FEIG ELECTRONIC

MANUAL

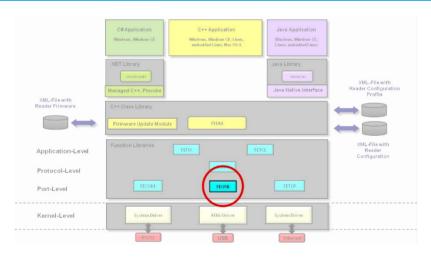# ID FEUSB

Software Support for USB (Universal Serial Bus)

Version 6.00.00 (Windows)

Version 4.02.06 (Windows CE)

Version 6.00.00 (Linux)

Operating System	Target		Notes
	32-Bit	64-Bit	
Windows Vista / 7 / 8 / 10	X	X	
Windows CE 6	X	-	On request
Linux	X	X	
Android	X	X	On request
Apple Max OS X	-	X	On request

Note

Licensing Agreement for Use of the Software

This is an agreement between you and FEIG ELECTRONIC GmbH (hereafter "FEIG") for use of the ID FEUSB program library and the present documentation, hereafter called licensing material. By installing and using the software you agree to all terms and conditions of this agreement without exception and without limitation. If you are not or not completely in agreement with the terms and conditions, you may not install the licensing material or use it in any way. This licensing material remains the property of FEIG ELECTRONIC GmbH and is protected by international copyright.

§1 Object and Scope of the Agreement

1. FEIG grants you the right to install the licensing material provided and to use it under the following conditions.

2. You may install all components of the licensing material on a hard disk or other storage medium. The installation and use may also be done on a network fileserver. You may create backup copies of the licensing material.

3. FEIG grants you the right to use the documented program library for developing your own application programs or program libraries, and you may sell the runtime file FEUSB.DLL, FEUSBCE.DLL, LIBFEUSB.x.y.z.DYLIB[1] or LIBFEUSB.SO.x.y.z[1] without licensing fees under the stipulation that these application programs or program libraries are used only together with USB devices developed by FEIG.

4. FEIG does not grant you the right to sell the USB driver files for Windows (OBIDUSB.SYS, OBIDUSB9.SYS, OBIDUSB.INF) supplied with the USB devices separately. These USB driver files are to be sold only in conjunction with FEIG USB devices

§2. Protection of the Licensing Material

1. The licensing material is the intellectual property of FEIG and its suppliers. It is protected in accordance with copyright, international agreements and relevant national statutes where it is used. The structure, organization and code of the software are a valuable business secret and confidential information of FEIG and its suppliers.

2. You agree not to change, modify, translate, reverse develop, decompile, disassemble the program library or the documentation or in any way attempt to discover the source code of this software.

3. To the extent that FEIG has applied protection marks, such as copyright marks and other legal restrictions in the licensing material, you agree to keep these unchanged and to use them unchanged in all complete or partial copies which you make.

4. The transmission of licensing material in part or in full is prohibited unless there is an explicit agreement to the contrary between you and FEIG. Application programs or program libraries which are created and sold in accordance with §1 Par. 3 of this Agreement are excepted.

§3 Warranty and Liability Limitations

1. You agree with FEIG that it is not possible to develop EDP programs such that they are error-free for all application conditions. FEIG explicitly makes you aware that the installation of a new program can affect already existing software, including such software that does not run at the same time as the new software. FEIG assumes no liability for direct or indirect damages, for consequential damages or special damages, including lost profits or lost savings. If you want to ensure that no already installed program will be affected, you should not install the present software.

2. FEIG explicitly notes that this software makes it possible for irreversible settings and adaptations to be made on devices which could destroy these devices or render them unusable. FEIG assumes no liability for such actions, regardless of whether they are carried out intentionally or unintentionally.

3. FEIG provides the software „as is" and without any warranty. FEIG cannot guarantee the performance or the results you obtain from using the software. FEIG assumes no liability or guarantee that the protection rights of third parties are not violated, nor that the software is suitable for a particular purpose.

4. FEIG calls explicit attention to the fact that the licensed material is not designed with components and testing for a level of reliability suitable for use in or in connection with surgical implants or as critical components in any life support systems whose failure to perform can reasonably be expected to cause significant injury to human health.
To avoid damage, injury, or death, the user or application designer must take reasonably prudent steps to protect against system failures.

---

[1] x.y.z represents the actual version number

§4 Concluding Provisions

1. This Agreement contains the complete licensing terms and conditions and supercedes any prior agreements and terms. Changes and additions must be made in writing.

2. If any provision this agreement is declared to be void, or if for any reason is declared to be invalid or of no effect, the remaining provisions shall be in no manner affected thereby but shall remain in full force and effect. Both parties agree to replace the invalid provision with one which comes closest to its original intention.

3. This agreement is subject to the laws of the Federal Republic of Germany. Place of jurisdiction is Weilburg.

Content

1. Introduction

The ID FEUSB support package is used to assist in programming communication-oriented software with data transport over USB and supports the languages ANSI-C, ANSI-C++ as well as any other language which can invoke the C functions.
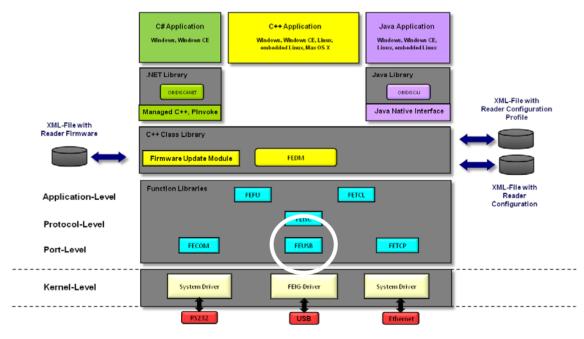
The support package offers a simple, device-independent function interface for USB devices in the FEIG family for all supported operating systems. The FEUSB is usually combined with an additional, device-specific function collection (e.g., ID FEISC).

The function collection is not designed to support any other USB devices except those from the FEIG family.

This library package can be used with the following operating systems:

Operating System	Target		Notes
	32-Bit	64-Bit	
Windows Vista / 7 / 8 10	X	X	
Windows CE 6	X	-	On request
Linux	X	X	
Android	X	X	On request
Apple Max OS X	-	X	On request

The library FEUSB is part of the first level of a hierarchical structured, multi-tier FEIG library stack. It is only designed for the data exchange between the USB driver of an operating system with an application. The following picture shows the multi-tier library stack.

Applications, based on the layer of FEUSB have to implement the protocol handling (building/splitting of protocol frames, CRC check, check of protocol frame). Thus, the implementation complexity is extensive and every Programmer should calculate the costs.

If the project forces to use only function libraries, the library FEISC from the next level should be chosen as the best API.

1.1. Shipment

This support package consists of files listed in the tables below. Normally, this package is shipped together with other libraries in a Software Development Kit (SDK) – e.g. ID ISC.SDK.Win.

1.1.1. Windows Vista / 7 / 8 / 10

File	Use
FEUSB.DLL	DLL with all functions
FEUSB.LIB	LIB file for linking for C/C++ projects
FEUSB.H	Header file for C/C++ projects

Additionally, the following driver files are essential for the use of FEIG USB devices. They are distributed with the driver CD, which is part of the USB device distribution or can be retrieved with the FEIG download server.

File	Use
OBIDUSB.SYS (V 2.50)	WHQL certified 32- and 64-Bit Windows kernel driver (Vista/7/8/10) for FEIG readers with USB interface
OBIDUSB.INF	Inf file for driver installation

1.1.2. Windows CE

Datei	Verwendung
FEUSBCE.DLL	DLL with all functions
FEUSBCE.LIB	LIB file for linking for C/C++ projects
FEUSB.H	Header file for C/C++ projects

Additionally, a platform dependent USB kernel driver is required and must be ordered separately.

1.1.3. Linux

File	Use
LIBFEUSB.SO.x.y.z[1]	Function library
FEUSB.H	Header file for C/C++ projects

Note:

The compiled library is linked against LibC V6 und LibStdC++ V6.

LIBFEUSB depends on the open source project libusb in the version 1.0.x which is not part of this support package. The binary of libusb can be downloaded from the project home page http://libusb.sourceforge.net and must be installed separately.

1.1.4. Mac OS X

File[2]	Use
LIBFEUSB.SO.x.y.z	Function library
FEUSB.H	Header file for C/C++ projects

Note:

LIBFEUSB depends on the open source project libusb in the version 1.0.x which is not part of this support package. The binary of libusb can be downloaded from the web and must be installed separately.

---

[1] x.y.z. represents the version number of the library file

[2] x.y.z represents the version number of the library file

2. Changes since the Previous Version

- Windows: Support for new Windows 10 compatible and WHQL certified USB kernel driver

- All other OS: no changes
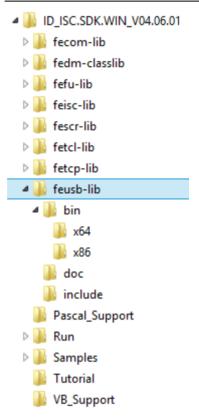
Please also note the revision history in the appendix to this document.

3. Installation

Normally, this package is shipped together with other libraries in a Software Development Kit (SDK). Copy the SDK into a directory of your choice.

The files of this library package can be found in the sub-directory feusb-lib.

3.1. 32-Bit Windows Vista / 7 / 8 / 10
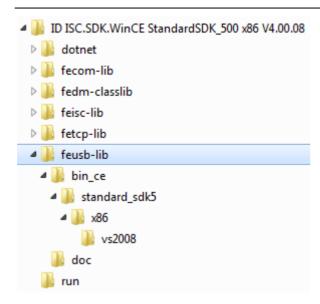
```
4 📁 ID_ISC.SDK.WIN_V04.06.01
   ▷ 📁 fecom-lib
   ▷ 📁 fedm-classlib
   ▷ 📁 fefu-lib
   ▷ 📁 feisc-lib
   ▷ 📁 fescr-lib
   ▷ 📁 fetcl-lib
   ▷ 📁 fetcp-lib
   4 📁 feusb-lib
      4 📁 bin
         📁 x64
         📁 x86
         📁 doc
         📁 include
      📁 Pascal_Support
   ▷ 📁 Run
   ▷ 📁 Samples
      📁 Tutorial
      📁 VB_Support
```

If you won't add your projects to the Samples path, we recommend the following steps:

- Copy FEUSB.DLL into the directory of the application program (recommended) or into the Windows system directory.

- Copy FEUSB.LIB into the project or LIB directory.

- Copy FEUSB.H into the project or INCLUDE directory.

Driver installation of OBIDUSB.SYS must be performed **<u>before</u>** the first connecting of a FEIG USB device into the PC. The installation is performed by an operating system wizard. For additional information, refer to the documentation included in the driver package.

3.2. Windows CE 6
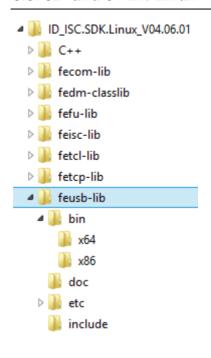
ID ISC.SDK.WinCE StandardSDK_500 x86 V4.00.08
- ▷ dotnet
- ▷ fecom-lib
- ▷ fedm-classlib
- ▷ feisc-lib
- ▷ fetcp-lib
- ▲ feusb-lib
  - ▲ bin_ce
    - ▲ standard_sdk5
      - ▲ x86
        - vs2008
  - doc
- run

If you won't add your projects to the Samples path, we recommend the following steps:

- Copy FEUSBCE.DLL into the system directory of the Windows CE system.

- Copy FEUSBCE.LIB into the project or LIB directory.

- Copy FEUSB.H into the project or INCLUDE directory

Driver installation of OBIDUSB.SYS must be performed before the first connecting of a FEIG USB device into the PC. Installation instruction is part of the driver kit.

Note: you cannot use the DLL together with eMbedded Visual Basic 3.0.

3.3. 32- and 64-Bit Linux

▲ 📁 ID_ISC.SDK.Linux_V04.06.01
  ▷ 📁 C++
  ▷ 📁 fecom-lib
  ▷ 📁 fedm-classlib
  ▷ 📁 fefu-lib
  ▷ 📁 feisc-lib
  ▷ 📁 fetcl-lib
  ▷ 📁 fetcp-lib
  ▲ 📁 feusb-lib
    ▲ 📁 bin
      📁 x64
      📁 x86
    📁 doc
   ▷ 📁 etc
    📁 include

Choose one option for installation:

Option 1: If an install.sh is shipped inside the SDK root directory, execute this install script. It will copy all library files into the directory /usr/lib resp. /usr/lib64 and create symbolic links for each library file. The header file can be copied into a directory of your choice.

Option 2: Copy all files of this support package into a directory of your choice and create symbolic links for libfeusb.so.x.y.z[1] in the directory /usr/lib resp. /usr/lib64 with the following calls:

cd /usr/lib (for 64 Bit : /usr/lib64)

ln –s /< your_directory>/libfeusb.so.x.y.z   libfeusb.so.x

ln –s /< your_directory>/libfeusb.so.x   libfeusb.so

ldconfig

Usage of libfeusb.so.x.y.z without administration rights:

Requirements:

    The udev daemon is running and handles the hotplugging of the usb devices.

    The application chmod must be located in the dircetory /bin.

- copy the file **41-feig.rules** to the directory /etc/udev/rules.d.

The compiled library is linked against LibC V6 und LibStdC++ V6.

---

[1] x.y.z represents the version number

3.3.1. libusb

LIBFEUSB depends on the open source project libusb in the version 1.0.x which is not part of this support package. This binary is part of your target system. libusb can be downloaded from the project home page http://libusb.sourceforge.net.

3.3.2. Configuration

LIBFEUSB can be configured via configuration file "/etc/feig/feusb.conf". Valid entries are:

- **feusb_debuglevel**=<level>
  <level> can be 0=off up to 3=max. For standard test purposes we recommend level 2.

- **libusb_debuglevel**=<level>
  <level> can be 0=off up to 4=max. To use libusb debugging, it is necessary to compile libusb as developer version.

- **feusb_hotplug**=<mode>
  Valid values for <mode> are:
  0: No hotplug support
  1: Hotplug is supported by LIBFEUSB. This is performed by an internal thread.
  2: Hotplug is supported by libusb. This feature is not supported on all targets (consider debug message "`fesub: [init]libUSB-HasHotPlug=0/1`").

- **feusb_updaterate**=<rate>
  In case of "feusb_hotplug=1" <rate> is the update rate of the internal thread in milliseconds.

If no configuration file exists, no debug output is performed. Hotplug is supported by LIBFEUSB with an update rate of 300 milliseconds.

3.3.3. Unit Tests

Two unit test tools are available for LIBFEUSB (we recommend feusb_debuglevel=2):
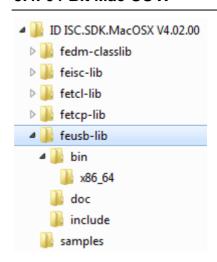
- **feusbTest**
  Use this tool to test basic USB communication with a FEIG reader. It consists of several test sections. Press "Enter" to perform the next section.

- **feusbTest2**
  Use this tool to test the hotplug functionality. Every 2 seconds this tool communicates with all connected FEIG readers. You can plug and unplug an arbitrary number of readers. Use the command line parameter `-i<msec>` to use a different communication rate.

3.4. 64-Bit Mac OS X

ID ISC.SDK.MacOSX V4.02.00
- ▷ fedm-classlib
- ▷ feisc-lib
- ▷ fetcl-lib
- ▷ fetcp-lib
- ▲ feusb-lib
  - ▲ bin
    - x86_64
  - doc
  - include
- samples

Choose one option for installation:

Option 1: If an install.sh is shipped inside the SDK root directory, execute this install script. It will copy all library files into the directory /usr/local/lib and create symbolic links for each library file. The header file can be copied into a directory of your choice.

Option 2: Copy all files of this support package into a directory of your choice and create symbolic links for libfeusb.x.y.z.dylib[1] in the directory /usr/local/lib with the following calls:cd /usr/local/lib

ln –s libfeusb.x.y.z.dylib   libfeusb.x.dylib

ln –s libfeusb.x.dylib   libfeusb.dylib

Note: The library is compiled under Mac OS X V10.7.3 with Xcode V4.3.2 and is compatible with the architecture x86_64. Newer compilations on request.

3.4.1. libusb

LIBFEUSB depends on the open source project libusb in the version 1.0.x which is not part of this support package. This binary is part of your target system. libusb can be downloaded from the project home page http://libusb.sourceforge.net.

---

[1] x.y.z represents the version number

3.5. Deactivating the Plug-and-Play Thread (Windows)

For observing the USB for FEIG readers, a thread is internally started to search cyclical for new devices. Newly detected readers are notified with the event mechanism (see 6.4. Event Signalling) if installed.

If this automatism is not applicable for your application you can prevent the start of the thread with the following steps:

a) Create a file feusb.conf

b) Add the text *nopnp*

c) Save this file in the directory of the application

4. Incorporating into the Application Program

4.1. Supported Development Tools

Operating System	Development Tool	Supported
Windows Vista / 7 / 8 / 10	Visual Studio	Yes
	Borland C++ Builder	Yes
	Embarcadero C++ Builder	Yes
Windows CE 6	eMbedded Visual C++ 4	No
	Visual Studio 2005	No
	Visual Studio 2008	Yes
Linux	GCC	Yes
Mac OS X	GCC	Yes, for projects with x86_64 architecture
	Xcode	Yes, for projects with x86_64 architecture

4.2. Incorporating into Visual Studio

1. Add include path for the header file in project settings (category C/C++)

2. Add fetcp.lib (optional with path) in project settings (category Linker)

4.3. Incorporating into Xcode

1. Add path for the header file in project settings (User Header Search Paths in category Search Paths)

2. add feusb.dylib with drag'n drop to your project

5. A Brief Introduction to USB

USB (Universal Serial Bus) represents a new standard for connecting peripherals in the PC environment. Compared with the serial interface, USB features especially plug & play capability and higher transfer speed. On the other hand, this new standard also requires deeper knowledge of the characteristics of USB if you want to access USB devices from the user software.

The FEUSB function collection gives the application programmer the necessary help in communicating with USB devices from the FEIG family. With the elementary knowledge provided in this section, any practiced programmer will be able to develop professional application programs[1].

USB is a single-master bus with the PC as master (host). Only this master can generate protocol activities. Up to 127 physical devices can be supported at the same time. The devices differ in their bus addresses, which are automatically assigned by the host. After a peripheral is plugged in, an initialization phase (enumeration) is automatically started in the host, which allows the host to load the appropriate driver(s). This process is always triggered by the operating system (regardless of manufacturer).

In physical terms a USB device always consists of at least one logical USB device. This means the communication data can be stacked within the device into several information channels, the so-called pipes. Each pipe has an end point assigned to it which corresponds physically to a FIFO.

A logical USB device can combine several pipes into an interface, and the host can install an appropriate driver for such an interface. The host obtains the information about the logical composition of a USB device during enumeration.

USB devices from the FEIG family are characterized in that they all have uniform interfaces. This means the special USB drivers can be categorized as device-independent within the FEIG family. The programmer does now however come into contact with these drivers, interfaces, pipes or bus addresses. For him a programming model has been developed which enables communication with FEIG USB devices in no more than four steps.

1. <u>Scan process</u>: a function invoke detects all FEIG devices on the USB and administers them in a scan list within the DLL.
2. <u>Device selection</u>: in the second step this scan list is used to select a USB device based on its serial number. The serial number is by the way the only feature which distinguishes the devices from each other.
3. <u>Open communications path</u>: in the third step a channel to this USB device is opened[2]. A data structure, the device object, is created internally in the DLL.
4. <u>Data exchange</u>: beginning with the fourth step data can be exchanged with the USB device.

---

[1] For readers interested in the details, we recommend „Universal Serial Bus System Architecture, Second Edition", Don Anderson, Addison Wesley, 2001

[2] The special function FEUSB_ScanAndOpen combines steps 1, 2 and 3 together. This function can be used if you are normally connecting only one USB device from the FEIG family.

A FEIG USB device can have one or more interfaces, and all the programmer needs is the FEUSB function collection in order to decide which data he has to send over which interface. For this purpose an additional function collection is provided for each FEIG device family. The programmer therefore does not need to deal with the peculiarities of the FEIG interface.

6. Programming Interface

6.1. Overview

The FEUSB combines for the user all the necessary functions and parameters for administering one or more simultaneously opened FEIG USB devices on the USB port of the PC. The object-oriented internal structure (see Fig. 1) is intentionally implemented to the outside as a function interface. This has the advantage that it is language-independent.

The library has self-administration which frees an application program from having to buffer any values, settings or other information. The driver manager in FEUSB keeps a list with all opened channels (created device objects) and each device object administers all relevant settings for its channel within its local memory. Exactly one opened channel is always connected between a device object and a certain FEIG USB device, and only the devices registered with its serial number can be accessed through this channel. A channel to an FEIG USB device can only be opened once.
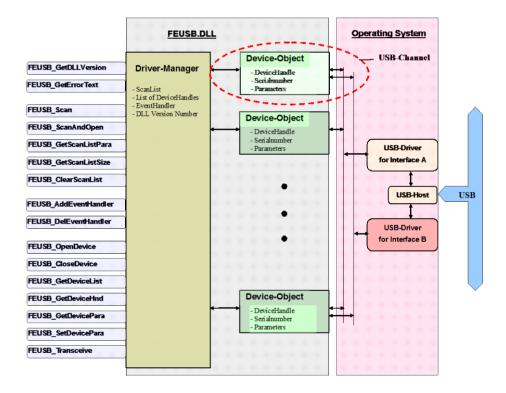
Fig. 1: Internal structure of FEUSB.DLL

The first step in establishing a connection with an FEIG USB device is to detect (scan procedure) one or all FEIG USB devices on the USB of the PC. Each found device is entered in the internal scan list but not opened.

Before the first communication a USB device must be selected from the scan list and the function FEUSB_OpenDevice used to open a channel to this device. If this function has been executed without error, you get a handle back with the return value which can be administered by the application program. Unique identification of the opened channel is possible only with this handle.

The handle(s) does/do not however need to be stored in the application program, since the driver manager internally administers a list of all opened channels. This list can be invoked using the function FEUSB_GetDeviceList.

A channel which has been opened with FEUSB_OpenDevice must always be closed using the function FEUSB_CloseDevice.

Every library function (exception: FEUSB_GetDLLVersion) has a return value which is always negative in case of error.

If an application program is opened multiple times, each program (instance) gets an empty device list with the function invoke FEUSB_GetDeviceList. This prevents mixing up of access rights under different program instances. Please note that in contrast to a serial interface, a USB channel can be opened again by any program or another instance! This means that different programs can exchange data quasi-simultaneously with one and the same USB device. The resulting possible access conflicts are not caught by the FEUSB.

6.2. Thread Security

In principle, all FEIG libraries are not fully thread safe. But respecting some guidance, a practical thread security can be realized allowing parallel execution of communication tasks. One should keep in mind, that all FEIG RFID readers work synchronously and can perform commands only in succession.

On the level of the transport layer (FECOM, FEUSB, FETCP) the communication with each port must be synchronized in the application, as the reader works synchronously. Using multiple ports and so multiple readers from different threads simultaneously is possible, as the internal port objects acts independently from each other.

6.3. Structure and Function of the Scan List

Opening a channel to a FEIG USB device is only possible with its individual serial number (Device ID). Before opening, a scan procedure must be used (with **FEUSB_Scan** or **FEUSB_ScanAndOpen**) to detect one (or more) FEIG USB devices on the USB port and the serial number(s) read out. The located USB devices are entered in the internal scan list and there administered according to their serial number. After opening (with **FEUSB_ScanAndOpen** or **FEUSB_OpenDevice**), the device handle of the channel is added to the scan list. In addition, a note is made that the USB device is ready.

The structure of the internal scan list contains the following data elements:

```
int      iScanNo;           // Index in scan list (>= 0)
DWORD dwDeviceID;           // Serial number (>0)
int      iDeviceHnd;        // Device-Handle (0: channel not opened; >0: channel opened)
char     cFamilyName[25];   // Name of device family (e.g. "FEIG HF")
char     cDeviceName[25];   // Device-Name (e.g. "ID ISC.PRH100-U")
bool     bPresent;          // Ready flag (true or false)
```

Each data element in the scan list can be read using the function **FEUSB_GetScanListPara**.

An essential data element is the ready flag *bPresent*, which indicates whether the device is still connected to the USB port after opening the channel. If you remove a USB device after a channel to it has been opened, the ready flag is set internally to false. The channel however remains opened. If the same device is then re-connected, the ready flag is set back to true and communication can immediately resume.

There are three ways to determine the readiness of a USB device:

- Query the ready flag with **FEUSB_GetScanListPara**(iIndex, „PRESENT", cValue)
- Query readiness with **FEUSB_IsDevicePresent**(iDevHnd)
- Establish an event signalling scheme (see Section 6.4. Event Signalling)

The scan list can be cleared at any time using the function **FEUSB_ClearScanList**. Any opened channels are not closed! The two scan functions can be used to then reconstruct the scan list at any time. Channels kept open are detected and the ready flags are correspondingly set. This means that clearing the scan list does not permanently lose important information.

You should not however use the scan list to close opened channels, since for the reasons indicated above it does not actively administer each open channel. It is better to always use the device list, which can be read out cyclically using the function **FEUSB_GetDeviceList**.

6.4. Event Signalling

For the plug & play events[1] **_Connect_** and **_Disconnect_**, event handling procedures can be installed individually for each event and regardless of whether the device is already listed in the internal scan list. As soon as a USB device is plugged in or unplugged, the corresponding signalling is performed. In this way, you can inform an application of the event asynchronous to the program sequence.

An event handling procedure must be installed using the function **FEUSB_AddEventHandler**. You can select from among three signaling methods: Message to invoking process, message to a window or use of a callback function.

An installed event handling procedure must be deleted only using the function **FEUSB_DelEventHandler**.

The structure **FEUSB_EVENT_INIT** contains the parameters necessary for signalling:

```
typedef struct _FEUSB_EVENT_INIT
{
    UINT uiFlag;      // Specifies use of the union (z.B. FEUSB_WND_HWND)
    UINT uiUse;       // Defines the event (e.g. FEUSB_CONNECT_EVENT)
    UINT uiMsg;       // Message code for dwThreadID and hwndWnd (e.g. WM_USER_xyz)
    union
    {
        DWORD  dwThreadID;              // for Thread-ID
        HWND    hwndWnd;                // for Window handle
        void       (*cbFct)(int, DWORD);    // for Callback function
    } Method[2];
} FEUSB_EVENT_INIT;
```

The core element in this structure is the **union**, which contains either the ID of a process, the handle of a window or a function pointer. The signalling form is selected using the parameter _uiFlag_. The _uiUse_ parameter is where you store the ID of the event to which you want to assign the handling method. You must store the message code for the message methods in _uiMsg_.

You may install multiple event handling methods for an event. However, each _dwThreadID_, _hwndWnd_ or _cbFct_ may only be used once per event.

Annotation to Linux: The connect signaling for FEIG USB devices with additional HID interface takes about 10 to 12 seconds.

---

[1] The event signaling can be generally disabled. See 3.5. Deactivating the Plug-and-Play Thread.

[2] Naming of the union using method is only for C programmers. C++ programmers access the union directly through the structure.

6.5. Functions[1]

- **void FEUSB_GetDLLVersion(** char* cVersion **)**
- **int FEUSB_GetErrorText(** int iError, char* cText **)**
- **int FEUSB_GetLastError(** int iDevHnd , int* iErrorCode, char* cErrorText **)**

- **int FEUSB_Scan(** int iScanOpt, FEUSB_SCANSEARCH* pSearchOpt **)**
- **int FEUSB_ScanAndOpen(** int iScanOpt, FEUSB_SCANSEARCH* pSearchOpt **)**
- **int FEUSB_GetScanListPara(** int iIndex, char* cPara, char* cValue **)**
- **int FEUSB_GetScanListSize()**
- **int FEUSB_ClearScanList()**

- **int FEUSB_AddEventHandler(** int iDevHnd, FEUSB_EVENT_INIT* pInit **)**
- **int FEUSB_DelEventHandler(** int iDevHnd, FEUSB_EVENT_INIT* pInit **)**

- **int FEUSB_OpenDevice(** long nDeviceID **)**
- **int FEUSB_CloseDevice(** int iDevHnd **)**
- **int FEUSB_GetDeviceList(** int iNext **)**
- **int FEUSB_GetDeviceHnd(** long nDeviceID **)**
- **int FEUSB_GetDevicePara(** int iDevHnd, char* cPara, char* cValue **)**
- **int FEUSB_SetDevicePara** (int iDevHnd, char* cPara, char* cValue)

- **int FEUSB_Transceive(** int iDevHnd, char* cInterface, int iDir, UCHAR* cSendData, int iSendLen, UCHAR* cRecData, int iRecLen **)**
- **int FEUSB_Transmit(** int iDevHnd, char* cInterface, UCHAR* cSendData, int iSendLen **)**
- **int FEUSB_Receive(** int iDevHnd, char* cInterface, UCHAR* cRecData, int iRecLen **)**

---

[1] **Note**: UCHAR is defined as an 8-bit unsigned char. In Visual Basic the compatible data type is the byte.

6.6. Function Descriptions

6.6.1. FEUSB_GetDLLVersion

Function	Gets the version number of the DLL
Syntax	**void FEUSB_GetDLLVersion(char* cVersion)**
Description	The function returns the version number of the DLL. *cVersion* is an empty, null-terminated character string for returning the version number. The string should be able to hold at least 256 characters. The string is filled with the current version number (e.g. "06.00.00"). Newer versions could however provide additional information.
Return value	None
Example	... #include "feusb.h" ... ... char cVersion[256]; FEUSB_GetDLLVersion(cVersion) // Code here for displaying the version number ... ...

6.6.2. FEUSB_GetDrvVersion (only for Windows)

Function	Gets version information from installed kernel-driver
Syntax	**int FEUSB_GetDrvVersion(char* cVersion)**
Description	The function returns version information from the installed kernel-driver. ATTENTION: this function returns the information only if the kernel-driver is load. This is the case when a channel to the USB reader is open. *cVersion* is an empty, null-terminated character string for returning the version information. The string should be able to hold at least 256 characters.
Return value	In case of error the function returns <0, otherwise 0. The list of error codes can be found in the appendix.

Example	...
	#include "feusb.h"
	...
	...
	char cVersion[256];
	if(0 == FEUSB_GetDrvVersion(cVersion))
	    // Code here for displaying the version information
	...
	...

6.6.3. FEUSB_GetErrorText

Function	Returns error text
Syntax	**int FEUSB_GetErrorText(int iError, char* cText)**
Description	The function returns an error text[1] for the error code. *iError* is the error code (always negative). *cText* is an empty, null-terminated string for returning the error text. The string should be able to hold at least 256 characters.
Return value	In case of error the function returns the code FEUSB_ERR_UNKNOWN_ERRORCODE, otherwise 0. The list of error codes can be found in the appendix.
| Example | ```
...
#include "feusb.h"
...
...
char cText[256];
int iErr = FEUSB_GetErrorText( -1100, cText )
    // Code here for displaying the error text
...
``` |

6.6.4. FEUSB_GetLastError

Function	Gets the last error code and transfers error text
Syntax	**int FEUSB_GetLastError(int iDevHnd , int* iErrorCode, char* cErrorText)**
Description	The function uses *iErrorCode* to transfer the last error code of the USB channel selected with *iDevHnd* and transfers the associated English error text in *cErrorText*
Return value	In case of no error, returns the function zero and in case of error a value less than zero. The list of error codes can be found in the appendix.
| Example | ```
...
#include "feusb.h"
...
...
char cErrorText[256];
int iErrorCode = 0;
...

int iBack = FEUSB_GetLastError( iDevHnd, &iErrorCode, cErrorText )
    // Code here for displaying the Text
...
...
``` |

---

[1] in English

6.6.5. FEUSB_Scan

Function	Detecting a single or all USB devices
Syntax	**int FEUSB_Scan(int iScanOpt, FEUSB_SCANSEARCH* pSearchOpt)**
Description	With this function the USB port is searched for devices with the FEIG identifier, and each found device is entered in the internal scan list. The parameter *iScanOpt* allows searching for one or all devices, or allows no longer existing devices to be deleted from the scan list. The index of the scan list is null-based. The parameter *pSearch Opt* is used for targeted searching with the option FEUSB_SCAN_SEARCH. If this option is not used, NULL must be passed in C/C++. In Visual Basic you pass the constant vbNullString.

The parameter *iScanOpt* controls the scan procedure and is comprised of *iScanOpt* = [*CommandID*] | [*OptionID*] | [*OptionID*].

CommandIDs:

- FEUSB_SCAN_FIRST searches for the device that was the first one to be registered by the operating system. The internal scan counter is thus set to 0. The scan list is cleared before the scanning process.

- FEUSB_SCAN_NEXT searches for the device that was next registered by the operating system. For this the internal scan counter is used, which is incremented with each successful FEUSB_SCAN_NEXT (up to max. 127). Note: each FEUSB_SCAN_FIRST resets the internal scan counter to 0!

- FEUSB_SCAN_NEW searches for a new device which is not yet entered in the scan list. Te internal counter is correspondingly set anew.

- FEUSB_SCAN_ALL allows searching for all devices on the USB port. The scan counter is correspondingly set anew. The scan list is first cleared and then reconstructed. You cannot therefore assume that a device previously entered in the scan list will be listed with the same index.

- FEUSB_SCAN_PACK deletes all devices from the internal scan list which are no longer found on the USB port.

OptionIDs:

- FEUSB_SCAN_SEARCH searches for a specific device on the USB port. You enter the search options in the parameter *pSearchOpt*[1]. This OptionID must always be linked to a CommandID.

- FEUSB_SCAN_PACK deletes all devices from the internal scan list which are no longer found on the USB port. The option is only used if the previous (optional) scan

---

[1] see appendix 8.4. Constants for the FEUSB_SCANSEARCH, 8.5. cFamilyName in the FEUSB_SCANSEARCH Structure und 8.6. cDeviceName in the FEUSB_SCANSEARCH Structure

	was without error. Caution: this option changes the list index for the already entered devices! This option can be combined with all others. But for the CommandID FEUSB_SCAN_ALL this is superfluous, since this automatically reconstructs the scan list.
Note	Clearing the scan list with **FEUSB_ClearScanList** does not (!) close the objects opened with **FEUSB_OpenDevice**.
Return value	If one or more USB devices were found, the return value contains with option FEUSB_SCAN_FIRST, FEUSB_SCAN_NEXT or FEUSB_SCAN_NEW the index of the device in the scan list or 0 with the option FEUSB_SCAN_ALL. In case of error the function returns a value less than null. Even when an error has occurred some devices can be detected and added to the scan list. After a scan using option FEUSB_SCAN_ALL you should therefore always use **FEUSB_ScanListSize** to check the size of the scan list. The list of error codes can be found in the appendix.
Scan Options	FEUSB_SCAN_FIRST 0x00000001 FEUSB_SCAN_NEXT 0x00000002 FEUSB_SCAN_NEW 0x00000003 FEUSB_SCAN_ALL 0x0000000F FEUSB_SCAN_SEARCH 0x00010000 FEUSB_SCAN_PACK 0x00020000
Search Options	FEUSB_SEARCH_FAMILY 0x00000001 FEUSB_SEARCH_PRODUCT 0x00000002 FEUSB_SEARCH_DEVICEID 0x00000004

Example	```
	...
	#include "feusb.h"
	...
	char cDeviceID[16];
	long nDeviceID;
	FEUSB_SCANSEARCH search;
	...
	// Set search options
	search.iMask = FEUSB_SEARCH_PRODUCT;
	strcpy(search.cDeviceName, "ID ISC.MR101-U");
	
	if(FEUSB_Scan(FEUSB_SCAN_FIRST, &search) == 0)
	{
	    if(FEUSB_GetScanListPara(0, "Device-ID", cDeviceID) == 0)
	    {
	        sscanf((const char*)cDeviceID, "%lx", &nDeviceID);
	        int iDevHnd = FEUSB_OpenDevice(nDeviceID);
	        if(iDevHnd < 0)
	        {
	            // Code here for error
	        }
	        else
	        {
	            // Code here for communication or other
	        }
	    }
	}
	``

6.6.6. FEUSB_ScanAndOpen

Function	Detecting a single or all USB devices and then opening the channels
Syntax	**int FEUSB_ScanAndOpen(int iScanOpt, FEUSB_SCANSEARCH* pSearchOpt)**
Description	This function combines the functions **FEUSB_Scan** and **FEUSB_OpenDevice**. This function is useful for directly opening a channel to a device in the frequently occurring case where exactly one FEIG device is found on the USB port. The description of the parameters can be found in the sections for the functions in question.
Return value	If one or more channels to USB devices could be opened without error, the return value contains with option FEUSB_SCAN_FIRST, FEUSB_SCAN_NEXT or FEUSB_SCAN_NEW the device handle or 0 with the option FEUSB_SCAN_ALL. In the last case the device handles of the opened USB channels must be queried with the function **FEUSB_GetScanListPara**. In case of error the function returns a value less than null. Even when an error has occurred some devices can be detected and the channel opened. After a ScanAndOpen call using option FEUSB_SCAN_ALL you should therefore always use **FEUSB_GetScanListSize** to check the size of the scan list. If the channel to the USB device was able to be opened without error, a handle (>0) is returned. In case of error the function returns a value less than zero. The list of error codes can be found in the appendix.
Cross-references	6.5.4. FEUSB_GetLastError, 6.5.10. FEUSB_OpenDevice
Example	see examples for **FEUSB_Scan** and **FEUSB_GetScanListSize**

6.6.7. FEUSB_GetScanListPara

Function	Reads a value from the scan list
Syntax	**int FEUSB_GetScanListPara(int iIndex, char* cPara, char* cValue)**
Description	This function gives you access to the values in the scan list. Each record contains several values for the found FEIG device. Access is gained using the null-based index *iIndex*.
	In the parameter *cPara* you indicate the identifier for the corresponding scan list value (see Parameter Field).
	cValue is an empty, null-terminated string for returning the scan list value. The string should be able to hold at least 25 characters.
Return value	In case of error the function returns a value less than zero. The list of error codes can be found in the appendix.
Parameter	The parameter identifiers are: „Device-ID"　　- serial number of the USB device in hexadecimal representation „DeviceHnd"　　- device handle for the USB channel „FamilyName"　- name of the device family for the device on the USB channel „DeviceName"　- name of the device on the USB channel „Present"　　　- USB device is connected (cValue=„1") or disconnected (cValue=„0")
Note	The **Device-ID** obtained represents a hex value. For example, "6D89573" is associated with Device-ID 0x06D89573 or 114857331.
	The following example shows how to convert the string:

	cDeviceID[16];
	long nDeviceID = 0;
	...
	if(FEUSB_GetScanListPara(index, "Device-ID", cDeviceID) == 0)
	{
	　　sscanf((const char*)cDeviceID, "%lx", &nDeviceID);
	　　iDeviceHnd = FEUSB_OpenDevice(nDeviceID);
	}

Example	see examples for **FEUSB_Scan** and **FEUSB_GetScanListSize**

6.6.8. FEUSB_GetScanListSize

Function	Gets the size of the scan list
Syntax	**int FEUSB_GetScanListSize()**
Description	This function is used to obtain the number of data records in the scan list.
Return value	In case of error the function returns a value less than zero. The list of error codes can be found in the appendix.

Example	<pre>...
	#include "feusb.h"
	...
	int iDevHnd;
	char cDeviceID[16];
	long nDeviceID;
	...
	...
	FEUSB_Scan(FEUSB_SCAN_ALL, NULL);
	
	for(int iCnt=0; iCnt = FEUSB_GetScanListSize(); iCnt++)
	{
	    if(FEUSB_GetScanListPara(iCnt, "Device-ID", cDeviceID) == 0)
	    {
	        sscanf((const char*)cDeviceID, "%lx", &nDeviceID);
	        iDevHnd = FEUSB_OpenDevice(nDeviceID);
	        if(iDevHnd < 0)
	        {
	            // Code here for error
	        }
	        else
	        {
	            // Code here for communication or other
	        }
	    }
	}
	...
	...</pre>

6.6.9. FEUSB_ClearScanList

Function	Clears the scan list
Syntax	**int FEUSB_ClearScanList()**
Description	This function clears the scan list. Already opened device objects are not automatically closed by this function. This means it is possible to perform a new scan to restore the scan list.
Return value	In case of error the function returns a value less than zero. The list of error codes can be found in the appendix.
Example	

6.6.10. FEUSB_OpenDevice

Function	Opens a channel for communication with an FEIG reader.
Syntax	**int FEUSB_OpenDevice(long nDeviceID)**
Description	The function opens a USB channel and internally creates a device object for administering the channel parameters. The returned handle *iDevHnd* identifies the channel from the outside.

nDeviceID is the serial number of the FEIG device on the USB channel you are opening.

The function invoke is ended with an error if the serial number was not found in the scan list.

After successful opening, the device handle is also stored in the scan list and the ready flag set. Thus simply reading out the scan list allows you to determine which devices were found, are open and ready.

The USB channel opened with **FEUSB_OpenDevice** must (!) be closed using the function **FEUSB_CloseDevice**. Otherwise the memory reserved by the DLL is not freed up again.

Repeated invoking of this function with the same serial number does not result in repeated opening of channels, rather the associated handle is returned. |
| **Return value** | If the channel to the USB device could be opened without error, a handle (>0) is returned. In case of error the function returns a value less than zero. The list of error codes can be found in the appendix. |
| **Example** | see examples for **FEUSB_Scan** and **FEUSB_GetScanListSize** |

6.6.11. FEUSB_CloseDevice

Function	Closes a USB channel to an FEIG device
Syntax	**int FEUSB_CloseDevice(int iDevHnd)**
Description	The function closes the USB channel specified by the parameter *iDevHnd* and frees up the reserved memory.
Return value	The return value is 0 if the channel was closed. In case of error the function returns a value less than zero. The list of error codes can be found in the appendix.

Example	<pre>...
	#include "feusb.h"
	...
	char cDeviceID[16];
	long nDeviceID;
	
	...
	if(FEUSB_Scan(FEUSB_SCAN_FIRST, NULL) == 0)
	{
	    if(FEUSB_GetScanListPara(0, "Device-ID", cDeviceID) == 0)
	    {
	        sscanf((const char*)cDeviceID, "%lx", &nDeviceID);
	        int iDevHnd = FEUSB_OpenDevice(nDeviceID);
	        if(iDevHnd < 0)
	        {
	            // Code here for error
	        }
	        else
	        {
	            int iErr = FEUSB_CloseDevice(iDevHnd);
	        }
	    }
	}
	
	...
	...</pre>

6.6.12. FEUSB_IsDevicePresent

Function	Checks for the presence of a USB device
Syntax	**int FEUSB_IsDevicePresent(int iDevHnd)**
Description	The function checks for the presence of the USB device on the USB channel specified by the parameter *iDevHnd*.
Return value	The return value is 1 if the USB device is ready for communication, otherwise 0. In case of error the function returns a value less than zero. The list of error codes can be found in the appendix.
Example	

6.6.13. FEUSB_GetDeviceList

Function	Uses the parameter *iNext* to get the first or following device handle from the internal list of the opened serial interfaces.
Syntax	**int FEUSB_GetDeviceList(int iNext)**
Description	The function returns a device handle from the internal list of device handles. If you pass a 0 for *iNext*, the first entry in the list is returned. If you use *iNext* to pass a device handle kept in the list, the entry following the device handle is obtained and returned. In this way, you can successively use the return value to go through the list from front to back and open all entries.
Return value	Once an entry is found, the return value is used to provide the device handle. When the end of the internal list is reached, i.e., the passed device handle has no successor, a 0 is returned. If no USB channel is open, FEUSB_ERR_EMPTY_DEVLIST is returned. In case of error the function returns a value less than zero. The list of error codes can be found in the appendix.

Example	```#include "feusb.h"```
	 ``` ... ```
	 ``` // gets the DeviceIDs for all open USB channels ```
	 ``` char cValue[16]; ```
	 ``` ... ```
	 ``` int iNextHnd = FEUSB_GetDeviceList(0);// get the first handle ```
	 ``` while(iNextHnd > 0) ```
	 ``` { // read out DeviceID here ```
	 ``` int iBack = FEUSB_GetDevicePara(iNextHnd, "Device-ID", cValue) ```
	 ``` printf("%s", cValue); // print to screen ```
	 ``` iNextHnd = FEUSB_GetDeviceList(iNextHnd); // get next handle ```
	 ``` } ```
	 ``` ... ```
	 ``` ... ```

Tip	When closing all open USB channels, it is advantageous to use a loop similar to the above example. Simply bear in mind that you cannot get a successor from a closed channel. In the following code fragment you can see how to close all open channels in a loop:
	
	``` ... ```
	 ``` int iCloseHnd, iNextHnd; ```
	
	 ``` iNextHnd = FEUSB_GetDeviceList(0); // get first handle ```
	 ``` while(iNextHnd > 0) ```
	 ``` { ```
	 ``` iCloseHnd = iNextHnd; ```
	 ``` iNextHnd = FEUSB_GetDeviceList(iNextHnd); // get next handle only ```
	 ``` FEUSB_CloseDevice(iCloseHnd); // only now close USB channel to the device ```
	 ``` } ```
	 ``` ... ```

6.6.14. FEUSB_GetDeviceHnd

Function	Gets the device handle from an open USB channel
Syntax	**int FEUSB_GetDeviceHnd(long nDeviceID)**
Description	This function provides an easy way to get the device handle of a previously opened USB channel. *dwDeviceID* is the serial number of the USB device. This function is a "reverse function" of **FEUSB_GetDevicePara** (iDevHnd, "Device-ID", cValue), which gets the serial number of the device on the USB channel for the device handle.
Return value	If the channel for the passed serial number was found, the device handle (>0) is returned. If the desired serial number was not found in the device list, a 0 is returned. In case of error the function returns a value less than zero. The list of error codes can be found in the appendix.
| **Example** | ```
...
#include "feusb.h"
...
...
int iDevHnd = FEUSB_OpenDevice( nDevice );
 if(iDevHnd < 0 )
{
    // Code here for error
}
else
{    // handle is obtained again using DeviceID
    iDevHnd = FEUSB_GetDeviceHnd( nDevice );
}
``` |

6.6.15. FEUSB_GetDevicePara

Function	Gets a parameter from the USB channel specified by *iDevHnd.*
Syntax	**int FEUSB_GetDevicePara(int iDevHnd, char* cPara, char* cValue)**
Description	The function gets the current value of a parameter. *cPara* is a null-terminated character string with the parameter identifier. *cValue* is an empty, null-terminated string for returning the parameter value. The string should be able to hold at least 128 characters.
Parameter identifiers	The parameter identifiers are: "Device-ID" - Serial number of the USB device in hexadecimal representation "FamilyName" - Name of the device family of the device on the USB channel "DeviceName" - Name of the device on the USB channel This data is not case-sensitive.
Return value	If there is no error, the function returns the value 0 and in case of error a value less than zero. The list of error codes can be found in the appendix.
Cross-reference	For additional information, see: 8.2. Parameter Identifiers.
| **Example** | ```
...
#include "feusb.h"
...
...
char cValue[128];
long nDeviceID;
...
if( FEUSB_GetDevicePara( iDevHnd, "Device-ID", cValue ) == 0 )
    {
        // code here for displaying the parameter
        ...
        // or conversion into DWORD
        sscanf((const char*)cValue, "%lx", &nDeviceID);
    }
...
...
}
``` |

6.6.16. FEUSB_SetDevicePara

Function	Sets a parameter for a USB channel to a new value.
Syntax	**int FEUSB_SetDevicePara(int iDevHnd, char* cPara, char* cValue)**
Description	The function passes a new parameter to the USB channel specified by *iDevHnd*. *cPara* is a null-terminated character string with the parameter identifier. *cValue* is a null-terminated character string with the new parameter value.

Parameter identifier	Value range	Default value	Unit	Comment
TIMEOUT	0...99999	1000	ms	This parameter can only be set for opened USB channels. If this setting should have an affect to all opened USB channels, the iDevHnd must be zero.
EXCLUSIVEACCESS (only for Windows)	0, 1	1	-	Activates (1) or deactivates (0) the exclusive access. This setting is global for all USB reader. Thus, iDevHnd must be set to 0.

Return value	If the USB channel with the new parameter value was able to be successfully initialized, a 0 is returned. In case of error the function returns a value less than zero. The list of error codes can be found in the appendix.
Cross-reference	For additional information, see: 8.2. Parameter Identifiers.
Example	

6.6.17. FEUSB_AddEventHandler

Function	Installs an event handler
Syntax	**int FEUSB_AddEventHandler(int iDevHnd, FEUSB_EVENT_INIT* pInit)**
Description	The function installs one of three possible types of event handlers. This method is used when an event occurs for which the method was installed. In this way you can achieve asynchronous response to an event in an application program.

The event handling mechanism is set up for the channel identified by *iDevHnd* or globally (*iDevHnd* = 0).

At the present time only global even handling methods can be installed.

Event	Description
FEUSB_CONNECT_EVENT	Signals when the device is plugged in
FEUSB_DISCONNECT_EVENT	Signals when the device is removed

Method 1: Message to thread (only for Windows; not for Visual Basic)

This method is used for exchanging messages between threads[1]. The thread uses the API function GetCurrentThreadID() to get the thread identifier and passes it as parameter dwThreadID in the **FEUSB_EVENT_INIT** structure.

To receive the message that was sent by FEUSB with the API function PostThreadMessage(..), the thread must provide a message handling function. The message code is freely selectable.

The **FEUSB_EVENT_INIT** structure is filled as follows:

```
uiFlag = FEUSB_THREAD_ID
uiUse = FEUSB_xyz_EVENT          // see Defines FEUSB.H
uiMsg = WM_USER + ...            // freely selectable, but higher than WM_USER[2]
dwThreadID = GetCurrentThreadID()
```

The MessageMap function in the application gets the following parameters:

Event	Channel	1st Parameter (wParam)	2nd Parameter (lParam)
Connect	Not opened	0	DeviceID
	Opened	DeviceHnd	DeviceID
Disconnect	Not opened	0	0
	opened	DeviceHnd	DeviceID

---

[1] Parallel execution path independent of the application program. The application program itself is a thread.
[2] See Windows documentation for SDK platform

Method 2: Message to window (only for Windows; not for Visual Basic)

This method is used when you want to send the message directly to a window. The API function GetWindow(..)[1] is used to get the handle from the window in question and pass it as parameter hwndWnd in the **FEUSB_EVENT_INIT** structure. To receive the message that was sent by FEUSB with the API function stThreadMessage(..), the thread must provide a message handling function. The message code is freely selectable.

The **FEUSB_EVENT_INIT** structure is filled as follows:

    uiFlag = FEUSB_WND_HWND
    uiUse = FEUSB_xyz_EVENT   // see Defines FEUSB.H
    uiMsg = WM_USER + ...   // freely selectable, but higher than WM_USER[2]
    hwndWnd = GetWindow(...)

The MessageMap function gets the same parameters as for the first method.

Method 3: Invoking a callback function

The callback method is used to install a function pointer for an event. When the event occurs, the function is opened by FEUSB. The contents of the function can be freely determined. The passing parameters are specified according to Method 1.

The **FEUSB_EVENT_INIT** structure is filled as follows:

    uiFlag = FEUSB_CALLBACK
    uiUse = FEUSB_xyz_EVENT   // see Defines FEUSB.H
    uiMsg is not needed
    cbFct = (void*)&YourFunctionName[3]

The callback function gets the same parameters as for the first method.

An installed event handler must be deleted using the function **FEUSB_DelEventHandler**.

When a USB channel is closed, all the event handlers installed for it are lost.

Cross-reference	For additional information, see: <u>6.4. Event Signalling</u>
Return value	If there is no error the function returns zero, and in case of error a value less than zero. The list of error codes can be found in the appendix.
| **Example** | ```
#include "feusb.h"
...
    // Set up message handler for events
    FEUSB_EVENT_INIT Init;

    Init.hwndWnd = this->GetSafeHwnd();
    Init.uiFlag  = FEUSB_WND_HWND;
    Init.uiUse   = FEUSB_DEV_DISCONNECT_EVENT; // Message always when a device is disconnected
    Init.uiMsg   = WM_USER_DEVICE_DISCONNECT;
    FEUSB_AddEventHandler(0, &Init);

    Init.uiUse   = FEUSB_DEV_CONNECT_EVENT;         // Message always when a device is connected
    Init.uiMsg   = WM_USER_DEVICE_CONNECT;
    FEUSB_AddEventHandler(0, &Init);
``` |

---

[1] When using the MFC class CWnd, the GetSafeHwnd() can also be used
[2] See Windows documentation for SDK platform
[3] The function has the prototype: void YourFunctionName(int, unsigned int)

6.6.18. FEUSB_DelEventHandler

Function	Deletes an event handler
Syntax	**int FEUSB_DelEventHandler(int iPortHnd, FEUSB_EVENT_INIT* pInit)**

Description	The function deletes an event handler previously installed using **FEUSB_AddEventHandler**. In the **FEUSB_EVENT_INIT** structure you specify the event handler to be deleted in detail.
	
	Deleting Method 1: Message to thread (only for Windows; not for Visual Basic)
	The **FEUSB_EVENT_INIT** structure is filled as follows:
	    uiFlag = FEUSB_THREAD_ID
	    uiUse = FEUSB_xyz_EVENT // see Defines in FEUSB.H
	    uiMsg is not needed
	    dwThreadID = GetCurrentThreadID()
	
	Deleting Method 2: Message to window (only for Windows; not for Visual Basic)
	The **FEUSB_EVENT_INIT** structure is filled as follows:
	    uiFlag = FEUSB_WND_HWND
	    uiUse = FEUSB_xyz_EVENT // see Defines in FEUSB.H
	    uiMsg is not needed
	    hwndWnd = GetWindow(…)
	
	Deleting Method 3: Invoking a callback function
	The **FEUSB_EVENT_INIT** structure is filled as follows:
	    uiFlag = FEUSB_CALLBACK
	    uiUse = FEUSB_xyz_EVENT // see Defines FEUSB.H
	    uiMsg is not needed
	    cbFct = (void*)&YourFunctionName[1]

Cross-reference	For additional information, see: 6.4. Event Signalling
Return value	If there is no error the function returns zero, and in case of error a value less than zero. The list of error codes can be found in the appendix.
Example	```#include "feusb.h"\n...\n // Delete message handler for events\n FEUSB_EVENT_INIT Init;\n\n Init.hwndWnd = this->GetSafeHwnd();\n Init.uiFlag = FEUSB_WND_HWND;\n Init.uiUse = FEUSB_DEV_DISCONNECT_EVENT;\n Init.uiMsg = 0;\n FEUSB_DelEventHandler(0, &Init);\n\n Init.uiUse = FEUSB_DEV_CONNECT_EVENT;\n Init.uiMsg = 0;\n FEUSB_DelEventHandler(0, &Init);```

---

[1] The function has the prototype: void YourFunctionName(int, unsigned int)

6.6.19. FEUSB_Transceive

Function	Function for communication (transmit and receive) on a USB channel
Syntax	**int FEUSB_Transceive(int iDevHnd, char* cInterface, int iDir, UCHAR* cSendData, int iSendLen, UCHAR* cRecData, int iRecLen)**
Description	The function sends the data contained in *cSendData* with length *iSendLen* over the device interface named in *cInterface* for the connected device. The received data are stored in *cRecData*. You must specify the maximum length of the buffer *cRecData* using the parameter *iRecLen*. If the number of received characters exceeds the value passed in *iRecLen*, the function is ended with an error. This function supports two Interfaces: • *cInterface* = "OBID-RCI": USB protocol for the first FEIG USB reader (ID ISC.PRH100-U and ID ISC.MR100-U). Of cause its complexity, the handling of the protocol exchange is undocumented. The communication with the reader is only possible with the function library ID FEISC. This interface is not supported by Linux. The parameter *iDir* determines the data direction: *iDir* = 0x01 IN-Transfer (host gets data from the device) *iDir* = 0x02 OUT-Transfer (host sends data to the device) • *cInterface* = "OBID-RCI2": USB protocol of second generation for FEIG USB reader. The protocol layout is identical with the protocol frame as it is documented in the system manual of the reader. The parameter *iDir* has no function and can be set to zero.
Interfaces	OBID-RCI, OBID-RCI2
Return value	If there are no errors, the function returns the length of the receive protocol, and in case of error a value less than zero. The list of error codes can be found in the appendix.
Example	

6.6.20. FEUSB_Transmit

Function	Function for communication on a USB channel
Syntax	**int FEUSB_Transmit(int iDevHnd, char* cInterface, UCHAR* cSendData, int iSendLen)**
Description	The function sends the data contained in *cSendData* with length *iSendLen* over the device interface named in *cInterface* for the connected device. Only *cInterface* = "OBID-RCI2" is supported.
Interfaces	OBID-RCI2
Return value	If there are no errors, the function returns zero. The list of error codes can be found in the appendix.
Example	

6.6.21. FEUSB_Receive

Function	Function for communication on a USB channel
Syntax	**int FEUSB_Receive(int iDevHnd, char* cInterface, UCHAR* cRecData, int iRecLen)**
Description	The function expects data over the device interface named in *cInterface* for the connected device. The received data are stored in *cRecData*. You must specify the maximum length of the buffer *cRecData* using the parameter *iRecLen*. If the number of received characters exceeds the value passed in *iRecLen*, the function is ended with an error. Only *cInterface* = "OBID-RCI2" is supported.
Interfaces	OBID-RCI2
Return value	If there are no errors, the function returns the length of the receive protocol, and in case of error a value less than zero. The list of error codes can be found in the appendix.
Example	

7. Dynamic Linking Under C++

If you want to link the FEUSB function collection dynamically to the application the library file must be loaded explicitly. A (program-related) disadvantage is that each function invoke to the DLL must then be done using a function pointer.

You will need to perform the following steps:

1. Load the DLL for run-time library

```
HMODULE hLib = LoadLibrary("feusb.dll");
```

2. Get a function pointer for run-time:

```
LPFN_FEUSB_GET_DLL_VERSION lpfn= GetProcAddress(hLib,

                                        "FECOM_GetDLLVersion");
```

3. Run the functions:

```
char cVersion[256];

lpfn(cVersion);
```

4. The loaded DLL must be freed again before quitting the application:

```
 if(hLib != NULL)

   FreeLibrary(hLib);
```


A prototype for a function point is defined for each function in the FEUSB in the header file feusb.h. In the example above this is LPFN_FEUSB_GET_DLL_VERSION. A practical approach is to get and save the required function pointer for the entire run-time of the program.

Tip: If you are using the C++ class library ID FEDM, you can use the function GetFeUsbFunction of the base class FEDM_Base to get the pointer. Then the DLL is automatically loaded the first time the function is invoked and then removed from the address space of the application in the class destructor. As a parameter you pass a constant to the function GetFeUsbFunction that identifies the function. These constants are defined in the header file feusb.h.

Example:

```
FEDM_ISCReader m_Reader; // FEDM_ISCReader is derived from FEDM_Base

LPFN_FEUSB_GET_ERROR_TEXT lpfn = NULL;

lpfn = (LPFN_FEUSB_GET_ERROR_TEXT)

        m_Reader.GetFeUsbFunction(FEUSB_GET_ERROR_TEXT);

if(lpfn != NULL)

  lpfn(iErrorCode, cErrorCode);
```

8. Appendix

8.1. Error Codes

Error constant	Value	Description
FEUSB_ERR_EMPTY_DEVICELIST	-1100	Device handle is empty (no device objects stored)
FEUSB_ERR_EMPTY_SCANLIST	-1101	Scan list is empty (no USB devices available)
FEUSB_POINTER_IS_NULL	-1102	A passed pointer is NULL
FEUSB_ERR_NO_MORE_MEM	-1103	Insufficient memory
FEUSB_ERR_SET_CONFIGURATION	-1104	The USB configuration could not be set
FEUSB_ERR_KERNEL	-1105	An error occurred in the kernel driver during USB transfer
FEUSB_ERR_UNSUPPORTED_OPTION	-1106	Unsupported option
FEUSB_ERR_UNSUPPORTED_FUNCTION	-1107	Unsupported function
FEUSB_ERR_NO_FEIG_DEVICE	-1110	USB device has no FEIG identifier
FEUSB_ERR_SEARCH_MISMATCH	-1111	No device(s) with the specified search criteria were found
FEUSB_ERR_NO_DEVICE_FOUND	-1112	No device(s) was/were found
FEUSB_ERR_DEVICE_IS_SCANNED	-1113	The device is already in the scan list
FEUSB_ERR_SCANLIST_OVERFLOW	-1114	Scan list is filled with 127 entries and an attempt was made to add another
FEUSB_ERR_UNKNOWN_HND	-1120	The passed device handle is unknown
FEUSB_ERR_HND_IS_NULL	-1121	The passed device handle is 0
FEUSB_ERR_HND_IS_NEGATIVE	-1122	The passed device handle is negative
FEUSB_ERR_NO_HND_FOUND	-1123	No device handle found in device handle list
FEUSB_ERR_TIMEOUT	-1130	Timeout when reading USB channel
FEUSB_ERR_NO_SENDDATA	-1131	No send data passed
FEUSB_ERR_UNKNOWN_INTERFACE	-1132	Unknown interface
FEUSB_ERR_UNKNOWN_DIRECTION	-1133	Unknown data direction
FEUSB_ERR_RECBUF_TO_SMALL	-1134	Receive buffer is too small
FEUSB_ERR_SENDDATA_LEN	-1135	Length of send data incorrectly indicated
FEUSB_ERR_UNKNOWN_DESCRIPTOR_TYPE	-1136	Unknown descriptor type
FEUSB_ERR_DEVICE_NOT_PRESENT	-1137	USB device not presently connected to USB port
FEUSB_ERR_DEVICE_NOT_SCANNED	-1140	Device was not previously scanned
FEUSB_ERR_DEVHND_NOT_IN_SCANLIST	-1141	Device not entered in scan list
FEUSB_ERR_DRIVERLIST	-1142	No driver list could be created in the USB driver

Error constant	Value	Description
FEUSB_ERR_UNKNOWN_PARAMETER	-1150	Pass parameter is unknown
FEUSB_ERR_PARAMETER_OUT_OF_RANGE	-1151	Pass parameter is too large or too small
FEUSB_ERR_ODD_PARAMETERSTRING	-1152	An unsupported option was invoked by the pass parameter
FEUSB_ERR_INDEX_OUT_OF_RANGE	-1153	The passed list index is not in the value range of 1...65535
FEUSB_ERR_UNKNOWN_SCANOPTION	-1154	Unknown scan option
FEUSB_ERR_UNKNOWN_ERRORCODE	-1155	Unknown error code
FEUSB_ERR_DEV_DESC_LENGTH	-1160	Length error in device descriptor
FEUSB_ERR_CFG_DESC_LENGTH	-1161	Length error in configuration descriptor
FEUSB_ERR_INTF_DESC_LENGTH	-1162	Length error in interface descriptor
FEUSB_ERR_ENDP_DESC_LENGTH	-1163	Length error in endpoint descriptor
FEUSB_ERR_HID_DESC_LENGTH	-1164	Length error in HD descriptor
FEUSB_ERR_STRG_DESC_LENGTH	-1165	Length error in string descriptor
FEUSB_ERR_READ_DEV_DESCRIPTOR	-1166	Device descriptor read error
FEUSB_ERR_READ_CFG_DESCRIPTOR	-1167	Configuration descriptor read error
FEUSB_ERR_READ_STRG_DESCRIPTOR	-1168	String descriptor read error
FEUSB_ERR_MAX_INTERFACES	-1170	The device has too many interfaces
FEUSB_ERR_MAX_ENDPOINTS	-1171	The device has too many endpoints
FEUSB_ERR_MAX_STRINGS	-1172	The device has too many strings

8.2. Parameter Identifiers

Parameter identifier	Value range	Default	Unit	Description
DeviceHnd	268435456... 536870911	-	-	Device handle Use: • FEUSB_GetScanListPara
Device-ID	Hexadecimal: 0x00000001 ... 0xFFFFFFFF Decimal: 1 ... 4294967295	-	-	Serial number of USB device Use: • FEUSB_GetScanListPara • FEUSB_GetDevicePara
Timeout	0...99999	1000	ms	Maximum wait time for receive protocol. Use: • FEUSB_SetDevicePara
FamilyName	s. 8.5. cFamilyName in the FEUSB_SCANSEARCH Structure	-	-	Name of reader family Use: • FEUSB_GetScanListPara • FEUSB_GetDevicePara
DeviceName	s. 8.6. cDeviceName in the FEUSB_SCANSEARCH Structure	-	-	Name of reader Use: • FEUSB_GetScanListPara • FEUSB_GetDevicePara
Interface	OBID-RCI, OBID-RCI2	-	-	Use: • FEUSB_GetScanListPara
TypeNumber		-	-	4 byte device type number Use: • FEUSB_GetDevicePara
Present	0, 1	-	-	Query device presence on USB channel Use: • FEUSB_GetDevicePara
ResetPipe	No value			Reset the end points for bulk transfer Use: • FEUSB_SetDevicePara Only for Linux
ResetDevice	No value			Perform a USB port reset and reinitialize the device Use: • FEUSB_SetDevicePara Only for Linux
ExclusiveAccess	0, 1	1	0	Activates (1) or deactivates (0) the exclusive access. This setting is global for all USB reader. Use: • FEUSB_SetDevicePara Only for Windows

8.3. Constants for the FEUSB_EVENT_INIT Structure

The constant definitions are contained in the file FEUSB.H and FEUSB.BAS.

Constant	Value	Use	Description
FEUSB_THREAD_ID	1	uiFlag	Event signaling with thread message
FEUSB_WND_HWND	2	uiFlag	Event signaling with window message
FEUSB_CALLBACK	3	uiFlag	Event signaling with callback function
FEUSB_CONNECT_EVENT	1	uiUse	Signaling when reader is connected
FEUSB_DISCONNECT_EVENT	2	uiUse	Signaling when reader is disconnected

8.4. Constants for the FEUSB_SCANSEARCH Structure

The constant definitions are contained in FEUSB.H, FEUSB.BAS and FEUSB.PAS.

Constant	Value	Use	Description
FEUSB_SEARCH_FAMILY	1	iMask	cFamilyName is searched
FEUSB_SEARCH_PRODUCT	2	iMask	cDeviceName is searched
FEUSB_SEARCH_DEVICEID	4	iMask	dwDeviceID is searched

8.5. cFamilyName in the FEUSB_SCANSEARCH Structure[1]

String	Description
"OBID i-scan Proximity"	
"OBID i-scan Midrange"	
"OBID i-scan UHF Midrange"	
"OBID i-scan UHF Long-Range"	
"OBID classic-pro"	

8.6. cDeviceName in the FEUSB_SCANSEARCH Structure[2]

String	Description
"ID ISC.PRH100-U"	Reader in the OBID i-*scan* Proximity family
"ID ISC.PRH101-U"	Reader in the OBID i-*scan* Proximity family
"ID ISC.MR100-U"	Reader in the OBID i-*scan* Midrange family
"ID ISC.MR101-U"	Reader in the OBID i-*scan* Midrange family
"ID ISC.MRU200"	Reader in the OBID i-*scan* UHF Midrange family
"ID CPR.04-USB"	Reader in the OBID *classic-pro* family
"ID CPR40.xx-U"	Reader in the OBID *classic-pro* family

---

[1] List to be expanded in the future
[2] List to be expanded in the future

8.7. Revision History

<u>V5.01.00</u>

- Windows: Discontinued support for uiFlag values FEUSB_THREAD_ID and FEUSB_WND_HWND in struct **FEUSB_EVENT_INIT**:

<u>V5.00.00</u>

- Discontinued support for Windows XP

- Discontinued support for CE 5

- Windows: bug fix for event notification

- Linux: migration from libusb 0.1.x to 1.0.x

<u>V4.02.06</u>

- - Windows: workaround for ID ISC.MRU200 because of extended request of string descriptors

- - Windows: deactivating of the plug-and-play thread with file feusb.conf (s. 3.5. Deactivating the Plug-and-Play Thread)

- - Linux: 1st version for 64-Bit

<u>V4.02.02</u>

- Windows / Windows CE:

  1. Bug fix when detecting USB-Sticks.

<u>V4.02.00</u>

- Windows:

  1. Migration of the development environment from Visual Studio 2008 to Visual Studio 2010.

  2. Improved thread safeness

  3. DLL without MFC

  4. First release of 64-Bit version

  5. Dynamic binding to Log-Manager

  6. Modification in internal plug-and-play mechanism for improved event handling

- Windows CE:

  1. Improved thread safeness

- Linux:

  1. No changes

- First release for Mac OS X, V10.7.3 or higher

V4.00.00

- Windows / Windows CE:

  1. Migration of the development environment from Visual Studio 6 to Visual Studio 2008.

  2. Adaptation of the callback declarations in *struct _FEUSB_EVENT_INIT* concerning the calling convention. Thus, this version of FEUSB is not compatible with the previous version and with applications compiled against the previous version of FEUSB. Code modifications are not necessary, but re-compilation of applications is mandatory.

  3. New error code -1138 (Error while transmit data)

  4. Extended internal error handling

  5. Bug fix (only for Windows CE): transformation of the Device-ID from Unicode string into an unsigned long value

- Linux:

  1. Bug fix for deactivating of the plug-and-play thread with file feusb.conf

V3.06.01

  Windows, Windows CE

- Extended internal error handling

- Support of unified Device-ID (for Linux already realized)

V3.05.00

- Ready for **Windows 7** (x86 and x64) by use of the kernel driver OBIDUSB V2.50

- **Linux**: rule file **41-feig.rules** enables an installation without root rights

V3.04.00

- **Windows** and **Windows CE**: modification in internal plug-and-play mechanism for improved event handling

V3.03.04

- New function **FEUSB_GetDrvVersion** for request of information about installed kernel-driver

- Request of error messages to error codes from kernel-driver

- Error correction for Windows 2000: limitation of every transfer to 4096 bytes.

V3.03.02

- Exclusive use of a USB reader with one applikation. This is a modification against former versions, where multiple applications could share one USB reader.

  This setting can be changed by using the parameter "ExclusiveAccess" with the function **FEUSB_SetDevicePara**.

- Support of bulk transfer for new reader generations

V3.00.00

- Compatibility with new kernel driver for Vista

V2.05.00

- First Linux version.

V2.03.02

- The new version is 100% backward compatible with the previous version.

- Bug fix for DeviceID > 0x7FFFFFFF

- Translation of the error code 0xE000100C in FEUSB_ERR_TIMEOUT (-1130)

V2.03.01

- The new version is 100% backward compatible with the previous version.

- Support for new USB protocols.

- New functions: **FEUSB_Transmit**, **FEUSB_Receive**, **FEUSB_SetDevicePara**

- New parameter: TIMEOUT

- New error code: -1106

V2.01.00

- Bug fix: communication hang-up after a previously failed communication.

V2.00.00

- The new version supports Windows XP.

- The kernel driver OBIDUSB.SYS and OBIDUSB9.SYS must be Version 2.00.

- New error codes: -1166, -1167, -1168

- New parameters for the function **FEUSB_GetDevicePara**: DeviceName, FamilyName

- New parameters for the function **FEUSB_GetScanListPara**: DeviceName, FamilyName

- Connect messaging to applications with device ID.

V1.02.00

- The new version is nearly 100% backward compatible with the previous version 1.00.00. The only incompatibility is in the value shift of the scan parameter FEUSB_SCAN_ALL from 0x03 to 0x0F.

- Messaging to applications also for readers which are not yet entered in the scan list.

- The device handle now has an offset of 0x10000000 (decimal 268435456) for direct use with other FEIG DLLs (e.g., FEISC.DLL).

- The new scan parameter FEUSB_SCAN_NEW allows searching for unscanned readers.

- New function: **FEUSB_GetLastError**.

- New error code: -1114 for scan list overflow.

V1.01.00

- Internal version

V1.00.00

- Due to a new kernel driver, this version is no longer compatible with previous versions.

- New functions: FEUSB_AddEventHandler, FEUSB_DelEventHandler, FEUSB_IsDevicePresent

V0.99.01

- Support for Windows 98, 98SE and 2000

- Support for open host controller interface (OHCI)and universal host controller interface (UHCI)

- New error code: -1103