Product Specification

smartCAR

USB 2.0 Stand-alone Device for CAN, LIN or K-LINE Interface

User Manual Version 1.3

GOEPEL electronic

Get the total Coverage!

GOEPEL electronic GmbH
Goeschwitzer Str. 58/60
D-07745 Jena
Phone: +49-3641-6896-597
Fax: +49-3641-6896-944
E-Mail: ats_support@goepel.com
http://www.goepel.com

Printed: 05.12.2011

Issue: December 2011

1 Installation

1.1 Hardware Installation

Generally hardware installation for smartCAR means exchanging the transceiver modules.

Please make absolutely certain that all of the hardware installation procedures described below are carried out with your system **switched off**.

If it is necessary to exchange transceiver modules, no intervention in the smartCAR device is required (see Change of Transceivers).

Doing this, pay attention to the general rules to avoid electrostatic discharging.

1.2 Driver Installation

For proper installation of the GOEPEL electronic USB drivers on your system, we recommend to execute the GUSB driver setup.
To do that, start the *GUSB-Setup-*.exe* setup program (of the supplied CD, "*" stands for the version number) and follow the instructions.

Your smartCAR can be operated under Windows® 2000/ XP as well as under Windows® 7/ 32 Bits and Windows® 7/ 64 Bits.

If you want to create your own software for a smartCAR, you possibly need additional files for user specific programming (*.LLB, *.H). These files are not automatically copied to the computer and have to be transferred individually from the supplied CD to your development directory.

The USB interface uses the high-speed data rate according to the USB2.0 specification (if possible, otherwise full-speed).

After driver installation, you can check whether the device is properly embedded by the system:

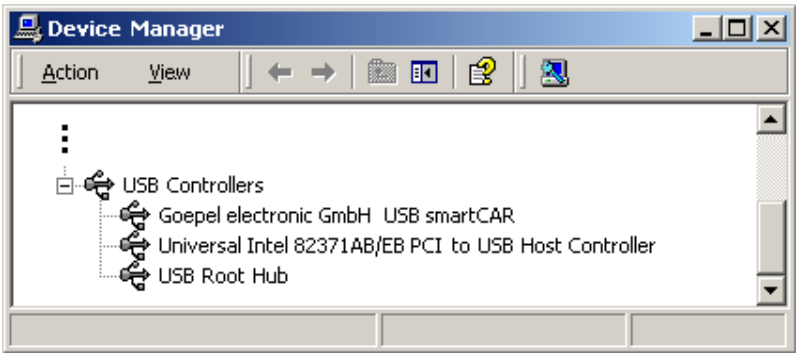

Figure 1-1: Display of Device Manager

Please note that the Device Manager shows ALL USB controllers.

2 Hardware

2.1 Definition

smartCAR is a GOEPEL electronic GmbH stand-alone device with USB 2.0 interface to be connected to a PC or laptop.
It was in particular developed for applications out of complex test systems (for example in garages).

Figure 2-1: smartCAR

smartCAR offers the following resources:

- 1 x CAN or 1x LIN or 1x K-Line
- 32Bit µController onBoard
- USB 2.0 Interface
- Power supply optionally via the USB interface or externally
- High flexibility by exchangeable transceiver modules

Please note that your smartCAR DOES NOT provide electric isolation between the USB system and the user interface.
Therefore, the UUT and all other devices connected with the smartCAR have to supplied either by isolated power supply units or all involved devices have to be connected to the same ground potential in a star-shaped manner.

2.2 Technical Specification

2.2.1 Dimensions

The dimensions of your smartCAR are given in millimeters (width x height x depth):

- 75 mm x 25 mm x 110 mm

2.2.2 smartCAR Characteristics

The smartCAR characteristics are shown in this table:

Symbol	Parameter	Min.	Typ.	Max.	Unit	Remarks
U_{BAT}	Power supply		8	27	V	Via USB interface or externally
	Transmission rate			1	MBaud	For CAN or
	Transmission rate			22	kBaud	For LIN or
	Transmission rate			150	kBaud	For K-Line
R_{bus}	Terminating resistor		120		Ohms	For CAN or
R_{Pullup}	Pull-up resistor		1000		Ohms	For K-Line

2.3 Construction

2.3.1 General

Figure 2-2 shows schematically the construction of a smartCAR:

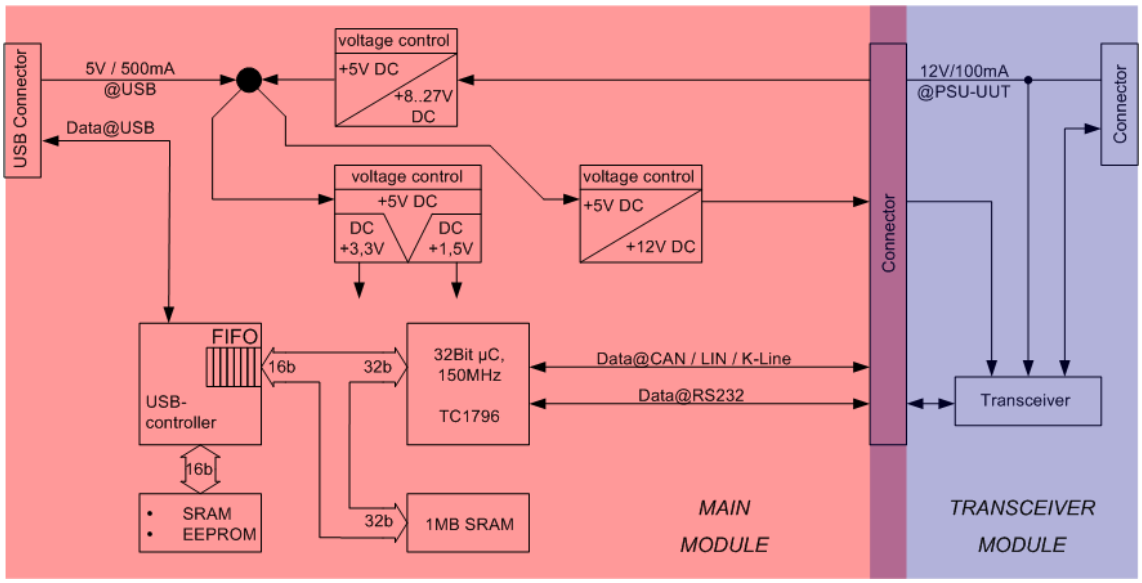

Figure 2-2: Block diagram of a smartCAR device

Warning

Please use only the delivered USB cable to connect your smartCAR device to the PC's USB interface.
Other cables may be inapplicable.

2.3.2 Addressing

In case of using several smartCAR devices at the same PC the individual device is exclusively addressed according to its serial number (see Control Software):
The device with the LEAST serial number is always the device with the number 1.

Tip

To improve clarity, we recommend to connect the individual smartCAR devices with the same PC in the order of ascending serial numbers.

2.3.3 Change of Transceivers

Figure 2-3 demonstrates the mechanical join between smartCAR's main module and transceiver module.

To change the transceiver module, separate the assembled one by top-bottom traction from the main module.

Figure 2-3 Change of Transceiver module

2.3.4 Communication Interfaces

2 x CAN-Interface Version 2.0b:

The type of the mounted transceiver is decisive for proper operation of a CAN interface in a network. Often CAN networks do only operate properly in the case that all members use a compatible type of transceiver.

To offer maximal flexibility to the users of the smartCAR device, the transceivers are designed as plug-in modules.
There are several types (high speed, low speed, single-wire etc.) that can be easily exchanged (see Figure 2-3).

U_{bat} is the internal connection for the power supply of the transceiver modules.

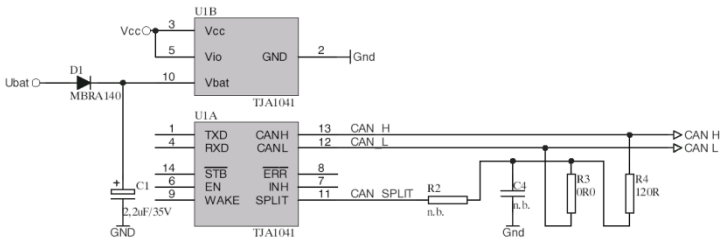

Figure 2-4: CAN interface

K-Line Interface (ISO 9141)

The transceiver is designed as a plug-in module.
Generally, the L9637 of ST is used for this type of transceiver.

U_{bat} is the internal connection for the power supply of the transceiver module.

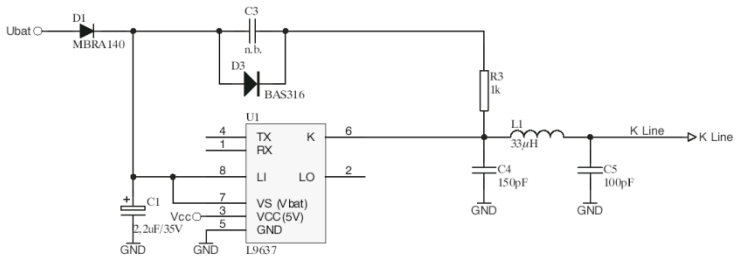

Abbildung 2-5
K-Line interface

LIN-Interface Version 2.0:

The transceiver is designed as a plug-in module.
Generally, the TJA1020 of Philips is used for this type of transceiver.

It is possible to change over between Master and Slave configuration per software using the relay with number 2.

U_{bat} is the internal connection for the power supply of the transceiver module.

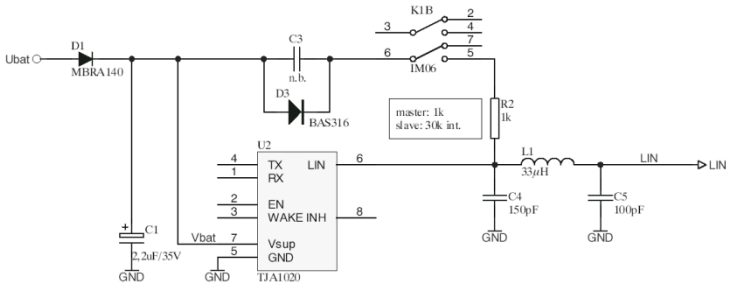

Figure 2-6:
LIN interface

2.3.5 Connector Assignments

For the access to the communication interface there is the RJ45 socket at the front side of your smartCAR device.
You may also use the SubD plug of the delivered cable.

Communication Interface

Type: RJ45 female

The assignments are shown in the following table:

Pin	CAN	K-Line/ LIN
1	U_{bat}	U_{bat}
2	n.c.	n.c.
3	CAN-High	n.c.
4	CAN-Low	n.c.
5	n.c.	K-Line/ LIN
6	n.c.	n.c.
7	n.c.	n.c.
8	GND	GND

Type: DSub 9 poles male (at the cable)

The assignments are shown in the following table:

Pin	CAN	K-Line/ LIN
1	n.c.	n.c.
2	CAN-Low	n.c.
3	GND	GND
4	n.c.	n.c.
5	n.c.	n.c.
6	n.c.	n.c.
7	CAN-High	K-Line/ LIN
8	n.c.	n.c.
9	U_{bat}	U_{bat}

USB Interface

At smartCAR's rear side there is the miniUSB-socket (with USB standard assignment) for the USB 2.0 interface.

2.4 Delivery Notes

A smartCAR delivery includes at least

- 1x smartCAR Main module and 1x smartCAR Transceiver module

At present the following types of Transceiver modules are available:

- 1x TJA1041 CAN Highspeed
- 1x TJA1054 CAN Lowspeed
- 1x AU5790 CAN Single Wire
- 1x L9637 K-Line
- 1x TJA 1020 LIN

When ordering a smartCAR, please give also a note regarding the type of the required Transceiver module.

Only by exchanging the Transceiver module (see Figure 2-3) you decide whether the smartCAR hardware interface is working as a CAN, LIN or K-Line interface.

3 Control Software

There are three ways to integrate the smartCAR hardware in your own applications:

- Programming via G-API
- Programming via DLL Functions
- Programming with LabVIEW

3.1 Programming via G-API

The G_API (GOEPEL-API) is the favored user interface for this GOEPEL hardware.

You can find all necessary information in the *G-API* folder of the delivered CD.

3.2 Programming via DLL Functions

Programming via DLL Fuctions is possible also in future for existing projects which can not be processed with the GOEPEL electronic programming interface G-API.

The GUSB_Platform expression used in the following function description stands for one individual smartCAR device.

For the used structures, data types and error codes refer to the headers – you find the corresponding files on the supplied CD (see also General Firmware Notes):

In this User Manual, Controller means ALWAYS the micro controller assigned to the CAN, LIN or K-Line interface of a smartCAR device. On the other hand, USB Controller means ALWAYS the controller providing the USB 2.0 interface of the smartCAR device.

3.2.1 Windows Device Driver

The DLL functions for programming using the Windows device driver are described in the following sections:

- Driver_Info
- DLL_Info
- Write_FIFO
- Read_FIFO
- Read_FIFO_Timeout
- Write_COMMAND
- Read_COMMAND

3.2.1.1 *Driver_Info*

The GUSB_Platform_Driver_Info function is for the status query of the hardware driver and for the internal initialization of the required handles.

Executing this function at least once is obligatory before calling any other function of the GUSB_Platform driver.

Format:

```
int GUSB_Platform_Driver_Info(GUSB_Platform_DriverInfo *pDriverInfo,
                              unsigned int LengthInByte)
```

Parameters:

Pointer, for example pDriverInfo
to a data structure
For the structure, see the *GUSB_Platform.h* file on the delivered CD

LengthInByte

Size of the storage area pDriverInfo is pointing to, in bytes

Description:

The GUSB_Platform_Driver_Info function returns information regarding the status of the hardware driver.

For this reason, the address of the pDriverInfo pointer has to be transferred to the function. By means of the LengthInByte parameter the function checks internally if the user memory is initialized correctly.

The function fills the structure pDriverInfo is pointing to with statements regarding the driver version, the number of all involved USB controllers (supported by this driver) and additional information, e.g. the serial number(s).

Making the hardware information available
as well as initializing the belonging handles is obligatory for the further use of the USB hardware.

3.2.1.2 DLL_Info

The GUSB_Platform_DLL_Info function is for the version number query of the DLL.

Format:

```
int GUSB_Platform_DLL_Info(GUSB_Platform_DLLinfo *DLLinformation)
```

Parameters

Pointer, for example DLLinformation
to a data structure
For the structure, see the *GUSB_Platform.h* file on the delivered CD

Description:

The GUSB_Platform_DLL_Info function returns the DLLinfo structure. The first integer value contains the version number of the *GUSB_Platform.dll*.

Example:

Version number **1.23** is returned as **123**,
and version number **1.60** as **160**.

3.2.1.3 *Write_FIFO*

With the GUSB_Platform_Write_FIFO function a command is sent to the Controller.

Format:

```
int GUSB_Platform_Write_FIFO(unsigned int DeviceName,
                             unsigned int DeviceNumber,
                             t_USB_FIFO_Interface_Buffer *pWrite,
                             unsigned int DataLength)
```

Parameters:

DeviceName

Type of the addressed device
(number declared in *GUSB_Platform_def.h*, for smartCAR = 13)

DeviceNumber

Number of the addressed device. In the case several devices of the same type are connected, numbering is carried out according to their serial numbers in ascending order (the device with the LEAST serial number has always the DeviceNumber 1).

Pointer, for example pWrite
to the write data area

DataLength

Size of the storage area pWrite is pointing to, in bytes
Data is consisting of Command Header and Command Bytes
See also Command Structure
(currently max. 1024 bytes per command)

Description:

The GUSB_Platform_Write_FIFO function sends a command to the Controller.
For the general structure see the General Firmware Notes section.

3.2.1.4 Read_FIFO

The GUSB_Platform_Read_FIFO function is for reading a response from the Controller.

Format:

```
int GUSB_Platform_Read_FIFO(unsigned int DeviceName,
                            unsigned int DeviceNumber,
                            t_USB_FIFO_Interface_Buffer *pRead,
                            unsigned int *DataLength)
```

Parameters:

DeviceName

Type of the addressed device
(number declared in *GUSB_Platform_def.h*, for smartCAR = 13).

DeviceNumber

Number of the addressed device. In the case several devices of the same type are connected, numbering is carried out according to their serial numbers in ascending order (the device with the LEAST serial number has always the DeviceNumber 1).

Pointer, for example pRead
to the reading buffer
After sucessful execution of the function, there is the data in this reading buffer, consisting of Response Header and Response Bytes
See also Response Structure
(currently max. 1024 bytes per response)

DataLength

Prior to function call: Size of the reading buffer in bytes (to be given)

After function execution: Number of bytes actually read

Description:

The GUSB_Platform_Read_FIFO function reads back the oldest response written by the Controller. In the case no response was received within the fixed Timeout of 100 ms, the function returns NO error, but the Number of bytes actually read is 0 !!!

3.2.1.5 Read_FIFO_Timeout

The GUSB_Platform_Read_FIFO_Timeout function is for reading a response from the Controller within the Timeout to be given.

Format:

```
int GUSB_Platform_Read_FIFO_Timeout(unsigned int DeviceName,
                                    unsigned int DeviceNumber,
                                    t_USB_FIFO_Interface_Buffer *pRead,
                                    unsigned int *DataLength,
                                    unsigned int Timeout)
```

Parameters:

DeviceName

Type of the addressed device
(number declared in *GUSB_Platform_def.h*, for smartCAR = 13).

DeviceNumber

Number of the addressed device. In the case several devices of the same type are connected, numbering is carried out according to their serial numbers in ascending order (the device with the LEAST serial number has always the DeviceNumber 1).

Pointer, for example pRead
to the reading buffer
After successful execution of the function, there is the data in this reading buffer, consisting of Response Header and Response Bytes
See also Response Structure
(currently max. 1024 bytes per response)

DataLength

Prior to function call: Size of the reading buffer in bytes (to be given)

After function execution: Number of bytes actually read

Timeout

To be given in milliseconds (500 as the standard value)

Description:

The GUSB_Platform_Read_FIFO_timeout function reads back the oldest response written by the Controller. In the case no response was received within the Timeout to be given, the function returns NO error, but the Number of bytes actually read is 0 !!!

3.2.1.6 Write_COMMAND

With the GUSB_Platform_Write_COMMAND a configuration command is sent to the USB Controller.

Format:

```
int GUSB_Platform_Write_COMMAND(unsigned int DeviceName,
                                unsigned int DeviceNumber,
                                t_USB_COMMAND_Interface_Buffer *pWrite,
                                unsigned int DataLength)
```

Parameters:

DeviceName

Type of the addressed device
(number declared in *GUSB_Platform_def.h*, for smartCAR = 13).

DeviceNumber

Number of the addressed device. In the case several devices of the same type are connected, numbering is carried out according to their serial numbers in ascending order (the device with the LEAST serial number has always the DeviceNumber 1).

Pointer, for example pWrite
to the write data area
See also USB Controller Control Commands
(currently max. 64 bytes per command)

DataLength

Size of the storage area pWrite is pointing to, in bytes

Description:

The GUSB_Platform_Write_COMMAND function sends a command to the USB Controller.
For the general structure, see the USB Controller Control Commands section.

3.2.1.7 Read_COMMAND

The GUSB_Platform_Read_COMMAND function is for reading a response from the USB Controller.

Format:

```
int GUSB_Platform_Read_COMMAND(unsigned int DeviceName,
                               unsigned int DeviceNumber,
                               t_USB_COMMAND_Interface_Buffer *pRead,
                               unsigned int *DataLength)
```

Parameters:

DeviceName

Type of the addressed device
(number declared in *GUSB_Platform_def.h*, for smartCAR = 13).

DeviceNumber

Number of the addressed device. In the case several devices of the same type are connected, numbering is carried out according to their serial numbers in ascending order (the device with the LEAST serial number has always the DeviceNumber 1).

Pointer, for example pRead
to the reading buffer
After sucessful execution of the function, there is the data in this reading buffer, consisting of Response Header and Response Bytes
See also USB Controller Control Commands
(currently min. 64 bytes per response)

DataLength

Prior to function call: Size of the reading buffer in bytes (to be given)

After function execution: Number of bytes actually read

Description:

The GUSB_Platform_Read_COMMAND function reads back the oldest response written by the USB Controller.

If several responses were provided by the USB Controller, up to two of these responses are written into the buffer of the USB Controller. More possibly provided responses get lost!

3.3 Programming with LabVIEW

3.3.1 LabVIEW via G-API

On the delivered CD there is a folder with VIs to call smartCAR devices under LabVIEW.

The LabVIEW VIs use the functions of the GOEPEL G-API for this.

3.3.2 LLB using the Windows Device Driver

On the delivered CD there is a folder with VIs to call smartCAR devices under LabVIEW.

The functions described in the Windows Device Driver section are used for this.

3.4 Further GOEPEL Software

PROGRESS, Program Generator and myCAR of GOEPEL electronic are comfortable programs for testing with GOEPEL hardware.

Please refer to the corresponding Software Manuals to get more information regarding these programs.

3.5 USB Controller Control Commands

The USB Controller is responsible for connecting the smartCAR device to the PC via USB 2.0.

Messages (generally USB commands) required for configuration can be sent to this USB Controller.

3.5.1 USB Command Structure

A USB command consists of four bytes Header and the Data (but Data is NOT required for all USB commands!).

The header of a USB command has the following structure:

Byte number	Indication	Contents
0	StartByte	0x23 ("#" ASCII character)
1	Command	(0x..) used codes according to USB Commands
2	reserved	0x00
3	reserved	0x00

3.5.2 USB Response Structure

Same as a USB command, also the USB response consists of four bytes Header and the Data (but Data is NOT returned by all USB commands!).

The header of a USB response has the following structure:

Byte number	Indication	Contents
0	StartByte	0x24
1	Command	(0x..) used codes according to USB Commands
2	Length	Length depending on the command
3	ErrorCode	Returns the error code of the command

3.5.3 USB Commands

At present there is only the READ_SW_VERSION USB command available.

Command	Indication	Description
0x04	READ_SW_VERSION	Provides the firmware version of the USB Controller Response: Byte 4: low byte of generic software version Byte 5: high byte of generic software version Byte 6: low byte of software version of functional part Byte 7: high byte of software version of functional part

4 Firmware Commands

4.1 General Firmware Notes

Things in common for all firmware commands on the micro controller of a smartCAR device are described in this chapter (see also Interfaces).

The explicit use of the firmware commands for your smartCAR hardware is only necessary if you want to work with your own applications without the GOEPEL electronic Programm Generator, PROGRESS or myCAR programs or the use of the G-API is not possible.

After the subsections

- Interfaces (see Interfaces)
- Data types (see Data Types)
- Header (see Header)
- Constants (see Constants)
- Command Structure (see Command Structure)
- Response Structure (see Response Structure)
- Command Acknowledgment (see Command Acknowledgment)
- Command Examples (see Command Examples)
- Bootloader (see Bootloader)
- Command Sequence (see Command Sequence)

there is the actual command description in the subsections

- General Firmware Commands (see General Firmware Commands)
- CAN Commands (see CAN Commands)
- LIN Commands (see LIN Commands)
- K-Line Commands (see K-Line Commands)

Warning

Please ensure to transfer always the value 0 for all command bytes and bits indicated reserved in the command descriptions.
In the case more command bytes as described are transferred, also these bytes must be filled with 0!

The firmware commands
0x03 Enable Functionalities
0x10 Software Reset
0xF0 Get Firmware Version
refer to the microcontroller of a smartCAR device (irrespective of the current Software Interface, see Interfaces).

4.1.1 Interfaces

Take the assignment of the Software interface to the corresponding Interface number for the Controller of a smartCAR device from this table:

Software Interface	Interface Number	Controller Number
CAN	1	1
LIN	4	1
K-Line	6	1

All firmware commands for a smartCAR device are operated by the same micro controller.

4.1.2 Data Types

As a rule data is handled as 8, 16 or 32 bit integer in the Intel format (little endian). That means low byte first and then high byte(s), see example for a 32 bits integer value:

Example: Identifier **0x00A534FE** (4 bytes)

Bytearray[x] **0xFE**

Bytearray[x+1] **0x34**

Bytearray[x+2] **0xA5**

Bytearray[x+3] **0x00**

Handling of floating point values is carried out in the
little endian IEEE-754 32 bit single precision format for float
and in the
little endian IEEE-754 64 bit double precision format for double.

Here are further examples for different data types:

Type of Data	Value	Bytes	Byte Offset
16 bits integer (short)	**0x1234**	**0x34** **0x12**	0 (Low Byte) 1 (High Byte)
32 bits integer (long)	**0x12345678**	**0x78** **0x56** **0x34** **0x12**	0 (Low Byte) 1 2 3 (High Byte)
32 bits floating point (float)	**123.456**	**0x79** **0xE9** **0xF6** **0x42**	0 (Low Byte) 1 2 3 (High Byte)
64 bits floating point (double)	**123.456**	**0x77** **0xBE** **0x9F** **0x1A** **0x2F** **0xDD** **0x5E** **0x40**	0 (Low Byte) 1 2 3 4 5 6 7 (High Byte)

4.1.3 Header

The following header is used for the complete data exchange:

Byte number	Indication	Description
0	StartByte	0x23 (“#” ASCII character)
1	Flags	Bit 0 = 1: Always command acknowledgment Bit 1 = 1: Command acknowledgment only in case of errors Bits 2..7: Reserved Therefore the following results for the byte value: 0: No command acknowledgment 1: Always command acknowledgment 2: Command acknowledgment only in case of errors
2, 3	Length	12..1024 Complete length of the command (Header + Parameters)
4	TargetAddress	**Address of the controller** (**1**, **command**) or address of the host application (0, response)
5	TargetPort	**Interface of the controller** (**1**, **4** or **6**, **command**) or port address of the host application (0, response)
6	SourceAddress	**Address of the host application** (**0**, **command**) or address of the controller (1, Response)
7	SourcePort	**Port address of the host application** (**0**, **command**) or interface of the controller (1, 4 or 6, response)
8	Type	0: Command 1: Response 2: Command acknowledgment
9	ApplicationHandle	The content of ApplicationHandle is sent back to the host unchanged by the controller in the case of responses
10	reserved	Reserved
11	Command	Command code (0x00..) Used codes according to firmware, see General Firmware Commands, CAN Commands, LIN Commands and K-Line Commands

The controlling of further smartCAR devices is carried out via the DeviceNumber in the GUSB_Platform driver.

If possible use always command acknowledgment (Flags/ bit 0 = 1 or bit 1 = 1) to get a feedback from the hardware after command execution.

The address of the host application (SourceAddress for commands, TargetAddress for responses) should be always 0.
In this case the port of the host application (SourcePort for commands, TargetPort for responses) can be allocated freely (generally 0).

4.1.4 Constants

The maximum quantity of Command bytes results from the following equotation:
PARAM_SIZE = MESSAGE_SIZE – HEADER_SIZE

Symbolic constant	Value for smartCAR	Remarks
MESSAGE_SIZE	1024	Maximum size of the command or response including Header in bytes
HEADER_SIZE	12	Size of the Header in bytes
PARAM_SIZE	1012	Maximum size of the parameters (Command bytes or Response bytes) in bytes

4.1.5 Command Structure

A command consists of Command header and Command bytes (but not all commands require Command bytes).

The Command header consists of the Header with Type = 0 (see Header) and the corresponding command code Command.
In this documentation, look for the Command at the beginning of each heading of the command descriptions.

The Command bytes are limited regarding their quantity by the maximum size of the command (MESSAGE_SIZE) and the size of the command header (HEADER_SIZE). The maximum quantity of Command bytes corresponds to the value of PARAM_SIZE (see Constants).

4.1.6 Response Structure

A response is built up of Response header and Response bytes.

The Response header consists of the Header with Type = 1 (see Header) and the command code Command according to the corresponding executed command.

The Response bytes are limited regarding their quantity by the maximum size of the command (MESSAGE_SIZE) and the size of the response header (HEADER_SIZE). The maximum quantity of Response bytes corresponds to the value of PARAM_SIZE (see Constants).

The values for TargetAddress, TargetPort, SourceAddress as well as SourcePort are exchanged as a result of the direction change (command/ response).

In the case of firmware commands with a response, this response is described following the description of the corresponding command.
Commands without this response description DO NOT create a response.

4.1.7 Command Acknowledgment

A command is acknowledged by a special response (the command acknowledgment response) to the host after its execution in the controller in case Bit 0 of Flags in the header is set to 1.

This response consists of the Header and Response bytes.

To be able to distinguish command acknowledgment responses from normal responses, Type = 2 is set.

Response:

Byte	Indication	Description
0..3	ErrorNumber	0: No error Otherwise: Error
4.. (3+N)	ErrorDescription	Zero terminated error string of N length (including terminating “Zero” character) 1 ≤ N ≤ (PARAM_SIZE – 4) (PARAM_SIZE see Constants)

In the case of commands with a response (e.g. 0xF0 Get Firmware Version), the controller sends the actual response first (if Bit 0 in Flags of the 12 byte header is set) and then the command acknowledgment response to the host. If several parameters of the command are invalid in this case, the controller sends only the command acknowledgment response (with error number and error description set accordingly). Therefore Type evaluation is absolutely necessary for 12 byte headers.

For working more efficiently with commands creating a response, set Bit 1 in Flags of the 12 byte header instead of Bit 0.
This way always only one response is sent by the controller:
The desired one if the command ran without error, or the command acknowledge response in the case of errors.

4.1.8 Command Examples

Example 1: CAN Command + Command Acknowledgment

The following example shows the individual bytes including the header of the 0x22 CAN Message Definition Command and the corresponding Command Acknowledgment.

The Command is written to the interface 1 (CAN) of the Controller of the third smartCAR device (by function call GUSB_Platform_Write_FIFO, see Control Software).
Then the Command Acknowledgment is read from the same controller of the same device.

Command (including header):

Byte Index	Byte Value	Indication	Description
0	**0x23**	StartByte	0x23 ("#" ASCII character)
1	**0x01**	Flags	Bit 0 = 1: Command acknowledgment (always) active
2	**0x20**	Length	Total length of the command = 12+20 = 32, LowByte
3	**0x00**		Total length of the command = 12+20 = 32, HighByte
4	**0x01**	TargetAddress	Address of the controller = 1 (see also Notes, following page)
5	**0x01**	TargetPort	Interface on the controller = 1
6	**0x00**	SourceAddress	Address of the host application = 0
7	**0x00**	SourcePort	Port address of the host application = 0
8	**0x00**	Type	0: Command
9	**0x00**	ApplicationHandle	Freely selectable, is returned unchanged in the Command Acknowledgment
10	**0x00**	reserved	Reserved, to be assigned by 0
11	**0x22**	Command	Command code for 0x22 CAN Message Definition
12	**0x23**	Id	Identifier (0x123), LowByte
13	**0x01**		Identifier (0x123), MidByte1
14	**0x00**		Identifier (0x123), MidByte2
15	**0x00**		Identifier (0x123), HighByte
16	**0xE8**	CycleTime	Cycle time (1000) in milliseconds, LowByte
17	**0x03**		Cycle time (1000) in milliseconds, HighByte
18	**0x01**	Mode	1: CAN Output message
19	**0x00**	PrepareMode	0: CAN No preparing of the message
20	**0x03**	MessageCount	Output the message 3 times
21	**0x06**	Dlc	Data length = 6
22	**0x11**	Data	Data byte 0 = 0x11
23	**0x22**		Data byte 1 = 0x22
24	**0x33**		Data byte 2 = 0x33
25	**0x44**		Data byte 3 = 0x44
26	**0x55**		Data byte 4 = 0x55
27	**0x66**		Data byte 5 = 0x66
28	**0x77**		Data byte 6 = 0x77 (value is irrelevant, as Dlc = 6)
29	**0x88**		Data byte 7 = 0x88 (value is irrelevant, as Dlc = 6)
30	**0x00**	reserved	Reserved, to be assigned by 0
31	**0x00**		Reserved, to be assigned by 0

Notes:

For USB addressing the Device via the Header of the corresponding Firmware command is INSUFFICIENT. Additionally the DeviceNumber for the GUSB_Platform driver must be adapted (see Programming via DLL Functions in the Control Software section).

This DeviceNumber complies with the serial number of the corresponding device in ascending order.

For this example, Interface 1 (CAN) of the third smartCAR device is addressed by TargetPort 1 in the Header and DeviceNumber 3 in the GUSB_Platform driver.

The required function is:

```
GUSB_Platform_Write_FIFO(13,      // DeviceName
                         3,       // DeviceNumber
                         pWrite,  // pWrite
                         32);     // DataLength
```

Here, pWrite is the pointer to the area including the bytes according to Byte Index 0 .. 31 of the Command (including Header) (previous page).

Command Acknowledgment (including header):

Byte Index	Byte Value	Indication	Description
0	**0x23**	StartByte	0x23 (“#” ASCII character)
1	**0x00**	Flags	No flags set
2	**0x11**	Length	Total length of the response = 12+5 = 17, LowByte
3	**0x00**		Total length of the response = 12+5 = 17, HighByte
4	**0x00**	TargetAddress	Address of the host application = 0
5	**0x00**	TargetPort	Port address of the host application = 0
6	**0x01**	SourceAddress	Address of the controller = 1
7	**0x01**	SourcePort	Interface on the controller = 1
8	**0x02**	Type	2: Command acknowledgment
9	**0x00**	ApplicationHandle	The content is sent back unchanged
10	**0x00**	reserved	Reserved
11	**0x22**	Command	Command code for 0x22 CAN Message Definition
12	**0x00**	ErrorNumber	Error number (0: no error), LowByte
13	**0x00**		Error number (0: no error), MidByte1
14	**0x00**		Error number (0: no error), MidByte2
15	**0x00**		Error number (0: no error), HighByte
16	**0x00**	ErrorDescription	Empty string (only the terminating “zero”)

Example 2: CAN Command + Response

The following example shows the individual bytes including the header of the 0xF2 CAN Monitor – Get List Item Command and the corresponding Response.

The Command is written to the interface 1 (CAN) of the Controller of the smartCAR device (by function call GUSB_Platform_Write_FIFO, see Control Software). Then the Response is read from the same controller of the same device.

Command (including header):

Byte Index	Byte Value	Indication	Description
0	**0x23**	StartByte	0x23 ("#" ASCII character)
1	**0x02**	Flags	Bit 1 = 1: Command acknowledgment only in the case of errors
2	**0x0C**	Length	Total length of the command = 12 (only header), LowByte
3	**0x00**		Total length of the command = 12 (only header), HighByte
4	**0x01**	TargetAddress	Address of the controller = 1 (see also Notes to Example 1)
5	**0x01**	TargetPort	Interface on the controller = 1
6	**0x00**	SourceAddress	Address of the host application = 0
7	**0x00**	SourcePort	Port address of the host application = 0
8	**0x00**	Type	0: Command
9	**0x00**	ApplicationHandle	Freely selectable, is returned unchanged in the Response
10	**0x00**	reserved	Reserved, to be assigned by 0
11	**0xF2**	Command	Command code for 0xF2 CAN Monitor – Get List Item
12	**0x23**	Id	Identifier (0x123), LowByte
13	**0x01**		Identifier (0x123), MidByte1
14	**0x00**		Identifier (0x123), MidByte2
15	**0x00**		Identifier (0x123), HighByte

Response (including header):

Byte Index	Byte Value	Indication	Description
0	**0x23**	StartByte	0x23 ("#" ASCII character)
1	**0x00**	Flags	No flags set
2	**0x24**	Length	Total length of the response = 12+24 = 36, LowByte
3	**0x00**		Total length of the response = 12+24 = 36, HighByte
4	**0x00**	TargetAddress	Address of the host application = 0
5	**0x00**	TargetPort	Port address of the host application = 0
6	**0x01**	SourceAddress	Address of the controller = 1
7	**0x01**	SourcePort	Interface on the controller = 1
8	**0x01**	Type	1: Response
9	**0x00**	ApplicationHandle	The content is sent back unchanged
10	**0x00**	reserved	Reserved
11	**0xF2**	Command	Command code for 0xF2 CAN Monitor – Get List Item
12	**0x23**	Id	Identifier (0x123), LowByte
13	**0x01**		Identifier (0x123), MidByte1
14	**0x00**		Identifier (0x123), MidByte2
15	**0x00**		Identifier (0x123), HighByte
16	**0x78**	TimeStamp	Time stamp (0x1235678), LowByte
17	**0x56**		Time stamp (0x1235678), MidByte1
18	**0x34**		Time stamp (0x1235678), MidByte2
19	**0x12**		Time stamp (0x1235678), HighByte
20	**0xE8**	MessageCount	Number of transmissions (1000), LowByte
21	**0x03**		Number of transmissions (1000), MidByte1
22	**0x00**		Number of transmissions (1000), MidByte2
23	**0x00**		Number of transmissions (1000), HighByte
24	**0x02**	Flags	Bit 1 = 1: sent CAN message (TX)
25	**0x07**	Dlc	Data length = 7
26	**0x01**	TimeStampResolution	Time stamp resolution is 400 nanoseconds
27	**0x00**	reserved	Reserved
28	**0x11**	Data	Data byte 0 = 0x11
29	**0x22**		Data byte 1 = 0x22
30	**0x33**		Data byte 2 = 0x33
31	**0x44**		Data byte 3 = 0x44
32	**0x55**		Data byte 4 = 0x55
33	**0x66**		Data byte 5 = 0x66
34	**0x77**		Data byte 6 = 0x77
35	**0x88**		Data byte 7 = 0x88 (value is irrelevant, as Dlc = 7)

Example 3: LIN Command + Command Acknowledgment

The following example shows the individual bytes including the header of the 0x30 LIN Frame Response Definition Command and the corresponding Command Acknowledgment.

The Command is written to the interface 4 (LIN) of the Controller of the smartCAR device (by function call GUSB_Platform_Write_FIFO, see Control Software). Then the Command Acknowledgment is read from the same controller of the same device.

Command (including header):

Byte Index	Byte Value	Indication	Description
0	**0x23**	StartByte	0x23 ("#" ASCII character)
1	**0x01**	Flags	Bit 0 = 1: Command acknowledgment (always) active
2	**0x1C**	Length	Total length of the command = 12+16 = 28, LowByte
3	**0x00**		Total length of the command = 12+16 = 28, HighByte
4	**0x01**	TargetAddress	Address of the controller = 1 (see also Notes to Example 1)
5	**0x04**	TargetPort	Interface on the controller = 4
6	**0x00**	SourceAddress	Address of the host application = 0
7	**0x00**	SourcePort	Port address of the host application = 0
8	**0x00**	Type	0: Command
9	**0x00**	ApplicationHandle	Freely selectable, is returned unchanged in the Command Acknowledgment
10	**0x00**	reserved	Reserved, to be assigned by 0
11	**0x30**	Command	Command code for 0x30 LIN Frame Response Definition
12	**0x12**	Id	Identifier = 0x12
13	**0x01**	Mode	1: LIN Output LIN frame response
14	**0x00**	PrepareMode	0: LIN No preparing of the LIN frame response
15	**0x03**	MessageCount	Output of LIN frame response 3 times
16	**0x06**	Dlc	Data length = 6
17	**0x00**	reserved	Reserved, to be assigned by 0
18	**0x00**		Reserved, to be assigned by 0
19	**0x00**		Reserved, to be assigned by 0
20	**0x11**	Data	Data byte 0 = 0x11
21	**0x22**		Data byte 1 = 0x22
22	**0x33**		Data byte 2 = 0x33
23	**0x44**		Data byte 3 = 0x44
24	**0x55**		Data byte 4 = 0x55
25	**0x66**		Data byte 5 = 0x66
26	**0x77**		Data byte 6 = 0x77 (value is irrelevant, as Dlc = 6)
27	**0x88**		Data byte 7 = 0x88 (value is irrelevant, as Dlc = 6)

Command Acknowledgment (including header):

Byte Index	Byte Value	Indication	Description
0	**0x23**	StartByte	0x23 (“#” ASCII character)
1	**0x00**	Flags	No flags set
2	**0x11**	Length	Total length of the response = 12+5 = 17, LowByte
3	**0x00**		Total length of the response = 12+5 = 17, HighByte
4	**0x00**	TargetAddress	Address of the host application = 0
5	**0x00**	TargetPort	Port address of the host application = 0
6	**0x01**	SourceAddress	Address of the controller = 1
7	**0x04**	SourcePort	Interfrace on the controller = 4
8	**0x02**	Type	2: Command Acknowledgment
9	**0x00**	ApplicationHandle	The content is sent back unchanged
10	**0x00**	reserved	Reserved
11	**0x30**	Command	Command code for 0x30 LIN Frame Response Definition
12	**0x00**	ErrorNumber	Error number (0: no error), LowByte
13	**0x00**		Error number (0: no error), MidByte1
14	**0x00**		Error number (0: no error), MidByte2
15	**0x00**		Error number (0: no error), HighByte
16	**0x00**	ErrorDescription	Empty string (only the terminating “zero”)

Example 4: LIN Command + Response

The following example shows the individual bytes including the header of the 0xF2 LIN Monitor – Get Small Buffer Items Command and the corresponding Response.

The Command is written to the interface 4 (LIN) of the Controller of the smartCAR device (by function call GUSB_Platform_Write_FIFO, see Control Software). Then the Response is read from the same controller of the same device.

Command (including header):

Byte Index	Byte Value	Indication	Description
0	**0x23**	StartByte	0x23 ("#" ASCII character)
1	**0x02**	Flags	Bit 1 = 1: Command acknowledgment only in the case of errors
2	**0x0C**	Length	Total length of the command = 12, LowByte
3	**0x00**		Total length of the command = 12, HighByte
4	**0x01**	TargetAddress	Address of the controller = 1 (see also Notes to Example 1)
5	**0x04**	TargetPort	Interface on the controller = 4
6	**0x00**	SourceAddress	Address of the host application = 0
7	**0x00**	SourcePort	Port address of the host application = 0
8	**0x00**	Type	0: Command
9	**0x00**	ApplicationHandle	Freely selectable, is returned unchanged in the Response
10	**0x00**	reserved	Reserved, to be assigned by 0
11	**0xF2**	Command	Command code for 0xF2 LIN Monitor – Get Small Buffer Items

Response (including header):

Byte Index	Byte Value	Indication	Description
0	**0x23**	StartByte	0x23 ("#" ASCII character)
1	**0x00**	Flags	No flags set
2	**0x38**	Length	Total length of the response = 12+4 + (2 * 20) = 56, LowByte
3	**0x00**		Total length of the response = 12+4 + (2 * 20) = 56, HighByte
4	**0x00**	TargetAddress	Address of the host application = 0
5	**0x00**	TargetPort	Port address of the host application = 0
6	**0x01**	SourceAddress	Address of the controller = 1
7	**0x04**	SourcePort	Interface on the controller = 4
8	**0x01**	Type	1: Respose
9	**0x00**	ApplicationHandle	The content is sent back unchanged
10	**0x00**	reserved	Reserved
11	**0xF2**	Command	Command code for 0xF2 LIN Monitor – Get Small Buffer Items
12	**0x02**	NumberOfItems	Number of monitor buffer entries (2), LowByte
13	**0x00**		Number of monitor buffer entries (2), MidByte1
14	**0x00**		Number of monitor buffer entries (2), MidByte2
15	**0x00**		Number of monitor buffer entries (2), HighByte
16	**0x42**	Flags	First monitor entry: Bit 1 and Bit 6 are set: Sent LIN frame response (TX) with checksum error
17	**0x07**	Length	First monitor entry: Data length including checksum = 7, i.e. 6 data bytes + 1 checksum byte (the remaining bytes are indefinite)
18	**0x92**	IdCode	First monitor entry: Identifier byte = 0x92 (Identifier 0x12 + parity bit)
19	**0x11**	Data	First monitor entry: Byte 0 = 0x11 = Data byte 0
20	**0x22**		First monitor entry: Byte 1 = 0x22 = Data byte 1
21	**0x33**		First monitor entry: Byte 2 = 0x33 = Data byte 2
22	**0x44**		First monitor entry: Byte 3 = 0x44 = Data byte 3
23	**0x55**		First monitor entry: Byte 4 = 0x55 = Data byte 4
24	**0x66**		First monitor entry: Byte 5 = 0x66 = Data byte 5
25	**0x77**		First monitor entry: Byte 6 = 0x77 = Checksum
26	**0x88**		First monitor entry: Byte 7 = 0x88 (value is indefinite, as Length = 7)
27	**0x99**		First monitor entry: Byte 8 = 0x99 (value is indefinite, as Length = 7)
28	**0x78**	StartTime	First monitor entry: Start time stamp (0x1235678), LowByte
29	**0x56**		First monitor entry: Start time stamp (0x1235678), MidByte1
30	**0x34**		First monitor entry: Start time stamp (0x1235678), MidByte2
31	**0x12**		First monitor entry: Start time stamp (0x1235678), HighByte
32	**0x1B**	BitTimeX8	First monitor entry: eight bit times (16667 corresponds to 416.675 µs ≈ 8/ 19200 Hz), LowByte
33	**0x41**		First monitor entry: eight bit times (16667 corresponds to 416.675 µs ≈ 8/ 19200 Hz), MidByte1
34	**0x00**		First monitor entry: eight bit times (16667 corresponds to 416.675 µs ≈ 8/ 19200 Hz), MidByte2

35	**0x00**		First monitor entry: eight bit times (16667 corresponds to 416.675 µs ≈ 8/ 19200 Hz), HighByte
36	**0x00**	Flags	Second monitor entry: no bits set: received LIN frame response (RX) without error
37	**0x09**	Length	Second monitor entry: Data length including checksum = 9, i.e. 8 Data bytes + 1 Checksum byte
38	**0xA3**	IdCode	Second monitor entry: Identifier byte = 0xA3 (Identifier 0x23 + Parity bit)
39	**0x11**	Data	Second monitor entry: Byte 0 = 0x11 = Data byte 0
40	**0x22**		Second monitor entry: Byte 1 = 0x22 = Data byte 1
41	**0x33**		Second monitor entry: Byte 2 = 0x33 = Data byte 2
42	**0x44**		Second monitor entry: Byte 3 = 0x44 = Data byte 3
43	**0x55**		Second monitor entry: Byte 4 = 0x55 = Data byte 4
44	**0x66**		Second monitor entry: Byte 5 = 0x66 = Data byte 5
45	**0x77**		Second monitor entry: Byte 6 = 0x77 = Data byte 6
46	**0x88**		Second monitor entry: Byte 7 = 0x88 = Data byte 7
47	**0x99**		Second monitor entry: Byte 8 = 0x99 = Checksum
48	**0x01**	StartTime	Second monitor entry: Start time stamp (0xABCDEF01), LowByte
49	**0xEF**		Second monitor entry: Start time stamp (0xABCDEF01), MidByte1
50	**0xCD**		Second monitor entry: Startzeitstempel (0xABCDEF01), MidByte2
51	**0xAB**		Second monitor entry: Start time stamp (0xABCDEF01), HighByte
52	**0x1A**	BitTimeX8	Second monitor entry: eight bit times (16666 corresponds to 416.65 µs ≈ 8/ 19200 Hz), LowByte
53	**0x41**		Second monitor entry: eight bit times (16666 corresponds to 416.65 µs ≈ 8/ 19200 Hz), MidByte1
54	**0x00**		Second monitor entry: eight bit times (16666 corresponds to 416.65 µs ≈ 8/ 19200 Hz), MidByte2
55	**0x00**		Second monitor entry: eight bit times (16666 corresponds to 416.65 µs ≈ 8/ 19200 Hz), HighByte

4.1.9 Bootloader

The smartCAR Controller has a Bootloader for firmware updates and downloading volatile programs to the RAM.

Just after switching on power supply, the Controller is in the Bootloader mode.

In order to change over from the Bootloader mode to the normal operating mode, the 0x10 Software Reset command must be sent to the Controller.

4.1.10 Command Sequence

After switching on, comply the following command sequence:

- 0x10 Software Reset firmware command
- 0x03 Enable Functionalities firmware command

4.2 General Firmware Commands

4.2.1 0x03 Enable Functionalities

This command enables (if available or possible) the following firmware elements for the selected controller:

- Functionalities

Controller selection is made by the TargetAddress parameter in the header of the command.

Find out the enabled firmware elements by the 0xF0 Get Firmware Version command.

This command does not have any command bytes.

4.2.2 0x10 Software Reset

This command resets the selected controller to the initial state.
A software reset is executed for the microcontroller.

Controller selection is made by the TargetAddress parameter in the header of the command.

The command does not have any command bytes.

4.2.3 0xF0 Get Firmware Version

This command is to query the firmware version of the selected controller.
Controller selection is made by the TargetAddress parameter in the header of the command.

The command does not have any command bytes.

Response:

Byte	Indication	Description
0..	Version	Firmware version as 0-terminated string

In the response string there are the following pieces of information:

- Firmware version (version)
- Creation date (date)
- Creation time (time)
- Enabled Functionalities (code)
  enabled by the 0x03 Enable Functionalities command

Code for possible CAN Functionalities:

code: 00000001-00000001-00000000-00000000 --> Diagnostics KWP2000 on TP1.6

code: 00000002-00000002-00000000-00000000 --> Diagnostics KWP2000 on TP2.0

code: 00000004-00000004-00000000-00000000 --> Diagnostics KWP2000 on ISOTP

code: 00000008-00000008-00000000-00000000 --> Diagnostics for GMLAN

code: 00000004-00000010-00000000-00000000 --> Diagnostics UDS on ISOTP

code: 00000010-00000020-00000000-00000000 --> Diagnostics J1939 on J1939 TP

In this representation the code bits for the Diagnostics and the corresponding Transport protocol are concatenated.

Code for possible K-Line Functionalities:

code: 00000000-00010000-00000000-00000000 --> Diagnostics KWP2000

code: 00000000-00020000-00000000-00000000 --> Diagnostics KWP1281

code: 00000000-00040000-00000000-00000000 --> Diagnostics ISO-9141-Ford

In the case of several enabled Functionalities, the resulting code is created by a bit oriented OR concatenation of the individual codes.

4.3 CAN Commands

The CAN commands for your GOEPEL hardware are described in this chapter.

For general information valid for all firmware commands refer to the General Firmware Notes section in this User Manual.

Optional Functionalities

For each CAN interface there are at most the following Optional Functionalities:

- Transport protocol CAN VWTP1.6
- Transport protocol CAN VWTP2.0
- Transport protocol CAN ISOTP
- Transport protocol GMLAN
- Transport protocol J1939
- Diagnostics KWP2000 on TP1.6
- Diagnostics KWP2000 on TP2.0
- Diagnostics KWP2000 on ISOTP
- Diagnostics GMLAN
- Diagnostics UDS on ISOTP
- Diagnostics J1939

After a power-on or software reset, available Optional Functionalities have to be enabled by 0x03 Enable Functionalities.

Then the following firmware commands should be executed in that order:

- 0x12 CAN Init Interface
- 0x1E CAN Node/ BAUD_RATE SET

Initial state:

After a power-on or software reset, the Baudrate of the CAN interfaces is 500 kBaud. If required, the Baudrate can be set to another value by 0x1E CAN Node/ BAUD_RATE SET.

The CAN interfaces are initialized for transferring 11 bit identifiers. Selection between 11 bit and 29 bit identifiers is made by the 0x12 CAN Init Interface command.

4.3.1 0x12 CAN Init Interface

This command resets the selected CAN interface without software reset to the initial state. Additionally, further configuration possibilities are offered.

Interface selection is made by the TargetAddress and TargetPort parameters in the header of the command.

The command bytes are optional. If there are no command bytes, the firmware runs with 0 for the optional command bytes.

Command:

Byte	Indication	Description
0	reserved	Reserved
1	IdMode	0: 11 bit identifiers (default) 1: 29 bit identifiers 2: 11 bit identifiers AND 29 bit identifiers (29 bit identifiers are marked by additional setting the most high bit (**0x80000000**) in the corresponding parameter Id)
2	CANAnalyzerMode	0: Normal Mode (default) 1: Analyzer Mode (no transmitting possible)
3	BlinkMode	0: Flickering of the LEDs deactivated (default) 1: Flickering of the LEDs activated
4	DisableNoAckPauses	0: Compliance of intermissions (100 ms) if no CAN Acknowledge is received (default) 1: No intermission if no CAN Acknowledge is received
5..7	reserved	Reserved

The DisableNoAckPauses parameter is used to deactivate the compliance of intermissions if no dominant bit is received in the acknowledge slot of the sent CAN frames.

With DisableNoAckPauses = 0 an intermission of 100 ms is kept after sending a CAN frame with not received acknowledge for 100 ms uninterruptedly. Sending/ waiting phases result this way.

With DisableNoAckPauses = 1 the intermissions (waiting phases) are suppressed: A CAN frame is sent as long as an acknowledge bit (dominant bit in the acknowledge slot) is received for this CAN frame.

4.3.2 0x14 CAN Set Bus Baudrate

The CAN bus baudrate can be changed at any time.

Setting of the baudrate is useful or necessary just after the 0x10 Software Reset or 0x12 CAN Init Interface commands.

Please refer also to section 0x1E CAN Node regarding setting the Baudrate.

Command:

Byte	Indication	Description
0, 1	Baudrate	Baudrate register value
2	TransceiverType	Type of transceiver module: 0: High Speed 1: Low Speed 2: Single Wire 3: Highlevel Lowspeed (Truck and Trailer)
3	reserved	Reserved

After executing this command the transceiver is ALWAYS in normal mode.

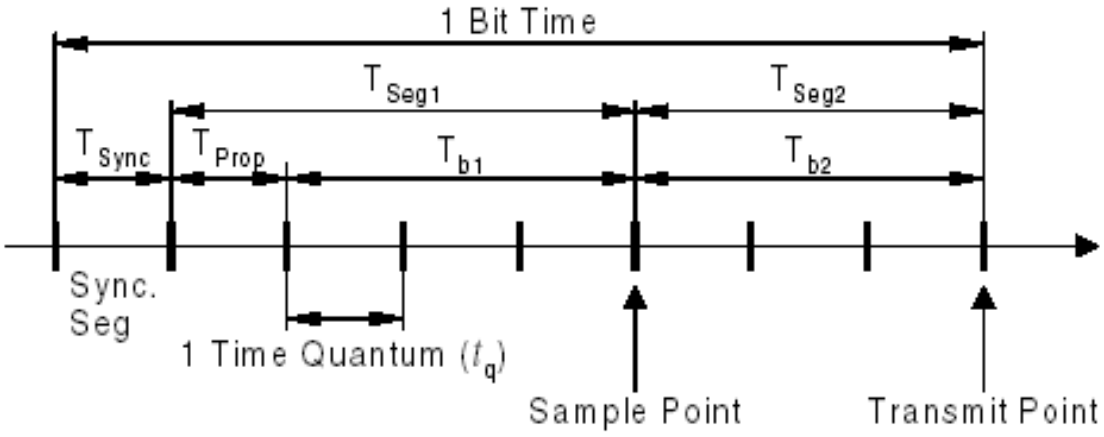

Figure 4-1: Structure of a CAN bit time

Calculation of the baudrate register value:

```
tq = (BRP + 1) / 40000000 Hz                                    (DIV8X = 0)
   = ((BRP + 1) * 8) / 40000000Hz                               (DIV8X = 1)

Tsync = 1 * tq
Tseg1 = (TSEG1 + 1) * tq (minimum 3 tq)
Tseg2 = (TSEG2 + 1) * tq (minimum 2 tq)

BitTime  = Tsync + Tseg1 + Tseg2 (minimum 8 tq)
Baudrate = 1 / BitTime
         = 1 / (Tsync + Tseg1 + Tseg2)
         = 1 / ((3 + TSEG1 + TSEG2) * tq)
         = 40000000Hz / ((BRP + 1) * (3 + TSEG1 + TSEG2))       (DIV8X = 0)
         = 40000000Hz / (8 * (BRP + 1) * (3 + TSEG1 + TSEG2))   (DIV8X = 1)
```

Tseg1, Tseg2, Tsync, tq and BitTime are times, TSEG1, TSEG2, DIV8X, BRP und SJW are bit fields in the CAN bit time register.

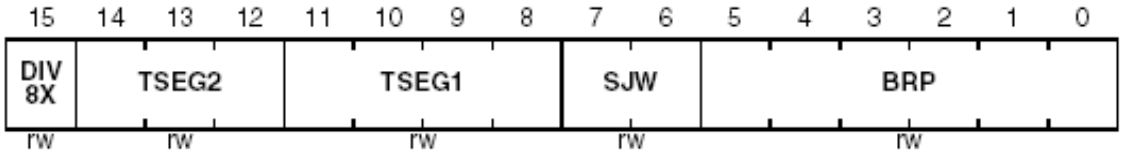

Figure 4-2: CAN bit time register

```
Tsjw  = (SJW + 1) * tq
Tseg1 >= Tsjw + Tprop
Tseg2 >= Tsjw
```

Examples of Register Values:

Register value	Baudrate [kBaud]	Sample Point [%]	Tsjw [tq]	Tseg1 [tq]	Tseg2 [tq]
0xBE89	25.000	80	3	15	4
0xB989	33.333	73,33	3	10	4
0x7A97	83.333	60	3	11	8
0x1667	100	80	2	7	2
0x165F	125	80	2	7	2
0x3447	500	60	2	5	4
0x1647	500	80	2	7	2
0x3443	1000	60	2	5	4
0x1643	1000	80	2	7	2

4.3.3 0x1E CAN Node

This command is intended to configure and control the CAN interface as a CAN Node.

The command is subdivided into several sub-commands distinguishable by the SubCmd parameter.

The command and response structure for all sub-commands is the same for all bytes up to Byte 3, while varieties occur starting with Byte 4 (if more than four bytes exist).

Command and Response:

Byte	Indication	Description
0	SubCmd	1: SET_FLAG_BY_ID 2: GET_FLAG_BY_ID 3: BAUD_RATE__SET 4: BAUD_RATE__GET
1..3	reserved	Reserved (to be set to 0)

FlagId:

Value	Meaning
0x0000	**DISABLE_SOFTWARE_TX_PATH** Deactivation of sending CAN messages. The reception of CAN messages remains unchanged.
0x0001	**DISABLE_NO_ACK_PAUSES** Deactivation of send pauses when no CAN acknowledge is received
0x0002	**DISABLE_BUS_OFF_WAITING** Deactivation of waiting after a bus off, this means the CAN controller is reinitialized immediately after a bus off

4.3.3.1 SET_FLAG_BY_ID

Sub-command with SubCmd = SET_FLAG_BY_ID

The command is used to set or clear an individual flag specified by FlagId.

Command:

Byte	Indication	Description
4, 5	Id	FlagId: Flag identifier (see FlagId in the 0x1E CAN Node section)
6	Value	0: Clear flag 1: Set flag
7	reserved	Reserved

4.3.3.2 GET_FLAG_BY_ID

Sub-command with SubCmd = GET_FLAG_BY_ID

The command is used to query an individual flag specified by FlagId.

Command:

Byte	Indication	Description
4, 5	Id	FlagId: Flag identifier (see FlagId in the 0x1E CAN Node section)
6, 7	reserved	Reserved

Response:

Byte	Indication	Description
4, 5	Id	FlagId: Flag identifier (see FlagId in the 0x1E CAN Node section)
6	Value	0: Flag is cleared 1: Flag is set
7	reserved	Reserved

4.3.3.3 BAUD_RATE SET

Sub-command with SubCmd = BAUD_RATE__SET

The SubCmd = BAUD_RATE__SET is used to set the baudrate.

Command:

Byte	Indication	Description
4..7	BaudRate	Baudrate in baud (e.g. 500000 for 500 kBaud)
8	SamplePoint_Min	Minimum sample point
9	SamplePoint_Max	Maximum sample point
10	NumberOfTimeQuanta_Min	Minimum number of time quanta (1+TSeg1+TSeg2=8..25)
11	NumberOfTimeQuanta_Max	Maximum number of time quanta (1+TSeg1+TSeg2=8..25), use 0 if you don't want to specify this parameter
12	TSeg1_Min	Minimum number of time quanta minus one before sample point (3..16)
13	TSeg1_Max	Maximum number of time quanta minus one before sample point (3..16)
14	TSeg2_Min	Minimum number of time quanta after sample point (2..8)
15	TSeg2_Max	Maximum number of time quanta after sample point (2..8)
16	Sjw_Min	Minimum resynchronization jump width (1..4)
17	Sjw_Max	Maximum resynchronization jump width (1..4)
18, 19	reserved	Reserved

Please note for Bytes 8..17: Set the value to 0 if you don't want to specify this parameter.

Response:

Byte	Indication	Description
4..7	BaudRate	Baud rate in baud (e.g. 500000 for 500 KBaud)
8..11	CanControllerClock	CAN controller clock frequency in Hz
12	SamplePoint	Sample point
13	NumberOfTimeQuanta	Number of time quanta (1+TSeg1+TSeg2)
14	TSeg1	Number of time quanta minus one before sample point (3..16)
15	TSeg2	Number of time quanta after sample point (2..8)
16	Sjw	Resynchronization jump width (1..4)
17..19	reserved	Reserved

4.3.3.4 BAUD_RATE GET

Sub-command with SubCmd = BAUD_RATE__GET

The SubCmd = BAUD_RATE__GET is used to get the baudrate.

Response:

Byte	Indication	Description
4..7	BaudRate	Baud rate in baud (e.g. 500000 for 500 KBaud)
8..11	CanControllerClock	CAN controller clock frequency in Hz
12	SamplePoint	Sample point
13	NumberOfTimeQuanta	Number of time quanta (1+TSeg1+TSeg2)
14	TSeg1	Number of time quanta minus one before sample point (3..16)
15	TSeg2	Number of time quanta after sample point (2..8)
16	Sjw	Resynchronization jump width (1..4)
17..19	reserved	Reserved

4.3.4 0x22 CAN Message Definition

This command defines the CAN message defined by Id.

Command:

Byte	Indication	Description
0..3	Id	Identifier
4, 5	CycleTime	Cycle time in milliseconds (1..32767)
6	Mode	0: No sending of the CAN message 1: Sending of the CAN message
7	PrepareMode	0: No preparing of the CAN message 1: Preparing of the CAN message
8	MessageCount	0: Always sending of the CAN message 1 ≤ N ≤ 255: Sending the CAN message N times
9	Dlc	Data length (0..8)
10..17	Data[0..7]	Data bytes 0..7
18, 19	reserved	Reserved

The setting of PrepareMode is used for parallel starting or stopping several CAN messages (see also 0x28 CAN Start Prepared Messages and 0x29 CAN Stop Prepared Messages).

4.3.5 0x23 CAN Change Prepare Mode

The command is used to change the PrepareMode of the message defined by Id.

In the case of PrepareMode = 1 several messages prepared before can be started/ stopped parallel.
The corresponding commands are 0x28 CAN Start Prepared Messages and 0x29 CAN Stop Prepared Messages.

Command:

Byte	Indication	Description
0..3	Id	Identifier
4	PrepareMode	0: No preparing of the CAN message 1: Preparing of the CAN message
5..7	reserved	Reserved

4.3.6 0x24 CAN Change Message Mode

The command is used to change the setting of the Mode and PrepareMode parameters of the CAN message defined by Id.

Then this CAN message can be sent MessageCount times.

Command:

Byte	Indication	Description
0..3	Id	Identifier
4	Mode	0: No sending of the CAN message 1: Sending of the CAN message
5	PrepareMode	0: No preparing of the CAN message 1: Preparing of the CAN message
6	MessageCount	0: Always sending of the CAN message 1 ≤ N ≤ 255: Sending the CAN message N times
7	reserved	Reserved

4.3.7 0x25 CAN Change Message Data

The command is used to change the setting of the Dlc and Data parameters of the CAN message defined by Id.

Then this CAN message can be sent MessageCount times.

Command:

Byte	Indication	Description
0..3	Id	Identifier
4	ChangeImmediately	0: Data transfer in the cycle 1: Immediate data transfer 2: Immediate data transfer considering a minimum delay (The message is sent at the earliest possible time after processing the minimum delay following the previous transmitting of this message)
5	MessageCount	0: Always sending of the CAN message 1 ≤ N ≤ 255: Sending the CAN message N times
6	Dlc	Data length (0..8)
7	reserved	Reserved
8..15	Mask[0..7]	Mask bytes 0..7
16..23	Data[0..7]	Data bytes 0..7 Data is assumed in accordance with the set mask byte bit positions; in the case no mask byte bits are set, the original data is assumed

In addition, the following parameters are necessary for ChangeImmediately = 2:

Byte	Indication	Description
24, 25	MinimumDelayTime	Minimum delay between the last sending and sending again of the CAN message with the identifier Id
26, 27	reserved	Reserved

4.3.8 0x28 CAN Start Prepared Messages

After calling this command, all CAN messages with PrepareMode = 1 are sent.

The command does not have any command bytes.

See also 0x22 CAN Message Definition.

4.3.9 0x29 CAN Stop Prepared Messages

After calling this command, the sending of all CAN messages with PrepareMode = 1 is stopped.

The command does not have any command bytes.

See also 0x22 CAN Message Definition.

4.3.10 0x2A CAN Delete one Message

Use this command to delete ONE CAN message defined by the 0x22 CAN Message Definition command.

After calling the command, not only sending of the CAN message defined by Id is stopped. Additionally, the CAN message itself is removed from the internal administration.

Sending again is only possible by 0x22 CAN Message Definition.

Command:

Byte	Indication	Description
0..3	Id	Identifier

4.3.11 0x52 CAN Monitor – Receiving Filter Definition

With this command selected identifiers can be detected with the monitor.

If the filter is active, all identifiers between StartId and EndId are filtered. In the case of a deactivated filter (Mode = 0), all identifiers "go through".

In the case only one identifier is to be filtered, use the same value for StartId and EndId (Mode = 1).

Command:

Byte	Indication	Description
0	Mode	0: No filter 1: Filter a range 2: Add a range 3: Remove a range
1..3	reserved	Reserved
4..7	StartId	The Range starts with this identifier
8..11	EndId	The Range ends with this identifier

For 11 bit identifiers any number of ranges can be filtered by calling this command several times (first with Mode = 1, then Mode = 2).

For 29 bit identifiers functionality is limited: Only 10 independent filter ranges can be filtered at most. "Independent filter ranges" means these filter ranges must not touch nor overlap each other.

4.3.12 0x54 CAN Monitor – Activation/ Deactivation

This command activates/ deactivates the monitor according to Mode. When activating, Buffer reception and List reception can be selected.

In the case of Buffer reception more parameters are required.

For Buffer reception, the CAN messages come in in succession into a ring buffer after passing the monitor filter as sent on the bus/ received by the bus. The ring buffer can buffer approximately up to 1024 CAN messages.

In the case of List reception (only 11 bit identifier), a list entry exists for each identifier. This list entry is updated when receiving/ transmitting the identifier. At any time it can be queried specificly by giving the identifier.

Command:

Byte	Indication	Description
0	Mode	0: Deactivating monitor 1: Activating Buffer reception (for the structure see below) 2: Activating List reception (for 11 bit identifiers only)

The following parameters are additionally required for Buffer reception (Mode = 1):

Byte	Indication	Description
1	BufferMode	1: Rx (received messages) 2: Tx (sent messages) 3: Rx + Tx (received and sent messages) 4: ErrorFrames 5: ErrorFrames + Rx 6: ErrorFrames + Tx 7: ErrorFrames + Tx + Rx
2	AutomaticEmpty	0: Empty of the buffer on request (with 0xF1) 1: Empty of the buffer automatically
3	reserved	Reserved

For Mode = 0 or 2 the bytes 1..3 are reserved (and should be initialized with 0).

After activating Buffer reception with AutomaticEmpty = 1, the selected controller independently sends the received CAN frames to the host. Therefore the host has to read out the controller cyclically. In this case, monitor responses have the same structure as the response of the 0xF1 CAN Monitor – Get Buffer Items command.

By activating the monitor with Mode = 1 or 2, the timer TimeStamp to create the time stamps is set to 0.

The following commands can be used to read monitor data:

In the case of Buffer reception with AutomaticEmpty = 0
0xF1 CAN Monitor – Get Buffer Items.

In the case of List reception
0xF2 CAN Monitor – Get List Item.

4.3.13 0x81 CAN TP – Configuration

The Transport Protocol (TP) for the multisession channel defined by Channel is defined by this command.

This firmware command can only be used in the case the transport protocol to be configured has been enabled before by 0x03 Enable Functionalities.

A transport protocol is required to exchange data packets with more than eight bytes of data length (as CAN messages include eight data bytes at most).

The transport protocol is processing a data segmentation to several CAN messages, the failure treatement when transferring data and the temporal adaptation of transmitter and receiver.

This transport protocol can be used for diagnostics among other things.

After selecting a valid transport protocol Type, the corresponding TP task starts.

Command:

Byte	Indication	Description
0	Channel	Multisession channel (starting with 0)
1	Type	Type of Transport protocol: 0: No transport protocol 1: TP1.6 2: TP2.0 3: ISOTP 4: GMLAN 5: J1939 (For the required structures see next pages)
2..3	reserved	Reserved

The following parameters are valid for TP1.6:

Byte	Indication	Description
4	SourceAddress	Own control unit address
5	TargetAddress	Control unit address of the test object
6	BlockSize	Blocksize (0..15, e.g. 3))
7	reserved	Reserved
8..11	SourceSetupId	Own identifier for channel setup (e.g. 0x200 + SourceAddress)
12..15	TargetSetupId	Test object identifier for channel setup (e.g. 0x200 + TargetAddress)
16..19	SourceChannelId	Own identifier for data exchange
20..23	TargetChannelId	Test object identifier for data exchange
24, 25	T1	Time T1 in milliseconds (Acknowledgmet timeout for data telegrams, e.g. 45 ms)
26, 27	T2	Time T2 in milliseconds (maximum time between two sending blocks, e.g. 450 ms)
28, 29	T3	Time T3 in milliseconds (minimum time between two telegrams, e.g. 5 ms)
30, 31	T4	Time T4 in milliseconds (Channel timeout if nothing is sent, e.g. 1000 ms)

The following parameters are valid for TP2.0:

Byte	Indication	Description
4	SourceAddress	Own control unit address
5	TargetAddress	Control unit address of the test object
6	BlockSize	Blocksize (0..15, e.g. 15)
7	ApplicationType	Type of application (for Diagnostics = 1)
8..11	SourceSetupId	Own identifier for channel setup (e.g. 0x200 + SourceAddress) Set SourceSetupId to 0xFFFFFFFF in the case of static channels
12..15	TargetSetupId	Test object identifier for channel setup (e.g. 0x200 + TargetAddress) Set TargetSetupId to 0xFFFFFFFF in the case of static channels
16..19	SourceChannelId	Own identifier for data exchange (the SourceChannelId is completely stipulated by the communication partner in the case of dynamic channels)
20..23	TargetChannelId	Test object identifier for data exchange
24, 25	T1	Time T1 in milliseconds (Acknowledgmet timeout for data telegrams, e.g. 100 ms)
26, 27	T3	Time T3 in milliseconds (minimum time between two telegrams, e.g. 5 ms)

The following parameters are valid for ISOTP:

Byte	Indication	Description
4	Physical-SourceAddress	Own physical control unit address (not necessary for PhysicalAddressingFormat = 0)
5	Physical-TargetAddress	Own physical control unit address (not necessary for PhysicalAddressingFormat = 0)
6	Functional-SourceAddress	Own functional control unit address(only for simulating ECUs, not necessary for FunctionalAddressingFormat = 0)
7	Functional-TargetAddress	Functional control unit address of the test object (not necessary for FunctionalAddressingFormat = 0)
8..11	PhysicalSourceId	Own physical identifier
12..15	PhysicalTargetId	Physical control unit identifier of the test object
16..19	FunctionalSourceId	Own functional identifier
20..23	FunctionalTargetId	Functional control unit identifier of the test object
24	Physical-AddressingFormat	Physical addressing format (generally 0, normal) 0 = normal 1 = extended 2 = mixed
25	Functional-AddressingFormat	Functional addressing format (generally 1, extended) 0 = normal 1 = extended 2 = mixed
26	BlockSize	Blocksize (e.g. 8) 0: No FlowControl frames are expected between the ConsecutiveFrames 1 ≤ N ≤ 255: After sending N ConsecutiveFrames one FlowControl frame is expected
27	SeparationTime	Time between CAN frames to be met by the communication partner, given in milliseconds, e.g. 0 ms
28	Use-OwnSeparationTime	For segmented sending the following time between the CAN messages is met: 0: The value for the SeparationTime received from the communication partner 1: The OwnSeperationTime
29	OwnSeparationTime	Self-meeting time between two CAN messages within a segmented data exchange, given in milliseconds, e.g. 10 ms
30	Flags	Generally all flags are 0. Bit 0 = 0: The SequenceNumber (SN) starts with 1 in the first ConsecutiveFrame (CF) after a FlowControl frame(FC) Bit 0 = 1: StartSNWithZero (The SequenceNumber (SN) starts with 0 in the first ConsecutiveFrame (CF) after a FlowControl frame (FC) Bits 1..7: Reserved
31	reserved	Reserved
32, 33	TimeoutAs	Sending timeout send-site, given in milliseconds, e.g. 250 ms
34, 35	TimeoutAr	Sending timeout receive-site, given in milliseconds, e.g. 250 ms
36, 37	TimeoutBs	FlowControl frame receiving timeout send-site, given in milliseconds, e.g. 250 ms
38, 39	TimeoutCr	ConsecutiveFrame receiving timeout receive-site, given in milliseconds, e.g. 250 ms

The following parameters are valid for GMLAN:

Byte	Indication	Description
4	Physical-SourceAddress	Own physical control unit address (not necessary for PhysicalAddressingFormat = 0)
5	Physical-TargetAddress	Own physical control unit address (not necessary for PhysicalAddressingFormat = 0)
6	Functional-SourceAddress	Own functional control unit address (only for simulating ECUs, not necessary for FunctionalAddressingFormat = 0)
7	Functional-TargetAddress	Functional control unit address of the test object (not necessary for FunctionalAddressingFormat = 0)
8..11	PhysicalRequestId	Physical request identifier
12..15	PhysicalResponseId	Physical response identifier
16..19	AllNode-FunctionalRequestId	Functional request identifier (e.g. 0x101)
20..23	reserved	Reserved
24	Physical-AddressingFormat	Physical addressing format (generally 0, normal) 0 = normal 1 = extended 2 = mixed
25	Functional-AddressingFormat	Functional addressing format (generally 1, normal) 0 = normal 1 = extended 2 = mixed
26	BlockSize	Blocksize (e.g. 8) 0: No FlowControl frames are expected between the ConsecutiveFrames 1 ≤ N ≤ 255: After sending N ConsecutiveFrames one FlowControl frame is expected
27	SeparationTime	Time between CAN frames to be met by the communication partner, given in milliseconds, e.g. 0 ms
28	Use-OwnSeparationTime	For segmented sending the following time between the CAN messages is met: 0: The value for the SeparationTime received from the communication partner 1: The OwnSeperationTime
29	OwnSeparationTime	Self-meeting time between two CAN messages within a segmented data exchange, given in milliseconds, e.g. 10 ms
30, 31	reserved	Reserved
32, 33	TimeoutAs	Sending timeout send-site, given in milliseconds, e.g. 250 ms
34, 35	TimeoutAr	Sending timeout receive-site, given in milliseconds, e.g. 250 ms
36, 37	TimeoutBs	Flow control frame receiving timeout send-site, given in milliseconds, e.g. 250 ms
38, 39	TimeoutCr	ConsecutiveFrame receiving timeout receive-site, given in milliseconds, e.g. 250 ms
40..43	UudtResponseId	UUDT response identifier (UUDT = unacknowledged unsegmented data transfer)

The following parameters are valid for J1939:

Byte	Indication	Description
4	SourceAddress	Own physical control unit address
5	DestinationAddress	Physical control unit address of the test object
6, 7	reserved	Reserved
8..11	TxTimeout	Sending timeout, given in microseconds, e.g. 250000 µs
12..15	RxTimeout	Receiving timeout, given in microseconds, e.g. 250000 µs
16..19	DelayTime	Pause between individual Data Transfer messages given in microseconds, e.g. 5000 µs
20..23	Tr	Timeout for sending Data Transfer messages given in microseconds (should be greater than DelayTime, e.g. 200000 µs)
24..27	Th	Timeout to temporarily interrupt the transmission After expiration of Th at the latest point in time, the transmission is resumed or a request to innterrupt the transmission is sent anew given in microseconds, e.g. 500000 µs
28..31	T1	Timeout for receiving a Data Transfer message given in microseconds, e.g. 750000 µs
32..35	T2	Timeout for receiving the first Data Transfer message after initializing a point-to-point transmission or resuming the transmission given in microseconds, e.g. 1250000 µs
36..39	T3	Timeout for receiving an acknowledge after sending a request to establish the connection or after sending the last Data Transfer message given in microseconds, e.g. 1250000 µs
40..43	T4	Timeout after a temporary interruption of the transmission given in microseconds, e.g. 1050000 µs

If the transport protocol is not needed any more for the indicated multisession channel, call the 0x81 CAN TP – Configuration command once again with Type = 0.

Then the corresponding transport protocol task stops, and claimed resources are available again.

Adressings formats

(PhysicalAddressingFormat and FunctionalAddressingFormat):

normal: There is no address information in the data bytes of a CAN message.

extended: The TargetAddress is in the first data byte of a CAN message.

mixed: For remote diagnostics (only for 29 bit identifier)
The AddressExtension is in the first data byte of a CAN message (the value of the TargetAddress parameter is used for this).

4.3.14 0x82 CAN TP – Multi session Channel Request

The command is used for the dynamical administration of multisession channels.

A multisession channel is requested for and the corresponding Channel number is returned in the response when Success = 1.

This multisession channel administration can be necessary if several applications or software threads do operate with the firmware and share multisession channels.

If always only one application works with the firmware, a multisession channel administration of the application is sufficient (no administration by the firmware is necessary).

The command does not have any command bytes.

Response:

Byte	Indication	Description
0	Success	0 = No multisession channel available 1 = Success, the following channel is available
1	Channel	Multisession channel (starting with 0)
2, 3	reserved	Reserved

4.3.15 0x83 CAN TP – Multi session Channel Release

The command is used for the dynamical administration of multisession channels. The multisession channel defined by Channel is released.

This multisession channel administration can be necessary if several applications or software threads do operate with the firmware and share multisession channels.

If always only one application works with the firmware, a multisession channel administration of the application is sufficient (no administration by the firmware is necessary).

Command:

Byte	Indication	Description
0	Channel	Multisession channel (starting with 0)
1..3	reserved	Reserved

4.3.16 0x8A CAN TP – Send Broadcast Data

Broadcast requests or broadcast responses for the multisession channel defined by Channel are sent by this command.

Command:

Byte	Indication	Description
0	Channel	Multisession channel (starting with 0)
1	Mode	0: Request 1: Request with retriggering 2: Response
2, 3	Length	Data length (number of data bytes)
4.. (3+Length)	Data	Data buffer (0..Length – 1 bytes)

Broadcast requests can be sent repeatedly with Mode = 1 (Request with retriggering).
Then repetition is deactivated by the 0x8C CAN TP – Stop Broadcast Retriggering command.

4.3.17 0x8B CAN TP – Get Broadcast Data

Query data of received broadcast telegrams (requests or responses) by this command.

Command:

Byte	Indication	Description
0	Channel	Multisession channel (starting with 0)
1..3	reserved	Reserved

Response:

Byte	Indication	Description
0	Channel	Multisession channel (starting with 0)
1	reserved	Reserved
2, 3	Length	Data length (number of data bytes)
4.. (3+Length)	Data	Data buffer (0..Length – 1 bytes)

4.3.18 0x8C CAN TP – Stop Broadcast Retriggering

This command stops cyclically transmitted broadcast telegrams started with 0x8A CAN TP – Send Broadcast Data and Mode = 1 (Request with retriggering).

Command:

Byte	Indication	Description
0	Channel	Multisession channel (starting with 0)
1..3	reserved	Reserved

4.3.19 0x8D CAN TP Control

This command is used to control a CAN transport channel. The command is subdivided into several sub-commands distinguishable by the Mode parameter.

The command and response structure for all sub-commands is the same for all bytes up to Byte 3, while varieties occur starting with Byte 4 (if more than four bytes exist).

Command and Response:

Byte	Indication	Description
0	Channel	Multisession channel (starting with 0)
1	Mode	0: Setting the monitor filter; The monitor filter is set automatically that way that CAN messages used by this TP channel pass it; (Previously, all unwanted CAN messages should be blocked by the monitor filter)
2, 3	reserved	Reserved

The following Command parameters are only valid for Mode = 0:

Byte	Indication	Description
4	FilterMode	0: Deactivation of monitor filter setting (the monitor filter remains unchanged) 1: Activation of monitor filter setting
5..7	reserved	Reserved

The following Response parameters are only valid for Mode = 0:

Byte	Indication	Description
4..7	reserved	Reserved

4.3.20 0xA0 CAN Diagnostics – Configuration

Configure the CAN diagnostic protocol for the multisession channel defined by Channel with this command.

The command with Type = 0 is also used to deactivate the complete diagnostics.

This firmware command can only be used in the case the diagnostic protocol to be configured has been enabled before by 0x03 Enable Functionalities.

A transport protocol is prerequisite for the diagnostics (as diagnostic requests and diagnostic responses can have more than eight data bytes of data length, but CAN messages include eight data bytes at most). If a valid Type of diagnostic protocol has been selected, the corresponding diagnostic task starts when executing this 0xA0 CAN Diagnostics – Configuration command.

Command:

Byte	Indication	Description
0	Channel	Multisession channel (starting with 0)
1	Type	Type of diagnostics: 0: No diagnostics 1: Diagnostics KWP2000 on TP1.6 2: Diagnostics KWP2000 on TP2.0 3: Diagnostics KWP2000 on ISOTP 4: Diagnostics for GMLAN 5: Diagnostics UDS on ISOTP 6: Diagnostics J1939 For the required structures see next pages
2	AutomaticEmptyFlags	Bit 0: Sending normal diagnostic responses automatically to the host (see 0xA3 CAN Diagnostics – Get Normal Response Buffer) Bit 1: Sending asynchronous diagnostic responses automatically to the host (see 0xA6 CAN Diagnostics – Get Asynchronous Response Buffer) Bit 2: Sending UUDT diagnostic responses automatically to the host (see 0xA7 CAN Diagnostics – Get UUDT Response Buffer) Bits 3..7: Reserved
3	Mode	0: Set default parameters, with initialization 1: Set parameters, with initialization 2: Only global timeout and setting of flags, no initialisization 3: Set default parameters, no initialization 4: Set parameters, no initialization
4..7	GlobalTimeout	Global timeout in milliseconds (starts directly before sending the requests and stops after complete receiving of the response or after sending the request successfully if no response is expected e.g. 10000 ms)
8..11	Flags	Generally, all flags are 0. Bit 0: Disable21Handling (the BusyRepeatRequest negative response is not treated) Bit 1: Disable23Handling (the RoutineNotComplete negative response is not treated) Bit 2: Disable78Handling (the negative response RequestCorrectlyReceivedResponsePending is not treated) Bit 3: AllResponsesAsSync (also unexpectedly received diagnostic responses are written to the normal diagnostic receiving buffer) Bits 4..31: Reserved

The following parameters are valid for KWP2000 on TP1.6:

Byte	Indication	Description
12, 13	P2max	Timeout given in milliseconds (maximum time between the end of the request and the beginning of the response, e.g. 600 ms)
14, 15	Repetitions	Number of request repetitions if the control unit does not react within the P2max timeout (e.g. 2)
16	AddressWord	Address word for excitation (ECU address + parity bit)
17	TargetAddress	Target address
18	SourceAddress	Source address
19	reserved	Reserved
20, 21	TesterPresentCycle	Cycle for Tester Present in milliseconds (only this parameter of the Tester Present service can be modified, e.g. 2000 ms)
22..23	reserved	Reserved

The following parameters are valid for KWP2000 on TP2.0:

Byte	Indication	Description
12, 13	P2max	Timeout given in milliseconds (maximum time between the end of the equest and the beginning of the response, e.g. 600 ms)
14, 15	Repetitions	Number of request repetitions if the control unit does not react within the P2max timeout (e.g. 2)

The following parameters are valid for KWP2000 on ISOTP:

Byte	Indication	Description
12, 13	P2max	Timeout given in milliseconds (maximum time between the end of the request and the beginning of the response, e.g. 200 ms)
14, 15	P3max	Timeout given in milliseconds (maximum time between the end of the request and the beginning of the response while ResponsePending, e.g. 5000 ms)
16, 17	Repetitions	Number of request repetitions if the control unit does not react within P2max or P3max timeouts (e.g. 2)
18, 19	reserved	Reserved
20..35	TesterPresent	TesterPresent service (for the structure see below)

The following parameters are valid for a TesterPresent entry for KWP2000 on ISOTP:

Byte	Indication	Description
0	Mode	Mode for TesterPresent 0 = deactivated 1 = physical 2 = functional
1	ResponseRequired	Response for TesterPresent is 0 = not expected 1 = expected
2, 3	Cycle	Cycle for TesterPresent in milliseconds (e.g. 1000 ms)
4..6	reserved	Reserved
7	Length	Data length of TesterPresent service (1..8)
8..15	Data	Data of TesterPresent service starting with Service ID, generally 0x3E

The following parameters are valid for GMLAN:

Byte	Indication	Description
12, 13	P2max	Timeout given in milliseconds (maximum time between the end of the request and the beginning of the response, e.g. 200 ms)
14, 15	P3max	Timeout given in milliseconds (maximum time between the end of the request and the beginning of the response while ResponsePending, e.g. 5100 ms)
16, 17	Repetitions	Number of request repetitions if the control unit does not react within the P2max or P3max timeouts (e.g. 2)
18, 19	reserved	Reserved
20..35	TesterPresent	TesterPresent service (for the structure see below)

The following parameters are valid for a TesterPresent entry for GMLAN:

Byte	Indication	Description
0	Mode	Mode for TesterPresent 0 = deactivated 1 = physical 2 = functional
1	ResponseRequired	Response for TesterPresent is 0 = not expected 1 = expected
2, 3	Cycle	Cycle for TesterPresent in milliseconds (e.g. 1000 ms)
4..6	reserved	Reserved
7	Length	Data length of TesterPresent service (1..8)
8..15	Data	Data of TesterPresent service starting with Service ID, generally 0x3E

The following parameters are valid for UDS on ISOTP:

Byte	Indication	Description
12, 13	P2max	Timeout given in milliseconds (maximum time between the end of the request and the beginning of the response, e.g. 200 ms)
14, 15	P3max	Timeout given in milliseconds (maximum time between the end of the request and the beginning of the response while ResponsePending, e.g. 5100 ms)
16, 17	Repetitions	Number of request repetitions if the control unit does not react within the P2max or P3max timeouts (e.g. 2)
18, 19	reserved	Reserved
20..35	TesterPresent	TesterPresent service (for the structure see below)

The following parameters are valid for a TesterPresent entry for UDS on ISOTP:

Byte	Indication	Description
0	Mode	Mode for TesterPresent 0 = deactivated 1 = physical 2 = functional
1	ResponseRequired	Response for TesterPresent is 0 = not expected 1 = expected
2, 3	Cycle	Cycle for TesterPresent in milliseconds (e.g. 1000 ms)
4..6	reserved	Reserved
7	Length	Data length of TesterPresent service (1..8)
8..15	Data	Data of TesterPresent service (starting with Service ID, generally 0x3E 0x00)

The following parameters are valid for J1939:

Byte	Indication	Description
12..15	T1Timeout	Maximum time between the end of the request and the beginning of the response Timeout given in microseconds, e.g. 500000 µs
16..19	T2Timeout	Time between receiving a “Busy” response and sending the request anew Timeout given in microseconds, e.g. 250000 µs
20, 21	Repetitions	Number of request repetitions if the control unit does not react within the T1Timeout, e.g. 2
22, 23	reserved	Reserved
24..37	TesterPresent	TesterPresent service (for the structure see below)

The following parameters are valid for a TesterPresent entry for J1939:

Byte	Indication	Description
0	Mode	Mode for TesterPresent 0 = deactivated 1 = physical
1	ResponseRequired	Response for TesterPresent is 0 = not expected 1 = expected
2	Length	Data length of TesterPresent service (3..11)
3	reserved	Reserved
4..7	Cycle	Cycle for TesterPresent service in microseconds, e.g. 250000 µs
8..18	Data	Data of TesterPresent service (see below, Message Structure for J1939)
19	reserved	Reserved

Message Structure for J1939:

J1939 messages are transmitted with 29 bits CAN identifiers. These identifiers are subdivided into different fields for encoding information regarding addressing or message content.

To define a J1939 message, the Parameter Group Number (PGN) must be given in the first three bytes, e.g. 0x00D900 for a Memory Access Request. Then the data bytes follow.

In the case of functional addressing (PGN ≥ 0xF0), additionally the Group Extension (GE) must be given in the third byte.

If new memory content is to be transferred to the ECU via Memory Access Request – Write, this memory content must be appended to the actual data bytes of the Memory Access Request (see the following example).

Example: Writing the data sequence 0x010203 via Memory Access Request – Write:

00	D9	00	01	15	00	00	00	80	FF	FF	01	02	03
PGN			Data bytes of Memory Access Request								Data to be written (New memory content)		

Received J1939 messages have the same structure.

As a diagnostics refers to a transport protocol, the corresponding transport protocol must be selected by 0x81 CAN TP – Configuration before starting the diagnostics with the 0xA0 CAN Diagnostics – Configuration command.

If the diagnostics is not needed any more for the indicated multisession channel, call the 0xA0 CAN Diagnostics – Configuration command once again with Type = 0.
Then the corresponding diagnostic task stops, and claimed resources are available again.

The following command sequence results for the diagnostics:

- In the case multisession channel administration is used: Request for the multisession channel by 0x82 CAN TP – Multi session Channel Request
- Select the transport protocol with 0x81 CAN TP – Configuration
- Select the diagnostics with 0xA0 CAN Diagnostics – Configuration
- Use diagnostic with the corresponding diagnostic commands
- Stop the diagnostics with 0xA0 CAN Diagnostics – Configuration and Type = 0
- Stop transport protocol with 0x81 CAN TP – Configuration and Type = 0
- In the case multisession channel administration is used: Release the multisession channel by 0x83 CAN TP – Multi session Channel Release

Addressing modes:
physical: Communication with an individual ECU (point-to-point-connection, Unicast)
functional: Communication with a group of ECUs (point-to-multipoint-connection, Broadcast)

4.3.21 0xA1 CAN Diagnostics – Start Session

This command starts a CAN diagnostic session for the multisession channel defined by Channel.

Additionally, the diagnostic connection is established.

Command:

Byte	Indication	Description
0	Channel	Multisession channel (starting with 0)
1	Mode	0: Physical addressing 1: Functional addressing **In addition:** No response to the request is necessary if the most significant bit is set (0x80)
2, 3	Length	Request length (0..(PARAM_SIZE – 4)) (for Length = 0 no request is sent)
4.. (3+Length)	Request	General CAN request: Consists of SID (service identifier) and data J1939 request: See Message Structure for J1939 in the 0xA0 CAN Diagnostics – Configuration section

4.3.22 0xA2 CAN Diagnostics – Send Request

This command is used to send a CAN diagnostic request for the multisession channel defined by Channel.

Prerequisite is the successful execution of the 0xA1 CAN Diagnostics – Start Session command before, and the diagnostic connection must NOT have been disconnected later.

It is necessary to execute this command several times in order to send larger diagnostic requests (e.g. 1100 bytes) caused by the size of the command (limited by MESSAGE_SIZE). In this case the Concatenate and Send parameters must be set accordingly.

Command:

Byte	Indication	Description
0	Channel	Multisession channel (starting with 0)
1	Mode	0: Physical addressing 1: Functional addressing **In addition:** No response to the request is necessary if the most significant bit is set (0x80)
2	Send	0 = No sending (only buffer filling) 1 = Sending
3	Concatenate	0 = Write from buffer beginning 1 = append
4	Segmentation	Segmentation flag für segmentation on diagnostic level 0 = Request not segmented 1 = Request segmented
5	reserved	Reserved
6, 7	Length	Request length (1..(PARAM_SIZE – 8))
8.. (7+Length)	Request	General CAN request: Consists of SID (service identifier) and data J1939 request: See Message Structure for J1939 in the 0xA0 CAN Diagnostics – Configuration section

The Segmentation flag refers to the diagnostic protocol.
As a rule it must not be set by a diagnostic tester.

4.3.23 0xA3 CAN Diagnostics – Get Normal Response Buffer

Query the normal CAN diagnostic response buffer for the multisession channel defined by Channel with this command.

Only expected diagnostic responses come into the normal diagnostic response buffer as the corresponding diagnostic request was sent before.

If a diagnostic response does not fit into a single response, the host has to call this command several times to fetch the remaining responses. The last of these responses contains the value "0" in the RemainingLength parameter.

In addition, the buffer should be read out as long as the Segmentation bit, the Busy bit or the BufferNotEmpty bit of Flags are set.

Command:

Byte	Indication	Description
0	Channel	Multisession channel (starting with 0)
1..3	reserved	Reserved

Response:

Byte	Indication	Description
0	Channel	Multisession channel (starting with 0)
1	LastErrorCode	Error code (0 = no error)
2	Flags	Bit 0 = 0: No segmentation on diagnostic level Bit 0 = 1: Segmentation (on diagnostic level) Bit 1 = 0: Idle Bit 1 = 1: Busy (a request has not been responded yet or successfully sent) Bit 2 = 0: Invalid (this buffer entry is invalid Bit 2 = 1: Valid (this buffer entry is valid) Bit 3 = 0: The normal diagnostic response buffer is empty Bit 3 = 1: BufferNotEmpty (the normal diagnostic response buffer is not empty yet) Bits 4..7: Reserved
3	State	State of diagnostics: 0: Not initialized 1: No connection 2: Connection is being established 3: Connection was established 4: Connection is released
4, 5	Length	Number of response bytes (0..(PARAM_SIZE – 8))
6, 7	RemainingLength	Number of remaining response bytes
8.. (7+Length)	Response	General CAN response: Consists of SID (service identifier) and data J1939 response: See Message Structure for J1939 in the 0xA0 CAN Diagnostics – Configuration section

4.3.24 0xA4 CAN Diagnostics – Stop Session

This command stops a running CAN diagnostic session for the multisession channel defined by Channel.

Additionally, the diagnostic connection is released.

To stop the complete diagnostics, call the 0xA0 CAN Diagnostics – Configuration command again with Type = 0.

Command:

Byte	Indication	Description
0	Channel	Multisession channel (starting with 0)
1	Mode	0: Physical addressing 1: Functional addressing **In addition:** No response to the request is necessary if the most significant bit is set (0x80)
2, 3	Length	Request length (0..(PARAM_SIZE – 4)) (for Length = 0 no request is sent)
4.. (3+Length)	Request	General CAN request: Consists of SID (service identifier) and data J1939 request: See Message Structure for J1939 in the 0xA0 CAN Diagnostics – Configuration section

4.3.25 0xA5 CAN Diagnostics – Get State

Query the CAN diagnostic state for the multisession channel defined by Channel with this command.
Additionally, the firmware internal LastErrorCode can be reset.

The value of the LastErrorCode in the Response corresponds to the value of the firmware internal LastErrorCode before its resetting.

Generally the firmware internal LastErrorCode is reset automatically without calling this 0xA5 CAN Diagnostics – Get State command by starting a diagnostic session by 0xA1 CAN Diagnostics – Start Session as well as stop of a diagnostic session with 0xA4 CAN Diagnostics – Stop Session and Length ≠ 0.

Command:

Byte	Indication	Description
0	Channel	Multisession channel (starting with 0)
1	ResetLastError	0 = Do not reset LastErrorCode 1 = Reset LastErrorCode
2, 3	reserved	Reserved

Response:

Byte	Indication	Description
0	Channel	Multisession channel (starting with 0)
1	LastErrorCode	Error code (0 = no error)
2	DiagType	Type of diagnostics: 0: No diagnostics 1: Diagnostics KWP2000 on TP1.6 2: Diagnostics KWP2000 on TP2.0 3: Diagnostics KWP2000 on ISOTP 4: Diagnostics GMLAN 5: Diagnostics UDS on ISOTP 6: Diagnostics J1939
3	State	State of diagnostics: 0: Not initialized 1: No connection 2: Connection is being established 3: Connection was established 4: Connection is released
4	Flags	Bit 0 = 0: Idle Bit 0 = 1: Busy (a request has not been responded yet or successfully sent) Bit 1 = 0: The normal diagnostic response buffer is empty Bit 1 = 1: SyncRxBufferNotEmpty (the normal diagnostic response buffer is not empty yet) Bit 2 = 0: The asynchronous diagnostic response buffer is empty Bit 2 = 1: AsyncRxBufferNotEmpty (the asynchronous diagnostic response buffer is not empty yet) Bit 3 = 0: The UUDT diagnostic response buffer is empty Bit 3 = 1: UudtRxBufferNotEmpty (the UUDT diagnostic response buffer is not empty yet) Bits 4..7: Reserved
5..7	reserved	Reserved

4.3.26 0xA6 CAN Diagnostics – Get Asynchronous Response Buffer

If the controller receives an unexpected CAN diagnostic response, in normal cases this response is written by the firmware to a separate diagnostic response buffer, the Asynchronous Diagnostic Response Buffer.

Query this buffer for the multisession channel defined by Channel with the 0xA6 CAN Diagnostics – Get Asynchronous Response Buffer command.

If a diagnostic response does not fit into a single response, the host has to call this command several times to fetch the remaining responses. The last one of these responses contains the value "0" in the RemainingLength parameter.

In addition, the buffer should be read out as long as the Segmentation bit or the BufferNotEmpty bit of Flags are set.

Command:

Byte	Indication	Description
0	Channel	Multisession channel (starting with 0)
1..3	reserved	Reserved

Response:

Byte	Indication	Description
0	Channel	Multisession channel (starting with 0)
1	LastErrorCode	Error code (0 = no error)
2	Flags	Bit 0 = 0: No segmentation on diagnostic level Bit 0 = 1: Segmentation (on diagnostic level) Bit 1 = 0: Idle Bit 1 = 1: Busy (a request has not been responded yet or successfully sent) Bit 2 = 0: Invalid (this buffer entry is invalid Bit 2 = 1: Valid (this buffer entry is valid) Bit 3 = 0: The asynchronous diagnostic response buffer is empty) Bit 3 = 1: BufferNotEmpty (the asynchronous diagnostic response buffer is not empty yet) Bits 4..7: Reserved
3	State	State of diagnostics: 0: Not initialized 1: No connection 2: Connection is being established 3: Connection was established 4: Connection is released
4, 5	Length	Number of response bytes (0..(PARAM_SIZE – 8))
6, 7	RemainingLength	Number of remaining response bytes
8.. (7+Length)	Response	General CAN response: Consists of SID (service identifier) and data J1939 response: See Message Structure for J1939 in the 0xA0 CAN Diagnostics – Configuration section

4.3.27 0xA7 CAN Diagnostics – Get UUDT Response Buffer

When the controller receives a CAN UUDTdiagnostic response (e.g. in the case of GMLAN), this response is written by the firmware to a separate diagnostic response buffer, the UUDT Diagnostic Response Buffer.

Query this buffer for the multisession channel defined by Channel with the 0xA7 CAN Diagnostics – Get UUDT Response Buffer command.

Command:

Byte	Indication	Description
0	Channel	Multisession channel (starting with 0)
1..3	reserved	Reserved

Response:

Byte	Indication	Description
0	Channel	Multisession channel (starting with 0)
1..3	reserved	Reserved
4, 5	NumberOf-Responses	Number of UUDT diagnostic responses (N)
6, 7	NumberOf-Remaining-Responses	Number of remaining responses
8.. 7+(12*N)	Responses	UUDT diagnostic responses (for the structure see below)

A UUDT diagnostic response has the following structure:

Byte	Indication	Description
0	Dlc	Data length (0..8)
1	Flags	Bit 0: BufferOverrun Bits 1..7: Reserved
2, 3	reserved	Reserved
4..11	Data[0..7]	Data bytes 0..7

The pure data of a UUDT diagnostic response (Data) has a different structure compared with usual diagnostic data:
As a rule the response service Id (RespSID) and possible negative response codes (negRespCode) are not transferred on the CAN bus.

Additionally, UUDT CAN messages can be sent parallel to USDT CAN messages (normal diagnostic CAN messages) on the CAN bus.
That means possible overlapping and: Between segmented transmission (and the USDT CAN messages) also UUDT CAN messages can be transferred!

4.3.28 0xB0 CAN TX-FIFO – Reset

Prior to sending with the Sending FIFO functionality, this command should be executed to reset that functionality.

Moreover executing the command is necessary e.g. after a bus short.

Command:

Byte	Indication	Description
0..3	reserved	Reserved

The Sending FIFO functionality serves to send any CAN messages as fast as possible.

A sending FIFO is necessary as the CAN messages come in packets from the PC, but are sent in succession with different speeds by the firmware according to the CAN bus baudrate and arbitration.

CAN messages are sent immediately or input into the TX-FIFO by the 0xB1 CAN TX-FIFO – Send one Message or 0xB2 CAN TX-FIFO – Send several Messages commands.

Query the TX-FIFO state by the 0xB3 CAN TX-FIFO – Get State command.

4.3.29 0xB1 CAN TX-FIFO – Send one Message

This command is used to send ONE CAN message with the Sending FIFO functionality.

The submitted CAN message is either sent immediately or input into the sending FIFO in the case a transmission with the Sending FIFO functionality is currently executed.

Command:

Byte	Indication	Description
0..3	Id	Identifier
4	Dlc	Data length (0..8)
5..7	reserved	Reserved
8..15	Data[0..7]	Data bytes 0..7

4.3.30 0xB2 CAN TX-FIFO – Send several Messages

This command is used to send SEVERAL CAN messages with the Sending FIFO functionality.

The CAN message submitted first is either sent immediately or input into the sending FIFO as all other submitted CAN messages.

Command:

Byte	Indication	Description
0..3	NumberOfItems	Number of CAN messages to be transmitted (N)
4.. 3+(N*16)	Items	CAN messages (regarding the structure see below)

A CAN message consists of the following 16 bytes:

Byte	Indication	Description
0..3	Id	Identifier
4	Dlc	Data length (0..8)
5..7	reserved	Reserved
8..15	Data[0..7]	Data bytes 0..7

4.3.31 0xB3 CAN TX-FIFO – Get State

Query the state of the Sending FIFO functionality by this command.

The command does not have any command bytes.

Response:

Byte	Indication	Description
0..3	NumberOf-FreeEntries	Number of unused sending entries
4..7	NumberOf-UsedEntries	Number of used sending entries

Together with the 0xB2 CAN TX-FIFO – Send several Messages command this command is required to send many CAN messages (e.g. 5,000) as fast as possible without delays if possible (only the pure transmission times remain).

4.3.32 0xF1 CAN Monitor – Get Buffer Items

The command is used to query monitor buffer items.
This command does not have any command bytes.

Response:

Byte	Indication	Description
0..3	NumberOfItems	Number of monitor buffer items
4..	Items	Monitor buffer entries (for the structure see below)

A monitor buffer entry has the following structure:

Byte	Indication	Description
0..3	TimeStamp	Time stamp with a resolution according to TimeStampResolution
4..7	Id	Identifier
8	Flags	Bit 0 = 0: 11 bits identifier (STD) Bit 0 = 1: 29 bits identifier (XTD) Bit 1 = 0: Received CAN message (RX) Bit 1 = 1: Sent CAN message (TX) Bit 2 = 0: No ErrorFrame Bit 2 = 1: ErrorFrame Bit 3: Reserved Bit 4 = 0: No event Bit 4 = 1: Event Bits 5, 6: Reserved Bit 7 = 0: No Bufferoverrun Bit 7 = 1: Bufferoverrun
9	Dlc	Data length (0..8)
10	TimeStampResolution	Time stamp resolution: 0: 10 microseconds 1: 400 nanoseconds
11	reserved	Reserved
12..19	Data[0..7]	Data bytes 0..7

The monitor buffer items are always sized 20 bytes, irrespective of the Dlc data length.

Error-Frames (bit 2 in Flags = 1) are additionally marked by Id = 0xFFFFFFFF, and in the data byte 0 (Data[0]) there is the LEC (LastErrorCode) of the on-chip CAN controller located on the chip of the 32-bit microcontroller:

Last Error Code	Meaning
0	No error
1	Stuff Error: More than five equal bits in a sequence have occurred in a part of a received message where this is not allowed.
2	Form Error: A fixed format part of a received frame has the wrong format.
3	Ack Error: The transmitted message was not acknowledged by another node.
4	Bit1 Error: During a message transmission, the CAN node tried to send a recessive level (1), but the monitored bus value was dominant (outside the arbitration field and the acknowledge slot).
5	Bit0 Error: Two different conditions are signaled by this code: a) During transmission of a message (or acknowledge bit, active error flag, overload flag), the CAN node tried to send a dominant level (0), but the monitored bus value was recessive (1). b) During bus-off recovery, this code is set each time a sequence of 11 recessive bits has been monitored. The CPU may use this code as indication, that the bus is not continuously disturbed.
6	CRC Error: The CRC checksum of the received CAN message was incorrect.
7..255	reserved

Events (bit 4 in Flags = 1) are additionally marked by Id = 0xFFFFFFFE, and in the data byte 0 (Data[0]) there is the event with the following meaning:

LEC	Description
0	Rising flank at the trigger input
1	Falling flank at the trigger input
2..255	reserved

4.3.33 0xF2 CAN Monitor – Get List Item

This command is to query one monitor list item of the CAN message indicated by Id.

Command:

Byte	Indication	Description
0..3	Id	Identifier

Response:

Byte	Indication	Description
0..3	Id	Identifier (always the same as that of the command)
4..7	TimeStamp	Time stamp with a resolution according to TimeStampResolution
8..11	MessageCount	Statement, how often the queried identifier was sent or received
12	Flags	Bit 0 = 0: 11 bits identifier (STD) Bit 0 = 1: 29 bits identifier (XTD) Bit 1 = 0: Received CAN message (RX) Bit 1 = 1: Sent CAN message (TX) Bits 2..7: Reserved
13	Dlc	Data length (0..8)
14	TimeStampResolution	Time stamp resolution: 0: 10 microseconds 1: 400 nanoseconds
15	reserved	Reserved
16..23	Data[0..7]	Data bytes 0..7

4.4 LIN Commands

The LIN commands for your GOEPEL hardware are described in this chapter.

For general information valid for all firmware commands refer to the General Firmware Notes section in this User Manual.

After a power-on or software reset, the following firmware commands should be executed in that order:

- 0x12 LIN Init Interface
- 0x14 LIN Set Interface Properties
- 0x15 LIN Set Checksum Model
  (in the case of LIN2.0 for the individual identifiers)

for LIN master:

- 0x81 LIN Relays – Setting
  (for switching on to master operation)
- 0x22 LIN Fill Schedule Table
- 0x23 LIN Fill Frame Response Table
- 0x28 LIN Master – Start Transmitting

for LIN slave:

- 0x23 LIN Fill Frame Response Table

Initial state:

After a power-on or software reset, the master task is deactivated, while the slave task is activated (for monitoring).

The slave task operates with an automatic baud rate recognition, a BreakDetectionThreshold of 9.5 bit times and with the classic checksum model (see Explanation). This way the LIN bus can be monitored with the LIN bus monitor without announcing the baud rate (0x54 LIN Monitor – Activation/ Deactivation command).

The firmware internal ResponseSpace and InterByteSpace variables are initialized by 0. However, on the LIN bus ResponseSpace and InterByteSpace appear with 0..1 bit times, as the UART is used for transmitting.

Explanation:

Classic checksum = checksum via data bytes

Enhanced checksum = checksum via identifier byte and data bytes

The firmware operates with a value of 9.5 bit times for the BreakDetectionThreshold in contrast to the LIN Specification (11 bit times) for bus monitoring, to be able to detect breaks shorter than 11 bit times.

At any time, the value for the BreakDetectionThreshold can be modified by the 0x46 LIN Set Break DetectionThreshold command.

Structure of a LIN Cluster

The following information is taken from the LIN Specification 2.0.

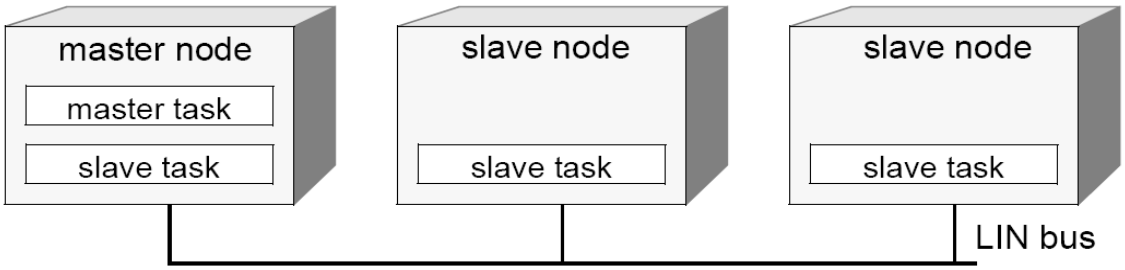

Figure 4-3: Structure of a LIN cluster

A LIN cluster consists of one master task and several slave tasks.

The LIN master (master node) includes one master task and one slave task.

Each LIN slave (slave node) includes only one slave task.

Structure of a LIN Frame

The following information is taken from the LIN Specification 2.0.

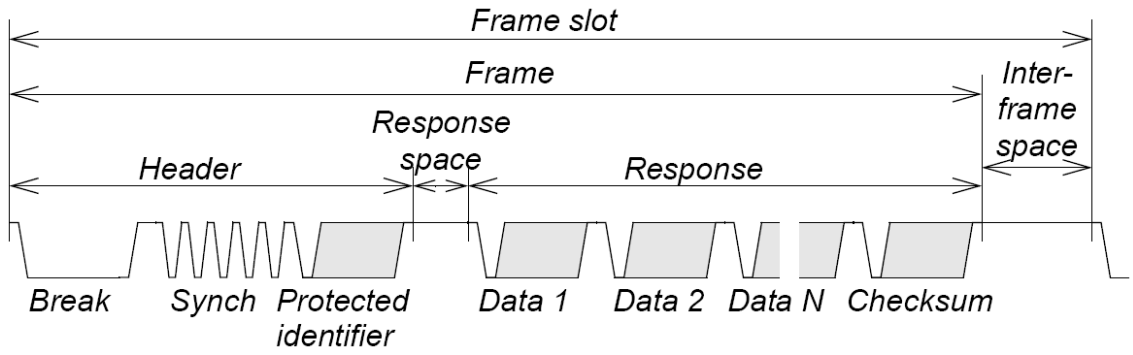

Figure 4-4: Structure of a LIN Frame

Essentially a LIN Frame (LIN message) consists of the LIN Frame Header and the LIN Frame Response.

The Header is exclusively sent by the master task of the LIN master (master node).

The Response is sent by a slave task of the master node or of a slave node.

The pause between Header and Response is called Response space.

The following example shows a LIN message of two data bytes, consisting of Header and Response.

All time-parameters whithin a LIN message changeable by the firmware can be seen (BreakTime, BreakDelimiterTime, ArbitrationTime, ResponseSpace and InterByteSpace):

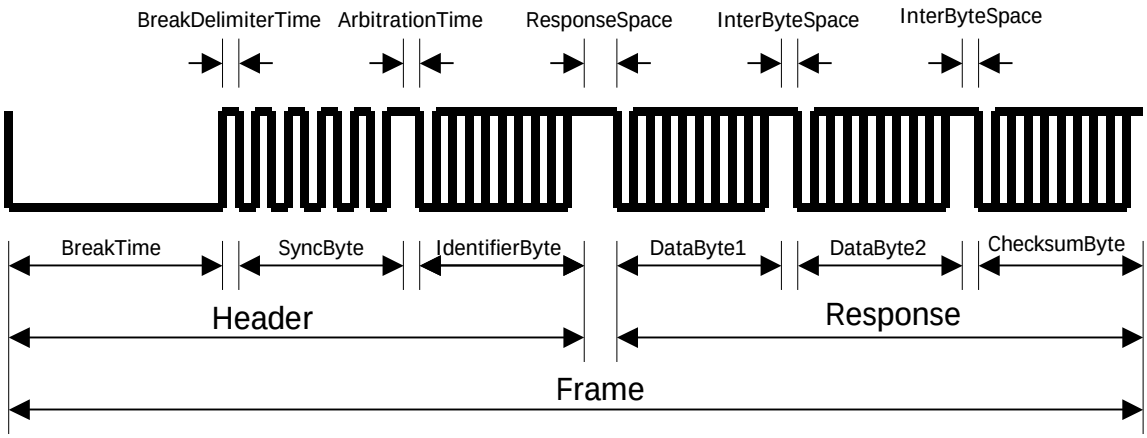

Figure 4-5: LIN message

The continous lines above and below the LIN signal indicate that this signal can have V_{Bat} (high) level as well as Gnd (low) level during the corresponding times.

4.4.1 0x12 LIN Init Interface

This command resets the selected LIN interface without software reset to the initial state. Additionally, further configuration possibilities are offered.
Interface selection is made by the TargetAddress and TargetPort parameters in the header of the command.

The command bytes are optional. If there are no command bytes, the firmware runs with 0 for the optional command bytes.

Command:

Byte	Indication	Description
0	reserved	Reserved
1	DontUseUartForTx	0: Use UART for transmitting (default) 1: No use of UART for transmitting
2	ResetRelays	0: No reset of the relays (default) 1: Reset of the relays
3	BlinkMode	0: Flickering of the LEDs deactivated (default) 1: Flickering of the LEDs activated

For sending without jitters (ResponseSpace Jitters) and modifications of ArbitrationTime, ResponseSpace and InterByteSpace, set the DontUseUartForTx parameter to 1, as in the case of sending with UART delays (jitters) may occur up to one bit time.

For other cases, sending with UART means only little CPU load.

4.4.2 0x14 LIN Set Interface Properties

The properties of the selected LIN interface are set with this command.

Command:

Byte	Indication	Description
0	EnableMasterTask	0: Deactivating the master task 1: Activating the master task (for the structure see below)
1	EnableSlaveTask	0: Deactivating the slave task 1: Activating the slave task (for the structure see below) **Note:** For monitoring the slave task must be activated, that means, the EnableSlaveTask parameter is to be set to 1.
2, 3	reserved	Reserved

Parameters of the master task:

Byte	Indication	Description
4..7	BaudRate	BaudRate in Hz
8..11	BreakTime	BreakTime in multiples of 25 ns (generally 13 Bit-times, e.g. 27083 for 19200 Baud) (Time that announces the start of a new LIN frame, duration of the low level of the Break, see Figure 4-5)
12..15	BreakDelimiterTime	BreakDelimiterTime in multiples of 25 ns (generally 1 Bit-time, e.g. 2083 for 19200 Baud) (Time between the end of the low level of the Break and the beginning of the StartBit of the SyncByte or duration of the high level of the BreakDelimiter, see Figure 4-5)
16..19	ArbitrationTime	only for Advanced library, otherwise to be initialized by 0 ArbitrationTime in multiples of 25 ns (generally 0) (Time between the end of the StopBit of the SyncByte and the beginning of the StartBit of the IdentifierByte, see Figure 4-5)

Parameters of the slave task:

Byte	Indication	Description
20	EnableBaudRate-Detection	0: Baudrate detection deactivated 1: Baudrate detection activated
21..23	reserved	Reserved
24..27	BaudRate	BaudRate in Hz
28..31	ResponseSpace	ONLY for Advanced library, otherwise to be initialized by 0 ResponseSpace in multiples of 25 ns (generally 0) (Time between the end of the StopBit of the IdentifierByte and the beginning of the StartBit of the Response or time between Header and Response, see Figure 4-5)
32..35	InterByteSpace	ONLY for Advanced library, otherwise to be initialized by 0 InterByteSpace in multiples of 25 ns (generally 0) (Time between the end of the StopBit of a Response DataByte and the beginning of the StartBit of the next Response DataByte, see Figure 4-5)

The monitor commands do NOT provide results if the slave task is deactivated!!!

Please note the following: The UART should NOT be used for transmitting to create the ArbitrationTime, ResponseSpace and InterByteSpace times (see 0x12 LIN Init Interface command).

Please note also: The sending moment for the LIN Frame response, defined by ResponseSpace, is falsified by the transceiver running times (receiving and transmitting running time).
The complete running time of a transceiver depends on its type. The range is between 5 and 15 µs (generally 8 to 9 µs).

4.4.3 0x15 LIN Set Checksum Model

Use this command to set the checksum model.

Command:

Byte	Indication	Description
0	CheckSumModel	0: Classic (checksum only via data bytes) 1: Enhanced (checksum via identifier and data bytes)
1	NumberOfIds	Number of identifiers (N)
2	SelectAll	0: Setting is valid only for the identifiers indicated by Ids 1: Setting is valid for all identifiers
3	reserved	Reserved
4.. (3+N)	Ids	Identifier list (one byte per identifier)

4.4.4 0x22 LIN Fill Schedule Table

Use this command to fill a LIN Schedule table.

Altogether there are 16 Schedule tables with 256 entries each in the firmware.

The command has to be executed several times if the table to be filled has more entries as fit into the command itself. In this case the Concatenate parameter must be set accordingly.

Command:

Byte	Indication	Description
0	ScheduleTable-Number	Number of the Schedule table (0..15)
1	Concatenate	0: Write from the beginning of the table 1: Append table entries
2	IdAsIdCode	0: The protected identifier is calculated according to the specification by the firmware 1: The protected identifier is assumed as submitted (sendig an invalid identifier is possible then!)
3..5	reserved	Reserved
6, 7	NumberOfItems	Number of table entries (N)
8.. 7+(N*8)	Items	Table entries (for the structure see below)

A Schedule table Item consists of the following eight bytes:

Byte	Indication	Description
0	Id	Identifier
1	FrameType	0: UnconditionalFrame – „normal“ messages 1: EventTriggeredFrame – event triggered messages 2: SporadicFrame – sporadic messages 3: DiagnosticFrame – diagnostic messages (0x3C and 0x3D)
2	FrameListIndex	List index for Event Triggered Frames and Sporadic Frames (starting with 0)
3	reserved	Reserved
4..7	Delay	Delay to the next LIN frame in multiples of 25 ns (e.g. 400000 for a Delay of 10 ms)

The Delay range of values is 10000..0xFFFFFF (16777215), that means 24 bits only.

That correponds to a
minimum delay of 250 µs and a
maximum delay of about 419 ms.

Similar to the Frames list for the Unconditional Frames of a LIN Description File (LDF file), there is possibly an Event_triggered_frames list for Event Triggered Frames or a Sporadic_frames list for Sporadic Frames.

FrameListIndex is the index in these lists.

4.4.5 0x23 LIN Fill Frame Response Table

Use this command to fill the LIN Frame response table.

The command has to be executed several times if the table to be filled has more entries as fit into the command itself.

The command automatically defines the LIN Frame responses given in the individual table Items by Id.

Command:

Byte	Indication	Description
0, 1	NumberOfItems	Number of table entries (N)
2, 3	reserved	Reserved
4.. 3+(N*12)	Items	Table entries (for the structure see below)

A LIN Frame response table Item consists of the following 12 Bytes:

Byte	Indication	Description
0	Id	Identifier (0x00..0x3F)
1	Length	Data length
2, 3	reserved	Reserved
4..11	Data[0..7]	Data bytes 0..7

As an alternative to this command there is the 0x30 LIN Frame Response Definition command, offering further features.

4.4.6 0x24 LIN Send WakeUp Request

Use this command to send one wakeup request.

Command:

Byte	Indication	Description
0..3	WakeUpTime	Duration of the dominant bus level in multiples of 25 ns, (e.g. 10000 for eine WakeUp time of 250 µs) (0 = Default WakeUpTime)

In the case of 0 as WakeUpTime, the default WakeUpTime is eight bit times. According to LIN 2.0 Specification, the WakeUpTime should have a value of 250 µs to 5 ms.

4.4.7 0x25 LIN Set Slave Task State

This command sets the slave controller in the sleep state or “awakes” it from that state.

Command:

Byte	Indication	Description
0	Awake	0: Sleep state 1: Awake state
1..3	reserved	Reserved

4.4.8 0x28 LIN Master – Start Transmitting

The master task starts processing the indicated LIN Schedule table. That means the corresponding LIN frame headers are sent.

Command:

Byte	Indication	Description
0	ScheduleTableNumber	Number of the Schedule table
1..3	reserved	Reserved

The indicated Schedule table is always processed starting with the FIRST entry.

4.4.9 0x29 LIN Master – Stop Transmitting

The master task stops to send LIN frame headers.

This command does not have any command bytes.

4.4.10 0x2A LIN Clear Schedule Table

Empty the indicated Schedule table by this command.

Command:

Byte	Indication	Description
0	ScheduleTableNumber	Number of the Schedule table
1..3	reserved	Reserved

4.4.11 0x2B LIN Remove Frame Response Table Items

With this command it is possible to remove all or only the entries defined by Ids from the LIN Frame response table.

Command:

Byte	Indication	Description
0, 1	NumberOfIds	Number of identifiers (N)
2	SelectAll	0: Remove only table items indicated by Ids 1: Remove all table items
3	reserved	Reserved
4.. (3+N)	Ids	Identifier list (one byte per identifier)

4.4.12 0x30 LIN Frame Response Definition

Use the command to define the LIN Frame response indicated by Id.

Defining a LIN Frame response by this command offers an alternative to 0x23 LIN Fill Frame Response Table to send LIN Frame responses. But this command offers further features, e.g. a limited sending number (MessageCount ≠ 0).

Please pay attention that a LIN Frame response is only sent if the master task (on this or on another LIN node) sent the corresponding LIN frame header before.

Command:

Byte	Indication	Description
0	Id	Identifier (0x00..0x3F)
1	Mode	0: No sending of the LIN Frame response 1: Sending of the LIN Frame response
2	reserviert	Reserved
3	MessageCount	0: Send LIN Frame response always 1 ≤ N ≤ 255: Send LIN Frame response N times
4	Dlc	Data length (0..8)
5..7	reserved	Reserved
8..15	Data[0..7]	Data bytes 0..7

Sending LIN frame headers is NOT released or influenced by this command.

4.4.13 0x31 LIN Delete Frame Response

After calling this command, not only the output of the LIN Frame response indicated by Id is stopped. The LIN Frame response itself is deleted from the internal administration.

Another LINFrame response output is only possible after executing the 0x30 LIN Frame Response Definition command.

Command:

Byte	Indication	Description
0	Id	Identifier (0x00..0x3F)
1..3	reserved	Reserved

4.4.14 0x40 LIN Set Bus BaudRate

This command sets the BaudRate parameter, but without having to call the 0x14 LIN Set Interface Properties command completely.

Command:

Byte	Indication	Description
0..3	BaudRate	BaudRate in Baud (generally 19200); In the case of 0 for BaudRate, the automatic baud rate detection is activated

The BaudRate range of values is 700..125000.
That corresponds to a minimum BaudRate of 700 Baud and a maximum BaudRate of 125 KBaud.

4.4.15 0x46 LIN Set Break Detection Threshold

This command sets the BreakDetectionThreshold parameter.

Command:

Byte	Indication	Description
0..3	BreakDetection-Threshold	BreakDetectionThreshold in percent of the bit time (generally 950) BreakDetectionThreshold for GOEPEL electronic Firmware: 9.5 bit-times (in contrast to the LIN Specification with 11 bit-times) for bus monitoring, to be able to detect breaks shorter than 11 bit-times

4.4.16 0x47 LIN Set WakeUp DelimiterTime

This command sets the WakeUpDelimiterTime parameter.

The WakeUpDelimiterTime determines the point in time the Master task (if activated) starts again processing the Schedule table (i.e. sending) after the end of the dominant level of a WakeUp.
According to LIN 2.0 Specification, all Slaves should be ready to receive LIN messages 100 ms after a WakeUp. That means 100 ms after the end of a WakeUp the Maste task should start communication again.

Command:

Byte	Indication	Description
0..3	WakeUpDelimiterTime	WakeUpDelimiterTime in multiples 25 ns, (e.g. 4000000 for a WakeUpDelimiterTime of 100 ms) (0: Default WakeUpDelimiterTime)

In the case of 0 for WakeUpDelimiterTime, the firmware uses the default WakeUpDelimiterTime of four bit times.

4.4.17 0x52 LIN Monitor – Receiving Filter Definition

With this command selected identifiers can be detected with the LIN monitor. In the case of a deactivated filter (that means Mode = 0), all identifiers “go through”.

If the filter is active, all identifiers between StartId and EndId (Mode = 1) or all identifiers indicated by Ids (Mode = 2) are filtered.

In the case only one identifier is to be filtered, use the same value for StartId and EndId (Mode = 1).

Command:

Byte	Indication	Description
0	Mode	0: No Filter 1: Filter a range (for the structure see below) 2: Filter certain identifiers (given by Ids, for the structure see below)
1..3	reserved	Reserved

Parameters for Mode = 1:

Byte	Indication	Description
4	StartId	Start identifier for the range
5	EndId	End identifier for the range
6, 7	reserved	Reserved

Parameters for Mode = 2:

Byte	Indication	Description
4	NumberOfIds	Number of identifiers (N)
5.. (4*N)	Ids	Identifier list (one byte per identifier)

4.4.18 0x54 LIN Monitor – Activation/ Deactivation

This command serves to activate/ deactivate the Buffer reception Mode.
For this Mode, the LIN messages come in in succession into a ring buffer after passing the monitor filter as sent on the bus/ received by the bus.

Command:

Byte	Indication	Description
0	Mode	0: Deactivating monitor 1: Activating Buffer reception (for the structure see below)

In addition, the following parameters are required for Buffer reception (Mode = 1):

Byte	Indication	Description
1	BufferMode	1: Rx (received LIN frame) 2: Tx (sent LIN frame) 3: Rx+Tx (received and sent LIN frames) 4: WakeUp 5: WakeUp + Rx 6: WakeUp + Tx 7: WakeUp + Tx + Rx
2	AutomaticEmpty	0: Empty of the buffer on request 1: Empty of the buffer automatically
3	Type	1: Small monitor entries

For Mode = 0 or 2, the bytes 1..3 are reserved (and should be initialized with 0).

The monitor commands do NOT provide results if the slave task is deactivated!!!

After activating Buffer reception with AutomaticEmpty = 1, the controller independently sends the received LIN frames to the host. Therefore the host has to read out the controller cyclically. In this case, monitor responses have the same structure as the responses of the 0xF2 LIN Monitor – Get Small Buffer Items command.

By activating the monitor with Mode = 1, the timer for creating the StartTime time stamp is set to 0
(see 0xF2 LIN Monitor – Get Small Buffer Items).

To read monitor data in the case of Buffer reception with AutomaticEmpty = 0, the 0xF2 LIN Monitor – Get Small Buffer Items command is used.

4.4.19 0x81 LIN Relays – Setting

All relays (SelectAll = 1) or the relays defined by Relays (SelectAll = 0, NumberOfRelays ≠ 0) are set by this command.

Command:

Byte	Indication	Description
0	NumberOfRelays	Number of the relays to be set (N)
1	SelectAll	0: Setting is valid only for the relays indicated by Relays 1: Setting is valid for all relays
2, 3	reserved	Reserved
4.. (3+N)	Relays	Relay list (in each byte there is the number of the corresponding relay)

Look for the numbers of the required relays in the Hardware section of this User Manual (see Communication Interfaces/ LIN).

4.4.20 0x82 LIN Relays – Resetting

All relays (SelectAll = 1) or the relays defined by Relays (SelectAll = 0, NumberOfRelays ≠ 0) are reset by this command.

Command:

Byte	Indication	Description
0	NumberOfRelays	Number of the relays to be reset (N)
1	SelectAll	0: Resetting is valid only for the relays indicated by Relays 1: Resetting is valid for all relays
2, 3	reserved	Reserved
4.. (3+N)	Relays	Relay list (in each byte there is the number of the corresponding relay)

Look for the numbers of the required relays in the Hardware section of this User Manual (see Communication Interfaces/ LIN).

4.4.21 0x83 LIN Relays – Direct Setting

Set (Relays bit = 1) or reset (Relays bit = 0) the relays according to the Relays bits by this command.

Command:

Byte	Indication	Description
0, 1	Relays	Bit 0: Relay 1 Bit 1: Relay 2 Bit 2: Relay 3 Bit 3: Relay 4 etc. Bit 15: Relay 16
2, 3	reserved	Reserved

Look for the numbers of the required relays in the Hardware section of this User Manual (see Communication Interfaces/ LIN).

4.4.22 0x84 LIN Relays – Get State

Query the state of the relays by this command.
The command does not have any command bytes.

Response:

Byte	Indication	Description
0, 1	Relays	Bit 0: Relay 1 Bit 1: Relay 2 Bit 2: Relay 3 Bit 3: Relay 4 etc. Bit 15: Relay 16
2, 3	reserved	Reserved

A set Relays bit indicates that the corresponding relay is set. Correspondingly, a reset Relays bit indicates a reset relay.

Look for the numbers of the required relays in the Hardware section of this User Manual (see Communication Interfaces/ LIN).

4.4.23 0xA0 LIN Diagnostics – Configuration

Configure the LIN diagnostic protocol for the multisession channel defined by Channel with this command.

The command with Type = 0 is also used to deactivate the complete diagnostic.

Command:

Byte	Indication	Description
0	Channel	Multisession channel (starting with 0)
1	Type	Type of diagnostics: 0: No diagnostics 1: Diagnostics in RAW mode 2: Diagnostics according to LIN 2.0 (For the required structures see next pages)
2	AutomaticEmpty	0: No automatic empty of response buffer 1: Automatic empty of response buffer (Control unit's diagnostic response is sent automatically to the host)
3	TxMethod	Sending or Scheduling mode for MasterRequest IDs and SlaveResponse IDs 0: Diagnostic identifiers (MasterRequest ID and SlaveResponse ID) are contained in the Schedule Table 1: Sending the MasterRequest ID or the SlaveResponse ID in a Sporadic Frame Slot, unless a Spodadic Frame is sent at this moment (regarding Frame Slot see also Figure 4-4) 2: Sending ONCE a MasterRequest ID or SlaveResponse ID at the end of the Schedule Table 3: Sending of ALL MasterRequest IDs and SlaveResponse Ids at the end of the Schedule Table, as long as the Diagnostic Request is sent and the Diagnostic Response is received completely 4: The normal Schedule Table is interrupted for a Diagnostic Request and its belonging Diagnostic Response as long as all corresponding MasterRequest IDs and SlaveResponse IDs are sent

For TxMethod = 2, 3, 4 usually a schedule delay of 192 bit times is used.

For a baudrate of 19200 Baud, this delay is 10 ms.
By the 0xA8 LIN Diagnostics – Change Timing command you can change this value.

The following parameters are valid for diagnostics in RAW Mode:

Byte	Indication	Description
4, 5	reserved	Reserved
6, 7	P2max	P2max timeout in milliseconds (maximum time between end of the requests and beginning of the response, e.g. 200 ms)
8, 9	P3max	P3max timeout in milliseconds (maximum time between end of the requests and beginning of the response during ResponsePending, e.g. 5100 ms)
9, 11	Repetitions	Number of repetitions of the request, if the ECU does not react within the P2max or P3max timeouts, e.g. 2
12	DefaultMasterData. Enabled	0: DefaultMasterRequestFrame deactivated 1: DefaultMasterRequestFrame activated
13..15	DefaultMasterData. reserved	Reserved
16..23	DefaultMasterData. Data	Data of DefaultMasterRequestFrame
24	DefaultSlaveData.- Enabled	0: DefaultSlaveResponseFrame deactivated 1: DefaultSlaveResponseFrame activated
25..27	DefaultSlaveData.- reserved	Reserved
28..35	DefaultSlaveData.- Data	Data of DefaultSlaveResponseFrame
36	TesterPresent.- Enabled	0: TesterPresent deactivated 1: TesterPresent activated
37	TesterPresent.- ResponseRequired	Response for TesterPresent is 0: Not expected 1: Expected
38, 39	TesterPresent.- Cycle	Cycle for TesterPresent in milliseconds, e.g. 1000 ms
40..47	TesterPresent.Data	RAW data of TesterPresent service
48	RxEndCondition	End recognition of diagnostic responses in the case of multi frames 0: Only single frames (no multi frames) 1: Empty slot (no response from the slave) 2: Default slave response frame 3: Same frame
49, 50	reserved	Reserved
51	NumberOf- SpecialResponses	Number of special diagnostic responses (N)
52.. 51+(N*20)	SpecialResponses	Special diagnostic response entries (e.g. 0x21 - busy-RepeatRequest or 0x23 – routineNotComplete) for the structure see next page

An entry in SpecialResponses consists of the following 20 bytes:

Byte	Indication	Description
0..7	Mask[0..7]	Mask bytes 0..7
8..15	Data[0..7]	Data bytes 0..7 (Data is compared with the received data in accordance with the set mask bytes bits)
16	Flags	Bit 0: Repeat request Bit 1: Change timing (P2max to P3max) Bit 2: Default frame Bit 3: Last frame Bit 4: Ignoring the received frame in the case data does not coincides according to Mask and Data Bits 5..7: Reserved
17..19	reserved	Reserved

Any received diagnostic response frame is compared with the SpecialResponses. This happens by a logical AND of the received data with Mask followed by binary comparision of the result with Data.

The following parameters are valid for diagnostics according to LIN2.0:

Byte	Indication	Description
4	NAD	Address of the control unit (**N**ODE **AD**DRESS)
5	reserved	Reserved
6, 7	P2max	P2max timeout in milliseconds (maximum time between end of the requests and beginning of the response, e.g. 200 ms)
8, 9	P3max	P3max timeout in milliseconds (maximum time between endof the requests and beginning of the response during ResponsePending, e.g. 5100 ms)
10, 11	Repetitions	Number of repetitions of the request, if the ECU does not react within the P2max or P3max timeouts, (e.g. 2)
12	TesterPresent.Enabled	0: TesterPresent is deactivated 1: TesterPresent is activated
13	TesterPresent. ResponseRequired	Response for TesterPresent is 0: Not expected 1: Expected
14, 15	TesterPresent.Cycle	Cycle for TesterPresent in milliseconds, e.g. 1000 ms
16..18	TesterPresent.- reserved	Reserved
19	TesterPresent.Length	Data length of TesterPresent service (1..8)
20..27	TesterPresent.Data	Data of TesterPresent service starting with Service ID, generally 0x3E

After selecting a valid diagnostic Type, the corresponding diagnostic task starts when executing the 0xA0 LIN Diagnostics – Configuration command.

If diagnostics is not needed any more, call the 0xA0 LIN Diagnostics – Configuration command once again with Type = 0.

Then the diagnostic task stops, and claimed resources are available again.

The following command sequence results for using the diagnostics:

- Select the Type of diagnostic by the 0xA0 LIN Diagnostics – Configuration command,
- Use diagnostics with its commands,
- Stop diagnostics by the 0xA0 LIN Diagnostics – Configuration command and Type = 0.

Addressing modes:

physical: Communication with an individual ECU (point-to-point-connection, Unicast)

functional: Communication with a group of ECUs (point-to-multipoint-connection, Broadcast)

4.4.24 0xA1 LIN Diagnostics – Start Session

This command starts a LIN diagnostic session for the multisession channel defined by Channel.

Additionally, the diagnostic connection is established.

Command:

Byte	Indication	Description
0	Channel	Multisession channel (starting with 0)
1	Mode	0: Physical addressing 1: Functional addressing **In addition:** No response to the request is necessary if the most significant bit is set (0x80)
2, 3	Length	Request length (0..(PARAM_SIZE – 4)) (for Length = 0 no request is sent)
4.. (3+Length)	Request	Request, consisting of SID (service identifier) and data

4.4.25 0xA2 LIN Diagnostics – Send Request

This command is used to send a LIN diagnostic request for the multisession channel defined by Channel.

Prerequisite is the successful execution of the 0xA1 LIN Diagnostics – Start Session command before, and the diagnostic connection must NOT have been disconnected.

It is necessary to execute this command several times in order to send larger diagnostic requests (e.g. 1100 bytes) caused by the size of the command (limited by MESSAGE_SIZE). In this case the Concatenate and Send parameters must be set accordingly.

Command:

Byte	Indication	Description
0	Channel	Multisession channel (starting with 0)
1	Mode	0: Physical addressing 1: Functional addressing **In addition:** No response to the request is necessary if the most significant bit is set (0x80)
2	Send	0 = No sending (only buffer filling) 1 = Sending
3	Concatenate	0 = Write from buffer beginning 1 = append
4	Segmentation	Segmentation flag for segmentation on diagnostic level 0 = Request not segmented 1 = Request segmented
5	reserved	Reserved
6, 7	Length	Request length (1..(PARAM_SIZE – 8))
8.. (7+Length)	Request	Request, consisting of SID (service identifier) and data

The Segmentation flag refers to the diagnostic protocol.
As a rule it must NOT be set by a diagnostic tester.

4.4.26 0xA3 LIN Diagnostics – Get ResponseBuffer

Query the LIN diagnostic response buffer for the multisession channel defined by Channel with this command.

If a diagnostic response does not fit into a single response, the host has to call this command several times to fetch the remaining responses. The last one of these responses contains the value "0" in the RemainingLength parameter.

In addition, the buffer should be read out as long as the Segmentation bit, the Busy bit or the BufferNotEmpty bit of Flags are set.

Command:

Byte	Indication	Description
0	Channel	Multisession channel (starting with 0)
1..3	reserved	Reserved

Response:

Byte	Indication	Description
0	Channel	Multisession-Kanal (beginnend mit 0)
1	LastErrorCode	Fehlercode (0 = kein Fehler)
2	Flags	Bit 0 = 0: No segmentation on diagnostic level Bit 0 = 1: Segmentation (segmentation on diagnostic level) Bit 1 = 0: Idle Bit 1 = 1: Busy (a request has not been responded yet or successfully sent) Bit 2 = 0: Invalid (this buffer entry is invalid Bit 2 = 1: Valid (this buffer entry is valid) Bit 3 = 0: The diagnostic response buffer is empty Bit 3 = 1: BufferNotEmpty (the diagnostic response buffer is not empty yet) Bits 4..7: Reserved
3	State	State of diagnostics: 0: Not initialized 1: No connection 2: Connection is being established 3: Connection was established 4: Connection is released
4, 5	Length	Number of response bytes (0..(PARAM_SIZE – 8))
6, 7	RemainingLength	Number of remaining response bytes
8.. (7+Length)	Response	Response, consisting of SID (service identifier) and data

4.4.27 0xA4 LIN Diagnostics – Stop Session

This command stops a running LIN diagnostic session for the multisession channel defined by Channel.

Additionally, the diagnostic connection is released.

To stop the complete diagnostic, call the 0xA0 LIN Diagnostics – Configuration command again with Type = 0.

Command:

Byte	Indication	Description
0	Channel	Multisession channel (starting with 0)
1	Mode	0: Physical addressing 1: Functional addressing **In addition:** No response to the request is necessary if the most significant bit is set (0x80)
2, 3	Length	Request length (0..(PARAM_SIZE – 4)) (for Length = 0 no request is sent)
4.. (3+Length)	Request	Request, consisting of SID (service identifier) and data

4.4.28 0xA5 LIN Diagnostics – Get State

Query the LIN diagnostic state for the multisession channel defined by Channel with this command.
Additionally, the firmware internal LastErrorCode can be reset.

The value of the LastErrorCode in the Response corresponds to the value of the firmware internal LastErrorCode before its resetting.

Generally the firmware internal LastErrorCode is reset automatically without calling this 0xA5 LIN Diagnostics – Get State command by starting a diagnostic session by 0xA1 LIN Diagnostics – Start Session and stop of a diagnostic session with 0xA4 LIN Diagnostics – Stop Session and Length ≠ 0.

Command:

Byte	Indication	Description
0	Channel	Multisession channel (starting with 0)
1	ResetLastError	0 = No reset of the LastErrorCode 1 = Reset of the LastErrorCode
2, 3	reserved	Reserved

Response:

Byte	Indication	Description
0	Channel	Multisession channel (starting with 0)
1	LastErrorCode	Error code (0 = no error)
2	DiagType	Type of diagnostics: 0: No diagnostics 1: Diagnostics in RAW Mode 2: Diagnostics according to LIN 2.0
3	State	State of diagnostics: 0: Not initialized 1: No connection 2: Connection is being established 3: Connection was established 4: Connection is released
4	Flags	Bit 0 = 0: Idle Bit 0 = 1: Busy (a request has not been responded yet or successfully sent) Bit 1 = 0: The diagnostic response buffer is empty Bit 1 = 1: RxBufferNotEmpty (the diagnostic response buffer is not empty yet) Bits 2..7: Reserved
5..7	reserved	Reserved

4.4.29 0xA8 LIN Diagnostics – Change Timing

Use this command to modify certain diagnostic timing parameters for the multisession channel defined by Channel.

Command:

Byte	Indication	Description
0	Channel	Multisession channel (starting with 0)
1	Mode	0: Modify Schedule Delay (relevant for TxMethod = 2, 3, 4 of the 0xA0 LIN Diagnostics – Configuration command) 1: Modify Sending Timeout (usually the Sending Timeout is 1000 ms)
2, 3	reserved	Reserved

In addition, the following parameters are necessary for Mode = 0:

Byte	Indication	Description
4..7	MasterRequest	Schedule delay for a master request in multiples of 25 ns
8..11	SlaveResponse	Schedule delay for a slave response in multiples of 25 ns

In addition, the following parameters are necessary for Mode = 1:

Byte	Indication	Description
4, 5	TxTimeout	Sending timeout in milliseconds
6, 7	reserved	Reserved

4.4.30 0xA9 LIN Diagnostics – Protocol Control

This command is used to control the LIN Diagnostic Protocol. The command is subdivided into several sub-commands distinguishable by the Mode parameter.

The command and response structure for all sub-commands is the same for all bytes up to Byte 3, while varieties occur starting with Byte 4 (if more than four bytes exist).

Command and Response:

Byte	Indication	Description
0	Channel	Multisession channel (starting with 0)
1	Mode	0: Changing the behavior
2, 3	reserved	Reserved

The following Command parameters are only valid for Mode = 0:

Byte	Indication	Description
4..7	Flags	In normal cases, ALL bits are 0 Bit 0: Disable21Handling (the negative response BusyRepeatRequest is not handled) Bit 1: Disable23Handling (the negative response RoutineNotComplete is not handled) Bit 2: Disable78Handling (the negative response RequestCorrectlyReceived_ResponsePending is not handled) Bit 3: Treat21As78Handling (the negative response BusyRepeatRequest is handled as negative response RequestCorrectlyReceived_ResponsePending) Bits 4..31: Reserved
8..11	reserved	Reserved

The following Response parameters are only valid for Mode = 0:

Byte	Indication	Description
4..7	reserved	Reserved

4.4.31 0xF2 LIN Monitor – Get Small Buffer Items

This command is to query small LIN monitor buffer entries.
The command does not have any command bytes.

Response:

Byte	Indication	Description
0..3	NumberOfItems	Number of monitor buffer entries
4..	Items	Monitor buffer entries (for the structure see below)

A small LIN monitor buffer item without additional time stamps consists of the following 20 Bytes:

Byte	Indication	Description
0	Flags	Bit 0: Identifier parity error Bit 1: Checksum error Bit 2: Inconsistent SyncByte Bit 3: Bit error Bit 4: Event (see IdCode) Bit 5: WakeUp Bit 6: Sent LIN Frame response (TX) Bit 7: Buffer overflow
1	Length	Data length including checksum (0..9)
2	IdCode	Generally the Identifier byte (consisting of identifier + parity bits); BUT, If Bit 4 of Flags is set, IdCode specifies the Event: IdCode = 0 - rising flank at the trigger input, IdCode = 1 - falling flank at the trigger input
3..11	Data	Data bytes and checksum
12..15	StartTime	Start time stamp as multiples of 400 ns
16..19	BitTimeX8	Bit time measured over eight bit times as multiples of 25 ns

Data contains the data bytes and the checksum following immediately to the last data byte.
Length = 3 indicates two data bytes (Data[0] and Data[1]) and one checksum byte (Data[2]).

4.5 K-Line Commands

The K-Line commands for your GOEPEL hardware are described in this chapter.

For general information valid for all firmware commands refer to the General Firmware Notes section in this User Manual.

Optional Functionalities

For each K-Line interface there are at most the following Optional Functionalities:

- Diagnostics KWP2000
- Diagnostics KWP1281
- Diagnostics ISO-9141-Ford

It is possible to select several Functionalities

After a power-on or software reset, available Optional Functionalities have to be enabled by 0x03 Enable Functionalities.

Then the following firmware command should be executed:

- 0x12 KLine Init Interface

Initial state:

After a power-on or software reset, all K-Line interfaces are in an inactive state (HIGH level).

The term Initialization used in this K-Line command description means the following:
The tester sends an initialization pattern to establish communication.

The protocol driver is based on the following documents:

KWP2000

ISO 14230-2:1999 Keyword Protocol 2000 - Part 2: Data link layer

ISO 14230-3:1999 Keyword Protocol 2000 - Part 3: Application layer

KWP1281

Robert Bosch GmbH: Funktionsbeschreibung der Diagnose VW/Audi (Y 265 K15 383 Ausgabe 04)

ISO-9141-Ford

Ford Automotive Operations: Global Diagnostic Specification: Part One (DS-3L5T-1A294-AA)

Protocol-specific designations and abbreviations used in this description are taken from these documents and marked by ***bold*** and ***Italic*** characters.

If parameters are marked as “reserved”, the contents of this field will be ignored. Nevertheless, the parameter must be transferred (for compatibility among other reasons). In practical operation, these values are to be initialized by 0.

Generally, the principle of the permanent change between request and response applies for the communication on the K-Line. That means that each request of the tester (here, K-Line protocol driver) causes a response of the control unit. If these responses are not used for controlling the protocol, they will be passed on to the host interface.

Protocol-specific exceptions regarding this request-response-change are intercepted by the protocol driver. That means that the principle of the “request-response-game” always applies for running K-Line specific communication via the protocol driver!

Error Behavior:

If critical errors, which stop the communication for example, are detected during processing commands, the protocol driver will be set into a “clean” initial status.

In this case, an error number is set internally, which can be inquired for example by the 0xA5 KLine Diagnostics – Get State command.

4.5.1 0x12 KLine Init Interface

This command resets the selected K-Line interface without software reset to the initial state. Additionally, further configuration possibilities are offered.
Interface selection is made by the TargetAddress and TargetPort parameters in the header of the command.

The command bytes are optional. If there are no command bytes, the firmware runs with 0 for the optional command bytes.

Command:

Byte	Indication	Description
0..2	reserved	Reserved
3	BlinkMode	0: Flickering of the LEDs deactivated (default) 1: Flickering of the LEDs activated

4.5.2 0xA0 KLine Diagnostics – Configuration

Configure the K-Line diagnostic protocol for the multisession channel defined by Channel with this command.

The command with Type = 0 is also used to deactivate the complete diagnostics.

This firmware command can only be used in the case the diagnostic protocol to be configured was enabled before by 0x03 Enable Functionalities.

All settings required for a diagnostic structure and flow are preset by this command. The settings made remain active till new settings will be made explicitly.

This command has a total length of 84 bytes. The data structure of this command is identical for all protocols. The interpretation of the individual fields varies according to the selected diagnostic protocol and to the type of initialization.

Generally Valid Parameters:

Byte	Indication	Description
0	Channel	Multisession channel (starting with 0)
1	Type	Type of diagnostics: 0: No diagnostics 1: Diagnostics KWP2000 2: Diagnostics KWP1281 3: Diagnostics ISO-9141-Ford (for the requires structures see next pages)
2, 3	reserved	Reserved
4, 5	Flags	Automatic sending of diagnostic responses to the host Bit 0 = 0: deactivated Bit 0 = 1: activated Bits 1..4: reserved Verifying the check sum (KWP2000 and ISO-9194-Ford) Bit 5 = 0: deactivated Bit 5 = 1: activated Bits 6..15: reserved
6, 7	Reserved	Reserved

If verifying the check sum is deactivated, the value of the check sum field will be ignored; otherwise verifying will be carried out.

Invalid check sums lead to “invalid check sum” errors then.

Parameterization of Keyword Protocol 2000 (KWP2000):

Byte	Indication	Description
8, 9	SourceAddress	Address of the tester to be used during the diagnostics, e.g. 0xF1
10, 11	TargetAddress	Address of the control unit to be used during the diagnostics
12, 13	P1min	minimum interbyte time for responses of the ECU, e.g. 0 ms
14, 15	P1max	maximum interbyte time for responses of the ECU, e.g. 20 ms
16, 17	P2min	minimum time gap between the request of the tester and the response of the ECU, or minimum time gap between two responses of the ECU, e.g. 25 ms
18, 19	P2max	maximum time gap between the request of the tester and the response of the ECU, or maximum time gap between two responses of the ECU, e.g. 50 ms
20, 21	P3min	minimum time gap between the response of the ECU and a new request of the tester, e.g. 55 ms
22, 23	P3max	maximum time gap between the response of the ECU and a new request of the tester, e.g. 2000 ms
24, 25	P4min	minimum interbyte time for requests of the tester, e.g. 5 ms
26, 27	P4max	maximum interbyte time for requests of the tester, e.g. 20 ms
28, 29	TesterPresent.-SourceAddress	Address of the tester to be used during the ***Tester Present*** service, e.g. same as SourceAddress
30, 31	TesterPresent.-TargetAddress	Address of the control unit to be used during the ***Tester Present*** service (e.g. same as TargetAddress)
32	TesterPresent.-UseResponseRequired-Parameter	The ***Tester Present Response Required*** parameter is 0: not used 1: used
33	TesterPresent.ResponseRequiredParameter	Value of the ***Tester Present Response Required*** parameter (if used), otherwise 0
34, 35	reserved	Reserved
36, 37	InitType	Type of initialization: 0: 5 Baud initialization 1: Fast initialization
38, 39	reserved	Reserved

Parameterization KWP2000 for 5 Baud Initialization:

Byte	Indication	Description
40, 41	reserved	Reserved
42, 43	TargetAddress	5 Baud address of the control unit
44, 45	W1min	minimum time gap between the end of the address byte and the start of the synchronization pattern, e.g. 60 ms
46, 47	W1max	maximum time gap between the end of the address byte and the start of the synchronization pattern, e.g. 300 ms
48, 49	W2min	minimum time gap between the end of the synchronization pattern and the start of ***Keybyte 1***, e.g. 5 ms
50, 51	W2max	maximum time gap between the end of the synchronization pattern and the start of ***Keybyte 1***, e.g. 20 ms
52, 53	W3min	minimum time gap between ***Keybyte 1*** and ***Keybyte 2***, e.g. 0 ms
54, 55	W3max	maximum time gap between ***Keybyte 1*** and ***Keybyte 2***, e.g. 20 ms
56, 57	W4min	minimum time gap between ***Keybyte 2*** of the ECU and its inversion by the tester as well as minimum time gap between the inverted ***Keybyte 2*** of the tester and the inverted address byte of the ECU, e.g. 25 ms
58, 59	W4max	maximum time gap between ***Keybyte 2*** of the ECU and its inversion by the tester as well as maximum time gap between the inverted ***Keybyte 2*** of the tester and the inverted address byte of the ECU, e.g. 50 ms
60, 61	W5min	minimum time gap before the tester starts to send the address byte, e.g. 300 ms
62, 63	W5max	maximum time gap before the tester starts to send the address byte, e.g. 300 ms
64..71	reserved	Reserved
72, 73	Parity	Parity for sending the address byte: 0: even 1: odd
74, 75	reserved	Reserved

Parameterization KWP2000 for Fast Initialization

Byte	Indication	Description
40, 41	SourceAddress	Address of the tester to be used during the initialization, e.g. 0xF1
42, 43	TargetAddress	Address of the control unit to be used during the initialization
44, 45	W5min	minimum time gap before the tester starts to send the address byte, e.g. 300 ms
46, 47	W5max	maximum time gap before the tester starts to send the address byte, e.g. 300 ms
48, 49	TWuPmin	minimum time period for the Wake-up pattern, e.g. 50 ms
50, 51	TWuPmax	maximum time period for the Wake-up pattern, e.g. 50 ms
52, 53	TIniLmin	minimum time period for the low phase of the Wake-up pattern, e.g. 25 ms
54, 55	TIniLmax	maximum time period for the low phase of the Wake-up pattern, e.g. 25 ms
The following parameters are valid up to the end of the Initialization, i.e. via these parameters the time response of the protocol can be separately set during the Initialization.		
56, 57	P1min	minimum interbyte time for responses of the ECU, e.g. 0 ms
58, 59	P1max	maximum interbyte time for responses of the ECU, e.g. 20 ms
60, 61	P2min	minimum time gap between the request of the tester and the response of the ECU, e.g. 25 ms
62, 63	P2max	maximum time gap between the request of the tester and the response of the ECU, e.g. 50 ms
64, 65	P3min	minimum time gap between the response of the ECU and a new request of the tester, e.g. 55 ms
66, 67	P3max	maximum time gap between the response of the ECU and a new request of the tester, e.g. 2000 ms
68, 69	P4min	minimum interbyte time for requests of the tester, e.g. 5 ms
70, 71	P4max	maximum interbyte gap time for requests of the tester, e.g. 20 ms
72, 73	BaudRate	BaudRate (usually 10400 Hz)
74, 75	reserved	Reserved

Parameterization KWP2000 (Continuation)

76, 77	BusyRepeatRequest-Max	Maximum number of ***Busy Repeat Request (0x21)*** responses to a request If the maximum number is exceeded, the response with the error code will be given to the host – the request will not be repeated. 0x0000..0xFFFE: number 0xFFFF: unlimited
78, 79	RoutineNotComplete-Max	Maximum number of ***Routine Not Complete (0x23)*** responses to a request If the maximum number is exceeded, the response with the error code will be given to the host – the request will not be repeated. 0x0000..0xFFFE: number 0xFFFF: unlimited
80, 81	RequestCorrectly-ReceivedResponse-PendingMax	Maximum number of ***Request Correctly Received Response Pending (0x78)*** responses to a request If the maximum number is exceeded, the communication will be aborted and a NO_RESPONSE error will be generated. 0x0000..0xFFFE: number 0xFFFF: unlimited
82, 83	reserved	Reserved

Parameterization of Keyword Protocol 1281 (KWP1281):

Byte	Indication	Description
8..11	reserved	Reserved
12, 13	t7min	minimum time gap between the bytes within one block, e.g. 1 ms
14, 15	t7max	maximum time gap between the bytes within one block, e.g. 55 ms
16, 17	t8min	minimum time for the repeated reception of the first byte of a block (if the slave has not received the last byte of a block), e.g. 1 ms
18, 19	t8max	maximum time for the repeated reception of the first byte of a block (if the slave has not received the last byte of a block),e.g. 200 ms
20, 21	t9min	minimum time gap between the end of a block and the start of the next block, e.g. 1 ms
22, 23	t9max	maximum time gap between the end of a block and the start of the next block, e.g. 200 ms
24..35	reserved	Reserved
36, 37	InitType	Type of initialization 0: 5 Baud initialization
38..41	reserved	Reserved
42, 43	TargetAddress	5 Baud address of the control unit
44, 45	t0min	minimum idle line before the start of initialization, e.g. 60 ms
46, 47	t0max	maximum idle line before the start of initialization, e.g. 300 ms
48, 49	t1min	minimum time gap between the correct initialization and the start of the synchronous byte, e.g. 80 ms
50, 51	t1max	maximum time gap between the correct initialization and the start of the synchronous byte, e.g. 210 ms
52, 53	t2min	minimum time gap between the synchronous byte and ***Keybyte 1***, e.g. 5 ms
54, 55	t2max	maximum time gap between the synchronous byte and ***Keybyte 1***, e.g. 20 ms
56, 57	t3min	minimum time gap between ***Keybyte 1*** and ***Keybyte 2***, e.g. 1 ms
58, 59	t3max	maximum time gap between ***Keybyte 1*** and ***Keybyte 2***, e.g. 20 ms
60, 61	t4min	minimum time gap between ***Keybyte 2*** and the complement of ***Keybyte 2***, e.g. 25 ms
62, 63	t4max	maximum time gap between ***Keybyte 2*** and he complement of ***Keybyte 2***, e.g. 50 ms
64, 65	t5min	minimum time gap between the complement of ***Keybyte 2*** and the repeated output of the synchronous byte (if the complement of ***Keybyte 2*** has not been received correctly by the control unit), e.g. 240 ms
66, 67	t5max	maximum time gap between the complement of ***Keybyte 2*** and the repeated output of the synchronous byte (if the complement of ***Keybyte 2*** has not been received correctly by the control unit), e.g. 1000 ms
68, 69	t6min	minimum time gap between the complement of ***Keybyte 2*** and the start of the ECU identification, e.g. 25 ms
70, 71	t6max	maximum time gap between complement keybyte 2 and the start of the ECU identification, e.g. 50 ms

Byte	Indication	Description
72, 73	Parity	Parity when sending the address byte: 0: even 1: odd
74, 75	reserved	Reserved
76, 77	MasterMaxBlockRetry	maximum number of new attempts if errors occur during the transmission of a block, e.g. 5 Input of the maximum number of repeated attempts to send a block, if errors occur during the transmission of a block (protocol driver is master). (Error e.g. no or faulty echo from the slave, NO_ACK-1 from the Slave) 0x0000..0xFFFE: Number 0xFFFF: unlimited
78, 79	SlaveMaxBlockRetry	maximum number of new attempts if errors occur during the reception of a block, e.g. 5 Input of the maximum number of repeated attempts to receive a block, if errors occur during the reception of a block (protocol driver is slave). (Error e.g. timeout during the reception of the next byte within one block of the master) 0x0000..0xFFFE: Number 0xFFFF: unlimited
80..83	reserved	Reserved

Parameterizing of ISO-9141-Ford:

Byte	Indication	Description
8, 9	SourceAddress	Address of the tester to be used during the diagnostics, e.g. 0xF1
10, 11	TargetAddress	Address of the control unit to be used during the diagnostics
12, 13	ReceiveInterByte-GapMin	***Tester Reception: Interbyte Gap Time min*** minimum interbyte time for responses of the ECU, e.g. 0 ms
14, 15	ReceiveInterByte-GapMax	***Tester Reception: Interbyte Gap Time max*** maximum interbyte time for responses of the ECU, e.g. 22 ms
16, 17	ResponseInterMsg-GapMin	***ECU Response Following A Tester Request & ECU Response Following Another ECU Message In A Sequence: Intermessage Gap Time min*** minimum time gap between the request of the tester and the response of the ECU, or minimum time gap between two responses of the ECU, e.g. 0 ms
18, 19	ResponseInterMsg-GapMax	***ECU Response Following A Tester Request & ECU Response Following Another ECU Message In A Sequence: Intermessage Gap Time max*** maximum time gap between the request of the tester and the response of the ECU, or maximum time gap between two responses of the ECU, e.g. 50 ms
20, 21	RequestInterMsg-GapMin	***Tester Request Following An ECU Response: Intermessage Gap Time min*** minimum time gap between the response of the ECU and a new request of the tester, e.g. 55 ms
22, 23	RequestInterMsg-GapMax	***Tester Request Following An ECU Response: Intermessage Gap Time max*** maximum time gap between the response of the ECU and a new request of the tester, e.g. 2000 ms
24, 25	TransmitInterByte-GapMin	***Tester Transmissions: Interbyte Gap Time min*** minimum interbyte time for requests of the tester, e.g. 6 ms
26, 27	TransmitInterByte-GapMax	***Tester Transmissions: Interbyte Gap Time max*** maximum interbyte time for requests of the tester, e.g. 6 ms
28..35	reserved	Reserved
36, 37	InitType	Type of initialization: 2: Specific initialization not required
38..75	reserved	Reserved
76, 77	BusyRepeatRequest-Max	maximum number of ***Busy Repeat Request (0x21)*** responses to a request If the maximum number is exceeded, the response with the error code will be given to the host – the request will not be repeated 0x0000..0xFFFE: Number 0xFFFF: unlimited
78..83	reserved	Reserved

Notes to the Parameterization of the Tester Present Service:

The so called ***Tester Present*** service (called “interchange of acknowledge blocks” for KWP1281) is used to maintain the communication. That means that if requests of the host are not received by the protocol driver (tester) during a defined period of time, this one must prevent the communication from being cut (ECU changes into the timeout) by transmitting specific messages.

The maximum time gap between the response of the ECU and a new request of the tester is the decisive parameter here
(KWP2000: P3max,
KWP1281: t9max,
ISO-9141-Ford: RequestInterMsgGapMax).

The relevant message is always generated by the protocol driver shortly before the time gap preset by the host will end.

Addressing modes:
physical: Communication with an individual ECU (point-to-point-connection, Unicast)
functional: Communication with a group of ECUs (point-to-multipoint-connection, Broadcast)

4.5.3 0xA1 KLine Diagnostics – Start Session

This command starts a K-Line diagnostic session for the multisession channel defined by Channel. Additionally, the diagnostic connection is established.

Before this command can be executed, the 0xA0 KLine Diagnostics – Configuration **command must be carried out.** The command is the precondition for sending a request (0xA2 KLine Diagnostics – Send Request).

The diagnostics is maintained till it will be explicitly stopped by means of 0xA4 KLine Diagnostics – Stop Session.

After the successful diagnostic setup (initialization) the ***Tester Present*** service (***ACK block*** interchange for KWP1281) will become active as soon as the idle timeout is exceeded on the K-Line (that means that no request was received from the tester after a defined time gap (before the end of the maximum interframe or interblock time).

Command:

Byte	Indication	Description
0	Channel	Multisession channel (starting with 0)
1	Mode	0: Physical addressing 1: Functional addressing **In addition:** No response to the request is necessary if the most significant bit is set (0x80)
2, 3	Length	Request length (At the present time, only Length = 0 is supported, the corresponding start service is transmitted according to the diagnostic type)
4.. (3+Length)	Request	Request, consisting of SID (service identifier) and data

The 0xA1 KLine Diagnostics –Start Session command for starting the diagnostics (or for opening the communication) is identical for all K-Line protocols at first sight on the part of the host. Within the K-Line driver, however, specific actions matched to the individually active protocol are released. All the protocols react differently within the diverse opening variants.

Generally one can say: The K-Line protocol driver always delivers a response to the 0xA1 KLine Diagnostics – Start Session command (either automatically or via the 0xA3 KLine Diagnostics – Get Response Buffer inquiry, depending on the response mode set)!

But depending on the protocol, the meaning of the response data varies. The receiver (host) is responsible for the correct interpretation.

The following table shall make clear the processes within the protocol driver running as the response to a 0xA1 KLine Diagnostics –Start Session command.

Attention is to be paid to the fact that a real separation between the diagnostic and transport protocols does not exist on the K-Line. To simplify it one can say that “Start communication” on the K-Line is to be considered identical to “Start diagnostics”.

KWP2000 (Fast initialization)	
Description	Response of K-Line driver
Here, ***StartCommunication*** is a normal” request (KWP2000 frame with three bytes header, ***SID*** = 0x81 – ***StartCommunication***”) which will be transmitted at a certain baud rate on the K-Line after a defined “idle” time and a specific ***Wake up Pattern (WuP)***. In case of success, the control unit responds with a response frame (form of the header depending on the ability of the control unit, ***SID*** = 0xC1, two ***keybytes*** in the data division). Then, the communication is considered to be opened, that means that the ***Tester Present*** service becomes active if there are not any requests.	Data division of the response frame with the two ***keybytes*** received by the ECU. ATTENTION: By ***keybyte 1*** the control unit gives information on its abilities.

KWP2000 (slow initialization – 5 Baud)	
Description	Response of K-Line driver
After an idle time (bus rest) the particular “initialization address” is transmitted at 5 Bauds. In case of success, the control unit puts out a pattern which allows the tester (K-Line driver) to synchronize itself to the baud rate of the control unit. Afterwards, the ECU sends two ***keybytes***. The tester on its part acknowledges the reception of the ***keybytes*** by returning ***keybyte 2*** inverted bit by bit to the ECU. Finally, the ECU returns the “initialization address” inverted bit by bit. Afterwards, communication is considered to be opened, that means the interchange of ***Tester Present*** blocks if there are not any requests. Attention: Now possibly varying address of the ECU.	The K-Line driver generates a response identical to the “KWP2000 with quick stimulation” from the ***Keybytes*** received within the stimulation. Thus, there are not any differences (on the part of the host) to the fast initialization.

KWP1281 (slow initialization – 5 Baud)	
Description	Response of K-Line driver
Similar to KWP2000 slow stimulation, but instead of sending the bit-by-bit inverted address, the control unit automatically starts putting out its identification string according to the regulations following KWP1281 protocol after having received the ***Keybyte 2*** complement. This identification is possibly distributed among several blocks. Afterwards, the communication is considered to be opened, that means the interchange of ***Acknowledge*** blocks if there are not any requests.	The K-Line driver transmits the received ECU-ID as a response to the host interface. ATTENTION: For KWP1281, the ***keybytes*** do not give any information on the ECU (always identical).

ISO-9141-Ford	
Description	Response of K-Line driver
The tester (K-Line driver) generates and transmits a “normal” request (***Mode*** = 0x10 – ***Diagnostic Mode Entry***) at 10400 Bauds and with the preset address parameters on the K-Line. In case of success, the control unit responds with a ***General Response*** block (***Mode*** = 0x7F, ***Response Code*** = 0x00). Then, the communication is considered to be opened, that means that the ***Tester Present*** service becomes active, if there are not any requests.	Data division of the ***General Response*** block transmitted by the ECU.

4.5.4 0xA2 KLine Diagnostics – Send Request

This command is used to send a K-Line diagnostic request for the multisession channel defined by Channel.

Prerequisite is the successful execution of the 0xA1 KLine Diagnostics – Start Session command before, and the diagnostic connection must NOT have been disconnected.

Depending on the setting (bit 0 in the Flags parameter for the 0xA0 KLine Diagnostics – Configuration command), the response to this request is either returned automatically to the host or has to be demanded by 0xA3 KLine Diagnostics – Get Response Buffer.

In the request, only the actual used data of the telegram to be generated by the protocol driver is transmitted (for KWP2000, ISO-9151-Ford: no header, no checksum – only ***ServiceID*** (or ***MODE*** byte) and data, for KWP1281: no block length, no block counter, no ETX block end byte – only block title and data).

Command:

Byte	Indication	Description
0	Channel	Multisession channel (starting with 0)
1	Mode	0: Physical addressing 1: Functional addressing **In addition:** No response to the request is necessary if the most significant bit is set (0x80)
2	Send	0 = No sending (only buffer filling) 1 = Sending
3	Concatenate	0 = Write from buffer beginning 1 = append
4	Segmentation	Segmentation flag für segmentation on diagnostic level 0 = Request not segmented 1 = Request segmented
5	reserved	Reserved
6, 7	Length	Request length (1..(PARAM_SIZE – 8))
8.. (7+Length)	Request	Request, consisting of SID (service identifier) and data

The Segmentation flag refers to the diagnostic protocol.
As a rule it must not be set by a diagnostic tester.

Within ONE 0xA2 Kline Diagnostics – Send Request command, a maximum number of PARAM_SIZE – 8 Request bytes can be transmitted.

It is necessary to execute this command several times in order to send larger diagnostic requests (e.g. 1100 bytes) caused by the size of the command (limited by MESSAGE_SIZE). In this case the Concatenate and Send parameters must be set accordingly.

Example of the Segmentation of Host Requests

The following example shall demonstrate the segmentation of commands sent to the driver.
It is assumed that a KWP2000 diagnostics was successfully opened.
"**0x1A, 0x9B**" – "Read control units" identification (***ReadECUIdentification Service***)" is to be transmitted.

Variant 1) Monolithic command:

Point in time t1: *Request (0xA2)* telegram from the host to the protocol driver

Command header with *OpCode* = 0xA2	**(u8)** 0x00 *Channel* = 0	**(u8)** 0x00 *Mode* = 0	**(u8)** 0x01 *Send* = 1	**(u8)** 0x00 *Concatenate* = 0	**(u8)** 0x00 *Segmentation* = 0	**(u8)** 0x00	**(u16)** 0x0002 *Length* = 2	**(u8)** 0x1A	**(u8)** 0x9B

Point in time t2 (t2 = t1 + x): KWP2000 telegram is generated and transmitted by the protocol driver

KWP2000 Header	**(u8)** 0x1A	**(u8)** 0x9B	**(u8)** KWP2000 Prüfsumme

Variant 2) Segmented command:

Point in time t1: *Request (0xA2)* telegram from the host to the protocol driver

Command header with *OpCode* = 0xA2	**(u8)** 0x00 *Channel* = 0	**(u8)** 0x00 *Mode* = 0	**(u8)** 0x00 *Send* = 0	**(u8)** 0x00 *Concatenate* = 0	**(u8)** 0x00 *Segmentation* = 0	**(u8)** 0x00	**(u16)** 0x0001 *Length* = 1	**(u8)** 0x1A

Point in time t2 (t2 = t1 + x): second *Request (0xA2)* telegram from the host to the protocol driver

Commands-Header mit *OpCode* = 0xA2	**(u8)** 0x00 *Channel* = 0	**(u8)** 0x00 *Mode* = 0	**(u8)** 0x01 *Send* = 1	**(u8)** 0x01 *Concatenate* = 1	**(u8)** 0x00 *Segmentation* = 0	**(u8)** 0x00	**(u16)** 0x0001 *Length* = 1	**(u8)** 0x9B

Point in time t3 (t3 = t2 + y): KWP2000 telegram is generated and transmitted by the protocol driver

KWP2000 Header	**(u8)** 0x1A	**(u8)** 0x9B	**(u8)** KW2000 checksum

The Segmentation flag is not set in the example as it refers to a diagnostic protocol.

The data interchange via the K-Line and the communication between the host and the protocol driver are based on the request-response-principle. That means each request causes **one** (!) response.

Within the different protocols exceptions to this principle do exist sometimes. These deviations are intercepted by the protocol driver by means of different mechanisms. In case of success, a request of the host does always result in a response given by the driver.

If a response of the protocol driver does not arrive at the host within the preset time gap, the state of the driver can (should) be checked via the 0xA5 KLine Diagnostics – Get State command.

In the following, this situation is briefly demonstrated with the help of a specific example:
An ECU according to KWP1281 gives a segmented identification string response to the ***Read control unit identification*** (***BT*** = 0x00) request sent by the tester. That means, the response of the ECU (the identification string) is distributed among several response blocks. According to KWP1281, each of these response blocks must be acknowledged by an ***Acknowledge*** block, if it was received by the tester (driver).

The driver identifies the end of the control unit response, if it receives an ***Acknowledge*** block from the ECU as a direct response to such an ***Acknowledge*** block sent on its part. On the basis of the received segments the protocol driver generates the response for the host now. In this case, the ***Acknowledge*** block is only used to control the sequence of the protocol. It is not included in the response of the driver sent to the host!

The ***Delete error memory*** (***BT*** = 0x05) command is used as an example to prove the opposite. The ECU directly reports the successful execution of this command by an ***Acknowledge*** block (no further responses!) Now, this response is not used to control the sequence of the protocol but to acknowledge the execution of a command. In this case, the ***Acknowledge*** block is passed on to the host as a response by the driver.

4.5.5 0xA3 KLine Diagnostics – Get Response Buffer

Query the K-Line diagnostic response buffer for the multisession channel defined by Channel with this command

By means of the command, the response to a previous diagnostic request (0xA2 KLine Diagnostics – Send Request) is collected.

The deactivation of the automatic transmission of diagnostic responses (bit 0 in the Flags parameter for the 0xA0 KLine Diagnostics – Configuration command is set to 0) **is the precondition for executing this command**.

The command is also used to query both the response to 0xA1 KLine Diagnostics – Start Session as well as the response to 0xA4 KLine Diagnostics – Stop Session (not for KWP1281!).

Command:

Byte	Indication	Description
0	Channel	Multisession channel (starting with 0)
1..3	reserved	Reserved

The response returned by the protocol driver only contains the user data division of the corresponding response (with *ServiceID* or *BlockTitle*, without header, check fields etc.) within the user data field. The response may possibly be segmented (i.e. distributed among several command telegrams).

Response:

Byte	Indication	Description
0	Channel	Multisession channel (starting with 0)
1	LastErrorCode	Error code (0 = no error)
2	Flags	Bit 0 = 0: No segmentation on diagnostic level Bit 0 = 1: Segmentation (segmentation on diagnostic level) Bit 1 = 0: Idle Bit 1 = 1: Busy (a request has not been responded yet or successfully sent) Bit 2 = 0: Invalid (this buffer item is invalid) Bit 2 = 1: Valid (this buffer item is valid) Bit 3 = 0: The diagnostic response buffer is empty Bit 3 = 1: BufferNotEmpty (the diagnostic response buffer is not empty yet) Bits 4..7: Reserved
3	State	State of diagnostics: 0: Not initialized 1: No connection 2: Connection is being established 3: Connection was established 4: Connection is released
4, 5	Length	Number of response bytes (0..(PARAM_SIZE – 8))
6, 7	RemainingLength	Number of remaining response bytes
8.. (7+Length)	Response	Response, consisting of SID (service identifier) and data

If a diagnostic response does not fit into a single response, the host has to call this command several times to fetch the remaining responses. The last of these responses contains the value "0" in the RemainingLength parameter.

In addition, the buffer should be read out as long as the Segmentation bit, the Busy bit or the BufferNotEmpty bit of Flags are set.

Interpretation of the Response:

The contents of the responses which can be got by this command after
0xA1 KLine Diagnostics – Start Session,
0xA2 KLine Diagnostics – Send Request and
0xA4 KLine Diagnostic – Stop Session
can have different meanings depending on the protocol used.

The K-Line protocol driver is NOT responsible for the interpretation of these responses!

4.5.6 0xA4 KLine Diagnostics – Stop Session

This command stops a running K-Line diagnostic session for the multisession channel defined by Channel. Additionally, the diagnostic connection is released.

Further 0xA2 KLine Diagnostics – Send Request commands are not possible till to the next 0xA1 KLine Diagnostics – Start Session command.

The settings made by 0xA0 KLine Diagnostics – Configuration are NOT reset by this command!

Command:

Byte	Indication	Description
0	Channel	Multisession channel (starting with 0)
1	Mode	0: Physical addressing 1: Functional addressing **In addition:** No response to the request is necessary if the most significant bit is set (0x80)
2, 3	Length	Request length (At the present time, only Length = 0 is supported, the corresponding stop service is transmitted according to the diagnostic type)
4.. (3+Length)	Request	Request, bestehend aus SID (Service-Identifier) und Daten

The 0xA4 KLine Diagnostics – Stop Session command for stopping the diagnostics (or for stopping the communication) is identical for all K-Line protocols at first sight on the part of the host.

Within the K-Line driver, however, specific actions matched to the individually active protocol are released.

All protocols react in a different way. Generally one can say:
The K-Line protocol driver always delivers a response to the 0xA4 KLine Diagnostics – Stop Session command (either automatically or via the 0xA3 KLine Diagnostics – Get Response Buffer query, depending on the response mode set)!
But depending on the protocol, the meaning of the response data varies. The receiver (host) is responsible for the correct interpretation.

The following table shall make clear the processes within the protocol driver running as the response to a 0xA4 KLine Diagnostics – Stop Session command.
Attention is to be paid to the fact that a real separation between the diagnostic and the transport protocols does not exist on the K-Line. To simplify it one can say that “Stop communication” on the K-Line is to be considered identical to “Stop diagnostics”.

KWP2000	
Description	Response of the K-Line driver
The K-Line driver generates and sends a ***Stop Communication*** request (KWP2000 frame, ***SID*** = 0x82). The ECU responds with a ***Stop Communication*** positive (or negative) response (KWP2000 frame, ***SID*** = 0xC2 or ***SID*** = 0x7F, 0x82, 0xXX). After a ***Positive Response*** communication will be finished.	Data division of the ***Stop Communication*** response of the ECU

KWP1281	
Description	Response of the K-Line driver
The K-Line driver generates and sends a ***DiagnosticEnd*** block (KWP1281 block, ***BT*** = 0x06). In case of success, the control unit responds with an ***Acknowledge*** block (***BT*** = 0x09) before the end of the communication or it stops the communication immediately without any acknowledgement.	Data division (***BT***) of the ***Acknowledge*** block (always, even if there is no block of the ECU)

ISO-9141-Ford	
Description	Response of the K-Line driver
The K-Line driver generates and sends a ***Request Operational State Entry*** block (***Mode*** = 0x20). The ECU responds with a ***General Response*** frame (***Mode*** = 0x7F, in case of success ***Response Code*** = 0x00). In case of success, the communication will be terminated then.	Data division of the ***General Response*** block

4.5.7 0xA5 KLine Diagnostics – Get State

Query the K-Line diagnostic state for the multisession channel defined by Channel with this command.
Additionally, the firmware internal LastErrorCode can be reset.

The value of the LastErrorCode in the Response corresponds to the value of the firmware internal LastErrorCode before its resetting.

Generally the firmware internal LastErrorCode is reset automatically without calling this 0xA5 KLine Diagnostics – Get State command by starting a diagnostic session by 0xA1 KLine Diagnostics – Start Session or by stop of a diagnostic session with 0xA4 KLine Diagnostics – Stop Session and Length ≠ 0.

Command:

Byte	Indication	Description
0	Channel	Multisession channel (starting with 0)
1	ResetLastError	0 = Does not reset LastErrorCode 1 = Resets LastErrorCode
2, 3	reserved	Reserved

Response:

Byte	Indication	Description
0	Channel	Multisession channel (starting with 0)
1	LastErrorCode	Error code (0 = no error)
2	DiagType	Type of diagnostics: 0: No diagnostics 1: Diagnostics KWP2000 2: Diagnostics KWP1281 3: Diagnostics ISO-9141-Ford
3	State	State of diagnostics: 0: Not initialized 1: No connection 2: Connection is being established 3: Connection was established 4: Connection is released
4	Flags	Bit 0 = 0: Idle Bit 0 = 1: Busy (a request has not been responded yet or successfully sent) Bit 1 = 0: The diagnostic response buffer is empty Bit 1 = 1: RxBufferNotEmpty (the buffer is not empty yet) Bits 2..7: Reserved
5..7	reserved	Reserved

I

K

L

M

P

R

S

T

U

W