Product Specification

PXI 3078

LIN/ KLine Interfaces

User Manual

Version 1.2

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: 26.04.2012

Issue: April 2012

1 Board Installation

1.1 Hardware Installation

Warning

Please make absolutely certain that all of the installation procedures described below are carried out with your system switched off and disconnected from the mains supply.

Tip

Please refer also to the user manual of your PXI system for additional installation instructions that possibly have to be followed.

Warning

Electro Static Discharge (ESD) can harm your system and destroy electronic components. This can lead to irreparable damage on both the PXI 3078 board and the system hosting it as well as to unexpected malfunction of your test system.
Therefore do not touch the board surface or any connector pins and electronic components.

The CompactPCI™ or PXI™ system is to be opened according to its conditions. A free slot is to be selected in your system.
Now, the slot cover is to be taken away from the slot selected.
To do this, unscrew the two fixation screws and remove the cover from the slot.

(If it is necessary to exchange transceiver modules, pay attention to the general rules to avoid electrostatic charging, see above.
Transceiver modules must never be removed or mounted with the power switched on!
Additionally, the right alignment is absolutely required.)

Insert the board carefully into the prepared slot.
Use the lever at the front plate in order to push in the board finally.

When the board has been inserted properly, it is to be fixed by means of the two screws at the front plate.
Now, the board has been installed correctly.

Afterwards, carry out the operations required at the system to make it ready for operation anew.

1.2 Driver Installation

1.2.1 Windows Device Driver Installation

PXI 3078 boards can be operated under Windows® 2000/ XP and under Windows®7/ 64 bit.

Due to the plug and play capability of Windows®, for every newly recognized hardware component a driver installation is started automatically via the hardware assistant.
The hardware assistant can carry out the installation of the device driver by using the *inf* file contained in the *GPxi3078* folder of the supplied CD.

If necessary, you can find the required *inf* files as follows:

- *Pxi3078.inf* for Windows® 2000/ XP
  in the *Driver PXI_PCI - W2K, WinXP (Version xx)* folder
- *Pxi3078_x64.inf* for Windows®7/ 64 Bit
  in the *Driver PXI_PCI - Win7_x64 (Version xx)* folder

It is not absolutely essential to restart the system.

The following step is only required in case you do not use the G-API (see also Control Software).

If you want to create your own software for the boards, 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.

1.2.2 VISA Device Driver Installation

For the time being, the PXI 3078 VISA device driver is available for Windows® 2000/ XP.

We recommend VISA V 4.3 and higher for the use with a PXI 3078.

First Step

Copy the *VISA_Driver...* directory from the *GPxi3078* folder of the delivered CD to your hard disk

(Recommendation: Complete directory to *C:* \).

Second Step

VISA for Windows 2000/ XP :

Due to the plug and play capability, for every newly recognized hardware component a driver installation is started automatically via the hardware assistant. Follow the instructions. Enter as target directory the one which contains the *PXI3078.inf* file (according to recommendation: *C:* \VISA_Driver...).

VISA for LabView RT :

For operating PXI 3078 boards under the RT operating system, use the *PXI3078.inf* file in the *C:* \VISA_Driver... directory.

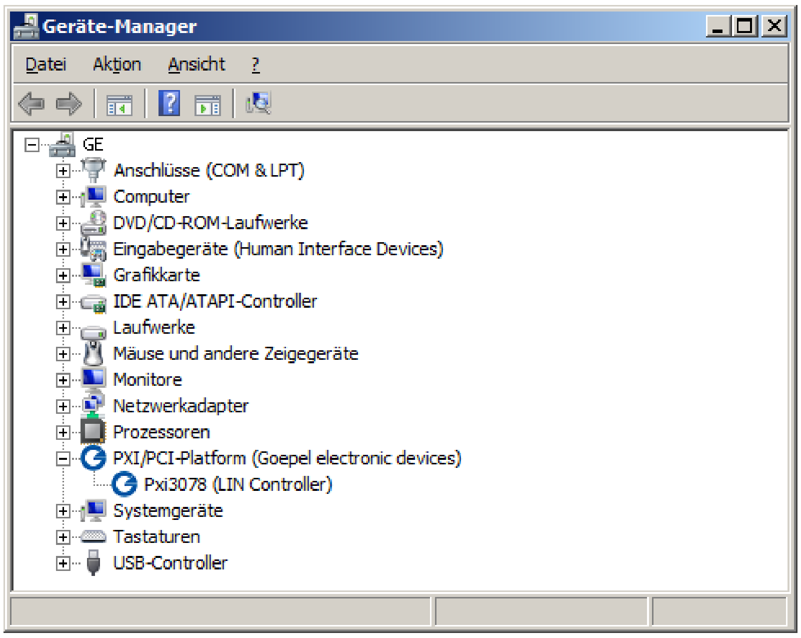
Copy this file to the \ni-rt \system folder of the embedded controller.
Use the NI Measurement & Automation Explorer for that.
The connected RT controller can be found under Remote Systems.
Open a pop-up menu by pressing the right mouse button.
In this menu, select the File Transfer entry and follow the instructions.

If you intend to create a *startup.rtexe* later, copy also your *cvi_lvrt.dll* file to the \ni-rt \system folder.

Third Step:

Reboot your computer to complete installation.

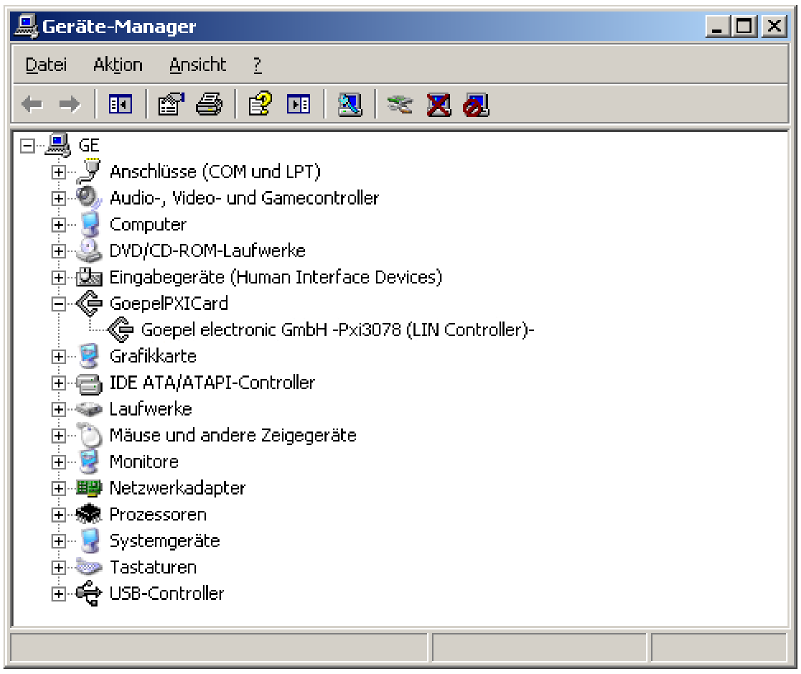
After driver and hardware installation, you can check whether the boards are properly imbedded by the system:


Geräte-Manager

Datei Aktion Ansicht ?

GE
- Anschlüsse (COM & LPT)
- Computer
- DVD/CD-ROM-Laufwerke
- Eingabegeräte (Human Interface Devices)
- Grafikkarte
- IDE ATA/ATAPI-Controller
- Laufwerke
- Mäuse und andere Zeigegeräte
- Monitore
- Netzwerkadapter
- Prozessoren
- PXI/PCI-Platform (Goepel electronic devices)
  - Pxi3078 (LIN Controller)
- Systemgeräte
- Tastaturen
- USB-Controller

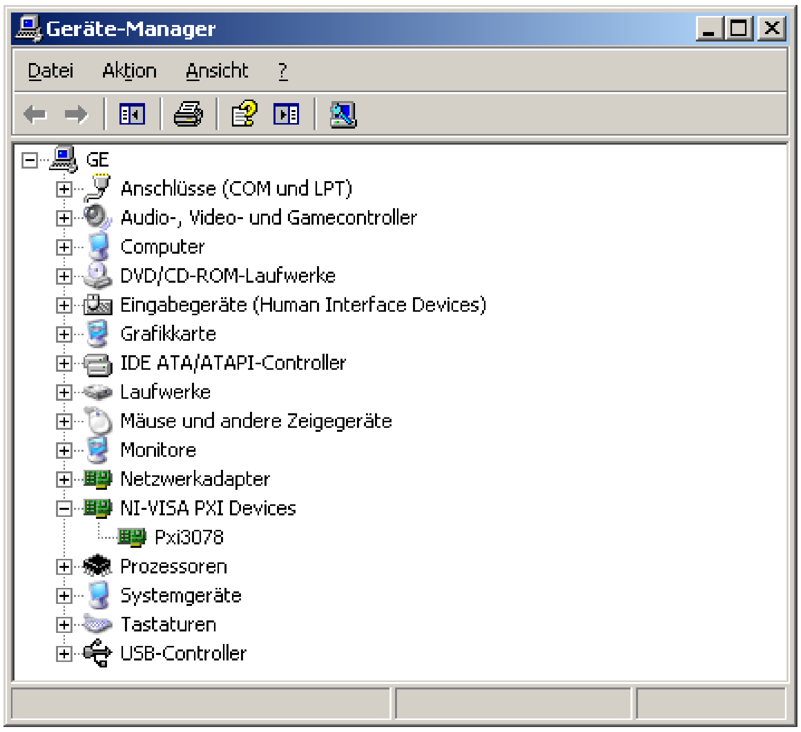
Figure 1-1: Windows 7 (64 Bit)


Geräte-Manager

Datei Aktion Ansicht ?

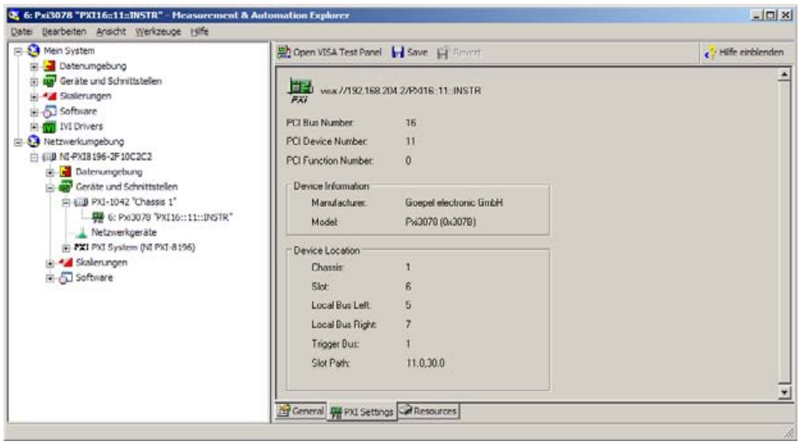
GE
- Anschlüsse (COM und LPT)
- Audio-, Video- und Gamecontroller
- Computer
- DVD/CD-ROM-Laufwerke
- Eingabegeräte (Human Interface Devices)
- GoepelPXICard
  - Goepel electronic GmbH -Pxi3078 (LIN Controller)-
- Grafikkarte
- IDE ATA/ATAPI-Controller
- Laufwerke
- Mäuse und andere Zeigegeräte
- Monitore
- Netzwerkadapter
- Prozessoren
- Systemgeräte
- Tastaturen
- USB-Controller

Figure 1-2: Windows XP



GE
- Anschlüsse (COM und LPT)
- Audio-, Video- und Gamecontroller
- Computer
- DVD/CD-ROM-Laufwerke
- Eingabegeräte (Human Interface Devices)
- Grafikkarte
- IDE ATA/ATAPI-Controller
- Laufwerke
- Mäuse und andere Zeigegeräte
- Monitore
- Netzwerkadapter
- NI-VISA PXI Devices
  - Pxi3078
- Prozessoren
- Systemgeräte
- Tastaturen
- USB-Controller

Figure 1-3:
VISA for Windows XP



- Mein System
  - Datenumgebung
  - Geräte und Schnittstellen
  - Skalierungen
  - Software
  - IVI Drivers
- Netzwerkumgebung
  - NI-PXI8196-2F10C2C2
    - Datenumgebung
    - Geräte und Schnittstellen
      - PXI-1042 "Chassis 1"
        - 6: Pxi3078 "PXI16::11::INSTR"
      - Netzwerkgeräte
      - PXI PXI System (NI PXI-8196)
    - Skalierungen
    - Software

visa://192.168.204.2/PXI16::11::INSTR

PCI Bus Number:	16
PCI Device Number:	11
PCI Function Number:	0

Device Information

Manufacturer:	Goepel electronic GmbH
Model:	Pxi3078 (0x3078)

Device Location

Chassis:	1
Slot:	6
Local Bus Left:	5
Local Bus Right:	7
Trigger Bus:	1
Slot Path:	11.0.30.0

Figure 1-4:
Visa for LabVIEW RT

2 Hardware PXI 3078

2.1 Definition

PXI 3078 LIN boards are communication boards of GOEPEL electronic GmbH , developed for use in a PXI™ bus (= PCI eXtensions for Instrumentation).

PXI 3078 can be used in general control technology, especially for applications in automotive technology. This board provides LIN nodes for data transfer in LIN networks.
It is constructed modular, realizing 2 LIN nodes.
The two nodes can also be configured as KLine nodes.
All nodes are galvanically isolated from the PXI™ interface.

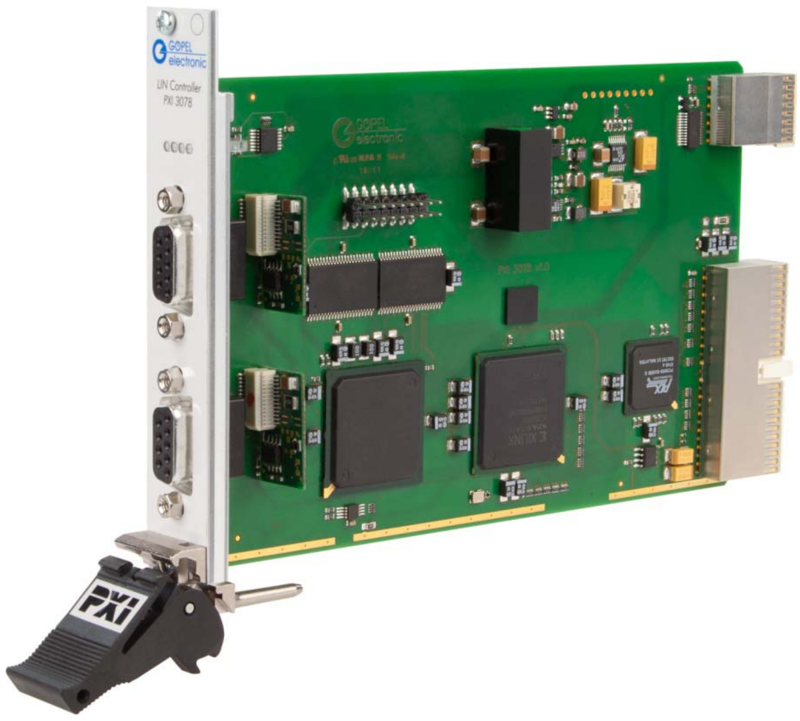


Figure 2-1: PXI 3078

2.2 General Data

The PXI 3078 communication board is connected to the host PC via the PCI bus, which can so control and configure the board. The board has a PCI controller, an FPGA and a TC1796 micro controller.

Any transceivers (LIN/ KLine) can be plugged in the two slots serving for the signal conversion to the corresponding bus standard.

The transceivers are galvanically isolated from the LIN/ KLine controller and the host interface.

2.2.1 PXI Interface

The PXI 3078 communication board is a plug-in board developed for the PXI™ bus (**P**CI e**X**tensions for **I**nstrumentation). Basis of this bus is the CompactPCI™ bus.

The board can be plugged into any desired slot of a CompactPCI™ or PXI™ system (except for slot 1). It can be definitely identified also in the case that several boards of this type are used in the same rack.

The maximal bus transmission rate is 133 MByte/s.

2.2.2 Dimensions

The dimensions of the board correspond to standard dimensions of the accompanying bus system:

- 160 mm x 100 mm x 20.32 mm (L x B x H)

2.2.3 PXI 3078 Characteristics

The PXI 3078 communication board has the following characteristics:

Symbol	Parameter	Min.	Typ.	Max.	Unit
V_{CC1}	Supply voltage 1	3.0	3.3	3.6	V
I_{CC1}	Input current 1			150	mA
V_{CC2}	Supply voltage 2	4.8	5.0	5.2	V
I_{CC2}	Input current 2			300	mA
T_A	Operating temperature	-20	25	60	°C
	Galvanical isolation			1500	VDC
V_{BusInt}	Voltage of the internal bus supply		12		V
I_{BusInt}	Current of the internal bus supply			100	mA
V_{BusExt}	Voltage of the external bus supply			26	V
I_{BusExt}	Current of the external bus supply			300	mA
	LIN Transmission rate		19.2	22	kBaud
	KLine Transmission rate		10.4	115	kBaud

2.3 Construction

2.3.1 General

The PXI 3078 LIN board has two LIN or KLine nodes.
The modular construction allows individual transceiver configuration.
The board recognizes the installed transceivers automatically.

The TC1796 micro controller is for deterministic processing and manipulating of the bus signals. FPGA and PCI controller provide the interface to the host PC via the PXI™ bus.

Two LEDs signalize the board status.

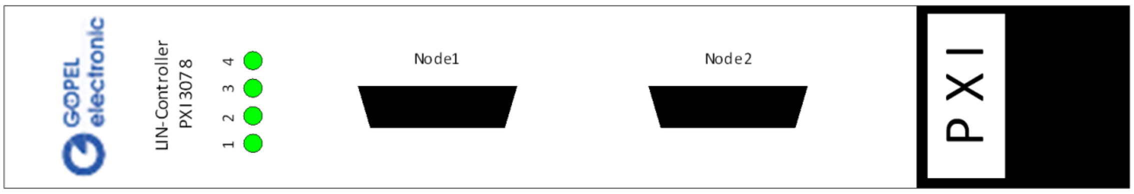


Figure 2-2: PXI 3078 Frontal view

2.3.2 LEDs

The four LEDs indicate the following states:

LED	State	Description
1 and 2	blinking alternately	Micro controller is in Bootloader mode
1	ON	Command processing for Node 1
2	ON	Command processing for Node 2
3		Reserved
4		Reserved

2.4 Function

2.4.1 Addressing

PXI racks do have an own geographical slot addressing of the backplane.
Numbering starts with 1 and can be seen at the cover's frontal side.
Mount always an embedded controller or an MXI card at slot 1.

The PXI 3078 board can read out this geographical slot address.

2.4.2 Frontal Plug connectors

Type: DSub 9 pole socket

The communication nodes are provided via these plug connectors at the frontal edge of the PXI 3078 board.

The assignment of both connectors is identical.
The following table shows the LIN assignment:

Pin	Signal name
1	Reserved
2	Reserved
3	Gnd
4	n.c.
5	n.c.
6	n.c.
7	LIN
8	Reserved
9	External bus supply (V_{BusExt})

The second table shows the KLine assignment:

Pin	Signal name
1	Reserved
2	L-Line
3	Gnd
4	n.c.
5	n.c.
6	n.c.
7	K-Line
8	Reserved
9	External bus supply (V_{BusExt})

2.4.3 Transceiver

The transceivers are designed as plug-in modules recognized by the Firmware automatically.

Any transceivers for LIN or KLine can be plugged in both nodes.

LIN:

Generally, the TJA1020 of Philips company is used as LIN transceiver.

In the standard delivery construction of the transceiver modules, switching over between Master and Slave configuration by a semiconductor relay is carried out per software. The 1kOhm pullup resistors for the LIN Master are switched ON/ OFF accordingly.

KLine:

Generally, the L9637D ST of Microelectronics company is used as KLine transceiver.

2.4.4 Transceiver Supply

Generally, the transceivers are supplied by the PXI 3078 board.
The V_{BusInt} internal bus supply voltage is 12V. It is galvanically isolated.

The V_{BAT} transceiver module's supply voltage can also supplied via the V_{BusExt} connection of the plug connectors. By this, users can operate the bus with voltages greater or less than 12V.
Then, the signal levels must be adapted accordingly. In this case, the internal bus supply voltage should be switched OFF by the software.
The voltage supplied here is limited by the transceiver module to 26V.

2.5 Supply notes

PXI 3078 boards are delivered in the following variants:

- 2x LIN Node or
- 2x KLINE Node

The LIN software contains the LIN communication according to the LIN 1.3, 2.0 and 2.1. specifications including LIN Diagnostics.

3 Control Software

There are three ways to embed PXI 3078 hardware in your own applications:

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

3.1 Programming via G-API

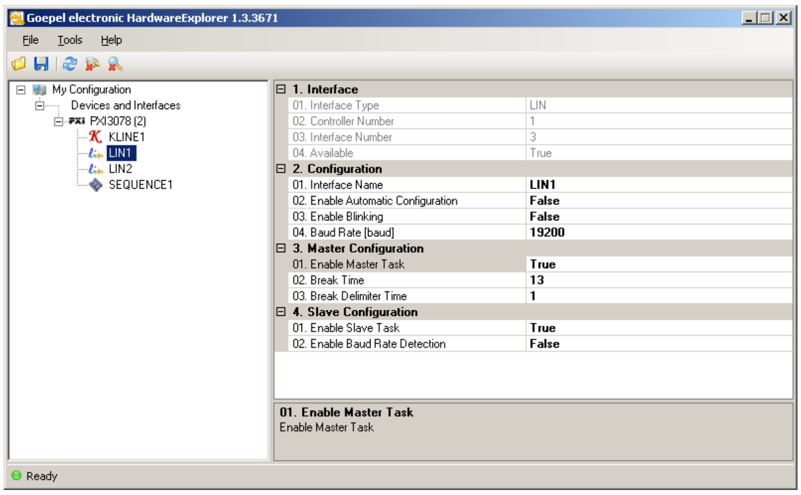
The G-API (GOEPEL-API) is a programming environment based on “C” for GOEPEL electronic hardware under Windows®. So the G-API is the preferred programming environment for this hardware.
It provides a wide, hardware independent command set for CAN, LIN, K-Line, FlexRay, MOST, LVDS, ADIO and Diagnostic services.
No matter whether a PXI/ PCI, USB and Ethernet device is used, the commands remain the same.

The hardware abstraction introduced with the G-API gives the test application parallel access to the hardware, allowing one application to access multiple hardware interfaces, as well as multiple applications can access the same hardware interface in parallel.

Another feature introduced by the G-API is the asynchronous hardware access. This means no execution blocking for pending firmware commands. The command acknowledgement is provided via a callback mechanism.

With the Hardware Explorer GOEPEL electronic provides an efficient hardware configuration and management tool, offering users an easy way to manage their hardware configurations and identifying specific hardware interfaces by logical names. Using logical interface names in the application saves from rebuilding the application when porting it to another interface or controller board, as the interface can be easily reassigned in the Hardware Explorer.
Furthermore, the Hardware Explorer provides a simple means of testing the interaction between hardware and software by executing the integrated self-tests.

The figure below shows the GOEPEL electronic Hardware Explorer:



My Configuration
- Devices and Interfaces
  - PXI PXI3078 (2)
    - KLINE1
    - LIN1
    - LIN2
    - SEQUENCE1

1. Interface	
01. Interface Type	LIN
02. Controller Number	1
03. Interface Number	3
04. Available	True
2. Configuration	
01. Interface Name	LIN1
02. Enable Automatic Configuration	False
03. Enable Blinking	False
04. Baud Rate [baud]	19200
3. Master Configuration	
01. Enable Master Task	True
02. Break Time	13
03. Break Delimiter Time	1
4. Slave Configuration	
01. Enable Slave Task	True
02. Enable Baud Rate Detection	False

01. Enable Master Task
Enable Master Task

Figure 3-1: Hardware Explorer

Please consult the G-API documentation for further information. This documentation and the installation software are located in the *G-API* folder of the supplied “Product Information” CD.

3.2 Programming via DLL Functions

Programming via DLL Functions is further possible for projects which can not be processed with the G-API GOEPEL electronic programming environment.

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).

3.2.1 Windows Device Driver

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

- DriverInfo
- Xilinx Read Write Register
- Write Instruction
- Read Response

The following type definitions are used:

`U8` – unsigned char

`U16` – unsigned short

`U32` – unsigned long

3.2.1.1 DriverInfo

The Pxi3078__DriverInfo function is used for the status query of the hardware driver.

Format:

```
S32 Pxi3078__DriverInfo(t_DriverInfo *pDriverInfo,
                                 U32  Length);
```

Parameters:

Pointer, for example pDriverInfo, to a data structure

For the structure, see the *Pxi3078_UserInterface.h* file on the supplied CD

Length
Size of the storage area pDriverInfo is pointing to, in Bytes

Description:

The Pxi3078__DriverInfo function returns information regarding the status of the hardware driver.

For this reason, the address of a pDriverInfo pointer has to be transferred to the function.
Within the function, the structure pDriverInfo is pointing to is filled with various pieces of information.

3.2.1.2 Xilinx Read Write Register

The Pxi3078__XilinxReadWriteRegister function allows FPGA access.

Format:

```
S32 Pxi3078__XilinxReadWriteRegister(U8 *pData,
                                     U32 *DataLength);
```

Parameters:

Pointer, for example pData, to the Write data area, consisting of Command Header and Command Bytes (see Command Structure)

DataLength
Size of the storage area pData is pointing to, in Bytes

Description:
See above

3.2.1.3 Write Instruction

The Pxi3078__WriteInstruction is used for sending a command to the PXI 3078 controller.

Format:

```
S32 Pxi3078__WriteInstruction(U8 *pData,
                              U16  DataLength);
```

Parameters:

Pointer, for example pData, to the Write data area, consisting of Command Header and Command Bytes (see Command Structure)

DataLength
Size of the storage area pData is pointing to, in bytes

Description:

The Pxi3078__WriteInstruction function sends a command to the PXI 3078 controller.

In the header of the structure pData is pointing to, there is the information regarding the PXI 3078 board to be activated by this function. Therefore this parameter has not to be given separately, see General Firmware Notes.

3.2.1.4 Read Response

The Pxi3078__ReadResponse function is used for reading a response from the PXI 3078 controller.

Format:

```
S32 Pxi3078__ReadResponse(U8  Device,
                           U8 *pData,
                          U32 *DataLength);
```

Parameters:

Device

Index of the PXI 3078 board, beginning left with 1

Pointer, for example pData, to the Read data area, consisting of Response Header and Response Bytes (see Response Structure)

DataLength

Value of the parameter before function call:
Size of the buffer pointed by pData in bytes

Value of the parameter after function call:
Number of bytes actually read

Description:

The Pxi3078__ReadResponse function reads back the oldest response written by the PXI 3078 controller.

If several responses have been provided by the controller, but not read, they are not lost but stored in the form of a list.

On calling up, the Pxi3078__ReadResponse function continues to supply data until this list contains no more entries.

3.2.2 VISA Device Driver

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

- Init
- Done
- Driver Info
- Write Data
- Read Data
- Xilinx Read Write Register

3.2.2.1 *Init*

The PXI3078_Init function is used to open VISA sessions for the system's PXI 3078 boards including initialization.

Format:

```
ViStatus PXI3078_Init(ViUInt32 *CardCount);
```

Parameter:

CardCount

Number of the system's PXI 3078 boards recognized by the VISA driver.

Description:

The PXI3078_Init function searches for all PXI 3078 boards of the system and opens the required sessions.
Additionally, board internal initializations are carried out.

Therefore this function must be executed as the first step.

3.2.2.2 *Done*

The PXI3078_Done function closes all VISA sessions of the system's PXI 3078 boards.

Format:

```
ViStatus PXI3078_Done(void);
```

Parameters:

None

Description:

The PXI3078_Done function closes all VISA sessions of the system's PXI 3078 boards.

No further access to the boards is possible after executing this function.

3.2.2.3 Driver Info

The PXI3078_DriverInfo function provides general information regarding the driver and the board.

Format:

```
ViStatus PXI3078_DriverInfo(PXI3078_StructDriverInfo *DriverData,
                                                ViChar *DeviceName);
```

Parameters:

Pointer, for example DriverData, to a Data structure
For the structure see the *PXI3078_API.h* file of the supplied CD

DeviceName
`Array[K_DEV_MAX][K_RES_NAME_LENGTH]`
(See *PXI3078_API.h*)

Description:

The PXI3078_DriverInfo function provides information regarding the driver and the system's PXI 3078 boards.

The DeviceName indicates the resource names registered by VISA. This information correlates with the display of NI MAX.

3.2.2.4 *Write Data*

The PXI3078_WriteData function is used for writing data to the PXI 3078 controller.

Format:

```
ViStatus PXI3078_WriteData(ViUInt8 *WriteData,
                          ViUInt32  Length_In_Bytes);
```

Parameters:

Pointer, for example WriteData, to the Write data area,
consisting of Command Header and Command Bytes
(see Command Structure)

Length_In_Bytes
Size of the storage area WriteData is pointing to, in bytes

Description:

The PXI3078_WriteData function allows writing data to the PXI 3078 boards.

In the header of the structure WriteData is pointing to, there is the information regarding the PXI 3078 board to be activated by the function.

Therefore this parameter is not to be given separately.

The structure is described in chapter General Firmware Notes.

3.2.2.5 Read Data

The PXI3078_ReadData function is used for reading data from the selected interface of the PXI 3078 controller.

Format:

```
ViStatus PXI3078_ReadData(ViUInt8  Card,
                           ViUInt8  Port,
                           ViUInt8 *ReadData,
                          ViUInt32 *DataLength);
```

Parameters:

Card

Index of the PXI 3078 board, beginning left with 1

Port

Interface number (1..2)

Pointer, for example ReadData, to the Read data area, consisting of Response Header and Response Bytes (see Response Structure)

DataLength

Value of the parameter before function call:
Size of the buffer pointed by ReadData in bytes

Value of the parameter after function call:
Number of bytes actually read

Description:

The PXI3078_ReadData function allows reading data provided by a PXI 3078 board
(see also Read Response in the Windows Device Driver section).

3.2.2.6 Xilinx Read Write Register

The PXI3078_XilinxReadWriteRegister function is used for reading register data of the FPGA.

Format:

```
ViStatus PXI3078_XilinxReadWriteRegister(ViUInt8 *ReadData,
                                         ViUInt32 *DataLength);
```

Parameters:

Pointer, for example ReadData , to the Read data area

DataLength

Value of the parameter before function call:
Size of the buffer pointed by ReadData in bytes

Value of the parameter after function call:
Number of bytes actually read

Description:

The PXI3078_XilinxReadWriteRegister function allows reading register data of the FPGA

3.3 Programming with LabVIEW

3.3.1 LabVIEW via G-API

The supplied CD contains VIs for activating PXI 3078 boards under LabVIEW.

These LabVIEW VIs use the functions of the GOEPEL G-API.

3.3.2 LLB using the Windows Device Driver

The supplied CD contains VIs for activating PXI 3078 boards under LabVIEW.

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

3.3.3 LLB using the VISA Device Driver

The supplied CD contains VIs for activating PXI 3078 boards under LabVIEW.

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

3.4 Further GOEPEL Software

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

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

4 Firmware Commands

4.1 General Firmware Notes

Things in common for all Firmware commands and responses on a GOEPEL electronic PXI 3078 board are described in this chapter (see also Interfaces).

After the subsections and chapters

- Interfaces
- Data Types
- Header
- Command Structure
- Response Structure
- Command Acknowledgment and
- Command Example

there is the actual command description in the subsections

- General Firmware Commands
- LIN Commands
- KLine 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!

4.1.1 Interfaces

The assignment of the Firmware to the corresponding Interface numbers (Nodes) for a PXI 3078 board can be taken from the following table:

Software Interface	Controller Number	Interface Number	PXI 3078 Interface
LIN	1	3	LIN1
		4	LIN2
K-LINE	1	5	K-LINE1
		6	K-LINE2

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 Byte)
	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:

Data type	Value	Bytes	Byte Offset
16 bit integer (short)	**0x1234**	**0x34** **0x12**	0 (Low Byte) 1 (High Byte)
32 bit integer (long)	**0x12345678**	**0x78** **0x56** **0x34** **0x12**	0 (Low Byte) 1 2 3 (High Byte)
32 bit floating point (float)	**123.456**	**0x79** **0xE9** **0xF6** **0x42**	0 (Low Byte) 1 2 3 (High Byte)
64 bit 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..4096 (see Command Structure) Total 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 on the controller** (**3**..**6** = **LIN**/ **KLINE**, **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 on the controller (3..6 = LIN/ KLINE, Response)
8	Type	0: Command 1: Response 2: Command acknowledgment
9	ApplicationHandle	The contents of ApplicationHandle is sent back to the host unchanged by the controller in the case of responses
10	ModuleAddress	Address (number) of the PXI 3078 board within the system, 1..12 Boards of other types are NOT considered!!!
11	Command	Command code (0x00..) Used codes according to Firmware, see General Firmware Commands, LIN Commands and KLine Commands

Tip

If possible use always command acknowledgment (Flags/ Bit 0 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 Command Structure

Generally, 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 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 numbers by the maximum size of the command (MESSAGE_SIZE) and the size of the Command header (HEADER_SIZE).

The maximum number of Command bytes (PARAM_SIZE) results from the following equation:

PARAM_SIZE = MESSAGE_SIZE - HEADER_SIZE

Symbolic constant	Value for PXI 3078	Remarks
MESSAGE_SIZE	4096	Maximum size of the command or response including Header in bytes
HEADER_SIZE	12	Size of the Header in bytes
PARAM_SIZE	4084	Maximum size of the parameters (Command bytes or Response bytes) in bytes

4.1.5 Response Structure

A response consists of Response header and the Response bytes. The Response header consists of the header with Type = 1 and the command code Command according to the corresponding executed command. 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.

On the other hand, there are commands without command bytes. Then, only the corresponding response is described.

4.1.6 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	Meaning
0..3	ErrorNumber	0: No error Otherwise: Error
4.. 3+N	ErrorDescription	Zero terminated error string of N length (including final “Zero” character) 1 ≤ N ≤ (PARAM_SIZE – 4) (PARAM_SIZE see Command Structure)

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.

Tip

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.

4.1.7 Command Example

The following example shows the individual bytes including the header of the 0x30 LIN Frame Response Definition command and the belonging command acknowledgment.

The command is sent to software interface 4 (LIN2) of the controller of the third PXI 3078 board (by function call Write Instruction, see Programming via DLL Functions).
Then, the command acknowledgment is read from the same controller of the same board.

Command (including Header):

Byte Index	Byte Value	Indication	Meaning
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
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, returned unchanged to the Acknowledgment
10	**0x03**	ModuleAddress	Address of the board = 3 (third PXI 3078 board)
11	**0x30**	Command	Command code for 0x30 LIN Frame Response Definition
12	**0x12**	Id	Identifier = 0x12
13	**0x01**	Mode	1: Output of the LIN frame response
14	**0x00**	PrepareMode	0: No preparing of the LIN frame response
15	**0x03**	MessageCount	Output LIN frame response 3 times
16	**0x06**	DLC	Data length = 6
17	**0x00**	reserved	Reserved
18	**0x00**		Reserved
19	**0x00**		Reserved
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	Meaning
0	`0x23`	StartByte	0x23 (“#” ASCII character)
1	`0x00`	Flags	No flag 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	Interface on the controller = 4
8	`0x02`	Type	2: Command acknowledgment
9	`0x00`	ApplicationHandle	The content is sent back unchanged
10	`0x03`	ModuleAddress	Address of the board = 3 (third PXI 3078 board)
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”)

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:

- Interfaces
- 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.

4.2.2 0x10 Software Reset

This command resets the selected controller to the initial state including a software reset.

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 used 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	Meaning
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 KLine Functionalities:

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

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

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

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

4.3 LIN Commands

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

General notes valid for all Firmware commands can be found in section General Firmware Notes.

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 Detection Threshold command.

Command sequence

After switching on or a software reset, available Optional Functionalities must be enabled by the 0x03 Enable Functionalities command.

Then, the following 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:

- 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

Structure of a LIN Cluster

The following information is taken from the LIN Specification 2.0.

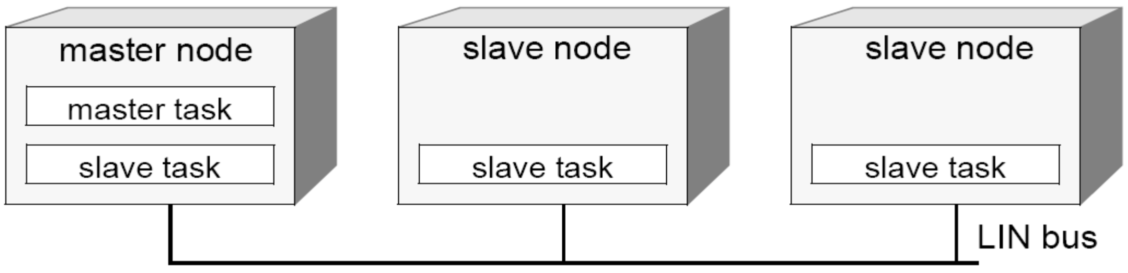


Figure 4-1: Structure of a LIN Cluster

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.

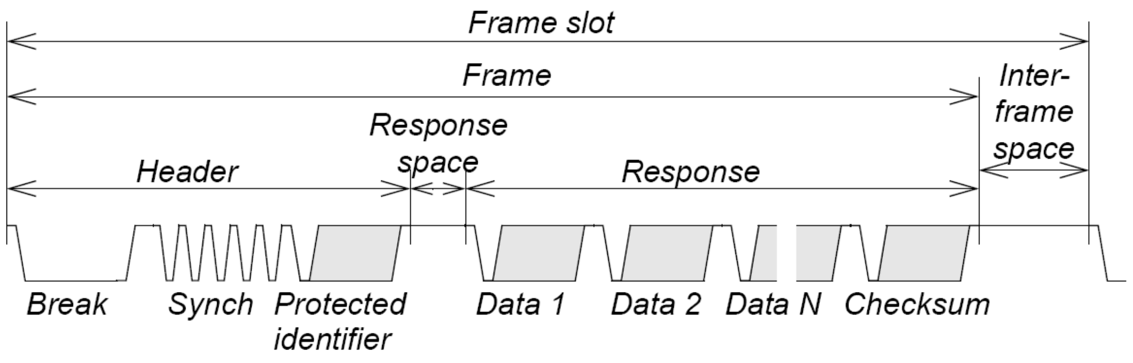


Figure 4-2: Structure of a LIN Frame

Essentially a LIN Frame consists of the Header and the Response.

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

The LIN Frame Response is sent by a slave task of the LIN Master (master node) or of a slave node.

The pause between the LIN Header and the LIN Frame Response is called Response space.

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

All time-parameters within a LIN message changeable by Firmware commands can be seen (BreakTime, BreakDelimiterTime, ArbitrationTime, ResponseSpace and InterByteSpace):

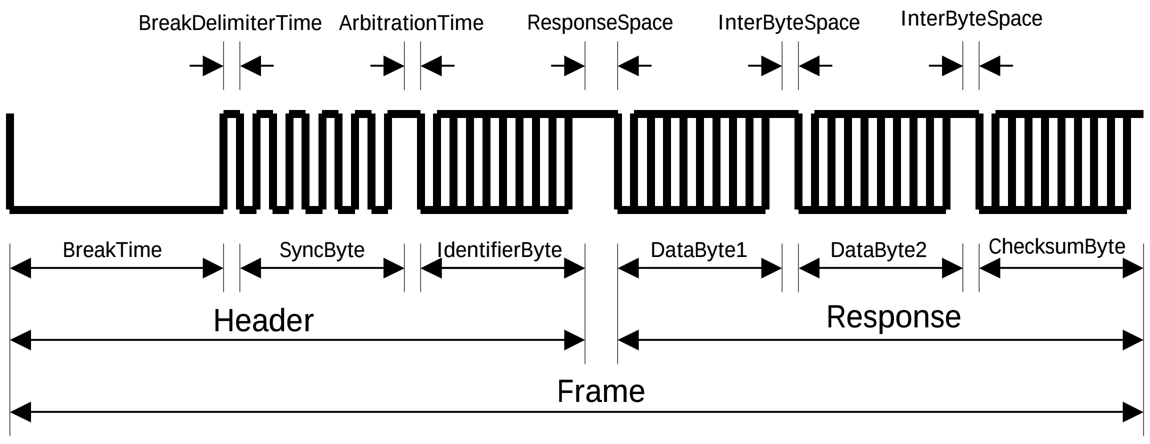


Figure 4-3: LIN Message (Frame)

The continuous lines above and below the LIN signal indicate that these signals can have V_{Bat} (high) level as well as Gnd (low) level during the corresponding periods of time.

4.3.1 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	Meaning
0	reserved	Reserved
1	DontUseUartForTx	0: Use UART for transmitting (default) 1: No use of UART for transmitting
2	reserved	Reserved
3	BlinkMode	0: Blinking of the LEDs deactivated (default) 1: Blinking of the LEDs activated

For sending without jitters (Response Space 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.

4.3.2 0x14 LIN Set Interface Properties

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

Command:

Byte	Indication	Meaning
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, i.e., the EnableSlaveTask parameter has to be set to 1
2, 3	reserved	Reserved

Parameters of the Master task:

Byte	Indication	Meaning
4..7	BaudRate	BaudRate in Hz
8..11	BreakTime	BreakTime in multiples of 25ns (Generally 13 Bit-times, e.g. 27,083 for 19,200 Baud) (Time that announces the start of a new LIN Frame, duration of the low level of the Break, see Figure 4-3)
12..15	BreakDelimiterTime	BreakDelimiterTime in multiples of 25ns (Generally 1 Bit-time, e.g. 2,083 for 19,200 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 4-3)
16..19	ArbitrationTime	Only for Advanced library, otherwise to be initialized by 0 ArbitrationTime in multiples of 25ns (generally 0) (Time between the end of the StopBit of the SyncByte and the beginning of the StartBit of the IdentifierByte, see Figure 4-3)

Parameters of the Slave task:

Byte	Indication	Meaning
20	Enable-BaudRateDetection	0: Baud Rate Detection deactivated 1: Baud Rate 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 25ns (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 4-3)
32..35	InterByteSpace	Only for Advanced library, otherwise to be initialized by 0 InterByteSpace in multiples of 25ns (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 4-3)

Warning

The monitor commands DO NOT provide results when the slave task is deactivated!!!

Warning

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 5µs and 15µs (generally 8µs to 9µs).

4.3.3 0x15 LIN Set Checksum Model

Use this command to set the checksum model.

Command:

Byte	Indication	Meaning
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)

4.3.4 0x1A LIN Node

This command is used for the configuration of the LIN interface as a LIN node.

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 (as far as more than four bytes exist).

Command and Response:

Byte	Indication	Meaning
0	Mode	4: PROPERTY__SET_BY_ID 5: PROPERTY__GET_BY_ID
1..3	reserved	Reserved

PropertyId:

Wert	Meaning
18	**BAUD_RATE** Baudrate in Hertz
19	**BAUD_RATE_DETECTION__IS_ENABLED** Baudrate detection (0 = deactivated, 1 = activated)
20	**UART__USE_FOR_TRANSMIT** Definition whether the UART is used for sending, without new initialization of the interface (0 = The UART is NOT used for sending 1 = The UART is used for sending) See also 0x12 LIN Init Interface command
49	**ENABLE_BUS_TERMINATION** 0 = Deactivation of the bus termination (pull up resistor) 1 = Activation of the bus termination (pull up resistor)
50	**ENABLE_INTERNAL_VBAT** 0 = Deactivation of the internal voltage supply for the transceiver 1 = Activation of the internal voltage supply for the transceiver Then, connecting an external voltage is not applicable

4.3.4.1 Property SetById

The SubCmd = PROPERTY__SET_BY_ID subcommand is used to set the value of a property specified by PropertyId.

Command:

Byte	Indication	Meaning
4, 5	Id	PropertyId: Property identifier (see PropertyId in section 0x1A LIN Node)
6, 7	Reserved	Reserved
8..11	Value	Value of the property

4.3.4.2 Property GetById

The SubCmd = GET_FLAG_BY_ID subcommand is used to query for the value of a property specified by PropertyId.

Command:

Byte	Indication	Meaning
4, 5	Id	PropertyId: Property identifier (see PropertyId in section 0x1A LIN Node)
6, 7	reserved	Reserved

Response:

Byte	Indication	Meaning
4, 5	Id	PropertyId: Property identifier (see PropertyId in section 0x1A LIN Node)
6, 7	Reserved	Reserved
8..11	Value	Value of the property

4.3.5 0x22 LIN Fill Schedule Table

Use this command to fill a LIN Schedule table.

Altogether there are 256 Schedule tables in the Firmware, the entries depend on the available memory range.

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	Meaning
0	ScheduleTable-Number	Number of the Schedule Table
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 (sending 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	Meaning
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 25ns (e.g. 400,000 for a Delay of 10ms)

Similar to the Frames list for the Unconditional Frames of a LIN Description File (LDF),
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.

4.3.6 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	Meaning
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	Meaning
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.

4.3.7 0x24 LIN Send WakeUp Request

Use this command to send one wakeup request.

Command:

Byte	Indication	Meaning
0..3	WakeUpTime	Duration of the dominant bus level in multiples of 25ns, (e.g. 10,000 for a WakeUp time of 250µs) (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 250µs to 5ms.

4.3.8 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	Meaning
0	Awake	0: Sleep state 1: Awake state
1..3	reserved	Reserved

4.3.9 0x26 LIN Send GotoSleep Command

This command activates the possibility to send the Go to Sleep Command. Just after the activation executing the command is completed.

The command does not have any command bytes.

Sending the Go to Sleep Command itself is executed by the LIN master by transmitting a LIN header with the identifier 0x3C (diagnostic master request frame) to the bus (a 0 is sent as the first data byte, then). This means exclusively the LIN master determines the point in time to send the Go to Sleep Command.

The Go to Sleep Command can be sent by any controller (also by the slave).

Warning

Sending the Go to Sleep Command is NOT possible while diagnostics is active (Type ≠ 0 for 0xA0 KLine Diagnostics – Configuration).

4.3.10 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	Meaning
0	ScheduleTableNumber	Number of the Schedule table
1..3	reserved	Reserved

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

4.3.11 0x29 LIN Master – Stop Transmitting

The master task stops to send LIN Frame headers by executing this command.

The command does not have any command bytes.

4.3.12 0x2A LIN Clear Schedule Table

Empty the indicated Schedule table by this command.

Command:

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

4.3.13 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	Meaning
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)

4.3.14 0x2C LIN Change Schedule Table

Change to another Schedule table by this command (even if the current table is just proceeded).

As a rule changing to another Schedule table is always carried out after proceeding the current table. By setting the ChangeImmediately flag an immediate changing can be enforced.

Command:

Byte	Indication	Meaning
0	ScheduleTableNumber	Number of the Schedule table
1	Flags	In <u>Normal cases</u> ALL flags are 0 Bit 0: StartSchedulerBefore (the Scheduler is started before changing the Schedule table) Bit 1: StopSchedulerAfter (the Scheduler is stopped after NumberOfCycles runs of the Schedule table) Bit 2: ChangeToPreviousTableAfter (the Scheduler changes back to the previous Schedule table after NumberOfCycles runs of the Schedule table) Bit 3: ChangeImmediately (the Scheduler changes immediately to the specified Schedule table) Bit 4: StartSchedulerOnWakeUp (the Scheduler starts automatically after a WakeUp) Bits 5..7: Reserved
2, 3	reserved	Reserved
4..7	NumberOfCycles	Number of Schedule table runs before changing back to the previous Schedule table (in the case the ChangeToPreviousTableAfter flag is set) or the Scheduler is stopped (in the case the StopSchedulerAfter flag is set). If none of both flags is set, NumberOfCycles is to be set to 0.

The specified Schedule table is <u>always</u> proceeded starting with the FIRST entry.

The Scheduler is responsible for proceeding the Schedule table in the Master task.

Therefore the Schedule table is NOT proceeded when the Scheduler is stopped.
Accordingly, NO LIN Frame headers are sent by the Master task.

4.3.15 0x2E LIN Master Task

This command is used to control the LIN Master Task.

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 (as far as more than four bytes exist).

Command and Response:

Byte	Indication	Meaning
0	SubCmd	0: SEND_HEADER
1..3	reserved	Reserved

4.3.15.1 Send Header

The SubCmd = SEND_HEADER subcommand is used to immediately send a LIN frame header without filling and starting a Schedule table.

Command:

Byte	Indication	Meaning
4	Flags	Bit 0 = 0: The protected identifier (PID) is calculated automatically Bit 0 = 1: The protected identifier (PID) is direct the value of Id Bits 1..7: Reserved
5	Id	LIN identifier
6, 7	Reserved	Reserved

4.3.16 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) and creating a group of LIN Frame responses (PrepareMode = 1).

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

Command:

Byte	Indication	Meaning
0	Id	Identifier (0x00..0x3F)
1	Mode	0: No sending of the LIN Frame response 1: Sending of the LIN Frame response
2	PrepareMode	0: No preparing of the LIN Frame response 1: Preparing of the LIN Frame response
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

“Preparing” serves to start or stop several LIN Frame responses parallel (see the 0x34 LIN Start Prepared Frame Responses and 0x35 LIN Stop Prepared Frame Responses commands).

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

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

Command:

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

4.3.18 0x32 LIN Change Frame Response Mode

The command serves to change the Mode and PrepareMode parameters of the LIN Frame response indicated by Id.

Then this LIN Frame response can be sent MessageCount times.

Command:

Byte	Indication	Meaning
0	Id	Identifier (0x00..0x3F)
1	Mode	0: No sending of the LIN Frame response 1: Sending of the LIN Frame response
2	PrepareMode	0: No preparing of the LIN Frame response 1: Preparing of the LIN Frame response
3	MessageCount	0: Send LIN Frame response always 1 ≤ N ≤ 255: Send LIN Frame response N times

4.3.19 0x33 LIN Change Frame Response Data

The command serves to change the Dlc and Data parameters of the LIN Frame response indicated by Id.

Then this LIN Frame response can be sent MessageCount times.

Command:

Byte	Indication	Meaning
0	Id	Identifier (0x00..0x3F)
1	reserved	Reserved
2	MessageCount	0: Send LIN Frame response always 1 ≤ N ≤ 255: Send LIN Frame response N times
3	Dlc	Data length (0..8)
4..11	Mask[0..7]	Mask bytes 0..7
12..19	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)

4.3.20 0x34 LIN Start Prepared Frame Responses

After calling this command, all LIN Frame responses with the PrepareMode parameter set to 1 are sent.

The command does not have any command bytes.

4.3.21 0x35 LIN Stop Prepared Frame Responses

After calling this command, sending of all LIN Frame responses with the PrepareMode parameter set to 1 is stopped.

The command does not have any command bytes.

4.3.22 0x3A LIN Define Message Counter

Define a message counter within a certain data byte of the data of a LIN Frame response by this command.

With each sending of the message this counter is increased by 1.

Command:

Byte	Indication	Meaning
0	Id	Identifier (0x00..0x3F)
1	Mode	0: Deactivate counter 1: Activate counter
2, 3	reserved	Reserved
4	Byte	Starting byte position of the counter within the LIN Frame response (0..7)
5	Bit	Starting bit position of the counter within Byte (0..7)
6	BitLength	Bit length of the counter (1..8)
7	StartValue	Starting value of the counter

For Mode = 0 the 4 to 7 command bytes are not relevant and may be left out.

4.3.23 0x40 LIN Set Bus Baud Rate

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

Command:

Byte	Indication	Meaning
0..3	BaudRate	BaudRate in Baud (generally 19,200); In the case of 0 for BaudRate, the automatic baud rate detection is activated

The BaudRate range of values is 700..125,000.
That corresponds to a minimum BaudRate of 700Baud and a maximum BaudRate of 125KBaud.

4.3.24 0x46 LIN Set Break Detection Threshold

This command sets the BreakDetectionThreshold parameter.

Command:

Byte	Indication	Meaning
0..3	BreakDetection-Threshold	BreakDetectionThreshold in per cent 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

4.3.25 0x47 LIN Set Wake Up Delimiter Time

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 100ms after a WakeUp. That means 100ms after the end of a WakeUp the Master task should start communication again.

Command:

Byte	Indication	Meaning
0..3	WakeUpDelimiterTime	WakeUpDelimiterTime in multiples 25ns, (e.g. 4,000,000 for a WakeUpDelimiterTime of 100ms) (0: Default WakeUpDelimiterTime)

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

4.3.26 0x50 LIN Monitor – Control

This command is used to control the monitor. 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 (as far as more than four bytes exist).

Command and Response:

Byte	Indication	Meaning
0	Mode	0: Control of monitor start-up behavior 1: Query of the current monitor time
1..3	reserved	Reserved

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

Byte	Indication	Meaning
4..7	Flags	Bit 0: DisableStartingOfMonitorTimeWithZero (the monitor does not start with time stamp = 0 for the start-up point in time, that means the internal monitor time is NOT reset) Bit 1: DontDiscardTooEarlyMonitorTimeStamps (monitor entries with a timestamp value prior to the start-up point in time of the monitor are NOT discarded; Example: A LIN message is just being sent, exactly while sending the monitor is started) Bits 2..31: Reserved
8..11	reserved	Reserved

There are no further Response parameters for Mode = 0.

The following Command parameters are only valid for Mode = 1:

Byte	Indication	Meaning
4..7	reserved	Reserved

The following Response parameters are only valid for Mode = 1:

Byte	Indication	Meaning
4..7	Flags	Bit 0: MonitorIsStarted (the monitor is running) Bits 1..31: Reserved
8..11	MonitorTimeOfStart	Monitor time during starting-up the monitor
12..15	CurrentMonitorTime	Current monitor time
16..19	reserved	Reserved

4.3.27 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	Meaning
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	Meaning
4	StartId	Start identifier for the filter range
5	EndId	End identifier for the filter range
6, 7	reserved	Reserved

Parameters for Mode = 2:

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

4.3.28 0x54 LIN Monitor – Activation/ Deactivation

In the case of activating this command, Buffer reception or List reception can be selected for the Mode parameter.
If you select Buffer reception, further parameters are required.

For Buffer reception, the LIN messages come in consecutively into a ring buffer after passing the monitor filter as sent on the bus/ received by the bus.
This internal ring buffer can buffer up to about 1,024 LIN messages.

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

Command:

Byte	Indication	Meaning
0	Mode	0: Deactivating monitor 1: Activating Buffer reception (for the structure see below) 2: Activating List reception

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

Byte	Indication	Meaning
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).

Warning

The monitor commands DO NOT provide results when the slave task is deactivated!!!

After activating Buffer reception with AutomaticEmpty = 1, the selected controller independently sends the received LIN Frames to the host. Therefore the host has to read out the controller(s) cyclically.

During that, the monitor responses have the same structure as the responses of the

0xF2 LIN Monitor – Get small Buffer Items (Type = 1) command.

By activating the monitor with Mode = 1 or 2, the timer for creating the StartTime time stamp is set to 0 (see e.g. 0xF2 LIN Monitor – Get small Buffer Items).

The following commands are available to read monitor data:

In the case of Buffer reception with AutomaticEmpty = 0

0xF2 LIN Monitor – Get small Buffer Items.

In the case of List reception

0xF4 LIN Monitor – Get small List Item.

4.3.29 0x64 LIN Sporadic Frame Definition

Define a Sporadic frame with the belonging Unconditional frames by this command.

Sporadic frames can only be sent by a master which sends the corresponding LIN Frame headers.

Command:

Byte	Indication	Meaning
0	SporadicFrame-ListIndex	List index (starting with 0), see also the 0x22 LIN Fill Schedule Table command
1..5	reserved	Reserved
6, 7	NumberOf-Unconditional-Frames	Number of Unconditional frames (N) belonging to a Sporadic frame
8.. 7+(N*4)	Unconditional-Frames	Unconditional frames belonging to a Sporadic frame

An Unconditional frame has the following structure (four bytes):

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

Similar to the Frames list for the Unconditional frames of a LIN Description File (LDF file), there is possibly a Sporadic_frames list for Sporadic frames.

SporadicFrameListIndex is the index in the Sporadic_frames list.

4.3.30 0x65 LIN Delete Sporadic Frame

Delete a Sporadic frame defined by 0x64 LIN Sporadic Frame Definition and the belonging Unconditional frames by this command.

Command:

Byte	Indication	Meaning
0	SporadicFrame-ListIndex	List index (starting with 0), see also the 0x22 LIN Fill Schedule Table command
1..3	reserved	Reserved

4.3.31 0x66 LIN Send Sporadic Frame

This command allows sending in the Sporadic frame Slot (regarding Frame Slot see also Figure 4-2).

The command is ONLY available for a LIN master.

Command:

Byte	Indication	Meaning
0	SporadicFrame-ListIndex	List index (starting with 0), see also 0x22 LIN Fill Schedule Table command
1	Mode	0: Deactivates sending 1: Activates sending (the header with UnconditionalFrameId is only sent ONCE)
2	Unconditional-FrameId	Identifier (0x00..0x3F) of the Unconditional frame belonging to this Sporadic frame
3	reserved	Reserved

The Unconditional frame specified by UnconditionalFrameId must be defined by the 0x23 LIN Fill Frame Response Table or 0x30 LIN Frame Response Definition commands same as each normal Unconditional frame.

4.3.32 0x67 LIN Sporadic Frame Control

This command is used to control a LIN Sporadic frame. 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 (as far as more than four bytes exist).

Command and Response:

Byte	Indication	Meaning
0	SporadicFrame-ListIndex	List index (starting with 0), see also the 0x22 LIN Fill Schedule Table command
1	Mode	0: Query of the state of an Unconditional frame within a Sporadic frame
2, 3	reserved	Reserved

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

Byte	Indication	Meaning
4	Unconditional-FrameId	Identifier of the Unconditional frame belonging to this Sporadic frame (0x00..0x3F)
5..7	reserved	Reserved

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

Byte	Indication	Meaning
4..7	Flags	Bit 0: TxPending (the Unconditional frame is still sent) Bits 1..31: Reserved
8	Unconditional-FrameId	Identifier of the Unconditional frame belonging this Sporadic frame (0x00..0x3F)
9..11	reserved	Reserved

4.3.33 0x68 LIN Event Triggered Frame Definition

Define an Event triggered frame with the belonging Unconditional frames by this command.

Execute the command on a master and on all slaves intended to send an Event triggered frame.

Command:

Byte	Indication	Meaning
0	EventTriggered-FrameListIndex	List index (starting with 0), see also the 0x22 LIN Fill Schedule Table command
1..3	reserved	Reserved
4	EventTriggered-Frame.Id	Identifier of the Event triggered frame (0x00..0x3F)
5	EventTriggered-Frame.Dlc	Data length of the Event triggered frame (0..8)
6	EventTriggered-Frame.Flags	Bit 0: IgnoreDlc – Ignoring of the data length Bits 1..7: Reserved
7	EventTriggered-Frame.reserved	Reserved
8, 9	reserved	Reserved
10, 11	NumberOf-Unconditional-Frames	Number of Unconditional frames (N) belonging to an Event triggered frame
12.. 11+(N*4)	Unconditional-Frames	The Unconditional Frames belonging to an Event triggered frame

An Unconditional frame has the following structure (four bytes):

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

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.

EventTriggeredFrameListIndex is the index in the Event_triggered_frames list.

With each reception of a LIN Frame response the master checks its content and starts sending the Unconditional frames belonging to this Event triggered frame in the case at least one of the following conditions is met:

- A transmission error occurred (e.g. wrong checksum).
- The first data byte does not include any of the Unconditional frames identifiers.
- The IgnoreDlc flag is not set, and the data length of the received LIN Frame response does not correspond to the data length configured by EventTriggeredFrame.Dlc.

4.3.34 0x69 LIN Delete Event Triggered Frame

Delete an Event triggered frame defined by 0x68 LIN Event Triggered Frame Definition and the belonging Unconditional frames by this command.

Command:

Byte	Indication	Meaning
0	EventTriggered-FrameListIndex	List index (starting with 0), see also the 0x22 LIN Fill Schedule Table command
1..3	reserved	Reserved

4.3.35 0x6A LIN Send Event Triggered Frame

This command allows sending in the Event triggered frame slot (Regarding Frame Slot see also Figure 4-2).

Command:

Byte	Indication	Meaning
0	EventTriggered-FrameListIndex	List index (starting with 0), see also the 0x22 LIN Fill Schedule Table command
1	Mode	0: Deactivates sending 1: Activates sending as long as the LIN Frame response specified by UnconditionalFrameId has been sent successful and fault-free on the bus
2	Unconditional-FrameId	Identifier (0x00..0x3F) of the Unconditional frame belonging to this Event triggered frame
3	reserved	Reserved

The Unconditional frame specified by UnconditionalFrameId must be defined by the 0x23 LIN Fill Frame Response Table or 0x30 LIN Frame Response Definition commands same as each normal Unconditional frame.

4.3.36 0x6B LIN Event Triggered Frame Control

This command is used to control a LIN Event triggered frame. 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 (as far as more than four bytes exist)

Command and Response:

Byte	Indication	Meaning
0	EventTriggered-FrameListIndex	List index (starting with 0), see also the 0x22 LIN Fill Schedule Table command
1	Mode	0: Query of the state of an Unconditional frame within a Sporadic frame 1: Setting the CollisionResolvingScheduleTable. This Schedule table is proceeded in case of a collision to resolve the collision.
2, 3	reserved	Reserved

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

Byte	Indication	Meaning
4	Unconditional-FrameId	Identifier of the Unconditional frame belonging to this Event triggered frame (0x00..0x3F)
5..7	reserved	Reserved

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

Byte	Indication	Meaning
4..7	Flags	Bit 0: TxPending (the Unconditional Frame is still sent, in the Event triggered frame slot or in the Unconditional frame slot) Bits 1..31: Reserved
8	Unconditional-FrameId	Identifier of the Unconditional frame belonging to this Event triggered frame (0x00..0x3F)
9..11	reserved	Reserved

The following Command parameters are only valid for Mode = 1:

Byte	Indication	Meaning
4	ScheduleTableNumber	Number of the Schedule Table used as CollisionResolvingScheduleTable
5..7	reserved	Reserved

4.3.37 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 diagnostics.

Command:

Byte	Indication	Meaning
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 4-2) 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 baud rate of 19,200 Baud, this delay is 10ms.
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	Meaning
4, 5	reserved	Reserved
6, 7	P2max	P2max timeout in milliseconds (maximum time between the end of the request and the beginning of the response, e.g. 200ms)
8, 9	P3max	P3max timeout in milliseconds (maximum time between the end of the request and the beginning of the response during ResponsePending, e.g. 5,100ms)
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	DefaultMaster-Data.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. 1,000ms
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	Meaning
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 frames in the case data does not coincide with 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 comparison of the result with Data.

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

Byte	Indication	Meaning
4	NAD	Control unit address (**N**ODE **AD**DRESS)
5	reserved	Reserved
6, 7	P2max	P2max timeout in milliseconds (maximum time between the end of the request and the beginning of the response, e.g. 200ms)
8, 9	P3max	P3max timeout in milliseconds (maximum time between the end of the request and the beginning of the response during ResponsePending, e.g. 5,100ms)
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. 1,000ms
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 diagnostic 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 LIN diagnostics:

- Select the Type of diagnostics 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)

4.3.38 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	Meaning
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 highest valid 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

4.3.39 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. 1,100 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	Meaning
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 highest valid 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.

4.3.40 0xA3 LIN Diagnostics – Get Response Buffer

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	Meaning
0	Channel	Multisession channel (starting with 0)
1..3	reserved	Reserved

Response:

Byte	Indication	Meaning
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 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 established 3: Connection has been 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

4.3.41 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 diagnostics, call the 0xA0 LIN Diagnostics – Configuration command again with Type = 0.

Command:

Byte	Indication	Meaning
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 highest valid 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

4.3.42 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 Diagnostic – 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	Meaning
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	Meaning
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 established 3: Connection has been 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

4.3.43 0xA8 LIN Diagnostics – Change Timing

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

Command:

Byte	Indication	Meaning
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 1,000ms)
2, 3	reserved	Reserved

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

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

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

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

4.3.44 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 (as far as more than four bytes exist).

Command and Response:

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

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

Byte	Indication	Meaning
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	Meaning
4..7	reserved	Reserved

4.3.45 0xF2 LIN Monitor – Get small Buffer Items

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

Response:

Byte	Indication	Meaning
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	Meaning
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 overrun
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 400ns
16..19	BitTimeX8	Bit time measured over eight bit times as multiples of 25ns

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]).

4.3.46 0xF4 LIN Monitor – Get small List Item

This command is to query a small LIN monitor list item of the LIN message indicated by Id.

Command:

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

Response:

Byte	Indication	Meaning
0	Id	Identifier (0x00..0x3F)
1..3	reserved	Reserved
4..7	FrameCounter	Frame counter
8	Flags	Bit 0: Identifier parity error Bit 1: Checksum error Bit 2: Checksum error Bit 3: Bit error Bits 4..5: Reserved Bit 6: Sent LIN Frame response (TX) Bit 7: Reserved
9	Length	Data length including checksum (0..9)
10	IdCode	Identifier byte (consisting of identifier and + parity bits)
11..19	Data	Data bytes and checksum
20..23	StartTime	Start time stamp as multiples of 400ns
24..27	BitTimeX8	Bit time measured over eight bit times as multiples of 25ns

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]).

4.4 KLine Commands

The KLine commands for your GOEPEL Hardware are described in this chapter.

General notes valid for all Firmware commands
can be found in section General Firmware Notes.

Initial state:

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

The term Initialization used in this KLine 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 KLine. That means that each request of the tester (here, KLine 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 KLine 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.

4.4.1 0x12 KLine Init Interface

This command resets the selected KLine 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	Meaning
0..2	reserved	Reserved
3	BlinkMode	0: Blinking of the LEDs deactivated (default) 1: Blinking of the LEDs activated

4.4.2 0x20 KLine FIFO

This command is intended to configure and control the KLine in FIFO operation (FIFO = **F**irst **I**n **F**irst **O**ut).
For this, there are the Tx FIFO for sending and the Rx IFO for receiving.

Warning

When the KLINE is in FIFO operation, no diagnostic commands like 0xA0 KLine Diagnostics – Configuration can be executed.

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 (as far as more than four bytes exist).

Command and Response:

Byte	Indication	Meaning
0	SubCmd	0: RESET 1: INIT 2: WRITE_TO_TX_FIFO 3: READ_FROM_RX_FIFO 4: GET_TX_FIFO_STATE 5: GET_RX_FIFO_STATE
1..3	reserved	Reserved

4.4.2.1 Reset

The SubCmd = RESET subcommand is used to completely reset the FIFO functionality. Then, diagnostic commands like 0xA0 KLine Diagnostics – Configuration can be executed again.

This subcommand does not have any further command and response bytes.

4.4.2.2 Init

The SubCmd = INIT subcommand is used for the initialization of the FIFO functionality.

Command:

Byte	Indication	Meaning
4..7	Flags	Bits 0.. 31: Reserved
8..11	BaudRate	Baudrate in Hertz (5 to 1,000,000)
12..15	TxBufferSize	Sending buffer size in bytes (0 = Using the Default Sending buffer size)
16..19	RxBufferSize	Reception buffer size in bytes (0 = Using the Default Reception buffer size)
20	UartMode	UART Mode (UART = **U**niversal **A**synchronous **R**eceiver **T**ransmitter) 0: 8 bit data 1: 7 bit data, 1 bit parity 2: 8 bit data, 1 bit parity
21	Parity	Parity 0: even parity (even) 1: odd parity (odd)
22	StopBits	Stop bit mode 0: 1 Stop bit 1: 2 Stop bits
23	Reserved	Reserved

Response:

Byte	Indication	Meaning
4..7	DesiredBaudRate	Desired baudrate in Hertz (corresponds to the BaudRate of the command)
8..11	ActualBaudRate	Baudrate in Hertz really used

4.4.2.3 Write to Tx FIFO

The SubCmd = WRITE_TO_TX_FIFO subcommand is used to write data to the Tx FIFO.

Command:

Byte	Indication	Meaning
4..7	NumberOfBytes	Number of data bytes to be written
8.. (7+NumberOfBytes)	Data	Data bytes

4.4.2.4 Read from Rx FIFO

The SubCmd = READ_FROM_RX_FIFO subcommand is used to read data from the Rx FIFO.

Response:

Byte	Indication	Meaning
4..7	Flags	Bit 0: FifoNotEmpty = There are still data bytes in the Rx FIFO (i.e. after reading out the Rx FIFO by this command) Bit 1: FifoOverrun = Between the last and the current reading out the Rx FIFO, one or more data bytes got lost as the Rx FIFO was completely full Bits 2.. 31: Reserved
8..11	NumberOfBytes	Number of data bytes
12.. (11+NumberOfBytes)	Data	Data bytes

4.4.2.5 Get Tx FIFO State

The SubCmd = GET_TX_FIFO_STATE subcommand is used to query the Tx FIFO state.

Response:

Byte	Indication	Meaning
4..7	Flags	Bit 0: FifoIsEmpty = The Tx FIFO is empty Bit 1: FifoIsFull = The Tx FIFO is full
8..11	FifoSize	Size of the Tx FIFO in bytes
12..15	FifoFillingLevel	Filling level of the Tx FIFO in bytes

4.4.2.6 Get Rx FIFO State

The SubCmd = GET_RX_FIFO_STATE subcommand is used to query the Rx FIFO state.

Response:

Byte	Indication	Meaning
4..7	Flags	Bit 0: FifoIsEmpty = The Rx FIFO is empty Bit 1: FifoIsFull = The Rx FIFO is full Bit 2: FifoOverrun = One or more received data bytes got lost as the Rx FIFO was completely full
8..11	FifoSize	Size of the Rx FIFO in bytes
12..15	FifoFillingLevel	Filling level of the Rx FIFO in bytes

4.4.3 0x21 KLine Node

This command is used for the configuration and control of the KLine interface.

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 (as far as more than four bytes exist).

Command and Response:

Byte	Indication	Meaning
0	SubCmd	0: PROPERTY__SET_BY_ID 1: PROPERTY__GET_BY_ID
1..3	reserved	Reserved

PropertyId:

Wert	Meaning
0	**ENABLE_RX_TX_SPLIT** 0 = Deactivation of the disconnection of Rx from Tx 1 = Activation of the disconnection of Rx from Tx This allows full duplex operation in case the corresponding transceiver is available (as for example for RS232) with one Rx and one Tx line each. Warning: The “normal” KLine transceiver does not support full duplex operation.
1	**ENABLE_BUS_TERMINATION** 0 = Deactivation of the bus termination (pull up resistor) 1 = Activation of the bus termination (pull up resistor)
2	**ENABLE_INTERNAL_VBAT** 0 = Deactivation of the internal voltage supply for the transceiver 1 = Activation of the internal voltage supply for the transceiver Then, connecting an external voltage is not applicable.

4.4.3.1 Property SetById

The SubCmd = PROPERTY__SET_BY_ID subcommand is used to set a property specified by PropertyId.

Command:

Byte	Indication	Meaning
4, 5	Id	PropertyId: Property identifier (see PropertyId in section 0x21 KLine Node)
6, 7	Reserved	Reserved
8..11	Value	Value of the property

4.4.3.2 *Property GetByID*

The SubCmd = GET_FLAG_BY_ID subcommand is used to query for the value of a property specified by PropertyId.

Command:

Byte	Indication	Meaning
4, 5	Id	PropertyId: Property identifier (see PropertyId in section 0x21 KLine Node)
6, 7	reserved	Reserved

Response:

Byte	Indication	Meaning
4, 5	Id	PropertyId: Property identifier (see PropertyId in section 0x21 KLine Node)
6, 7	Reserved	Reserved
8..11	Value	Value of the property

4.4.4 0x54 KLine Monitor

This command is used for the configuration and control of the KLine Monitor.

Warning

For using the monitor, the FIFO functionality has to be activated (see 0x20 KLine FIFO) or a diagnostic protocol has to be selected (see 0xA0 KLine Diagnostics – Configuration).
This way the right Baudrate value, among others, is set.

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 (as far as more than four bytes exist).

Command and Response:

Byte	Indication	Meaning
0	SubCmd	0: RESET 1: CONFIG__NORMAL 3: START 4: STOP 5: READ__NORMAL
1..3	reserved	Reserved

4.4.4.1 Reset

The SubCmd = RESET subcommand is used for the complete reset and deactivation of the monitor.

This subcommand does not have any further command and response bytes.

4.4.4.2 Config Normal

The SubCmd = CONFIG__NORMAL subcommand is used for configuring the KLine monitor with normal monitor entries.

Command:

Byte	Indication	Meaning
4..7	Flags	Bit 0: AutomaticEmpty = Received monitor entries are sent to the host automatically; The responses have the same structure as the responses to the Read Normal subcommand Bits 1.. 31: Reserved
8..11	BufferSize	Monitor buffer size as number of monitor entries

4.4.4.3 Start

The SubCmd = START subcommand is used to start the monitor. For this, the monitor must have been already configured (see Config Normal).

This subcommand does not have any further command and response bytes.

4.4.4.4 Stop

The SubCmd = STOP subcommand is used to stop the monitor.

This subcommand does not have any further command and response bytes.

4.4.4.5 Read Normal

The SubCmd = READ_NORMAL subcommand is used to read normal monitor entries.

Response:

Byte	Indication	Meaning
4..7	NumberOfItems	Number of normal monitor entries
8..	Items	Normal monitor entries

Structure of an individual normal monitor entry:

Byte	Indication	Meaning
0..3	Timestamp	Time stamp with TimestampResolution resolution
4	TimestampResolution	Time stamp resolution 0: 400 nanoseconds
5	Flags	Bit 0: Reserved Bit 1: Tx = Sent data byte Bit 2: Collision = Collision by sending and receiving a data byte at the same time Bits 3..6: Reserved Bit 7: BufferOverrun = Buffer overrun
6	Data	Data byte
7	reserved	Reserved

4.4.5 0xA0 KLine Diagnostics – Configuration

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

All settings required for a diagnostic construction and flow are preset by this command.
The settings made remain active till new settings will be made explicitly. This command is the precondition for the 0xA1 KLine Diagnostics – Start Session command of a KLine Diagnostics.

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	Meaning
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	Meaning
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. 0ms
14, 15	P1max	Maximum interbyte time for responses of the ECU, e.g. 20ms
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. 25ms
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. 50ms
20, 21	P3min	Minimum time gap between the response of the ECU and a new request of the tester, e.g. 55ms
22, 23	P3max	Maximum time gap between the response of the ECU and a new request of the tester, e.g. 2,000ms
24, 25	P4min	Minimum interbyte time for requests of the tester, e.g. 5ms
26, 27	P4max	Maximum interbyte time for requests of the tester, e.g. 20ms
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.Response RequiredParameter	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	Meaning
40, 41	reserved	Reserved
42, 43	TargetAddress	5 Baud address of the ECU
44, 45	W1min	Minimum time gap between the end of the address byte and the start of the synchronization pattern, e.g. 60ms
46, 47	W1max	Maximum time gap between the end of the address byte and the start of the synchronization pattern, e.g. 300ms
48, 49	W2min	Minimum time gap between the end of the synchronization pattern and the start of ***Keybyte 1***, e.g. 5ms
50, 51	W2max	Maximum time gap between the end of the synchronization pattern and the start of ***Keybyte 1***, e.g. 20ms
52, 53	W3min	Minimum time gap between ***Keybyte 1*** and ***Keybyte 2***, e.g. 0ms
54, 55	W3max	Maximum time gap between ***Keybyte 1*** and ***Keybyte 2***, e.g. 20ms
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. 25ms
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. 50ms
60, 61	W5min	Minimum time gap before the tester starts to send the address byte, e.g. 300ms
62, 63	W5max	Maximum time gap before the tester starts to send the address byte, e.g. 300ms
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	Meaning
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. 300ms
46, 47	W5max	Maximum time gap before the tester starts to send the address byte, e.g. 300ms
48, 49	TWuPmin	Minimum time period for the Wake up Pattern, e.g. 50ms
50, 51	TWuPmax	Maximum time period for the Wake up Pattern, e.g. 50ms
52, 53	TIniLmin	Minimum time period for the Low phase of the Wake up Pattern, e.g. 25ms
54, 55	TIniLmax	Maximum time period for the Low phase of the Wake up Pattern, e.g. 25ms
The following parameters are valid till completing initialization, i.e., the time behavior of the protocol during initialization can be defined separately via these parameters.		
56, 57	P1min	Minimum interbyte time for responses of the ECU, e.g. 0ms
58, 59	P1max	Maximum interbyte time for responses of the ECU, e.g. 20ms
60, 61	P2min	Minimum time gap between the request of the tester and the response of the ECU, e.g. 25ms
62, 63	P2max	Maximum time gap between the request of the tester and the response of the ECU, e.g. 50ms
64, 65	P3min	Minimum time gap between the response of the ECU and a new request of the tester, e.g. 55ms
66, 67	P3max	Maximum time gap between the response of the ECU and a new request of the tester, e.g. 2,000ms
68, 69	P4min	Minimum interbyte time for requests of the tester, e.g. 5ms
70, 71	P4max	Maximum interbyte time for requests of the tester, e.g. 20ms
72, 73	BaudRate	Baud rate (generally 10.400 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	Flags	Bit 0 = 0: No separate length byte in the header of the frame Bit 0 = 1: Separate length byte in the header of the frame Bit 1 = 0: TargetByte and SourceByte in the header of the frame Bit 1 = 1: No AddressBytes in the header of the frame Bits 2..7: Reserved
83	reserved	Reserved

Parameterization of Keyword Protocol 1281 (KWP1281):

Byte	Indication	Meaning
8..11	reserved	Reserved
12, 13	t7min	Minimum time gap between the bytes within one block, e.g. 1ms
14, 15	t7max	Maximum time gap between the bytes within one block, e.g. 55ms
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. 1ms
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. 200ms
20, 21	t9min	Minimum time gap between the end of a block and the start of the next block, e.g. 1ms
22, 23	t9max	Maximum time gap between the end of a block and the start of the next block, e.g. 500ms
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 ECU
44, 45	t0min	Minimum idle line before the start of initialization, e.g. 60ms
46, 47	t0max	Maximum idle line before the start of initialization, e.g. 300ms
48, 49	t1min	Minimum time gap between the correct initialization and the start of the synchronous byte, e.g. 80ms
50, 51	t1max	Maximum time gap between the correct initialization and the start of the synchronous byte, e.g. 210ms
52, 53	t2min	Minimum time gap between the synchronous byte and ***Keybyte 1***, e.g. 5ms
54, 55	t2max	Maximum time gap between the synchronous byte and ***Keybyte 1***, e.g. 20ms
56, 57	t3min	Minimum time gap between ***Keybyte 1*** and ***Keybyte 2***, e.g. 1ms
58, 59	t3max	Maximum time gap between ***Keybyte 1*** and ***Keybyte 2***, e.g. 20ms
60, 61	t4min	Minimal time gap between ***Keybyte 2*** and complement of ***Keybyte 2***, e.g. 25ms
62, 63	t4max	Maximum time gap between ***Keybyte 2*** and complement of ***Keybyte 2***, e.g. 5ms
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. 240ms
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. 1,000ms
68, 69	t6min	Minimum time gap between the complement of ***Keybyte 2*** and the start of the ECU identification, e.g. 25ms
70, 71	t6max	Maximum time gap between the complement of ***Keybyte 2*** and the start of the ECU identification, e.g. 50ms

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

Parameterization of ISO-9141-Ford:

Byte	Indication	Meaning
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. 0ms
14, 15	ReceiveInterByte-GapMax	***Tester Reception: Interbyte Gap Time max*** Maximum interbyte time for responses of the ECU, e.g. 22ms
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. 0ms
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. 50ms
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. 55ms
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. 2,000ms
24, 25	TransmitInterByte-GapMin	***Tester Transmissions: Interbyte Gap Time min*** Minimum interbyte time for requests of the tester, e.g. 6ms
26, 27	TransmitInterByte-GapMax	***Tester Transmissions: Interbyte Gap Time max*** Maximum interbyte time for requests of the tester, e.g. 6ms
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)

4.4.6 0xA1 KLine Diagnostics – Start Session

This command starts a KLine 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 diagnostic 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 KLine (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	Meaning
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 highest valid bit is set (0x80)
2, 3	Length	Request length (At present, 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 KLine protocols at first sight on the part of the host. Within the KLine 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 KLine 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 KLine.
To simplify it one can say that “Start communication” on the KLine is to be considered identical to “Start diagnostics”.

KWP2000 (Fast initialization)	
Description	Response of KLine 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 KLine 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 KLine 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 (KLine 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 KLine driver generates a response identical to “KWP2000 (Fast initialization) from the ***Keybytes*** received within the initialization. Thus, there are not any differences (on the part of the host) to fast initialization.

KWP1281 (slow initialization – 5 Baud)	
Description	Response of KLine driver
Similar to KWP2000 (slow initialization), 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 KLine 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 KLine driver
The tester (KLine driver) generates and transmits a “normal” request (***Mode*** = 0x10 – ***Diagnostic Mode Entry***) at 10,400 Bauds and with the preset address parameters on the KLine. 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.

4.4.7 0xA2 KLine Diagnostics – Send Request

This command is used to send a KLine 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	Meaning
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 highest valid 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)) (For Length = 0 no request is sent)
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. 1,100 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 for 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 has been successfully opened, yet. "**`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 Checksum

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

Command Header with *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 KLine 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 has been 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.

4.4.8 0xA3 KLine Diagnostics – Get Response Buffer

Query the KLine 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	Meaning
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	Meaning
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: BufferEmpty (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 established 3: Connection has been 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 Diagnostics – Stop Session can have different meanings depending on the protocol used.

The KLine protocol driver is NOT responsible for the interpretation of these responses!

4.4.9 0xA4 KLine Diagnostics – Stop Session

This command stops a running KLine 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 the next 0xA1 KLine Diagnostics – Start Session command.

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

Command:

Byte	Indication	Meaning
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 highest valid bit is set (0x80)
2, 3	Length	Request length (At present, only Length = 0 is supported, the corresponding stop service is transmitted according to the diagnostic type)
4.. (3+Length)	Request	Request, consisting of SID (service identifier) and data

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

Within the KLine driver, however, specific actions matched to the individually active protocol are released.

All protocols react in a different way. Generally one can say:
The KLine 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 diagnostics and the transport protocols does not exist on the KLine. To simplify it one can say that “Stop communication” on the KLine is to be considered identical to “Stop diagnostics”.

KWP2000	
Description	Response of the KLine driver
The KLine 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 KLine driver
The KLine 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 acknowledgment.	Data division (***BT***) of the ***Acknowledge*** block (always, even if there is no block of the ECU)

ISO-9141-Ford	
Description	Response of the KLine driver
The KLine 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

4.4.10 0xA5 KLine Diagnostics – Get State

Query the KLine 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	Meaning
0	Channel	Multisession channel (starting with 0)
1	ResetLastError	0 = Do not reset LastErrorCode 1 = Reset LastErrorCode
2, 3	reserved	Reserved

Response:

Byte	Indication	Meaning
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 established 3: Connection has been 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