OBID®
Welcome to RFID

HANDBUCH

ID FEUSB

Version 4.02.06 (Windows)

Version 4.02.06 (Windows CE)

Version 4.02.00 (Linux)

Software-Support für USB

Universal Serial Bus

Betriebssystem	Ausführung		Anmerkungen
	32-Bit	64-Bit	
Windows XP	X	(X)	bei 64-Bit nur mit 32-Bit Laufzeitsystem
Windows Vista / 7 / 8	X	X	
Windows CE	X	-	
Linux	X	X	
Apple Max OS X	-	X	ab V10.7.3, Architektur x86_64

final
public (B)
2014-03-17
H00501-18d-ID-B.doc

FEIG ELECTRONIC

Hinweis

© Copyright 2000-2014 by FEIG ELECTRONIC GmbH
Lange Straße 4
D-35781 Weilburg-Waldhausen
eMail: info@feig.de
internet: http://www.feig.de

OBID® ist eingetragenes Warenzeichen der FEIG ELECTRONIC GmbH.

Alle früheren Ausgaben verlieren mit diesem Handbuch ihre Gültigkeit.
Die Angaben in diesem Handbuch können ohne vorherige Ankündigung geändert werden.

Weitergabe sowie Vervielfältigung dieses Dokuments, Verwertung und Mitteilung ihres Inhalts sind nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlung verpflichtet zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmuster-Eintragung vorbehalten.

Die Zusammenstellung der Informationen in diesem Handbuch erfolgt nach bestem Wissen und Gewissen. FEIG ELECTRONIC GmbH übernimmt keine Gewährleistung für die Richtigkeit und Vollständigkeit der Angaben in diesem Handbuch. Insbesondere kann FEIG ELECTRONIC GmbH nicht für Folgeschäden aufgrund fehlerhafter oder unvollständiger Angaben haftbar gemacht werden. Da sich Fehler, trotz aller Bemühungen nie vollständig vermeiden lassen, sind wir für Hinweise jederzeit dankbar.

FEIG ELECTRONIC GmbH übernimmt keine Gewährleistung dafür, dass die in diesem Dokument enthaltenden Informationen frei von fremden Schutzrechten sind. FEIG ELECTRONIC GmbH erteilt mit diesem Dokument keine Lizenzen auf eigene oder fremde Patente oder andere Schutzrechte.

Die in diesem Handbuch gemachten Installationsempfehlungen gehen von günstigsten Rahmenbedingungen aus. FEIG ELECTRONIC GmbH übernimmt keine Gewähr für die einwandfreie Funktion einer OBID®-Anlage in systemfremden Umgebungen.

OBID® and OBID i-*scan*® are registered trademarks of FEIG ELECTRONIC GmbH.

Windows is a registered trademark of Microsoft Corporation in the United States and other countries.

Windows Vista is either a registered trademark or trademark of Microsoft Corporation in the United States and/or other countries

Linux® is a registered Trademark of Linus Torvalds.

Apple, Mac, Mac OS, OS X, Cocoa and Xcode are trademarks of Apple Inc., registered in the U.S. and other countries

Lizenzvertrag über die Nutzung der Software

Dies ist ein Vertrag zwischen Ihnen und der FEIG ELECTRONIC GmbH (nachfolgend "FEIG") über die Nutzung der überlassenen Programmbibliothek ID FEUSB und die vorliegende Dokumentation, nachfolgend Lizenzmaterial genannt. Mit der Installation und Benutzung der Software erklären Sie sich mit allen Bestimmungen dieses Vertrages ausnahmslos und ohne Einschränkung einverstanden. Wenn Sie mit den Bestimmungen dieses Vertrages nicht oder nicht vollständig einverstanden sind, dürfen Sie das Lizenzmaterial nicht installieren oder anderweitig benutzen. Das überlassene Lizenzmaterial ist Eigentum der FEIG ELECTRONIC GmbH und ist international urheberrechtlich geschützt.

§1 Vertragsgegenstand und Vertragsumfang

1. FEIG gewährt Ihnen das Recht, das überlassene Lizenzmaterial zu installieren und zu den nachstehenden Bedingungen zu nutzen.
2. Sie dürfen sämtliche Bestandteile des Lizenzmaterials auf einer Festplatte oder einem sonstigen Speichermedium installieren. Die Installation und Nutzung darf auch auf einem Netzwerk-Fileserver erfolgen. Sie dürfen Sicherheitskopien des Lizenzmaterials anfertigen.
3. FEIG gewährt Ihnen das Recht die dokumentierte Programmbibliothek für die Entwicklung eigener Anwendungsprogramme oder Programmbibliotheken zu verwenden und Sie dürfen die Laufzeitdatei FEUSB.DLL, FEUSBCE.DLL, LIBFEUSB.x.y.z.DYLIB[1] oder LIBFEUSB.SO.x.y.z[1] ohne Abgabe von Lizenzgebühren vertreiben, unter der Voraussetzung, dass diese Anwendungsprogramme oder Programmbibliotheken nur zusammen mit von FEIG entwickelten USB-Geräten verwendet werden.
4. FEIG gewährt Ihnen nicht das Recht die mit USB-Geräten mitgelieferten USB-Treiberdateien für Windows (OBIDUSB.SYS, OBIDUSB9.SYS, OBIDUSB.INF) separat zu vertreiben. Diese USB-Treiberdateien dürfen nur zusammen mit FEIG USB-Geräten weitervertrieben werden.

§2 Schutz des Lizenzmaterials

1. Das Lizenzmaterial ist geistiges Eigentum von FEIG und seinen Lieferanten. Es ist gemäß Urheberrecht, internationalen Verträgen und einschlägigen Gesetzen des Landes geschützt, in dem sie genutzt wird. Struktur, Organisation und Code der Software sind wertvolles Geschäftsgeheimnis und vertrauliche Information von FEIG und seinen Lieferanten.
2. Sie verpflichten sich, die Programmbibliothek sowie die Dokumentation nicht zu ändern, anzupassen, zu übersetzen, rückzuentwickeln, zu dekompilieren, zu disassemblieren oder auf andere Weise zu versuchen, den Quellcode dieser Software herauszufinden.
3. Soweit FEIG im Lizenzmaterial Schutzvermerke, wie Copyright-Vermerke und andere Rechtsvorbehalte angebracht hat, sind Sie verpflichtet, diese unverändert beizubehalten sowie in alle von Ihnen hergestellten vollständigen oder teilweisen Kopien in unveränderter Form zu übernehmen.
4. Die Weitergabe von Lizenzmaterial ist weder vollständig noch auszugsweise gestattet, solange dazu keine explizite anderslautende Vereinbarung zwischen Ihnen und FEIG getroffen wurde. Nicht betroffen von dieser Regelung sind solche Anwendungsprogramme oder Programmbibliotheken, die gem. §1 Absatz 3. dieser Vereinbarung erstellt und vertrieben werden.

§3 Gewährleistung und Haftungsbeschränkungen

1. Sie stimmen mit FEIG darüber überein, dass es nicht möglich ist, EDV-Programme so zu entwickeln, dass sie für alle Anwendungsbedingungen fehlerfrei sind. FEIG weist Sie ausdrücklich darauf hin, dass die Installation eines neuen Programms bereits vorhandene Software beeinflussen kann, und zwar auch solche Software, die nicht gleichzeitig mit der neuen Software ausgeführt wird. FEIG haftet in keinem Fall für direkte oder indirekte Schäden, für Folgeschäden oder Sonderschäden, Einschließlich entgangenen Geschäftsgewinn oder entgangener Einsparungen. Wenn Sie sicherstellen wollen, dass es zu keinerlei Beeinflussung eines bereits installierten Programms kommt, dürfen Sie die vorliegende Software nicht installieren.
2. FEIG weist ausdrücklich darauf hin, dass mit der Software irreversible Einstellungen und Anpassungen an Geräten vorgenommen werden können, wodurch diese Geräte zerstört oder unbrauchbar gemacht werden können. FEIG übernimmt für derartiges Handeln unabhängig davon ob dies bewußt oder unbewußt erfolgte keinerlei Gewährleistung.
3. FEIG liefert Ihnen die Software "wie besehen" ohne jegliche Gewährleistung. FEIG kann für die Leistung oder die Ergebnisse, die Sie durch die Nutzung der Software erzielen, nicht garantieren. FEIG übernimmt keine Gewährleistung oder Garantie dafür, dass keine Schutzrechte Dritter verletzt werden, auch nicht dafür, dass die Software für irgendeinen bestimmten Zweck geeignet ist.
4. FEIG weist ausdrücklich darauf hin, dass das Lizenzmaterial nicht für den Einsatz mit oder in medizinischen Geräten oder für Geräte für lebenserhaltende Maßnahmen konzipiert ist, bei denen ein Fehler eine Gefahr für menschliches Leben oder für die gesundheitliche Unversehrtheit zur Folge haben kann.

[1] x.y.z repräsentiert die Versionsnummer

Der Anwender des Lizenzmaterials ist dafür verantwortlich, geeignete Maßnahmen zu ergreifen um Gefahren, Schäden oder Verletzungen zu vermeiden.

§4 Schlußbestimmungen

1. Dieser Vertrag enthält die vollständigen Lizenzbestimmungen und ersetzt alle eventuell vorangegangenen Regelungen und Absprachen. Änderungen und Ergänzungen bedürfen der Schriftform.
2. Sollte eine der in diesem Vertrag enthaltenen Bestimmungen unwirksam sein oder werden, so wird die Gültigkeit der übrigen Bestimmungen hierdurch nicht berührt. Beide Vertragsparteien verpflichten sich, die unwirksame Bestimmung durch eine solche wirksame Bestimmung zu ersetzen, die dem wirtschaftlichem Zweck der zu ersetzenden Bestimmung am nächsten kommt.
3. Dieser Vertrag unterliegt dem Recht der Bundesrepublik Deutschland. Gerichtsstand ist Frankfurt a. M.

Inhalt:

1. Einleitung 7

1.1. Lieferumfang 9

1.1.1. Windows XP / Vista / 7 / 8 9
1.1.2. Windows CE 9
1.1.3. Linux 10
1.1.4. Mac OS X 10

2. Änderungen gegenüber der Vorversion 11

3. Installation 12

3.1. 32-und 64-Bit Windows XP/Vista/7/8 12

3.2. Windows CE 13

3.3. 32- und 64-Bit Linux 14

3.3.1. libusb 15

3.4. 64-Bit Mac OS X 16

3.4.1. libusb 16

3.5. Deaktivieren des Plug-and-play Threads 17

4. Einbindung in das Anwendungsprogramm 18

4.1. Unterstützte Entwicklungsumgebungen 18

4.2. Einbindung in Visual Studio 18

4.3. Einbindung in Xcode 18

5. Eine Kurzeinführung in USB 19

6. Programmierschnittstelle 21

6.1. Übersicht 21

6.2. Threadsicherheit 22

6.3. Aufbau und Funktion der Scanliste 23

6.4. Ereignissignalisierung 24

6.5. Liste der Funktionen 25

6.6. Funktionsbeschreibungen 26

6.6.1. FEUSB_GetDLLVersion 26
6.6.2. FEUSB_GetDrvVersion (nur für Windows) 26
6.6.3. FEUSB_GetErrorText 27
6.6.4. FEUSB_GetLastError 27
6.6.5. FEUSB_Scan 28

6.6.6. FEUSB_ScanAndOpen 30
6.6.7. FEUSB_GetScanListPara 31
6.6.8. FEUSB_GetScanListSize 32
6.6.9. FEUSB_ClearScanList 32
6.6.10. FEUSB_OpenDevice 33
6.6.11. FEUSB_CloseDevice 34
6.6.12. FEUSB_IsDevicePresent 34
6.6.13. FEUSB_GetDeviceList 35
6.6.14. FEUSB_GetDeviceHnd 36
6.6.15. FEUSB_GetDevicePara 37
6.6.16. FEUSB_SetDevicePara 38
6.6.17. FEUSB_AddEventHandler 39
6.6.18. FEUSB_DelEventHandler 41
6.6.19. FEUSB_Transceive 42
6.6.20. FEUSB_Transmit 43
6.6.21. FEUSB_Receive 43

7. Dynamische Bindung unter C++ 44

8. Anhang 45

8.1. Fehlercodes 45

8.2. Liste der Parameterkennungen 47

8.3. Liste der Konstanten für die FEUSB_EVENT_INIT-Struktur 48

8.4. Liste der Konstanten für die FEUSB_SCANSEARCH-Struktur 49

8.5. Liste der cFamilyName in der FEUSB_SCANSEARCH-Struktur 49

8.6. Liste der cDeviceName in der FEUSB_SCANSEARCH-Struktur 49

8.7. Änderungshistorie 51

1. Einleitung

Das Supportpaket ID FEUSB dient zur Unterstützung bei der Programmierung von Kommunikations-orientierter Software mit Datentransport über USB und unterstützt die Sprachen ANSI-C, ANSI-C++ und prinzipiell jede andere Sprache, die C-Funktionen aufrufen kann.

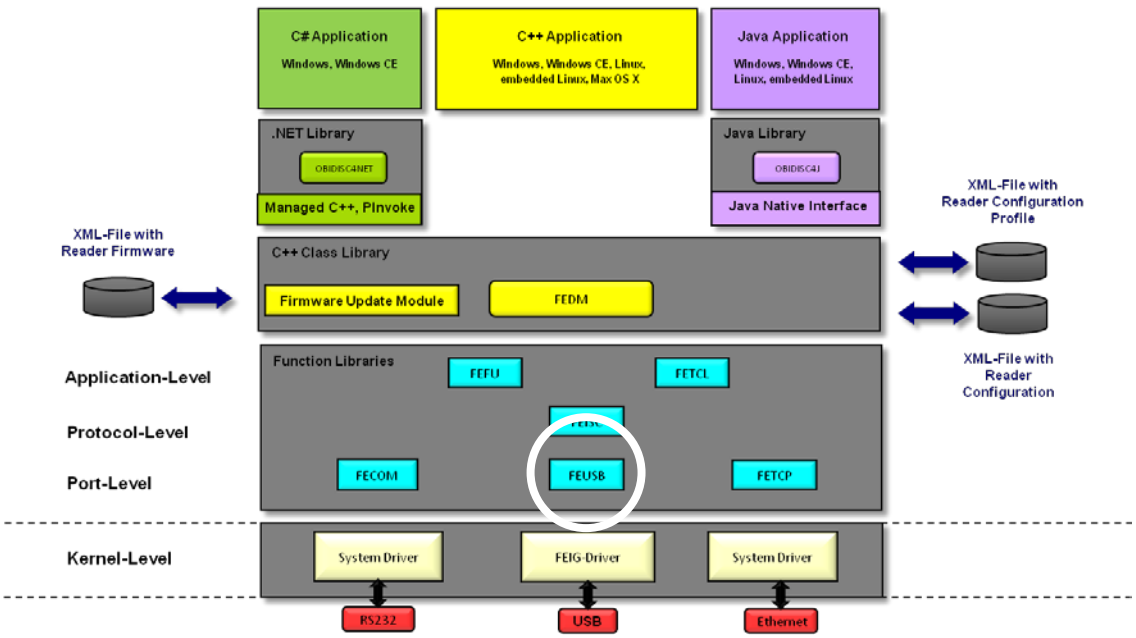
Mit dem Supportpaket wird eine einfache, Geräte-unabhängige Funktionsschnittstelle zu USB-Geräten der OBID®-Familie für die unterstützten Betriebssysteme angeboten. Üblicherweise wird die FEUSB mit einer weiteren, Geräte-spezifischen Funktionssammlung (z.B. ID FEISC) kombiniert.

Die Funktionssammlung kann prinzipiell keine anderen USB-Geräte als die aus der OBID®-Familie unterstützen.

Verwendet werden kann die Bibliothek mit folgenden Betriebssystemen:

Betriebssystem	Ausführung		Anmerkungen
	32-Bit	64-Bit	
Windows XP	X	(X)	bei 64-Bit nur mit 32-Bit Laufzeitsystem
Windows Vista / 7 / 8	X	X	
Windows CE	X	-	
Linux	X	X	
Apple Max OS X	-	X	ab V10.7.3, Architektur x86_64

Die Bibliothek FEUSB bildet die erste Ebene in dem mehrschichtigen, hierarchisch strukturierten Aufbau von FEIG-Bibliotheken. Mit ihr wird ausschließlich die Transportschicht zum USB-Treiber des Betriebssystems realisiert. Das nachfolgende Bild zeigt eine Übersicht über alle Bibliotheken.



Programmierer, die sich für diese Schicht als Integrationsoption entscheiden, müssen das Protokollhandling (Aufbau/Zerlegung von Protokollrahmen, CRC-Prüfung, Längenprüfung) in Ihrer

Applikation implementieren. Dadurch entsteht erheblicher Programmieraufwand und es gilt abzuwägen, ob der Einstieg auf diesem Level zwingend notwendig ist.

Wer nur auf die Funktionsbibliotheken zurückgreifen muss oder möchte, sollte die Bibliothek FEISC als nächst höheres API wählen.

1.1. Lieferumfang

Dieses Supportpaket besteht aus den nachfolgend aufgelisteten Dateien. In der Regel wird das Paket mit anderen Bibliotheken in einem speziell für das jeweilige Betriebssystem zusammengestellten Software Development Kit (SDK) – z.B. ID ISC.SDK.Win - ausgeliefert.

1.1.1. Windows XP / Vista / 7 / 8

Datei	Verwendung
FEUSB.DLL	DLL mit allen Funktionen
FEUSB.LIB	LIB-Datei zum Linken für C/C++-Projekte
FEUSB.H	Header-Datei für C/C++-Projekte

Zusätzlich werden folgende Treiberdateien benötigt, die sich auf der Treiber-CD (im Lieferumfang zum USB-Gerät) befinden bzw. über den Download-Server erhältlich sind.

Datei	Verwendung
OBIDUSB.SYS (V 2.50)	WHQL-zertifizierter 32- und 64-Bit Windows-Kernel-Treiber (XP/Vista/7/8) für OBID®-Leser mit USB-Schnittstelle
OBIDUSB.INF	Inf-Datei zur Treiberinstallation

1.1.2. Windows CE

Datei	Verwendung
FEUSBCE.DLL	DLL mit allen Funktionen
FEUSBCE.LIB	LIB-Datei zum Linken für C/C++-Projekte
FEUSB.H	Header-Datei für C/C++-Projekte

Zusätzlich ist ein speziell für die Windows CE Plattform kompilierter Treiber notwendig, der separat bestellt werden muss.

1.1.3. Linux

Datei[1]	Verwendung
LIBFEUSB.SO.x.y.z	Funktions-Bibliothek mit allen Funktionen
FEUSB.H	Header-Datei für C/C++-Projekte

Anmerkung:
LIBFEUSB basiert auf der Open-Source Entwicklung libusb in der Version 0.1.12, die nicht Bestandteil dieses Paketes ist. libusb kann unter http://libusb.sourceforge.net bezogen werden und muss separat installiert werden.

1.1.4. Mac OS X

Datei[2]	Verwendung
LIBFEUSB.SO.x.y.z	Funktions-Bibliothek mit allen Funktionen
FEUSB.H	Header-Datei für C/C++-Projekte

Anmerkung:
LIBFEUSB basiert auf der Open-Source Entwicklung libusb in der Version 0.1.13 beta, die nicht Bestandteil dieses Paketes ist. Das Binary von libusb kann unter http://www.ellert.se/twain-sane/ bezogen werden und muss separat installiert werden.

[1] x.y.z repräsentiert die Versionsnummer der Bibliotheksdatei

[2] x.y.z repräsentiert die Versionsnummer der Bibliotheksdatei

2. Änderungen gegenüber der Vorversion

- Windows:

  1. Workaround für ID ISC.MRU200 wegen erweiterter Abfrage von String-Deskriptoren
  2. Deaktivierung des Plug&Play-Threads mit Datei feusb.conf (s. 3.5. Deaktivieren des Plug-and-play Threads)

- Windows CE

  Keine Änderungen

- Linux:

  Version für 64-Bit

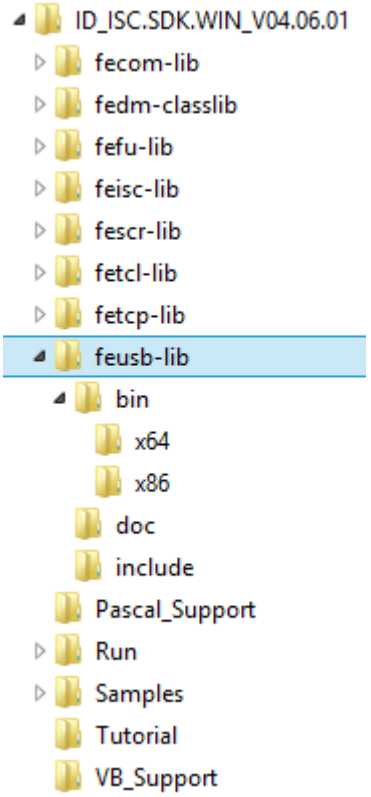
Bitte beachten Sie auch die Änderungshistorie im Anhang.

3. Installation

Das Supportpaket wird in der Regel mit einem Software Development Kit (SDK) ausgeliefert. Kopieren Sie das SDK in ein Verzeichnis Ihrer Wahl.

Die Dateien dieses Supportpakets finden sich im Verzeichnis feusb-lib.

3.1. 32-und 64-Bit Windows XP/Vista/7/8

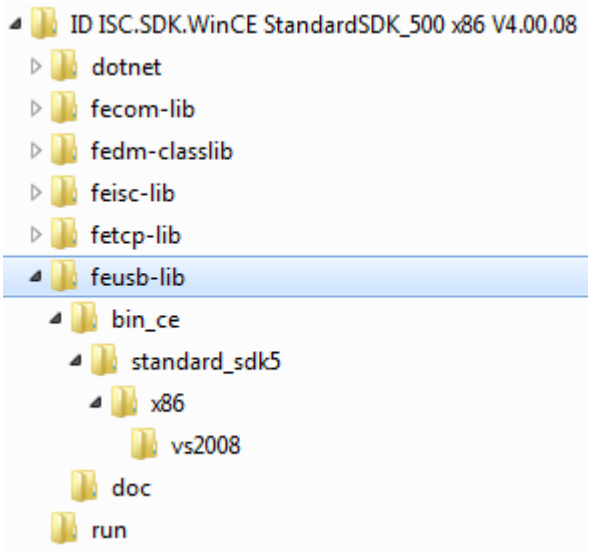


Wenn eigene Projekte nicht im SDK-Verzeichnis angelegt werden sollen, dann empfiehlt sich folgende Vorgehensweise:

- Kopieren Sie FEUSB.DLL in das Verzeichnis des Anwendungsprogramms (empfohlen) oder in das Systemverzeichnis von Windows.
- Kopieren Sie FEUSB.LIB in das Projekt- oder LIB-Verzeichnis
- Kopieren Sie FEUSB.H in das Projekt- oder INCLUDE-Verzeichnis

Die Treiberinstallation des OBIDUSB Kerneltreibers muss **vor** dem ersten Einstecken eines FEIG USB-Gerätes erfolgen. Die Installation wird von einem Assistenten des Betriebssystems durchgeführt. Nähere Hinweise dazu finden Sie im Begleitdokument zum USB-Treiber.

3.2. Windows CE



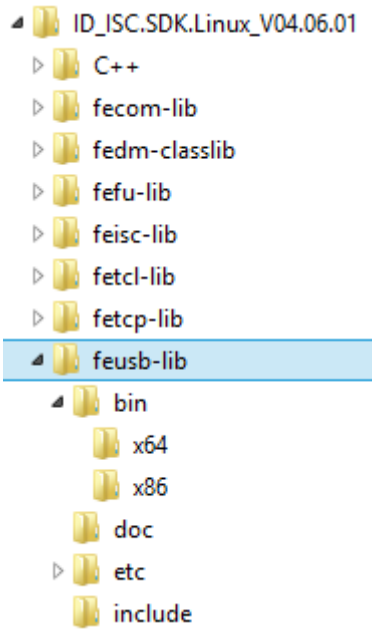
Wenn eigene Projekte nicht im SDK-Verzeichnis angelegt werden sollen, dann empfiehlt sich folgende Vorgehensweise:

- Kopieren Sie die Datei FEUSBCE.DLL in das Systemverzeichnis des Windows CE Rechners.
- Kopieren Sie FEUSBCE.LIB in das Projekt- oder LIB-Verzeichnis.
- Kopieren Sie FEUSB.H in das Projekt- oder INCLUDE-Verzeichnis

Die Treiberinstallation des OBIDUSB Kerneltreibers muss vor dem ersten Einstecken eines FEIG USB-Gerätes erfolgen. Installationshinweise werden mit dem Treiber mitgeliefert.

Hinweis: die DLL kann nicht mit eMbedded Visual Basic 3.0 verwendet werden.

3.3. 32- und 64-Bit Linux



Zur Installation gibt es zwei Optionen:

Option 1: Falls eine install.sh im SDK-Verzeichnis vorliegt, führen Sie diese aus. Damit werden alle Bibliotheken in das Verzeichnis /usr/lib bzw. /usr/lib64 kopiert und alle symbolischen Links angelegt. Die Headerdatei können Sie in ein Verzeichnis Ihrer Wahl kopieren.

Option 2: Kopieren Sie die Dateien dieses Supportpakets in Verzeichnisse Ihrer Wahl und Erzeugen Sie symbolische Links auf die Bibliotheksdatei libfeusb.so.x.y.z[1] im Verzeichnis /usr/lib bzw. /usr/lib64 durch folgende Aufrufe:

cd /usr/lib (für 64 Bit : /usr/lib64)

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

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

ldconfig

Verwendung der libfeusb.so.x.y.z ohne Administrator-Rechte:

Voraussetzungen:

Der udev daemon wird zur Plug and Play Erkennung von Geräten verwendet.

Die Applikation chmod muss sich im Verzeichnis /bin befinden.

- Kopieren Sie die Datei **41-feig.rules** in den Ordner /etc/udev/rules.d.

Anmerkung:

x86 : Die Bibliothek wurde unter SuSE Linux 11.1 mit der GNU Compiler Collection V4.3.2 erstellt.

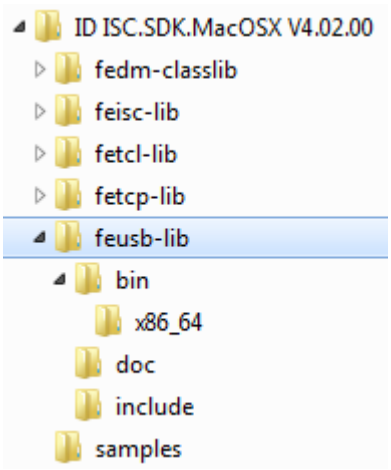
X64: Die Bibliothek wurde unter SuSE Linux 11.2 mit der GNU Compiler Collection V4.4.1 erstellt.

[1] x.y.z repräsentiert die Versionsnummer der Bibliotheksdatei

3.3.1. libusb

LIBFEUSB basiert auf der Open-Source Entwicklung libusb in der Version 0.1.12, die nicht Bestandteil dieses Paketes ist. libusb kann unter http://libusb.sourceforge.net bezogen werden und muss separat installiert werden.

3.4. 64-Bit Mac OS X



Zur Installation gibt es zwei Optionen:

Option 1: Falls eine install.sh im SDK-Verzeichnis vorliegt, führen Sie diese aus. Damit werden alle Bibliotheken in das Verzeichnis /usr/local/lib kopiert und alle symbolischen Links angelegt. Die Headerdatei können Sie in ein Verzeichnis Ihrer Wahl kopieren.

Option 2: Kopieren Sie die Dateien dieses Supportpakets in Verzeichnisse Ihrer Wahl und Erzeugen Sie symbolische Links auf die Bibliotheksdatei libfeusb.x.y.z.dylib[1] im Verzeichnis /usr/local/lib durch folgende Aufrufe:

cd /usr/local/lib

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

ln –s libfeusb.x.dylib libfeusb.dylib

Anmerkung: Die Bibliothek wurde unter Mac OS X V10.7.3 mit Xcode V4.3.2 erstellt. Die Bibliothek ist mit der Architektur x86_64 kompatibel.

3.4.1. libusb

LIBFEUSB basiert auf der Open-Source Entwicklung libusb in der Version 0.1.13 beta, die nicht Bestandteil dieses Paketes ist. Das Binary von libusb kann unter http://www.ellert.se/twain-sane/ bezogen werden und muss separat installiert werden.

[1] x.y.z repräsentiert die Versionsnummer der Bibliotheksdatei

3.5. Deaktivieren des Plug-and-play Threads

Für das Erkennen von USB-Lesern wird in der Bibliothek ein Thread gestartet, der zyklisch nach neuen USB-Lesern sucht, die anschließend über den Event-Mechanismus (s. 6.3. Ereignissignalisierung) einer Applikation gemeldet werden können.

Für den Fall, dass dieser Automatismus nicht gewünscht wird, kann man diesen mit den folgenden Schritten abschalten:

a) Erstellen Sie eine Datei feusb.conf

b) Fügen Sie der Datei den Text *nopnp* hinzu

c) Speichern Sie die Datei im Verzeichnis der Applikation

4. Einbindung in das Anwendungsprogramm

4.1. Unterstützte Entwicklungsumgebungen

Betriebssystem	Entwicklungsumgebung	Unterstützung
Windows XP / Vista / 7 / 8	Visual Studio	ja
	Borland C++ Builder	ja
	Embarcadero C++ Builder	ja
Windows CE	eMbedded Visual C++ 4	ja
	Visual Studio 2005 / 2008	ja
Linux	GCC	ja
Mac OS X	GCC	ja, für Projekte mit x86_64 Architektur
	Xcode ≥ V4.3.2	ja, für Projekte mit x86_64 Architektur

4.2. Einbindung in Visual Studio

1. Include-Pfad zur Headerdatei in den Projekteinstellungen (Kategorie C/C++) hinzufügen
2. die LIB-Datei in den Projekteinstellungen (Kategorie Linker) eintragen

4.3. Einbindung in Xcode

1. Pfad zur Headerdatei in den Projekteinstellungen (Kategorie Search Paths und dort für User Header Search Paths) hinzufügen
2. die DYLIB-Datei per Drag-and-Drop dem Projekt hinzufügen

5. Eine Kurzeinführung in USB

Mit dem USB (Universal-Serial-Bus) hat sich ein neuer Standard für den Anschluß von Peripherie im PC-Umfeld etabliert. Gegenüber dem seriellen Interface sind vor allem die Plug&Play-Fähigkeit und die höhere Transfergeschwindigkeit hervorzuheben. Auf der anderen Seite verlangt der neue Standard aber auch tiefgreifendes Wissen über die Eigenschaften des USB, wenn man aus der Anwendersoftware heraus auf USB-Geräte zugreifen möchte.

Mit der Funktionssammlung FEUSB soll dem Anwendungsprogrammierer die notwendige Hilfestellung zur Kommunikation mit USB-Geräten (Devices) aus der OBID®-Familie gegeben werden. Mit wenigen Kenntnissen, die in diesem Abschnitt vermittelt werden, wird jeder geübte Programmierer in der Lage sein, professionelle Anwendungsprogramme zu entwickeln[1].

Der USB ist ein Single-Master-Bus mit dem PC als Master (Host). Nur dieser Master kann Protokollaktivitäten auslösen. Unterstützt werden gleichzeitig bis zu 127 physikalische Geräte. Die Geräte werden durch Busadressen unterschieden, die vom Host automatisch vergeben werden. Nach dem Einstecken eines Peripheriegerätes wird im Host unmittelbar eine Initialisierungsphase (Enumeration) gestartet, die dem Host erlaubt, den oder die geeigneten Treiber zu laden. Dieser Prozeß wird immer vom Betriebssystem (herstellerunabhängig) ausgelöst.

Physikalisch gesehen besteht ein USB-Gerät aus mindestens einem logischen USB-Gerät. Das bedeutet, dass die Kommunikationsdaten sich innerhalb des Gerätes in mehrere Informationskanäle, den sogenannten Pipes, aufspalten können. Jeder Pipe ist ein Endpoint zugeordnet, der physikalisch einem FIFO (First-In-First-Out) entspricht.

Ein logisches USB-Gerät kann nun mehrere Pipes zu einem Interface zusammenfassen und für ein solches Interface kann der Host einen geeigneten Treiber installieren. Die Informationen über die logische Zusammensetzung eines USB-Gerätes erhält der Host während der Enumeration.

USB-Geräte aus der OBID®-Familie zeichnen sich nun dadurch aus, dass sie alle einheitliche Interfaces haben. Somit kann man die speziellen USB-Treiber als geräteunabhängig innerhalb der OBID®-Familie einstufen. Der Programmierer kommt aber mit diesen Treibern, Interfaces, Pipes oder Busadressen nicht in Berührung. Für ihn wurde ein Programmiermodell entwickelt, das ihm in maximal vier Schritten die Kommunikation mit OBID® USB-Geräten ermöglicht:

1. Scan-Vorgang: Mit einem Funktionsaufruf werden alle OBID® USB-Geräte am USB erkannt und in einer Scanliste innerhalb der DLL verwaltet.
2. Geräteauswahl: Im zweiten Schritt wird aus dieser Scanliste ein USB-Gerät anhand seiner Seriennummer ausgewählt. Die Seriennummer der Geräte sind übrigens das einzige Unterscheidungsmerkmal untereinander.

[1] Dem an Details interessierten Leser sei das Buch „USB“ aus dem Franzis-Verlag, Hrsg. H. J. Kelm, empfohlen (ISBN 3-7723-7964-8)

3. Kommunikationsweg öffnen: Mit dem dritten Schritt wird ein Kanal zu diesem USB-Gerät geöffnet[1]. Dafür wird intern eine Datenstruktur, das Device-Objekt, in der DLL angelegt.
4. Datenaustausch: Ab dem vierten Schritt können Daten mit dem USB-Gerät ausgetauscht werden.

Ein OBID® USB-Gerät kann nun ein oder mehrere Interfaces haben und mit der Funktionssammlung FEUSB allein müßte der Programmierer entscheiden, welche Daten er über welches Interface schicken muß. Dies wird ihm durch eine zusätzliche Funktionssammlung, die für jede OBID®-Gerätefamilie zur Verfügung steht, abgenommen. Es muß sich deshalb ein Programmierer nicht mit den Besonderheiten der OBID®-Interfaces auseinandersetzen.

[1] Mit der speziellen Funktion FEUSB_ScanAndOpen sind die Schritte 1, 2 und 3 zusammengefaßt. Sie kann verwendet werden, wenn man gewöhlich nur ein USB-Gerät aus der OBID-Familie anschließt.

6. Programmierschnittstelle

6.1. Übersicht

Die FEUSB kapselt für den Anwender alle notwendigen Funktionen und Parameter zum Verwalten von einem oder mehreren gleichzeitig geöffneten OBID® USB-Geräten am USB des PC. Der objektorientierte innere Aufbau (s. Abb. 1) ist nach außen hin bewußt als eine Funktionsschnittstelle herausgeführt. Dies hat den Vorteil der Sprachunabhängigkeit.

Die Bibliothek hat eine Selbstverwaltung, die ein Anwendungsprogramm davon befreit, irgendwelche Werte, Einstellungungen oder Sonstiges zwischenspeichern zu müssen. Der Treiber-Manager in FEUSB führt eine Liste mit allen geöffneten Kanälen (erzeugten Device-Objekten) und jedes Device-Objekt verwaltet alle relevanten Einstellungen für seinen Kanal innerhalb seines lokalen Speichers. Mit einem Device-Objekt ist immer genau ein geöffneter Kanal zu einem bestimmten OBID® USB-Gerät verbunden und über diesen Kanal kann nur das mit seiner Seriennummer eingetragene Gerät angesprochen werden. Ein Kanal zu einem OBID® USB-Gerät kann nur einmal geöffnet werden.

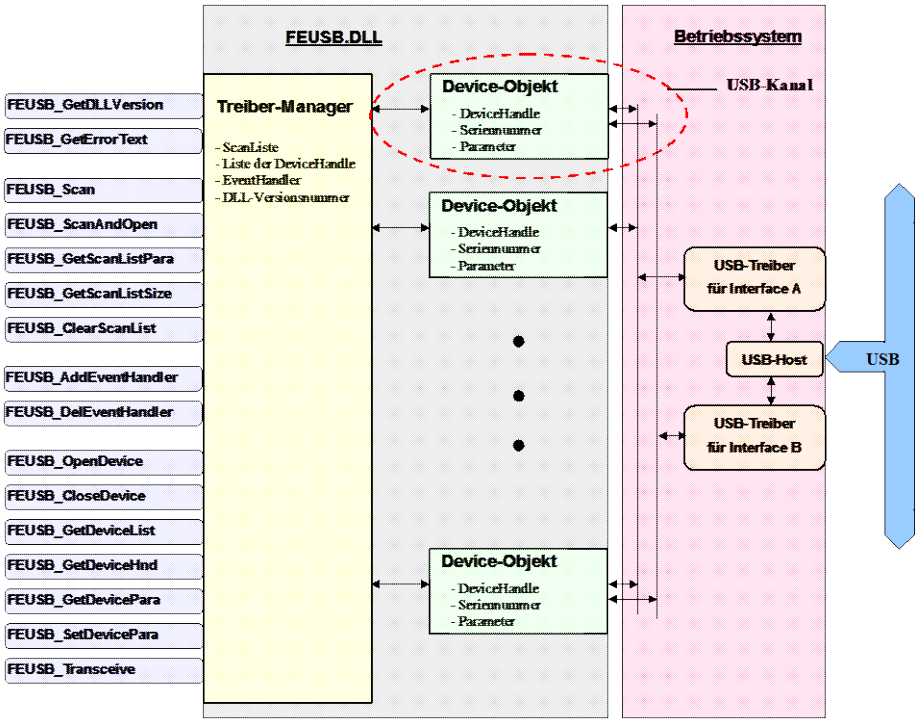


Abbildung 1: Interner Aufbau von FEUSB.DLL

Der erste Schritt zum Verbindungsaufbau zu einem OBID® USB-Gerät ist das Ermitteln (Scan-Vorgang) eines oder aller OBID® USB-Geräte am USB des PC. Jedes gefundene Gerät wird in die interne Scanliste eingetragen, aber nicht geöffnet.

Vor der ersten Kommunikation muß ein USB-Gerät aus der Scanliste ausgewählt und mit der Funktion FEUSB_OpenDevice ein Kanal zu diesem Gerät geöffnet werden. Wenn diese Funktion fehlerfrei ausgeführt werden konnte, erhält man mit dem Rückgabewert einen Handle, der vom Anwendungsprogramm verwaltet werden kann. Nur mit diesem Handle ist eine eindeutige Identifikation des geöffneten Kanals möglich. Der oder die Handle müssen aber nicht im Anwendungsprogramm gespeichert werden, denn der Treiber-Manager verwaltet intern eine Liste aller geöffneten Kanäle. Diese Liste kann mit der Funktion FEUSB_GetDeviceList abgerufen werden. Mit den Handle, die man damit sukzessive erhält, kann man anschließend mit der Funktion FEUSB_GetDevicePara alle Einstellungen für diesen USB-Device auslesen.

Ein mit FEUSB_OpenDevice geöffneter Kanal muß unbedingt wieder mit der Funktion FEUSB_CloseDevice geschlossen werden.

Jede Bibliotheksfunktion (Ausnahme: FEUSB_GetDLLVersion) hat einen Rückgabewert, der im Fehlerfall immer negativ ist.

Wird ein Anwendungsprogramm mehrfach aufgerufen, erhält jedes Programm (Instanz) mit dem Funktionsaufruf FEUSB_GetDeviceList eine leere Device-Liste. Dadurch wird eine Vermischung von Zugriffsrechten unter verschiedenen Programm-Instanzen verhindert. Beachten Sie bitte, dass im Gegensatz zur seriellen Schnittstelle ein USB-Kanal von jedem Programm oder einer weiteren Instanz davon neu geöffnet werden kann! Das bedeutet, dass verschiedene Programme quasi gleichzeitig mit ein und demselben USB-Gerät Daten austauschen können. Die dabei möglichen Zugriffskonflikte werden durch die FEUSB nicht abgefangen.

6.2. Threadsicherheit

Alle FEIG-Bibliotheken sind prinzipiell nicht vollständig threadsicher. Unter Beachtung einiger Regeln kann man dennoch Parallelität in der Ausführung von Kommunikationsaufgaben und damit praktische Threadsicherheit erreichen. Man muss auch wissen, dass alle OBID® RFID-Leser immer nur eine Aktion ausführen können, also synchron arbeiten.

Auf der Ebene der Transportschicht (FECOM, FEUSB, FETCP) kann über jede Verbindung nur synchron kommuniziert werden, weil auch die OBID-Leser nur synchron arbeiten. Threadsicher sind die Port-Objekte untereinander, weil diese unabhängig voneinander sind. Es ist demnach möglich, dass z. B. zwei Threads mit zwei OBID-Lesern über zwei verschiedene TCP-Verbindungen kommunizieren.

6.3. Aufbau und Funktion der Scanliste

Das Öffnen eines Kanals zu einem OBID® USB-Gerät ist nur mit seiner individuellen Seriennummer (Device-ID) möglich. Vor dem Öffnen muß deshalb mit einem Scanvorgang (mit **FEUSB_Scan** oder **FEUSB_ScanAndOpen**) ein (oder mehrere) OBID® USB-Gerät(e) am USB-Port ermittelt und die Seriennummer(n) ausgelesen werden. Die gefundenen USB-Geräte werden in der internen Scanliste eingetragen und dort anhand ihrer Seriennummer verwaltet. Nach dem Öffnen (mit **FEUSB_ScanAndOpen** oder **FEUSB_OpenDevice**) wird der Device-Handle des Kanals in die Scanliste nachgetragen. Zusätzlich wird vermerkt, dass das USB-Gerät betriebsbereit ist.

Die Struktur der internen Scanliste enthält folgende Datenelemente:

```
int     iScanNo;          // Index in Scanliste (>= 0)
DWORD   dwDeviceID;       // Seriennummer (>0)
int     iDeviceHnd;       // Device-Handle (0: Kanal nicht geöffnet; >0: Kanal geöffnet)
char    cFamilyName[25];  // Name der Gerätefamilie (z.B. "OBID i-scan Proximity")
char    cDeviceName[25];  // Device-Name (z.B. "ID ISC.PRH100-U")
bool    bPresent;         // Bereitschaftsflag (true oder false)
```

Jedes Datenelement der Scanliste kann man mit der Funktion **FEUSB_GetScanListPara** auslesen.

Ein wesentliches Datenelement ist das Bereitschaftsflag *bPresent*, das anzeigt, ob das Gerät nach dem Öffnen des Kanals noch am USB-Port angeschlossen ist. Entfernt man ein USB-Gerät, nachdem ein Kanal zu diesem Gerät geöffnet wurde, wird intern das Bereitschaftsflag auf false gesetzt. Der Kanal bleibt aber geöffnet. Wird dasselbe Gerät wieder angeschlossen, wird das Bereitschaftsflag wieder auf true gesetzt und die Kommunikation kann sofort wieder aufgenommen werden.

Die Bereitschaft eines USB-Gerätes kann man auf drei Arten ermitteln:

- Abfragen des Bereitschaftsflags mit **FEUSB_GetScanListPara**(iIndex, „PRESENT“, cValue)
- Abfragen der Bereitschaft mit **FEUSB_IsDevicePresent**(iDevHnd)
- Einrichtung einer Ereignissignalisierung (s. Kapitel 6.4. Ereignissignalisierung)

Die Scanliste kann jederzeit mit der Funktion **FEUSB_ClearScanList** gelöscht werden. Dabei werden geöffnete Kanäle nicht geschlossen! Mit den beiden Scan-Funktionen kann man anschließend jederzeit die Scanliste erneut aufbauen. Dabei werden offen gehaltene Kanäle erkannt und auch die Bereitschaftsflags wieder entsprechend gesetzt. Mit dem Löschen der Scanliste gehen demnach keine wichtigen Informationen dauerhaft verloren.

Zum Schließen geöffneter Kanäle sollte allerdings die Scanliste nicht herangezogen werden, da sie aus den genannten Gründen nicht jeden offenen Kanal aktuell verwaltet. Besser ist es, immer die Deviceliste zu verwenden, die mit der Funktion **FEUSB_GetDeviceList** zyklisch ausgelesen werden kann.

6.4. Ereignissignalisierung

Für die Plug&Play-Ereignisse[1] ***Connect*** und ***Disconnect*** können, getrennt für jedes Ereignis und unabhängig davon, ob das Gerät bereits in der internen Scanliste geführt wird, Ereignisbehandlungsmaßnahmen installiert werden. Sobald ein USB-Gerät eingesteckt oder abgezogen wird, wird die entsprechende Signalisierung ausgeführt. Auf diese Weise kann man einer Applikation asynchron zum Programmablauf das Ereignis mitteilen.

Eine Ereignisbehandlungsmaßnahme muss mit der Funktion **FEUSB_AddEventHandler** installiert werden. Man kann zwischen drei verschiedenen Signalisierungsmethoden wählen: Nachricht an aufrufenden Prozess, Nachricht an ein Fenster oder Verwendung einer Callback-Funktion.

Eine installierte Ereignisbehandlungsmaßnahme muss mit der Funktion **FEUSB_DelEventHandler** wieder entfernt werden.

Die Struktur **FEUSB_EVENT_INIT** enthält die für die Signalisierung notwendigen Parameter:

```
typedef struct _FEUSB_EVENT_INIT
{
    UINT uiFlag;      // Spezifiziert die Verwendung der union (z.B. FEUSB_WND_HWND)
    UINT uiUse;       // Definiert den Event (z.B. FEUSB_CONNECT_EVENT)
    UINT uiMsg;       // Message-Code für dwThreadID und hwndWnd (z.B. WM_USER_xyz)
    union
    {
        DWORD  dwThreadID;              // für Thread-ID
        HWND   hwndWnd;                 // für Window-Handle
        void   (*cbFct)(int, DWORD);    // für Callback-Funktion
    } Method[2];
} FEUSB_EVENT_INIT;
```

Kernelement der Struktur ist die **union**, die entweder die ID eines Prozesses, das Handle eines Fensters oder einen Funktionszeiger enthält. Die Auswahl der Signalisierungsform wird mit dem Parameter *uiFlag* vorgenommen. Im Parameter *uiUse* hinterlegt man eine Kennung des Ereignisses, der man die Behandlungsmethode zuordnen möchte. Für die Nachrichtenmethoden muss man in *uiMsg* den Messagecode hinterlegen.

Man kann zu einem Ereignis mehrere Behandlungsmethoden installieren. Aber jede *dwThreadID*, *hwndWnd* oder *cbFct* kann nur einmal pro Ereignis verwendet werden.

Anmerkung zu Linux: Die Connect-Signalisierung von OBID® USB-Geräten mit zusätzlichem HID-Interface dauert ca. 10..12 Sekunden.

[1] Die Ereignissignalisierung kann man generell abschalten. Siehe 3.5. Deaktivieren des Plug-and-play Threads

[2] Die Benennung der union mit Method ist ausschließlich für C-Programmierer. C++-Programmierer greifen auf die union direkt über die Struktur zu.

6.5. Liste der Funktionen[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] Hinweis: UCHAR ist definiert als 8-bit unsigned char.

6.6. Funktionsbeschreibungen

6.6.1. FEUSB_GetDLLVersion

Funktion	Ermittelt die Versionsnummer der DLL.
Syntax	**void FEUSB_GetDLLVersion(char* cVersion)**
Beschreibung	Die Funktion gibt die Versionsnummer der DLL zurück. *cVersion* ist eine leere, nullterminierte Zeichenkette zur Rückgabe der Versionsnummer. Die Zeichenkette sollte wenigstens 256 Zeichen aufnehmen können. In der Zeichenkette wird aktuellen Versionsnummer zurückgegeben (z. B. “04.02.04"). Neuere Versionen könnten aber weitere Informationen liefern.
Rückgabewert	ohne
Beispiel	... #include "feusb.h" ... ... char cVersion[256]; FEUSB_GetDLLVersion(cVersion); // hier Code zum Anzeigen der Versionsnummer ... ...

6.6.2. FEUSB_GetDrvVersion (nur für Windows)

Funktion	Gibt die Versionsinformationen des Kerneltreibers zurück.
Syntax	**int FEUSB_GetDrvVersion(char* cVersion)**
Beschreibung	Die Funktion gibt die Versionsinformationen zum installierten Kerneltreibers zurück. **ACHTUNG**: die Funktion kann nur bei geladenem Treiber verwendet werden. Dies ist der Fall, wenn ein Kanal zu einem USB-Leser geöffnet ist. *cVersion* ist eine leere, nullterminierte Zeichenkette zur Rückgabe der Versionsinformationen. Die Zeichenkette sollte wenigstens 256 Zeichen aufnehmen können.
Rückgabewert	Im Fehlerfall liefert die Funktion <0 zurück, ansonsten 0. Die Liste der Fehlercodes findet sich im Anhang.

Beispiel	...
	#include "feusb.h"
	...
	...
	char cVersion[256];
	if(0 == FEUSB_GetDrvVersion(cVersion))
	// hier Code zum Anzeigen der Versionsinformationen
	...
	...

6.6.3. FEUSB_GetErrorText

Funktion	Gibt Fehlertext zurück
Syntax	**int FEUSB_GetErrorText(int iError, char* cText)**
Beschreibung	Die Funktion gibt zum übergebenen Fehlercode einen Fehlertext[1] zurück. *iError* ist der Fehlercode (immer negativ). *cText* ist eine leere, nullterminierte Zeichenkette zur Rückgabe des Fehlertextes. Die Zeichenkette sollte wenigstens 256 Zeichen aufnehmen können.
Rückgabewert	Im Fehlerfall liefert die Funktion den Code FEUSB_ERR_UNKNOWN_ERRORCODE zurück, ansonsten 0. Die Liste der Fehlercodes findet sich im Anhang.

Beispiel	...
	#include "feusb.h"
	...
	...
	char cText[256];
	int iErr = FEUSB_GetErrorText(-1100, cText);
	// hier Code zum Anzeigen des Fehlertextes
	...
	...

6.6.4. FEUSB_GetLastError

Funktion	Ermittelt den letzten Fehlercode und übergibt Fehlertext
Syntax	**int FEUSB_GetLastError(int iDevHnd , int* iErrorCode, char* cErrorText)**
Beschreibung	Die Funktion übergibt in *iErrorCode* den letzten Fehlercode des mit *iDevHnd* ausgewählten USB-Kanals zurück und übergibt in *cErrorText* den zugehörigen englischen Fehlertext. Der Puffer für *cErrorText* sollte 256 Zeichen aufnehmen können. Setzt man *iDevHnd* auf 0, wird der letzte Fehler des Objekt-Managers zurückgegeben.
Rückgabewert	Im fehlerfreien Fall liefert die Funktion Null und im Fehlerfall einen Wert kleiner als Null zurück. Die Liste der Fehlercodes findet sich im Anhang.

Beispiel	...
	#include "feusb.h"
	...
	...
	char cErrorText[256];
	int iErrorCode = 0;
	...
	
	int iBack = FEUSB_GetLastError(iDevHnd, &iErrorCode, cErrorText);
	// hier Code zum Anzeigen des Textes
	...
	...

[1] in englisch

6.6.5. FEUSB_Scan

Funktion	Ermittlung eines einzelnen oder aller USB-Geräte
Syntax	**int FEUSB_Scan(int iScanOpt, FEUSB_SCANSEARCH* pSearchOpt)**
Beschreibung	Mit dieser Funktion wird der USB nach Geräten mit FEIG-Kennung abgesucht und jedes gefundene Gerät in der internen Scanliste eingetragen. Der Parameter *iScanOpt* erlaubt das Suchen nach einem oder nach allen Geräten., bzw. erlaubt das Entfernen von nicht mehr vorhandenen Geräten aus der Scanliste. Der Index der erstellten Scanliste ist null-basiert. Die Parameter *pSearchOpt* wird für die gezielte Suche mit der Option FEUSB_SCAN_SEARCH eingesetzt. Wird diese Option nicht genutzt, muß in C/C++ NULL übergeben werden. In Visual Basic übergibt man die Konstante vbNullString. Der Parameter *iScanOpt* steuert den Scan-Vorgang und setzt sich zusammen aus *iScanOpt* = [*SteuerID*] | [*OptionID*] | [*OptionID*]. SteuerIDs: • FEUSB_SCAN_FIRST sucht nach dem Gerät, das als erstes vom Betriebssystem registriert wurde. Der interne Scan-Zähler wird deshalb auf 0 gesetzt. Die Scanliste wird vor dem Scan-Vorgang gelöscht. • FEUSB_SCAN_NEXT sucht nach dem Gerät, das als nächstes vom Betriebssystem registriert wurde. Dazu wird der interne Scan-Zähler verwendet, der mit jedem erfolgreichen FEUSB_SCAN_NEXT inkrementiert wird (bis max. 127). Achtung: jedes FEUSB_SCAN_FIRST setzt den internen Scan-Zähler wieder auf 0 zurück! • FEUSB_SCAN_NEW sucht nach einem neuen, noch nicht in der Scanliste eingetragenen Gerät. Der interne Scan-Zähler wird entsprechend neu gesetzt. • FEUSB_SCAN_ALL erlaubt das Suchen aller Geräte am USB. Der Scan-Zähler wird entsprechend neu gesetzt. Die Scanliste wird zuerst gelöscht und dann neu aufgebaut. Es ist also nicht sichergestellt, dass ein zuvor in der Scanliste eingetragenes Gerät wieder mit demselben Index geführt wird. • FEUSB_SCAN_PACK entfernt alle Geräte aus der internen Scanliste, die nicht mehr am USB gefunden werden. OptionIDs: • FEUSB_SCAN_SEARCH sucht gezielt nach einem Gerät am USB. Im Parameter *pSearchOpt*[1] gibt man die Suchoptionen an. Diese OptionID muß immer mit einer SteuerID verknüpft werden. • FEUSB_SCAN_PACK entfernt alle Geräte aus der internen Scanliste, die nicht

[1] s. Anhang 8.4. Liste der Konstanten für die FEUSB_SCANSEARCH-Struktur, 8.5. Liste der cFamilyName in der FEUSB_SCANSEARCH-Struktur und 8.6. Liste der cDeviceName in der FEUSB_SCANSEARCH-Struktur

	mehr am USB gefunden werden. Die Option wird nur ausgeführt, wenn der vorhergehende (optionale) Scan-Vorgang fehlerfrei war. Vorsicht: diese Option verändert den Listenindex der bereits eingetragenen Geräte! Diese Option kann mit allen kombiniert werden. Für die SteuerID FEUSB_SCAN_ALL ist dies aber überflüssig, da damit automatisch die Scanliste neu aufgebaut wird.
Hinweis	Ein Löschen der Scanliste mit **FEUSB_ClearScanList** schließt nicht (!) die mit **FEUSB_OpenDevice** angelegten Objekte.
Rückgabewert	Konnten ein oder mehrere USB-Geräte gefunden werden, wird im Rückgabewert für die Option FEUSB_SCAN_NEXT oder FEUSB_SCAN_NEW der Index des Gerätes in der Scanliste zurückgegeben und mit der Option FEUSB_SCAN_FIRST oder FEUSB_SCAN_ALL eine 0. Im Fehlerfall liefert die Funktion einen Wert kleiner als Null zurück. Auch im Fehlerfall können einige Geräte erkannt und in die Scanliste aufgenommen worden sein. Nach einem Scan-Vorgang mit der Option FEUSB_SCAN_ALL sollte deshalb immer die Größe der Scanliste mit **FEUSB_ScanListSize** überprüft werden. Die Liste der Fehlercodes findet sich im Anhang.
Scan-Optionen	FEUSB_SCAN_FIRST 0x00000001 FEUSB_SCAN_NEXT 0x00000002 FEUSB_SCAN_NEW 0x00000003 FEUSB_SCAN_ALL 0x0000000F FEUSB_SCAN_SEARCH 0x00010000 FEUSB_SCAN_PACK 0x00020000
Such-Optionen	FEUSB_SEARCH_FAMILY 0x00000001 FEUSB_SEARCH_PRODUCT 0x00000002 FEUSB_SEARCH_DEVICEID 0x00000004
Beispiel	...

```
#include "feusb.h"
...
char cDeviceID[16];
long nDeviceID;
FEUSB_SCANSEARCH search;
...
// Suchoptionen einstellen
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 )
        {
            // hier Code für den Fehlerfall
        }
        else
        {
            // hier Code für Kommunikation oder anderes
        }
    }
}
```

6.6.6. FEUSB_ScanAndOpen

Funktion	Ermittlung eines einzelnen oder aller USB-Geräte mit anschließender Öffnung der Kanäle
Syntax	**int FEUSB_ScanAndOpen(int iScanOpt, FEUSB_SCANSEARCH* pSearchOpt)**
Beschreibung	Diese Funktion kombiniert die Funktionen **FEUSB_Scan** und **FEUSB_OpenDevice**. Gerade für den häufig anzunehmenden Fall, dass sich genau ein OBID®-Gerät am USB befindet, kann man mit dieser Funktion unmittelbar ein Kanal zum Gerät öffnen. Die Beschreibung der Parameter finden sich in den Kapiteln der genannten Funktionen.
Rückgabewert	Konnten ein oder mehrere USB-Geräte gefunden und geöffnet werden, wird im Rückgabewert für die Optionen FEUSB_SCAN_FIRST, FEUSB_SCAN_NEXT oder FEUSB_SCAN_NEW der Device-Handle zurückgegeben. Mit der Option FEUSB_SCAN_ALL wird eine 0 zurückgegeben und der Device-Handle der geöffneten Kanäle muß mit der Funktion FEUSB_GetScanListPara ermittelt werden. Im Fehlerfall liefert die Funktion einen Wert kleiner als Null zurück. Auch im Fehlerfall können einige Geräte erkannt und in die Scanliste aufgenommen worden sein. Nach einem ScanAndOpen-Vorgang mit der Option FEUSB_SCAN_ALL sollte deshalb immer die Größe der Scanliste mit **FEUSB_GetScanListSize** überprüft werden. Die Liste der Fehlercodes findet sich im Anhang.
Querverweise	6.5.4. FEUSB_GetLastError, 6.5.10. FEUSB_OpenDevice
Beispiel	s. Beispiele zu **FEUSB_Scan** und **FEUSB_GetScanListSize**

6.6.7. FEUSB_GetScanListPara

Funktion	Liest einen Wert aus der Scanliste
Syntax	**int FEUSB_GetScanListPara(int iIndex, char* cPara, char* cValue)**
Beschreibung	Mit dieser Funktion hat man Zugriff auf die Werte in der Scanliste. Jeder Datensatz enthält einige Werte zu einem gefunden OBID®-Gerät. Der Zugriff erfolgt durch den nullbasierten Index *iIndex*. Im Parameter *cPara* gibt man die Kennung für den betreffenden Scanlistenwert an (s. Feld Parameter). *cValue* ist eine leere, nullterminierte Zeichenkette zur Rückgabe des Scanlistenwertes. Die Zeichenkette sollte wenigstens 25 Zeichen aufnehmen können.
Rückgabewert	Im Fehlerfall liefert die Funktion einen Wert kleiner als Null zurück. Die Liste der Fehlercodes findet sich im Anhang.
Parameter	Die Parameter-Kennungen sind: „Device-ID“ - Seriennummer des USB-Gerätes in hexadezimal Darstellung „DeviceHnd“ - Device-Handle zum USB-Kanal „FamilyName“ - Name der Gerätefamilie des Device am USB-Kanal „DeviceName“ - Name des Device am USB-Kanal „Present“ - USB-Gerät angeschlossen (cValue=„1“) oder entfernt (cValue=„0“)

Hinweis	Die ermittelte **Device-ID** repräsentiert einen Wert im hexadezimalen Zahlensystem. Zum Beispiel gehört zu "6D89573" die Device-ID 0x06D89573 bzw. 114857331.
	
	Das folgende Beispiel zeigt, wie man die Zeichenkette umwandeln muß:
	
	cDeviceID[16];
	long nDeviceID = 0;
	...
	if(FEUSB_GetScanListPara(index, "Device-ID", cDeviceID) == 0)
	{
	sscanf((const char*)cDeviceID, "%lx", &nDeviceID);
	iDeviceHnd = FEUSB_OpenDevice(nDeviceID);
	}

Beispiel	s. Beispiele zu **FEUSB_Scan** und **FEUSB_GetScanListSize**

6.6.8. FEUSB_GetScanListSize

Funktion	Ermittelt die Größe der Scanliste
Syntax	**int FEUSB_GetScanListSize()**
Beschreibung	Mit dieser Funktion ermittelt man die Anzahl der Datensätze in der Scanliste.
Rückgabewert	Im Fehlerfall liefert die Funktion einen Wert kleiner als Null zurück. Die Liste der Fehlercodes findet sich im Anhang.

Beispiel	...
	#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)
	  {
	  // hier Code für den Fehlerfall
	  }
	  else
	  {
	  // hier Code für Kommunikation oder anderes
	  }
	  }
	}
	...
	...

6.6.9. FEUSB_ClearScanList

Funktion	Löscht Scanliste
Syntax	**int FEUSB_ClearScanList()**
Beschreibung	Diese Funktion löscht die Scanliste. Bereits angelegte Device-Objekte werden dadurch nicht automatisch geschlossen. Eine Restaurierung der Scanliste mit einem erneuten Scan ist also möglich.
Rückgabewert	Im Fehlerfall liefert die Funktion einen Wert kleiner als Null zurück. Die Liste der Fehlercodes findet sich im Anhang.
Beispiel	

6.6.10. FEUSB_OpenDevice

Funktion	Öffnet einen Kanal zur Kommunikation mit einem OBID®-Leser.
Syntax	**int FEUSB_OpenDevice(long nDeviceID)**
Beschreibung	Die Funktion öffnet einen USB-Kanal und legt intern ein Device-Objekt zur Verwaltung der Kanal-Parameter an. Der zurückgelieferte Handle *iDevHnd* identifiziert den Kanal von außen. *nDeviceID* ist die Seriennummer des OBID®-Gerätes am zu öffnenden USB-Kanal. Der Funktionsaufruf wird mit einem Fehler beendet, wenn die Seriennummer nicht in der Scanliste gefunden wurde. Nach einem erfolgreichen Öffnen wird der Device-Handle zusätzlich in der Scanliste hinterlegt und das Bereitschaftsflag gesetzt. Dadurch kann man mit dem Auslesen der Scanliste schon ermitteln, welche Geräte gefunden, eventuell geöffnet und bereit sind. Der mit **FEUSB_OpenDevice** geöffnete USB-Kanal muß (!) mit der Funktion **FEUSB_CloseDevice** wieder geschlossen werden. Andernfalls wird der von der DLL reservierte Speicher nicht wieder freigegeben. Ein wiederholtes Aufrufen dieser Funktion mit derselben Seriennummer führt nicht zum mehrmaligen Öffnen von Kanälen, sondern es wird der zugehörige Handle zurückgegeben.

Hinweis	Die mit der Funktion **FEUSB_GetScanListPara** ermittelte Device-ID repräsentiert einen Wert im hexadezimalen Zahlensystem. Zum Beispiel gehört zu "6D89573" die Device-ID 0x06D89573 bzw. 114857331.
	Das folgende Beispiel zeigt, wie man die Zeichenkette umwandeln muß:
	cDeviceID[16];
	long nDeviceID = 0;
	...
	if(FEUSB_GetScanListPara(index, "Device-ID", cDeviceID) == 0)
	{
	sscanf((const char*)cDeviceID, "%lx", &nDeviceID);
	iDeviceHnd = FEUSB_OpenDevice(nDeviceID);
	}

Rückgabewert	Wenn der Kanal zum USB-Gerät fehlerfrei geöffnet werden konnte, wird ein Handle (>0) zurückgeliefert. Im Fehlerfall liefert die Funktion einen Wert kleiner als Null zurück. Die Liste der Fehlercodes findet sich im Anhang.
Beispiel	s. Beispiele zu **FEUSB_Scan** und **FEUSB_GetScanListSize**

6.6.11. FEUSB_CloseDevice

Funktion	Schließt einen USB-Kanal zu einem OBID®-Gerät
Syntax	**int FEUSB_CloseDevice(int iDevHnd)**
Beschreibung	Die Funktion schließt die durch den Parameter *iDevHnd* angegebenen USB-Kanal und gibt den reservierten Speicher wieder frei.
Rückgabewert	Der Rückgabewert ist 0, wenn der Kanal geschlossen wurde. Im Fehlerfall liefert die Funktion einen Wert kleiner als Null zurück. Die Liste der Fehlercodes findet sich im Anhang.

Beispiel	...
	#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)
	  {
	  // hier Code für den Fehlerfall
	  }
	  else
	  {
	  int iErr = FEUSB_CloseDevice(iDevHnd);
	  }
	  }
	}
	
	...
	...

6.6.12. FEUSB_IsDevicePresent

Funktion	Prüft Bereitschaft eines USB-Gerätes
Syntax	**int FEUSB_IsDevicePresent(int iDevHnd)**
Beschreibung	Die Funktion prüft an dem durch den Parameter *iDevHnd* angegebenen USB-Kanal die Bereitschaft des USB-Gerätes.
Rückgabewert	Der Rückgabewert ist 1, wenn das USB-Gerät kommunikationsbereit ist, andernfalls 0. Im Fehlerfall liefert die Funktion einen Wert kleiner als Null zurück. Die Liste der Fehlercodes findet sich im Anhang.
Beispiel	

6.6.13. FEUSB_GetDeviceList

Funktion	Ermittelt in Abhängigkeit vom Parameter *iNext* den ersten oder den nachfolgenden Device-Handle aus der internen Liste der geöffneten seriellen Schnittstellen.
Syntax	**int FEUSB_GetDeviceList(int iNext)**
Beschreibung	Die Funktion gibt ein Device-Handle aus der internen Liste der Device-Handles zurück. Übergibt man für *iNext* eine 0, wird der erste Eintrag aus der Liste zurückgegeben. Übergibt man mit *iNext* ein in der Liste geführten Device-Handle, wird der dem Device-Handle nachfolgende Eintrag ermittelt und zurückgegeben. Man kann auf diese Weise durch sukzessives Einsetzen des Rückgabewertes die Liste von vorne nach hinten durchlaufen und alle Einträge abrufen.
Rückgabewert	Wenn ein Eintrag gefunden wurde, wird mit dem Rückgabewert der Device-Handle geliefert. Ist das Ende der internen Liste erreicht, also der übergebene Device-Handle keinen Nachfolger hat, wird eine 0 zurückgegeben. Ist kein USB-Kanal geöffnet, wird FEUSB_ERR_EMPTY_DEVLIST zurückgeliefert. Im Fehlerfall liefert die Funktion einen Wert kleiner als Null zurück. Die Liste der Fehlercodes findet sich im Anhang.

Beispiel	`#include "feusb.h"`
	`...`
	`// ermittelt die DeviceIDs aller geöffneten USB-Kanäle`
	`char cValue[16];`
	`...`
	`int iNextHnd = FEUSB_GetDeviceList( 0 );  // den ersten Handle ermitteln`
	`while( iNextHnd > 0 )`
	`{  // hier DeviceID auslesen`
	`  int iBack = FEUSB_GetDevicePara( iNextHnd, “Device-ID“, cValue )`
	`  printf(„%s“, cValue);  // Ausgabe auf Bildschirm`
	`  iNextHnd = FEUSB_GetDeviceList( iNextHnd );  // nächsten Handle ermitteln`
	`}`
	`...`
	`...`

Tip	Beim Schließen aller geöffneten USB-Kanäle bedient man sich gerne einer Schleife, ähnlich der im oberen Beispiel. Nur muß man bedenken, dass man von einem geschlossenen Kanal keinen Nachfolger mehr ermitteln kann. In dem folgenden Codefragment wird gezeigt, wie man in einer Schleife alle geöffneten kanäle schließen kann:
	`...`
	`int iCloseHnd, iNextHnd;`
	
	`iNextHnd = FEUSB_GetDeviceList( 0 );  // den ersten Handle ermitteln`
	`while( iNextHnd > 0 )`
	`{`
	`  iCloseHnd = iNextHnd;`
	`  iNextHnd = FEUSB_GetDeviceList( iNextHnd );  // erst nächsten Handle ermitteln`
	`  FEUSB_CloseDevice( iCloseHnd );  // jetzt erst USB-Kanal zum Gerät schließen`
	`}`
	
	`...`

6.6.14. FEUSB_GetDeviceHnd

Funktion	Ermittelt von einem geöffneten USB-Kanal den Device-Handle.
Syntax	**int FEUSB_GetDeviceHnd(long nDeviceID)**
Beschreibung	Mit dieser Funktion kann auf einfache Weise der Device-Handle eines zuvor geöffneten USB-Kanals ermittelt werden. *nDeviceID* ist die Seriennummer des USB-Gerätes. Diese Funktion ist eine "Umkehrfunktion" zu **FEUSB_GetDevicePara**(iDevHnd, "Device-ID", cValue), die zum Device-Handle die Seriennummer des Gerätes am USB-Kanal ermittelt.
Rückgabewert	Wenn der Kanal zur übergebenen Seriennummer gefunden wurde, wird der Device-Handle (>0) zurückgeliefert. Konnte in der Device-Liste die gesuchte Seriennummer nicht gefunden werden, wird eine 0 zurückgegeben. Im Fehlerfall liefert die Funktion einen Wert kleiner als Null zurück. Die Liste der Fehlercodes findet sich im Anhang.

Beispiel	...
	#include "feusb.h"
	...
	...
	int iDevHnd = FEUSB_OpenDevice(nDevice);
	if(iDevHnd < 0)
	{
	// hier Code für den Fehlerfall
	}
	else
	{ // handle wird mittels DeviceID erneut ermittelt
	iDevHnd = FEUSB_GetDeviceHnd(nDevice);
	}

6.6.15. FEUSB_GetDevicePara

Funktion	Ermittelt von dem mit *iDevHnd* bestimmten USB-Kanal einen Parameter.
Syntax	**int FEUSB_GetDevicePara(int iDevHnd, char* cPara, char* cValue)**
Beschreibung	Die Funktion ermittelt den aktuellen Wert eines Parameters. *cPara* ist eine nullterminierte Zeichenkette mit der Parameterkennung *cValue* ist eine leere, nullterminierte Zeichenkette zur Rückgabe des Parameterwerts. Die Zeichenkette sollte wenigstens 128 Zeichen aufnehmen können.
Parameter-kennungen	Die Parameterkennungen sind: "Device-ID" - Seriennummer des USB-Gerätes in hexadezimal Darstellung „FamilyName“ - Name der Gerätefamilie des Device am USB-Kanal „DeviceName“ - Name des Device am USB-Kanal Die Groß-/Kleinschreibung ist nicht relevant.
Rückgabewert	Im fehlerfreien Fall liefert die Funktion den Wert 0 und im Fehlerfall einen Wert kleiner als Null zurück. Die Liste der Fehlercodes findet sich im Anhang.
Querverweis	Weitere Informationen in: 8.2. Liste der Parameterkennungen.

Beispiel	...
	#include "feusb.h"
	...
	...
	char cValue[128];
	long nDeviceID;
	...
	if(FEUSB_GetDevicePara(iDevHnd, "Device-ID", cValue) == 0)
	{
	// hier Code zum Anzeigen des Parameters
	...
	// oder Umwandlung in DWORD
	sscanf((const char*)cValue, "%lx", &nDeviceID);
	}
	...
	...
	}

6.6.16. FEUSB_SetDevicePara

<table>
<tr><td>Funktion</td><td colspan="5">Setzt einen Parameter eines USB-Kanals auf einen neuen Wert.</td></tr>
<tr><td>Syntax</td><td colspan="5">int FEUSB_SetDevicePara(int iDevHnd, char* cPara, char* cValue)</td></tr>
<tr><td>Beschreibung</td><td colspan="5">Die Funktion übergibt an den mit iDevHnd benannten USB-Kanal einen neuen Parameter.

cPara ist eine nullterminierte Zeichenkette mit der Parameterkennung.

cValue ist eine nullterminierte Zeichenkette mit dem neuen Parameterwert.</td></tr>
<tr><td></td><td>Parameterkennung</td><td>Wertebereich</td><td>Defaultwert</td><td>Einheit</td><td>Kommentar</td></tr>
<tr><td></td><td>TIMEOUT</td><td>0...99999</td><td>1000</td><td>ms</td><td>Kann nur temporär für geöffneten Kanal gesetzt werden.

Kann global für alle geöffneten USB-Kanäle gesetzt werden, wenn iDevHnd = 0 ist.</td></tr>
<tr><td></td><td>EXCLUSIVEACCESS

(nur für Windows)</td><td>0, 1</td><td>1</td><td>-</td><td>Aktiviert (1) oder deaktiviert (0) den exklusiven Zugriff auf USB-Leser. Diese Aktion wirkt sich auf alle danach geöffneten USB-Leser aus. Deshalb muss iDevHnd auf 0 gesetzt werden.</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Rückgabewert</td><td colspan="5">Wenn der USB-Kanal mit dem neuen Parameterwert fehlerfrei initialisiert werden konnte, wird eine 0 zurückgeliefert. Im Fehlerfall liefert die Funktion einen Wert kleiner als Null zurück. Die Liste der Fehlercodes findet sich im Anhang.</td></tr>
<tr><td>Querverweis</td><td colspan="5">Weitere Informationen in: 8.2. Liste der Parameterkennungen.</td></tr>
<tr><td>Beispiel</td><td colspan="5"></td></tr>
</table>

6.6.17. FEUSB_AddEventHandler

<table>
<tr><td>Funktion</td><td colspan="4">Eine Ereignisbehandlungsmaßnahme wird installiert</td></tr>
<tr><td>Syntax</td><td colspan="4">int FEUSB_AddEventHandler(int iDevHnd, FEUSB_EVENT_INIT* pInit)</td></tr>
<tr><td>Beschreibung</td><td colspan="4">Die Funktion installiert eine von drei möglichen Ereignisbehandlungsmethode. Diese Methode kommt dann zur Anwendung, wenn ein Event auftritt, für das die Methode installiert wurde. Auf diese Weise ist eine asynchrone Reaktion auf Ereignisse in einem Applikationsprogramm möglich.

Die Ereignisbehandlungsmethode wird für die mit iDevHnd identifizierten Kanal oder global (iDevHnd = 0) eingerichtet.

Z. Zt können nur globale Ereignisbehandlungsmethoden installiert werden.</td></tr>
<tr><td></td><td>Ereignis</td><td colspan="3">Beschreibung</td></tr>
<tr><td></td><td>FEUSB_CONNECT_EVENT</td><td colspan="3">Signalisierung beim Einstecken des Geräts</td></tr>
<tr><td></td><td>FEUSB_DISCONNECT_EVENT</td><td colspan="3">Signalisierung beim Entfernen des Geräts</td></tr>
<tr><td></td><td colspan="4">1. Methode: Nachricht an Thread (nur für Windows; nicht für Visual Basic)
Diese Methode verwendet man für den Nachrichtenaustausch zwischen Threads[1]. Der Thread ermittelt mit der API-Funktion GetCurrentThreadID() den Thread-Identifier und übergibt diesen als Parameter dwThreadID in der FEUSB_EVENT_INIT-Struktur.
Der Thread muß für den Empfang der Nachricht, die von FEUSB mit der API-Funktion PostThreadMessage(..) verschickt wurde, eine Nachrichtenbehandlungsfunktion bereitstellen. Der Nachrichtencode ist frei wählbar.
Die FEUSB_EVENT_INIT-Struktur wird wie folgt ausgefüllt:
uiFlag = FEUSB_THREAD_ID
uiUse = FEUSB_xyz_EVENT // siehe Defines FEUSB.H
uiMsg = WM_USER + ... // frei wählbar, aber oberhalb von WM_USER[2]
dwThreadID = GetCurrentThreadID()

Die MessageMap-Funktion in der Applikation bekommt folgende Parameter übergeben:</td></tr>
<tr><td></td><td>Ereignis</td><td>Kanal</td><td>1. Parameter (wParam)</td><td>2. Parameter (lParam)</td></tr>
<tr><td></td><td rowspan="2">Connect</td><td>nicht geöffnet</td><td>0</td><td>DeviceID</td></tr>
<tr><td></td><td>geöffnet</td><td>DeviceHnd</td><td>DeviceID</td></tr>
<tr><td></td><td rowspan="2">Disconnect</td><td>nicht geöffnet</td><td>0</td><td>0</td></tr>
<tr><td></td><td>geöffnet</td><td>DeviceHnd</td><td>DeviceID</td></tr>
</table>

[1] Paralleler, vom Applikationsprogramm unabhängiger Ausführungspfad. Auch das Applikationsprogramm ist ein Thread.

[2] Siehe Windows-Dokumentation zur Platform-SDK

	2. Methode: Nachricht an Fenster (nur für Windows; nicht für Visual Basic)
	
	Diese Methode verwendet man, wenn die Nachricht direkt an ein Fenster geschickt werden soll. Von dem betreffenden Fenster wird mit der API-Funktion GetWindow(..)[1] der Handle ermittelt und als Parameter hwndWnd in der **FEUSB_EVENT_INIT**-Struktur übergeben. Das Fenster muß für den Empfang der Nachricht, die von FEUSB mit der API-Funktion SendMessage(..) verschickt wurde, eine Nachrichtenbehandlungsfunktion bereitstellen. Der Nachrichtencode ist frei wählbar.
	Die **FEUSB_EVENT_INIT**-Struktur wird wie folgt ausgefüllt:
	uiFlag = FEUSB_WND_HWND
	uiUse = FEUSB_xyz_EVENT // siehe Defines FEUSB.H
	uiMsg = WM_USER + ... // frei wählbar, aber oberhalb von WM_USER[2]
	hwndWnd = GetWindow(...)
	Die MessageMap-Funktion erhält dieselben Parameter wie die der ersten Methode.
	
	3. Methode: Aufruf einer Callback-Funktion
	
	Mit der Callback-Methode wird ein Funktionszeiger für ein Ereignis installiert. Tritt das Ereignis ein, wird die Funktion von FEUSB aufgerufen. Der Inhalt der Funktion kann frei bestimmt werden. Die Übergabeparameter sind allerdings entsprechend der 1. Methode festgelegt.
	Die **FEUSB_EVENT_INIT**-Struktur wird wie folgt ausgefüllt:
	uiFlag = FEUSB_CALLBACK
	uiUse = FEUSB_xyz_EVENT // siehe Defines FEUSB.H
	uiMsg wird nicht benötigt
	cbFct = (void*)&IhrFunktionsName
	
	Eine installierte Ereignisbehandlungsmethode muß wieder mit der Funktion **FEUSB_DelEventHandler** entfernt werden.
	
	Beim Schließen eines USB-Kanals gehen alle für diesen Kanal installierten Ereignisbehandlungsmethoden verloren.

Querverweis	Weitere Informationen in: 6.4. Ereignissignalisierung
Rückgabewert	Im fehlerfreien Fall liefert die Funktion Null und im Fehlerfall einen Wert kleiner als Null zurück. Die Liste der Fehlercodes findet sich im Anhang.

Beispiel	#include "feusb.h"
	...
	// Message-Handler für Events einrichten
	FEUSB_EVENT_INIT Init;
	
	Init.hwndWnd = this->GetSafeHwnd();
	Init.uiFlag = FEUSB_WND_HWND;
	Init.uiUse = FEUSB_DEV_DISCONNECT_EVENT;// Message immer, wenn Gerät abgezogen
	Init.uiMsg = WM_USER_DEVICE_DISCONNECT;
	FEUSB_AddEventHandler(0, &Init);
	
	Init.uiUse = FEUSB_DEV_CONNECT_EVENT; // Message immer, wenn Gerät eingestöpselt
	Init.uiMsg = WM_USER_DEVICE_CONNECT;
	FEUSB_AddEventHandler(0, &Init);

[1] Bei Verwendung der MFC-Klasse CWnd kann auch die Methode GetSafeHwnd() benutzt werden

[2] Siehe Windows-Dokumentation zur Platform-SDK

6.6.18. FEUSB_DelEventHandler

Funktion	Eine Ereignisbehandlungsmaßnahme wird entfernt
Syntax	**int FEUSB_DelEventHandler(int iPortHnd, FEUSB_EVENT_INIT* pInit)**

Beschreibung	Die Funktion entfernt eine zuvor mit **FEUSB_AddEventHandler** installierte Ereignisbehandlungsmaßnahme. In der **FEUSB_EVENT_INIT**-Struktur spezifiziert man die zu entfernende Ereignisbehandlungsmaßnahme im Detail.
	
	Entfernung der 1. Methode: Nachricht an Thread (nur für Windows; nicht für Visual Basic)
	Die **FEUSB_EVENT_INIT**-Struktur wird wie folgt ausgefüllt:
	uiFlag = FEUSB_THREAD_ID
	uiUse = FEUSB_xyz_EVENT // siehe Defines in FEUSB.H
	uiMsg wird nicht benötigt
	dwThreadID = GetCurrentThreadID()
	
	Entfernung der 2. Methode: Nachricht an Fenster (nur für Windows; nicht für Visual Basic)
	Die **FEUSB_EVENT_INIT**-Struktur wird wie folgt ausgefüllt:
	uiFlag = FEUSB_WND_HWND
	uiUse = FEUSB_xyz_EVENT // siehe Defines in FEUSB.H
	uiMsg wird nicht benötigt
	hwndWnd = GetWindow(...)
	
	Entfernung der 3. Methode: Aufruf einer Callback-Funktion
	Die **FEUSB_EVENT_INIT**-Struktur wird wie folgt ausgefüllt:
	uiFlag = FEUSB_CALLBACK
	uiUse = FEUSB_xyz_EVENT // siehe Defines FEUSB.H
	uiMsg wird nicht benötigt
	cbFct = (void*)&IhrFunktionsName

Querverweis	Weitere Informationen in: 6.4. Ereignissignalisierung
Rückgabewert	Im fehlerfreien Fall liefert die Funktion Null und im Fehlerfall einen Wert kleiner als Null zurück. Die Liste der Fehlercodes findet sich im Anhang.
Beispiel	#include "feusb.h" ... // Message-Handler für Events entfernen FEUSB_EVENT_INIT Init; Init.hwndWnd = this->GetSafeHwnd(); Init.uiFlag = FEUSB_WND_HWND; Init.uiUse = FEUSB_DEV_DISCONNECT_EVENT; Init.uiMsg = 0; FEUSB_DelEventHandler(0, &Init); Init.uiUse = FEUSB_DEV_CONNECT_EVENT; Init.uiMsg = 0; FEUSB_DelEventHandler(0, &Init);

6.6.19. FEUSB_Transceive

Funktion	Funktion zur Kommunikation (Transmit und Receive) über einen USB-Kanal.
Syntax	**int FEUSB_Transceive(int iDevHnd, char* cInterface, int iDir, UCHAR* cSendData, int iSendLen, UCHAR* cRecData, int iRecLen)**
Beschreibung	Die Funktion schickt die in *cSendData* enthaltenen Daten der Länge *iSendLen* über das in *cInterface* benannte Geräte-Interface an das angeschlossene USB-Gerät. Die Empfangsdaten werden in *cRecData* hinterlegt. Mit dem Parameter *iRecLen* muß die maximale Länge des Puffers *cRecData* angegeben werden. Übersteigt die Anzahl der empfangenen Zeichen den in *iRecLen* übergebenen Wert, wird die Funktion mit einem Fehler beendet. Es gibt zwei Interfaces zu unterscheiden: • *cInterface* = "OBID-RCI": USB-Protokoll der ersten OBID i-*scan*® USB-Leser (ID ISC.PRH100-U und ID ISC.MR100-U). Der Aufbau des Protokolls ist wegen der Komplexität undokumentiert. USB-Protokolle können nur über die Bibliothek ID FEISC an den Leser gesendet werden. Dieses Interface wird von Linux nicht unterstützt. Der Parameter *iDir* bestimmt die Datenrichtung: *iDir* = 0x01 IN-Transfer (der Host holt Daten vom Gerät) *iDir* = 0x02 OUT-Transfer (der Host schickt Daten an das Gerät) • *cInterface* = "OBID-RCI2": USB-Protokoll der zweiten Generation für OBID i-*scan*® USB-Leser. Der Aufbau des Protokolls ist identisch mit dem im Systemhandbuch zum Leser dokumentierten Protokollrahmen. Der Parameter *iDir* hat keine Bedeutung und kann zu 0 gesetzt werden.
Interfaces	OBID-RCI, OBID-RCI2
Rückgabewert	Im fehlerfreien Fall liefert die Funktion die Länge des Empfangsprotokolls und im Fehlerfall einen Wert kleiner als Null zurück. Die Liste der Fehlercodes findet sich im Anhang.
Beispiel	

6.6.20. FEUSB_Transmit

Funktion	Funktion zur Kommunikation über einen USB-Kanal.
Syntax	**int FEUSB_Transmit(int iDevHnd, char* cInterface, UCHAR* cSendData, int iSendLen)**
Beschreibung	Die Funktion schickt die in *cSendData* enthaltenen Daten der Länge *iSendLen* über das in *cInterface* benannte Geräte-Interface an das angeschlossene USB-Gerät. Es wird nur das *cInterface* = “OBID-RCI2“ unterstützt.
Interfaces	OBID-RCI2
Rückgabewert	Im fehlerfreien Fall liefert die Funktion Null und im Fehlerfall einen Wert kleiner als Null zurück. Die Liste der Fehlercodes findet sich im Anhang.
Beispiel	

6.6.21. FEUSB_Receive

Funktion	Funktion zur Kommunikation über einen USB-Kanal.
Syntax	**int FEUSB_Receive(int iDevHnd, char* cInterface, UCHAR* cRecData, int iRecLen)**
Beschreibung	Die Funktion erwartet über das in *cInterface* benannte Geräte-Interface vom angeschlossene USB-Gerät Empfangsdaten. Die Empfangsdaten werden in *cRecData* hinterlegt. Mit dem Parameter *iRecLen* muß die maximale Länge des Puffers *cRecData* angegeben werden. Übersteigt die Anzahl der empfangenen Zeichen den in *iRecLen* übergebenen Wert, wird die Funktion mit einem Fehler beendet. Es wird nur das *cInterface* = “OBID-RCI2“ unterstützt.
Interfaces	OBID-RCI2
Rückgabewert	Im fehlerfreien Fall liefert die Funktion die Länge des Empfangsprotokolls und im Fehlerfall einen Wert kleiner als Null zurück. Die Liste der Fehlercodes findet sich im Anhang.
Beispiel	

7. Dynamische Bindung unter C++

Für den Fall, dass eine Applikation die Funktionssammlung FEUSB dynamisch an die Applikation binden soll, muss die Bibliotheksdatei explizit geladen werden. Ein (programmtechnischer) Nachteil ist, dass jeder Funktionsaufruf in die DLL dann über einen Funktionszeiger erfolgen muss.

Folgende Schritte sind zu tun:

1. die DLL zur Laufzeit laden:

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

2. einen Funktionszeiger zur Laufzeit ermitteln:

```
LPFN_FEUSB_GET_DLL_VERSION lpfn= GetProcAddress(hLib,
                                                "FECOM_GetDLLVersion");
```

3. die Funktion ausführen:

```
char cVersion[256];

lpfn(cVersion);
```

4. die geladene DLL muß vor dem Beenden der Applikation wieder entladen werden:

```
if(hLib != NULL)
  FreeLibrary(hLib);
```

Für jede Funktion in der FEUSB ist ein Prototyp für einen Funktionszeiger in der Headerdatei feusb.h definiert. Im obigen Beispiel ist dies `LPFN_FEUSB_GET_DLL_VERSION`. Praktischerweise ermittelt und speichert man sich jeden benötigten Funktionszeiger für die gesamte Laufzeit des Programms.

Tipp: benutzt man die C++ Klassenbibliothek ID FEDM, dann kann man die Funktion GetFeUsbFunction der Basisklasse FEDM_Base zur Zeigerermittlung verwenden. Dabei wird die DLL automatisch mit dem ersten Funktionsaufruf geladen und im Destruktor der Klasse wieder aus dem Adressraum der Applikation entfernt. Der Funktion GetFeUsbFunction übergibt man als Parameter eine Konstante, die die Funktion identifiziert. Diese Konstanten sind in der Headerdatei feusb.h definiert.

Beispiel:

```
FEDM_ISCReader m_Reader; // FEDM_ISCReader ist von FEDM_Base abgeleitet

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. Anhang

8.1. Fehlercodes

Fehler-Konstante	Wert	Beschreibung
FEUSB_ERR_EMPTY_DEVICELIST	-1100	Device-Handleliste ist leer (keine Device-Objekte angelegt)
FEUSB_ERR_EMPTY_SCANLIST	-1101	Scanliste ist leer (keine USB-Geräte verfügbar)
FEUSB_POINTER_IS_NULL	-1102	ein übergebener Pointer ist NULL
FEUSB_ERR_NO_MORE_MEM	-1103	Speichermangel ist aufgetreten
FEUSB_ERR_SET_CONFIGURATION	-1104	die USB-Konfiguration konnte nicht gesetzt werden
FEUSB_ERR_KERNEL	-1105	es ist ein Fehler innerhalb des Kernel-Treibers beim USB-Transfer aufgetreten.
FEUSB_ERR_UNSUPPORTED_OPTION	-1106	Nicht unterstützte Option
FEUSB_ERR_UNSUPPORTED_FUNCTION	-1107	Nicht unterstützte Funktion
FEUSB_ERR_NO_FEIG_DEVICE	-1110	USB-Gerät hat keine FEIG-Kennung
FEUSB_ERR_SEARCH_MISMATCH	-1111	es wurde(n) kein(e) Gerät(e) mit den angegebenen Suchkriterien gefunden
FEUSB_ERR_NO_DEVICE_FOUND	-1112	es wurde(n) kein(e) Gerät(e) gefunden
FEUSB_ERR_DEVICE_IS_SCANNED	-1113	das Gerät ist schon in der Scanliste
FEUSB_ERR_SCANLIST_OVERFLOW	-1114	Scanliste ist mit 127 Einträgen gefüllt und es wird versucht, einen weiteren Eintrag hinzuzufügen
FEUSB_ERR_UNKNOWN_HND	-1120	der übergebene Device-Handle ist unbekannt
FEUSB_ERR_HND_IS_NULL	-1121	der übergebene Device-Handle ist 0
FEUSB_ERR_HND_IS_NEGATIVE	-1122	der übergebene Device-Handle ist negativ
FEUSB_ERR_NO_HND_FOUND	-1123	kein Device-Handle in Device-Handleliste gefunden
FEUSB_ERR_TIMEOUT	-1130	Timeout beim Lesen vom USB-Kanal
FEUSB_ERR_NO_SENDDATA	-1131	keine Sendedaten übergeben
FEUSB_ERR_UNKNOWN_INTERFACE	-1132	unbekanntes Interface
FEUSB_ERR_UNKNOWN_DIRECTION	-1133	unbekannte Datenrichtung
FEUSB_ERR_RECBUF_TO_SMALL	-1134	der Empfangspuffer ist zu klein
FEUSB_ERR_SENDDATA_LEN	-1135	die Länge der Senddaten ist falsch angegeben

Fehler-Konstante	Wert	Beschreibung
FEUSB_ERR_UNKNOWN_DESCRIPTOR_TYPE	-1136	unbekannter Deskriptor-Typ
FEUSB_ERR_DEVICE_NOT_PRESENT	-1137	das USB-Gerät ist z. Zt nicht am USB-Port angeschlossen
FEUSB_ERR_TRANSMIT_PROCESS	-1138	Fehler beim Übertragen der Sendedaten
FEUSB_ERR_DEVICE_NOT_SCANNED	-1140	das Gerät wurde zuvor nicht eingescannt
FEUSB_ERR_DEVHND_NOT_IN_SCANLIST	-1141	das Gerät ist nicht in der Scanliste eingetragen
FEUSB_ERR_DRIVERLIST	-1142	es konnte keine Treiberliste im USB-Treiber erzeugt werden
FEUSB_ERR_UNKNOWN_PARAMETER	-1150	Übergabeparameter ist nicht bekannt
FEUSB_ERR_PARAMETER_OUT_OF_RANGE	-1151	Übergabeparameter zu groß oder zu klein
FEUSB_ERR_ODD_PARAMETERSTRING	-1152	eine nicht unterstützte Option wurde per Übergabeparameter aufgerufen
FEUSB_ERR_INDEX_OUT_OF_RANGE	-1153	der übergebene Listenindex liegt nicht im gültigen Wertebereich von 1..65535
FEUSB_ERR_UNKNOWN_SCANOPTION	-1154	unbekannte Scanoption
FEUSB_ERR_UNKNOWN_ERRORCODE	-1155	unbekannter Fehlercode
FEUSB_ERR_DEV_DESC_LENGTH	-1160	Längenfehler im Device-Deskriptor
FEUSB_ERR_CFG_DESC_LENGTH	-1161	Längenfehler im Configuration-Deskriptor
FEUSB_ERR_INTF_DESC_LENGTH	-1162	Längenfehler im Interface-Deskriptor
FEUSB_ERR_ENDP_DESC_LENGTH	-1163	Längenfehler im Endpoint-Deskriptor
FEUSB_ERR_HID_DESC_LENGTH	-1164	Längenfehler im HID-Deskriptor
FEUSB_ERR_STRG_DESC_LENGTH	-1165	Längenfehler im String-Deskriptor
FEUSB_ERR_READ_DEV_DESCRIPTOR	-1166	Lesefehler Device-Deskriptor
FEUSB_ERR_READ_CFG_DESCRIPTOR	-1167	Lesefehler Configuration-Deskriptor
FEUSB_ERR_READ_STRG_DESCRIPTOR	-1168	Lesefehler String-Deskriptor
FEUSB_ERR_MAX_INTERFACES	-1170	das Gerät hat zu viele Interfaces
FEUSB_ERR_MAX_ENDPOINTS	-1171	Das Gerät hat zu viele Endpoints
FEUSB_ERR_MAX_STRINGS	-1172	Das Gerät hat zu viele Strings

8.2. Liste der Parameterkennungen

Parameterkennung	Wertebereich	Default	Einheit	Beschreibung
DeviceHnd	268435456... 536870911	-	-	Device-Handle Verwendung: • FEUSB_GetScanListPara
Device-ID	Hexadezimal: 0x00000001 ... 0xFFFFFFFF Dezimal: 1 ... 4294967295	-	-	Seriennummer im USB-Gerät Verwendung: • FEUSB_GetScanListPara • FEUSB_GetDevicePara
Timeout	0...99999	1000	ms	Maximale Wartezeit auf Empfangsprotokoll Verwendung: • FEUSB_SetDevicePara
FamilyName	s. 8.5. Liste der cFamilyName in der FEUSB_SCANSEARCH-Struktur	-	-	Name der Leserfamiliy Verwendung: • FEUSB_GetScanListPara • FEUSB_GetDevicePara
DeviceName	s. 8.6. Liste der cDeviceName in der FEUSB_SCANSEARCH-Struktur	-	-	Name des Lesers Verwendung: • FEUSB_GetScanListPara • FEUSB_GetDevicePara
Present	0, 1	-	-	Abfrage der Geräte-Präsenz am USB-Kanal Verwendung: • FEUSB_GetScanListPara
ExclusiveAccess	0, 1	1	-	Aktiviert (1) oder deaktiviert (0) den exklusiven Zugriff auf USB-Leser. Diese Aktion wirkt sich auf alle danach geöffneten USB-Leser aus. Verwendung: • FEUSB_SetDevicePara Nur für Windows verwendbar

8.3. Liste der Konstanten für die FEUSB_EVENT_INIT-Struktur

Die Konstantendefinitionen sind in der Datei FEUSB.H bzw. FEUSB.BAS enthalten.

Konstante	Wert	Verwendung	Beschreibung
FEUSB_THREAD_ID	1	uiFlag	Ereignissignalisierung mit Thread-Nachricht
FEUSB_WND_HWND	2	uiFlag	Ereignissignalisierung mit Window-Nachricht
FEUSB_CALLBACK	3	uiFlag	Ereignissignalisierung mit Callback-Funktion
FEUSB_CONNECT_EVENT	1	uiUse	Signalisierung beim Einstecken des Lesers
FEUSB_DISCONNECT_EVENT	2	uiUse	Signalisierung beim Abziehen des Lesers

8.4. Liste der Konstanten für die FEUSB_SCANSEARCH-Struktur

Die Konstantendefinitionen sind in der Datei FEUSB.H, FEUSB.BAS bzw. FEUSB.PAS enthalten.

Konstante	Wert	Verwendung	Beschreibung
FEUSB_SEARCH_FAMILY	1	iMask	cFamilyName wird ausgewertet
FEUSB_SEARCH_PRODUCT	2	iMask	cDeviceName wird ausgewertet
FEUSB_SEARCH_DEVICEID	4	iMask	dwDeviceID wird ausgewertet

8.5. Liste der cFamilyName in der FEUSB_SCANSEARCH-Struktur[1]

Zeichenkette	Beschreibung
"OBID i-scan Proximity"	
"OBID i-scan Midrange"	
"OBID i-scan UHF Midrange"	
"OBID i-scan UHF Long-Range"	
"OBID classic-pro"	

8.6. Liste der cDeviceName in der FEUSB_SCANSEARCH-Struktur[2]

Zeichenkette	Beschreibung
"ID ISC.PRH100-U"	Leser der OBID i-*scan* Proximity-Familie
"ID ISC.PRH101-U"	Leser der OBID i-*scan* Proximity-Familie
"ID ISC.MR100-U"	Leser der OBID i-*scan* Midrange-Familie
"ID ISC.MR101-U"	Leser der OBID i-*scan* Midrange-Familie
"ID ISC.MR102-U"	Leser der OBID i-*scan* Midrange-Familie
"ID ISC.LR2500-A"	Leser der OBID i-*scan Long-R*ange-Familie
"ID ISC.LR2500-B"	Leser der OBID i-*scan Long-R*ange-Familie
"ID ISC.MRU102-U"	Leser der OBID i-*scan* UHF Midrange-Familie
"ID ISC.MRU200"	Leser der OBID i-*scan* UHF Midrange-Familie
"ID ISC.LRU3000"	Leser der OBID i-*scan* UHF Long-Range-Familie

[1] Liste wird in Zukunft erweitert
[2] Liste wird in Zukunft erweitert

"ID CPR.04-USB"	Leser der OBID *classic-pro*-Familie
"ID CPR40.xx-U"	Leser der OBID *classic-pro*-Familie
"ID CPR44.xx-U"	Leser der OBID *classic-pro*-Familie

8.7. Änderungshistorie

V4.02.02

- Windows/Windows CE:
  1. Fehlerkorrektur für Plug & Play bei Erkennung von USB-Sticks

V4.02.00

- Windows:
  1. Migration der Entwicklungsumgebung von Visual Studio 2008 zu Visual Studio 2010.
  2. Verbesserte Threadsicherheit
  3. DLL jetzt ohne MFC
  4. Erstes Release der 64-Bit Version
  5. Anbindung an Log-Manager
  6. Modifikationen am internen Plug-and-Play Mechanismus für verbessertes Event-Handling

- Windows CE:
  1. Verbesserte Threadsicherheit

- Linux:
  1. Keine Änderungen

- Erste Release-Version für Mac OS X, ab V10.7.3

V4.00.00

- Windows / Windows CE:
  1. Migration der Entwicklungsumgebung von Visual Studio 6 zu Visual Studio 2008.
  2. Anpassung der Callback Funktionsdeklarationen in ***struct _FEUSB_EVENT_INIT*** bzgl. der Calling-Konvention. Daher ist diese Version nicht kompatibel zur Vorgängerversion und nicht kompatibel mit Anwendungen, die mit der Vorgängerversion kompiliert wurden. Codeanpassungen sind nicht notwendig, aber Anwendungen müssen neu kompiliert werden.
  3. Neuer Fehlercode -1138 (Fehler beim Übertragen der Sendedaten)
  4. Erweiterte interne Fehlerbehandlung

5. Fehlerbehebung (nur Windows CE): Umwandlung der Device-ID von Unicode-String in unsigned long Wert

- Linux:
  1. Fehlerbehebung bei Deaktivierung des Plug&Play-Threads

V3.06.01

Windows und Windows CE

- Erweiterte interne Fehlerbehandlung
- Unterstützung für einheitliche Device-ID (für Linux bereits enthalten)

V3.05.00

- Support für **Windows 7** (x86 und x64) bei Verwendung des Kerneltreibers OBIDUSB V2.5
- **Linux**: Regeldatei **41-feig.rules** ermöglicht eine Installation ohne Root-Rechte

V3.04.00

- **Windows** und **Windows CE**: Modifikationen am internen Plug-and-Play Mechanismus für verbessertes Event-Handling

V3.03.04

- neue Funktion **FEUSB_GetDrvVersion** zur Abfrage der Kerneltreiber-Version
- Ausgabe von Fehlertexten zu Fehlercodes des Kerneltreibers
- Fehlerkorrektur für Windows 2000: max. Transferlänge auf 4096 begrenzt.

V3.03.02

- Exklusive Verwendung der USB-Leser in einer Applikation. Dies ist eine Änderung gegenüber früheren Versionen, in denen mehrere Applikationen simultan mit einem USB-Leser kommunizieren konnten.

  Die simultane Nutzung kann aber mit dem Parameter “ExclusiveAccess“ über die Funktion **FEUSB_SetDevicePara** gesteuert werden.
- Unterstützung für Bulk-Transfer neuerer Leser-Generationen

V3.00.00

- Kompatibilität mit neuem Kerneltreiber für 32-Bit Vista

V2.05.00

- Erste Linux-Version

V2.03.02

- Die neue Version ist zu 100% rückwärtskompatibel zur Vorgängerversion.
- Bugfix für DeviceID > 0x7FFFFFFF
- Umsetzung des Fehlercodes 0xE000100C in FEUSB_ERR_TIMEOUT (-1130)

V2.03.01

- Unterstützung für neue USB-Protokolle.
- Neue Funktionen: **FEUSB_Transmit**, **FEUSB_Receive**, **FEUSB_SetDevicePara**
- Neuer Parameter: TIMEOUT
- Neuer Fehlercode: -1106

V2.01.00

- Fehlerkorrektur behebt 'Kommunikationsklemmer' nach zuvor fehlerhafter Kommunikation.

V2.00.00

- Die neue Version unterstützt Windows XP.
- Der Kerneltreiber OBIDUSB.SYS bzw. OBIDUSB9.SYS muß die Version 2.00 haben.
- neue Fehlercodes: -1166, -1167, -1168
- neue Parameter für die Funktion **FEUSB_GetDevicePara**: DeviceName, FamilyName
- neue Parameter für die Funktion **FEUSB_GetScanListPara**: DeviceName, FamilyName
- Connect-Benachrichtigung an Applikationen mit Device-ID

V1.02.00

- Die neue Version ist fast zu 100% rückwärtskompatibel zur Vorgängerversion 1.00.00. Die einzige Inkompatibilität besteht in der Werteverschiebung des Scan-Parameters FEUSB_SCAN_ALL von 0x03 nach 0x0F.
- Benachrichtigung an Applikationen auch für Leser, die noch nicht in der Scanliste eingetragen sind.
- Der Device-Handle hat nun einen Offset von 0x10000000 (dezimal 268435456) für die direkte Verwendung mit anderen FEIG-DLLs (z.B. FEISC.DLL).
- Der neue Scan-Parameter FEUSB_SCAN_NEW erlaubt die Suche nach ungescannten Lesern.
- Neue Funktion: **FEUSB_GetLastError**.
- Neuer Errorcode: -1114 für Overflow der Scanliste.

V1.01.00

- interne Version

V1.00.00

- Die Version ist, bedingt durch einen neuen Kerneltreiber, nicht mehr kompatibel zur Vorgängerversion.
- Neue Funktionen: FEUSB_AddEventHandler, FEUSB_DelEventHandler, FEUSB_IsDevicePresent

V0.99.01

- Unterstützung für Windows 98, 98SE und 2000
- Unterstützung für Open- und Universal-Host-Controller-Interface
- neuer Fehlercode: -1103