Keyword Protocol 2000
Data Link Layer
Recommended Practice

Status: **Version 1.5**

Date: **October 1, 1997**

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

- **Adam Opel AG**
- **AISIN AW CO., Limited Japan**
- **Audi AG / Volkswagen AG**
- **BMW AG**
- **Daimler-Benz AG**
- **debis Systemhaus GmbH**
- **DELCO Electronics Europe**
- **DSA Daten und Systemtechnik GmbH**
- **ETAS GmbH & Co. KG**
- **FEV Motorentechnik GmbH & Co. KG**
- **GenRad Europe Ltd.**
- **GM Europe GmbH Service Technology Group Int'l Operations**
- **Hella KG**
- **Isuzu Motors Ltd.**
- **Kelsey-Hayes**
- **LucasVarity**
- **MAN Nutzfahrzeuge AG**
- **Mecel AB**
- **Robert Bosch GmbH**
- **Saab Automobile AB**
- **Siemens AG**
- **Softing GmbH**
- **VDO Adolf Schindling AG**

THIS PAGE INTENTIONALLY LEFT BLANK

Table of contents

Introduction

This document is based on the ISO 14230-2 International Standard. Changes are indicated by changing the font from "Arial" to "Times New Roman"! In addition, the section “Definitions and Abbreviations” and the section “Conventions” of the Keyword Protocol 2000 Implementation of Diagnostic Services Recommended Practice document fully applies to this Recommended Practice document.

The Keyword Protocol 2000 Data Link Layer Recommended Practice is based on the International Standard and has been established in order to define common requirements for the communication of diagnostic services for diagnostic systems. The figure below shows the "Hierarchy of Documents" and indicates the level of commonality between the documents.

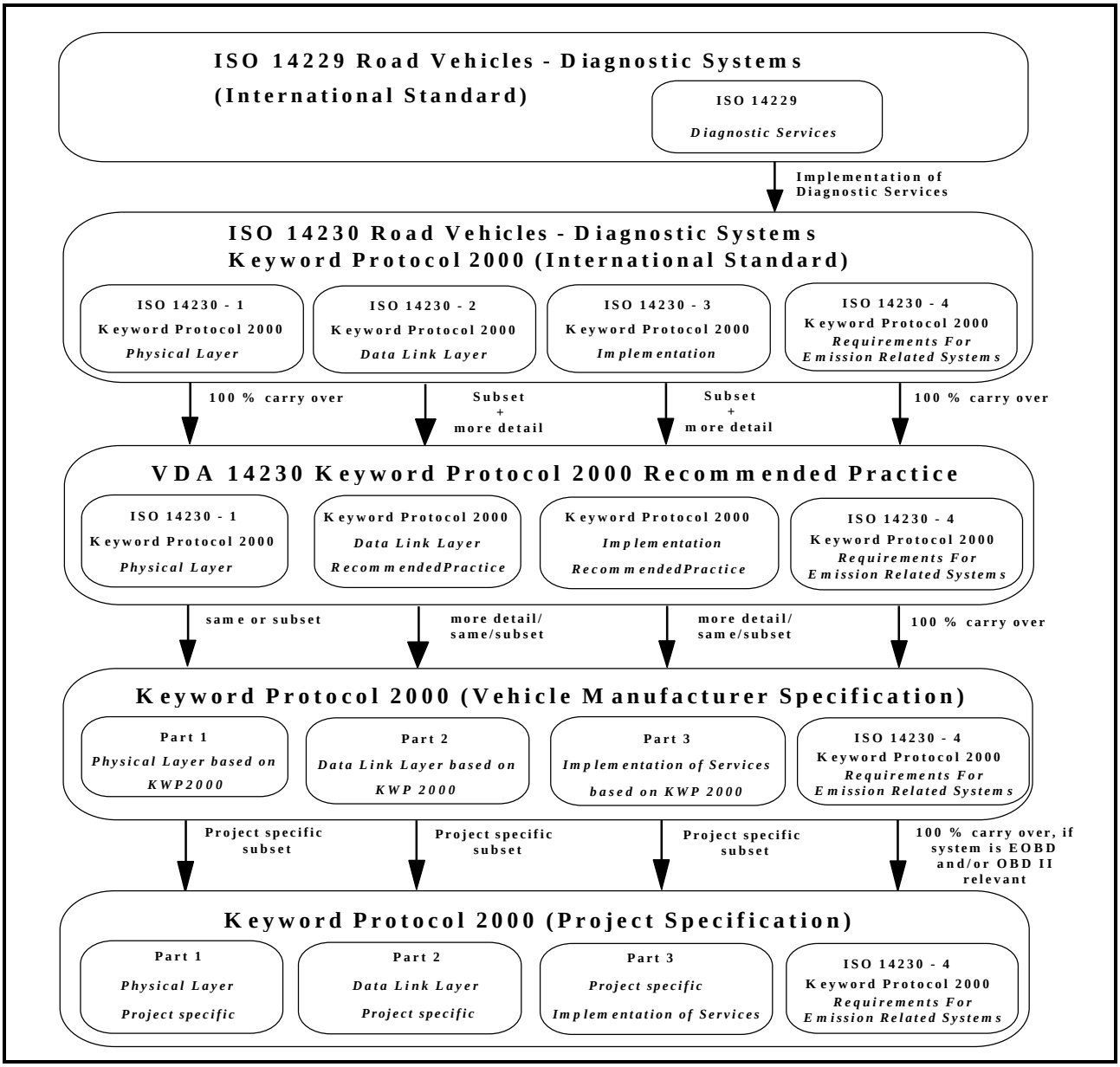


Figure 0 - Hierarchy of Documents

- The ISO 14229 Diagnostic Services (International Standard) document specifies common requirements for the implementation of diagnostic services.

- The ISO 14230 Keyword Protocol 2000 (International Standard) documents specify the requirements of the "Physical Layer, Data Link Layer and Implementation of Diagnostic Services" in three (3) different documents.

- The Keyword Protocol 2000 Recommended Practice documents are based on the International Standard ISO 14230 and therefore fully compatible. The Keyword Protocol 2000 - Physical Layer, Data Link Layer, and the Implementation documents are a "subset with additional detail based on mutual agreement between all companies listed on the cover sheet of this document".

- The Keyword Protocol 2000 (Vehicle Manufacturer Specification) document is based on the KWP 2000 Recommended Practice document. The Part 1 Physical Layer, Part 2 Data Link Layer, and the Part 3 Implementation of Services documents shall specify more detail or the same or a subset of the KWP 2000 Recommended Practice.

- The Keyword Protocol 2000 (Project Specification) document (e.g. Engine Management) is based on the vehicle manufacturer specification document. The document of the system supplier specifies system and customer specific Part 1 Physical Layer, Part 2 Data Link Layer and Part 3 Implementation details (e.g. data streams, diagnostic trouble codes, input/output controls, etc.).

The KWP 2000 Data Link Layer Recommended Practice is based on the Open System Interconnection (O.S.I.) 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 client (tester) and an Electronic Control Unit (ECU) are broken into:

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

Mapping of the Diagnostic Services and Keyword Protocol 2000 onto the O.S.I. model is shown below. ISO 14230 will consist of the following parts, under the general title Road vehicles - Diagnostic systems - Keyword Protocol 2000:

- Part 1: Physical Layer
- Part 2: Data Link Layer, Error Handling, Byte Arbitration
- Part 3: Implementation of Diagnostic Services

See figure below.

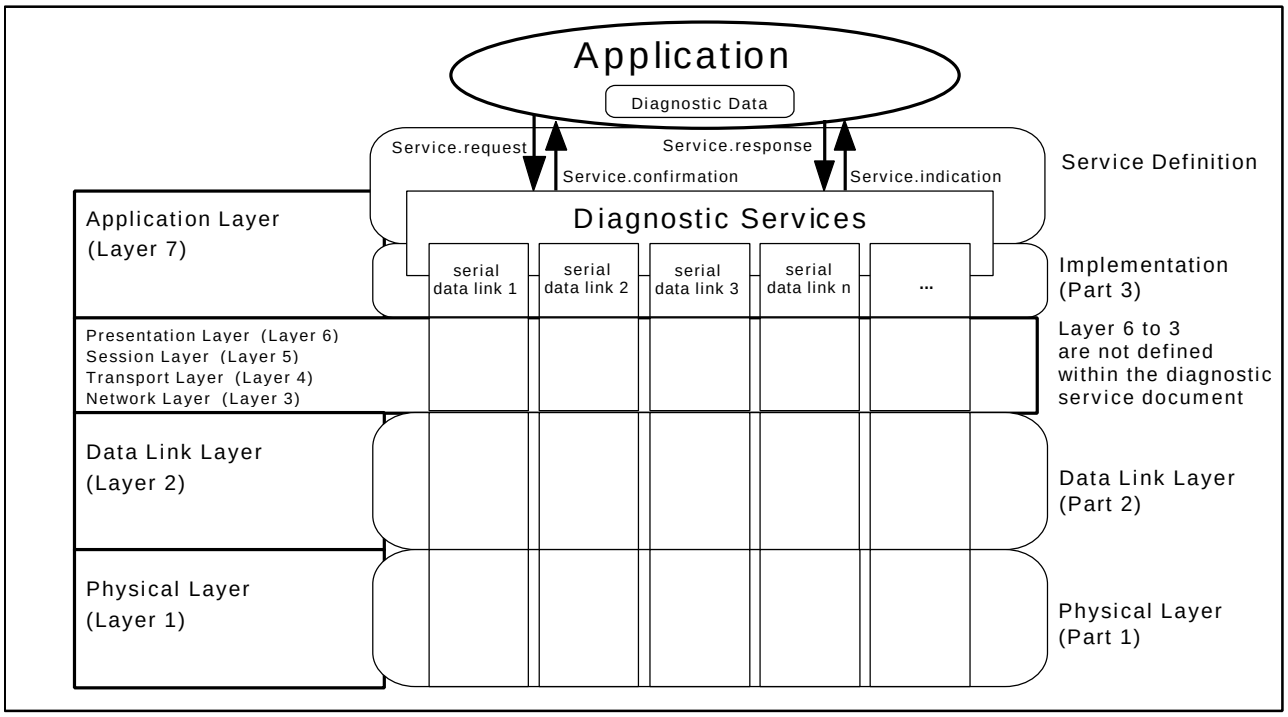


Figure 1 - Mapping of the diagnostic services on the O.S.I. Model

Example of serial data links: KWP 2000, VAN, CAN, J1850...

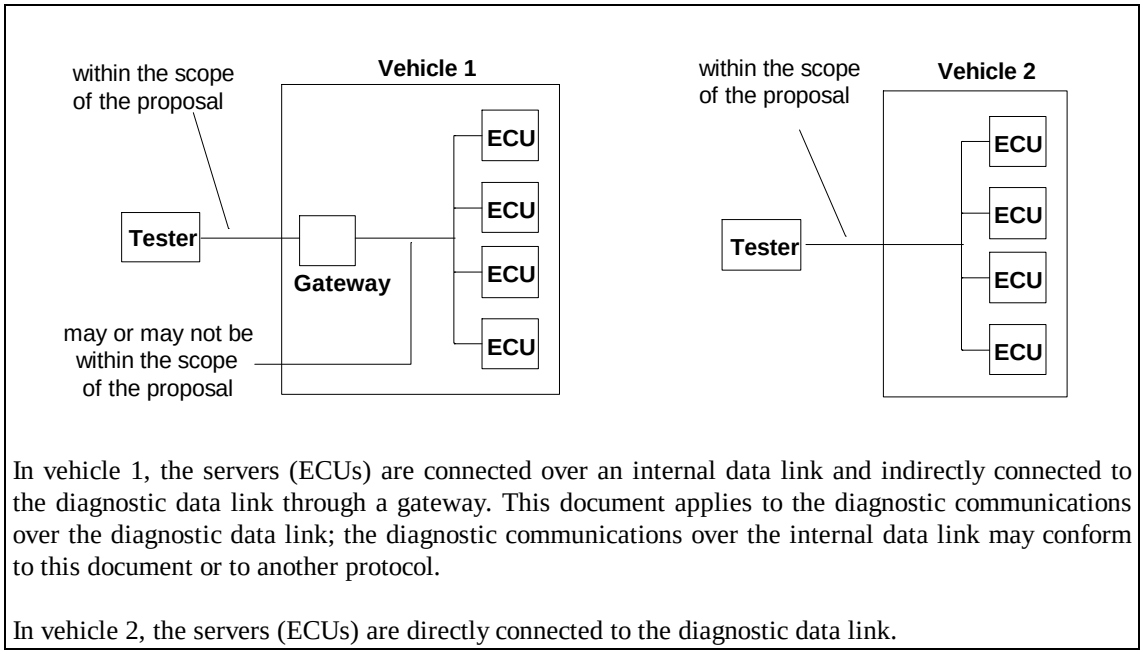
1 - Scope

This International Standard specifies common requirements of diagnostic services which allow a client (tester) to control diagnostic functions in an on-vehicle Electronic Control Unit (e.g. Electronic Fuel Injection, Automatic Gear Box, Anti-lock Braking System, ...) connected to a serial data link embedded in a road vehicle.
It specifies only layer 2 (Data Link Layer). Included are all definitions which are necessary to implement the services (described in "Keyword Protocol 2000 - Part 3: Implementation Recommended Practice") on a serial link (described in "Keyword Protocol 2000 - Part 1: Physical Layer"). Also included are some communication services which are needed for communication/session management and a description of error handling.
This Standard does not specify the requirements for the implementation of diagnostic services.
The physical layer may be used as a multi-user-bus, so a kind of arbitration or bus management is necessary. There are several proposals which are not part of this document. The car manufacturers are responsible for the correct working of bus management.
Communication between server (ECU)s are not part of this document.

The vehicle diagnostic architecture of this standard applies to:

- a single client (tester) that may be temporarily or permanently connected to the on-vehicle diagnostic data link and
- several on-vehicle electronic control units connected directly or indirectly

See figure below.



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

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

Figure 2 - Vehicle diagnostic architecture

2 - Normative reference

2.1 - ISO standards

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

ISO 7498-1:1984	Information processing systems - Open systems interconnection - Basic reference model.
ISO TR 8509:1987	Information processing systems - Open systems interconnection - Conventions of services.
ISO 4092:1988/Cor.1:1991	Road vehicles - Testers for motor vehicles - Vocabulary Technical Corrigendum 1. SAE J2012 Diagnostic Trouble Codes
ISO 9141:1989	Road vehicles - Diagnostic systems - Requirements for interchange of digital information
ISO 9141-2:1994	Road vehicles - Diagnostic systems- Part 2: CARB requirements for interchange of digital information
ISO/DIS 14229:1996	Road Vehicles - Diagnostic systems - Diagnostic Services Specification
ISO/DIS 14230-1:1996	Road Vehicles - Diagnostic systems - Keyword Protocol 2000 - Part 1: Physical Layer
ISO/DIS 14230-3:1996	Road Vehicles - Diagnostic systems - Keyword Protocol 2000 - Part 3: Implementation
ISO/WD 14230-4:1996	Road Vehicles - Diagnostic systems - Keyword Protocol 2000 - Part 4: Requirements For Emission Related Systems

Note: All documents which are of the status “Draft International Standard” are marked “DIS”!
All documents which are of the status “Working Draft” are marked “WD”!

2.2 - Other standards

KWP 2000 Implementation	Keyword Protocol 2000 Implementation Recommended Practice
SAE J1979: Dec,1991	E/E Diagnostic Test Modes
SAE J2012: 1994	Recommended Format and Messages for Diagnostic Trouble Codes
SAE J2178 :June, 1993	Class B Data Communication Network Messages

3 - Physical topology

Keyword Protocol 2000 is a bus concept (see diagram below). The figure below shows the general form of this serial link.

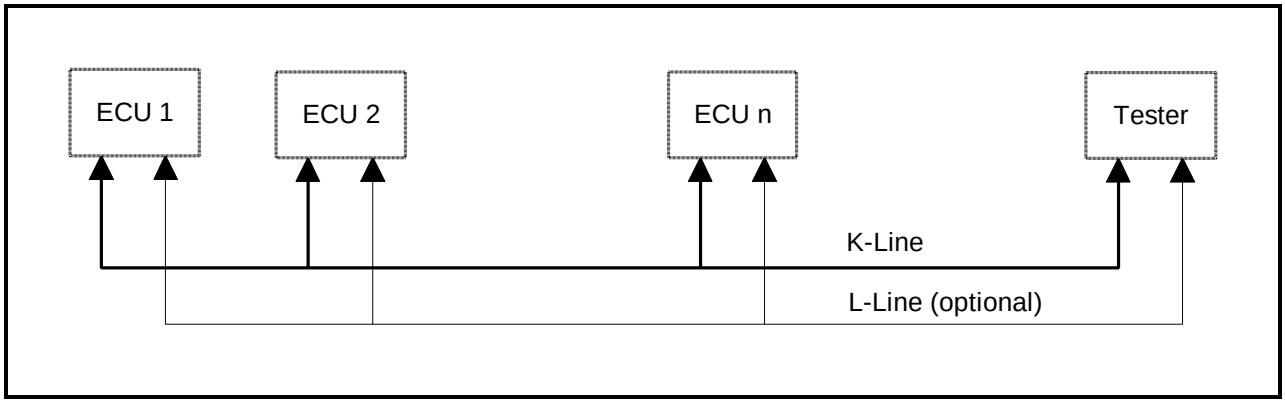


Figure 3 - Topology

"K - Line" is used for communication and initialisation, "L-Line" (optional) is used for initialisation only. Special cases are node-to-node-connection, that means only one server (ECU) on the line which also can be a bus converter.

KWP 2000 Data Link Layer Recommended Practice addition:
It is recommended to no longer support the L-Line in server (ECU) hardware.
Client (tester) hardware shall support the L-Line if compliance to SAE J1978 OBD II Scan Tool is required.
Please refer to "ISO 14230 KWP 2000 Part 1 - Physical Layer section 4 - Allowed Configurations" for more detail.

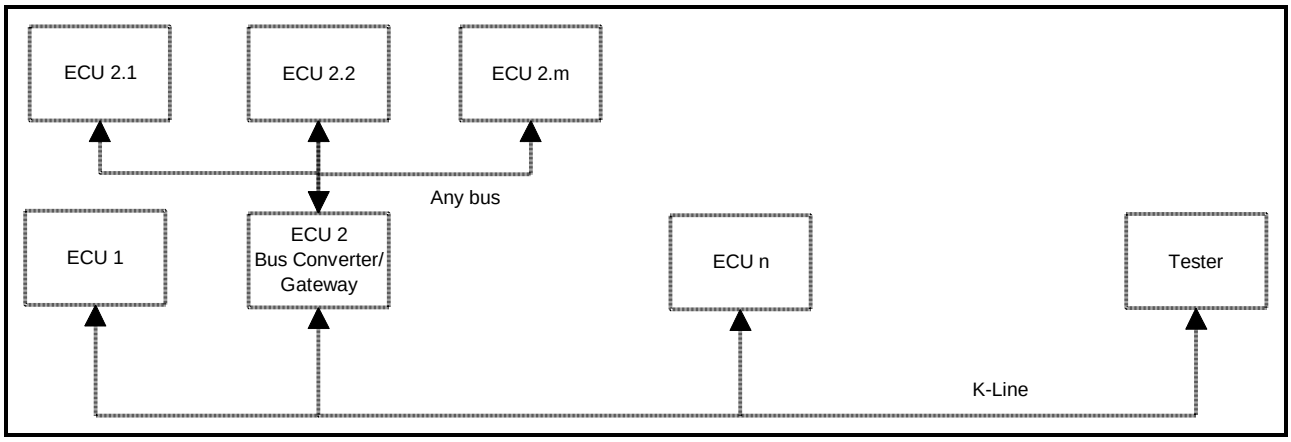


Figure 4 - Gateway topology example

Above figure shows an example of multiple servers (ECUs) connected with the K-Line to the client (tester). ECU 2 functions as a gateway (bus converter) operating on a bus system (e.g. CAN, SAE J1850).

4 - Message structure

This section describes the structure of a message.
The message structure consists of three parts:

- header
- data bytes
- checksum

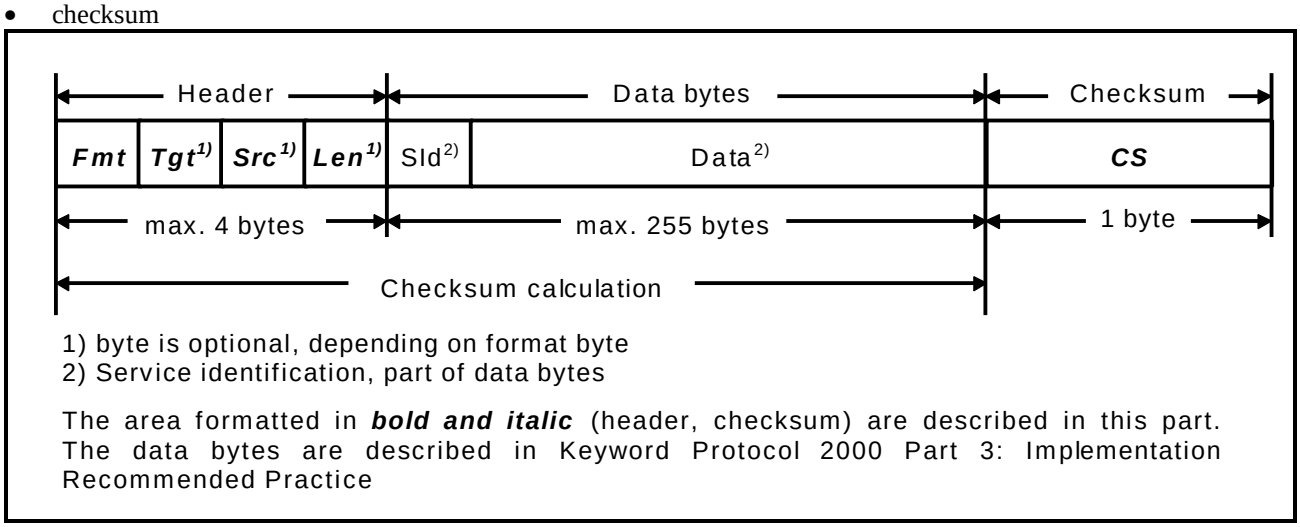


Figure 5 - Message block structure

4.1 - Header

The header consists of maximum of 4 bytes. A format byte includes information about the form of the message. The target and source address byte are optional for use with multi node or single node connections. The optional separate length byte allows message lengths up to 255 bytes.

4.1.1 - Format byte

The format byte contains a 6 bit length information and a 2 bit address mode information. The client (tester) is informed about use of the header bytes by the key bytes (see 5.1.5.1)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
		L5	L4	L3	L2	L1	L0
		Bits L0 - L5 define the length of the message					
		Define the length of a message from the beginning of the data field (Service Identification byte included) to checksum byte (not included). A message length of 1 to 63 bytes is possible. If L0 to L5 = 0 then the additional length byte is included.					
A1	**A0**	**A1 and A0 define the form of the header which will be used by the message.**					
0	0	no address information					
0	1	Exception mode (CARB). The CARB mode is not specified in this document. CARB uses format bytes $68 (0110 1000) and $48 (0100 1000). For more details refer to ISO 9141-2 and SAE J1979 documents.					
1	0	with address information, physical addressing					
1	1	with address information, functional addressing					

Table 4.1.1 - Format byte structure

4.1.2 - Target address byte

This is the target address for the message and is always used together with the source address byte. It may be a physical or a functional address. Physical addresses may be the 5 baud address byte (see ISO 9141:1989) or addresses according to SAE J2178 Part 1 (see appendix A). Functional addresses are listed in appendix B. The target address byte is optional and only necessary on multi node bus topologies. For node-to-node connections it may be omitted. For emission related (CARB) messages this byte is defined in ISO 14230 KWP 2000 Part 4: Requirements For Emission Related Systems.

4.1.3 - Source address byte

This is the address of the transmitting device. It must be a physical address. There are the same possibilities for the values as described for physical target address bytes. Addresses for client (tester)s are listed in SAE J2178 Part 1 (see appendix B). This byte is optional (always used together with the target address byte) and only necessary on multi node bus topologies. For node-to-node connections it may be omitted.

4.1.4 - Length byte

This byte is provided if the length in the header byte (L0 to L5) is set to 0. It allows the user to transmit messages with data fields longer then 63 bytes. With shorter messages it may be omitted. This byte defines the length of a message from the beginning of the data field (service identification byte included) to checksum byte (not included). A data length of 1 to 255 bytes is possible. The longest message consists of a maximum of 260 byte. For messages with data fields of less than 64 bytes there are two possibilities: Length may be included in the format byte or in the additional length byte. An server (ECU) needs not to support both possibilities, the client (tester) is informed about the capability of an server (ECU) through the key bytes (see section 5.1.5.1).

Length	**Length provided in**	
	Format byte	**Length byte**
< 64	XX00 0000	present
< 64	XXLL LLLL	not present
>= 64	XX00 0000	present

Table 4.1.4 - Format byte structure

XX: 2 bit address mode information
LL LLLL: 6 bit length information

4.1.5 - Use of header bytes

With the above definitions there are four different forms of message. These are shown diagramatically below.

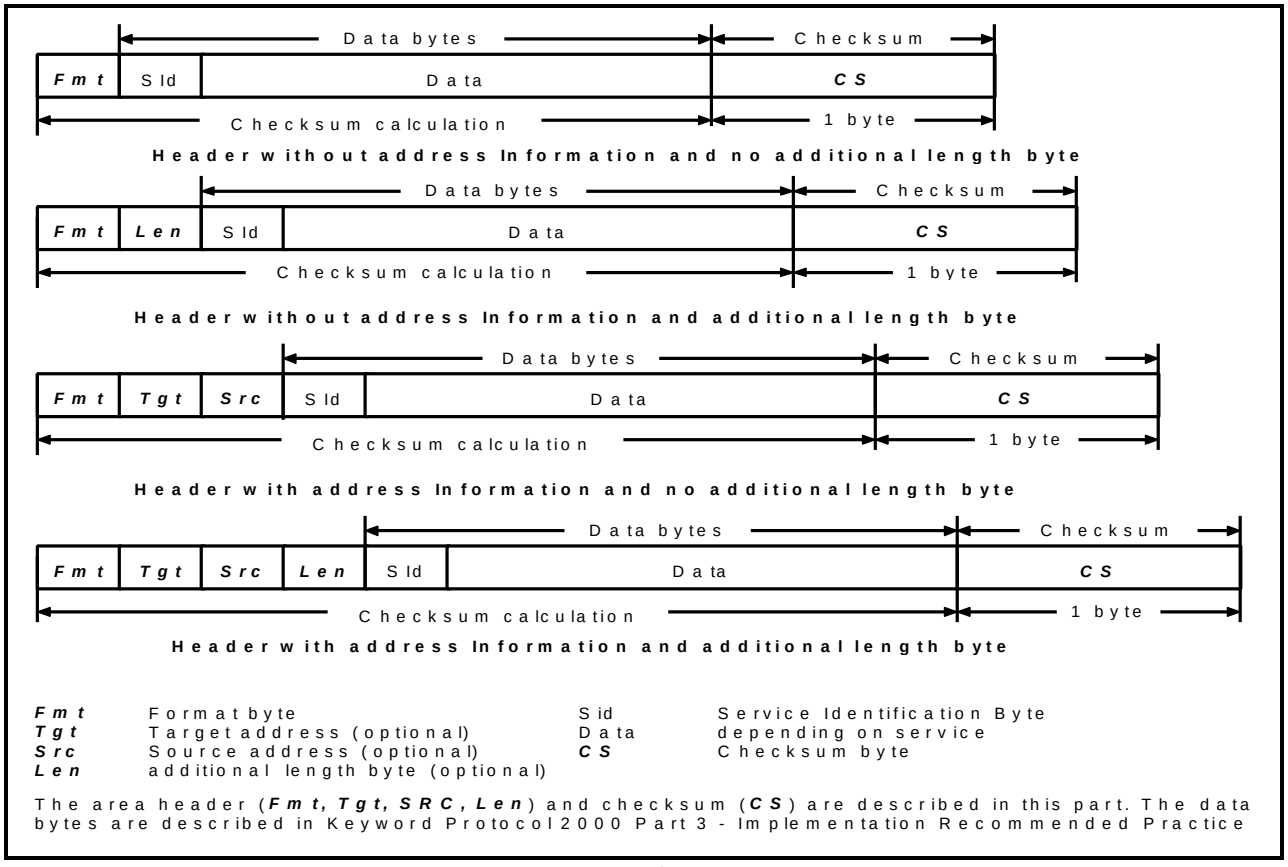


Figure 6 - Use of header bytes

4.2 - Data Bytes

The data field may contain up to 63 or up to 255 bytes of information, depending on the use of length information. The first byte of the data field is the Service Identification Byte. It may be followed by parameters and data depending on the selected service. These bytes are defined in "Keyword Protocol 2000 - Part 3: - Implementation" (for diagnostic services) and in section 5 of this document (for communication services).

4.3 - Checksum Byte

The checksum byte (CS) inserted at the end of the message block is defined as the simple 8-bit sum series of all bytes in the message, excluding the checksum.

If the message is <1> <2> <3> ... <N> , <CS>

where <i> (1 <=### i <=### N) is the numeric value of the ith byte of the message,

then <CS> = <CS>N

where <CS>i (i = 2 to N)

is defined as <CS>i = { <CS> i-1 + <i> }### Modulo 256 and <CS>1 = <1>

Additional security may be included in the data field as defined by the manufacturer.

4.4 - Timing

During normal operation the following timing parameters are relevant:

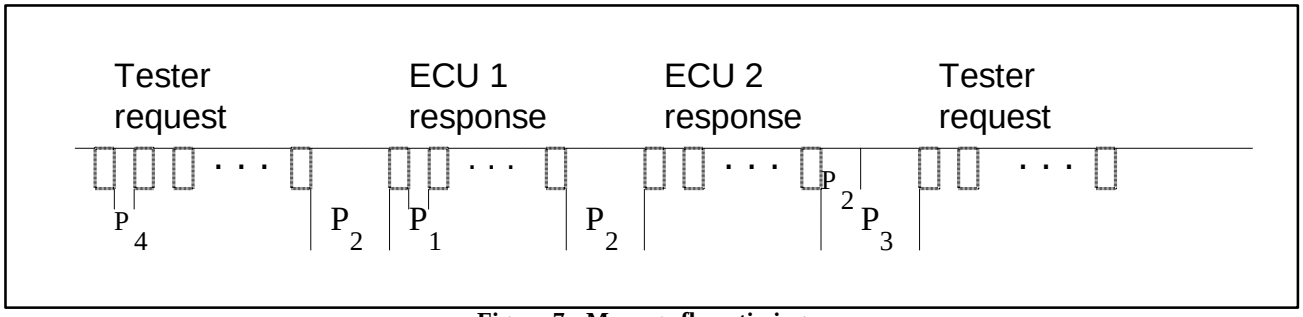


Figure 7 - Message flow, timing

Value	Description
P1	Inter byte time for server (ECU) response
P2	Time between client (tester) request and server (ECU) response or two server (ECU) responses
P3	Time between end of server (ECU) responses and start of new client (tester) request
P4	Inter byte time for client (tester) request

Table 4.4 - Timing parameter definition

There are two sets of default timing parameters:
- One set for normal functional and physical addressed communication. Longer timings are necessary to allow any technics of bus management.
- One is set restricted for physical addressing to allow faster communication.

The client (tester) is informed about the capability of an server (ECU) through the key bytes (see section 5.1.5.1).
Timing parameters may be changed with the communication service "accessTimingParameter" (see section 5.3).
Users must take care for limits listed below and the restrictions:
- P3min > P4min
- Pimin <### Pimax for i=1,...,4

There may be some more restrictions when the client (tester) and listening servers (ECUs) detect the end of a message by timeout. In this case the following restrictions are valid:
- P2min > P4max,
- P2min > P1max

In case of functional addressing - that means that there may be more than one response to one request - some more restrictions may be added.
In case of functional initialisation
- P2max < P3min
- accessTimingParameter service not permitted

Note: If gateways to networks are used in a physical point to point connection with functional initialisation (client (tester) to server (ECU/gateway) connection) a modified timing may be needed to support proper communication on the "K-Line". In such case the system supplier and vehicle manufacturer may require the implementation of the accessTimingParameter service.

In case of physical initialisation more than one response from one server (ECU) is possible (see figure 9).
- It is in the designers responsibility to ensure proper communication in the case of changing the timing parameters from the default values.
- He also has to make sure that the chosen communication parameters are possible for all servers (ECUs) which participate in the session.
- The possible values depend on the capabilities of the server (ECU). In some cases the server (ECU) possibly needs to leave its normal operation mode for switching over to a session with different communication parameters.

4.4.1 - Normal and Extended Timing

The tables below show the timing parameters which are used as default, the limits within which they can be changed and the resolution which may be used to set a new value (with the communication service accessTimingParameter, see section 5.3).

Timing	minimum values [ms]			maximum values [ms]		
Parameter	**lower limit**	**default**	**Resolution**	**default**	**upper limit**	**Resolution**
P1	0	0	---	20	20	---
P2	0	25	0.5	50	calculation see table 3	see table 3
P3	0	55	0.5	5000	∞ ($FF)	250
P4	0	5	0.5	20	20	---

Table 4.4.1.1 - Normal Timing Parameter Set (for functional and physical addressing) (all values in ms)

Timing	minimum values [ms]			maximum values [ms]		
Parameter	**lower limit**	**default**	**Resolution**	**default**	**upper limit**	**Resolution**
P1	0	0	---	20	20	---
P2	0	0	0.5	1000	calculation see table 3	see table 3
P3	0	0	0.5	5000	∞ ($FF)	250
P4	0	5	0.5	20	20	---

Table 4.4.1.2 - Extended Timing Parameters Set (for physical addressing only) (all values in ms)

Timing Parameter	Hex value	Resolution	max. value in [ms]	Maximum value calculation method [ms]
P2max	01 to F0	25	25 to 6000	(hex value) ✸ (Resolution)
	F1	see maximum value calculation method	6400	(low nibble of hex value) ✸ 256 ✸ 25 Example of $FA: ($0A ✸ $0100) ✸ 25 = 64000
	F2		12800	
	F3		19200	
	F4		25600	
	F5		32000	
	F6		38400	
	F7		44800	
	F8		51200	
	F9		57600	
	FA		64000	
	FB		70400	
	FC		76800	
	FD		83200	
	FE		89600	
	FF	---	∞	Not Applicable

Table 4.4.1.3 - P2max Timing Parameter calculation

- Proposed P2max timing parameter calculation method (values > 6000 [ms]). The P2max timing parameter calculation uses 25 [ms] resolution in the range of $01 to $F0. Beginning with $F1 a different calculation method shall be used by the server and the client in order to reach P2max timing values greater than 6000 [ms].

  Calculation Formula for P2max values > $F0

  Calculation_Of_P2max [ms] = (low nibble of P2max) ✸ 256 ✸ 25 [ms]

Note: The P2max timing parameter value shall always be a single byte value in the accessTimingParameter service. The timing modifications shall be activated by implementation of the accessTimingParameter service!

The figure below shows the timing during and after initialisation.

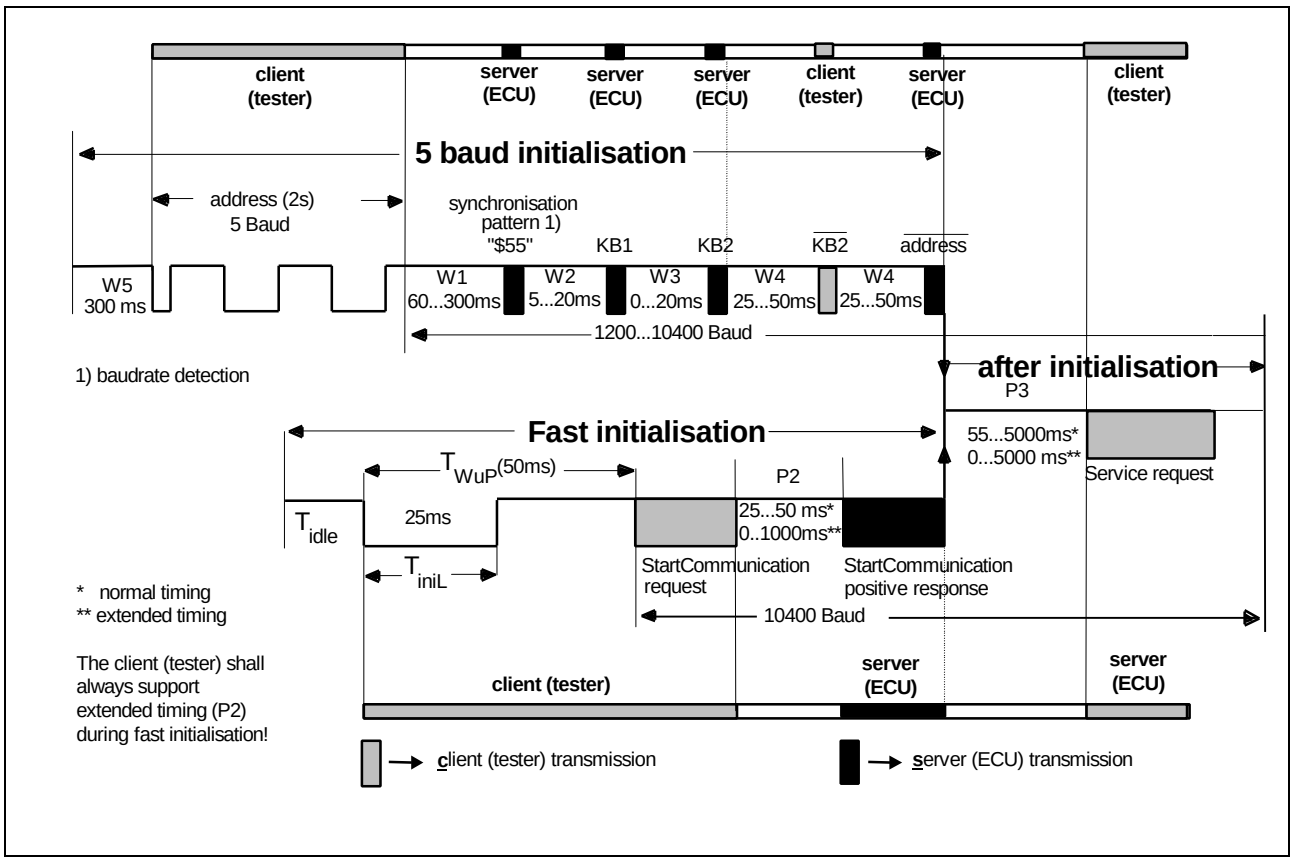


Figure 8 - Timing during and after initialisation

4.4.2 - Timing Exceptions

The extended P2 timing window is a possibility for (a) server(s) to extend the time to respond on a request message. A P2max timing exception is only allowed with the use of one or multiple negative response message(s) with response code $78 (requestCorrectlyReceived-ResponsePending) by the server(s). This response code shall only be used by a server in case it cannot send a positive or negative response message based on the client's request message within the active P2 timing window. This response code shall manipulate the P2max timing parameter value in the server and the client. The P2max timing parameter is set to the value (in ms) of the P3max timing parameter. The client shall remain in the receive mode. The server(s) shall send multiple negative response messages with the negative response code $78 if required.

As soon as the server has completed the task (routine) initiated by the request message it shall send either a positive or negative response message (negative response message with a response code other than $78) based on the last request message received. When the client has received the response message which has been preceded by the negative response message(s) with response code $78, the client and the server shall reset the P2max timing parameter to the previous timing value. The client shall not repeat the request message after the reception of a negative response message with response code $78.

4.4.3 Server (ECU) response data segmentation

Server (ECU) response data segmentation shall only be used if a client (tester) has sent a request message which causes the server (ECU) to split the response message content (data bytes) into several response messages. Data segmentation shall be detected by the client (tester) by comparing source addresses (if present) and Service IDs which must be identical for all response messages during segmentation. This method shall only be used by a server (ECU) if internal server (ECU) resources (i.e. transmit buffer length) are restricted. Server (ECU) response data segmentation shall not be supported in periodic transmission mode. This procedure shall also be used to meet the requirements of ISO 14230 Keyword Protocol 2000 Part 4: Requirements For Emission Related Systems.

Note: If data segmentation is used P3min shall always be > P2max. Data segmentation shall only be used with normal timing parameters.

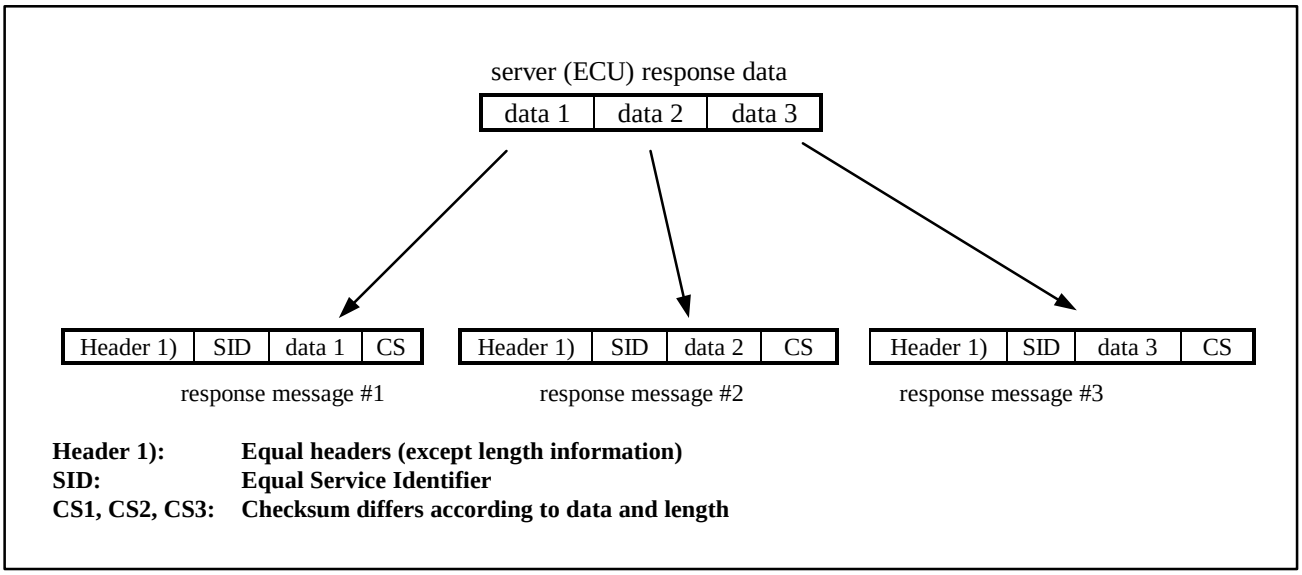


Figure 9 - Data segmentation

4.4.4 - Physical Initialisation

4.4.4.1 - Physical initialisation - single positive response message

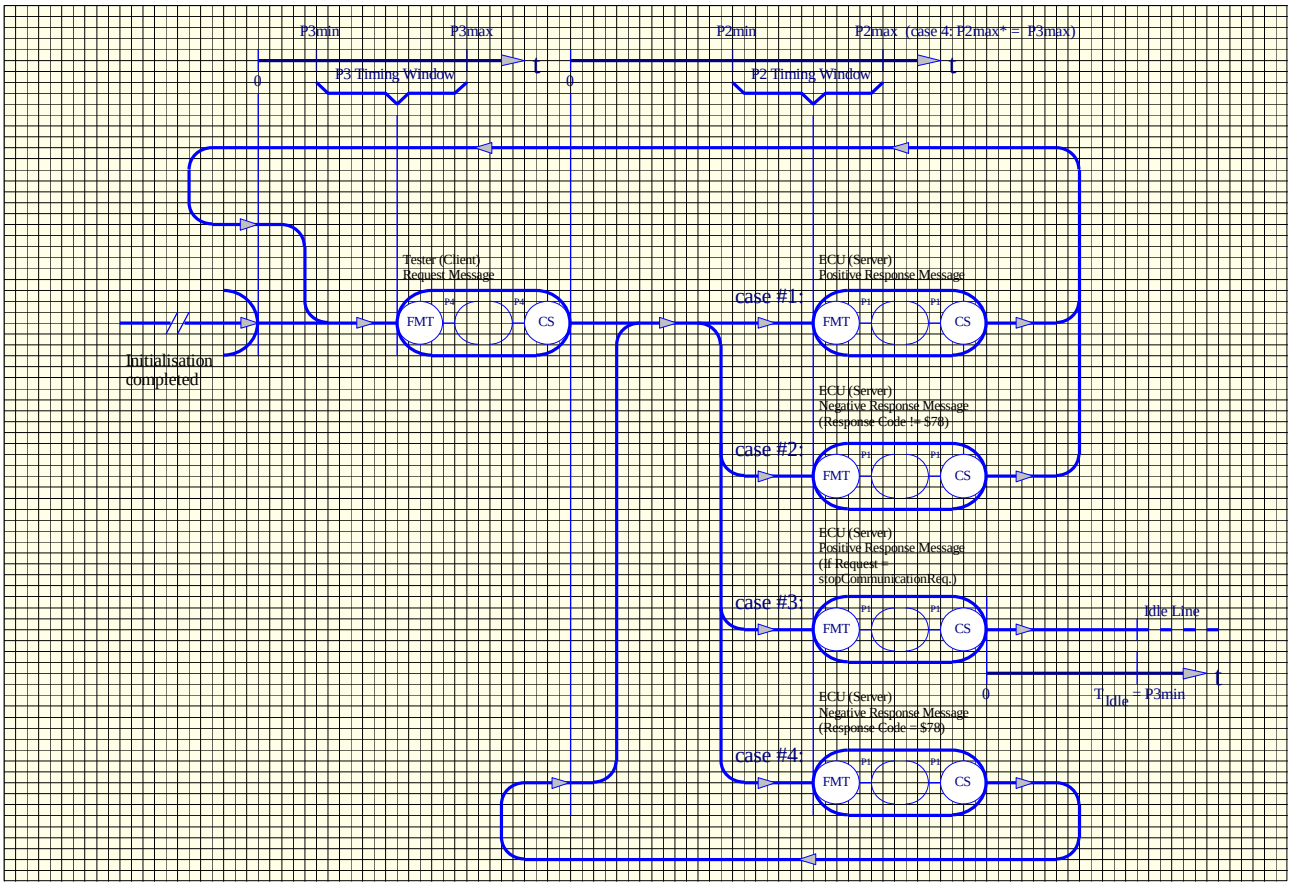


Figure 10 - Physical initialisation - single positive response message

Above figure is based on a physical initialisation (5 baud or fast initialisation) followed by a client (tester) request message (target address = single server (ECU)) and a single response message from the server (ECU). This supports four different cases (see table below).

Case	Description
#1	This case specifies a single positive response message (not stopCommunication) of the server (ECU) preceded by a request message (not stopCommunication) of the client (tester). The response message is sent within the P2 timing window and followed by a client (tester) request message within the P3 timing window.
#2	This case specifies a single negative response message with the negative response code NOT equal to '$78' (requestCorrectlyReceived-ResponsePending) of the server (ECU) preceded by a request message of the client (tester). The response message is sent within the P2 timing window and followed by a client (tester) request message within the P3 timing window.
#3	This case specifies a single positive StopCommunication response message of the server (ECU) preceded by a stopCommunication request message of the client (tester). The response message is sent within the P2 timing window. The P3 time, which has been started after completion of the stopCommunication response message, must first reach the active P3min = TIdle timing value before an "Idle Line" becomes active.
#4	This case specifies one or multiple negative response message with the negative response code set to '$78' (requestCorrectlyReceived-ResponsePending) of the server (ECU) preceded by a request message of the client (tester). The first response message is sent within the P2 timing window. Response code '$78' causes the server (ECU) and the client (tester) to modify the P2max timing parameter to change to the P2max*=P3max timing parameter. The client (tester) shall NOT send any request message. This shall provide the server (ECU) more time to prepare for the succeeding response message. The negative response message is followed by another response message as described in case #1, #2. #3 and #4. It depends on the request message of the client (tester) and/or the behaviour of the server (ECU) whether case #1, #2, #3 or #4 becomes active. After completion of case #1, #2 or #3 the P2max*=P3max timing parameter is reset (in both, server (ECU) and client (tester)) to the previous P2max timing parameter after successful completion of the response message. The timing conditions are kept as long as case #4 is repeated. **Note:** Case #4 must always be terminated with case #1 or #2 or #3 (only if request message = stopCommunication)!

Table 4.4.4.1 - Physical initialisation - single positive response message

4.4.4.2 - Physical initialisation - more than one positive response message

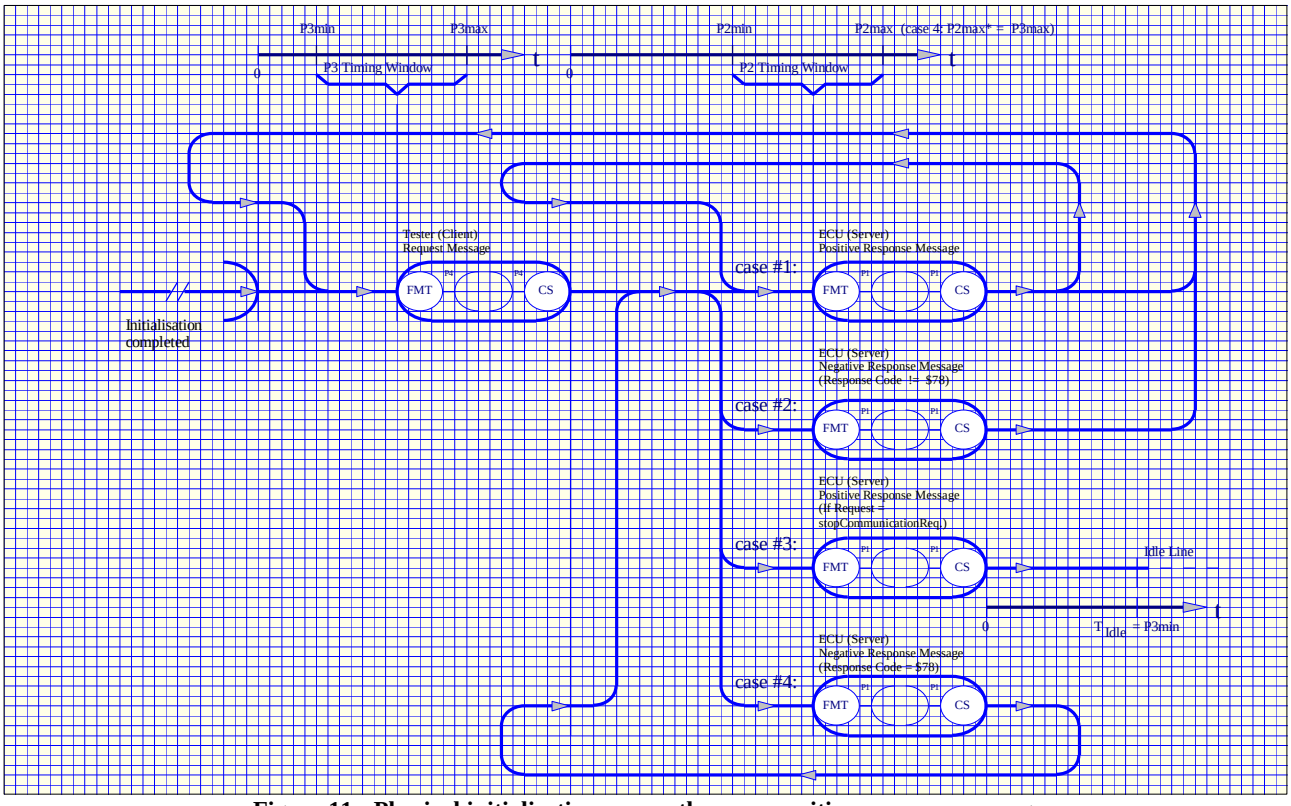


Figure 11 - Physical initialisation - more than one positive response message

Above figure is based on a physical initialisation (5 baud or fast initialisation) followed by a client (tester) request message (target address = single server (ECU)) and more than one positive response message from the server (ECU). This supports four different cases (see table below).

Case	Description
#1	This case specifies more than one positive response messages (not stopCommunication) of the server (ECU) preceded by a request message (not stopCommunication) of the client (tester). The response messages are sent within the P2 timing window. After completion of all positive response messages the last is followed by a client (tester) request message within the P3 timing window. **Note:** More than one positive response message requires data byte accumulation handling in the client (tester). After a positive response message only further positive response messages are allowed (no negative response messages).
#2	This case specifies a single negative response message with the negative response code NOT equal to '$78' (requestCorrectlyReceived-ResponsePending) of the server (ECU) preceded by a request message of the client (tester). The response message is sent within the P2 timing window and followed by a client (tester) request message within the P3 timing window.
#3	This case specifies a single positive stopCommunication response message of the server (ECU) preceded by a stopCommunication request message of the client (tester). The response message is sent within the P2 timing window. The P3 time, which has been started after completion of the stopCommunication response message, must first reach the active P3min = TIdle timing value before an "Idle Line" becomes active.
#4	This case specifies one or multiple negative response message with the negative response code set to '$78' (requestCorrectlyReceived-ResponsePending) of the server (ECU) preceded by a request message of the client (tester). The first response message is sent within the P2 timing window. Response code '$78' causes the server (ECU) and the client (tester) to modify the P2max timing parameter to change to the P2max*=P3max timing parameter. The client (tester) shall NOT send any request message. This shall provide the server (ECU) more time to prepare for the succeeding response message. The negative response message is followed by another response message as described in case #1, #2. #3 and #4. It depends on the request message of the client (tester) and/or the behaviour of the server (ECU) whether case #1, #2, #3 or #4 becomes active. After completion of case #1, #2 or #3 the P2max*=P3max timing parameter is reset (in both, server (ECU) and client (tester)) to the previous P2max timing parameter after successful completion of the response message. The timing conditions are kept as long as case #4 is repeated. **Note:** Case #4 must always be terminated with case #1 or #2 or #3 (only if request message = stopCommunication)!

Table 4.4.4.2 - Physical initialisation - more than one positive response message

4.4.4.3 - Physical initialisation - periodic transmission

The Keyword Protocol 2000 Periodic Transmission Mode shall be enabled by starting a diagnostic session with the startDiagnosticSession service and the diagnosticMode (DCM_) parameter set to $82 for PeriodicTransmission.
PeriodicTransmission shall be supported in connection with physical initialisation, normal and modified timing. The description below explains in steps how the PeriodicTransmission mode shall be activated, handled and de-activated.

Step #1: To enable the PeriodicTransmission mode in the client (tester) and the server (ECU) the client (tester) shall transmit a startDiagnosticSession request message containing the diagnosticMode parameter for the PeriodicTransmission. After the reception of the first positive response message from the server (ECU) the PeriodicTransmission mode is enabled and periodic transmission mode communication structure and timing becomes active. From now on, the server (ECU) shall periodically transmit the last response message with current (updated, if available) data content, until the client (tester) sends a request message within the timing window P3. The timing parameters can be changed within the possible limits of the **„periodic transmission timing parameter“** set with the communication service accessTimingParameters.

Step #2: After reception of any request message e.g. readDataByLocalIdentifier within the timing window P3, the server (ECU) shall periodically transmit the corresponding response message which can be either a positive or a negative response message.

Step #3: After reception of a stopDiagnosticSession or stopCommunication request message within the timing window P3, the server (ECU) shall transmit the corresponding response message only once.
After reception of a stopDiagnosticSession or stopComunication positive response message the periodicTransmissionMode is disabled and the default diagnostic session with the default timing values, defined by the key bytes becomes active.
After reception of a stopDiagnosticSession or stopComunication negative response message the periodicTransmissionMode shall continue. In such case the server (ECU) shall transmit negative response messages unless the client (tester) sends a new request message within the timing window P3.

During the standardDiagnosticModeWithPeriodicTransmission the following rules have to be considered:

- The timing window P3 starts and ends before the timing window P2 starts.

Note: While the default diagnostic session is running, P3max is always greater than P2min. This is caused by the fact that the timing parameters P2min and P3max are measured with different timing resolutions. The resolution of P2min is 0.5 ms and P3max is 250 ms. This results in the maximum value for P2min of 127.5 ms ($FF ✸ 0.5 ms) and the minimum value for P3max of 250 ms ($01 ✸ 250 ms).

- The client (tester) has to ignore the original P3max timing parameter and shall generate a new P3max timing parameter for the internal detection of the end of the timing window P3 which is called from now on P3max*. The timing parameter P3max* shall be generated as described in the following formula:

  - P3max* = P2min - 2 ms.
  - P3min = 5 ms

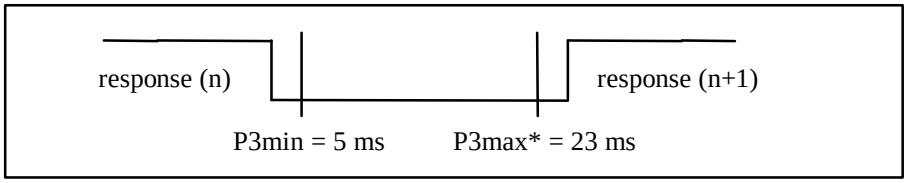


Figure 12 - Periodic transmission "P3 jump-in timing window with default values"

- The timing window P3 starts at P3min and ends at P3max*. It is important for the client (tester) to guarantee a minimum size of the P3 jump-in window for the start of a request message.
- With these default timing conditions the timing window P3 starts at P3min (5ms) and ends at P3max* (23ms)
- To modify the timing parameter in the Periodic Transmission Mode the client (tester) shall read the possible limits from the server (ECU) with the service accessTimingParameter (readLimitsOfPossibleValues). The client (tester) shall then set the timing parameters to the values received from the server (ECU) with the accessTimingParameter (setParameters) request message. After the reception of the first positive response message from the server (ECU) the timing becomes active. The timing parameter P3max* shall be generated as described in the following formula below:

  - P3max* = P2min - 2 ms.

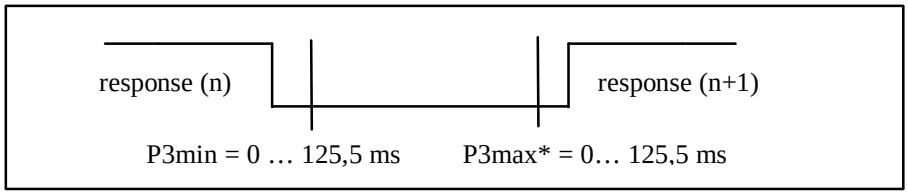


Figure 13 - Periodic transmission with modified "P3max*" timing parameter

Default and optimised timing parameter values:

- The timing table below specifies the timing parameter values with the diagnostic mode PeriodicTransmission.
- P1max shall not exceed P2min. This is required in order to support re-synchronisation between the server (ECU) and client (tester) to meet the error handling requirements.

Timing	minimum values [ms]			maximum values [ms]		
Parameter	lower limit	default	resolution	default	upper limit	resolution
P1	0	0	---	20	20	0.5
P2	2	25	0.5	50	57600 ($F9)	25
P3	0	5	0.5	5000	63500	250
P3* (calculated)	**0**	**5**	**0.5**	**23 (P2min - 2)**	**125,5 (P2min - 2)**	**0.5**
P4	0	5	0.5	20	20	0.5

Table 4.4.4.3 - Timing parameter - periodic transmission

- The **P2max** timing parameter shall not exceed 57600 ms ($F9).
- The **P3max and P3max*** timing parameters are defined different in function and timing values.
  - ⇨ The **P3max** timing parameter is only used for time out detection during neagtive response message handling with the response code “$78 requestCorrectlyReceived-responsePending”.
  - ⇨ The calculated **P3max*** timing parameter is used as the upper limit of the P3 timing window for the client (tester) to send the next response message.

When implementing the standardDiagnosticModeWithPeriodicTransmission the following important note must be considered:

Note: In case the client (tester) lost the periodic transmission mode the timing parameter and a communication break-down criterion shall be defined by the system supplier and the vehicle manufacturer.

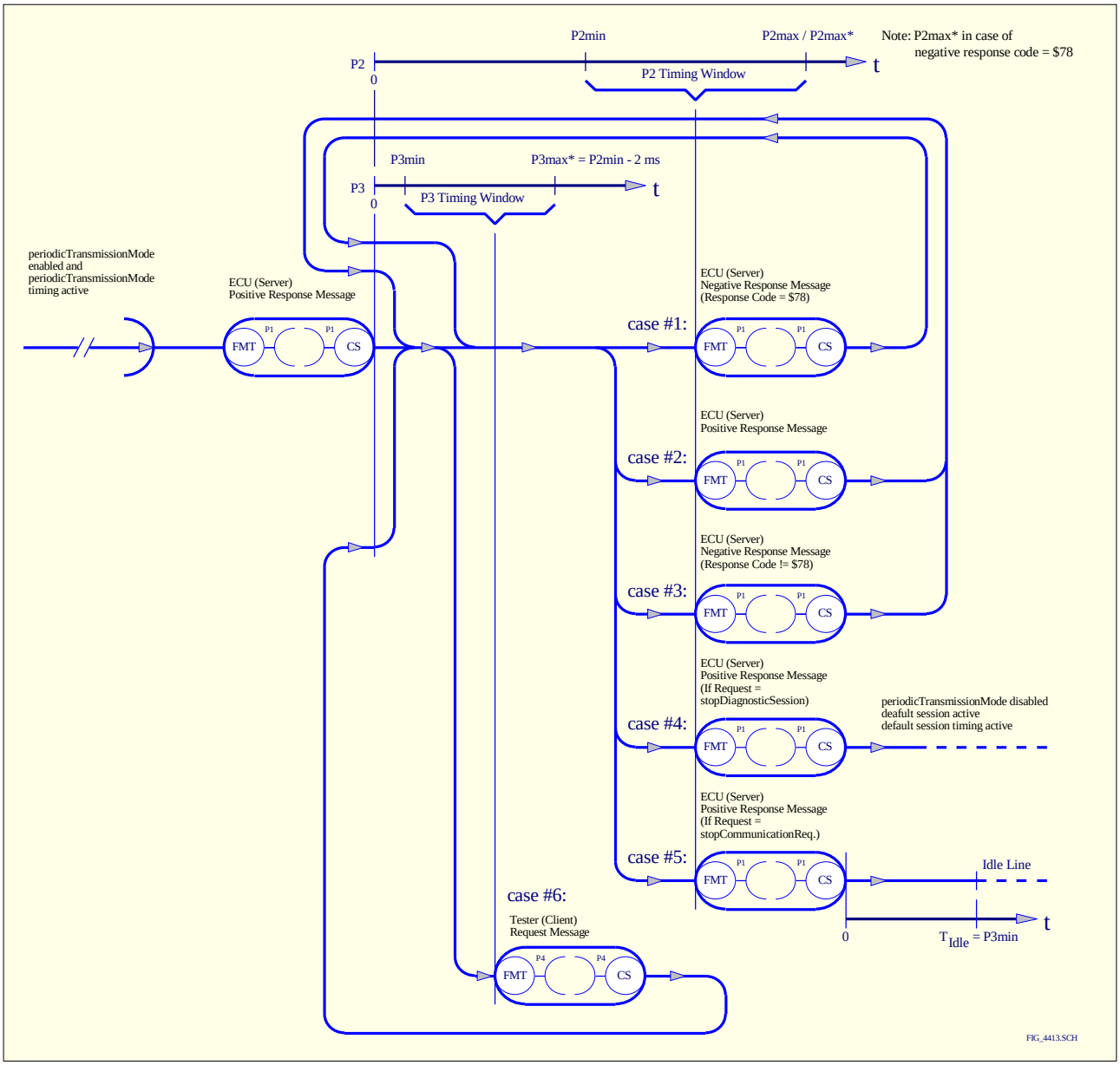


Figure 14 - Physical initialisation - periodic transmission

Above figure is based on a physical initialisation (5 baud or fast initialisation) with the diagnostic mode PeriodicTransmission activated. The client (tester) has transmitted a request message (target address = single server (ECU)) which is followed by periodically transmitted response messages from the server (ECU). This supports five different cases (see table below).

Case	Description
#1	This case specifies periodically transmitted negative response messages with response code equal to $78 of the server (ECU) preceded by a request message of the client (tester). The first response messages is sent within the P2 timing window. For all following negative response messages with response code equal to $78 the P2max* timing becomes active. A detailed specification about the timing behaviour in case of a response code $78 is specified in section 4.1.1 Timing exceptions.
#2	This case specifies periodically transmitted positive response messages (not stopDiagnosticSession and stopCommunication) of the server (ECU) preceded by a request message (not stopDiagnosticSession and not This case specifies periodically transmitted negative response messages with response codes not equal to $78 of his stopCommunication) of the client (tester). The response messages are sent within the P2 timing window.
#3	case specifies one transmitted stopDiagnosticSession positive response messages of the server (ECU) preceded the server (ECU) preceded by a request message of the client (tester). The response messages are sent within t he P2 timing window.
#4	T by a stopDiagnosticSession request message of the client (tester). The response message is sent within the P2 timing window. After the reception of a the stopDiagnosticSession positive response message the standardDiagnosticModeWithPeriodicTransmission shall be disabled and the default diagnostic session with normal timing and default values shall be active.
#5	This case specifies one transmitted stopCommunication positive response messages of the server (ECU) preceded by a stopDiagnosticSession request message of the client (tester). The response message is sent within the P2 timing window. After the reception of a the StopCommunication positive response message the standardDiagnosticModeWithPeriodicTransmission shall be disabled and the default diagnostic session with normal timing and default values shall be active.
#6	This case specifies any transmitted request message of the client (tester) within the P3 timing window. The timing window P3 starts and ends before the timing window P2 during the standardDiagnosticModeWithPeriodicTransmission. As soon as the standardDiagnosticModeWithPeriodicTransmission is activated the communication structure changes and diagnosticModeWithPeriodicTransmission default timing parameter become active. (P3min = 5ms, P3max* = P2min - 2 ms).

Table 4.4.1.3.1 - Physical initialisation - periodic transmission

4.4.4.3.1 - KWP 2000 Periodic Transmission Mode Message Flow A

This section specifies the conditions to enable the Periodic Transmission mode ($82) in the client (tester) and the server (ECU).

STEP#1 startDiagnosticSession(DM_PeriodicTransmission)

time	Client (tester) Request Message	Hex
P3	startDiagnosticSession.ReqSId[	10
	diagnosticMode = DMWPT PeriodicTransmission]	82

time	Server (ECU) Positive Response Message #1	Hex	Server (ECU) Negative Response Message	Hex
P2	startDiagnosticSession.PosRspSId[	50	negativeResponse Service Identifier	7F
	diagnosticMode = DMWPT PeriodicTransmisson]	82	startDiagnosticSession.ReqSId[	21
			responseCode]	xx

***** PeriodicTransmission mode enabled *****
PeriodicTransmission mode default timing values active

time	Server (ECU) Positive Response Message #2	Hex
P2	startDiagnosticSession.PosRspSId[	50
	diagnosticMode = SDMWPT]	82
●		●
●		●

time	Server (ECU) Positive Response Message #n	Hex
P2	startDiagnosticSession.PosRspSId[	50
	diagnosticMode = SDMWPT]	82
	goto STEP#2	

STEP#2 e.g. readDataByLocalIdentifier

time	Client (tester) Request Message	Hex
P3*	readDataByLocalIdentifier[	21
	RLI = recordLocalIdentifier]	01

time	Server (ECU) Positive Response Message #1	Hex	Server (ECU) Negative Response Message	Hex
P2	readDataByLocalIdentifier.PosRspSId[	61	negativeResponse Service Identifier	7F
	RLI = recordLocalIdentifier]	01	readDataByLocalIdentifier.ReqSId[	21
	:	xx	responseCode]	xx
●		●		●
●		●		●

time	Server (ECU) Positive Response Message #n	Hex	Server (ECU) Negative Response Message	Hex
P2	readDataByLocalIdentifier.PosRspSId[	61	negativeResponse Service Identifier	7F
	RLI = recordLocalIdentifier]	01	readDataByLocalIdentifier.ReqSId[	21
	:	xx	responseCode]	xx
	goto STEP#3		**goto STEP#3**	

STEP#3 e.g. stopDiagnosticSession

time	Client (tester) Request Message	Hex
P3*	stopDiagnosticSession.ReqSId[]	20

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	stopDiagnosticSession.PosRspSId[]	60	negativeResponse Service Identifier	7F
			stopDiagnosticSession.ReqSId[	20
			responseCode]	xx
	{standardDiagnosticModeWithPeriodicTransmission disabled - default diagnostic session enabled and normal timing default values active}		{Periodic transmission continue}	

4.4.4.3.2 - KWP 2000 Periodic Transmission Mode Message Flow B

This section specifies the conditions to enable the Periodic Transmission mode ($82) in the client (tester) and the server (ECU).
It also specifies the use of the service accessTimingParameter within this mode to modify the PeriodicTransmission mode default timing.

STEP#1 startDiagnosticSession(DM_PeriodicTransmission)

time	Client (tester) Request Message	Hex
P3	startDiagnosticSession.ReqSId[	10
	diagnosticMode = SDMWPT PeriodicTransmission]	82

time	Server (ECU) Positive Response Message #1	Hex	Server (ECU) Negative Response Message	Hex
P2	startDiagnosticSession.PosRspSId[	50	negativeResponse Service Identifier	7F
	diagnosticMode = SDMWPT PeriodicTransmisson]	82	startDiagnosticSession.ReqSId[	21
			responseCode]	xx

***** PeriodicTransmission mode enabled *****
PeriodicTransmission mode default timing values active

time	Server (ECU) Positive Response Message #2	Hex
P2	startDiagnosticSession.PosRspSId[	50
	diagnosticMode = SDMWPT]	82

● ●
● ●

time	Server (ECU) Positive Response Message #n	Hex
P2	startDiagnosticSession.PosRspSId[	50
	diagnosticMode = SDMWPT]	82
	goto STEP#2	

STEP#2 accessTimingParameters(readLimitsOfPossibleValues)

time	Client (tester) Request Message	Hex
P3*	accessTimingParameter[	83
	TPI = readLimitsOfPossibleValues]	00

time	Server (ECU) Positive Response Message #2	Hex	Server (ECU) Negative Response Message	Hex
P2	accessTimingParameter.PosRspSId[	C3	negativeResponse Service Identifier	7F
	TPI = readLimitsOfPossibleValues]	00	accessTimingParameter.ReqSId[	83
	:	xx	responseCode]	xx
	goto STEP#3		**return(responseCode)**	

● ● ●
● ● ●

time	Server (ECU) Positive Response Message #2	Hex	Server (ECU) Negative Response Message	Hex
P2	accessTimingParameters.PosRspSId[	C3	negativeResponse Service Identifier	7F
	TPI = readLimitsOfPossibleValues]	00	accessTimingParameters.ReqSId[	83
	:	xx	responseCode]	xx
	goto STEP#3		**return(responseCode)**	

STEP#3 accessTimingParameters(setParameters)

time	Client (tester) Request Message	Hex
P3*	accessTimingParameters[	83
	TPI = setParameters	03
	P2min	xx
	P2max	xx
	P3min	xx
	P3max	xx
	P4min]	xx

time	Server (ECU) Positive Response Message #1	Hex	Server (ECU) Negative Response Message	Hex
P2	accessTimingParameters.PosRspSId[	C3	negativeResponse Service Identifier	7F
	TPI = setParameters]	03	accessTimingParameters.ReqSId[	83
			responseCode]	xx

***** modified standardDiagnosticModeWithPeriodicTransmission timing parameter active *****

time	Server (ECU) Positive Response Message #2	Hex	Server (ECU) Negative Response Message	Hex
P2	accessTimingParameters.PosRspSId[	C3	negativeResponse Service Identifier	7F
	TPI = setParameters]	03	accessTimingParameters.ReqSId[	83
			responseCode]	xx

time	Server (ECU) Positive Response Message #3	Hex	Server (ECU) Negative Response Message	Hex
P2	accessTimingParameters.PosRspSId[	C3	negativeResponse Service Identifier	7F
	TPI = setParameters]	03	accessTimingParameters.ReqSId[	83
			responseCode]	xx
●		●		●
●		●		●

time	Server (ECU) Positive Response Message #n	Hex	Server (ECU) Negative Response Message	Hex
P2	accessTimingParameters.PosRspSId[	C3	negativeResponse Service Identifier	7F
	TPI = setParameters]	03	accessTimingParameters.ReqSId[	83
			responseCode]	xx
	goto STEP#4		**return(responseCode)**	

STEP#4 e.g. readDataByLocalIdentifier

time	Client (tester) Request Message	Hex
P3*	readDataByLocalIdentifier[	21
	RLI = recordLocalIdentifier]	01

time	Server (ECU) Positive Response Message #1	Hex	Server (ECU) Negative Response Message	Hex
P2	readDataByLocalIdentifier.PosRspSId[	61	negativeResponse Service Identifier	7F
	RLI = recordLocalIdentifier	01	readDataByLocalIdentifierSId[	21
	:]	xx	responseCode]	xx
●		●		●
●		●		●

time	Server (ECU) Positive Response Message #n	Hex	Server (ECU) Negative Response Message	Hex
P2	readDataByLocalIdentifier.PosRspSId[	61	negativeResponse Service Identifier	7F
	RLI = recordLocalIdentifier	01	readDataByLocalIdentifierSId[	21
	:]	xx	responseCode]	xx
	goto STEP#5		**goto STEP#5**	

STEP#5 e.g. stopDiagnosticSession

time	Client (tester) Request Message	Hex
P3*	stopDiagnosticSession.ReqSId[]	20

time	Server (ECU) Positive Response Message	Hex	Server (ECU) Negative Response Message	Hex
P2	stopDiagnosticSession.PosRspSId[]	60	negativeResponse Service Identifier	7F
			stopDiagnosticSession.ReqSId[	20
			responseCode]	xx
	{standardDiagnosticModeWithPeriodicTransmission disabled - default diagnostic session enabled and normal timing default values active}		{Periodic transmission continue}	

4.4.4.4 Physical initialisation - more than one server (ECU) initialised

The client (tester) shall have the possibility to make physically addressed initialisations of more than one server (ECU), sc. "multiple physical initialisations", by sending physically addressed StartCommunication Request messages to several servers (ECU's) (without sending StopCommunication in-between). In this case the Wake up Pattern shall be transmitted prior to each StartCommunication Request message. In this way a server (ECU) that is not yet initialised only has to listen for the Wake up Pattern.
This means that requests addressed to other initialised servers (ECU's) and responses addressed to the client (tester) from other initialised servers (ECU's) will appear on the K-line while the server (ECU) is initialised.
Reception of the Wake up Pattern, by servers (ECU:s) already initialised, shall be handled with normal error handling (see section 6) and shall not result in a communication reset in the server (ECU).
In case of multiple physical initialisations the following restrictions are valid:

- Header with target and source address must be supported/used by all initialised servers (ECU's)
- If different keywords are used by different initialised servers (ECU's). It is the client's (tester's) responsibility to treat each initialised server (ECU) properly (according to keybytes reported).
- It's recommended not to use the service ATP (see next restriction).
- max(P2max(ecu_1), ..., P2max(ecu_n)) < min(P3max(ecu_1), ..., P3max(ecu_n)). Without this consideration it's possible to loose the communication to one or more ECUs because of a time-out detection within one or more ECU's.

It's recommended to use ECUs with identical timing parameters, e.g. default timíng -> KeyWord = 2027d (Header with target and source, additional length possible, normal timing)

Note: The timing requirements in section 4.4 still prevents the client (tester) from having more than one active request at any one time.
It is in the designer's responsibility to ensure proper communication (error handling) in the case of using multiple physical initialisation. Normal physical error handling may cause serious problems in communication and error handling. The proper solution is to use functional error handling (see error handling in clause 6) instead of. That means, that functional error handling is necessary.

4.4.4.4.1 Message flow example

This example shows the physical initialisation of two ECU's.

STEP#1 startCommunication to ECU1

time	Client (tester) Request Message	Hex
W5	Wake Up Pattern	
	[Fmt = physical addressing Tgt = ECU1] startCommunication.ReqSId	81 11 81

time	Server (ECU1) Positive Response Message	Hex
P2	startCommunication.PosRspSId	C1

STEP#2 e.g. readDataByLocalIdentifier to ECU1

time	Client (tester) Request Message	Hex
P3	readDataByLocalIdentifier[RLI = recordLocalIdentifier]	21 01

time	Server (ECU1) Positive Response Message	Hex	Server (ECU1) Negative Response Message	Hex
P2	readDataByLocalIdentifier.PosRspSId[	61	negativeResponse Service Identifier	7F
	:	01	readDataByLocalIdentifier.ReqSId	21
	:	xx	responseCode	xx

STEP#3 startCommunication to ECU2

time	Client (tester) Request Message	Hex
P3	Wake Up Pattern	
	[Fmt = physical addressing Tgt = ECU2] startCommunication.ReqSId	81 23 81

time	Server (ECU2) Positive Response Message	Hex
P2	startCommunication.PosRspSId	C1

STEP#4 e.g. readDataByLocalIdentifier to ECU1

time	Client (tester) Request Message	Hex
P3	readDataByLocalIdentifier[RLI = recordLocalIdentifier]	21 01

time	Server (ECU1) Positive Response Message	Hex	Server (ECU1) Negative Response Message	Hex
P2	readDataByLocalIdentifier.PosRspSId[	61	negativeResponse Service Identifier	7F
	:	01	readDataByLocalIdentifier.ReqSId	21
	:	xx	responseCode	xx

STEP#5e.g. readDataByLocalIdentifier to ECU2

time	Client (tester) Request Message	Hex
P3	readDataByLocalIdentifier[RLI = recordLocalIdentifier]	21 01

time	Server (ECU2) Positive Response Message	Hex	Server (ECU2) Negative Response Message	Hex
P2	readDataByLocalIdentifier.PosRspSId[	61	negativeResponse Service Identifier	7F
	:	01	readDataByLocalIdentifier.ReqSId	21
	:	xx	responseCode	xx

STEP#6 stopCommunication ECU1

time	Client (tester) Request Message	Hex
P3	stopCommunication.ReqSId	82

time	Server (ECU1) Positive Response Message	Hex	Server (ECU1) Negative Response Message	Hex
P2	stopCommunication.PosRspSId[]	C2	negativeResponse Service Identifier	7F
			stopCommunication.ReqSId[	82
			responseCode]	xx

STEP#7 e.g. readDataByLocalIdentifier to ECU2

time	Client (tester) Request Message	Hex
P3	readDataByLocalIdentifier[RLI = recordLocalIdentifier]	21 01

time	Server (ECU2) Positive Response Message	Hex	Server (ECU2) Negative Response Message	Hex
P2	readDataByLocalIdentifier.PosRspSId[	61	negativeResponse Service Identifier	7F
	:	01	readDataByLocalIdentifier.ReqSId	21
	:	xx	responseCode	xx

STEP#8 stopCommunication ECU2

time	Client (tester) Request Message	Hex
P3	stopCommunication.ReqSId	82

time	Server (ECU2) Positive Response Message	Hex	Server (ECU2) Negative Response Message	Hex
P2	stopCommunication.PosRspSId[]	C2	negativeResponse Service Identifier	7F
			stopCommunication.ReqSId[	82
			responseCode]	xx

4.4.5 - Functional initialisation

With this procedure a group of servers (ECUs) is initialised. In case of 5 baud initialisation address bytes which define a functional group of servers (ECUs) are listed in Appendix A. In case of fast initialisation the format byte $C1 (HM3) of the start communication request shall be used. During communication it is possible to switch from functional to physical addressing (HM3 to HM2).
During functional communication (HM3) the target address of a request has to be interpreted as a functional (group) address, the source address is the physical address of the client (tester). In any case in a response the target and source addresses are physical addresses.

In case of 5 baud initialisation the response of key bytes shall be done by only one server (ECU) (Arbitration).
In case of fast initialisation all addressed servers (ECUs) shall send a startCommunication response message.
Functional initialisation requires that server(s) (ECU(s) must support arbitration (refer to section 7).
Functional initialisation requires normal timing parameters with default values.
The use of the accessTimingParameter request is not permitted.
If 5 baud initialisation is used, all servers (ECUs) shall use 10400 baud.

4.4.5.1 - Functional initialisation - single positive response message - single server (ECU) addressed

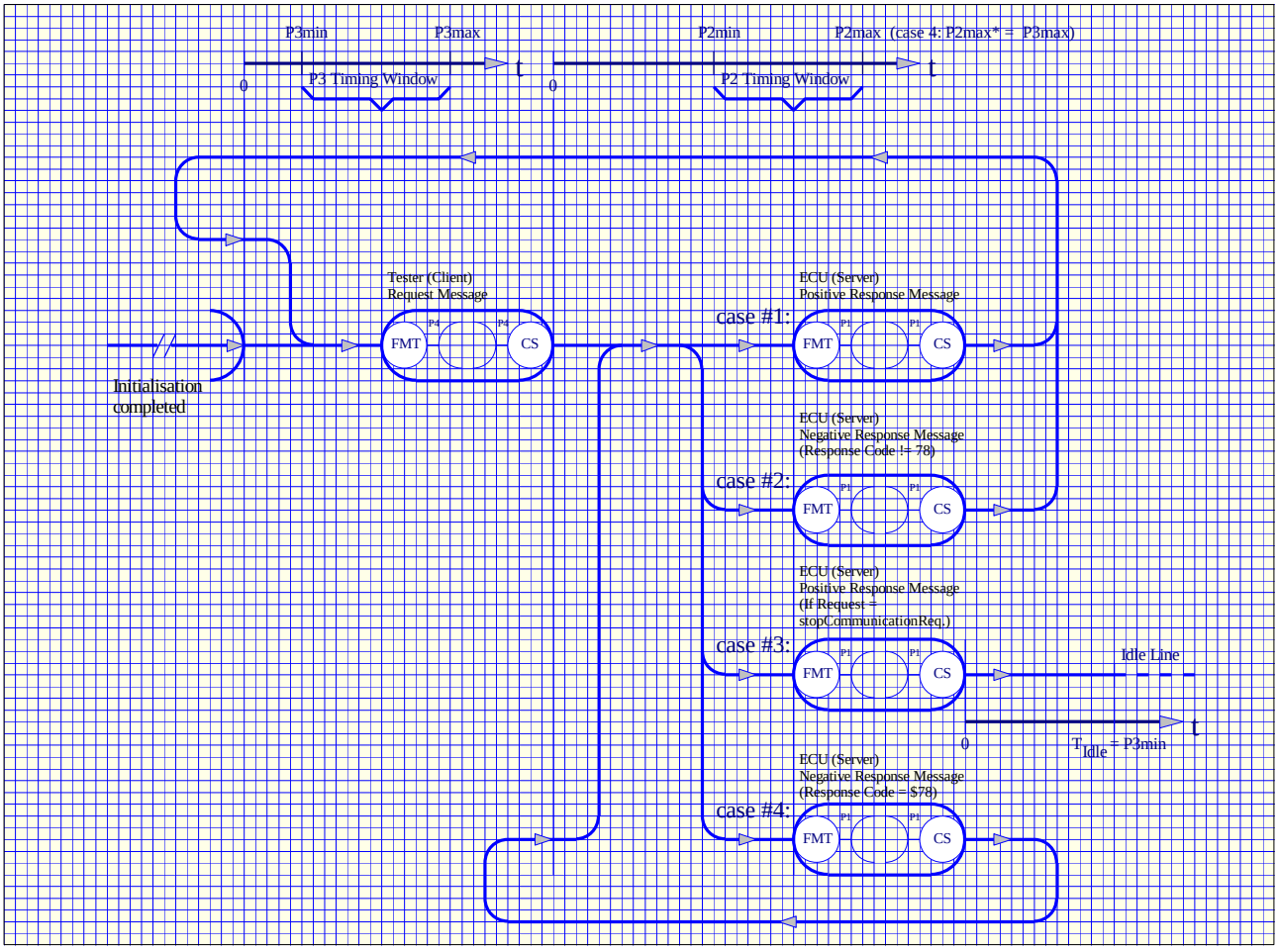


Figure 15 - Functional initialisation - single positive response message - single server (ECU) addressed

Above figure is based on a functional initialisation (5 baud or fast initialisation) followed by a client (tester) request message (target address = single server (ECU)) and a single response message from the server (ECU) addressed. This supports four different cases (see table below).

Case	Description
#1	This case specifies a single positive response message (not stopCommunication) of the server (ECU) preceded by a request message (not stopCommunication) of the client (tester). The response message is sent within the P2 timing window and followed by a client (tester) request message within the P3 timing window.
#2	This case specifies a single negative response message with the negative response code NOT equal to '$78' (requestCorrectlyReceived-ResponsePending) of the server (ECU) preceded by a request message of the client (tester). The response message is sent within the P2 timing window and followed by a client (tester) request message within the P3 timing window.
#3	This case specifies a single positive stopCommunication response message of the server (ECU) preceded by a stopCommunication request message of the client (tester). The response message is sent within the P2 timing window. The P3 time, which has been started after completion of the StopCommunication response message, must first reach the active P3min = TIdle timing value before an "Idle Line" becomes active.
#4	This case specifies one or multiple negative response message with the negative response code set to '$78' (requestCorrectlyReceived-ResponsePending) of the server (ECU) preceded by a request message of the client (tester). The first response message is sent within the P2 timing window. Response code '$78' causes the server (ECU) and the client (tester) to modify the P2max timing parameter to change to the P2max*=P3max timing parameter. The client (tester) shall NOT send any request message. This shall provide the server (ECU) more time to prepare for the succeeding response message. The negative response message is followed by another response message as described in case #1, #2. #3 and #4. It depends on the request message of the client (tester) and/or the behaviour of the server (ECU) whether case #1, #2, #3 or #4 becomes active. After completion of case #1, #2 or #3 the P2max*=P3max timing parameter is reset (in both, server (ECU) and client (tester)) to the previous P2max timing parameter after successful completion of the response message. The timing conditions are kept as long as case #4 is repeated. **Note:** Case #4 must always be terminated with case #1 or #2 or #3 (only if request message = stopCommunication)!

Table 4.4.2.1 - Functional initialisation - single positive response message - single server (ECU) addressed

4.4.5.2 - Functional initialisation - more than one response message - single server (ECU) addressed

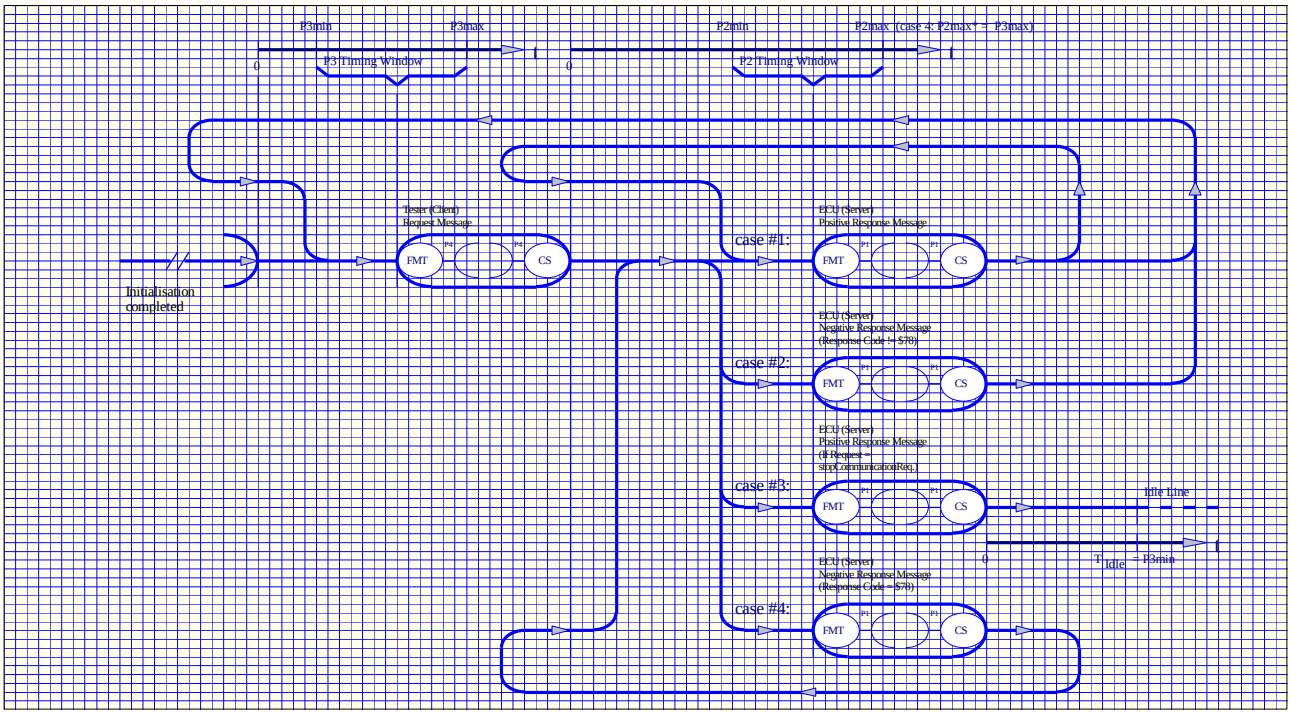


Figure 16 - Functional initialisation - more than one response message - single server (ECU) addressed

Above figure is based on a functional initialisation (5 baud or fast initialisation) followed by a client (tester) request message (target address = single server (ECU)) and more than one response message from the server (ECU) addressed. This supports four different cases (see table below).

Case	Description
#1	This case specifies more than one positive response message (not stopCommunication) of the server (ECU) preceded by a request message (not stopCommunication) of the client (tester). The response messages are sent within the P2 timing window. After completion of all positive response messages the last is followed by a client (tester) request message within the P3 timing window. **Note:** After a positive response message only further positive response messages are allowed (no neg. response messages).
#2	This case specifies a single negative response message with the negative response code NOT equal to '$78' (requestCorrectlyReceived-ResponsePending) of the server (ECU) preceded by a request message of the client (tester). The response message is sent within the P2 timing window and followed by a client (tester) request message within the P3 timing window.
#3	This case specifies a single positive stopCommunication response message of the server (ECU) preceded by a stopCommunication request message of the client (tester). The response message is sent within the P2 timing window. The P3 time, which has been started after completion of the stopCommunication response message, must first reach the active P3min = TIdle timing value before an "Idle Line" becomes active.
#4	This case specifies one or multiple negative response message with the negative response code set to '$78' (requestCorrectlyReceived-ResponsePending) of the server (ECU) preceded by a request message of the client (tester). The first response message is sent within the P2 timing window. Response code '$78' causes the server (ECU) and the client (tester) to modify the P2max timing parameter to change to the P2max*=P3max timing parameter. The client (tester) shall NOT send any request message. This shall provide the server (ECU) more time to prepare for the succeeding response message. The negative response message is followed by another response message as described in case #1, #2. #3 and #4. It depends on the request message of the client (tester) and/or the behaviour of the server (ECU) whether case #1, #2, #3 or #4 becomes active. After completion of case #1, #2 or #3 the P2max*=P3max timing parameter is reset (in both, server (ECU) and client (tester)) to the previous P2max timing parameter after successful completion of the response message. The timing conditions are kept as long as case #4 is repeated. **Note:** Case #4 must always be terminated with case #1 or #2 or #3 (only if request message = stopCommunication)!

Table 4.4.2.2 - Functional initialisation - more than one response message - single server (ECU) addressed

4.4.5.3 - Functional initialisation - single positive response message - more than one server (ECU)

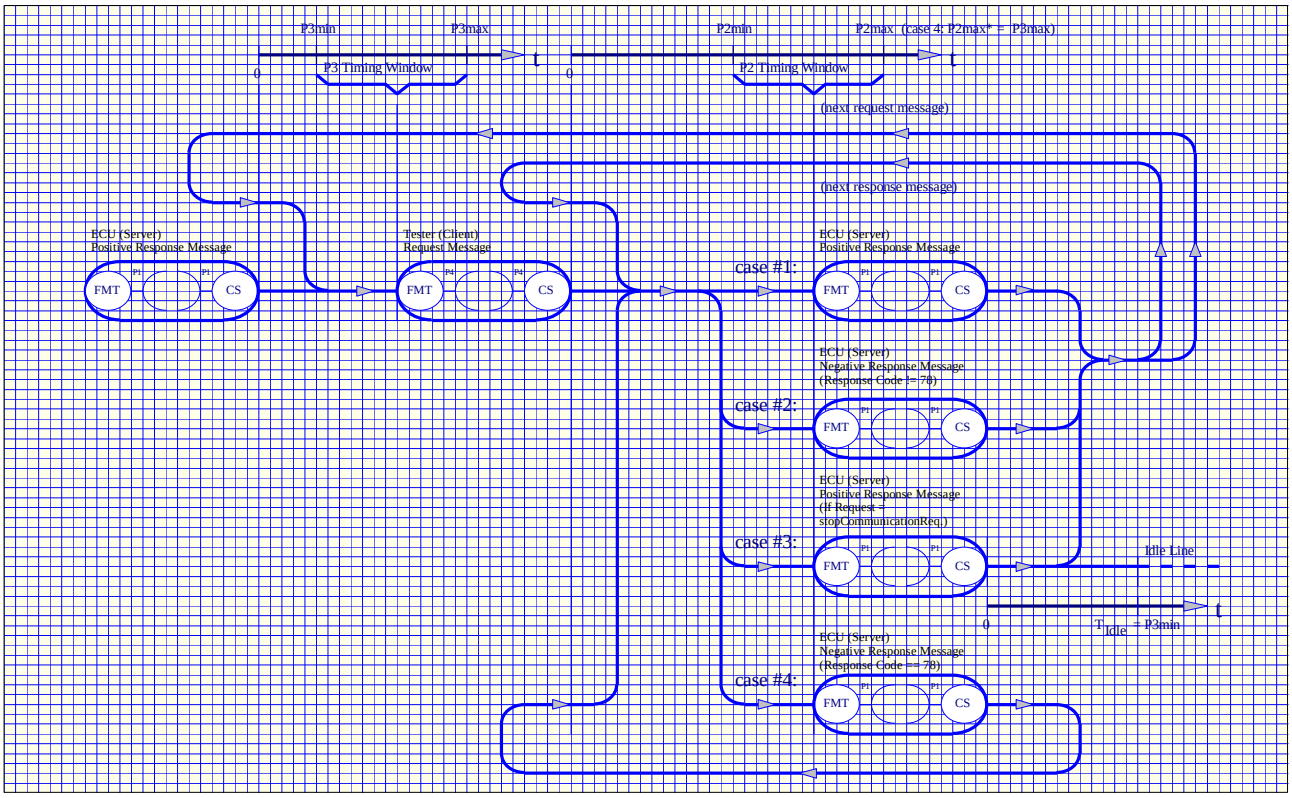


Figure 17 - Functional initialisation - single positive response message - more than one server (ECU)

Above figure is based on a functional initialisation (5 baud or fast initialisation) followed by a client (tester) request message (target address = more than one server (ECU)) and a single response message from each server (ECU) addressed. This supports four different cases (see table below).

Case	Description
#1	This case specifies a single positive response message (not stopCommunication) of the servers (ECUs) preceded by a request message (not stopCommunication) of the client (tester). The response message is sent within the P2 timing window and followed by a client (tester) request message within the P3 timing window.
#2	This case specifies a single negative response message with the negative response code NOT equal to '$78' (requestCorrectlyReceived-ResponsePending) of the servers (ECUs) preceded by a request message of the client (tester). The response message is sent within the P2 timing window and followed by a client (tester) request message within the P3 timing window.
#3	This case specifies a single positive stopCommunication response message of the servers (ECUs) preceded by a stopCommunication request message of the client (tester). The response message is sent within the P2 timing window. The P3 time, which has been started after completion of the stopCommunication response message, must first reach the active P3min = TIdle timing value before an "Idle Line" becomes active.
#4	This case specifies one or multiple negative response message with the negative response code set to '$78' (requestCorrectlyReceived-ResponsePending) of the servers (ECUs) preceded by a request message of the client (tester). The first response message is sent within the P2 timing window. Response code '$78' causes the server (ECU) and the client (tester) to modify the P2max timing parameter to change to the P2max*=P3max timing parameter. The client (tester) shall NOT send any request message. This shall provide the servers (ECUs) more time to prepare for the succeeding response message. The negative response message is followed by another response message as described in case #1, #2. #3 and #4. It depends on the request message of the client (tester) and/or the behaviour of the servers (ECUs) whether case #1, #2, #3 or #4 becomes active. After completion of case #1, #2 or #3 the P2max*=P3max timing parameter is reset (in both, servers (ECUs) and client (tester)) to the previous P2max timing parameter after successful completion of the response message. The timing conditions are kept as long as case #4 is repeated. **Note:** Case #4 must always be terminated with case #1 or #2 or #3 (only if request message = stopCommunication)!

Table 4.4.2.3 - Functional initialisation - single response message - more than one server (ECU)

4.4.5.4 - Functional initialisation - more than one response message - more than one server (ECU)

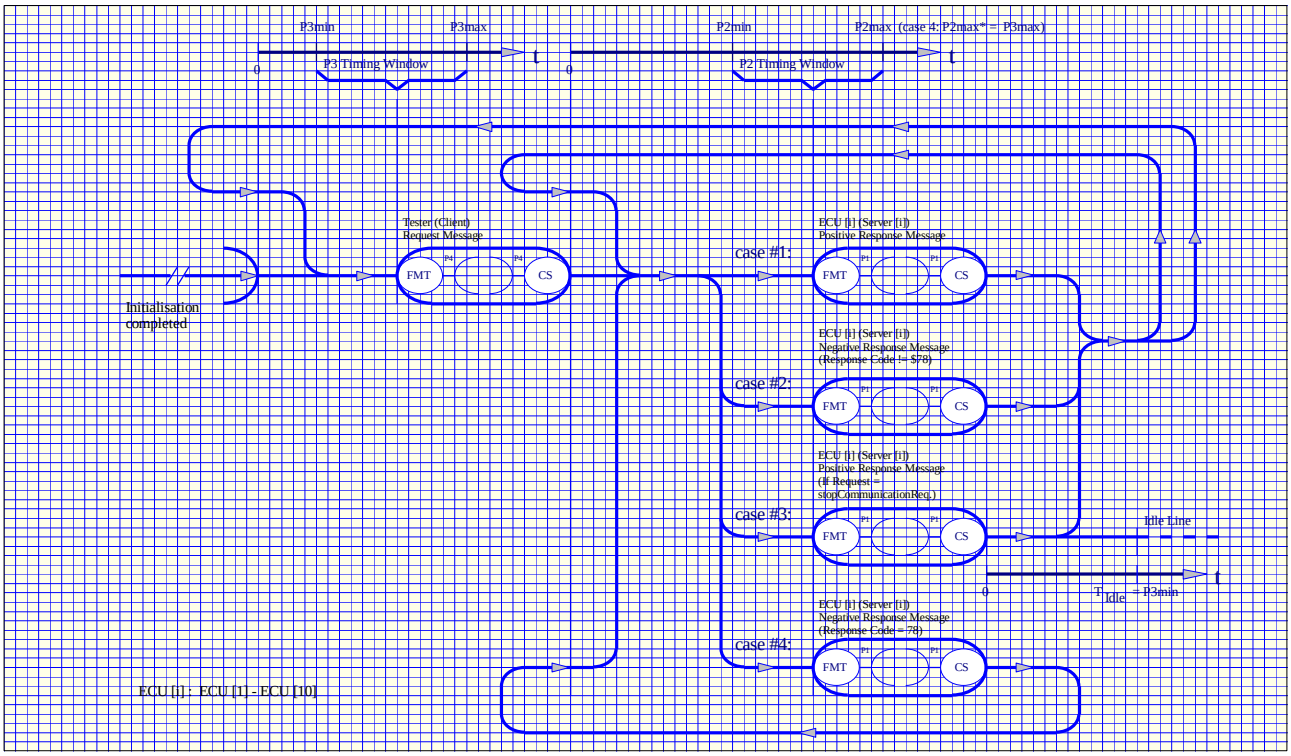


Figure 18 - Functional initialisation - more than one response message - more than one server (ECU)

Above figure is based on a functional initialisation (5 baud or fast initialisation) followed by a client (tester) request message (target address = more than one server (ECU)) and more than 1 response message from the servers (ECUs) addressed. This supports four different cases (see table below).

Case	Description
#1	This case specifies a single positive response message (not stopCommunication) of the server (ECU) preceded by a request message (not stopCommunication) of the client (tester). The response message is sent within the P2 timing window and followed by a client (tester) request message within the P3 timing window. Note: After a positive response message only further positive response messages from the same server (ECU) are allowed (no negative response messages).
#2	This case specifies a single negative response message with the negative response code NOT equal to '$78' (requestCorrectlyReceived-ResponsePending) of the server (ECU) preceded by a request message of the client (tester). The response message is sent within the P2 timing window and followed by a client (tester) request message within the P3 timing window.
#3	This case specifies a single positive stopCommunication response message of the server (ECU) preceded by a stopCommunication request message of the client (tester). The response message is sent within the P2 timing window. The P3 time, which has been started after completion of the stopCommunication response message, must first reach the active P3min = Tidle timing value before an "Idle Line" becomes active.
#4	This case specifies one or multiple negative response message with the negative response code set to '$78' (requestCorrectlyReceived-ResponsePending) of the server (ECU) preceded by a request message of the client (tester). The first response message is sent within the P2 timing window. Response code '$78' causes the server (ECU) and the client (tester) to modify the P2max timing parameter to change to the P2max*=P3max timing parameter. The client (tester) shall NOT send any request message. This shall provide the server (ECU) more time to prepare for the succeeding response message. The negative response message is followed by another response message as described in case #1, #2. #3 and #4. It depends on the request message of the client (tester) and/or the behaviour of the server (ECU) whether case #1, #2, #3 or #4 becomes active. After completion of case #1, #2 or #3 the P2max*=P3max timing parameter is reset (in both, server (ECU) and client (tester)) to the previous P2max timing parameter after successful completion of the response message. The timing conditions are kept as long as case #4 is repeated. **Note:** Case #4 must always be terminated with case #1 or #2 or #3 (only if request message = stopCommunication)!

Table 4.4.2.4 - Functional initialisation - more than one response message - more than one server (ECU)

5 - Communication services

Some services are necessary to establish and maintain communication. Those are not diagnostic services because they do not appear on the application layer. They are described in the same formal way and with the same conventions as the Diagnostic Services (see Keyword Protocol 2000 - Part 3: Implementation): A service table and a verbal description of the service procedure; parameters are mandatory (M), selectable (S), conditional (C) or user optional (U). A description of implementation on the physical layer of Keyword Protocol 2000 is added.

In general services are not mandatory, only startCommunication service must be implemented.
The startCommunication service and the accessTimingParameter service are used for starting a diagnostic communication. In order to perform any diagnostic service, communication must be initialised and the communication parameters need to be appropriate to the desired diagnostic mode. A chart describing this is shown in figure 5.

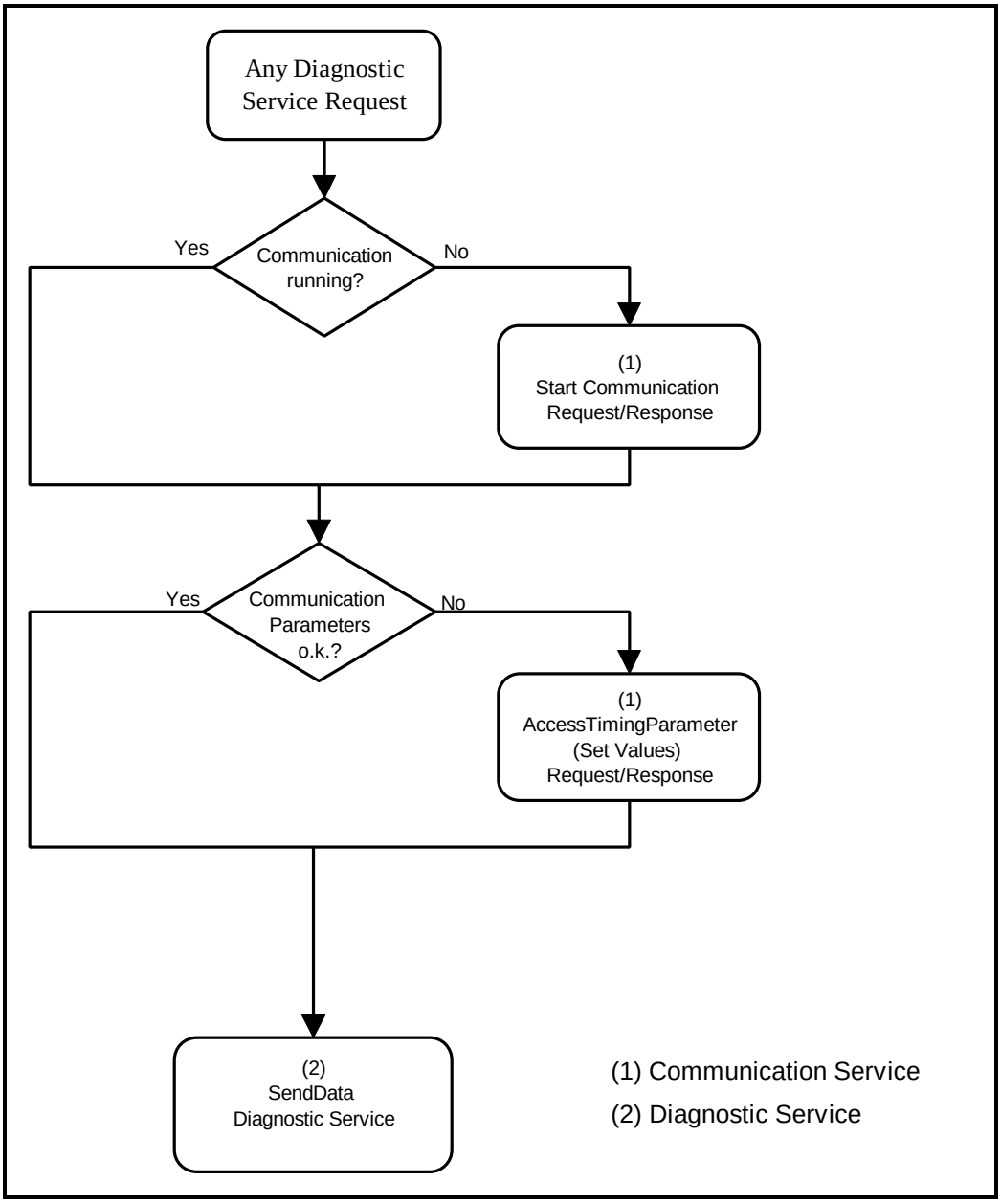


Figure 19 - Use of communication services

Service Identifier value summary table

The left column of the following table lists all communication services described in the KWP 2000 - Data Link Layer Recommended Practice, 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".

Communication Service Name	KWP 2000 Implementation			
	Negative Response value	Request Hex Value	Response Hex Value	Section No.
startCommunication	NO	81	C1	5.1
stopCommunication	**7F**	82	C2	5.2
accessTimingParameter	**7F**	83	C3	5.3

Table 5 - Service Identifier value summary table

5.1 - StartCommunication Service

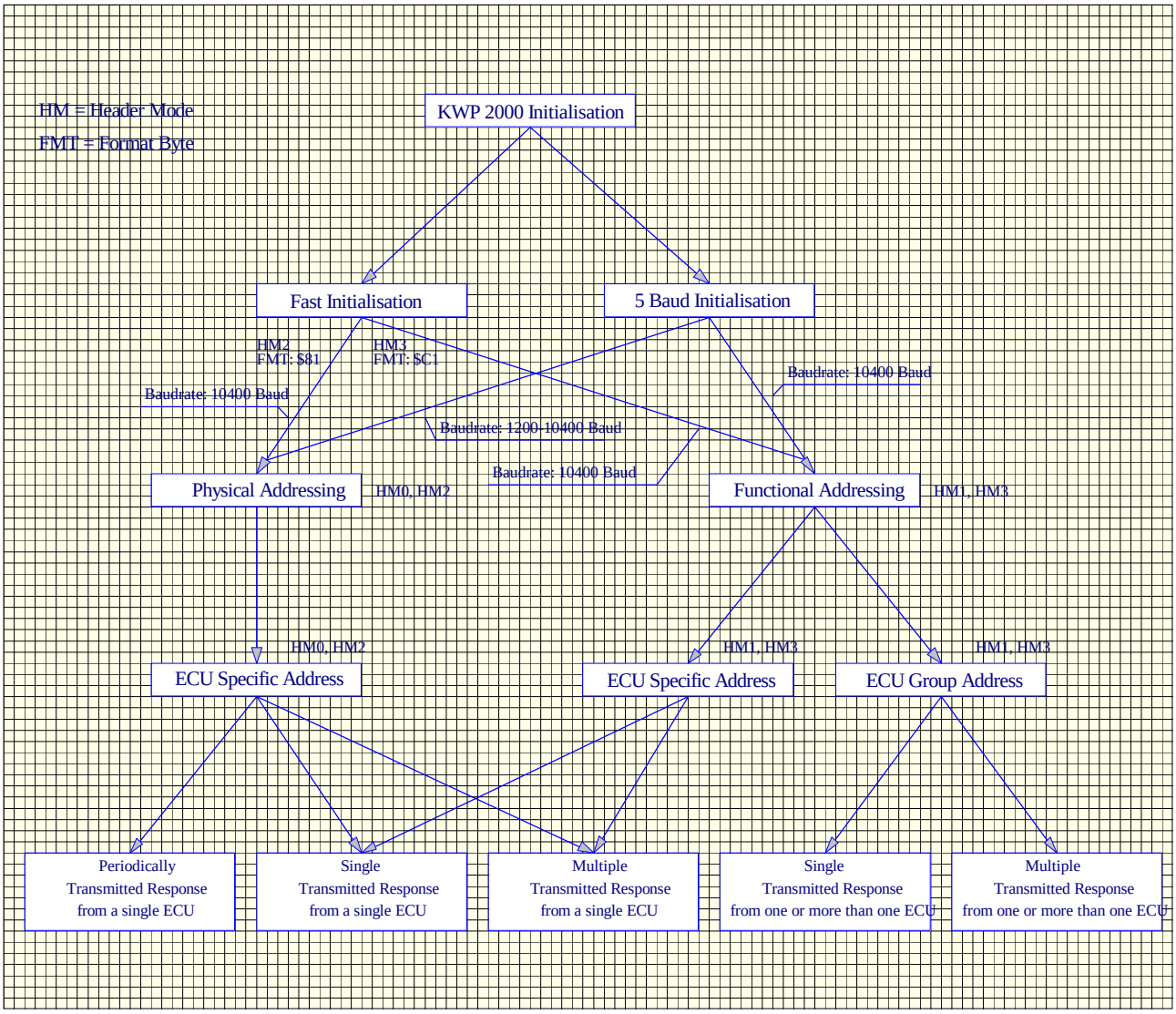


Figure 20 - Functional initialisation - more than one response - more than one server (ECU)

5.1.1 - Service Definition

5.1.2 - Service Purpose

The purpose of this KWP 2000 communication layer service is to initialise the communication link for the exchange of diagnostic data.

5.1.3 - Service Table

StartCommunication Request	M
Initialisation Mode Identifier	M
Target Initialisation Address	M
Source Initialisation Address	C
StartCommunication Positive Response	M
Key bytes	M

Table 5.1.3 - StartCommunication Service

C: Source initialisation address is added if Initialisation Mode Identifier = Fast Initialisation

Application Note: The way of initialisation is determined by the Initialisation Mode Identifier, the value of this parameter may be CARB initialisation, 5 baud initialisation or fast initialisation.

5.1.4 - Service Procedure

Upon receiving a startCommunication indication primitive, the server (ECU) shall check if the requested communication link can be initialised under the present conditions. Valid conditions for the initialisation of a diagnostic communication link are described in section 5.1.5 “Implementation” of this document.
Then the server (ECU) shall perform all actions necessary to initialise the communication link and send a startCommunication response primitive with the positive response parameters selected.

If the communication link cannot be initialised by any reason, the server (ECU) shall maintain its normal operation (see section 6, Error handling).

5.1.5 - Implementation

The startCommunication Service is used to initialise a communication on the “K-line”. There are different possibilities to initialise:

Identifier	Initialisation mode
1	CARB initialisation
2	5 baud initialisation
3	Fast initialisation

Table 5.1.5 - Initialisation mode identifier

The figure below shows the three possibilities and the server (ECU) status after each kind of initialisation.
After finishing the initialisation the server (ECU)s are in the same status, regardless of the initialisation mode:

- all communication parameters are set to default values according to the key bytes.
- server (ECU) is waiting for the first request of the client (tester) for a time period of P3.
- server (ECU) is in the default diagnostic mode (= has a well defined functionality).

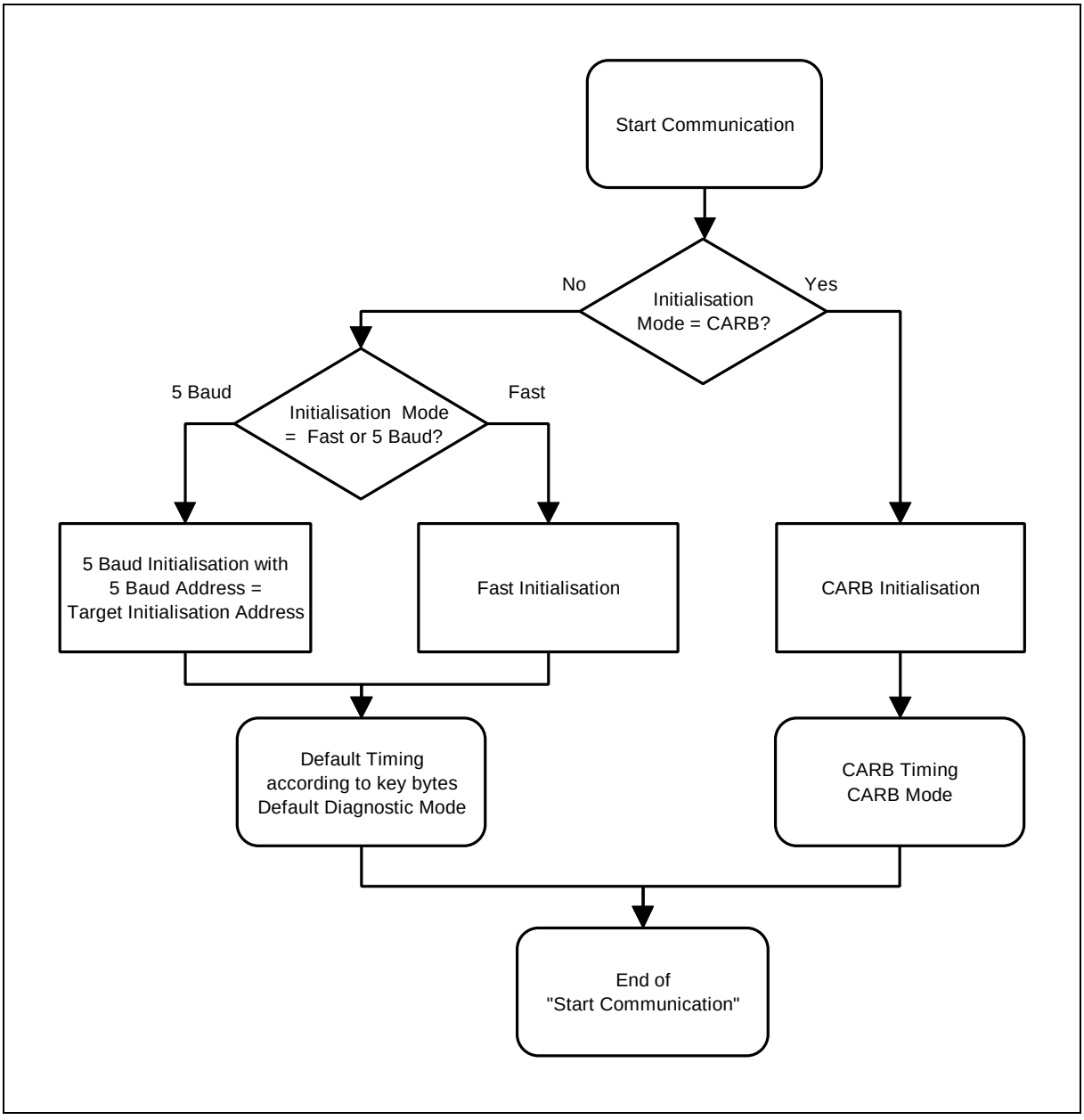


Figure 21 - Initialisation modes

There are general facts that are common to all modes of initialisation:

- Prior to any activity there shall be a bus-idle time.
- Then the client (tester) sends an initialisation pattern.
- All information which is necessary to establish communication is contained in the response of the server (ECU).

5.1.5.1 - Key bytes

With these bytes a server (ECU) informs the client (tester) about the supported header, timing and length information. So a server (ECU) not necessarily has to support all possibilities. The decoding of the key bytes is defined in ISO 9141:1989. KB1 = Low Byte, KB2 = High Byte, 7 bit, odd parity.

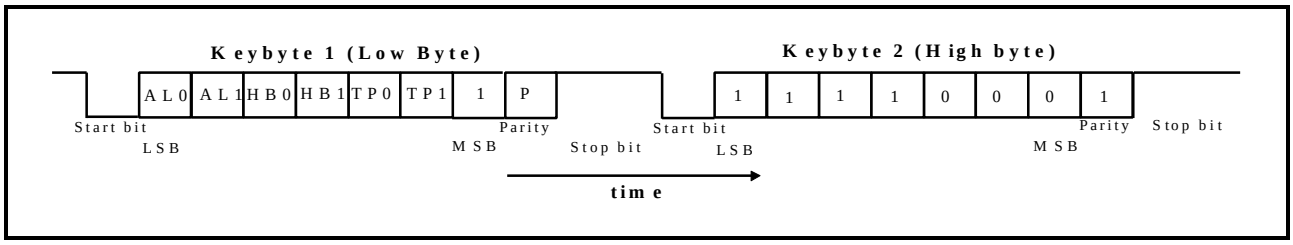


Figure 22 - Key bytes

Bit	Bit state = '0'	Bit state = '1'
AL0	length information in format byte not supported	length information in format byte supported
AL1	additional length byte not supported	additional length byte supported
HB0	1 byte header not supported	1 byte header supported
HB1	Target/Source address in header not supported	Target/Source address in header supported
TP0*)	normal timing parameter set	extended timing parameter set
TP1*)	extended timing parameter set	normal timing parameter set

Table 5.1.5.1.1 - Decoding Key bytes

***) only TP0,TP1 = 0,1 and 1,0 allowed**

The following table lists all key bytes specified by this document:

Key bytes				Supported message format by the server (ECU)		
Binary: KB2	**KB1**	**Hex**	**Dec.*)**	**Length information**	**Type of header**	
1000 1111	1101 0000	$8FD0	2000	not supported by KWP2000 - Data Link Layer Recommended Practice		
1000 1111	1101 0101	$8FD5	2005	format byte	1 byte header	extended timing
1000 1111	1101 0110	$8FD6	2006	additional length byte		
1000 1111	0101 0111	$8F57	2007	both modes possible		
1000 1111	1101 1001	$8FD9	2009	format byte	Header with target and source address information	
1000 1111	1101 1010	$8FDA	2010	additional length byte		
1000 1111	0101 1011	$8F5B	2011	both modes possible		
1000 1111	0101 1101	$8F5D	2013	format byte	Both types of header supported	
1000 1111	0101 1110	$8F5E	2014	additional length byte		
1000 1111	1101 1111	$8FDF	2015	both modes possible		
1000 1111	1110 0101	$8FE5	2021	format byte	1 byte header	normal timing
1000 1111	1110 0110	$8FE6	2022	additional length byte		
1000 1111	0110 0111	$8F67	2023	both modes possible		
1000 1111	1110 1001	$8FE9	2025	format byte	Header with target and source address information	
1000 1111	1110 1010	$8FEA	2026	additional length byte		
1000 1111	0110 1011	$8F6B	2027	both modes possible		
1000 1111	0110 1101	$8F6D	2029	format byte	Both types of header supported	
1000 1111	0110 1110	$8F6E	2030	additional length byte		
1000 1111	1110 1111	$8FEF	2031	both modes possible		

Table 5.1.5.1.2 - Possible values of key bytes

***): Calculation of decimal value: clear the parity bit of both key bytes, multiply key byte 2 by 2^7 and add key byte 1.**

A client (tester) which fulfils Keyword Protocol 2000 according to KWP 2000 - Data Link Layer Recommended Practice shall support 100% functionality, independent from the key bytes transferred from the server (ECU).
A server (ECU) transmits is supported functionality concerning timing, length and address with its key bytes. The server's (ECU's) which transmit and receive messages are inside of this definition. The header format of a server (ECU) response message must be the same as of the client (tester) request message with the exception of the length information. For example, based on a client (tester) request message with format, target and source header information the server (ECU) must send a response message containing the same header information.

Note: Keyword 2000D is not allowed.

5.1.5.2 - Initialisation with 5 Baud address word

5.1.5.2.1 - ISO 9141-2 initialisation

For CARB purposes only the 5 baud initialisation is used. It is a functional initialisation. Description see ISO 14230 KWP 2000 Part 4: Requirements For Emission Related Systems or ISO 9141-2.
Messages are send to all emission related servers (ECUs).

5.1.5.2.2 - 5 baud initialisation

The general form of a 5 baud initialisation is shown in the figure below. A 5 Baud address byte is transferred from the client (tester) on the "K-Line" and on the "L-Line". After sending the 5 baud address byte the client (tester) will maintain the "L-Line" on high level.
After receiving the 5 baud address byte the server (ECU) will transmit the synchronisation pattern "$55" and the two key bytes with the actual communication baud rate. The client (tester) transmits key byte 2 (inverse), then the server (ECU) transmits the address byte (inverse).
In the case of physical initialisation the server (ECU) has to answer according to figure 8.
Using functional initialisation (that means that more than one server (ECU) is initialised) the vehicle manufacturer has to take care, that all server (ECU)s use the same option of the protocol. Only one server (ECU) has to perform the sequence of initialisation.

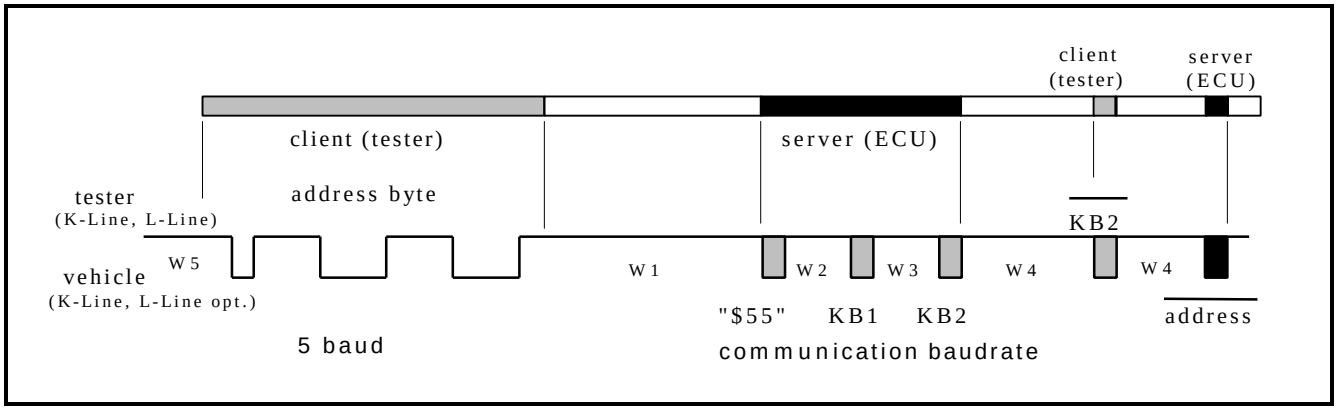


Figure 23 - 5 baud initialisation

Timing	Values in ms		
Parameter	**min.**	**max.**	**Description**
W1	60	300	Time from end of the address byte to start of synchronisation pattern
W2	5	20	Time from end of the synchronisation pattern to the start of key byte 1
W3	0	20	Time between key byte 1 and key byte 2
W4	25	50	Time between key byte 2 (from the server (ECU)) and its inversion from the client (tester). Also the time from the inverted key byte 2 from the client (tester) and the inverted address from the server (ECU)
W5	300	-	Time before the client (tester) starts to transmit the address byte

Table 5.1.5.2.2 - Timing values for 5 baud initialisation

These are fixed values. They cannot be changed by the accessTimingParameter service.
Key bytes as defined above. Baudrates from 1200 to 10400 Baud are allowed for communication. The client (tester) will recognise the baudrate from the synchronisation byte ($55).

5.1.5.2.3 - Functional initialisation

5.1.5.2.3.1 Functional initialisation between client (tester) and a group of servers (ECUs)

With this procedure a group of servers (ECUs) is initialised. This requires that the format byte of the startCommunication request message includes the value $C1. Once a successful functional initialisation between client (tester) and a group of servers (ECUs) has taken place the client (tester) may switch between functional and physical addressing in the request messages based on diagnostic requirements.

The following definitions apply during and after a functional initialisation:

- Bits A1, A0 of the format byte shall be set to '1, 1' if the request message is functionally addressed
- Bits A1, A0 of the format byte shall be set to '1, 0' for (a) response message(s) which is always a physically addressed message

During functional communication the target address of a request message has to be interpreted as a functional (group) address. The source address is the physical address of the client (tester). In response messages target and source address are always physical addresses.

- A functional initialisation of a group of servers (ECUs) requires each server (ECU) to send a physically addressed positive response message to the client (tester).
- A functional initialisation requires that the servers (ECUs) must support arbitration as specified in this document.
- A functional initialisation requires normal timing parameters with default values if more than one server (ECU) is connected.
- After a functional initialisation of more than one server (ECU) the accessTimingParameter service is not permitted.

Address bytes which define a functional group of server (ECU)s are listed in Appendix A.
Other manufacturer defined functional address bytes according to ISO 9141:1989 (i.e. odd parity) are possible.
Functional addressing is only possible, if all servers (ECUs) of a functional group use equal baud rates.
CARB initialisation is a special case of functional addressing.

5.1.5.2.3.2 Functional initialisation between client (tester) and gateway

A gateway server (ECU) configuration which connects on one side to a network and on the other side to the client (tester) shall behave as specified below during initialisation and communication. Server #1 (ECU #1) and server #n (ECU #n) are physically connected but not initialised.

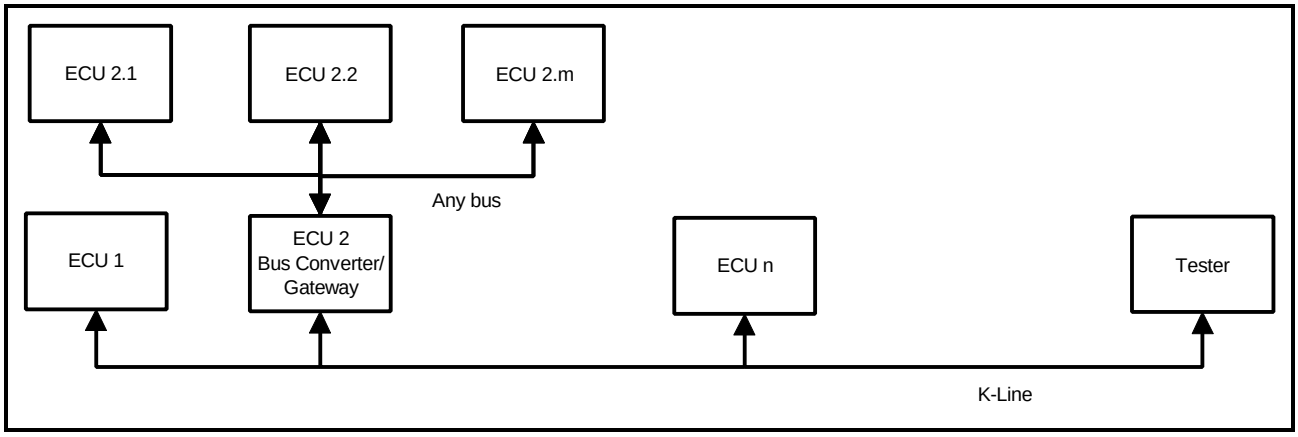


Figure 24 - Functional initialisation between client (tester) and gateway example

- A functional initialisation of a gateway server (ECU) which represents a group of servers (ECUs) in the network may send only one physically addressed positive response message to the client (tester) with the gateway source address instead of response messages from all servers (ECUs) with their own physical source address on the network.

- After initialisation the gateway may support two (2) cases of communication:
  case #1: The gateway may respond with a single or with multiple response messages on behalf of each server (ECU) of the network based on a request message from the client (tester) which includes the physical address of the network server (ECU) as the target address. In such case the gateway shall use the physical server (ECU) source address.
  case #2: The gateway may respond with a single or with multiple response messages based on a request message from the client (tester) which includes the physical address of the gateway. In such case the gateway shall use its own gateway address as the source address.
- The accessTimingParameter service shall be supported by the gateway server (ECU) in case of timing requirements which can not be fulfilled by the normal timing parameters with default values.

5.1.5.2.4 - Physical initialisation

With this procedure only a single server (ECU) is initialised.
5 baud initialisation address is according to ISO 9141:1989. Odd Parity is used. Address bytes are manufacturer controlled.

5.1.5.3 - Fast Initialisation

All servers (ECUs) which are initialised must use a baud rate of 10400 baud for initialisation and communication.
The client (tester) transmits a Wake up Pattern (WuP) on "K- and L-Line" synchronously. The pattern begins after an idle time on "K-line" with a low time of TiniL. The client (tester) transmits the first bit of the startCommunication service after a time of tWuP following the first falling edge.

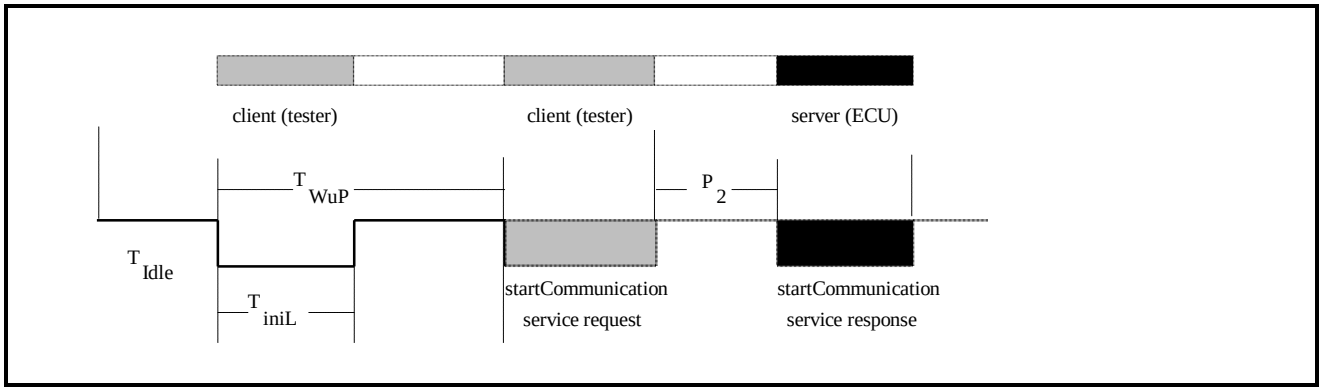


Figure 25 - Fast initialisation

Values of TWuP and TiniL are defined in the table below:

timing parameter	minimum timing value	timing value	maximum timing value
TiniL	24 ms	25 +/- 1 ms	26 ms
TWuP	49 ms	50 +/- 1 ms	51 ms

Table 5.1.5.3.1 - Timing values for fast initialisation

There are different possibilities for the idle time TIdle:
- First transmission after power on: TIdle = W5
- After completion of StopCommunication service: TIdle = P3min
- After stopping communication by timeout P3max: TIdle = 0 (starting with a falling edge)

The transfer of a Wake up Pattern as described above is followed by a startCommunication Request from the client (tester) and a response from the server (ECU). The first message of a fast initialisation always uses a header with target and source address and without additional length byte. A server (ECU) may answer back with or without address information and length byte and tells its supported modes within the key bytes. The pattern begins with a falling edge after an idle time on K-line. While communication is running it’s not necessary for a server (ECU) to react when receiving a WuP.

Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	Format byte physical addressing functional addressing	M	xx=[81 C1]	FMT
2	Target address byte	M	xx	TGT
3	Source address byte	M	xx	SRC
4	**startCommunication Request Service Id**	**M**	**81**	**STC**
5	Checksum	M	xx	CS

Table 5.1.5.3.2 - StartCommunication Request Message

Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	Format byte	M	xx	FMT
2	Target address byte	C1	xx	TGT
3	Source address byte	C1	xx	SRC
4	Additional length byte	C2	xx	LEN
5	**startCommunication Positive Response Service Id**	**S**	**C1**	**STCPR**
6	Key byte 1 (see 5.1.5.1 for use of key bytes) Key byte 2	M	xx xx	KB1 KB2
7	Checksum	M	xx	CS

Table 5.1.5.3.3 - StartCommunication Positive Response Message

C1: Format byte is 10xx xxxx or 11xx xxxx
C2: Format byte is xx00 0000.

5.2 - StopCommunication Service

5.2.1 - Service Definition

5.2.1.1 - Service Purpose

The purpose of this KWP 2000 communication layer service is to terminate a diagnostic communication.

5.2.1.2 - Service Table

StopCommunication Request	M
StopCommunication Positive Response	S
StopCommunication Negative Response	S
Response Code	M

Table 5.2.1.2 - StopCommunication Service

5.2.1.3 - Service Procedure

Upon receiving a stopCommunication indication primitive, the server (ECU) shall check if the current conditions allow to terminate this communication. In this case the server shall perform all actions necessary to terminate this communication.
If it is possible to terminate the communication, the server (ECU) shall issue a stopCommunication response primitive with the positive response parameters selected, before the communication is terminated. If the communication cannot be terminated by any reason, the server shall issue a StopCommunication response primitive with the negative response parameter selected.

5.2.2 - Implementation

Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	Format byte	M	xx	FMT
2	Target address byte	C1	xx	TGT
3	Source address byte	C1	xx	SRC
4	Additional length byte	C2	xx	LEN
5	**stopCommunication Request Service Id**	**M**	**82**	**SPC**
6	Checksum	M	xx	CS

Table 5.2.2.1 - StopCommunication Request Message

Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	Format byte	M	xx	FMT
2	Target address byte	C1	xx	TGT
3	Source address byte	C1	xx	SRC
4	Additional length byte	C2	xx	LEN
5	**stopCommunication Positive Response Service Id**	**S**	**C2**	**SPCPR**
6	Checksum	M	xx	CS

Table 5.2.2.2 - StopCommunication Positive Response Message

Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	Format byte	M	xx	FMT
2	Target address byte	C1	xx	TGT
3	Source address byte	C1	xx	SRC
4	Additional length byte	C2	xx	LEN
5	**negative Response Service Id**	**S**	**7F**	**SPCNR**
6	stopCommunication Request Service Identification	S	82	STC
7	ResponseCode*) = generalReject	M	xx = 10	RC
8	Checksum	M	xx	CS

Table 5.2.2.3 - StopCommunication Negative Response Message

***) Other response codes possible, see Keyword Protocol 2000 - Part 3: Implementation**
C1: Format byte is 10xx xxxx or 11xx xxxx
C2: Format byte is xx00 0000.

5.3 - AccessTimingParameter Service

5.3.1 - Service Definition

5.3.1.1 - Service Purpose

The purpose of this KWP 2000 communication layer service is to read and change the default timing parameters of a communication link for the duration this communication link is active.

Note: The use of this service is complex and depends on the server's (ECU's) capability and the physical topology in the vehicle. The user of this service is responsible for proper operation in the "K-line" with other servers (ECUs).

5.3.1.2 - Service Table

AccessTimingParameter Request	
Timing Parameter Identifier (TPI)	M
P2min	C1
P2max	C1
P3min	C1
P3max	C1
P4min	C1
AccessTimingParameter Positive Response	**S**
Timing Parameter Identifier (TPI)	M
P2min	C2
P2max	C2
P3min	C2
P3max	C2
P4min	C2
AccessTimingParameter Negative Response	**S**
Response Code	M
Timing Parameter Identifier (TPI)	M

Table 5.3.1.2 - AccessTimingParameter Service

C1: Condition is TPI = SetValues
C2: Condition is TPI = Read limits, read current values

5.3.1.3 - Service Procedure

This procedure has four (4) different modes:

- readLimitsOfPossibleTimingParameters
- setTimingParametersToDefaultValues
- readCurrentlyActiveTimingParameters
- setTimingParametersToGivenValues

Upon receiving an accessTimingParameter indication primitive with TPI = 00, the server (ECU) shall read the timing parameter limits, that is the values that the server (ECU) is capable of supporting.
If the read access to the timing parameter is successful, the server (ECU) shall send an accessTimingParameter response primitive with the positive response parameters.
If the read access to the timing parameters is not successful, the server (ECU) shall send an accessTimingParameter response primitive with the negative response parameters.
Upon receiving an accessTimingParameter indication primitive with TPI = 01, the server shall change all timing parameters to the default values and send an accessTimingParameter response primitive with the positive response parameters before the default timing parameters become active.
If the timing parameters cannot be changed to default values for any reason, the server (ECU) shall maintain the communication link and send an accessTimingParameter response primitive with the negative response parameters.
Upon receiving an accessTimingParameter indication primitive with TPI = 10, the server (ECU) shall read the currently used timing parameters.
If the read access to the timing parameters is successful, the server (ECU) shall send an accessTimingParameter response primitive with the positive response parameters.
If the read access to the currently used timing parameters is impossible for any reason, the server (ECU) shall send an accessTimingParameter response primitive with the negative response parameters.
Upon receiving an accessTimingParameter indication primitive with TPI = 11, the server (ECU) shall check if the timing parameters can be changed under the present conditions.
If the conditions are valid, the server (ECU) shall perform all actions necessary to change the timing parameters and send an accessTimingParameter response primitive with the positive response parameters before the new timing parameter limits become active.
If the timing parameters cannot be changed by any reason, the server (ECU) shall maintain the communication link and send an accessTimingParameter response primitive with the negative response parameters.

5.3.2 - Implementation

Selection of mode (read/write/current/limits) is by the Timing Parameter Identifier (TPI):

Hex	Description	Mnemonic
00	readLimitsOfPossibleTimingParameter	RLOPTP
01	setTimingParametersToDefaultValues	STPTDV
02	readCurrentlyActiveTimingParameters	RCATP
03	setTimingParametersToGivenValues	STPTGV
04 - FF	reservedByDocument These values are reserved by this document for future definition.	RBD

Table 5.3.2.1 - TimimgParameterIdentifier values

Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	Format byte	M	xx	FMT
2	Target address byte	C1	xx	TGT
3	Source address byte	C1	xx	SRC
4	Additional length byte	C2	xx	LEN
5	**AccessTimingParameter Request Service Id**	**S**	**83**	**ATP**
6	TimingParameterIdentifier = [readLimitsOfPossibleTimingParameter, setTimingParametersToDefaultValues, readCurrentlyActiveTimingParameters, setTimingParametersToGivenValues]	M	xx=[00, 01, 02, 03]	TPI_ RLOPTP STPTDV RCATP STPTGV
7	P2min	C3	xx	P2MIN
8	P2max	C3	:	P2MAX
9	P3min	C3	:	P3MIN
10	P3max	C3	:	P3MAX
11	P4min	C3	xx	P4MIN
12	Checksum	M	xx	CS

Table 5.3.2.2 - AccessTimingParameter Request Message

C1: Format byte is '10xx xxxx' or '11xx xxxx'
C2: Format byte is 'xx00 0000'
C3: Condition is valid if TPI = setTimingParametersToGivenValues

Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	Format byte	M	xx	FMT
2	Target address byte	C1	xx	TGT
3	Source address byte	C1	xx	SRC
4	Additional length byte	C2	xx	LEN
5	**AccessTimingParameter Positive Response Service Id**	**M**	**C3**	**ATPPR**
6	TimingParameterIdentifier = [readLimitsOfPossibleTimingParameter, setTimingParametersToDefaultValues, readCurrentlyActiveTimingParameters, setTimingParametersToGivenValues]	M	xx=[00, 01, 02, 03]	TPI_ RLOPTP STPTDV RCATP STPTGV
7	P2min	C4	xx	P2MIN
8	P2max	C4	:	P2MAX
9	P3min	C4	:	P3MIN
10	P3max	C4	:	P3MAX
11	P4min	C4	xx	P4MIN
12	Checksum	M	xx	CS

Table 5.3.2.3 - AccessTimingParameter Positive Response Message

C1: Format byte is '10xx xxxx' or '11xx xxxx'
C2: Format byte is 'xx00 0000'
C4: Condition is valid if TPI = readLimitsOfPossibleTimingParameter, readCurrentlyActiveTimingParameters

Byte	Parameter Name	CVT	Hex Value	Mnemonic
1	Format byte	M	xx	FMT
2	Target address byte	C1	xx	TGT
3	Source address byte	C1	xx	SRC
4	Additional length byte	C2	xx	LEN
5	**negative Response Service Id**	**S**	**7F**	**ATPNR**
6	AccessTimingParameter Request Service Id	S	83	ATP
7	ResponseCode*) = generalReject	M	xx = 10	RC
8	Checksum	M	xx	CS

Table 5.3.2.4 - AccessTimingParameter Negative Response Message

***) Other response codes are possible, see KWP 2000 - Implementation Recommended Practice**
C1: Format byte is '10xx xxxx' or '11xx xxxx'
C2: Format byte is 'xx00 0000'.

5.4 - SendData Service

5.4.1 - Service Definition

5.4.1.1 - Service Purpose

The purpose of this KWP 2000 communication layer service is to transmit the data from the service request over a KWP 2000 communication link.

5.4.1.2 - Service Table

SendData Request	**M**
Service Data	M
SendData Positive Response	**S**
SendData Negative Response	**S**
Response Code	M

Table 5.4.1.2 - SendData Service

5.4.1.3 - Service Procedure

Upon a SendData request from the application layer, the respective data link layer entity of the message transmitter will perform all actions necessary to transmit the parameters of the request by a KWP 2000 message. This includes the determination of the message header (incl. the format byte), the concatenation of the message data, the checksum calculation, idle recognition, the transmission of message bytes and the timing surveillance (arbitration).
Upon receiving a message over a KWP 2000 communication link, the respective data link layer entity of the message receiver will perform all actions necessary to provide the received information to the respective application layer. This includes the recognition of a message start, the timing surveillance, the reception of message bytes, a checksum check, segmenting of the message data based on the format information and delivery of the message data to the application layer with a SendData indication primitive.
If the service was successfully completed (i.e. the message was transmitted), a SendData response primitive with the positive response parameter selected is delivered from the data link layer entity of the transmitting device to the respective application layer entity.
If the service cannot be performed by the data link layer entity of the transmission device, a SendData response primitive with the negative response parameters selected is delivered to the respective application layer entity.

6 - Error Handling

6.1 - Error handling during physical/functional Fast Initialisation

6.1.1 - Client (tester) Error handling during physical/functional Fast Initialisation

Client (tester) detects an …	Action
… error in Tidle (W5 or P3min)	The client (tester) is responsible to keep to the idle time. The server (ECU) is responsible to keep to the time P1min. In case of an error the client (tester) must wait for Tidle again.
... error in P1min	No observation necessary. P1min is always 0 ms.
... error in P1max (P1max time-out)	The client (tester) shall ignore the response and shall open a new timing window P2 to receive a directly repeated response from the same server (ECU) or a response from another server (ECU). If the server (ECU) does not repeat the response, the client (tester) shall wait for P3max time-out and afterwards the client (tester) may start a new initialisation beginning with a wake up pattern (Tidle= 0 ms).
... error in P2min	No observation necessary. P2min is always 0 ms during initialisation.
... error in P2max (no valid response from any server (ECU))	If the client (tester) does not receive any response, the client (tester) shall wait for P3max time-out and afterwards may start a new initialisation beginning with a wake up pattern (Tidle= 0 ms).
... error in startCommunication positive response (Byte collision) (response contents) (response checksum)	The client (tester) shall ignore the response and shall open a new timing window P2 to receive a directly repeated response from the same server (ECU) or a response from another server (ECU). If the server (ECU) does not repeat the response, the client (tester) shall wait for P3max time-out and afterwards the client (tester) may start a new initialisation beginning with a wake up pattern (Tidle= 0 ms).

Table 6.1.1 - Client (tester) Error Handling during physical/functional Fast initialisation

6.1.2 - Server (ECU) Error Handling during physical Fast Initialisation

Server (ECU) detects an ...	Action
... error in Tidle (W5 or P3min)	No observation necessary. The client (tester) is responsible to keep to the idle time Tidle .
... error in wake-up-pattern	The server (ECU) shall not respond and shall be able to detect immediately a new wake-up pattern sequence.
... error in P4min	No observation necessary. The client (tester) is responsible to keep to the time P4min.
... error in P4max (P4max time-out)	The server (ECU) shall not respond and shall be able to detect immediately a new wake-up pattern sequence.
... error in startCommunication request (checksum, contents)	The server (ECU) shall not respond and shall be able to detect immediately a new wake-up pattern sequence.
... not allowed client (tester) source address or server (ECU) target address	The server (ECU) shall not respond and shall be able to detect immediately a new wake-up pattern sequence.

Table 6.1.2 - Server (ECU) Error Handling during physical initialisation

6.1.3 - Server (ECU) Error Handling during functional Fast Initialisation (normal timing only)

Server (ECU) detects an ...	Action
... error in Tidle (W5 or P3min)	No observation necessary. The client (tester) is responsible to keep to the idle time Tidle.
... error in wake-up-pattern	The server (ECU) shall not respond and shall be able to detect immediately a new wake-up pattern sequence.
... error in P4min	No observation necessary. The client (tester) is responsible to keep to the time P4min
... error in P4max (P4max time-out)	The server (ECU) shall not respond and shall be able to detect immediately a new wake-up pattern sequence.
... error in startCommunication request (checksum), (contents)	The server (ECU) shall not respond and shall be able to detect immediately a new wake-up pattern sequence.
... error in startCommunication Positive response, (Byte collision)	The server (ECU) shall repeat the response within a new timing window P2 considering arbitration.
... not allowed client (tester) target address or server (ECU) source address	The server (ECU) shall not respond and shall be able to detect immediately a new wake-up pattern sequence.

Table 6.1.3 - Server (ECU) Error Handling during functional initialisation

6.2 - Error handling after physical/functional Initialisation

6.2.1 - Client (tester) communication Error handling (after physical/functional Initialisation)

Client (tester) detects an ...	Action
... error in P1min	No observation necessary. P1min is always 0 ms.
... error in P1max (P1max time-out)	The client (tester) shall ignore the response and shall open a new timing window P2 to receive a directly repeated response from the same server (ECU) or a response from another server (ECU). If the server (ECU) does not repeat the response, the client (tester) may repeat the same request in a new timing window P3.
... error in P2min	No observation necessary. The server (ECU) is responsible to keep to the time P2min.
... error in P2max (no valid response from any server (ECU) or missing responses)	The client (tester) shall repeat the last request twice, each within in a new timing window P3 (i.e. three transmission total). Any following appropriate action is client (tester) dependent if the client (tester) does not receive a response.
... error in server (ECU) response (checksum), (contents)	The client (tester) shall repeat the last request twice, each within in a new timing window P3 (i.e. three transmission total). Any following appropriate action is client (tester) dependent if the client (tester) does not receive a response.

Table 6.2.1 - Client (tester) communication Error Handling after physical/functional initialisation

6.2.2 - Server (ECU) communication Error Handling after physical Initialisation

Server (ECU) detects an	Action
... error in P3min	No observation necessary. The client (tester) is responsible to keep to the time P3min.
... error in P3max (P3max time-out)	The server (ECU) shall reset communication and shall be able to detect immediately a new wake-up-pattern sequence.
... error in P4min	No observation necessary The client (tester) is responsible to keep to the time P4min
... error in P4max (P4max time-out)	The server (ECU) shall ignore the request and shall open a new timing window P3 to receive a new request from the client (tester).
... error in client (tester) request (header or checksum)	The server (ECU) shall ignore the request and shall open a new timing window P3 to receive a new request from the client (tester).
...not allowed source or target address	The server (ECU) shall ignore the request and shall open a new timing window P3 to receive a new request from the client (tester).

Table 6.2.2 - Server (ECU) communication Error Handling after physical initialisation

6.2.3 - Server (ECU) Error Handling after functional Initialisation

Server (ECU) detects an ...	Action
... error in P3min	No observation necessary. The client (tester) is responsible to keep to the time P3min.
... error in P3max (P3max time-out)	The server (ECU) shall reset communication and shall be able to detect immediately a new wake-up-pattern sequence.
... error in P4min	No observation necessary. The client (tester) is responsible to keep to the time P4min.
... error in P4max (P4max time-out	The server (ECU) shall ignore the request and shall open a new timing window P3 to receive a new request from the client (tester).
... error in client (tester) request (header or checksum)	The server (ECU) shall ignore the request and shall open a new timing window P3 to receive a new request from the client (tester).
...not allowed source or target address	The server (ECU) shall ignore the request and shall open a new timing window P3 to receive a new request from the client (tester).
... error in its own Response (byte collision)	If the server (ECU) detects a byte collision within its own response, it must repeat the response within a new timing window P2 considering the arbitration.

Table 6.2.3 - Server (ECU) Error Handling after functional initialisation

7 - Arbitration

Arbitration is required during wake up and during normal communications, when more than one responding server (ECU) is involved (functional addressing). A mechanism is required for each case, and these will be discussed separately.

Before discussing the arbitration mechanisms in detail an explanation follows about some general concepts and terms:

- Collision avoidance
- Collision detection
- Protocol timing

7.1 - Collision Avoidance

In order to avoid collisions, a server (ECU) will monitor any bus activity before starting a transmission. If a server (ECU) detects bus activity within a specific time window arbitration is lost.

The better the resolution of the bus activity detection, the less likely is a collision. Activity detection can be performed either byte-wise, using a reception, or at a bitwise level using start bit detection.
Looking for a falling edge of a start bit provides the best bus activity detection resolution.

7.2 - Collision detection

The two possible bit levels are represented by a dominant and a recessive bit. If two ore more servers (ECUs) transmit different bits simultaneously the result is a destructive collision. The servers (ECUs) transmitting recessive bits will see dominant bits. The servers (ECUs) transmitting dominant bits will see also the dominant bits, and will not recognise that a collision has occurred.
A requirement of this arbitration mechanism is that a transmitting server (ECU) shall be able to monitor its own transmissions. By comparing what it receives (feedback from K-Line) with what it transmits, destructive collisions can be detected. When a difference exists then a collision has occurred.
Transmissions can be monitored for collision at the bit or byte level. Whichever mechanism is implemented, all transmissions must be monitored. The action required in the event of a collision is discussed in section 7.3.2.

7.3 - Protocol timing

Timing parameters are defined in this document. For functional addressing, which requires the arbitration mechanism , only „normal timing parameters“ shall be used.

7.3.1 - Arbitration during Wake Up

7.3.1.1 - 5 baud initialisation

Following the reception of the address byte, those servers (ECUs) recognising the address will attempt to response by transmitting a $55 byte.
The point in time at which the $55 byte is transmitted is selected randomly within W1: a time between 60 and 300 ms. This time is termed W1random and each server (ECU) will calculate its own value for each wake up.
If during W1random bus activity is detected then arbitration is lost and the server (ECU) validates the rest of the wake up.
If a server (ECU) detects an error during wake up then it looks for the next address byte.

7.3.1.2 - Fast initialisation

See section 7.3.2 - Arbitration during normal communication

7.3.2 - Arbitration during normal communication

Normal communication takes place following a successful wake up and before P3max (5000ms). The client (tester) transmits request messages to which all servers (ECUs) have to transmit a response message.
A server (ECU) may transmit its response message when no bus activity has been detected for some back-off period. The back-off period is in the range defined by P2 and needs to be different for each server (ECU).
If bus activity is detected before the back-off period has elapsed then arbitration is lost. The back-off period timing starts again after the end of the message.
Destructive collisions and corruptions are detected by recognising a difference between the transmitted and received bytes. On the detection of a corruption the transmission of the response message is aborted and the back-off period is adjusted.
Following the detection of an error a server (ECU) may choose to re-transmit its response after P2min (25 ms) and before P2max (50 ms) has elapsed. The client (tester) may not re-transmit the request before a bus inactive period of P3min (55 ms).

Appendix A - ECU/Tester addresses for 5 baud initialisation

This section is part of the International Standard.

Physical Addresses:

- According to ISO 9141:1989
- Address byte consists of 1 start bit, 7 bit address, 1 parity bit (odd parity), at least 1 stop bit.
- Addresses are controlled by car manufacturers.

Functional Addresses:

- Addresses with values < $80 are reserved for future standardisation.
- Addresses with values > $80 are manufacturer specific.

Appendix B - ECU/Tester addresses for fast initialisation

This section is for information only.

Addresses may be the same as used for 5 baud initialisation (see Appendix A) or according to SAE J2178 Part 1.

Powertrain Controllers	**Hex**
Integration/Manufacturer Expansion	00 - 0F
Engine Controllers	10 - 17
Transmission Controllers	18 - 1F
Chassis Controllers	
Integration/Manufacturer Expansion	20 - 27
Brake Controllers	28 - 2F
Steering Controllers	30 - 37
Suspension Controllers	38 - 3F
Body Controllers	
Integration/Manufacturer Expansion	40 - 57
Restraints	58 - 5F
Driver Information/Displays	60 - 6F
Lightning	70 - 7F
Entertainment/Audio	80 - 8F
Personal Communication	90 - 97
Climate Control (HVAC)	98 - 9F
Convenience (Doors,Seats,Windows,etc)	A0 - BF
Security	C0 - C7
Future Expansion:	C8 - CF
Manufacturer Specific	D0 - EF
Off-Board Tester/Diagnostic Tools:	F0 - FD
All Nodes	FE
Null Nodes	FF

Table B.1 - ECU/Tester Addresses for fast initialisation

Appendix C - Data stream examples according to section 4.4

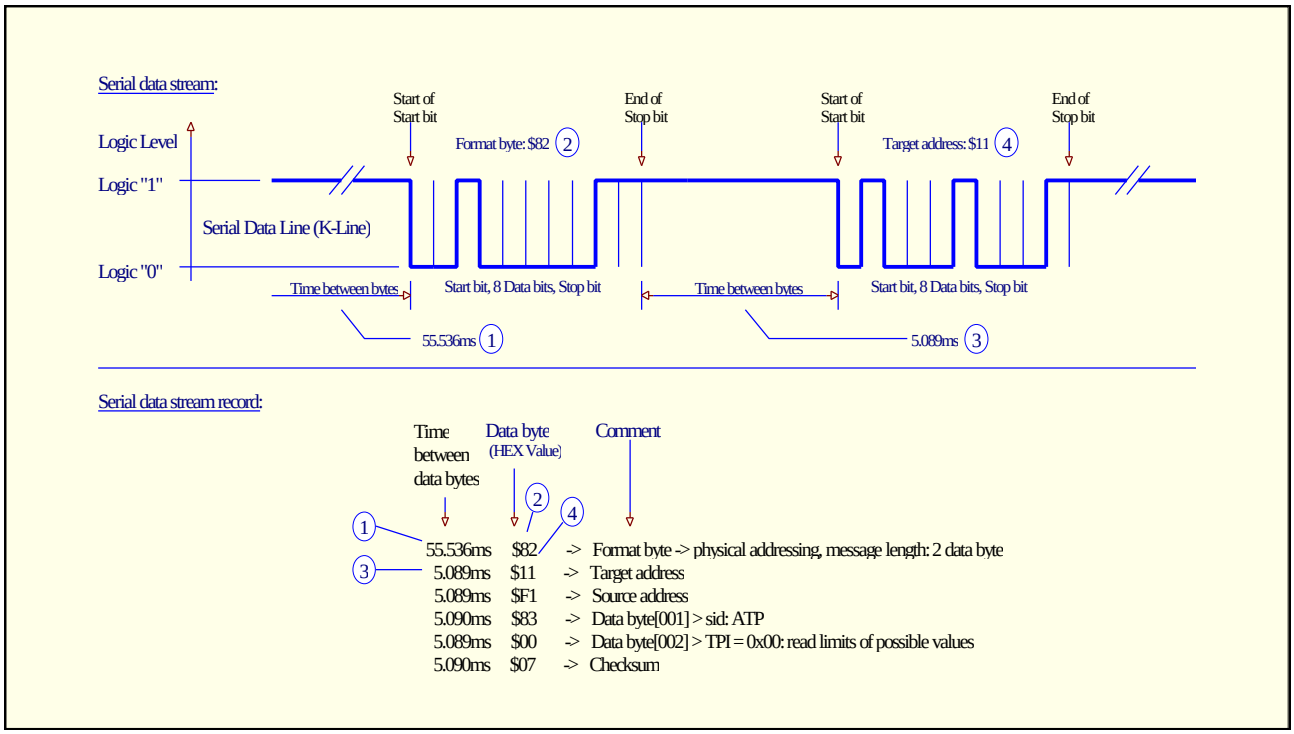


Figure C.1 - Explanation of a serial data stream record

Above figure is an explanation example of the interpretation of a sample serial data stream record. It shall clarify the relation between the time stamp, byte and comment. The time stamp of each byte indicates the time expired between the previous byte and the byte which the time stamp relates to (end of stop bit of previous byte and start of start bit of next byte). The byte is in hexadecimal ($) notation. Each byte has its own protocol / message specific comment. The number in circles (①, , ,) reference the appropriate section in the serial data stream record.

Physical Initialisation - single response message
case #1: Protocol: Key bytes = 2031Dec. Timing: Normal timing with default values

<REQUEST> ATP - accessTimingParameters

55.536ms $82 -> Format byte -> physical addressing, message length: 2 data byte
5.089ms $11 -> Target address
5.089ms $F1 -> Source address
5.090ms $83 -> Data byte[001] -> SID: ATP
5.089ms $00 -> Data byte[002] -> TPI = 0x00: read limits of possible values
5.090ms $07 -> Checksum

<POSITIVE RESPONSE> ATPPR - accessTimingParameterPositiveResponse

32.086ms $87 -> Format byte -> physical addressing, message length: 7 data byte
0.001ms $F1 -> Target address
0.002ms $11 -> Source address
0.002ms $C3 -> Data byte[001] -> SID: ATPPR
0.001ms $00 -> Data byte[002] -> TPI = 0x00: read limits of possible values
0.003ms $00 -> Data byte[003] -> P2min = 0x00 = 0.00ms (0.5ms Res.)
0.001ms $FE -> Data byte[004] -> P2max = 0xFE = 6350.00ms (25ms Res.)
0.002ms $01 -> Data byte[005] -> P3min = 0x01 = 0.50ms (0.5ms Res.)
0.002ms $28 -> Data byte[006] -> P3max = 0x28 = 10000.00ms (250ms Res.)
0.002ms $00 -> Data byte[007] -> P4min = 0x00 = 0.00ms (0.5ms Res.)
0.001ms $73 -> Checksum

case #2: Protocol: Key bytes = 2031Dec. Timing: Normal timing with default values

<REQUEST> RRRBLI - requestRoutineResultsByLocalIdentifier

2965.343ms $82 -> Format byte - > physical addressing + message length: 2 data byte
5.089ms $11 -> Target address
5.089ms $F1 -> Source address
5.090ms $33 -> Data byte[001]-> sid: RRRBLI
5.089ms $01 -> Data byte[002]
5.090ms $B8 -> Checksum

<NEGATIVE RESPONSE> NACK - noAcknowledge (responseCode = $23)

32.744ms $83 -> Format byte -> physical addressing + message length: 3 data byte
0.001ms $F1 -> Target address
0.002ms $11 -> Source address
0.002ms $7F -> Data byte[001]-> response SID: NACK
0.002ms $33 -> Data byte[002]-> request SID: RRRBLI
0.001ms $23 -> Data byte[003]-> response code: $23 -> routineNotComplete
0.001ms $5A -> Checksum

case #3: Protocol: Key bytes = 2031Dec. Timing: Normal timing with default values

<REQUEST> SPC - stopCommunication

55.456ms $81 -> Format byte -> physical addressing + message length: 1 data byte
5.089ms $11 -> Target address
5.090ms $F1 -> Source address
5.089ms $82 -> Data byte[001]-> SID: SPC
5.089ms $05 -> Checksum

<POSITIVE RESPONSE> SPC - stopCommunicationPositiveResponse

38.695ms $81 -> Format byte -> physical addressing + message length: 1 data byte
0.001ms $F1 -> Target address
0.003ms $11 -> Source address
0.001ms $C2 -> Data byte[001]-> SID: SPCPR
0.002ms $45 -> Checksum

case #4: Protocol: Key bytes = 2031Dec. Timing: Normal timing with default values
P2min = 0 ms; P2max = 6350ms; P3min = 50ms; P3max = 10000ms; P4min = 0ms

<REQUEST> IOCBLI - inputOutputControlByLocalIdentifier

55.190ms $84 -> Format byte - > physical addressing + message length: 4 data byte
5.088ms $11 -> Target address
5.089ms $F1 -> Source address
5.089ms $30 -> Data byte[001]-> SID: IOCBLI
5.090ms $32 -> Data byte[002]
5.089ms $08 -> Data byte[003]
5.089ms $BB -> Data byte[004]
5.090ms $AB -> Checksum

<NEGATIVE RESPONSE> NACK - noAcknowledge (responseCode = $78)

25.266ms $80 -> Format byte - > physical addressing + message length: add.length byte required
0.002ms $F1 -> Target address
0.002ms $11 -> Source address
0.002ms $03 -> Length byte - > message length: 3 data byte
0.001ms $7F -> Data byte[001]-> response SID: NACK
0.002ms $30 -> Data byte[002]-> request SID: IOCBLI
0.001ms $78 -> Data byte[003]-> responseCode: $78 -> requestCorrectlyReceived - ResponsePending
0.002ms $AC -> Checksum

<NEGATIVE RESPONSE> NACK - noAcknowledge (responseCode = $78)

9525.266ms $80 -> Format byte - > physical addressing + message length: add. length byte required
0.002ms $F1 -> Target address
0.002ms $11 -> Source address
0.002ms $03 -> Length byte - > message length: 3 data byte
0.001ms $7F -> Data byte[001]-> response SID: NACK
0.002ms $30 -> Data byte[002]-> request SID: IOCBLI
0.001ms $78 -> Data byte[003]-> responseCode: $78 -> requestCorrectlyReceived - ResponsePending
0.002ms $AC -> Checksum

<NEGATIVE RESPONSE> NACK - noAcknowledge (responseCode = $78)

9525.266ms $80 -> Format byte - > physical addressing + message length: add. length byte required
0.002ms $F1 -> Target address
0.002ms $11 -> Source address
0.002ms $03 -> Length byte - > message length: 3 data byte
0.001ms $7F -> Data byte[001]-> response SID: NACK
0.002ms $30 -> Data byte[002]-> request SID: IOCBLI
0.001ms $78 -> Data byte[003]-> responseCode: $78 -> requestCorrectlyReceived - ResponsePending
0.002ms $AC -> Checksum

<POSITIVE RESPONSE> IOCBLIPR - inputOutputControlByLocalIdentifierPositiveResponse

9527.375ms $80 -> Format byte -> physical addressing + message length: add. length byte required
0.002ms $F1 -> Target address
0.002ms $11 -> Source address
0.002ms $05 -> Length byte - > message length: 5 data byte
0.001ms $70 -> Data byte[001]-> SID: IOCBLIPR.
0.002ms $32 -> Data byte[002]
0.002ms $08 -> Data byte[003]
0.002ms $32 -> Data byte[004]
0.001ms $08 -> Data byte[005]
0.002ms $6B -> Checksum

Physical Initialisation - periodic transmissions
(message flow A, normal Timing, default timing values)

STEP#1 startDiagnosticSession(DM_PeriodicTransmission)

<REQUEST> STDS - startDiagnosticSession

55.456ms $82 -> Format byte -> physical addressing + message length: 2 data byte
5.089ms $11 -> Target address
5.090ms $F1 -> Source address
5.089ms $10 -> Data byte[001]-> sid: STDS
5.089ms $82 -> Data byte[002]-> diagnosticMode: standardDiagnosticModeWithPeriodicTransmission
5.089ms $16 -> Checksum

<POSITIVE RESPONSE> STDSPR - startDiagnosticSessionPositiveResponse #1

25.695ms $82 -> Format byte -> physical addressing + message length: 1 data byte
0.001ms $F1 -> Target address
0.003ms $11 -> Source address
0.001ms $50 -> Data byte[001]-> sid: STDSPR
0.001ms $82 -> Data byte[002]-> diagnosticMode: standardDiagnosticModeWithPeriodicTransmission
0.002ms $56 -> Checksum

***** standardDiagnosticModeWithPeriodicTransmission enabled *****
standardDiagnosticModeWithPeriodicTransmission default timing values active

<POSITIVE RESPONSE> STDSPR - startDiagnosticSessionPositiveResponse #2

25.695ms $82 -> Format byte -> physical addressing + message length: 2 data byte
0.001ms $F1 -> Target address
0.003ms $11 -> Source address
0.001ms $50 -> Data byte[001]-> sid: STDSPR
0.001ms $82 -> Data byte[002]-> diagnosticMode: standardDiagnosticModeWithPeriodicTransmission
0.002ms $56 -> Checksum

- •
- •
- •
- •

<POSITIVE RESPONSE> STDSPR - startDiagnosticSessionPositiveResponse #n

25.695ms $82 -> Format byte -> physical addressing + message length: 2 data byte
0.001ms $F1 -> Target address
0.003ms $11 -> Source address
0.001ms $50 -> Data byte[001]-> sid: STDSPR
0.001ms $82 -> Data byte[002]-> diagnosticMode: standardDiagnosticModeWithPeriodicTransmission
0.002ms $56 -> Checksum

STEP#2 e.g. readDataByLocalIdentifier

<REQUEST> RDBLI - readDataByLocalIdentifier

5.456ms $82 -> Format byte -> physical addressing + message length: 2 data byte
5.089ms $11 -> Target address
5.090ms $F1 -> Source address
5.089ms $21 -> Data byte[001]-> sid: RDBLI
5.089ms $01 -> Data byte[002]-> recordLocalIdentifier:
5.089ms $A6 -> Checksum

<RESPONSE> RDBLIPR - readDataByLocalIdentifierPositiveResponse #1

25.148ms $89 -> Format byte -> physical addressing + message length: 9 data byte
0.001ms $F1 -> target address
0.002ms $11 -> source address
0.001ms $61 -> Data byte[001]-> sid: RDBLIPR
0.003ms $02 -> Data byte[002]-> recordLocalIdentifier:
0.003ms $00 -> Data byte[003]
0.003ms $00 -> Data byte[004]
0.003ms $00 -> Data byte[005]
0.003ms $00 -> Data byte[006]
0.003ms $00 -> Data byte[007]
0.003ms $00 -> Data byte[008]
0.003ms $00 -> Data byte[009]
0.002ms $EE -> Checksum

<RESPONSE> RDBLIPR - readDataByLocalIdentifierPositiveResponse #2

25.148ms $89 -> Format byte -> physical addressing + message length: 9 data byte
0.001ms $F1 -> target address
0.002ms $11 -> source address
0.001ms $61 -> Data byte[001]-> sid: RDBLIPR
0.003ms $02 -> Data byte[002]-> recordLocalIdentifier:
0.003ms $00 -> Data byte[003]
0.003ms $00 -> Data byte[004]
0.003ms $00 -> Data byte[005]
0.003ms $00 -> Data byte[006]
0.003ms $00 -> Data byte[007]
0.003ms $00 -> Data byte[008]
0.003ms $00 -> Data byte[009]
0.002ms $EE -> Checksum

- • • •

<RESPONSE> RDBLIPR - readDataByLocalIdentifierPositiveResponse #n

25.148ms $89 -> Format byte -> physical addressing + message length: 9 data byte
0.001ms $F1 -> target address
0.002ms $11 -> source address
0.001ms $61 -> Data byte[001]-> sid: RDBLIPR
0.003ms $02 -> Data byte[002]-> recordLocalIdentifier:
0.003ms $00 -> Data byte[003]
0.003ms $00 -> Data byte[004]
0.003ms $00 -> Data byte[005]
0.003ms $00 -> Data byte[006]
0.003ms $00 -> Data byte[007]
0.003ms $00 -> Data byte[008]
0.003ms $00 -> Data byte[009]
0.002ms $EE -> Checksum

STEP#3 e.g. stopDiagnosticSession

<REQUEST> SPDS - stopDiagnosticSession

5.456ms $81 -> Format byte -> physical addressing + message length: 1 data byte
5.089ms $11 -> Target address
5.090ms $F1 -> Source address
5.089ms $20 -> Data byte[001]-> sid: SPDS
5.089ms $A3 -> Checksum

<POSITIVE RESPONSE> SPDS - stopDiagnosticSessionPositiveResponse

25.695ms $81 -> Format byte -> physical addressing + message length: 1 data byte
0.001ms $F1 -> Target address
0.003ms $11 -> Source address
0.001ms $60 -> Data byte[001]-> sid: SPDSPR
0.002ms $E3 -> Checksum

***** standardDiagnosticModeWithPeriodicTransmission disabled *****
default diagnostic session enabled and normal timing default values active

STEP#4 any request

<REQUEST> ???? - any request
55.456ms $81 -> Format byte -> physical addressing + message length: 1 data byte
5.089ms $11 -> Target address
5.090ms $F1 -> Source address

Physical Initialisation - periodic transmissions
(message flow B, normal Timing, modified timing values)

STEP#1 startDiagnosticSession(DM_PeriodicTransmission)
<REQUEST> STDS - startDiagnosticSession

55.456ms $82 -> Format byte -> physical addressing + message length: 2 data byte
5.089ms $11 -> Target address
5.090ms $F1 -> Source address
5.089ms $10 -> Data byte[001]-> sid: STDS
5.089ms $82 -> Data byte[002]-> diagnosticMode: standardDiagnosticModeWithPeriodicTransmission
5.089ms $16 -> Checksum

<POSITIVE RESPONSE> STDSPR - startDiagnosticSessionPositiveResponse #1

25.695ms $82 -> Format byte -> physical addressing + message length: 1 data byte
0.001ms $F1 -> Target address
0.003ms $11 -> Source address
0.001ms $50 -> Data byte[001]-> sid: STDSPR
0.001ms $82 -> Data byte[002]-> diagnosticMode: standardDiagnosticModeWithPeriodicTransmission
0.002ms $56 -> Checksum

***** standardDiagnosticModeWithPeriodicTransmission enabled *****
standardDiagnosticModeWithPeriodicTransmission default timing values active

<POSITIVE RESPONSE> STDSPR - startDiagnosticSessionPositiveResponse #2

25.695ms $82 -> Format byte -> physical addressing + message length: 2 data byte
0.001ms $F1 -> Target address
0.003ms $11 -> Source address
0.001ms $50 -> Data byte[001]-> sid: STDSPR
0.001ms $82 -> Data byte[002]-> diagnosticMode: standardDiagnosticModeWithPeriodicTransmission
0.002ms $56 -> Checksum

- •
- •

<POSITIVE RESPONSE> STDSPR - startDiagnosticSessionPositiveResponse #n

25.695ms $82 -> Format byte -> physical addressing + message length: 2 data byte
0.001ms $F1 -> Target address
0.003ms $11 -> Source address
0.001ms $50 -> Data byte[001]-> sid: STDSPR
0.001ms $82 -> Data byte[002]-> diagnosticMode: standardDiagnosticModeWithPeriodicTransmission
0.002ms $56 -> Checksum

STEP#2 accessTimingParameters(readLimitsOfPossibleValues)
<REQUEST> ATP - accessTimingParameters

5.536ms $82 -> Format byte -> physical addressing, message length: 2 data byte
5.089ms $11 -> Target address
5.089ms $F1 -> Source address
5.090ms $83 -> Data byte[001] -> sid: ATP
5.089ms $00 -> Data byte[002] -> TPI = 0x00: read limits of possible values
5.090ms $07 -> Checksum

<POSITIVE RESPONSE> ATPPR - AccessTimingParameterPositiveResponse #1

25.086ms $87 -> Format byte -> physical addressing, message length: 7 data byte
0.001ms $F1 -> Target address
0.002ms $11 -> Source address
0.002ms $C3 -> Data byte[001] -> sid: ATPPR
0.001ms $00 -> Data byte[002] -> TPI = 0x00: read limits of possible values
0.003ms $14 -> Data byte[003] -> P2min = 0x14 = 10.00ms (0.5ms Res.)
0.001ms $02 -> Data byte[004] -> P2max = 0x02 = 50.00ms (25ms Res.)
0.002ms $00 -> Data byte[005] -> P3min = 0x00 = 0.00ms (0.5ms Res.)
0.002ms $14 -> Data byte[006] -> P3max = 0x14 = 5000.00ms (250ms Res.)
0.002ms $00 -> Data byte[007] -> P4min = 0x00 = 0.00ms (0.5ms Res.)
0.001ms $76 -> Checksum

- •
- •

<POSITIVE RESPONSE> ATPPR - AccessTimingParameterPositiveResponse #n

25.086ms $87 -> Format byte -> physical addressing, message length: 7 data byte
0.001ms $F1 -> Target address
0.002ms $11 -> Source address
0.002ms $C3 -> Data byte[001] -> sid: ATPPR
0.001ms $00 -> Data byte[002] -> TPI = 0x00: read limits of possible values
0.003ms $14 -> Data byte[003] -> P2min = 0x14 = 10.00ms (0.5ms Res.)
0.001ms $02 -> Data byte[004] -> P2max = 0x02 = 50.00ms (25ms Res.)
0.002ms $00 -> Data byte[005] -> P3min = 0x00 = 0.00ms (0.5ms Res.)
0.002ms $14 -> Data byte[006] -> P3max = 0x14 = 5000.00ms (250ms Res.)
0.002ms $00 -> Data byte[007] -> P4min = 0x00 = 0.00ms (0.5ms Res.)
0.001ms $76 -> Checksum

STEP#3 accessTimingParameters(setParameters)
<REQUEST> ATP - accessTimingParameters

5.536ms $87 -> Format byte -> physical addressing, message length: 2 data byte
5.089ms $11 -> Target address
5.089ms $F1 -> Source address
5.002ms $83 -> Data byte[001] -> sid: ATP
5.001ms $03 -> Data byte[002] -> TPI = 0x03: set parameters
5.003ms $14 -> Data byte[003] -> P2min = 0x14 = 10.00ms (0.5ms Res.)
5.001ms $02 -> Data byte[004] -> P2max = 0x02 = 50.00ms (25ms Res.)
5.002ms $00 -> Data byte[005] -> P3min = 0x00 = 0.00ms (0.5ms Res.)
5.002ms $14 -> Data byte[006] -> P3max = 0x14 = 5000.00ms (250ms Res.)
5.002ms $00 -> Data byte[007] -> P4min = 0x00 = 0.00ms (0.5ms Res.)
5.001ms $39 -> Checksum

<POSITIVE RESPONSE> ATPPR - AccessTimingParameterPositiveResponse #1

25.536ms $82 -> Format byte -> physical addressing, message length: 2 data byte
0.089ms $F1 -> Target address
0.089ms $11 -> Source address
0.002ms $C3 -> Data byte[001] -> sid: ATPPR
0.001ms $03 -> Data byte[002] -> TPI = 0x03: set parameters
0.001ms $4A -> Checksum

***** standardDiagnosticModeWithPeriodicTransmission modified timing values active *****

<POSITIVE RESPONSE> ATPPR - AccessTimingParameterPositiveResponse #2

10.536ms $82 -> Format byte -> physical addressing, message length: 2 data byte
0.089ms $F1 -> Target address
0.089ms $11 -> Source address
0.002ms $C3 -> Data byte[001] -> sid: ATPPR
0.001ms $03 -> Data byte[002] -> TPI = 0x03: set parameters
0.001ms $4A -> Checksum

- ● ●
- ● ●

<POSITIVE RESPONSE> ATPPR - AccessTimingParameterPositiveResponse #5

10.536ms $82 -> Format byte -> physical addressing, message length: 2 data byte
0.089ms $F1 -> Target address
0.089ms $11 -> Source address
0.002ms $C3 -> Data byte[001] -> sid: ATPPR
0.001ms $03 -> Data byte[002] -> TPI = 0x03: set parameters
0.001ms $4A -> Checksum

STEP#5 e.g. readDataByLocalIdentifier
<REQUEST> RDBLI - readDataByLocalIdentifier

2.456ms $82 -> Format byte -> physical addressing + message length: 2 data byte
0.089ms $11 -> Target address
0.090ms $F1 -> Source address
0.089ms $21 -> Data byte[001]-> sid: RDBLI
0.089ms $01 -> Data byte[002]-> recordLocalIdentifier:
0.089ms $A6 -> Checksum

<RESPONSE> RDBLIPR - readDataByLocalIdentifierPositiveResponse #1

10.148ms $89 -> Format byte -> physical addressing + message length: 9 data byte
0.001ms $F1 -> target address
0.002ms $11 -> source address
0.001ms $61 -> Data byte[001]-> sid: RDBLIPR
0.003ms $02 -> Data byte[002]-> recordLocalIdentifier:
0.003ms $00 -> Data byte[003]
0.003ms $00 -> Data byte[004]
0.003ms $00 -> Data byte[005]
0.003ms $00 -> Data byte[006]
0.003ms $00 -> Data byte[007]
0.003ms $00 -> Data byte[008]
0.003ms $00 -> Data byte[009]
0.002ms $EE -> Checksum

<RESPONSE> RDBLIPR - readDataByLocalIdentifierPositiveResponse #2

10.148ms $89 -> Format byte -> physical addressing + message length: 9 data byte
0.001ms $F1 -> target address
0.002ms $11 -> source address
0.001ms $61 -> Data byte[001]-> sid: RDBLIPR
0.003ms $02 -> Data byte[002]-> recordLocalIdentifier:
0.003ms $00 -> Data byte[003]
0.003ms $00 -> Data byte[004]
0.003ms $00 -> Data byte[005]
0.003ms $00 -> Data byte[006]
0.003ms $00 -> Data byte[007]
0.003ms $00 -> Data byte[008]
0.003ms $00 -> Data byte[009]
0.002ms $EE -> Checksum

- •
- •

<RESPONSE> RDBLIPR - readDataByLocalIdentifierPositiveResponse #n

10.148ms $89 -> Format byte -> physical addressing + message length: 9 data byte
0.001ms $F1 -> target address
0.002ms $11 -> source address
0.001ms $61 -> Data byte[001]-> sid: RDBLIPR
0.003ms $02 -> Data byte[002]-> recordLocalIdentifier:
0.003ms $00 -> Data byte[003]
0.003ms $00 -> Data byte[004]
0.003ms $00 -> Data byte[005]
0.003ms $00 -> Data byte[006]
0.003ms $00 -> Data byte[007]
0.003ms $00 -> Data byte[008]
0.003ms $00 -> Data byte[009]
0.002ms $EE -> Checksum

STEP#6 e.g. stopDiagnosticSession
<REQUEST> SPDS - stopDiagnosticSession

2.456ms $81 -> Format byte -> physical addressing + message length: 1 data byte
0.089ms $11 -> Target address
0.090ms $F1 -> Source address
0.089ms $20 -> Data byte[001]-> sid: SPDS
0.089ms $A3 -> Checksum

<POSITIVE RESPONSE> SPDS - stopDiagnosticSessionPositiveResponse

10.695ms $81 -> Format byte -> physical addressing + message length: 1 data byte
0.001ms $F1 -> Target address
0.003ms $11 -> Source address
0.001ms $60 -> Data byte[001]-> sid: SPDSPR
0.002ms $E3 -> Checksum

***** standardDiagnosticModeWithPeriodicTransmission disabled *****
default diagnostic session enabled and normal timing default values active

STEP#7 any request
<REQUEST> ???? - any request
55.456ms $81 -> Format byte -> physical addressing + message length: 1 data byte
5.089ms $11 -> Target address
5.090ms $F1 -> Source address
:
:
:

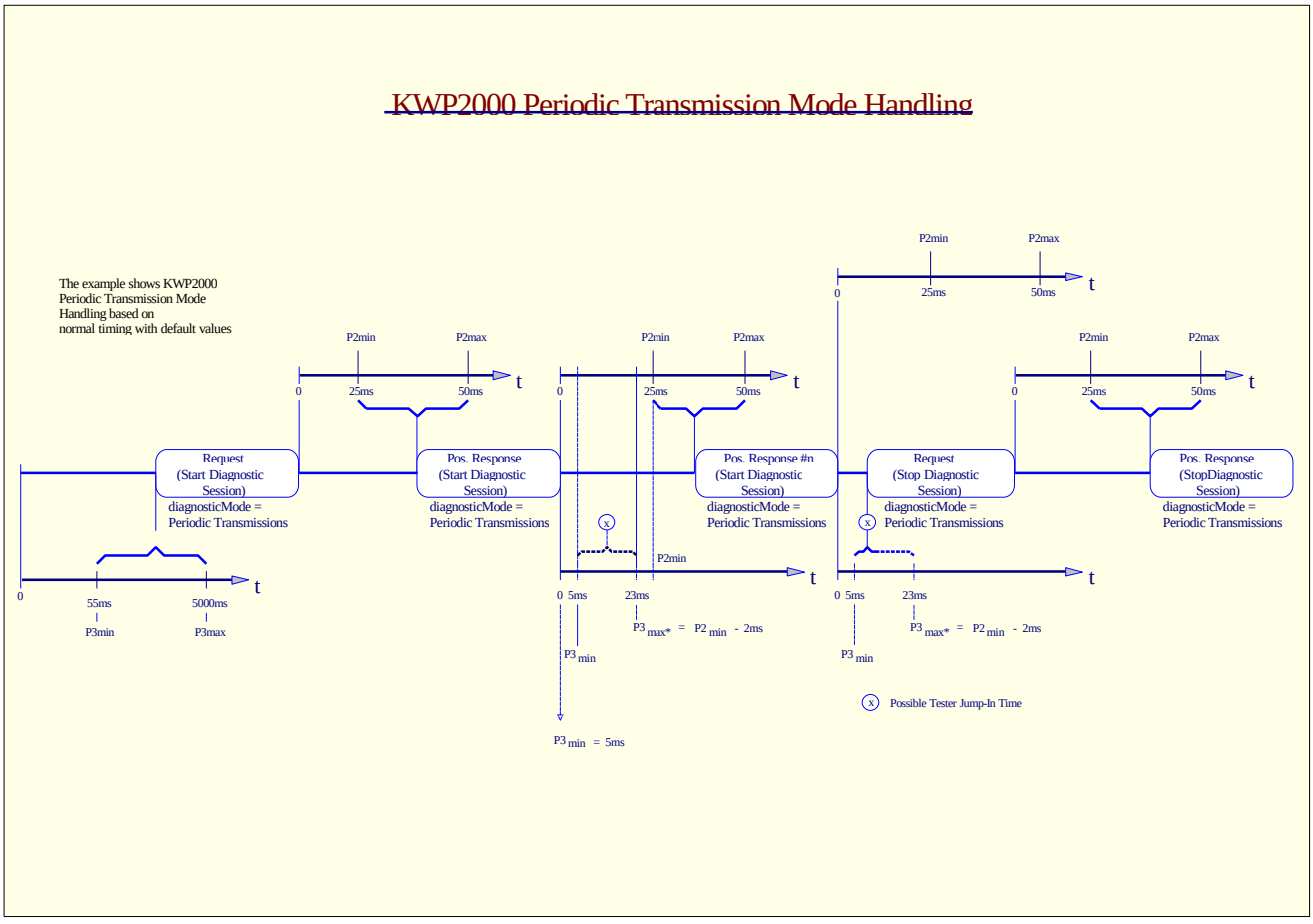


Figure C.2 - Keyword Protocol 2000 Periodic Transmission Mode Handling

Appendix D - StartDevelopmentModeCommunication

The startDevelopmentModeCommunication service provides the possibility to initialise protocols different to Keyword Protocol 2000 in a similar way.
The startDevelopmentModeCommunication service shall only be implemented for the purpose of initialising protocols which support functionality not within the scope of Keyword Protocol 2000. This shall only be allowed in mutual agreement between the vehicle manufacturer and the system supplier, e.g. manufacturer calibration.
All servers (ECUs) which are initialised must use a baud rate of 10400 baud. The application device (ApD) transmits a special startDevelopmentMode service after the wake up pattern on the "K-Line".

Note: The termination of the development mode protocol shall be part of the specification of the development mode protocol. After the termination the server (ECU) shall be ready for a Keyword Protocol 2000 initialisation.

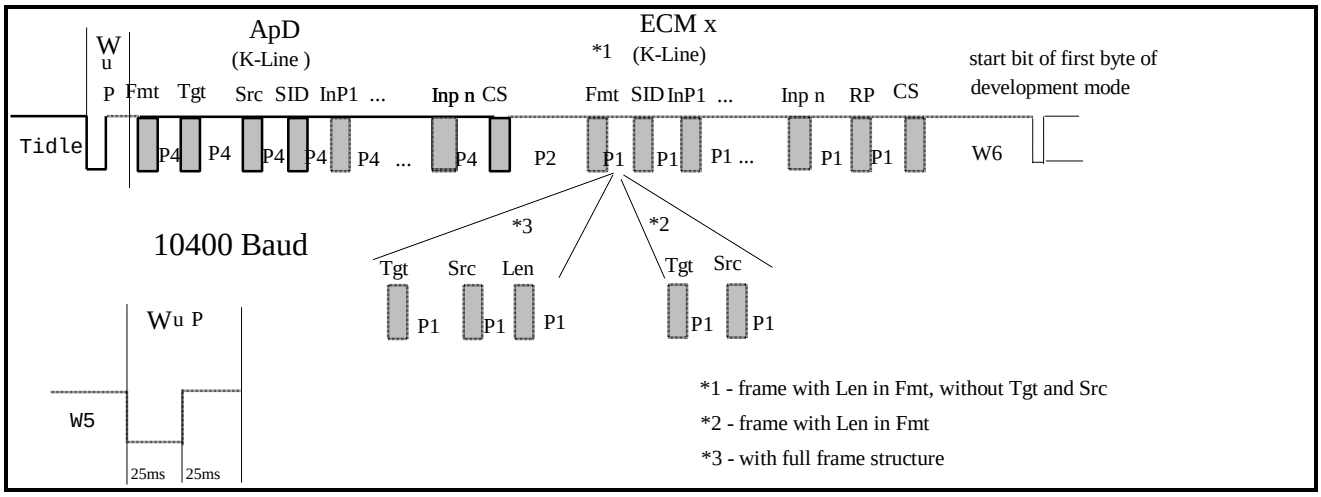


Figure D.1 - StartDevelopmentModeCommunication

Timing parameters are (all values in ms):

	min	max	Description
Tidle	-	-	see section 5.1.5.3
W6	-	-	manufacturer/supplier specific

Table D.1 - Definition of initialisation timing parameters

Definition of communication timing parameters:

The following timing parameter values "P1, P2 and P4" are fixed values. They cannot be changed with the accessTimingParameter service.

- P1: Interbyte time for ECM response
- P2: Intermessage time for servers (ECUs)
- P4: Interbyte time for client (tester) request

	lower limit values			upper limit values		
	min.	default	resolution	default	max.	resolution
P1	0	0	---	20	20	---
P2	0	25	0.5	50	50	25
P4	0	5	0.5	20	20	---

Table D.2 - Definition of communication timing parameters

Time for initialisation of development mode:

Tinit = Tup + 5*P4 + P2 + 4(7)*P1 + W6 + 11(14)*Tchar

best case = 49ms + 5*5ms + 25ms + 4*0ms + 30ms + 11*0.9615ms = 139,6 ms
(ECM-response without Tgt and Src, Len in Fmt-byte, P1lower default, P2lower default, P4lower default)

worst case = 51ms + 5*5ms + 50ms + 7*20ms + 30ms + 14*0.9615ms = 309,5ms
(ECM-response with full frame, P1upper default, P2upper default, P4lower default)

Byte #	Parameter Name	CVT	Hex Value	Mnemonic
#1	Format byte	M	xx	FMT
#2	Target address 1)	C1	xx	TGT
#3	Source address	C1	xx	SRC
#4	**startDevelopmentMode Request Service Id**	**M**	**xx 2)**	**STDM**
#5	initialisationParameter#1	U	xx	INP#1
:	:	U	xx	:
#n-1	initialisationParameter#m	U	xx	INP#m
#n	Checksum	M	xx	CS

Table D.3 - startDevelopmentMode Request Message

1) Target address in messages with address information (5 baud initialisation address).
2) To be defined by vehicle manufacturer
C1 Format byte is 10xx xxxx or 11xx xxxx

Byte #	Parameter Name	CVT	Hex Value	Mnemonic
#1	Format byte	M	xx	FMT
#2	Target address	C1	xx	TGT
#3	Source address	C1	xx	SRC
#4	Additional length byte	C2	xx	LEN
#5	**startDevelopmentMode Positive Response Service Id**	**S**	**xx**	**STDMPR**
#6	initialisationParameter#1 2)	U	xx	INP#1
:	:	U	xx	:
:	initialisationParameter#m	U	xx	INP#m
:	recordParameter#1 { communicationIdentifier }	U	xx	RP#1
:	:	U	xx	:
#n-1	recordParameter#m	U	xx	RP#m
#n	Checksum	M	xx	CS

Table D.3 - startDevelopmentMode Positive Response Message

C1 Format byte is 10xx xxxx or 11xx xxxx
C2 Format byte is xx00 0000.
2) In case of McMess this sequence is used also as a mode selector. Two modes are selectable:
- **$A6 : McMess with high baudrate (> 100 KBaud)**
- **$A5 : McMess with low baudrate (standard PC baudrates)**

Byte #	Parameter Name	CVT	Hex Value	Mnemonic
#1	Format byte	M	xx	FMT
#2	Target address byte	C1	xx	TGT
#3	Source address byte	C1	xx	SRC
#4	Additional length byte	C2	xx	LEN
#5	**negative Response Service Id**	**S**	**7F**	**NR**
#6	**startDevelopmentMode Request Service Id**	**M**	**xx**	**STDM**
#7	ResponseCode = generalReject	M	10	RC
#8	Checksum	M	xx	CS

Table D.4 - startDevelopmentMode Negative Response Message

C1 Format byte is 10xx xxxx or 11xx xxxx
C2 Format byte is xx00 0000.

Revision History Log

Date	Section	Description of change	Requested by
09SEP96	page 1, entire document	New document title: KWP 2000 Data Link Layer Specification	AG1
09SEP96	Introduction, figure 0	replace German Specification with new title	AG1
09SEP96	Section 4.4 Data segmentation	Note added: Data segmentation shall be used only with normal timing parameters	AG 1
09SEP96	Section 5.1.2.2.3, table	Corrected Mnemonic in table: STC (request Sid), STCPR (positive response SID)	AG 1
09SEP96	Section 5.2.2, table	Corrected Mnemonic in table: SPC (request Sid), SPCPR (positive response SID), SPCNR (negative response Sid)	AG 1
24SEP96	Cover page	Added DELCO Electronics Europe	Pischke, Bosch
05JAN97	Section 2, Normative Reference	Added remark to all Draft International Standard documents	Pischke, Bosch
05JAN97	Section 2.2	New section: Other standards. Added KWP 2000 Implementation document	Feiter, GM STG-IO
05JAN97	entire document	Reformatted entire document to be compatible with current ISO/DIS 14230 Keyword Protocol 2000 Part 2: Data Link Layer. Re-numbered all figures and some tables. Some figures have been copied from the ISO/DIS and others which are not included in the ISO/DIS have been re-drawn without any technical change.	Feiter, GM STG-IO
05JAN97	section 5.1.5.2.3	Added section 5.1.5.2.3.1 Functional initialisation between client (tester) and a group of servers (ECUs) Added section 5.1.5.2.3.2 Functional initialisation between client (tester) and gateway	Preuschoff, MB Feiter, GM STG-IO
18JAN97	table 4.4.1.3.1	changed description and figures: "P3max* = P2min - 5 ms" to "P3max* = P2min - 2 ms"	Feiter, GM STG-IO
20JAN97	section 4.4.4.3	**Table 4.4.2.3 - Timing parameter - periodic transmission** The upper limit of the calculated P2max timing parameter value has changed from 6350 to 89600 ms.	Feiter, GM STG-IO
26JAN97	section 4.4.4.3	• The **P2max** timing parameter shall not exceed 57600 ms ($F9). Changed value in table. • The **P3min and P3min*** timing parameters shall always be identical in function and timing values. • The **P3max and P3max*** timing parameters are defined different in function and timing values. ⇨ The **P3max** timing parameter is only used for time out detection during neagtive response message handling with the response code "$78 requestCorrectlyReceived-responsePending". ⇨ The calculated P3max* timing parameter is used as the upper limit of the P3 timing window for the client (tester) to send the next response message.	AG1
26JAN97	figure 1	Added ISO 14230-4 document to figure 1.	AG1

Date	Section	Description of change	Requested by
26JAN97	section Introduction	Added the following sentence: In addition, the section "Definitions and Abbreviations" and the section "Conventions" of the Keyword Protocol 2000 Implementation of Diagnostic Services Recommended Practice document fully applies to this Recommended Practice document.	Heistert, DSA
26JAN97	section 5.1.5.2.3.2	Changed "shall" to "may" in 1st bullet of description in this section.	AG1
29JAN97	section 4.4	Added Note: If gateways to networks are used in a physical point to point connection with functional initialisation (client (tester) to server (ECU/gateway) connection) a modified timing may be needed to support proper communication on the "K-Line". In such case the system supplier and vehicle manufacturer may require the implementation of the accessTimingParameter service.	Preuschoff, MB
29JAN97	section 5.1.5.2.3.2	Added new section for gateway to network implementation	Preuschoff, MB, Feiter, GM STG-IO
08MAR97	title page	Added ETAS company	Hoffmann, ETAS
02APR97	section 4.4	Corrected referenced figure from „figure 4" to „figure 9" (behind Note)	Magnusson, MECEL
02APR97	tables 4.4.4.1 4.4.4.2 4.4.2.1 to 4.4.2.4	Changed Note in tables from „Case #4 must always be terminated with case #1 or #2 (only if request message = stopCommunication) or #3!" to „Case #4 must always be terminated with case #1 or #2 or #3 (only if request message = stopCommunication)!"	Magnusson, MECEL
02APR97	section 4.4.4.3	Changed within „Step #1": „The timing parameters can be changed within the possible limits of the normal timing parameter set with the communication service accessTimingParameters." to „The timing parameters can be changed within the possible limits of the **„periodic transmission timing parameter"** set with the communication service accessTimingParameters."	Magnusson, MECEL
02APR97	section 4.4.4.3, periodic transmission	Changed text for figure 12 from „... P3 jump-in timing window" to "... P3 jump-in timing window with default values". Changed second sentence after figure 12 from „With this timing conditions the fix timing window P3 starts at P3min (5ms)and ends at P3max* (23ms)" to „With these default timing values the timing window P3 starts at P3min (5ms) and ends at P3max* (23ms)". Changed figure 13: Added P3min and correct range for P3 Changed presentation of P3max* in timing parameter table (table 4.4.4.3) from formula „P2min -2" to values and formula „125,5 (P2min-2)" Removed wrong reference to response codes in examples. Corrected message flow: After a negative response the periodic transmission must continue.	Magnusson, MECEL
02APR97	4.4.5	Functional Initialisation: Removed sentence „In any case the mode bits in a response are equal to the mode bits in the request." This sentence is wrong, because the response is always physical.	Magnusson, MECEL

Date	Section	Description of change	Requested by
02APR97	table 6.2.1 table 6.2.2	Replaced the word „may“ with „shall“ Replaced „(P1max time-out)“ with „(P3max time-out)“ and „(P4max time-out)“	Magnusson, MECEL
02APR97	table 5.1.5.1.2	Changed table heading from „Supported“ to „Supported message format by the server (ECU)“	Walter, GM STG IO
22APR97	title page	Removed company „ACG Technical Centre Luxembourg“ (It is Delco)	Poull, Delco
22APR97	section 4.4.4.3	Removed client (tester) jump in window of 5 ms	AG1
22APR97	section 5.1.5.2.1	Changed section heading from „CARB initialisation“ to „ISO 9141-2 initialisation“	Walter, GM STG IO
11JUN97	section 4.4.4.4	Description of swedish task force about multiple physical initialisation added.	Walter, GM STG IO
11JUN97	title page	AISIN AW CO., Limited Japan	Walter, GM STG IO
01JULI97	title page	changed Mercedes-Benz AG to Daimler-Benz AG	Preuschoff, DB
01JULI97	***entire document***	***Version 1.3: Pre-Release for VDA publication***	***AG1***
09JUL97	section 4.4.4.4	Note added: „It is in the designer’s responsibility to ensure proper communication (error handling) in the case of using multiple physical initialisation. Normal physical error handling may cause serious problems in communication and error handling. The proper solution is to use functional error handling (see error handling in clause 6) instead of. That means, that functional error handling is necessary.“	AG1
09JUL97	section 5.1.5.2.2 figure 23	Corrected figure 23: The inverted addres byte will be send by the server (ECU), not the client (tester)	AG1
09JUL97	section 4.4.4.3.1 4.4.3.2	Message flows updated because of missing service options	AG1
09JUL97	section 4.4.4.3	Changed timing range of P3max* from „8...125,5ms“ to „0...125,5ms“. Update of P2min lower limit from 10ms to 2 ms because of new range for P3max* (P3max* = P2min - 2).	AG1
09JUL97	section 4.4.1 figure 8	Corrected figure 8: The inverted addres byte of the 5 baud initialisation will be send by the server (ECU), not the client (tester) Fast initialisation: Changed „W5=300ms“ to „Tidle“	AG1
17JUL97	title page	Added companies: - MAN Nutzfahrzeuge AG - VDO Adolf Schindling AG - Isuzu Motors Ltd.	Walter, GM STG IO
17JUL97	5.1.5.3	Added additional description for WuP: „The pattern begins with a falling edge after an idle time on K-line. While communication is running it’s not necessary for a server (ECU) to react when receiving a WuP.“. Added additional description for Tidle = 0: „Tidle = 0 (starting with a falling edge)“	AG1

Date	Section	Description of change	Requested by
17JUL97	5.1.5.1	Added additional description about message format: „The header format of a server (ECU) Reponse message must be the same as of the client (Tester) Request Message, excepted the length information (e.g. a client (Tester) Request with format, target and source information must be answered with a response containing the same header information).“	Möller, Opel
12SEP97	history log	Column for version number added	Walter, GM STG IO

Date	Version of document	Section	Description of change	Requested by
12SEP97	1.3	title page	Added companies: - FEV Motorentechnik GmbH & Co. KG - Kelsey Hayes	Walter, GM STG IO
16SEP97	*1.4*	*entire document*	*Version 1.4: Release for VDA publication*	*AG1*
29SEP97	1.4	title page	Bottom line of title page removed.	*AG1*
29SEP97	1.4	Appendix D	Service Id '0x0A' changed into 'xx', because it is a vehicle manufacturer specific service Id. Note „2) To be defined by vehicle manufacturer" added.	*AG1*
29SEP97	1.4	title page	Added companies: - LucasVarity	Walter, GM STG IO
01OCT97	***1.5***	***entire document***	***Final release for VDA publication: Version 1.5***	***AG1***