Product Specification

smartCAR

with extended Command Set
USB 2.0 Stand-alone Device for
CAN, LIN or K-LINE   Interface

User Manual Version 1.2

GÖPEL electronic

Get the total Coverage!

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

The content of the manual is subject to change without prior notice and is supplied for information only.
Hardware and software might be modified also without prior notice due to technical progress.

In case of inaccuracies or errors appearing in this manual, GOEPEL electronic GmbH assumes no liability or responsibility.

GOEPEL electronic GmbH is neither liable for direct damages nor consequential damages from the company's product applications.

Printed: 05.12.2011

Issue: December 2011

1 Installation

1.1 Hardware Installation

Generally hardware installation for smartCAR means exchanging the transceiver modules.

Warning

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

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

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

1.2 Driver Installation

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

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

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

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

After driver installation, you can check whether the device is properly embedded by the system.
The following figure shows the successful embedding of a smartCAR:

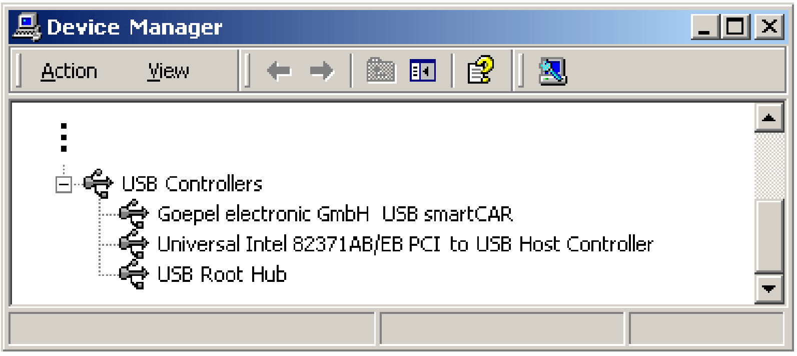
Device Manager
Action
View
⋮
USB Controllers
Goepel electronic GmbH USB smartCAR
Universal Intel 82371AB/EB PCI to USB Host Controller
USB Root Hub

Figure 1-1: Display of Device Manager

ℹ Please note that the Device Manager shows ALL USB controllers.

2 Hardware

2.1 Definition

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

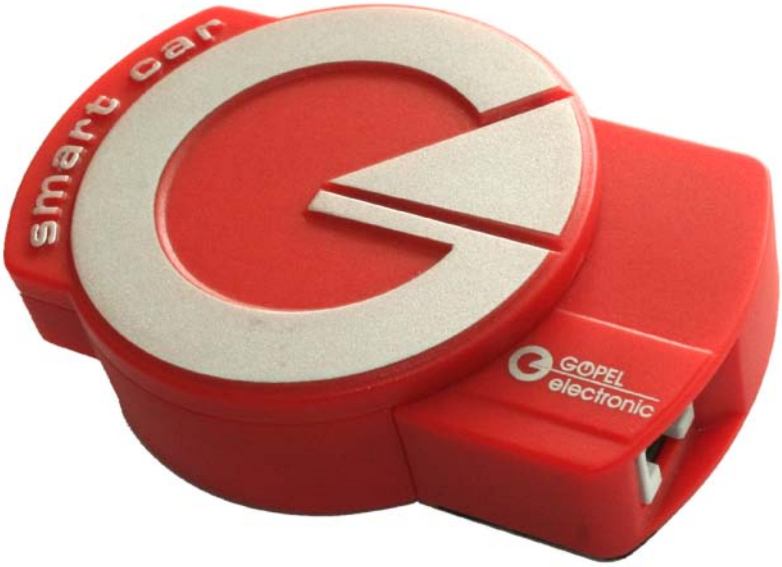

Figure 2-1: smartCAR

smartCAR offers the following resources:

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

Warning

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

2.2 Technical Specification

2.2.1 Dimensions

The dimensions of your smartCAR are given as follows (width x height x depth):

- 75 mm x 25 mm x 110 mm

2.2.2 smartCAR Characteristics

The smartCAR characteristics are shown in this table:

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

2.3 Construction

2.3.1 General

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

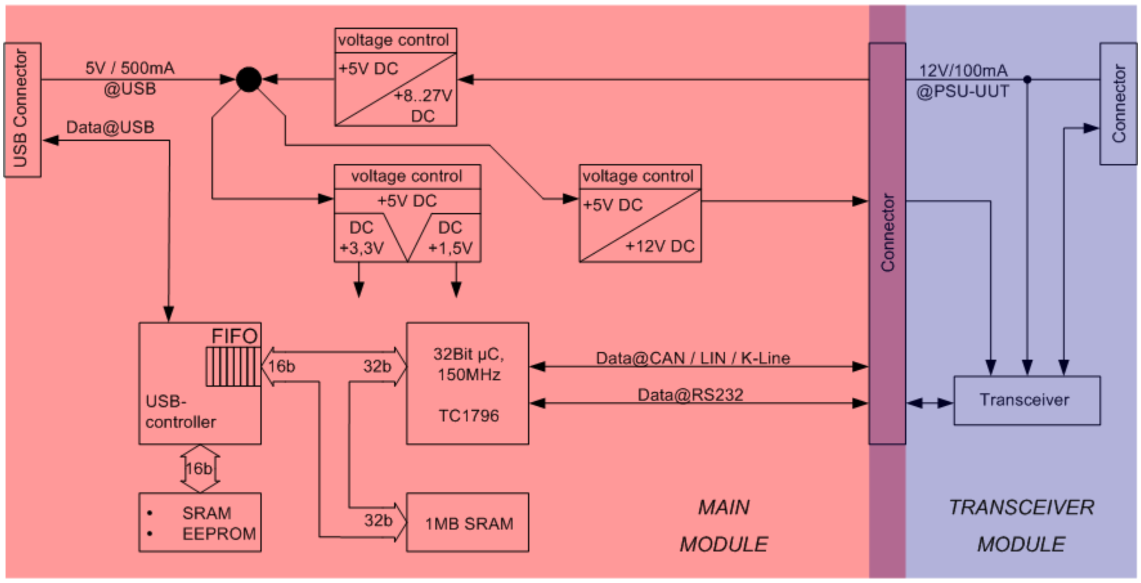

Figure 2-2: Block diagram of a smartCAR device

Warning

Please use only the delivered USB cable to connect your smartCAR device to the PC's USB interface.

Other cables may be inapplicable.

2.3.2 Addressing

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

The device with the LEAST serial number is always the device with the number 1.

Tip

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

2.3.3 Change of Transceivers

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

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

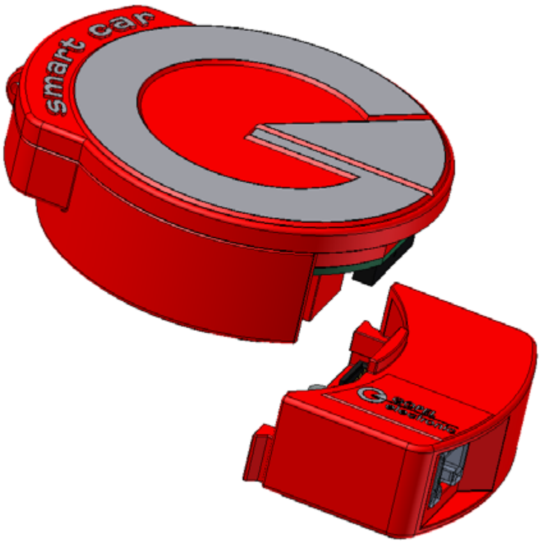

Figure 2-3
Change of Transceiver module

2.3.4 Communication Interfaces

2 x CAN-Interface Version 2.0b:

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

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

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

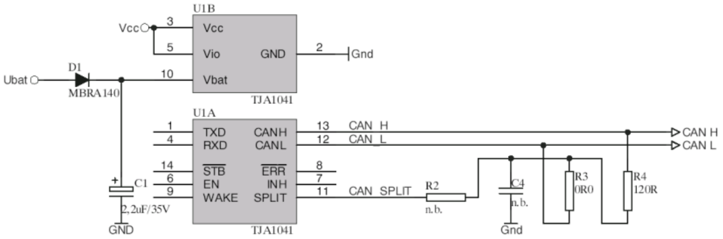

Figure 2-4:
CAN interface

K-Line Interface (ISO 9141)

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

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

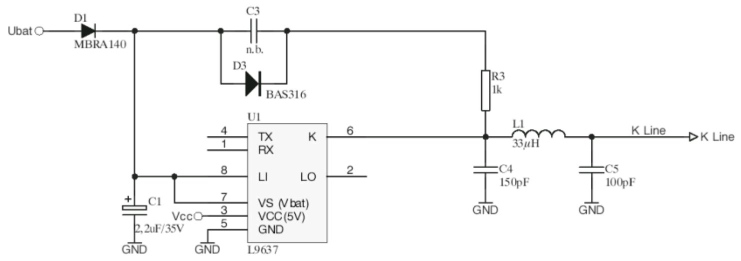

Abbildung 2-5
K-Line interface

LIN-Interface Version 2.0:

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

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

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

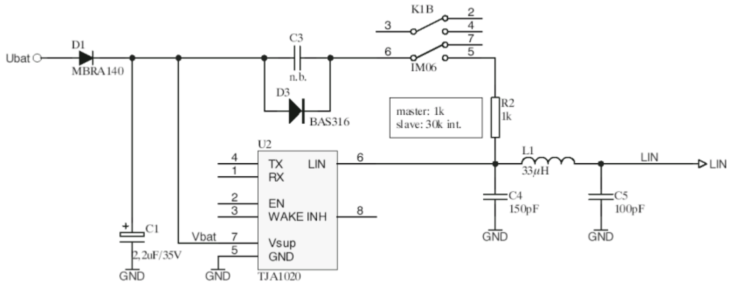

Figure 2-6:
LIN interface

2.3.5 Connector Assignments

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

Communication Interface

Type: RJ45 female

The assignments are shown in the following table:

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

Type: DSub 9 poles male (at the cable)

The assignments are shown in the following table:

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

USB Interface

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

2.4 Delivery Notes

A smartCAR delivery includes at least

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

At present the following types of Transceiver modules are available:

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

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

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

3 Control Software

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

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

3.1 Programming via G-API

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

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

3.2 Programming via DLL Functions

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

We would be pleased to send the GOEPEL Firmware documentation to you on your request. Please get in touch with our sales department in case you need that.

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

For the used structures, data types and error codes refer to the headers – you find the corresponding files on the supplied CD.

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

On the other hand, USB Controller means ALWAYS the controller providing the USB 2.0 interface of the smartCAR device.

3.2.1 Windows Device Driver

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

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

3.2.1.1 Driver_Info

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

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

Format:

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

Parameters:

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

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

Description:

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

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

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

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

3.2.1.2 DLL_Info

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

Format:

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

Parameters

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

Description:

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

Example:

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

3.2.1.3 *Write_FIFO*

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

Format:

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

Parameters:

DeviceName

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

DeviceNumber

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

Pointer, for example pWrite
to the write data area

DataLength

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

Description:

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

3.2.1.4 *Read_FIFO*

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

Format:

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

Parameters:

DeviceName

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

DeviceNumber

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

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

DataLength

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

After function execution: Number of bytes actually read

Description:

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

3.2.1.5 Read_ FIFO_Timeout

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

Format:

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

Parameters:

DeviceName

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

DeviceNumber

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

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

DataLength

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

After function execution: Number of bytes actually read

Timeout

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

Description:

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

3.2.1.6 Write_ COMMAND

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

Format:

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

Parameters:

DeviceName

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

DeviceNumber

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

Pointer, for example pWrite
to the write data area

DataLength

Size of the storage area pWrite is pointing to, in bytes
See also USB Controller Control Commands
(currently max. 64 bytes per command)

Description:

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

3.2.1.7 Read_ COMMAND

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

Format:

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

Parameters:

DeviceName

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

DeviceNumber

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

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

DataLength

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

After function execution: Number of bytes actually read

Description:

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

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

3.3 Programming with LabVIEW

3.3.1 LabVIEW via G-API

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

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

3.3.2 LLB using the Windows Device Driver

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

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

3.4 Further GOEPEL Software

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

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

3.5 USB Controller Control Commands

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

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

3.5.1 USB Command Structure

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

The header of a USB command has the following structure:

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

3.5.2 USB Response Structure

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

The header of a USB response has the following structure:

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

3.5.3 USB Commands

At present there is only the READ_SW_VERSION USB command available.

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