestTransferExit.PosRsp[TRNFRSPP] {ECM}
P3	StopDiagnosticSession.Req[] {TGT=$FE}	
P2		StopDiagnosticSession.PosRsp[] {ECM}
P3	StopCommunication.Req[] {TGT=$FE}	
P2		StopCommunication.PosRsp[] {ECM}

13.5 ReadDataByLocalIdentifier service with DynamicallyDefineLocalIdentifier

13.5.1 Tester display parameter selection

While diagnosing an ECM (engine control module) the service technician gets a large list of parameters displayed on the tester screen. The technician selects the following four parameters out of the list which are of interest to him:

— throttle position sensor;

— idle air control;

— engine coolant temperature sensor;

— engine speed.

The technician selects these parameters to be displayed in the order as indicated in table 163.

Table 163 — Parameter order in tester display

Display position	Parameter description
1	Engine Coolant Temperature Sensor
2	Engine Speed
3	Throttle Position Sensor
4	Idle Air Control

13.5.2 ECM parameter definition

Within the ECM the parameters given in 13.5.1 are stored as indicated in table 164.

Table 164 — Parameter definition and source in ECM

Parameter description	Definition mode	Location in ECM	Memory size
Engine Coolant Temperature Sensor	DefineByLocalIdentifier = $01	position $07 in record identified by RecordLocalIdentifier $3A	2 bytes
Engine Speed	DefineByCommonIdentifier = $02	position $03 in record identified by RecordCommonIdentifier $B754	1 byte
Throttle Position Sensor	DefineByMemoryAddress = $03	start address $01DE05	3 bytes
Idle Air Control	DefineByLocalIdentifier = $01	position $02 in record identified by RecordLocalIdentifier $D2	1 byte

Within KWP 2000 the DynamicallyDefineLocalIdentifier service is used to assign a LocalIdentifier to these four parameters which are afterwards read by the ReadDataByLocalIdentifier service.

The tester defines the value "$25" for this LocalIdentifier which identifies a new record consisting of the four parameters selected by the service technician.

There are several possibilities for the tester to define this new record by using the DynamicallyDefineLocalIdentifier service. Two examples are given in 13.5.3 and 13.5.4.

13.5.3 Definition by single request message

The tester uses only one DynamicallyDefineLocalIdentifier request message to define the entire record. This message has the content given in table 165.

Table 165 — DynamicallyDefineLocalIdentifier request message#1

Data Byte	Parameter Name	Hex Value	Mnemonic
1	**DynamicallyDefineLocalIdentifier Request Service Id**	**2C**	**DLIDDY**
2	DynamicallyDefinedLocalIdentifier	25	**DDLOCID**
3	DefinitionMode=[DefineByLocalIdentifier]	01	**DEFMODE**
4	PositionInDynamicallyDefinedLocalIdentifier	03	**PIDYDLID**
5	MemorySize	02	**MEMSIZE**
6	RecordLocalIdentifier	3A	**RLOCID**
7	PositionInRecordLocalIdentifier	07	**PIRLOCID**
8	DefinitionMode=[DefineByCommonIdentifier]	02	**DEFMODE**
9	PositionInDynamicallyDefinedLocalIdentifier	01	**PIDYDLID**
10	MemorySize	01	**MEMSIZE**
11	RecordCommonIdentifier (High Byte)	B7	**RCIDHB**
12	RecordCommonIdentifier (Low Byte)	54	**RCIDLB**
13	PositionInRecordCommonIdentifier	03	**PIRLOCID**
14	DefinitionMode=[DefineByMemoryAddress]	03	**DEFMODE**
15	PositionInDynamicallyDefinedLocalIdentifier	02	**PIDYDLID**
16	MemorySize	02	**MEMSIZE**
17	MemoryAddress (High Byte)	01	**MEMAHB**
18	MemoryAddress (Middle Byte)	DE	**MEMAMB**
19	MemoryAddress (Low Byte)	05	**MEMALB**
20	DefinitionMode=[DefineByLocalIdentifier]	01	**DEFMODE**
21	PositionInDynamicallyDefinedLocalIdentifier	04	**PIDYDLID**
22	MemorySize	01	**MEMSIZE**
23	RecordLocalIdentifier	D2	**RLOCID**
24	PositionInRecordLocalIdentifier	02	**PIRLOCID**

13.5.4 Definition by multiple request messages

The tester for example uses three DynamicallyDefineLocalIdentifier request messages to define the record. The messages have the content given in tables 166 to 168.

Table 166 — DynamicallyDefineLocalIdentifier request message#1

Data Byte	Parameter Name	Hex Value	Mnemonic
1	**DynamicallyDefineLocalIdentifier Request Service Id**	**2C**	**DLIDDY**
2	DynamicallyDefinedLocalIdentifier	25	**DDLOCID**
3	DefinitionMode=[DefineByLocalIdentifier]	01	**DEFMODE**
4	PositionInDynamicallyDefinedLocalIdentifier	03	**PIDYDLID**
5	MemorySize	02	**MEMSIZE**
6	RecordLocalIdentifier	3A	**RLOCID**
7	PositionInRecordLocalIdentifier	07	**PIRLOCID**
8	DefinitionMode=[DefineByLocalIdentifier]	01	**DEFMODE**
9	PositionInDynamicallyDefinedLocalIdentifier	04	**PIDYDLID**
10	MemorySize	01	**MEMSIZE**
11	RecordLocalIdentifier	D2	**RLOCID**
12	PositionInRecordLocalIdentifier	02	**PIRLOCID**

Table 167 — DynamicallyDefineLocalIdentifier request message#2

Data Byte	Parameter Name	Hex Value	Mnemonic
1	**DynamicallyDefineLocalIdentifier Request Service Id**	**2C**	**DLIDDY**
2	DynamicallyDefinedLocalIdentifier	25	**DDLOCID**
3	DefinitionMode=[DefineByCommonIdentifier]	02	**DEFMODE**
4	PositionInDynamicallyDefinedLocalIdentifier	01	**PIDYDLID**
5	MemorySize	01	**MEMSIZE**
6	RecordCommonIdentifier (High Byte)	B7	**RCIDHB**
7	RecordCommonIdentifier (Low Byte)	54	**RCIDLB**
8	PositionInRecordCommonIdentifier	03	**PIRLOCID**

Table 168 — DynamicallyDefineLocalIdentifier request message#3

Data Byte	Parameter Name	Hex Value	Mnemonic
1	**DynamicallyDefineLocalIdentifier Request Service Id**	**2C**	**DLIDDY**
2	DynamicallyDefinedLocalIdentifier	25	**DDLOCID**
3	DefinitionMode=[DefineByMemoryAddress]	03	**DEFMODE**
4	PositionInDynamicallyDefinedLocalIdentifier	02	**PIDYDLID**
5	MemorySize	02	**MEMSIZE**
6	MemoryAddress (High Byte)	01	**MEMAHB**
7	MemoryAddress (Middle Byte)	DE	**MEMAMB**
8	MemoryAddress (Low Byte)	05	**MEMALB**

13.5.5 Reading the data record

After defining the new record referenced by the DynamicallyDefinedLocalIdentifier "$25" the tester sends the ReadDataByLocalIdentifier request message to read the values of the four parameters. The values shall only be updated on the tester display upon request by the service technician. Therefore the tester sets the parameter TransmissionMode to "single"; see table 169.

Table 169 — ReadDataByLocalIdentifier request message

Data Byte	Parameter Name	Hex Value	Mnemonic
1	**ReadDataByLocalIdentifier Request Service Id**	**21**	**RDBLID**
2	RecordLocalIdentifier	25	**RLOCID**
3	TransmissionMode=[single]	01	**TXM**

After receiving the ReadDataByLoacalIdentifier request message the ECM sends the ReadDataByLocalIdentifier positive response message with the content given in table 170.

Table 170 — ReadDataByLocalIdentifier request message

Data Byte	Parameter Name	Hex Value	Mnemonic
1	**ReadDataByLocalIdentifier Positive Response Service Id**	**61**	**RDBLIDPR**
2	RecordLocalIdentifier	25	**RLOCID**
3	Engine Speed (High Byte)	xx	**RECVAL**
4	Engine Speed (Low Byte)	xx	**RECVAL**
5	Throttle Position Sensor (High Byte)	xx	**RECVAL**
6	Throttle Position Sensor (Low Byte)	xx	**RECVAL**
7	Engine Coolant Temperature Sensor	xx	**RECVAL**
8	Idle Air Control	xx	**RECVAL**

Annex A
(informative)

Bibliography

[1] ISO 4092: 1988, *Road vehicles — Diagnostic systems for motor vehicles — Vocabulary*.

[2] ISO/IEC 7498-1:1994, *Information technology — Open Systems Interconnection —Basic Reference Model: The basic model.*

[3] ISO/IEC 10731:1994, *Information technology — Open Systems Interconnection — Basic Reference Model — Conventions for the definition of OSI services*.

[4] SAE J 1587: 1996, *Electronic data interchange between microcomputer systems in heavy-duty vehicle applications.* (Joint SAE/TMC publication)

[5] SAE J 1962: 1995, *Diagnostic connector*.

[6] SAE J 1978: 1994, *OBD II scan tool*.

[7] SAE J 2012: 1996, *Recommended practice for diagnostic trouble code definitions.* [Pratique recommandée pour les définitions des codes d'erreur de diagnostic]

[8] SAE J 2186: 1996, *E/E data link security*.

[9] SAE J 2190: 1993, *Enhanced E/E diagnostic test modes.*

ICS 43.180

Descriptors: road vehicles, motor vehicles, electronic equipment, electronic control units, diagnostic systems, digital technics, information interchange, protocols, application layer.

Price based on 81 pages