OBID®
OBID i-scan®

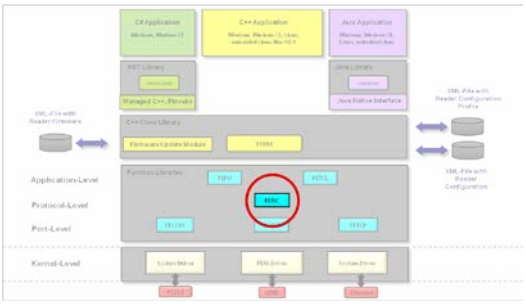
HANDBUCH

ID FEISC

Version 7.03.00

Software-Support für

OBID i-*scan*® und OBID® *classic-pro*



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	
Android	X		Auf Anfrage
Apple Max OS X	-	X	ab V10.7.3, Architektur x86_64

Hinweis

 FEIG ELECTRONIC GmbH
Lange Straße 4
D-35781 Weilburg-Waldhausen
Germany
Tel.: +49 6471 3109-0
http://www.feig.de

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

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.

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 FEISC 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 ID FEISC für die Entwicklung eigener Anwendungsprogramme oder Programmbibliotheken zu verwenden und Sie dürfen die Laufzeitdatei FEISC.DLL, FEISCCE.DLL, LIBFEISC.x.y.z.DYLIB[1] oder LIBFEISC.SO.x.y.z[1] ohne Abgabe von Lizenzgebühren vertreiben, unter der Voraussetzung, dass diese Anwendungsprogramme oder Programmbibliotheken dazu dienen, Geräte und / oder Anlagen anzusteuern oder zu betreiben, die von FEIG entwickelt und / oder vertrieben werden.
4. Dieses Lizenzmaterial kann Software Dritter enthalten. Bei Verwendung dieser Drittanbieter-Software gelten die im Abschnitt Lizenzbestimmungen Dritter genannten Lizenzbestimmungen.

§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.

[1] x.y.z repräsentiert die Versionsnummer

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.
   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.

Lizenzbestimmungen Dritter

Lizenzbestimmung der openSSL Organisation

Die nachfolgende Lizenbestimmung ist relevant für den Fall, dass verschlüsselte Datenübertragung zur Anwendung kommt.

```
LICENSE ISSUES
==============

The OpenSSL toolkit stays under a dual license, i.e. both the conditions of the OpenSSL License and
the original SSLeay license apply to the toolkit. See below for the actual license texts. Actually
both licenses are BSD-style Open Source licenses. In case of any license issues related to OpenSSL
please contact openssl-core@openssl.org.

OpenSSL License
---------------

/* ====================================================================
Copyright (c) 1998-2008 The OpenSSL Project.  All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions
and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions
and the following disclaimer in the documentation and/or other materials provided with the
distribution.

3. All advertising materials mentioning features or use of this software must display the following
acknowledgment:
"This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit.
(http://www.openssl.org/)"

4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to endorse or promote products
derived from this software without prior written permission. For written permission, please contact
openssl-core@openssl.org.

5. Products derived from this software may not be called "OpenSSL" nor may "OpenSSL" appear in their
names without prior written permission of the OpenSSL Project.

6. Redistributions of any form whatsoever must retain the following acknowledgment:
"This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit
(http://www.openssl.org/)"

THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES,
INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE OpenSSL PROJECT OR ITS CONTRIBUTORS BE
LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF
THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
==================================================================== */

This product includes cryptographic software written by Eric Young (eay@cryptsoft.com). This product
includes software written by Tim Hudson (tjh@cryptsoft.com).


Original SSLeay License
-----------------------

Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com) All rights reserved.

This package is an SSL implementation written by Eric Young (eay@cryptsoft.com).
The implementation was written so as to conform with Netscapes SSL.

This library is free for commercial and non-commercial use as long as the following conditions are
aheared to.  The following conditions apply to all code found in this distribution, be it the RC4,
RSA, lhash, DES, etc., code; not just the SSL code.  The SSL documentation included with this
```

distribution is covered by the same copyright terms except that the holder is Tim Hudson (tjh@cryptsoft.com).

Copyright remains Eric Young's, and as such any Copyright notices in the code are not to be removed.
If this package is used in a product, Eric Young should be given attribution as the author of the parts of the library used.
This can be in the form of a textual message at program startup or in documentation (online or textual) provided with the package.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. All advertising materials mentioning features or use of this software must display the following acknowledgement:
"This product includes cryptographic software written by Eric Young (eay@cryptsoft.com)"
The word 'cryptographic' can be left out if the rouines from the library being used are not cryptographic related :-).

4. If you include any Windows specific code (or a derivative thereof) from the apps directory (application code) you must include an acknowledgement:
"This product includes software written by Tim Hudson (tjh@cryptsoft.com)"

THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The licence and distribution terms for any publically available version or derivative of this code cannot be changed. i.e. this code cannot simply be copied and put under another distribution licence [including the GNU Public Licence.]

Inhalt:

1. Einleitung

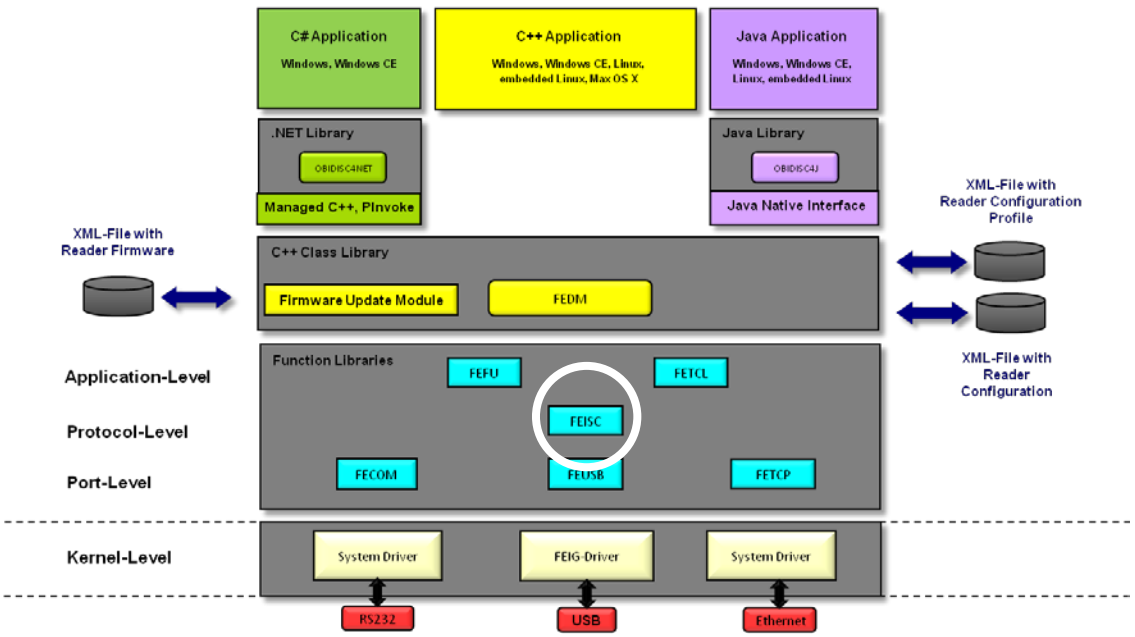
Das Supportpaket ID FEISC dient zur Unterstützung bei der Programmierung von Anwendungs-Software, die OBID i-*scan*®- und/oder OBID® *classic-pro* Leser integrieren und unterstützt die Sprachen ANSI-C, ANSI-C++ und prinzipiell jede andere Sprache, die C-Funktionen aufrufen kann.

Dieses Supportpaket enthält eine einfache Funktionsschnittstelle zum OBID®-Leser, indem für jedes in den Systemhandbüchern der OBID®-Leserfamilien dokumentierte Protokoll eine eigene Funktion existiert. Zur Datenübertragung wird eine Bibliothek der Transportschicht (FECOM, FEUSB, FETCP) zur Laufzeit dynamisch gebunden.

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	
Android	X		Auf Anfrage
Apple Max OS X	-	X	ab V10.7.3, Architektur x86_64

Die Bibliothek FEISC bildet die zweite Ebene in dem mehrschichtigen, hierarchisch strukturierten Aufbau von FEIG-Bibliotheken. Mit ihr wird ausschließlich die Protokollschicht (Aufbau/Zerlegung von Protokollrahmen, CRC-Prüfung, Längenprüfung) realisiert. Das nachfolgende Bild zeigt eine Übersicht über alle Bibliotheken.



Programmierer, die sich für diese Schicht als Integrationsoption entscheiden oder entscheiden müssen (Pascal, Delphi, VB6, LabView), können sich auf grundlegende Kommunikationsaufgaben konzentrieren. Mag die Steuerung der Leser noch einfach sein, sind die Transponder-Kommandos und das Handling der Leser-Betriebsarten Buffered-Read-Mode oder Notification-Mode doch mit erheblichem Aufwand verbunden und es gilt abzuwägen, ob der Einstieg für C++ Programmierer auf diesem Level zwingend notwendig ist.

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

1.1.2. Windows CE

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

1.1.3. Linux

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

1.1.4. Mac OS X

Datei[2]	Verwendung
LIBFEISC.x.y.z.dylib	Funktions-Bibliothek mit allen Funktionen
FEISC.H	Header-Datei für C/C++-Projekte

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

2. Änderungen gegenüber der Vorversion

- Neue Funktion **FEISC_0x1E_TableDataExchange**
- Support für Android auf Anfrage verfügbar

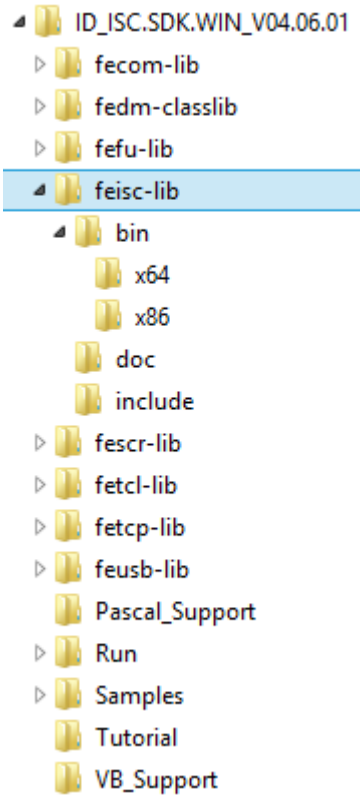
Bitte beachten Sie auch die Änderungshistorie im Anhang.

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 feisc-lib.

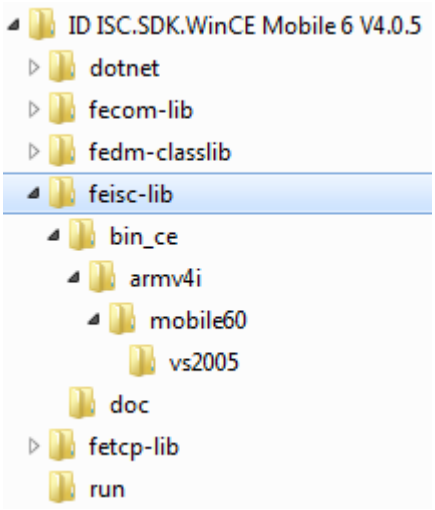
3.1. 32- und 64-Bit Windows XP/Vista/7/8



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

- Kopieren Sie FEISC.DLL in das Verzeichnis des Anwendungsprogramms (empfohlen) oder in das Systemverzeichnis von Windows.
- Kopieren Sie FEISC.LIB in das Projekt- oder LIB-Verzeichnis
- Kopieren Sie FEISC.H in das Projekt- oder INCLUDE-Verzeichnis
- Für den Fall, dass verschlüsselte Datenübertragung zur Anwendung kommt, muss auch die openSSL Bibliotheksdatei libeay32.dll in das Verzeichnis des Anwendungsprogramms kopiert werden. Bitte beachten Sie in diesem Fall auch die Lizenzbedingungen zu openSSL (http://www.openssl.org).

3.2. Windows CE

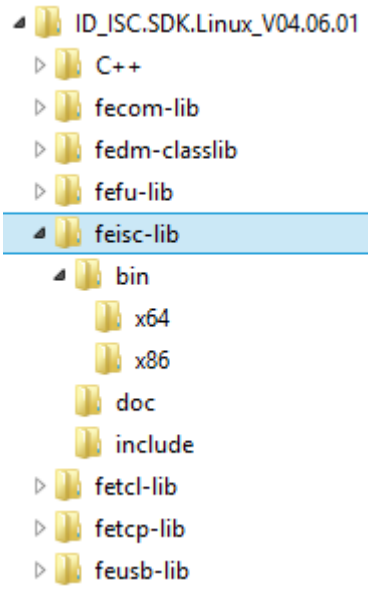


Wenn eigene Projekte nicht im SDK-Verzeichnis angelegt werden sollen, dann empfiehlt sich folgende Vorgehensweise:

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

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 libfeisc.so.x.y.z[3] im Verzeichnis /usr/lib bzw. /usr/lib64 durch folgende Aufrufe:

cd /usr/lib (für 64 Bit : /usr/lib64)

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

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

ldconfig

Für den Fall, dass verschlüsselte Datenübertragung zur Anwendung kommt, muss auch die openSSL Bibliotheksdatei libcrypto.so installiert sein. Bitte beachten Sie in diesem Fall auch die Lizenzbedingungen zu openSSL (http://www.openssl.org).

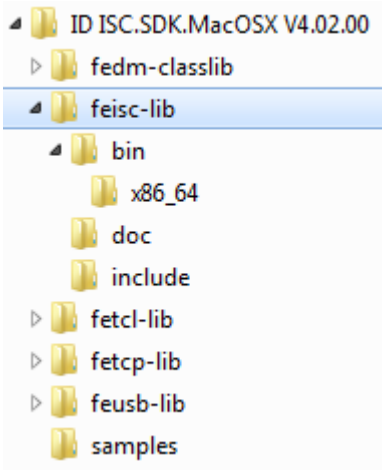
Anmerkung:

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.

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

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 libfeisc.x.y.z.dylib[4] im Verzeichnis /usr/local/lib durch folgende Aufrufe:

cd /usr/local/lib

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

ln –s libfeisc.x.dylib libfeisc.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.

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

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

ID FECOM und/oder ID FEUSB und/oder ID FETCP müssen ebenfalls in Ihr Projekt eingebunden werden, wenn Funktionen daraus aufgerufen werden.

Für den Fall, dass verschlüsselte Datenübertragung zur Anwendung kommt, muss auch die openSSL Bibliotheksdatei libeay32.dll (Windows) bzw. libcrypto.so (Linux) installiert sein. Bitte beachten Sie in diesem Fall auch die Lizenzbedingungen zu openSSL (http://www.openssl.org).

5. Programmierschnittstelle

5.1. Übersicht

Die Bibliothek FEISC kapselt für den Programmierer alle notwendigen Funktionen und Parameter zur einfachen Kommunikation mit Lesern der OBID i-*scan*®-Familie. Dadurch ist es möglich, in Verbindung mit den Supportpaketen ID FECOM, ID FETCP oder ID FEUSB alle Protokolle aus dem Systemhandbuch der OBID i-*scan*®- oder OBID® *classic-pro* Leserfamilie direkt mit einem Funktionsaufruf auszuführen.

Die Funktionen in FEISC sind ausschließlich für die interne Verwaltung, den Protokollaufbau, die Protokollzerlegung und eventuell notwendige Fehlerausgaben zuständig. Mit der FEISC allein kann keine Kommunikation mit einem OBID i-*scan*®- oder OBID® *classic-pro* Leser durchgeführt werden. Es kann aber eine Protokollausgabe initiiert und mittels FECOM über eine asynchrone, serielle Schnittstelle oder mittels FETCP über eine Ethernet-Verbindung bzw. FEUSB über den Universal Serial Bus (USB) mit einem OBID i-*scan*®- oder OBID® *classic-pro* Leser kommuniziert werden. Andere Schnittstellentreiber können über den Plug-In Mechanismus eingebunden werden.

Die Verwendung der FEUSB zur Kommunikation mit OBID® USB-Geräten ist dagegen zwingend.

Kernelemente der Bibliothek sind der Objekt-Manager und die zur Laufzeit erzeugten Leser-Objekte.

Der Objekt-Manager realisiert eine Selbstverwaltung, die ein Anwendungsprogramm davon befreit, irgendwelche Werte, Einstellungungen oder Sonstiges zwischenspeichern zu müssen: Er führt eine Liste mit allen erzeugten Leser-Objekten. Das Leser-Objekt ist der zentrale Programmteil, der die Protokoll-Funktionen ausführt und bei Verwendung der FECOM eine Verbindung zur seriellen Schnittstelle, bei Verwendung der FETCP ein TCP/IP-Server bzw. bei Verwendung der FEUSB ein Kanal zu einem USB-Gerät zugewiesen bekommt. Jedes Leser-Objekt verwaltet alle für seine Protokollaufgaben relevanten Einstellungen innerhalb seines lokalen Speichers.

Vor der ersten Verwendung muß ein Leser-Objekt angelegt werden. Dies wird von der Funktion **FEISC_NewReader** ausgeführt. Im fehlerfreien Fall erhält man mit dem Rückgabewert einen Handle, der vom Anwendungsprogramm als Zugriffsnummer verwendet wird. Nur mit diesem Handle ist eine eindeutige Identifikation des erzeugten Leser-Objekts möglich. Nutzt man die Selbstverwaltung, kann die Objekt-Liste mit der Funktion **FEISC_GetReaderList** abgerufen werden. Mit den Handles, die man damit sukzessive erhält, kann man anschließend mit der Funktion **FEISC_GetReaderPara** alle dieses Objekt betreffende Einstellungen auslesen.

Ein mit **FEISC_NewReader** erzeugtes Leser-Objekt muß unbedingt wieder mit der Funktion **FEISC_DeleteReader** aus dem Speicher entfernt werden.

Wird ein Anwendungsprogramm mehrfach aufgerufen, erhält jedes Programm (Instanz) mit dem Funktionsaufruf **FEISC_GetReaderList** eine leere Objekt-Liste. Dadurch wird eine Vermischung von Zugriffsrechten unter verschiedenen Programm-Instanzen verhindert.

Der objektorientierte innere Aufbau (s. Abb. 1) ist nach außen hin bewußt als eine Funktionsschnittstelle herausgeführt. Dies hat den Vorteil der Sprachunabhängigkeit.

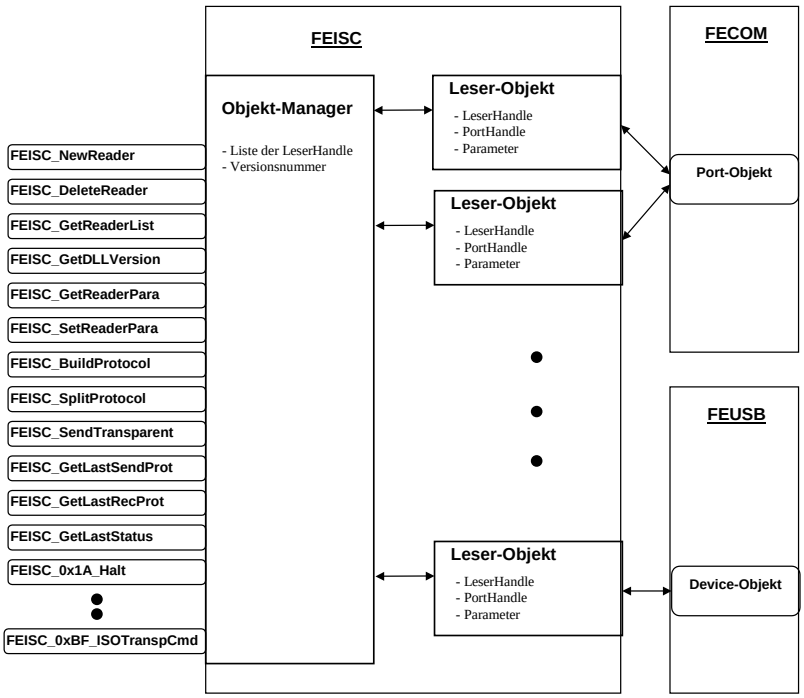


Abbildung 1: Interner Aufbau von FEISC

Die Abbildung 1 verdeutlicht, dass mehrere Leser-Objekte sich eine gemeinsame serielle Schnittstelle in FECOM bzw. einen gemeinsamen Kanal in FEUSB teilen können. Solange der Zugriff auf das Port-Objekt innerhalb eines Arbeits-Threads sequentiell geschieht, werden keine Konflikte auftreten.

Fast jede Bibliotheks-Funktion hat einen Rückgabewert, der im Fehlerfall immer negativ ist.

5.2. 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.

Auf der Ebene der Protokollschicht (FEISC) ist Parallelität über separate Leser-Objekte möglich, wenn jedes Leser-Objekt mit einer eigenen Kommunikations-Schnittstelle verbunden ist. Eine Ausnahme gilt für die vier speziellen Funktionen FEISC_BuildxxProtocol, FEISC_SplitxxProtocol, die einen globalen Puffer für Protokolldaten nutzen.

5.3. Parameterübergabe

Einige Funktionen unterstützen die Parameterübergabe sowohl als nullterminierte Zeichenkette als auch als Array von Hex-Zahlen. Für beide Datenformate ist die Übergabe als Datentyp UCHAR★ möglich. Die Interpretation des Übergabewertes wird mit dem Funktionsparameter *iDataFormat* angegeben.

iDataFormat	Parameterübergabe	wird interpretiert als Zeiger auf
0	0x23, 0x56, 0xFA, 0xA6 (Intern entspricht 0x23 dem Zeichen "#"; 0x56 dem Zeichen "V"; usw.)	Array von UCHAR
1	"2356FAA6" (je zwei Zeichen werden als ein Hex-Wert interpretiert: Beispiel: "23" -> 0x23)	nullterminierte Zeichenkette

Alle anderen Parameter, die als UCHAR zu übergeben sind, sind als Hex-Wert (z. B. 0x23) anzugeben. Eine Übergabe per Zeichenkette ist nicht möglich!

Hinweis: UCHAR wird als Abkürzung (#define) für "unsigned char" verwendet.

5.4. Asynchrone Tasks zur Entlastung von Applikationen

Eine immer wiederkehrende Aufgabe von Applikationen ist die Inventarisierung von Transpondern im Antennenfeld des Lesers oder der Empfang von Notifications. Idealerweise sollten diese Aktionen im Hintergrund ablaufen und die Applikation dann informieren, wenn Transponder im Feld sind bzw. die Notification eingetroffen ist. Auch Kommunikationsvorgänge mit langer Antwortzeit sollten asynchron ablaufen.

Exakt diese Funktionalität kann u.a. mit der Funktion **FEISC_StartAsyncTask** realisiert werden. Intern wird dazu ein Thread gestartet, der das komplexe Handling mit dem Leser übernimmt und die Antwortdaten per Callback-Funktion an die Applikation liefert.

Asynchrone Tasks sind für mehrere Anwendungsfälle definiert, u.a. für Inventory im Host-Mode oder für den Empfang von Buffered-Read-Mode Daten im Notification Mode.

Asynchrone Tasks kann man für mehrere Leser gleichzeitig spezifizieren, sofern sie mit der Funktion **FEISC_NewReader** ein eigenes Objekt in der DLL erstellt bekamen. Problematisch sind Leser am RS485-Bus. In einem solchen Fall kann man immer nur einen Leser gleichzeitig ‚beobachten', weil diese an derselben Schnittstelle angeschlossen sind.

In der nachfolgenden Tabelle sind die Besonderheiten der Tasks dargelegt:

Task	TaskID	Anmerkungen
Einmaliger Inventory	FEISC_TASKID_FIRST_NEW_TAG	Ein Task kann nur gestartet werden, wenn folgende Option in der Firmware des Lesers intergriert ist: Das Leserprotokoll [0xB0][0x01] Inventory muß in seinem Mode-Byte ein optionales NOTIFY-Flag unterstützen. Nach dem Empfang des Leserprotokolls innerhalb der vorgegebenen Zeit, beendet sich der Task selbständig. Kommt es zu einer Zeitüberschreitung, wird die Callback-Funktion aufgerufen und der Status 0x01 (kein Transponder im Lesefeld) übermittelt und der Task beendet. Im Fehlerfall wird der Task immer sofort beendet und die Callback-Funktion übergibt den Fehlercode. Unterstützt werden die drei Schnittstellen Seriell, USB und TCP/IP, wobei die Schnittstellen vor dem Starten des Tasks geöffnet sein müssen. Der selbständige Verbindungsaufbau per TCP/IP vom Leser oder einem geeigneten Konverter zur Übermittlung der Daten ist nicht möglich. Callback-Funktion in FEISC_TASK_INIT: cbFct1 Die Daten sind über den Zeiger *ucRspData* verfügbar und entsprechen in der Struktur der Antwort des Protokolls [0xB0] [0x01] ISO Command Inventory, die im Systemhandbuch zum Leser dokumentiert ist.

Task	TaskID	Anmerkungen
Repetierender Inventory	FEISC_TASKID_EVERY_NEW_TAG	Es gelten die Bedingungen des einmaligen Inventory mit folgemdem Unterschied: Der repetierende Inventory definiert eine zyklische Aufgabe, die nur durch **FEISC_CancelAsyncTask** beendet werden kann. Ein Zyklus entspricht einem einmaligen Inventory und endet in einer Warteschleife, bis der nächste Zyklus von der Applikation durch den Aufruf von **FEISC_TriggerAsyncTask** erneut angestoßen wird. Durch die Applikations-seitige Triggerung wird sichergestellt, dass eine Applikation Zeit für die Entgegennahme und Bearbeitung der Inventarisierungsdaten erhält. Callback-Funktion in FEISC_TASK_INIT: cbFct1 Die Daten sind über den Zeiger *ucRspData* verfügbar und entsprechen in der Struktur der Antwort des Protokolls [0xB0] [0x01] ISO Command Inventory, die im Systemhandbuch zum Leser dokumentiert ist.
Empfang von Notifications	FEISC_TASKID_NOTIFICATION	Ein Task sollte nur gestartet werden, wenn der Notification-Mode in der Firmware des Lesers intergriert und aktiviert ist. Unterstützt wird nur die Kommunikation über TCP/IP. Mögliche Verbindungsoptionen sind (s. Systemhandbuch zum Leser): - Temporärer Verbindungsaufbau durch den Leser für die Dauer der Datenübertragung - Dauerhafter Verbindungsaufbau durch den Leser (in Planung) - Dauerhafter Verbindungsaufbau durch den Host (in Planung) Der Task definiert eine endlose Aufgabe, die nur durch **FEISC_CancelAsyncTask** beendet werden kann, bzw. im Fehlerfall während der Initialisierungsphase, nach dem Aufruf der Callback-Funktion, sofort beendet wird. Der Task wartet auf den Empfang der Buffered-Read-Mode Daten und ruft anschließend die Callback-Funktion auf. Nach der Rückkehr der Callback-Funktion können sofort wieder Daten vom Leser entgegengenommen werden. Bei Übertragungsfehlern wird die Callback-Funktion mit dem Fehlercode aufgerufen und anschließend die Empfangsprozedur fortgesetzt. Wenn die Keep-Alive Option aktiviert ist (empfohlen), dann wird eine Unterbrechung der Netzwerkverbindung erkannt, der empfangende Socket geschlossen und anschließend neu initialisert. Dadurch ist sichergestellt, dass der RFID-Leser nach der Wiederherstellung der Verbindung erneut eine Verbindung aufbauen kann. Hinweis: je nach Einstellung des Lesers können in kürzesten Abständen sehr viele Daten vom Leser verschickt werden. Ohne Handshake-Mechanismen (s. Systemhandbuch zum Leser) können u.U. Daten verloren gehen, wenn der Host für die Quantität der Notifications nicht geeignet ist. Callback-Funktion in FEISC_TASK_INIT: cbFct1 und cbFct2 Die Daten sind über den Zeiger *ucRspData* verfügbar und entsprechen in der Struktur der Antwort des Protokolls [0x21] Read Buffer bzw. [0x22] Read Buffer, die im Systemhandbuch zum Leser dokumentiert ist.

Task	TaskID	Anmerkungen
SAM Kommunikation	FEISC_TASKID_SAM_COMMAND	Ein einmaliger Task zur Kommunikation mit einem SAM (Security Application Module) im OBID® *classic-pro* Leser mit SAM-Sockel wird mit der Funktion **FEISC_0xC0_SAMCmd** gestartet. Nach dem Empfang des Leserprotokolls innerhalb der vorgegebenen Zeit, beendet sich der Task selbständig. Kommt es zu einer Zeitüberschreitung, wird die Callback-Funktion aufgerufen und der Fehlercode -4082 (FEISC_ERR_TASK_TIMEOUT) übermittelt und der Task beendet. Im Fehlerfall wird der Task immer sofort beendet und die Callback-Funktion übergibt den Fehlercode. Unterstützt werden die Schnittstellen Seriell und USB, wobei die Schnittstellen vor dem Starten des Tasks geöffnet sein müssen. Callback-Funktion in FEISC_TASK_INIT: cbFct1 Die Daten sind über den Zeiger *ucRspData* verfügbar und entsprechen in der Struktur der Antwort des Protokolls [0xC0] SAM Commands, die im Systemhandbuch zum Leser dokumentiert ist.
Command Queue	FEISC_TASKID_COMMAND_QUEUE	Ein einmaliger Task zur Ausführung eines [0xBC] Command Queue im OBID® *classic-pro* Leser wird mit der Funktion **FEISC_0xBC_CmdQueue** gestartet. Nach dem Empfang des Leserprotokolls innerhalb der vorgegebenen Zeit, beendet sich der Task selbständig. Kommt es zu einer Zeitüberschreitung, wird die Callback-Funktion aufgerufen und der Fehlercode -4082 (FEISC_ERR_TASK_TIMEOUT) übermittelt und der Task beendet. Im Fehlerfall wird der Task immer sofort beendet und die Callback-Funktion übergibt den Fehlercode. Unterstützt werden die Schnittstellen Seriell und USB, wobei die Schnittstellen vor dem Starten des Tasks geöffnet sein müssen. Callback-Funktion in FEISC_TASK_INIT: cbFct1 Die Daten sind über den Zeiger *ucRspData* verfügbar und entsprechen in der Struktur der Antwort des Protokolls [0xBC] Command Queue, die im Systemhandbuch zum Leser dokumentiert ist.

Task	TaskID	Anmerkungen
MAX Event	FEISC_TASKID_MAX_EVENT	Ein Task sollte nur gestartet werden, wenn der Access-Mode in der Firmware des Lesers intergriert und aktiviert ist. Unterstützt wird nur die Kommunikation über TCP/IP bei temporärem Verbindungsaufbau durch den Leser für die Dauer der Datenübertragung Der Task definiert eine endlose Aufgabe, die nur durch **FEISC_CancelAsyncTask** beendet werden kann, bzw. im Fehlerfall während der Initialisierungsphase, nach dem Aufruf der Callback-Funktion, sofort beendet wird. Der Task wartet auf den Empfang der Eventdaten und ruft anschließend die Callback-Funktion auf. Nach der Rückkehr der Callback-Funktion können sofort wieder Daten vom Leser entgegengenommen werden. Bei Übertragungsfehlern wird die Callback-Funktion mit dem Fehlercode aufgerufen und anschließend die Empfangsprozedur fortgesetzt. Wenn die Keep-Alive Option aktiviert ist (empfohlen), dann wird eine Unterbrechung der Netzwerkverbindung erkannt, der empfangende Socket geschlossen und anschließend neu initialisert. Dadurch ist sichergestellt, dass der RFID-Leser nach der Wiederherstellung der Verbindung erneut eine Verbindung aufbauen kann. Callback-Funktion in FEISC_TASK_INIT: cbFct3 Die Daten sind über den Zeiger *ucRspData* verfügbar und entsprechen in der Struktur der Antwort des Protokolls [0x1F] [0x05] Read Table für TableID = 0x05 (EventTable), die im Systemhandbuch zum Leser dokumentiert ist.
People Counter Event	FEISC_TASKID_PEOPLE_COUNTER	Ein Task sollte nur gestartet werden, wenn der Notification-Mode in der Firmware des Lesers intergriert und aktiviert ist und mindestens eine externe Funktionseinheit vom Typ ID ISC.ANTGPC (People Counter) angeschlossen ist. Der Task ist mit dem des Notification identisch. Deshalb gilt die dort beschriebene Spezifikation. Ein People Counter Event benötigt keinen Handshake-Mechanismus. Callback-Funktion in FEISC_TASK_INIT: cbFct1 und cbFct2 Die Daten sind über den Zeiger *ucRspData* verfügbar und entsprechen in der Struktur der Antwort des Protokolls [0x77] Get Counter, die im Systemhandbuch zum GatePeopleCounter dokumentiert ist.

Das interne Task-Verhalten wird wesentlich durch die Struktur **FEISC_TASK_INIT** bestimmt, die mit **FEISC_StartAsyncTask** übergeben wird. Sie enthält u.a. die für die Callback-Funktion notwendigen Parameter.

```
typedef struct _FEISC_TASK_INIT
{
    void*           pAny;           // pointer to anything, which is reflected as the first parameter
                                    // in the callback function (e.g. can be used to pass the object pointer)
    unsigned char   ucBusAdr;       // busaddress for serial communication
    unsigned int    uiChannelType;  // defines the channel type to be used
    int             iConnectByHost; // if 0: TCP/IP connection is initiated by reader. otherwise by host
    char            cIPAdr[16];     // server ip address
                                    // note: only for channel type FEISC_TASK_CHANNEL_TYPE_NEW_TCP
    int             iPortAdr;       // server or host port address
                                    // note: only for channel type FEISC_TASK_CHANNEL_TYPE_NEW_TCP
    UINT            uiTimeout;      // timeout for asynchronous task in steps of 100ms or
                                    // timeout for notification task in steps of 1s
    UINT            uiFlag;         // specifies the use of the union (e.g. FEISC_TASKCB_1)

    // only for authentication in notification mode
    bool            bCryptoMode;    // security mode on/off
    unsigned int    uiAuthentKeyLength; // authent key length
    unsigned char   ucAuthentKey[32];   // authent key

    // only for notification or max event mode
    bool            bKeepAlive;             // if true, keep alive option will be enabled (recommended)
    unsigned int    uiKeepAliveIdleTime;    // wait time in ms for first probe after connection is dropped down
                                            // for Linux: time is rounded up to seconds
    unsigned int    uiKeepAliveProbeCount;  // only for Linux: number of probes
                                            // for Windows Server 2003, and XP it is fixed to 5 by Microsoft
                                            // for Windows Vista and later it is fixed to 10 by Microsoft
    unsigned int    uiKeepAliveIntervalTime; // wait time in ms between probes
                                             // for Linux: time is rounded up to seconds

    union
    {
        // for notification and inventory task, SAM and Queue Command response, People Counter event
        void    (*cbFct1)(  void* pAny,             // [in] pointer to anything (from struct _FEISC_TASK_INIT)
                            int iReaderHnd,         // [in] reader handle of FEISC
                            int iTaskID,            // [in] task identifier from FEISC_StartAsyncTask(..)
                            int iError,             // [in] OK (=0), error code (<0) or status byte from reader (>0)
                            unsigned char ucCmd,    // [in] reader command
                            unsigned char* ucRspData, // [in] response data
                            int iRspLen);           // [in] length of response data

        // only for notification task and People Counter event
        void    (*cbFct2)(  void* pAny,             // [in] pointer to anything (from struct _FEISC_TASK_INIT)
                            int iReaderHnd,         // [in] reader handle of FEISC
                            int iTaskID,            // [in] task identifier from FEISC_StartAsyncTask(..)
                            int iError,             // [in] OK (=0), error code (<0) or status byte from reader (>0)
                            unsigned char ucCmd,    // [in] reader command
                            unsigned char* ucRspData, // [in] response data
                            int iRspLen,            // [in] length of response data
                            char* cIPAdr,           // [in] ip address of the reader
                            int iPortNr);           // [in] local port number which received the notification
```

```
        // only for MAX notification task
        void    (*cbFct3)(  void* pAny,                 // [in] pointer to anything (from struct _FEISC_TASK_INIT)
                            int iReaderHnd,             // [in] reader handle of FEISC
                            int iTaskID,                // [in] task identifier from FEISC_StartAsyncTask(..)
                            int iError,                 // [in] OK (=0), error code (<0) or status byte from reader (>0)
                            unsigned char ucCmd,        // [in] reader command
                            unsigned char* ucRspData,   // [in] response data
                            int iRspLen,                // [in] length of response data
                            char* cIPAdr,               // [in] ip address of the reader
                            int iPortNr,                // [in] local port number which received the notification
                            unsigned char& ucAction);   // [out] action set by host application
    }Method[5];

    union
    {
        int iNotifyWithAck;         // 0: notification without acknowledge
                                    // 1: notification with acknowledge
    }InData[4]

} FEISC_TASK_INIT;
```

Kernelement der Struktur ist die *union* (Method), die einen oder mehrere Funktionszeiger enthält. Die Auswahl der Callback-Funktion wird mit dem Parameter *uiFlag* vorgenommen. Der Parameter *pAny* kann für beliebige Daten verwendet werden und wird im ersten Parameter der Callback-Funktion zurückgegeben. C++ Programmierer können damit einen Zeiger des aufrufenden Objektes in die statisch deklarierte Callback-Funktion übertragen bekommen und so auf Klassenfunktionen zugreifen. *uiTimeout* definiert die Zeitüberschreitung für einen Inventory-Zyklus bzw. die maximale Zeit zum Empfang eines Notification-Protokolls. Die Wertigkeit ist abhängig von den Vorgaben im Systemhandbuch des Lesers zum Protokoll [0xB0][0x01] Inventory bzw. für Notification-Tasks in Sekunden.

Die Strukturvariablen *cIPAdr* und *iPortAdr* sind ausschließlich für den Notification-Task vorgesehen. Bei Verwendung des TCP/IP-Kanals für den Inventory-Task muß der Socket vor dem Start des asynchronen Tasks bereits geöffnet sein.

Wichtiger Hinweis: vor der Verwendung der Struktur FEISC_TASK_INIT muss diese mit 0 initialisiert werden: z.B. mit memset(myTaskInit, 0, sizeof(FEISC_TASK_INIT));

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

5.5. Ereignissignalisierung an Applikationen[6]

Für einige Ereignisse können Ereignisbehandlungsmaßnahmen installiert werden. Sobald z. B. ein Sendeprotokoll über eine Schnittstelle ausgegeben wird, kann man zusätzlich, asynchron zum Programmablauf, der Applikation dieses Ereignis mitteilen. In der Applikation muß dafür eine entsprechende Funktion bereitstehen (s. 5.7.10. FEISC_AddEventHandler). Ereignisbehandlungen dürfen nicht mit der Bearbeitung von Ereignissen, ausgelöst durch das Starten eines asynchronen Tasks, verwechselt werden.

Eine Ereignisbehandlungsmaßnahme muß mit der Funktion **FEISC_AddEventHandler** installiert werden. Man kann zwischen fünf verschiedenen Signalisierungsmethoden wählen: Nachricht an aufrufenden Prozeß, Nachricht an ein Fenster, Verwendung einer (von zwei) Callback-Funktion oder Signalisierung mit einem Windows-API-Event.

Eine installierte Ereignisbehandlungsmaßnahme muß mit der Funktion **FEISC_DelEventHandler** wieder entfernt werden.

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

```
typedef struct _FEISC_EVENT_INIT
{
    void* pAny;       // Zeiger auf beliebiges Element, das im ersten Parameter der 4. Callback-Funktion übertragen
                      // wird. Hier kann man z.B. den this-Zeiger an eine statische Klassenmethode übergeben.
    UINT uiUse;       // Definiert den Event (z.B. FEISC_PRT_EVENT)
    UINT uiMsg;       // Message-Code für dwThreadID und hwndWnd (z.B. WM_USER_xyz)
    UINT uiFlag;      // Spezifiziert die Verwendung der union (z.B. FEISC_WND_HWND)
    union
    {
        DWORD     dwThreadID;                  // für Thread-ID
        HWND      hwndWnd;                     // für Window-Handle
        void      (*cbFct)(int, int);          // für 1. Callback-Funktion
        void      (*cbFct2)(BSTR, int, int);   // für 2. Callback-Funktion
        void      (*cbFct4)(void*, const char*, int);  // für 4. Callback-Funktion (3. Callback nicht öffentlich)
        HANDLE    hEvent;                      // für Event-Handle
    }Method[7];

} FEISC_EVENT_INIT;
```

Kernelement der Struktur ist die *union*, die entweder die ID eines Prozesses, das Handle eines Fensters, einen Funktionszeiger oder das Handle eines Windows-API-Events enthält. Die Auswahl der Signalisierungsform wird mit dem Parameter *uiFlag* vorgenommen. Im Parameter *uiUse* hinterlegt man eine Kennung für das Ereignis, 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*, *cbFct*, *cbFct2*, *cbFct4* oder *hEvent* kann nur einmal pro Ereignis verwendet werden.

[6] Für Linux C/C++ Projekte nur eingeschränkt nutzbar

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

5.6. Sicherheit in der Datenübertragung

5.6.1. Übersicht

Optional können OBID i-*scan*®- oder OBID® *classic-pro* Leser die Protokolle über Ethernet (TCP/IP) verschlüsselt übertragen. Zur Anwendung kommt ein AES-Algorithmus mit einer Schlüssellänge von 256 Bit. Der Authentifizierungsschlüssel (Passwort) ist im Leser gespeichert und kann nicht ausgelesen werden. Der Kryptomode im Leser ist ab Werk abgeschaltet.

Die Datenverschlüsselung basiert auf Funktionen der Open-Source Organisation openSSL (http://www.openssl.org), die in der Bibliotheksdatei libeay32.dll (Windows) bzw. libcrypto.so (Linux) enthalten sind. Die Bindung an die openSSL-Bibliothek erfolgt erst zur Laufzeit beim ersten Aufruf einer openSSL-Funktion. Dies bededeutet, dass alle Applikationen, die keine Datenverschlüsselung nutzen, die genannte openSSL-Bibliothek nicht installieren müssen. Für den Fall, dass verschlüsselte Datenübertragung zur Anwendung kommt, müssen Sie die Lizenzbedingungen zu openSSL beachten.

Die Datenverschlüsselung wird durch das Aktivieren des Kryptomodes in der Leserkonfiguration mit einem anschließenden CPU-Reset eingeschaltet. Danach akzeptiert ein Leser im Kryptomode ausschließlich verschlüsselte Protokolle. Vor dem ersten Leserbefehl muss mit einem Authent-Befehl (FEISC_0xAE_ReaderAuthent), mit dem das Passwort (Werkseinstellung: Passwort besteht aus Nullen) verschlüsselt übertragen wird, eine Session gestartet werden. Jedes nachfolgende Protokoll wird dann automatisch verschlüsselt übertragen.

Hinweis: Nach der ersten Authentifizierung sollte ein neues Passwort vergeben und eine erneute Authentifizierung mit dem neuen Passwort durchgeführt werden. Diese Vorgehensweise – erst in den Krytomode wechseln und dann ein Passwort vergeben – stellt sicher, dass das neue Passwort verschlüsselt übertragen wird! Andernfalls wird das neue Passwort im Klartext übertragen.

5.6.2. Rückmeldung von Fehlerzuständen

Ein Leser im Kryptomode lehnt alle nicht verschlüsselten Protokolle mit dem Status 0x19 (Crypto Processing Error) ab.

Ein Leser im Klartextmode lehnt alle verschlüsselten Protokolle mit dem Status 0x82 (Command not available) ab.

Ein Reader-Authent mit einem falschen Passwort wird mit dem Status 0x12 (Authent Error) signalisiert.

Ein Leser im Kryptomode signalisiert mit dem Status 0x19 (Crypto Processing Error) einen fehlerhaften Zustand in der Datenverschlüsselung. Der Host muss sich daraufhin erneut authentifizieren.

Die Fehlercodes -4093 und -4094 aus FEISC_0x..-Funktionen signalisieren einen Host-seitigen fehlerhaften Zustand in der Datenverschlüsselung. Der Host muss sich daraufhin erneut authentifizieren.

Der Fehlercode -4090 signalisiert einen Fehler beim Laden der openSSL-Bibliotheksdatei. Möglicherweise ist diese Bibliotheksdatei nicht installiert oder eine nicht kompatible Version ist installiert.

5.6.3. Hinweise für den Programmierer

Der Programmierer, der die Datenverschlüsselung in sein Projekt – auch nachträglich - integriert, muss nur wenige Aspekte beachten:

1. Alle Kommunikationsfunktionen FEISC_0x… sind sowohl für die verschlüsselte als auch unverschlüsselte Datenübertragung geeignet.
2. Es muss sichergestellt sein, dass jeder OBID i-*scan*®- oder OBID® *classic-pro* Leser über ein eigenes Leser-Objekt programmiert wird, denn dieses verwaltet die individuell für jeden Leser kalkulierten Sessiondaten.
3. Nach einem Verbindungsaufbau mit FETCP_Connect muss ein Reader-Authent erfolgen.
4. Erhält der Host nach der Übertragung eines verschlüsselten oder unverschlüsselten Protokolls den Status 0x19 muss er einen Reader-Authent ausführen.
5. Erhält die Applikation die Fehlercodes -4093 oder -4094 muss ein Reader-Authent ausgeführt werden.
6. Die Datenübertragung im Notification- bzw. Access-Mode erfolgt bei aktiviertem Kryptomode verschlüsselt. Deshalb muss das Passwort in der Struktur FEISC_TASK_INIT hinterlegt werden.
7. Wird der Kryptomode in der Leserkonfiguration abgeschaltet, wechselt das Leser-Objekt mit dem nächsten unverschlüsselten Protokoll selbständig wieder in den Mode der unverschlüsselten Datenübertragung. Das bestehende Leser-Objekt kann also weiter benutzt werden. Ebenso ist ein Verbindungsabbau und erneuter Verbindungsaufbau nicht notwendig.

5.7. Liste der Funktionen

In dem Support-Paket sind sehr viele Funktionen für unterschiedliche Aufgabestellungen enthalten. Zur besseren Orientierung sind sie in Gruppen aufgeteilt.

Verwaltungs-Funktionen für Leser-Objekte

- **int FEISC_NewReader(** int iPortHnd **)**
- **int FEISC_DeleteReader(** int iReaderHnd **)**
- **int FEISC_GetReaderList(** int iNext **)**
- **int FEISC_GetReaderPara(** int iReaderHnd, char* cPara, char* cValue **)**
- **int FEISC_SetReaderPara(** int iReaderHnd, char* cPara, char* cValue **)**
- **void FEISC_GetDLLVersion(** char* cVersion **)**
- **int FEISC_GetErrorText(** int iErrorCode, char* cErrorText **)**
- **int FEISC_GetStatusText(** UCHAR ucStatus, char* cStatusText **)**
- **int FEISC_AddEventHandler(** int iReaderHnd, FEISC_EVENT_INIT* pInit **)**
- **int FEISC_DelEventHandler(** int iReaderHnd, FEISC_EVENT_INIT* pInit **)**

Funktionen für Plug-in Objekte zur Anbindung alternativer Schnittstellen

- **int FEISC_PI_Get(** const char* cLibName, void** pPlugIn **)**
- **int FEISC_PI_Install(** int iReaderHnd, void* pPlugIn **)**
- **int FEISC_PI_Remove(** int iReaderHnd **)**
- **int FEISC_PI_OpenPort(** int iReaderHnd, char* cPortDefinition **)**
- **int FEISC_PI_ClosePort(** int iReaderHnd **)**
- **int FEISC_PI_GetPortPara(** int iReaderHnd, char* cPara, char* cValue**)**
- **int FEISC_PI_SetPortPara(** int iReaderHnd, char* cPara, char* cValue**)**
- **int FEISC_PI_GetDLLVersion(** int iReaderHnd, char* cVersion **)**
- **int FEISC_PI_GetErrorText(** int iReaderHnd, int iErrorCode, char* cErrorText **)**

Protokoll-Funktionen

- **int FEISC_BuildSendProtocol(** int iReaderHnd, UCHAR cBusAdr, UCHAR cCmdByte, UCHAR* cSendData, int iDataLen, UCHAR* cSendProt, int iDataFormat **)**
- **int FEISC_BuildRecProtocol(** int iReaderHnd, UCHAR cBusAdr, UCHAR cCmdByte, UCHAR cStatus, UCHAR* cRecData, int iDataLen, UCHAR* cRecProt, int iDataFormat **)**
- **int FEISC_SplitSendProtocol(** int iReaderHnd, UCHAR* cSendProt, int iSendLen, UCHAR* cBusAdr, UCHAR* cCmdByte, UCHAR* cSendData, int* iDataLen, int iDataFormat **)**
- **int FEISC_SplitRecProtocol(** int iReaderHnd, UCHAR* cRecProt, int iRecLen, UCHAR* cBusAdr, UCHAR* cCmdByte, UCHAR* cRecData, int* iDataLen, int iDataFormat **)**

Abfrage-Funktionen

- **int FEISC_GetLastSendProt(** int iReaderHnd, UCHAR* cSendProt, int iDataFormat **)**
- **int FEISC_GetLastRecProt(** int iReaderHnd, UCHAR* cRecProt, int iDataFormat **)**
- **int FEISC_GetLastState(** int iReaderHnd, char* cStatusText **)**
- **int FEISC_GetLastRecProtLen(** int iReaderHnd **)**
- **int FEISC_GetLastError(** int iReaderHnd , int* iErrorCode, char* cErrorText **)**

Allgemeine Kommunikations-Funktionen

- **int FEISC_SendTransparent(** int iReaderHnd, UCHAR* cSendProt, int iSendLen, UCHAR* cRecProt, int iRecLen, int iCheckSum, int iDataFormat **)**
- **int FEISC_Transmit(** int iReaderHnd, UCHAR* cSendProt, int iSendLen, int iCheckSum, int iDataFormat **)**
- **int FEISC_Receive(** int iReaderHnd, UCHAR* cRecProt, int iRecLen, int iCheckSum, iDataFormat **)**

Spezielle Kommunikations-Funktionen

- **int FEISC_0x18_Destroy(** int iReaderHnd, UCHAR cBusAdr, UCHAR cMode, UCHAR* cEPC, UCHAR* cPW **)**
- **int FEISC_0x1A_Halt(** int iReaderHnd, UCHAR cBusAdr **)**
- **int FEISC_0x1B_ResetQuietBit(** int iReaderHnd, UCHAR cBusAdr **)**
- **int FEISC_0x1C_EASRequest(** int iReaderHnd, UCHAR cBusAdr **)**
- **int FEISC_0x1E_TableDataExchange**(int iReaderHnd, UCHAR cBusAdr, UCHAR cSubCmd, UCHAR cMode, UCHAR cDevice, UCHAR cBank, UCHAR cTableID, UCHAR* cReqData, int iReqLen, UCHAR* cRspData, int* iRspLen **)**
- **int FEISC_0x1F_MAXDataExchange**(int iReaderHnd, UCHAR cBusAdr, UCHAR cSubCmd, UCHAR cMode, UCHAR cTableID, UCHAR* cReqData, int iReqLen, UCHAR* cRspData, int* iRspLen, int iDataFormat **)**
- **int FEISC_0x21_ReadBuffer(** int iReaderHnd, UCHAR cBusAdr, UCHAR cSets, UCHAR* cTrData, UCHAR* cRecSets, UCHAR* cRecDataSets, int iDataFormat **)**
- **int FEISC_0x22_ReadBuffer(** int iReaderHnd, UCHAR cBusAdr, int iSets, UCHAR* cTrData, UCHAR* cRecSets, int* iRecDataSets, int iDataFormat **)**
- **int FEISC_0x31_ReadDataBufferInfo(** int iReaderHnd, UCHAR cBusAdr, UCHAR* cTabSize, UCHAR* cTabStart, UCHAR* cTabLen, int iDataFormat **)**
- **int FEISC_0x32_ClearDataBuffer(** int iReaderHnd, UCHAR cBusAdr **)**
- **int FEISC_0x33_InitBuffer(** int iReaderHnd, UCHAR cBusAdr **)**
- **int FEISC_0x34_ForceNotifyTrigger(** int iReaderHnd, UCHAR cBusAdr, UCHAR cMode **)**
- **int FEISC_0x52_GetBaud(** int iReaderHnd, UCHAR cBusAdr **)**
- **int FEISC_0x55_StartFlashLoader(** int iReaderHnd **)**
- **int FEISC_0x55_StartFlashLoaderEx(** int iReaderHnd, UCHAR cBusAdr **)**
- **int FEISC_0x63_CPUReset(** int iReaderHnd, UCHAR cBusAdr **)**
- **int FEISC_0x64_SystemReset(** int iReaderHnd, UCHAR cBusAdr, UCHAR cMode **)**
- **int FEISC_0x65_SoftVersion(** int iReaderHnd, UCHAR cBusAdr, UCHAR* cVersion, int iDataFormat **)**
- **int FEISC_0x66_ReaderInfo(** int iReaderHnd, UCHAR cBusAdr, UCHAR cMode, UCHAR* cInfo, int iDataFormat **)**
- **int FEISC_0x69_RFReset(** int iReaderHnd, UCHAR cBusAdr **)**
- **int FEISC_0x6A_RFOnOff(** int iReaderHnd, UCHAR cBusAdr, UCHAR cRF **)**
- **int FEISC_0x6B_CentralizedRFSync(** int iReaderHnd, UCHAR cBusAdr, UCHAR cMode, UCHAR cTxChannel, int iTxPeriod, UCHAR cRes1, UCHAR cRes2 **)**
- **int FEISC_0x6C_SetNoiseLevel(** int iReaderHnd, UCHAR cBusAdr, UCHAR* cLevel, int iDataFormat **)**
- **int FEISC_0x6D_GetNoiseLevel(** int iReaderHnd, UCHAR cBusAdr, UCHAR* cLevel, int iDataFormat **)**
- **int FEISC_0x6E_RdDiag(** int iReaderHnd, UCHAR cBusAdr, UCHAR cMode, UCHAR* cData **)**
- **int FEISC_0x6F_AntennaTuning(** int iReaderHnd, UCHAR cBusAdr **)**
- **int FEISC_0x71_SetOutput(** int iReaderHnd, UCHAR cBusAdr, int iOS, int iOSF, int iOSTime, int iOutTime **)**
- **int FEISC_0x72_SetOutput(** int iReaderHnd, UCHAR cBusAdr, UCHAR cMode, UCHAR cOutN, UCHAR* pRecords**)**
- **int FEISC_0x74_ReadInput(** int iReaderHnd, UCHAR cBusAdr, UCHAR* cInput **)**
- **int FEISC_0x75_AdjAntenna(** int iReaderHnd, UCHAR cBusAdr, UCHAR* cLevel, int iDataFormat **)**

- **int FEISC_0x76_CheckAntennas(** int iReaderHnd, UCHAR cBusAdr, UCHAR* cAntOut, int* iAntOutLen **)**
- **int FEISC_0x80_ReadConfBlock(** int iReaderHnd, UCHAR cBusAdr, UCHAR cConfAdr, UCHAR* cConfBlock, int iDataFormat **)**
- **int FEISC_0x81_WriteConfBlock(** int iReaderHnd, UCHAR cBusAdr, UCHAR cConfAdr, UCHAR* cConfBlock, int iDataFormat **)**
- **int FEISC_0x82_SaveConfBlock(** int iReaderHnd, UCHAR cBusAdr, UCHAR cConfAdr **)**
- **int FEISC_0x83_ResetConfBlock(** int iReaderHnd, UCHAR cBusAdr, UCHAR cConfAdr **)**
- **int FEISC_0x85_SetSysTimer(** int iReaderHnd, UCHAR cBusAdr, UCHAR* cTime, int iDataFormat **)**
- **int FEISC_0x86_GetSysTimer(** int iReaderHnd, UCHAR cBusAdr, UCHAR* cTime, int iDataFormat **)**
- **int FEISC_0x87_SetSystemDate(** int iReaderHnd, UCHAR cBusAdr, UCHAR cCentury, UCHAR cYear, UCHAR cMonth, UCHAR cDay, UCHAR cTimezone, UCHAR cHour, UCHAR cMinute, int iMilliSecond **)**
- **int FEISC_0x88_GetSystemDate(** int iReaderHnd, UCHAR cBusAdr, UCHAR* cCentury, UCHAR* cYear, UCHAR* cMonth, UCHAR* cDay, UCHAR* cTimezone, UCHAR* cHour, UCHAR* cMinute, int* iMilliSecond **)**
- **int FEISC_0x8A_ReadConfiguration(** int iReaderHnd, UCHAR cBusAdr, UCHAR cDevice, UCHAR cBank, UCHAR cMode, int iReqBlockAdr, UCHAR cReqBlockCount, UCHAR* cRspBlockCount, UCHAR* cRspBlockSize, UCHAR* cRspData**)**
- **int FEISC_0x8B_WriteConfiguration(** int iReaderHnd, UCHAR cBusAdr, UCHAR cDevice, UCHAR cBank, UCHAR cMode, UCHAR cReqBlockCount, UCHAR cReqBlockSize, UCHAR* cReqData**)**
- **int FEISC_0x8C_ResetConfiguration(** int iReaderHnd, UCHAR cBusAdr, UCHAR cDevice, UCHAR cBank, UCHAR cMode, int iReqBlockAdr, UCHAR cReqBlockCount**)**
- **int FEISC_0x9F_Piggyback_Command(** int iReaderHnd, UCHAR cBusAdr, UCHAR cMode, UCHAR cDevice, UCHAR cPort, UCHAR* cReqPrt, int iReqLen, UCHAR* cRspPrt, int* iRspLen**)**
- **int FEISC_0xA0_RdLogin(** int iReaderHnd, UCHAR cBusAdr, UCHAR* cRd_PW, int iDataFormat **)**
- **int FEISC_0xA2_WriteMifareKeys(** int iReaderHnd, UCHAR cBusAdr, UCHAR cType, UCHAR cAdr, UCHAR* cKey, int iDataFormat **)**
- **int FEISC_0xA3_Write_DES_AES_Keys(** int iReaderHnd, UCHAR cBusAdr, UCHAR cMode, UCHAR cReaderKeyIndex, UCHAR cAuthentMode, UCHAR cKeyLen, UCHAR* cKey, int iDataFormat **)**
- **int FEISC_0xAD_WriteReaderAuthentKey(** int iReaderHnd, UCHAR cBusAdr, UCHAR cMode, UCHAR cKeyType, UCHAR cKeyLen, UCHAR* cKey, int iDataFormat **)**
- **int FEISC_0xAE_ReaderAuthent(** int iReaderHnd, UCHAR cBusAdr, UCHAR cMode, UCHAR cKeyType, UCHAR cKeyLen, UCHAR* cKey, int iDataFormat **)**
- **int FEISC_0xB0_ISOCmd**(int iReaderHnd, UCHAR cBusAdr, UCHAR* cReqData, int iReqLen, UCHAR* cRspData, int* iRspLen, int iDataFormat **)**
- **int FEISC_0xB1_ISOCustAndPropCmd**(int iReaderHnd, UCHAR cBusAdr, UCHAR cMfr, UCHAR* cReqData, int iReqLen, UCHAR* cRspData, int* iRspLen, int iDataFormat **)**
- **int FEISC_0xB2_ISOCmd**(int iReaderHnd, UCHAR cBusAdr, UCHAR* cReqData, int iReqLen, UCHAR* cRspData, int* iRspLen, int iDataFormat **)**
- **int FEISC_0xB3_EPCCmd**(int iReaderHnd, UCHAR cBusAdr, UCHAR* cReqData, int iReqLen, UCHAR* cRspData, int* iRspLen, int iDataFormat **)**
- **int FEISC_0xB3_EPCCmd**(int iReaderHnd, UCHAR cBusAdr, UCHAR* cReqData, int iReqLen, UCHAR* cRspData, int* iRspLen, int iDataFormat **)**
- **int FEISC_0xB4_EPC_UHF_Cmd**(int iReaderHnd, UCHAR cBusAdr, UCHAR cMfr, UCHAR* cReqData, int iReqLen, UCHAR* cRspData, int* iRspLen, int iDataFormat **)**
- **int FEISC_0xBB_C1G2_TranspCmd**(int iReaderHnd, UCHAR cBusAdr, int iMode, UCHAR ucTxPara, UCHAR ucRxPara, unsigned int uiTs, int iRspLength, UCHAR* cReqData, int iReqLen, UCHAR* cRspData, int* iRspLen **)**

- **int FEISC_0xBC_CmdQueue**(int iReaderHnd, int iMode, int iCmdCount, UCHAR* cCmdQueue, int iCmdQueueLen, FEISC_TASK_INIT* pInit **)**
- **int FEISC_0xBD_ISOTranspCmd**(int iReaderHnd, UCHAR cBusAdr, int iMode, int iRspLength, UCHAR* cReqData, int iReqLen, UCHAR* cRspData, int* iRspLen, int iDataFormat **)**
- **int FEISC_0xBE_ISOTranspCmd**(int iReaderHnd, UCHAR cBusAdr, int iMode, int iRspLength, UCHAR* cReqData, int iReqLen, UCHAR* cRspData, int* iRspLen, int iDataFormat **)**
- **int FEISC_0xBF_ISOTranspCmd**(int iReaderHnd, UCHAR cBusAdr, int iMode, int iRspLength, UCHAR* cReqData, int iReqLen, UCHAR* cRspData, int* iRspLen, int iDataFormat **)**
- **int FEISC_0xC0_SAMCmd**(int iReaderHnd, int iSlot, UCHAR* cReqData, int iReqLen, FEISC_TASK_INIT* pInit **)**
- **int FEISC_0xC0_SAMCmd_Sync**(int iReaderHnd, UCHAR cBusAdr, int iSlot, int iTimeout, UCHAR* cReqData, int iReqLen, UCHAR* cRspData, int* iRspLen **)**
- **int FEISC_0xC1_DESFireCmd**(int iReaderHnd, UCHAR cBusAdr, UCHAR cSubCmd, UCHAR cMode, UCHAR* cAppID, UCHAR cReaderKeyIndex, UCHAR* cReqData, int iReqLen, UCHAR* cRspData, int* iRspLen, int iDataFormat **)**
- **int FEISC_0xC2_MifarePlusCmd**(int iReaderHnd, UCHAR cBusAdr, UCHAR cSubCmd, UCHAR cMode, UCHAR* cReqData, int iReqLen, UCHAR* cRspData, int* iRspLen, int iDataFormat **)**
- **int FEISC_0xC3_DESFireCmd**(int iReaderHnd, UCHAR cBusAdr, UCHAR cSubCmd, UCHAR cMode, UCHAR* cReqData, int iReqLen, UCHAR* cRspData, int* iRspLen, int iDataFormat **)**

Spezielle Funktionen für asynchrone Tasks

- **int FEISC_StartAsyncTask(** int iReaderHnd, int iTaskID, FEISC_TASK_INIT* pInit, void* pInput **)**
- **int FEISC_CancelAsyncTask(** int iReaderHnd **)**
- **int FEISC_TriggerAsyncTask(** int iReaderHnd **)**

5.7.1. Welche Funktion für welchen OBID i-*scan*® und OBID® *classic-pro* Leser ?

Die Command-Matrizen für OBID i-*scan*® und OBID® *classic-pro* Leser finden sich in den jeweiligen Systemhandbüchern.

Generell gilt, dass die Bibliothek FEISC alle Commands mit allen Optionen von allen Lesern unterstützt.

5.7.2. FEISC_NewReader

Funktion	Legt ein Leser-Objekt an.
Syntax	**int FEISC_NewReader(int iPortHnd)**
Beschreibung	Es wird ein Leser-Objekt erzeugt. Nur mit einem Leser-Objekt können die Protokollfunktionen ausgeführt werden. *iPortHnd*[8] ist der Handle eines mit der Funktion **FECOM_OpenPort** aus FECOM erzeugten Port-Objekts oder mit der Funktion **FEUSB_OpenDevice** erzeugten Device-Objekts oder mit der Funktion **FETCP_Connect** erzeugten TCP/IP-Socket-Objekts. Dieser Handle erlaubt die direkte Weitergabe von Protokollen an FECOM, FETCP bzw. FEUSB. Die Übergabe einer 0 ist ebenfalls zulässig. Möchte man ein Leser-Objekt an einen eigenen Portreiber binden, muß man die Konstante FEISC_PLUGIN übergeben und zuvor mit der Funktion FEISC_InstallPlugIn den eigenen Porttreiber installiert haben. Prinzipiell können mehrere Leser-Objekte ihre Kommunikation über dieselbe serielle COM-Schnittstelle, denselben TCP/IP-Socket bzw. denselben USB-Kanal ausführen. Wenn Datenverschlüsselung realisiert werden soll, muss jeder Leser ein eigenes Leser-Objekt erhalten! *iPortHnd* nutzt zur Unterscheidung der Protokollausgabe an FECOM, FETCP oder FEUSB das erste Byte (MSB) des PortHandle: *iPortHnd* = 0x0XXXXXXX[9] führt zur Ausgabe an FECOM.DLL/SO *iPortHnd* = 0x1XXXXXXX führt zur Ausgabe an FEUSB.DLL/SO *iPortHnd* = 0x2XXXXXXX führt zur Ausgabe an FETCP.DLL/SO Den Wert des im Leser-Objekt gespeicherten PortHandle kann man nachträglich mit der Funktion **FEISC_SetReaderPara** verändern. Ein mit **FEISC_NewReader** erzeugtes Leser-Objekt muß (!) mit der Funktion **FEISC_DeleteReader** aus dem Speicher entfernt werden. Andernfalls wird der von der Bibliothek reservierte Speicher nicht wieder freigegeben.
Rückgabewert	Wenn ein Leser-Objekt fehlerfrei erstellt 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	#include "feisc.h" #include "fecom.h" ... char cPortNr[4]; sprintf(cPortNr, “%d“, 1); // Integer in Char wandeln int iPortHnd = FECOM_OpenPort(cPortNr); // COM:1 soll geöffnet werden if(iPortHnd < 0)

[8] iPortHnd wird in diesem Dokument durchgehend auch stellvertretend für iDevHnd bzw. iSocketHnd verwendet

[9] X kennzeichnet beliebigen Hex-Wert

```
{
    // hier Code für den Fehlerfall
}
else
{   // Leser-Objekt öffnen
    int iReaderHnd = FEISC_NewReader( iPortHnd );
}
```

5.7.3. FEISC_DeleteReader

Funktion	Löscht ein Leser-Objekt.
Syntax	**int FEISC_DeleteReader(int iReaderHnd)**
Beschreibung	Die Funktion löscht das durch den Parameter *iReaderHnd* angegebene Leser-Objekt und gibt den reservierten Speicher wieder frei.
Rückgabewert	Der Rückgabewert ist 0, wenn die Aktion erfolgreich war. Im Fehlerfall liefert die Funktion einen Wert kleiner als Null zurück. Die Liste der Fehlercodes findet sich im Anhang.
Beispiel	... #include "feisc.h" ... ... int iErr; int iReaderHnd = FEISC_NewReader(0); if(iReaderHnd < 0) { // hier Code für den Fehlerfall } ... ... if(iReaderHnd > 0) { iErr = FEISC_DeleteReader(iReaderHnd); ... } ... ...

5.7.4. FEISC_GetReaderList

Funktion	Ermittelt in Abhängigkeit vom Parameter *iNext* den ersten oder den nachfolgenden Leser-Handle aus der internen Liste der erzeugten Leser-Objekte.
Syntax	**int FEISC_GetReaderList(int iNext)**
Beschreibung	Die Funktion gibt ein Leser-Handle aus der internen Liste der Leser-Handle zurück. Übergibt man für *iNext* eine 0, wird der erste Eintrag aus der Liste zurückgegeben. Übergibt man mit *iNext* ein in der Liste geführten Leser-Handle, wird der dem Leser-Handle nachfolgende Eintrag ermittelt und zurückgegeben. Man kann auf diese Weise durch sukzessives Einsetzen des Rückgabewertes die Liste durchlaufen und alle Einträge abrufen.
Rückgabewert	Wenn ein Eintrag gefunden wurde, wird mit dem Rückgabewert der Leser-Handle geliefert. Ist das Ende der internen Liste erreicht, also der übergebene Leser-Handle keinen Nachfolger hat, wird eine 0 zurückgegeben. Ist kein Leser-Objekt angelegt, wird FEISC_ERR_EMPTY_LIST zurückgeliefert. Im Fehlerfall liefert die Funktion einen Wert kleiner als Null zurück. Die Liste der Fehlercodes findet sich im Anhang.
Beispiel	... #include "feisc.h" ... // Beispielfunktion zum Aufstellen einer Liste von Leser-Objekten void ReaderList(void) { int iNextHnd = FEISC_GetReaderList(0); // den ersten Handle ermitteln while(iNextHnd > 0) { // hier z. B. Code zum Sammeln der Handle und Auslesen von Parametern ... iNextHnd = FEISC_GetReaderList(iNextHnd); // nächsten Handle ermitteln } ... // hier z. B. Code zum Anzeigen einer Liste }
Tip	Beim Schließen aller erzeugten Leser-Objekte bedient man sich gerne einer Schleife, ähnlich der im oberen Beispiel. Nur muß man bedenken, dass man von einem gelöschten Leser-Objekt keinen Nachfolger mehr ermitteln kann. In dem folgenden Codefragment wird gezeigt, wie man in einer Schleife alle erzeugten Leser-Objekte löschen kann: ... int iNextHnd, iCloseHnd, iError; iNextHnd = FEISC_GetReaderList(0); // den ersten Handle ermitteln while(iNextHnd > 0) { iCloseHnd = iNextHnd; iNextHnd = FEISC_GetReaderList(iNextHnd); // erst nächsten Handle ermitteln iError = FEISC_DeleteReader(iCloseHnd); // jetzt erst Leser-Objekt entfernen }

5.7.5. FEISC_GetDLLVersion

Funktion	Ermittelt die Versionsnummer der DLL bzw. SO.
Syntax	**void FEISC_GetDLLVersion(char* cVersion)**
Beschreibung	Die Funktion gibt die Versionsnummer der DLL bzw. SO zurück. *cVersion* ist eine leere, nullterminierte Zeichenkette zur Rückgabe der Versionsnummer. Die Zeichenkette sollte wenigstens 256 Zeichen aufnehmen können. In der Zeichenkette wird aktuelle Versionsnummer zurückgegeben (z.B. "07.02.02"). Neuere Versionen könnten aber weitere Informationen liefern.
Rückgabewert	ohne
Beispiel	... #include "feisc.h" ... ... char cVersion[256]; FEISC_GetDLLVersion(cVersion); // hier Code zum Anzeigen der Versionsnummer ... ...

5.7.6. FEISC_GetErrorText

Funktion	Ermittelt Fehlertext zum Fehlercode
Syntax	**int FEISC_GetErrorText(int iErrorCode, char* cErrorText)**
Beschreibung	Die Funktion übergibt in *cErrorText* den zum *iErrorCode* zugehörigen Fehlertext. Der Puffer für *cErrorText* sollte 256 Zeichen aufnehmen können.
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 "feisc.h" ... ... char cErrorText[256]; ... int iBack = FEISC_GetErrorText(FEISC_ERR_PROTLEN, cErrorText) // hier Code zum Anzeigen des Textes ... ...

5.7.7. FEISC_GetStatusText

Funktion	Ermittelt Kurztext zum Statusbyte
Syntax	**int FEISC_GetStatusText(UCHAR ucStatus, char* cStatusText)**
Beschreibung	Die Funktion übergibt in *cStatusText* den zum *ucStatus* zugehörigen Kurztext. Der Puffer für *cStatusText* sollte 128 Zeichen aufnehmen können.
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 "feisc.h" ... ... char cStatusText[128]; ... int iBack = FEISC_GetStatusText(0x01, cStatusText) // hier Code zum Anzeigen des Textes ... ...

5.7.8. FEISC_GetReaderPara

Funktion	Ermittelt von einem Leser-Objekt einen Parameter.
Syntax	**int FEISC_GetReaderPara(int iReaderHnd, 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 Parameterwertes. Die Zeichenkette sollte wenigstens 128 Zeichen aufnehmen können. *iReaderHnd* ist der Handle zum Leser-Objekt.
Parameter-kennungen	Die Parameterkennungen sind: PortHnd[10], LogProt, LogFile, LogFilename, RecBusAdr, Language, ChkRecBusAdr, ConvHexToString, SendStr, RecStr, IsProtToAppLocked, und FrameSupport
Querverweis	Weitere Informationen in: 5.7.9. FEISC_SetReaderPara und 6.2. Liste der Parameterkennungen
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.
Beispiel	... #include "feisc.h" ... ... char cValue[128]; int iPortHnd; ... if(!FEISC_GetReaderPara(handle, "PortHnd", cValue)) { // Wandlung von Char in Integer sscanf(cValue, “%d”, &iPortHnd); // hier z. B. Code zur Verwendung des PortHandle ... } ... ... }

[10] Man beachte die Anmerkungen zum PortHandle in 5.6.2. FEISC_NewReader

5.7.9. FEISC_SetReaderPara

Funktion	Setzt einen Parameter eines Leser-Objekts auf neuen Wert.
Syntax	**int FEISC_SetReaderPara(int iReaderHnd, char* cPara, char* cValue)**
Beschreibung	Die Funktion übergibt an ein Leser-Objekt einen neuen Parameter. Das Leser-Objekt speichert den neuen Wert und macht ihn sofort zum aktuellen Parameter. *cPara* ist eine nullterminierte Zeichenkette mit der Parameterkennung. *cValue* ist eine nullterminierte Zeichenkette mit dem neuen Parameterwert. *iReaderHnd* ist der Handle zum Leser-Objekt.
Parameter-kennungen	Die Parameterkennungen sind: PortHnd[11], LogProt, LogFile, LogFilename, Language, ChkRecBusAdr, ConvHexToString, LockProtToApp, UnlockProtToApp und FrameSupport
Querverweis	Weitere Informationen in: 5.7.8. FEISC_GetReaderPara und 6.2. Liste der Parameterkennungen
Rückgabewert	Wenn das Leser-Objekt 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.
Beispiel	// das Beispiel zeigt, dass einem Leser-Objekt nachträglich ein neuer PortHandle zugewiesen // werden kann. Nach der Zuweisung läuft die Kommunikation über den neuen Port. ... #include "feisc.h" #include "fecom.h" ... ... int iErr; char cPortHnd[9]; char cPortNr[4]; sprintf(cPortNr, “%d“, 1); // Integer in Char wandeln ... int iPortHnd = FECOM_OpenPort(cPortNr); // COM:1 soll geöffnet werden if(iPortHnd > 0) { sscanf(cPortHnd, “%d“, & iPortHnd); // Integer in Char wandeln iErr = FEISC_SetReaderPara(iReaderHnd, "PortHnd", cPortHnd); // ab hier Kommunikation über neuen Port möglich ... } ... ...

[11] Man beachte die Anmerkungen zum PortHandle in 5.6.2. FEISC_NewReader

5.7.10. FEISC_AddEventHandler

<table>
<tr><td>Funktion</td><td>Eine Ereignisbehandlungsmaßnahme wird installiert</td></tr>
<tr><td>Syntax</td><td>int FEISC_AddEventHandler(int iReaderHnd, FEISC_EVENT_INIT* pInit)</td></tr>
<tr><td>Beschreibung</td><td>Die Funktion installiert eine von fünf 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 nur für das mit iReaderHnd identifizierte Leser-Objekt eingerichtet. Das bedeutet, dass man bei Bedarf für jedes Leser-Objekt diese Installation durchführen muß.
<table>
<tr><td>Ereignis</td><td>Beschreibung</td></tr>
<tr><td>FEISC_PRT_EVENT</td><td>je ein Ereignis für Sende- und Empfangsprotokoll[12]</td></tr>
<tr><td>FEISC_SNDPRT_EVENT</td><td>Ereignis für Sendeprotokoll[10]</td></tr>
<tr><td>FEISC_RECPRT_EVENT</td><td>Ereignis für Empfangsprotokoll[10]</td></tr>
<tr><td>FEISC_SCANNER_EVENT</td><td>Ereignis für empfangenes Scanner-Protokoll[13]
(nicht für Linux)</td></tr>
</table>
<u>1. Methode: Nachricht an Thread (nicht für Linux, Mac OS X)</u>
Diese Methode verwendet man für den Nachrichtenaustausch zwischen Threads[14]. Der Thread ermittelt mit der Windows-API-Funktion GetCurrentThreadID() den Thread-Identifier und übergibt diesen als Parameter dwThreadID in der FEISC_EVENT_INIT-Struktur.
Der Thread muß für den Empfang der Nachricht, die von FEISC mit der Windows-API-Funktion PostThreadMessage(..) verschickt wurde, eine Nachrichtenbehandlungs-funktion (MessageMap-Funktion) bereitstellen. Der Nachrichtencode ist frei wählbar.
Die FEISC_EVENT_INIT-Struktur wird wie folgt ausgefüllt:
uiUse = FEISC_xyz_EVENT // siehe Defines feisc.h
uiMsg = WM_USER + ... // frei wählbar, aber oberhalb von WM_USER[15]
uiFlag = FEISC_THREAD_ID
dwThreadID = GetCurrentThreadID()
Die MessageMap-Funktion in der Applikation bekommt im 1. Parameter (WPARAM) den Zeiger auf den String und im 2. Parameter das Statusbyte des Empfangsprotokolls bzw. ein Fehlercode übergeben. Dabei ist zu beachten, dass der Zeiger auf den String mit int gecastet ist und deshalb mit dem cast-Operator (LPCTSTR) bei Zuweisung an ein CString Datentyp bzw. (char*) an eine C-Zeichenkette zurückzuwandeln ist.</td></tr>
</table>

[12] Ereignis wird nur ausgelöst, wenn der Parameter LogProt auf 1 gesetzt wurde (Standard: 0)
[13] Siehe auch Beschreibung zum Parameter ConvHexToString in: 6.2. Liste der Parameterkennungen
[14] Paralleler, vom Applikationsprogramm unabhängiger Ausführungspfad. Auch das Applikationsprogramm ist ein Thread.
[15] Siehe Windows-Dokumentation zur Platform-SDK

2. Methode: Nachricht an Fenster (nicht für Linux, Mac OS X)

Diese Methode verwendet man, wenn die Nachricht direkt an ein Fenster geschickt werden soll. Von dem betreffenden Fenster wird mit der Windows-API-Funktion GetWindow(..)[16] der Handle ermittelt und als Parameter hwndWnd in der **FEISC_EVENT_INIT**-Struktur übergeben. Das Fenster muß für den Empfang der Nachricht, die von FEISC mit der Windows-API-Funktion PostMessage(..) verschickt wurde, eine Nachrichtenbehandlungsfunktion (MessageMap-Funktion) bereitstellen. Der Nachrichtencode ist frei wählbar.

Die **FEISC_EVENT_INIT**-Struktur wird wie folgt ausgefüllt:

```
uiUse = FEISC_xyz_EVENT          // siehe Defines feisc.h
uiMsg = WM_USER + ...            // frei wählbar, aber oberhalb von WM_USER[17]
uiFlag = FEISC_WND_HWND
hwndWnd = GetWindow(...)
```

Die MessageMap-Funktion erhält dieselben Parameter wie die der ersten Methode.

3. Methode: Aufruf der 1. Callback-Funktion (nicht für Mac OS X)

Mit der 1. Callback-Methode wird ein Funktionszeiger für ein Ereignis installiert. Tritt der Event ein, wird die Funktion von FEISC aufgerufen. Der Inhalt der Funktion kann frei bestimmt werden. Die Übergabeparameter sind in der ersten Methode beschrieben.

Die **FEISC_EVENT_INIT**-Struktur wird wie folgt ausgefüllt:

```
uiUse = FEISC_xyz_EVENT          // siehe Defines feisc.h
uiMsg wird nicht benötigt
uiFlag = FEISC_CALLBACK
cbFct = (void*)&IhrFunktionsName[18]
```

4. Methode: Aufruf der 2. Callback-Funktion (nicht für Linux, Mac OS X)

Mit der 2. Callback-Methode wird ein Funktionszeiger für ein Ereignis installiert. Tritt der Event ein, wird die Funktion von FEISC aufgerufen. Der Inhalt der Funktion kann frei bestimmt werden. Die Übergabeparameter sind:

```
BSTR - Zeiger auf einen Unicode-Textpuffer
int  - Anzahl Zeichen im Textpuffer
int  - Statusbyte oder Fehlercode
```

Die **FEISC_EVENT_INIT**-Struktur wird wie folgt ausgefüllt:

```
uiUse = FEISC_xyz_EVENT          // siehe Defines feisc.h
uiMsg = 0                        // wird nicht benötigt
uiFlag = FEISC_CALLBACK_2
cbFct2 = (void*)&IhrFunktionsName[19]
```

5. Methode: Aufruf der 4. Callback-Funktion

Mit der 4. Callback-Methode (die 3. ist nicht öffentlich) wird ein Funktionszeiger für ein Ereignis installiert. Tritt der Event ein, wird die Funktion von FEISC aufgerufen. Der Inhalt der Funktion kann frei bestimmt werden. Die Übergabeparameter sind:

```
void* pAny        - pAny aus FEISC_EVENT_INIT
const char* cMsg  - Zeiger auf Text
```

[16] Bei Verwendung der MFC-Klasse CWnd kann auch die Methode GetSafeHwnd() benutzt werden
[17] Siehe Windows-Dokumentation zur Platform-SDK
[18] Die Funktion hat den Prototyp: void IhrFunktionsName(int, int)
[19] Die Funktion hat den Prototyp: void IhrFunktionsName(BSTR, int, int)

	int iStatus - Statusbyte oder Fehlercode Die **FEISC_EVENT_INIT**-Struktur wird wie folgt ausgefüllt: uiUse = FEISC_xyz_EVENT // siehe Defines feisc.h uiMsg wird nicht benötigt uiFlag = FEISC_CALLBACK_4 pAny = this // Zeiger auf Klasseninstanz, der mit der Callback-Funktion zurück übertragen wird // wenn pAny nicht benötigt wird, dann setzt man ihn auf NULL cbFct4 = (void*)&IhrFunktionsName[20] 6. Methode: Setzen eines Events (nicht für Linux, Mac OS X) Mit der Event-Methode wird ein Event-Handle für ein Ereignis installiert. Tritt ein Ereignis ein, wird der Event von FEISC mit der Windows-API-Funktion SetEvent(...) gesetzt. Auf Seiten der Anwendung wartet man mit der Windows-API-Funktion WaitForSingleObject(...) auf den Event. Da man keine Parameter erhalten kann, muß man mit einer geeigneten Funktion den gewünschten Parameter abfragen. Der gesetzte Event muß vom Anwendungsprogramm mit der Windows-API-Funktion ResetEvent(...) wieder zurückgesetzt werden. Die **FEISC_EVENT_INIT**-Struktur wird wie folgt ausgefüllt: uiUse = FEISC_xyz_EVENT // siehe Defines feisc.h uiMsg wird nicht benötigt uiFlag = FEISC_EVENT hEvent = CreateEvent(..) Jede installierte Ereignisbehandlungsmethode muß wieder mit der Funktion **FEISC_DelEventHandler** entfernt werden. Beim Entfernen eines Leser-Objekts gehen alle für dieses Objekt installierten Ereignisbehandlungsmethoden verloren.
Querverweis	Weitere Informationen in: 5.7.11. FEISC_DelEventHandler, 5.5. Ereignissignalisierung an Applikationen und 6.3. Liste der Konstanten für die FEISC_EVENT_INIT-Struktur
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.

[20] Die Funktion hat den Prototyp: void IhrFunktionsName(void*, const char*, int)

5.7.11. FEISC_DelEventHandler

Funktion	Eine Ereignisbehandlungsmaßnahme wird entfernt
Syntax	**int FEISC_DelEventHandler(int iReaderHnd, FEISC_EVENT_INIT* pInit)**
Beschreibung	Die Funktion entfernt eine zuvor mit **FEISC_AddEventHandler** installierte Ereignisbehandlungsmaßnahme. In der **FEISC_EVENT_INIT**-Struktur spezifiziert man die zu entfernende Ereignisbehandlungsmaßnahme im Detail. Entfernung der 1. Methode: Nachricht an Thread (nicht für Linux, Mac OS X) Die **FEISC_EVENT_INIT**-Struktur wird wie folgt ausgefüllt: uiUse = FEISC_xyz_EVENT // siehe Defines in feisc.h uiMsg wird nicht benötigt uiFlag = FEISC_THREAD_ID dwThreadID = GetCurrentThreadID() Entfernung der 2. Methode: Nachricht an Fenster (nicht für Linux, Mac OS X) Die **FEISC_EVENT_INIT**-Struktur wird wie folgt ausgefüllt: uiUse = FEISC_xyz_EVENT // siehe Defines in feisc.h uiMsg wird nicht benötigt uiFlag = FEISC_WND_HWND hwndWnd = GetWindow(...) Entfernung der 3. Methode: 1. Callback-Funktion (nicht für Visual Basic, Mac OS X) Die **FEISC_EVENT_INIT**-Struktur wird wie folgt ausgefüllt: uiUse = FEISC_xyz_EVENT // siehe Defines feisc.h uiMsg wird nicht benötigt uiFlag = FEISC_CALLBACK cbFct = (void*)&IhrFunktionsName Entfernung der 4. Methode: 2. Callback-Funktion (nicht für Linux, Mac OS X) Die **FEISC_EVENT_INIT**-Struktur wird wie folgt ausgefüllt: uiUse = FEISC_xyz_EVENT // siehe Defines feisc.h uiMsg = 0 // wird nicht benötigt uiFlag = FEISC_CALLBACK_2 cbFct2 = (void*)&IhrFunktionsName Entfernung der 5. Methode: 4. Callback-Funktion (nicht für Visual Basic) Die **FEISC_EVENT_INIT**-Struktur wird wie folgt ausgefüllt: uiUse = FEISC_xyz_EVENT // siehe Defines feisc.h uiMsg wird nicht benötigt uiFlag = FEISC_CALLBACK_4 cbFct4 = (void*)&IhrFunktionsName Entfernung der 5. Methode: Event-Handles (nicht für Linux, Mac OS X) Die **FEISC_EVENT_INIT**-Struktur wird wie folgt ausgefüllt: uiUse = FEISC_xyz_EVENT // siehe Defines feisc.h uiMsg wird nicht benötigt uiFlag = FEISC_EVENT hEvent = IhrEventHandle;

Querverweis	Weitere Informationen in: 5.5. Ereignissignalisierung an Applikationen, 5.7.10. FEISC_AddEventHandler und 6.3. Liste der Konstanten für die FEISC_EVENT_INIT-Struktur
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.

5.7.12. FEISC_StartAsyncTask

<table>
<tr><td>Funktion</td><td colspan="2">Ein Inventarisierungs- bzw. Notificationtask wird asynchron zur Applikation gestartet</td></tr>
<tr><td>Syntax</td><td colspan="2">int FEISC_StartAsyncTask(int iReaderHnd, int iTaskID, FEISC_TASK_INIT* pInit, void* pInput)</td></tr>
<tr><td>Beschreibung</td><td colspan="2">Mit dieser Funktion wird ein asynchroner Task gestartet. Ein asynchroner Task ist ein interner Thread, der z.B. ein Inventory-Kommando an den Leser sendet und für eine Zeit Timeout auf die Antwort wartet. Die Signalisierung der Antwortdaten bzw. der Abbruchbedingung an die Applikation erfolgt mit dem Aufruf einer Callback-Funktion.

Das Taskverhalten wird im Parameter iTaskID spezifiziert. Folgendei Aufgaben sind z.Zt. definiert:</td></tr>
<tr><td></td><td>FEISC_TASKID_FIRST_NEW_TAG</td><td>startet einen einmaligen Inventarisierungs-auftrag</td></tr>
<tr><td></td><td>FEISC_TASKID_EVERY_NEW_TAG</td><td>startet einen repetierenden Inventarisie-rungsauftrag</td></tr>
<tr><td></td><td>FEISC_TASKID_NOTIFICATION</td><td>startet einen dauerhaften Auftrag zum Empfang von Notifications</td></tr>
<tr><td></td><td>FEISC_TASKID_SAM_COMMAND</td><td>startet einen einmaligen Auftrag zum Empfang der SAM-Daten</td></tr>
<tr><td></td><td>FEISC_TASKID_COMMAND_QUEUE</td><td>startet einen einmaligen Auftrag zum Empfang der Antwort</td></tr>
<tr><td></td><td>FEISC_TASKID_MAX_EVENT</td><td>startet einen dauerhaften Auftrag zum Empfang von Access-Notifications</td></tr>
<tr><td></td><td>FEISC_TASKID_PEOPLE_COUNTER</td><td>startet einen dauerhaften Auftrag zum Empfang von Counter-Notifications</td></tr>
<tr><td></td><td colspan="2">Alle für den Task und für die Callback-Funktion relevanten Daten sind in der Struktur FEISC_TASK_INIT zusammengefasst. Diese Struktur wird im Kapitel 5.4. Asynchrone Tasks zur Entlastung von Applikationen genauer beschrieben.

Wichtiger Hinweis: die Struktur FEISC_TASK_INIT muss vor der Verwendung immer mit 0 initialisiert werden z.B. mittels memset(myTaskInit, 0, sizeof(FEISC_TASK_INIT));

Der letzte Parameter pInput wird z.Zt. nicht beachtet. Man sollte immer NULL (vbNull) übergeben.

iReaderHnd ist der Handle zum Leser-Objekt.</td></tr>
<tr><td>Querverweise</td><td colspan="2">Weitere Informationen zu asynchronen Tasks finden sich im Kapitel 5.4. Asynchrone Tasks zur Entlastung von Applikationen.

5.7.13. FEISC_CancelAsyncTask

5.7.14. FEISC_TriggerAsyncTask</td></tr>
<tr><td>Anmerkung</td><td colspan="2">Asynchrone Inventarisierungstasks benutzen das Protokoll [0xB0][0x01] Inventory mit</td></tr>
</table>

	der Option NOTIFY im Mode-Byte. Leser, die diese Option nicht unterstützen, können für asynchrone Tasks nicht verwendet werden. Nähere Informationen zum Protokoll [0xB0][0x01] Inventory finden sich im Systemhandbuch zur OBID *i-scan*® oder OBID® *classic-pro* Leserfamilie.
Rückgabewert	Im fehlerfreien Fall wird eine 0 zurückgegeben. Ein Wert kleiner als 0 zeigt einen Fehler an. Die Liste der Fehlercodes findet sich im Anhang.

5.7.13. FEISC_CancelAsyncTask

Funktion	Ein Inventarisierungs- bzw. Notificationtask wird abgebrochen
Syntax	**int FEISC_CancelAsyncTask(int iReaderHnd)**
Beschreibung	Mit dieser Funktion wird ein asynchroner Task sofort beendet. Allerdings wird nur der interne Thread beendet. Der Task im Leser kann nicht unterbrochen werden. Den einmaligen Inventory (gestartet mit TaskID = FEISC_TASKID_FIRST_NEW_TAG) sollte man normalerweise nicht mit dieser Funktion beenden. Den repetierenden Inventory (gestartet mit TaskID = FEISC_TASKID_EVERY_NEW_TAG) sollte man dann mit dieser Funktion beenden, wenn die Callback-Funktion beendet wurde und der interne Thread auf den nächsten Trigger wartet. So ist sichergestellt, dass der Task im Leser beendet ist und er wieder Leserprotokolle bearbeiten kann. Notificationtasks müssen immer mit dieser Funktion beendet werden. Zur Vermeidung von Deadlocks wird der Task nicht beendet, wenn der Ausführungspfad des Tasks innerhalb der Callback-Funktion liegt. In diesem Fall kehrt die Funktion sofort mit dem Rückgabewert FEISC_ERR_THREAD_CANCEL_ERROR (-4084) zurück und die Applikation muß FEISC_CancelAsyncTask solange aufrufen, bis der Rückgabewert nicht mehr -4084 ist. Applikations-seitig muss sichergestellt werden, dass die Callback-Funktion immer zuverlässig zurückkehrt. *iReaderHnd* ist der Handle zum Leser-Objekt.
Querverweise	Weitere Informationen zu asynchronen Tasks finden sich im Kapitel 5.4. Asynchrone Tasks zur Entlastung von Applikationen. 5.7.12. FEISC_StartAsyncTask 5.7.14. FEISC_TriggerAsyncTask
Rückgabewert	Im fehlerfreien Fall wird eine 0 zurückgegeben. Ein Wert kleiner als 0 zeigt einen Fehler an. Die Liste der Fehlercodes findet sich im Anhang.

5.7.14. FEISC_TriggerAsyncTask

Funktion	Der nächste Zyklus im Inventarisierungstask wird angestoßen
Syntax	**int FEISC_TriggerAsyncTask(int iReaderHnd)**
Beschreibung	Mit dieser Funktion wird der nächste Inventory-Zyklus im asynchroner Task angestoßen. Der asynchrone Task muß zuvor mit der TaskID = FEISC_TASKID_EVERY_NEW_TAG gestartet sein. Aufgerufen wird diese Funktion immer dann, nachdem die Callback-Funktion verlassen wurde. Ohne diesen Aufruf bleibt ein Task mit repetierender Funktion in einer Warteschleife hängen. *iReaderHnd* ist der Handle zum Leser-Objekt.
Querverweise	Weitere Informationen zu asynchronen Tasks finden sich im Kapitel 5.4. Asynchrone Tasks zur Entlastung von Applikationen. 5.7.12. FEISC_StartAsyncTask 5.7.13. FEISC_CancelAsyncTask
Rückgabewert	Im fehlerfreien Fall wird eine 0 zurückgegeben. Ein Wert kleiner als 0 zeigt einen Fehler an. Die Liste der Fehlercodes findet sich im Anhang.

5.7.15. FEISC_BuildSendProtocol

Funktion	The transmitted parameters and data are used to build a receive protocol with a protocol frame.
Syntax	**int FEISC_BuildSendProtocol(int iReaderHnd, UCHAR cBusAdr, UCHAR cCmdByte, UCHAR* cSendData, int iDataLen, UCHAR* cSendProt, int iDataFormat)**
Beschreibung	Diese Funktion baut aus den übergebenen Parametern Busadresse (*cBusAdr*), Steuerbyte (*cCmdByte*), Sendedaten (*cSendData*) und der Information über die Länge der Sendedaten (*iDataLen*) ein komplettes Sendeprotokoll mit Protokollrahmen zusammen. Der Protokollstring wird in *cSendProt* als Hex-Array (*iDataFormat*=0) oder Zeichenkette (*iDataFormat*=1) hinterlegt. Der Puffer für *cSendProt* muß in der Dimension mindestens um eins größer als die erwartete Protokollänge sein, da ein NUL-Zeichen angehängt wird. Nähere Informationen zum Protokollrahmen finden sich im Systemhandbuch zur OBID *i-scan*® oder OBID® *classic-pro* Leserfamilie. *iReaderHnd* ist der Handle zum Leser-Objekt. Das zusammengebaute Protokoll wird nicht an einen Porttreiber (z.B. FECOM) weitergegeben.
Querverweis	Weitere Informationen zu *iDataFormat* in Kapitel 5.3. Parameterübergabe
Anmerkung	Diese Funktion unterstützt nicht die USB-Protokolle.
Rückgabewert	Im fehlerfreien Fall wird im Rückgabewert die Länge von cSendProt angegeben. Im Fehlerfall wird ein negativer Wert zurückgegeben. Die Liste der Fehlercodes findet sich im Anhang.
Beispiel	... int BuildTestProtocol(int iReaderHnd) { int iErr, iDataLen; UCHAR cSendData[32], cSendProt[256]; UCHAR cBusAdr = 0xFF; UCHAR cCmdByte= 0x6A; ... cSendData[0] = 0x01; cSendData[1] = '\0'; iDataLen = 1; // Sende-Protokoll aufbauen iErr = FEISC_BuildSendProtocol(iReaderHnd, cBusAdr, cCmdByte, cSendData, iDataLen, cSendProt, 0); ... }

5.7.16. FEISC_BuildRecProtocol

Funktion	Aus den übergebenen Parametern und Daten wird ein Empfangs-Protokoll mit Protokollrahmen zusammengebaut.
Syntax	**int FEISC_BuildRecProtocol(int iReaderHnd, UCHAR cBusAdr, UCHAR cCmdByte, UCHAR cStatus, UCHAR* cRecData, int iDataLen, UCHAR* cRecProt, int iDataFormat)**
Beschreibung	Diese Funktion baut aus den übergebenen Parametern Busadresse (*cBusAdr*), Steuerbyte (*cCmdByte*), Statusbyte (*cStatus*), Empfangsdaten (*cRecData*) und der Information über die Länge der Empfangsdaten (*iDataLen*) ein komplettes Empfangsprotokoll mit Protokollrahmen zusammen. Der Protokollstring wird in *cRecProt* als Hex-Array (*iDataFormat*=0) oder Zeichenkette (*iDataFormat*=1) hinterlegt. Der Puffer für *cRecProt* muß in der Dimension mindestens um eins größer als die erwartete Protokollänge sein, da ein NUL-Zeichen angehängt wird. Nähere Informationen zum Protokollrahmen finden sich im Systemhandbuch zur OBID *i-scan*® oder OBID® *classic-pro* Leserfamilie. *iReaderHnd* ist der Handle zum Leser-Objekt. Das zusammengebaute Protokoll wird nicht an einen Porttreiber (z.B. FECOM) weitergegeben.
Querverweis	Weitere Informationen zu *iDataFormat* in Kapitel 5.3. Parameterübergabe
Anmerkung	Diese Funktion unterstützt nicht die USB-Protokolle.
Rückgabewert	Im fehlerfreien Fall wird im Rückgabewert die Länge von *cRecProt* angegeben. Im Fehlerfall wird ein negativer Wert zurückgegeben. Die Liste der Fehlercodes findet sich im Anhang.
Beispiel	Analog zu FEISC_BuildSendProt

5.7.17. FEISC_SplitSendProtocol

Funktion	Der übergebene Protokollstring wird zerlegt.
Syntax	**int FEISC_SplitSendProtocol(int iReaderHnd, UCHAR* cSendProt, int iSendLen, UCHAR* cBusAdr, UCHAR* cCmdByte, UCHAR* cSendData, int* iDataLen, int iDataFormat)**
Beschreibung	Diese Funktion zerlegt die in *cSendProt* enthaltenen Daten in Busadresse (*cBusAdr*), Steuerbyte (*cCmdByte*), Senddaten (*cSendData*) und der Information über die Länge der Senddaten (*iDataLen*). Der Protokollstring in *cSendProt* muß als Hex-Array (*iDataFormat*=0) oder Zeichenkette (*iDataFormat*=1) mit einer Längenangabe in *iSendLen* übergeben werden. *cSendData* wird als Hex-Array (*iDataFormat*=0) oder Zeichenkette (*iDataFormat*=1) interpretiert. Nähere Informationen zum Protokollrahmen finden sich im Systemhandbuch zur OBID *i-scan*® oder OBID® *classic-pro* Leserfamilie. *iReaderHnd* ist der Handle zum Leser-Objekt. Diese Funktion ist unabhängig vom Porttreiber (z.B. FECOM).
Querverweis	Weitere Informationen zu *iDataFormat* in Kapitel 5.3. Parameterübergabe
Anmerkung	Diese Funktion unterstützt nicht die USB-Protokolle.
Rückgabewert	Im fehlerfreien Fall wird eine 0 zurückgegeben. Ein Wert kleiner als 0 zeigt einen Fehler an. Die Liste der Fehlercodes findet sich im Anhang.
Beispiel	Analog zu FEISC_SplitRecProt

5.7.18. FEISC_SplitRecProtocol

Funktion	Der übergebene Protokollstring wird zerlegt.
Syntax	**int FEISC_SplitRecProtocol(int iReaderHnd, UCHAR* cRecProt, int iRecLen, UCHAR* cBusAdr, UCHAR* cCmdByte, UCHAR* cRecData, int* iDataLen, int iDataFormat)**
Beschreibung	Diese Funktion zerlegt die in *cRecProt* enthaltenen Daten in Busadresse (*cBusAdr*), Steuerbyte (*cCmdByte*), Empfangsdaten (*cRecData*) und der Information über die Länge der Empfangsdaten (*iDataLen*). Der Protokollstring in *cRecProt* muß als Hex-Array (*iDataFormat*=0) oder Zeichenkette (*iDataFormat*=1) mit einer Längenangabe in *iRecLen* übergeben werden. *cRecData* wird als Hex-Array (*iDataFormat*=0) oder Zeichenkette (*iDataFormat*=1) interpretiert. Nähere Informationen zum Protokollrahmen finden sich im Systemhandbuch zur OBID *i-scan*® oder OBID® *classic-pro* Leserfamilie. *iReaderHnd* ist der Handle zum Leser-Objekt. Diese Funktion ist unabhängig vom Porttreiber (z.B. FECOM).
Querverweis	Weitere Informationen zu *iDataFormat* in Kapitel 5.3. Parameterübergabe
Anmerkung	Diese Funktion unterstützt nicht die USB-Protokolle.
Rückgabewert	Im fehlerfreien Fall wird das Statusbyte des Empfangsprotokolls zurückgegeben. Ein Wert größer als 0x00 zeigt eine Ausnahmebedingung des Lesers an. Die Liste der Fehlercodes findet sich im Anhang.
Beispiel	// das nachfolgende Codefragment setzt initialisierte Port- und Leser-Objekte voraus. #include "feisc.h" #include "fecom.h" ... int iStatus, iRecLen; UCHAR cBusAdr, cCmdByte; UCHAR cSendProt[256], cRecProt[256], cRecData[256]; int iDataLen = 0; ... // Protokoll senden und empfangen iRecLen = FECOM_Transceive(iPortHnd, cSendProt, cSendProt[0], cRecProt, 256); if(iRecLen > 0) { // Empfangs-Protokoll zerlegen iStatus = FEISC_SplitRecProtocol(iReaderHnd, cRecProt, iRecLen, &cBusAdr, &cCmdByte, cRecData, &iDataLen, 0); if(iStatus == 0) // Statusbyte == 0x00 { // Empfangsdaten verarbeiten ... } }

5.7.19. FEISC_SendTransparent

Funktion	Ein Protokollstring wird direkt über den Porttreiber ausgegeben und das Empfangsprotokoll zurückgegeben.
Syntax	**int FEISC_SendTransparent(int iReaderHnd, UCHAR* cSendProt, int iSendLen, UCHAR* cRecProt, int iMaxRecLen, int iCheckSum, int iDataFormat)**
Beschreibung	Diese Funktion ist dazu geeignet, mit Editoren erstellte Protokollstrings an einen Leser zu senden. Das setzt allerdings eingehende Kenntnisse zum Protokollrahmen voraus. Das in *cSendProt* enthaltene Protokoll mit Protokollrahmen wird optional mit der Checksumme ergänzt (*iCheckSum* = 1), das Empfangsprotokoll in *cRecProt* abgelegt. Beide Puffer sind als Hex-Array (*iDataFormat*=0) oder Zeichenkette (*iDataFormat*=1) zu interpretieren. Im Parameter *iSendLen* muß die Länge des Protokolls (Anzahl Zeichen in *cSendProt*) angegeben werden. Der Puffer des Empfangsprotokolls (mit Standard Protokollrahmen) sollte sicherheitshalber 256 Zeichen (*iDataFormat*=0) bzw. 512 Zeichen (*iDataFormat*=1) aufnehmen können. Diese Puffergröße muß in *iMaxRecLen* angegeben werden. Bei Advanced Protokollrahmen müssen die Puffer eventuell entsprechend vergrößert werden. *iReaderHnd* ist der Handle zum Leser-Objekt.
Querverweis	Weitere Informationen zu *iDataFormat* in Kapitel 5.3. Parameterübergabe
Anmerkung	Diese Funktion unterstützt alle FEIG Porttreiber (FECOM, FEUSB, FETCP) und auch per Plug-In Technik eingebundene Porttreiber anderer Hersteller.
Rückgabewert	Im fehlerfreien Fall wird die Anzahl der in *cRecProt* enthaltenen Zeichen übergeben. Die Liste der Fehlercodes findet sich im Anhang.
Beispiel	`int outLen, inLen;` `UCHAR cSendProt[256];` `UCHAR cRecProt[256];` `  // Sendeprotokoll definieren` `cSendProt[0] = 0x06;  // Längenbyte` `cSendProt[1] = 0xFF;  // Adressbyte` `cSendProt[2] = 0x80;  // Steuerbyte` `cSendProt[3] = 0x00;  // Konfigurations-Adresse im Leser` `outLen = 4;` `  // Protokoll senden, zuvor Checksumme berechnen und anhängen` `inLen = FEISC_SendTransparent( iReaderHnd, cSendProt, outLen, cRecProt, 256, 1, 0 );` `if( inLen > 0)` `{  // ab hier Code zur Auswertung der Empfangsdaten` `  ...` `}`

5.7.20. FEISC_Transmit

Funktion	Ein Protokollstring wird direkt über die Schnittstelle ausgegeben.
Syntax	**int FEISC_Transmit(int iReaderHnd, UCHAR* cSendProt, int iSendLen, int iCheckSum, int iDataFormat)**
Beschreibung	Diese Funktion ist dazu geeignet, mit Editoren erstellte Protokollstrings an einen Leser zu senden. Das setzt allerdings eingehende Kenntnisse zum Protokollrahmen voraus. Es wird nach dem Senden des Protokolls *cSendProt* nicht auf ein Antwortprotokoll gewartet. Das in *cSendProt* enthaltene Protokoll mit Protokollrahmen wird optional mit der Checksumme ergänzt (*iCheckSum*=1). *cSendProt* wird als Hex-Array (*iDataFormat*=0) oder als Zeichenkette (*iDataFormat*=1) interpretiert. Im Parameter *iSendLen* muß die Länge des Protokolls (Anzahl Zeichen in *cSendProt*) angegeben werden. Wenn *iDataFormat*=1 ist, dann ist *iSendLen* also genau doppelt so groß, als im Fall *iDataFormat*=0. *iReaderHnd* ist der Handle zum Leser-Objekt.
Querverweis	Weitere Informationen zu *iDataFormat* in Kapitel 5.3. Parameterübergabe
Anmerkung	Diese Funktion unterstützt alle FEIG Porttreiber (FECOM, FEUSB, FETCP) und auch per Plug-In Technik eingebundene Porttreiber anderer Hersteller.
Rückgabewert	Im fehlerfreien Fall wird eine 0 übergeben. Die Liste der Fehlercodes findet sich im Anhang.
Beispiel	int outLen; UCHAR cSendProt[256]; ... // Sendeprotokoll definieren cSendProt[0] = 0x06; // Längenbyte cSendProt[1] = 0xFF; // Adressbyte cSendProt[2] = 0x80; // Steuerbyte für Read Configuration cSendProt[3] = 0x00; // Konfigurations-Adresse im Leser outLen = 4; // Protokoll senden, zuvor Checksumme berechnen und anhängen FEISC_Transmit(iReaderHnd, cSendProt, outLen, 1, 0); ...

5.7.21. FEISC_Receive

Funktion	Ein Protokollstring wird direkt von der Schnittstelle eingelesen.
Syntax	**int FEISC_Receive(int iReaderHnd, UCHAR* cRecProt, int iMaxRecLen, int iDataFormat)**
Beschreibung	Diese Funktion liest ein Protokoll aus dem Empfangspuffer aus und hinterlegt es in *cRecProt*. Wenn ein Leser bereits mehrere Protokolle abgeschickt hat, liest die Funktion alle Protokolle ein. *cRecProt* enthält in diesem Fall alle Protokolle. Es können maximal 256 ASCII-Zeichen aus dem Empfangspuffer übernommen werden. Der Puffer des Empfangsprotokolls (mit Standard Protokollrahmen) sollte sicherheitshalber 256 Zeichen (*iDataFormat*=0) bzw. 512 Zeichen (*iDataFormat*=1) aufnehmen können. Diese Puffergröße muß in *iMaxRecLen* angegeben werden. Bei Advanced Protokollrahmen müssen die Puffer eventuell entsprechend vergrößert werden. *iReaderHnd* ist der Handle zum Leser-Objekt.
Querverweis	Weitere Informationen zu *iDataFormat* in Kapitel 5.3. Parameterübergabe
Anmerkung	Diese Funktion unterstützt alle FEIG Porttreiber (FECOM, FEUSB, FETCP) und auch per Plug-In Technik eingebundene Porttreiber anderer Hersteller.
Rückgabewert	Im fehlerfreien Fall wird die Anzahl der in cRecProt enthaltenen Zeichen übergeben. Wenn *iDataFormat*=1 ist, dann ist die Anzahl genau doppelt so groß, als im Fall *iDataFormat*=0. Die Liste der Fehlercodes findet sich im Anhang.
Beispiel	int inLen; UCHAR cRecProt[256]; ... // Protokoll auslesen inLen = FEISC_Receive(iReaderHnd, cRecProt, 256, 0); ...

5.7.22. FEISC_GetLastSendProt

Funktion	Der zuletzt gesendete Protokollstring wird zurückgegeben.
Syntax	**int FEISC_GetLastSendProt(int iReaderHnd, UCHAR* cSendProt, int iDataFormat)**
Beschreibung	Mit dieser Funktion kann man aus einem Leser-Objekt das zuletzt ausgegebene Sendeprotokoll ermitteln. Alle Funktionen, die mit **FEISC_0x...** beginnen, sowie die Funktion **FEISC_SendTransparent** hinterlegen im Leser-Objekt dieses Protokoll. Der Puffer für das Sendeprotokoll *cSendProt* sollte sicherheitshalber 256 Zeichen (*iDataFormat*=0) bzw. 512 Zeichen (*iDataFormat*=1) aufnehmen können. *cSendProt* ist als Hex-Array (*iDataFormat*=0) oder Zeichenkette (*iDataFormat*=1) zu interpretieren. *iReaderHnd* ist der Handle zum Leser-Objekt.
Querverweis	Weitere Informationen zu *iDataFormat* in Kapitel 5.3. Parameterübergabe
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall die in *cSendProt* enthaltenen Zeichenzahl. Die Liste der Fehlercodes findet sich im Anhang.

5.7.23. FEISC_GetLastRecProt

Funktion	Der zuletzt empfangene Protokollstring wird zurückgegeben.
Syntax	**int FEISC_GetLastRecProt(int iReaderHnd, UCHAR* cRecProt, int iDataFormat)**
Beschreibung	Mit dieser Funktion kann man aus einem Leser-Objekt das letzte Empfangsprotokoll ermitteln. Alle Funktionen, die mit **FEISC_0x...** beginnen, sowie die Funktion **FEISC_SendTransparent** hinterlegen im Leser-Objekt dieses Protokoll. Der Puffer für das Empfangsprotokoll *cRecProt* sollte sicherheitshalber 256 Zeichen (*iDataFormat*=0) bzw. 512 Zeichen (*iDataFormat*=1) aufnehmen können. *cRecProt* ist als Hex-Array (*iDataFormat*=0) oder Zeichenkette (*iDataFormat*=1) zu interpretieren. *iReaderHnd* ist der Handle zum Leser-Objekt.
Querverweis	Weitere Informationen zu *iDataFormat* in Kapitel 5.3. Parameterübergabe
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall die in *cRecProt* enthaltenen Zeichenzahl. Die Liste der Fehlercodes findet sich im Anhang.

5.7.24. FEISC_GetLastState

Funktion	Das im letzten Empfangsprotokoll enthaltene Statusbyte wird zurückgegeben.
Syntax	**int FEISC_GetLastStatus(int iReaderHnd, char* cStatusText)**
Beschreibung	Mit dieser Funktion kann man aus einem Leser-Objekt das Statusbyte und einen Kurztext zum Statusbyte des letzten Empfangsprotokolls ermitteln. Alle Funktionen, die mit **FEISC_0x...** beginnen, sowie die Funktion **FEISC_SendTransparent** hinterlegen im Leser-Objekt das letzte Statusbyte. Der Puffer für den Kurztext *cStateText* sollte 256 Zeichen aufnehmen können. *iReaderHnd* ist der Handle zum Leser-Objekt.
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte. Die Liste der Fehlercodes findet sich im Anhang.

5.7.25. FEISC_GetLastRecProtLen

Function	Die Länge des letzten Empfangsprotokolls wird ermittelt
Syntax	**int FEISC_GetLastRecProtLen(int iReaderHnd)**
Description	Manchmal ist es hilfreich, aus der Protokollänge die Länge der darin enthaltenen Daten ermitteln zu können. Genau diese Protokollänge wird mit dieser Funktion ermittelt. Beispiel: die Funktion **FEISC_0x21_ReadBuffer** liefert einige Datensätze einer Datenstruktur. Man könnte die Gesamtlänge der Daten durch Analyse der Datensätze ermitteln, viel einfacher ist es jedoch, wenn man die Protokollänge heranzieht und 6 Bytes für den Protokollrahmen und weitere 2 Bytes für die Parameter *cTrData* und *cRecDataSets* abzieht. *iReaderHnd* ist der Handle zum Leser-Objekt.
Return value	Der Rückgabewert enthält im fehlerfreien Fall die Protokollänge. Die Liste der Fehlercodes findet sich im Anhang.

5.7.26. FEISC_GetLastError

Funktion	Ermittelt den letzten Fehlercode und übergibt Fehlertext
Syntax	**int FEISC_GetLastError(int iReaderHnd , int* iErrorCode, char* cErrorText)**
Beschreibung	Die Funktion übergibt in *iErrorCode* den letzten Fehlercode des mit *iReaderHnd* ausgewählten Leser-Objekts zurück und übergibt in *cErrorText* den zugehörigen englischen Fehlertext. Der Puffer für *cErrorText* sollte 256 Zeichen aufnehmen können. Setzt man *iReaderHnd* 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 "feisc.h" ... ... char cErrorText[256]; int iErrorCode = 0; ... int iBack = FEISC_GetLastError(iReaderHnd, &iErrorCode, cErrorText) // hier Code zum Anzeigen des Textes ... ...

5.7.27. FEISC_0x18_Destroy

Funktion	Funktion zerstört einen Transponder unwiderruflich.
Syntax	**int FEISC_0x18_Destroy(int iReaderHnd, UCHAR cBusAdr, UCHAR cMode, UCHAR* cEPC, UCHAR* cPW)**
Beschreibung	Die Funktion zerstört den im Antennenfeld befindlichen Transoponder unwiderruflich. *cMode* ist das Mode Byte zum Befehl. *cEPC* ist ein Zeiger auf die EPC oder UID des Transponders. Die Länge der EPC/UID wird intern anhand des Modebytes und des EPC-Headers berechnet. *cPW* ist ein Zeiger auf 3 Byte Passwort. *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im multijob-Leser eingestellte Busadresse.
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.28. FEISC_0x1A_Halt

Funktion	Funktion zum Stummschalten von Transpondern
Syntax	**int FEISC_0x1A_Halt(int iReaderHnd, UCHAR cBusAdr)**
Beschreibung	Ein zuvor selektierten Transponder wird mit dieser Funktion abgeschaltet. Mit der Funktion **FEISC_0x69_RFReset** können alle abgeschalteten Transponder wieder aktiviert werden. *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im Leser eingestellte Busadresse.
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.29. FEISC_0x1B_ResetQuietBit

Funktion	Funktion zum Rücksetzen der Quiet-Bits.
Syntax	**int FEISC_0x1B_ResetQuietBit(int iReaderHnd, UCHAR cBusAdr)**
Beschreibung	Die Funktion setzt die Quiet-Bits im Transpondertyp I-Code zurück. *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im Leser eingestellte Busadresse.
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.30. FEISC_0x1C_EASRequest

Funktion	Funktion zum Senden des EAS Request.
Syntax	**int FEISC_0x1C_EASRequest(int iReaderHnd, UCHAR cBusAdr)**
Beschreibung	Die Funktion sendet ein EAS Request an den Transpondertyp I-Code. *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im Leser eingestellte Busadresse.
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.31. FEISC_0x1E_TableDataExchange

Funktion	Funktion zum Datentransfer mit einem Reader
Syntax	**int FEISC_0x1F_TableDataExchange(int iReaderHnd, UCHAR cBusAdr, UCHAR cSubCmd, UCHAR cMode, UCHAR cDevice, UCHAR cBank, UCHAR cTableID, UCHAR* cReqData, int iReqDataLen, UCHAR* cRspData, int* iRspDataLen)**
Beschreibung	Mit dieser Funktion werden alle Datensätze verschiedener Tabellen in einen Reader geschrieben oder davon gelesen. *cSubCmd* enthält das Steuerbyte das die Aktion definiert. *cMode* enthält optionale Flags. *cDevice* spezifiziert den internen Prozessor. *cBank* spezifiziert die Speicherbank. *cTableID* spezifiziert die Tabelle entsprechend dem Systemhandbuch. Die Tabellendaten sind in *cReqData* zu hinterlegen. In *iReqDataLen* muß die Anzahl der in *cReqData* enthaltenen Zeichen angegeben werden. Die Anzahl der zurückgegebenen Datenbytes in *cRspData* ist in *iRspDataLen* angegeben. Dabei ist folgendes zu beachten: der Puffer für die Empfangsdaten *cRspData* muß so dimensioniert werden, dass alle Empfangsdaten gespeichert werden können. *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im multijob-Leser eingestellte Busadresse.
Hinweis	Diese Funktion ist eine Low-Level-Funktion und sollte nicht direkt benutzt werden. Zur Applikationserstellung mit myAxxess Readern ist die komfortable C++ Bibliothek FEDM verfügbar.
Querverweis	Weitere Informationen zu *iDataFormat* in Kapitel 5.3. Parameterübergabe
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.32. FEISC_0x1F_MAXDataExchange

Funktion	Funktion zum Datentransfer mit einem myAxxess Reader
Syntax	**int FEISC_0x1F_MAXDataExchange(int iReaderHnd, UCHAR cBusAdr, UCHAR cSubCmd, UCHAR cMode, UCHAR cTableID, UCHAR* cReqData, int iReqDataLen, UCHAR* cRspData, int* iRspDataLen, int iDataFormat)**
Beschreibung	Mit dieser Funktion werden alle Datensätze verschiedener Tabellen in den myAxxess Reader geschrieben oder davon gelesen. *cSubCmd* enthält das Steuerbyte das die Aktion definiert. *cMode* enthält optionale Flags. *cTableID* spezifiziert die Tabelle entsprechend dem Systemhandbuch. Die Tabellendaten sind in *cReqData* zu hinterlegen. In *iReqDataLen* muß die Anzahl der in *cReqData* enthaltenen Zeichen angegeben werden. Der Parameter *iDataFormat* bestimmt, ob die Sendedaten in *cReqData* und die Empfangsdaten in *cRspData* als Hex-Array (*iDataFormat*=0) oder als Zeichenkette (*iDataFormat*=1) zu interpretieren sind. Das bedeutet im Fall *iDataFormat*=1, dass die Größe des Puffers *cRspData* doppelt so groß ist, als im Fall *iDataFormat*=0. Im Parameter *iReqLen* muß die Länge der Sendedaten (Anzahl Zeichen in *cReqData*) angegeben werden. Wenn *iDataFormat*=1 ist, dann ist *iReqLen* also genau doppelt so groß, als im Fall *iDataFormat*=0. Analog dazu ist die Längenangabe für den Empfangspuffer (*iRspDataLen*) auszuwerten. Die Anzahl der zurückgegebenen Datenbytes in *cRspData* ist in *iRspDataLen* angegeben. Dabei ist folgendes zu beachten: der Puffer für die Empfangsdaten *cRspData* muß so dimensioniert werden, dass alle Empfangsdaten gespeichert werden können. *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im multijob-Leser eingestellte Busadresse.
Hinweis	Diese Funktion ist eine Low-Level-Funktion und sollte nicht direkt benutzt werden. Zur Applikationserstellung mit myAxxess Readern ist die komfortable C++ Bibliothek FEDM verfügbar.
Querverweis	Weitere Informationen zu *iDataFormat* in Kapitel 5.3. Parameterübergabe
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.33. FEISC_0x21_ReadBuffer

Funktion	Funktion zum Datentransfer mit einem Transponder
Syntax	**int FEISC_0x21_ReadBuffer(int iReaderHnd, UCHAR cBusAdr, UCHAR cSets, UCHAR* cTrData, UCHAR* cRecSets, UCHAR* cRecDataSets, int iDataFormat)**
Beschreibung	Die Funktion liest aus der internen Datentabelle die Anzahl Datensätze *cSets* aus und legt die Daten in *cRecDataSets* ab. Die Anzahl der übertragenen Datensätze ist limitiert durch den Protokollrahmen. *cTrData* definiert die Struktur eines Datensatzes in *cRecDataSets*. Die Anzahl der zurückgegebenen Datensätze in *cRecDataSets* ist in *cRecSets* angegeben. Der Parameter *iDataFormat* bestimmt, ob die Empfangsdaten in *cRecDataSets* als ein Hex-Array oder als eine Zeichenkette zu interpretieren ist. *cRecSets* und *cTrData* bestehen immer aus 1 Hex-Zeichen. Der Puffer von *cRecDataSets* sollte folgend dimensioniert sein: • iDataFormat=0: 256 Zeichen (incl 1 NUL-Zeichen) • iDataFormat=1: 512 Zeichen (incl. 1 NUL-Zeichen Die in *cRecDataSets* enthaltenen Daten sind in der Reihenfolge eingefügt, wie dies im Systemhandbuch zur OBID *i-scan*® -Familie dokumentiert ist. *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im multijob-Leser eingestellte Busadresse.
Hinweis	Die Funktion führt keine Überprüfung der Daten in *cRecDataSets* anhand der in *cTrData* angegebenen Datenstruktur durch.
Querverweis	Weitere Informationen zu *iDataFormat* in Kapitel 5.3. Parameterübergabe sowie **FEISC_0x33_InitBuffer, FEISC_0x31_ReadDataBufferInfo, FEISC_0x32_ClearDataBuffer**
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.34. FEISC_0x22_ReadBuffer

Funktion	Funktion zum Datentransfer mit einem Transponder
Syntax	**int FEISC_0x22_ReadBuffer(int iReaderHnd, UCHAR cBusAdr, int iSets, UCHAR* cTrData, int* iRecSets, UCHAR* cRecDataSets, int iDataFormat)**
Beschreibung	Die Funktion liest aus der internen Datentabelle die Anzahl Datensätze *iSets* aus und legt die Daten in *cRecDataSets* ab. *cTrData* definiert die Struktur eines Datensatzes in *cRecDataSets*. Die Anzahl der zurückgegebenen Datensätze in *cRecDataSets* ist in *iRecSets* angegeben. Der Parameter *iDataFormat* bestimmt, ob die Empfangsdaten in *cRecDataSets* als ein Hex-Array oder als eine Zeichenkette zu interpretieren ist. *cTrData* besteht immer aus 1 Hex-Zeichen. Der Puffer von *cRecDataSets* sollte so dimensioniert sein, dass alle Transponderdaten hineinpassen. Wenn *iDataFormat*=1 ist, dann muß der Puffer nochmals verdoppelt werden. Die in *cRecDataSets* enthaltenen Daten sind in der Reihenfolge eingefügt, wie dies im Systemhandbuch zur OBID *i-scan*® -Familie dokumentiert ist. *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im multijob-Leser eingestellte Busadresse.
Hinweis	Die Funktion führt keine Überprüfung der Daten in *cRecDataSets* anhand der in *cTrData* angegebenen Datenstruktur durch.
Querverweis	Weitere Informationen zu *iDataFormat* in Kapitel 5.3. Parameterübergabe sowie **FEISC_0x33_InitBuffer, FEISC_0x31_ReadDataBufferInfo, FEISC_0x32_ClearDataBuffer**
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.35. FEISC_0x31_ReadDataBufferInfo

Funktion	Funktion ermittelt Tabellenparameter des internen DataBuffers
Syntax	**int FEISC_0x31_ReadDataBufferInfo(int iReaderHnd, UCHAR cBusAdr, UCHAR* cTabSize, UCHAR* cTabStart, UCHAR* cTabLen, int iDataFormat)**
Beschreibung	Die Funktion liest zu der internen Datentabelle die Tabellenparameter aus und legt sie in *cTabSize* , *cTabStart* und *cTabLen* ab. Der Parameter *iDataFormat* bestimmt, ob die Tabellenparameter jeweils als ein Hex-Array oder als eine Zeichenkette zu interpretieren sind. Die Puffer von *cTabSize* , *cTabStart* und *cTabLen* müssen jeweils folgend dimensioniert sein: • iDataFormat=0: 3 Zeichen (incl 1 NUL-Zeichen) • iDataFormat=1: 5 Zeichen (incl. 1 NUL-Zeichen *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im multijob-Leser eingestellte Busadresse.
Querverweis	Weitere Informationen zu *iDataFormat* in Kapitel 5.3. Parameterübergabe sowie **FEISC_0x21_ReadBuffer, FEISC_0x22_ReadBuffer, FEISC_0x33_InitBuffer, FEISC_0x32_ClearDataBuffer**
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.36. FEISC_0x32_ClearDataBuffer

Funktion	Funktion löscht ausgelesene Einträge aus der internen Datentabelle
Syntax	**int FEISC_0x32_ClearDataBuffer(int iReaderHnd, UCHAR cBusAdr)**
Beschreibung	Die Funktion löscht die mit einem **FEISC_0x21_ReadBuffer** oder **FEISC_0x22_ReadBuffer** ausgelesenen Einträge aus der Reader-internen Datentabelle. *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im multijob-Leser eingestellte Busadresse.
Querverweis	**FEISC_0x21_ReadBuffer, FEISC_0x22_ReadBuffer, FEISC_0x33_InitBuffer, FEISC_0x31_ReadDataBufferInfo**
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.37. FEISC_0x33_InitBuffer

Funktion	Funktion zum Initialisieren der Leser-internen Datentabelle
Syntax	**int FEISC_0x33_InitBuffer(int iReaderHnd, UCHAR cBusAdr)**
Beschreibung	Die Funktion initialisiert die interne Datentabelle für den Buffered Read Mode. *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im Leser eingestellte Busadresse.
Querverweis	**FEISC_0x21_ReadBuffer**, **FEISC_0x22_ReadBuffer**, **FEISC_0x31_ReadDataBufferInfo**
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.38. FEISC_0x34_ForceNotifyTrigger

Funktion	Funktion zum Auslösen einer Notification
Syntax	**int FEISC_0x34_ForceNotifyTrigger(int iReaderHnd, UCHAR cBusAdr, UCHAR ucMode)**
Beschreibung	Die Funktion löst eine Notification aus, die Daten aus der internen Datentabelle für den Buffered Read Mode an den Host überträgt. Die Funktion kehrt nach der Ausführung sofort und noch vor dem Versenden der Notification zurück. Diese Funktion kann nur genutzt werden, wenn mit **FEISC_StartAsyncTask** ein Hintergrund-Task für den Empfang von Notifications gestartet wurde, sofern diese Bibliothek für die Verarbeitung der Notifications verwendet werden soll. Der Parameter *ucMode* ist z.Zt. ungenutzt und sollte 0x00 enthalten. *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im Leser eingestellte Busadresse.
Querverweis	5.4. Asynchrone Tasks zur Entlastung von Applikationen 5.7.12. FEISC_StartAsyncTask
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.39. FEISC_0x52_GetBaud

Funktion	Test-Funktion zum Ermitteln der Baudrate und Parität.
Syntax	**int FEISC_0x52_GetBaud(int iReaderHnd, UCHAR cBusAdr)**
Beschreibung	Kann das Antworttelegramm empfangen werden, sind die vorgegebene Baudrate und Parität identisch mit der des Lesers. *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im Leser eingestellte Busadresse.
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.40. FEISC_0x55_StartFlashLoader

Funktion	Die Funktion startet den Flash-Loader.
Syntax	**int FEISC_0x55_StartFlashLoader(int iReaderHnd)**
Beschreibung	Die Funktion startet den Flash-Loader des Lesers. Der Leser muß die Busadresse 0 haben. *iReaderHnd* ist der Handle zum Leser-Objekt.
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.41. FEISC_0x55_StartFlashLoaderEx

Funktion	Die Funktion startet den Flash-Loader.
Syntax	**int FEISC_0x55_StartFlashLoader(int iReaderHnd, UCHAR cBusAdr)**
Beschreibung	Die Funktion startet den Flash-Loader des Lesers. Der Leser kann eine beliebige Busadresse haben. *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im Leser eingestellte Busadresse.
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.42. FEISC_0x63_CPUReset

Funktion	Funktion löst einen Reset in der CPU des Lesers aus.
Syntax	**int FEISC_0x63_CPUReset(int iReaderHnd, UCHAR cBusAdr)**
Beschreibung	Funktion löst einen Reset in der CPU des Lesers aus. *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im Leser eingestellte Busadresse.
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.43. FEISC_0x64_SystemReset

Funktion	Funktion löst einen Reset in einem Teil des Lesers aus.
Syntax	**int FEISC_0x64_SystemReset(int iReaderHnd, UCHAR cBusAdr, UCHAR cMode)**
Beschreibung	Funktion löst einen Reset in einem durch cMode bestimmten Teil des Lesers aus. *cMode* bestimmt den Controller, der einen Reset ausführen soll. *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im Leser eingestellte Busadresse.
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.44. FEISC_0x65_SoftVersion

Funktion	Funktion liest die Versionsnummer des Lesers aus.
Syntax	**int FEISC_0x65_SoftVersion(int iReaderHnd, UCHAR cBusAdr, UCHAR* cVersion, int iDataFormat)**
Beschreibung	Die Versionsnummer des Lesers wird ermittelt und in *cVersion* hinterlegt. Der Parameter *iDataFormat* bestimmt, ob die Versionsnummer in *cVersion* als Hex-Array oder als Zeichenkette zu interpretieren ist. Der Puffer für die Version muß mindestens 8 Byte (*iDataFormat*=0) bzw. 15 Byte (*iDataFormat*=1) aufnehmen können. Dabei ist 1 Byte für das NUL-Zeichen vorgesehen. *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im Leser eingestellte Busadresse.
Querverweis	Weitere Informationen zu *iDataFormat* in Kapitel 5.3. Parameterübergabe
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.45. FEISC_0x66_ReaderInfo

Funktion	Funktion liest Informationen zum Lesers aus.
Syntax	**int FEISC_0x66_ReaderInfo(int iReaderHnd, UCHAR cBusAdr, UCHAR cMode, UCHAR* cInfo, int iDataFormat)**
Beschreibung	Die Information des mit *cMode* adressierten Teil des Lesers wird ermittelt und in *cInfo* hinterlegt. Der Parameter *iDataFormat* bestimmt, ob die Information in *cInfo* als Hex-Array oder als Zeichenkette zu interpretieren ist. Für die Pufferdimensionierung ist das Systemhandbuch des Lesers heranzuziehen, wobei ein zusätzliches Zeichen für das NUL-Zeichen hinzukommt. Für *iDataFormat*=1 muß die Größe des Puffers verdoppelt werden. *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im Leser eingestellte Busadresse.
Querverweis	Weitere Informationen zu *iDataFormat* in Kapitel 5.3. Parameterübergabe
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.46. FEISC_0x69_RFReset

Funktion	Funktion löst einen Reset für das Antennenfeld des Lesers aus.
Syntax	**int FEISC_0x69_RFReset(int ReaderHnd, UCHAR cBusAdr)**
Beschreibung	Funktion löst einen Reset für das Antennenfeld des Lesers aus. Alle zuvor mit der Funktion **FEISC_0x1A_Halt** abgeschalteten Transponder werden wieder aktiviert. *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im Leser eingestellte Busadresse.
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.47. FEISC_0x6A_RFOnOff

Funktion	Funktion zum Ein-/Ausschalten des Antennenfeldes.
Syntax	**int FEISC_0x6A_RFOnOff(int iReaderHnd, UCHAR cBusAdr, UCHAR cRF)**
Beschreibung	Eine 0 in *cRF* schaltet das Antennenfeld aus. Eine 1 in *cRF* schaltet das Antennenfeld ein. *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im Leser eingestellte Busadresse.
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.48. FEISC_0x6B_CentralizedRFSync

Funktion	Funktion zur Synchronisation von Antennenfeldern.
Syntax	**int FEISC_0x6B_CentralizedRFSync (int iReaderHnd, UCHAR cBusAdr, UCHAR cMode, UCHAR cTxChannel, int iTxPeriod, UCHAR cRes1, UCHAR cRes2)**
Beschreibung	Die Parameter sind im Systemhandbuch zum Leser dokumentiert. *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im Leser eingestellte Busadresse.
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.49. FEISC_0x6C_SetNoiseLevel

Funktion	Funktion zum Setzen der Noise-Level.
Syntax	**int FEISC_0x6C_SetNoiseLevel(int iReaderHnd, UCHAR cBusAdr, UCHAR* cLevel, int iDataFormat)**
Beschreibung	*cLevel* enthält die 3 Level-Werte, die als Hex-Array mit zusammen 6 Bytes (iDataFormat=0) oder als eine Zeichenkette mit zusammen 12 Bytes (iDataFormat=1) übergeben werden. *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im Leser eingestellte Busadresse.
Querverweis	Weitere Informationen zu *iDataFormat* in Kapitel 5.3. Parameterübergabe
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.50. FEISC_0x6D_GetNoiseLevel

Funktion	Funktion zum Lesen der Noise-Level.
Syntax	**int FEISC_0x6D_GetNoiseLevel(int iReaderHnd, UCHAR cBusAdr, UCHAR* cLevel, int iDataFormat)**
Beschreibung	In *cLevel* werden die ausgelesenen 3 Level-Werte hinterlegt. Der Puffer für *cLevel* muß folgend dimensioniert sein: 1. iDataFormat=0: 7 Byte (incl. NUL-Zeichen) *2.* iDataFormat=1: 13 Byte (incl. NUL-Zeichen) *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im Leser eingestellte Busadresse.
Querverweis	Weitere Informationen zu *iDataFormat* in Kapitel 5.3. Parameterübergabe
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.51. FEISC_0x6E_RdDiag

Funktion	Funktion zur Reader Diagnose.
Syntax	**int FEISC_0x6E_RdDiag(int iReaderHnd, UCHAR cBusAdr, UCHAR cMode, UCHAR* cData)**
Beschreibung	Die Funktion gibt für die in *cMode* hinterlegte Kennung Diagnosewerte zurück. Der Puffer für die Empfangsdaten *cData* muß ausreichend dimensioniert werden. *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im Leser eingestellte Busadresse.
Querverweis	Weitere Informationen zu *iDataFormat* in Kapitel 5.3. Parameterübergabe
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.52. FEISC_0x6F_AntennaTuning

Funktion	Funktion schaltet Leser in Spezialmodus.
Syntax	**int FEISC_0x6F_AntennaTuning(int ReaderHnd, UCHAR cBusAdr)**
Beschreibung	Funktion schaltet Leser in einen speziellen Tuning Modus. Der Leser verläßt diesen Modus nur mit einem CPU-Reset. *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im Leser eingestellte Busadresse.
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.53. FEISC_0x71_SetOutput

Funktion	Funktion aktiviert die Ausgänge des Lesers.
Syntax	**int FEISC_0x71_SetOutput(int iReaderHnd, UCHAR cBusAdr, int iOS, int iOSF, int iOSTime, int iOutTime)**
Beschreibung	Die Funktion aktiviert die Ausgänge des Lesers. Alle Zeiten werden intern im Leser mit 100 multipliziert und sind in der Einheit ms zu interpretieren. Es gelten die Wertebereiche aus dem Systemhandbuch zur ISC-Leserfamilie. *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im Leser eingestellte Busadresse.
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.54. FEISC_0x72_SetOutput

Funktion	Funktion aktiviert die Ausgänge des Lesers.
Syntax	**int FEISC_0x72_SetOutput(int iReaderHnd, UCHAR cBusAdr, UCHAR cMode, UCHAR cOutN, UCHAR* pRecords)**
Beschreibung	Die Funktion aktiviert die in *cOutN* übergebene Anzahl Ausgänge des Lesers. Die Zustände eines jeden Ausgangs sind in *pRecords* zusammengefaßt. *pRecords* wird entsprechend der Dokumentation im Systemhandbuch zum Leser zusammengestellt. Der Parameter *cMode* enthält das Mode-Byte. *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im Leser eingestellte Busadresse.
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.55. FEISC_0x74_ReadInput

Funktion	Funktion liest den Staus der digitalen Eingänge.
Syntax	**int FEISC_0x74_ReadInput(int iReaderHnd, UCHAR cBusAdr, UCHAR* cInput)**
Beschreibung	Die Funktion liest die digitalen Eingänge und hinterlegt den Status in *cInput*. Die Länge von *cInput* ist 1. *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im Leser eingestellte Busadresse.
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.56. FEISC_0x75_AdjAntenna

Funktion	Funktion zum Lesen des Antennen-Wertes.
Syntax	**int FEISC_0x75_AdjAntenna(int iReaderHnd, UCHAR cBusAdr, UCHAR* cValue, int iDataFormat)**
Beschreibung	In *cValue* wird der ausgelesene Antennen-Wert hinterlegt. Der Puffer für *cValue* muß folgend dimensioniert sein: 3. iDataFormat=0: 3 Byte (incl. NUL-Zeichen) *4.* iDataFormat=1: 5 Byte (incl. NUL-Zeichen) *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im Leser eingestellte Busadresse.
Querverweis	Weitere Informationen zu *iDataFormat* in Kapitel 5.3. Parameterübergabe
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.57. FEISC_0x76_CheckAntennas

Funktion	Funktion zum Detektieren von Antennen.
Syntax	**int FEISC_0x76_CheckAntennas(int iReaderHnd, UCHAR cBusAdr, UCHAR cMode, UCHAR* cAntOut, int* iAntOutLen)**
Beschreibung	In *cMode* ist für zukünftige Optionen. In *cAntOut* werden Flagfelder für detektierte Antennen zurückgegeben. *iAntOutLen* gibt an, wie viel Byte in *cAntOut* enthalten sind. Es sind maximal 5 Byte möglich. Der Puffer für *cAntOut* muss deshalb für die Aufnahme von 5 Byte dimensioniert sein. *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im Leser eingestellte Busadresse.
Querverweis	Weitere Informationen zu *iDataFormat* in Kapitel 5.3. Parameterübergabe
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.58. FEISC_0x80_ReadConfBlock

Funktion	Funktion liest einen Konfigurationsblock aus dem Leser.
Syntax	**int FEISC_0x80_ReadConfBlock(int iReaderHnd, UCHAR cBusAdr, UCHAR cConfAdr, UCHAR* cConfBlock, int iDataFormat)**
Beschreibung	Mit dieser Funktion kann man einen Konfigurationsblock aus der Adresse *cConfAdr* des Lesers auslesen. Die ausgelesenen Daten in *cConfBlock* sind als Hex-Array (*iDataFormat*=0) oder als Zeichenkette (*iDataFormat*=1) zu interpretieren. Der Puffer für die Konfigurationsdaten *cConfBlock* muß folgend dimensioniert sein: 1. *iDataFormat*=0: 15 Bytes (incl. 1 NUL-Zeichen) 2. *iDataFormat*=1: 29 Bytes (incl. 1 NUL-Zeichen) *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im Leser eingestellte Busadresse.
Querverweis	Weitere Informationen zu *iDataFormat* in Kapitel 5.3. Parameterübergabe
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.59. FEISC_0x81_WriteConfBlock

Funktion	Funktion schreibt einen Konfigurationsblock in den Leser.
Syntax	**int FEISC_0x81_WriteConfBlock(int iReaderHnd, UCHAR cBusAdr, UCHAR cConfAdr, UCHAR* cConfBlock, int iDataFormat)**
Beschreibung	Mit dieser Funktion kann man einen Konfigurationsblock in die Adresse *cConfAdr* des Lesers schreiben. Die Konfigurationsdaten müssen als Hex-Array (*iDataFormat*=0) oder Zeichenkette (*iDataFormat*=1) in *cConfBlock* hinterlegt werden. Der Puffer mit den Konfigurationsdaten muß 14 Byte (*iDataFormat*=0) bzw. 28 Byte (*iDataFormat*=1) enthalten. *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im Leser eingestellte Busadresse.
Querverweis	Weitere Informationen zu *iDataFormat* in Kapitel 5.3. Parameterübergabe
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.60. FEISC_0x82_SaveConfBlock

Funktion	Funktion sichert einen Konfigurationsblock im Leser.
Syntax	**int FEISC_0x82_SaveConfBlock(int iReaderHnd, UCHAR cBusAdr, UCHAR cConfAdr)**
Beschreibung	Mit dieser Funktion kann man einen Konfigurationsblock der Adresse *cConfAdr* vom RAM-Speicher in den EEPROM-Speicher schreiben und damit dauerhaft sichern. *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im Leser eingestellte Busadresse.
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.61. FEISC_0x83_ResetConfBlock

Funktion	Funktion lädt Werkseinstellung in einen Konfigurationsblock im Leser.
Syntax	**int FEISC_0x83_ResetConfBlock(int iReaderHnd, UCHAR cBusAdr, UCHAR cConfAdr)**
Beschreibung	Mit dieser Funktion kann man in einen Konfigurationsblock der Adresse *cConfAdr* die Parameter der Werkseinstellung laden. *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im Leser eingestellte Busadresse.
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.62. FEISC_0x85_SetSysTimer

Funktion	Setzt die Systemzeit im Leser.
Syntax	**int FEISC_0x85_SetSysTimer(int iReaderHnd, UCHAR cBusAdr, UCHAR* cTime, int iDataFormat)**
Beschreibung	Die Funktion initialisiert die Systemzeit im Leser. Der Puffer *cTime* muß 4 Bytes (*iDataFormat*=0) enthalten oder eine Zeichenkette mit 8 Zeichen (*iDataFormat*=1) sein. *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im Leser eingestellte Busadresse.
Querverweis	Weitere Informationen zu *iDataFormat* in Kapitel 5.3. Parameterübergabe
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.63. FEISC_0x86_GetSysTimer

Funktion	Liest die Systemzeit aus dem Leser
Syntax	**int FEISC_0x86_GetSysTimer(int iReaderHnd, UCHAR cBusAdr, UCHAR* cTime, int iDataFormat)**
Beschreibung	Diese Funktion ermittelt die Systemzeit des Lesers. Der Puffer für *cTime* muß wie folgt dimensioniert sein: 5. *iDataFormat*=0: 5 Zeichen (incl. 1 NUL-Zeichen) *6.* *iDataFormat*=1: 9 Zeichen (incl. 1 NUL-Zeichen) *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im Leser eingestellte Busadresse.
Querverweis	Weitere Informationen zu *iDataFormat* in Kapitel 5.3. Parameterübergabe
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.64. FEISC_0x87_SetSystemDate

Funktion	Setzt Systemdatum und -zeit im Leser.
Syntax	**int FEISC_0x87_SetSystemDate(int iReaderHnd, UCHAR cBusAdr, UCHAR cCentury, UCHAR cYear, UCHAR cMonth, UCHAR cDay, UCHAR cTimezone, UCHAR cHour, UCHAR cMinute, int iMilliSecond)**
Beschreibung	Die Funktion initialisiert Systemdatum und -zeit im Leser. Die Parameter haben folgende Bedeutung: *cCentury* : Jahrhundert (z.B. 20) *cYear* : Jahreszahl (z.B. 4) *cMonth* : Monat (z.B. 10) *cDay* : Tag (z.B. 5) *cTimezone* : Zeitzone (z.Zt. ungenutzt) *cHour* : Stunde (z.B. 15) *cMinute* : Minuten (z.B. 13) *iMilliSecond* : Millisekunden, enthält auch die Sekunden (z.B. 1234 für 1s und 234ms) *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im Leser eingestellte Busadresse.
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.65. FEISC_0x88_GetSystemDate

Funktion	Liest Systemdatum und -zeit aus dem Leser
Syntax	**int FEISC_0x88_GetSystemDate(int iReaderHnd, UCHAR cBusAdr, UCHAR* cCentury, UCHAR* cYear, UCHAR* cMonth, UCHAR* cDay, UCHAR* cTimezone, UCHAR* cHour, UCHAR* cMinute, int* iMilliSecond)**
Beschreibung	Diese Funktion ermittelt Systemdatum und -zeit des Lesers. Die Bedeutung der Parameter sind in 5.7.63. FEISC_0x87_SetSystemDate erläutert. *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im Leser eingestellte Busadresse.
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.66. FEISC_0x8A_ReadConfiguration

Funktion	Funktion liest Konfigurationsblöcke aus dem Leser.
Syntax	**int FEISC_0x8A_ReadConfiguration(int iReaderHnd, UCHAR cBusAdr, UCHAR cDevice, UCHAR cBank, UCHAR cMode, int iReqBlockAdr, UCHAR cReqBlockCount, UCHAR* cRspBlockCount, UCHAR* cRspBlockSize, UCHAR* cRspData)**
Beschreibung	Mit dieser Funktion kann man einen Konfigurationsblock, mehrere oder alle Konfigurationsblöcke ab der Adresse *cReqBlockAdr* des Lesers auslesen. Die ausgelesenen Daten sind in *cRspData* in aufsteigender Adress-Reihenfolge hinterlegt. Der Parameter *cDevice* benennt den Controller im Leser, *cBank* den Konfigurationsspeicher und *cMode* zusätzliche Optionen. Weitere Informationen finden sich im Systemhandbuch des Lesers. Der Puffer für die Konfigurationsdaten *cRspData* muss so dimensioniert sein, dass cReqBlockCount x cRspBlockSize Bytes gespeichert werden können. *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im Leser eingestellte Busadresse.
Querverweis	
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.67. FEISC_0x8B_WriteConfiguration

Funktion	Funktion schreibt Konfigurationsblöcke in den Leser.
Syntax	**int FEISC_0x8B_WriteConfiguration(int iReaderHnd, UCHAR cBusAdr, UCHAR cDevice, UCHAR cBank, UCHAR cMode, UCHAR cReqBlockCount, UCHAR cReqBlockSize, UCHAR* cReqData)**
Beschreibung	Mit dieser Funktion kann man einen Konfigurationsblock, mehrere oder alle Konfigurationsblöcke in den Leser schreiben. Die Konfigurationsdaten in *cReqData* sind in aufsteigender Adress-Reihenfolge abzulegen und enthalten vorangestellt jeweils die Konfigurationsadresse. Der Parameter *cDevice* benennt den Controller im Leser, *cBank* den Konfigurationsspeicher und *cMode* zusätzliche Optionen. Weitere Informationen finden sich im Systemhandbuch des Lesers. *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im Leser eingestellte Busadresse.
Querverweis	
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.68. FEISC_0x8C_ResetConfiguration

Funktion	Funktion lädt Werkseinstellung in Konfigurationsblöcke im Leser.
Syntax	**int FEISC_0x8C_ResetConfiguration(int iReaderHnd, UCHAR cBusAdr, UCHAR cDevice, UCHAR cBank, UCHAR cMode, int iReqBlockAdr, UCHAR cReqBlockCount)**
Beschreibung	Mit dieser Funktion kann man einen Konfigurationsblock, mehrere oder alle Konfigurationsblöcke ab der Adresse *cReqBlockAdr* im Leser in die Werkseinstellung zurücksetzen. Der Parameter *cDevice* benennt den Controller im Leser, *cBank* den Konfigurationsspeicher und *cMode* zusätzliche Optionen. Weitere Informationen finden sich im Systemhandbuch des Lesers. *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im Leser eingestellte Busadresse.
Querverweis	
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.69. FEISC_0x9F_Piggyback_Command

Funktion	Transport-Funktion für ein Protokoll an eine externe Funktionseinheit.
Syntax	**int FEISC_0x9F_Piggyback_Command(int iReaderHnd, UCHAR cBusAdr, UCHAR cMode, UCHAR cDevice, UCHAR cPort, UCHAR* cReqPrt, int iReqLen, UCHAR* cRspPrt, int* iRspLen)**
Beschreibung	Diese Funktion transportiert das in *cReqPrt* enhaltene Protokoll an einen Leser, der es an eine angeschlossene externe Funktionseinheit (z. B. People Counter ID ISC.ANTGPC) weiterleitet. Zum Aufbau des eingebetteten Protokolls kann man die Funktion FEISC_BuildSendProtocol verwenden. Der Parameter *cDevice* benennt den Typ der externen Funktionseinheit, *cPort* den Kommunikationsport im Leser und *cMode* zusätzliche Optionen. Weitere Informationen finden sich im Systemhandbuch zur externen Funktionseinheit. Der Puffer für das Empfangsprotokoll *cRspPrt* muss ausreichend dimensioniert sein. Das Antwortprotokoll kann anschließend mit der Funktion FEISC_SplitRecProtocol analysiert und zerlegt werden. *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im Leser eingestellte Busadresse.
Querverweis	5.7.15. FEISC_BuildSendProtocol 5.7.18. FEISC_SplitRecProtocol
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.70. FEISC_0xA0_RdLogin

Funktion	Funktion führt einen Login im Leser aus.
Syntax	**int FEISC_0xA0_RdLogin(int iReaderHnd, UCHAR cBusAdr, UCHAR* cRd_PW, int iDataFormat)**
Beschreibung	Die Funktion führt mit dem Paßwort *cRd_PW* einen Leser-Login durch. Der Parameter *iDataFormat* bestimmt, ob das Paßwort in *cRd_PW* als Hex-Array (*iDataFormat*=0) oder als Zeichenkette (*iDataFormat*=1) zu interpretieren ist. *iReaderHnd* ist der Handle zum Leser-Object. *cBusAdr* ist die im Leser eingestellte Busadresse.
Querverweis	Weitere Informationen zu *iDataFormat* in Kapitel 5.3. Parameterübergabe
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.71. FEISC_0xA2_WriteMifareKeys

Funktion	Funktion schreibt Authentifikationsschlüssel in den Leser.
Syntax	**int FEISC_0xA2_WriteMifareKeys(int iReaderHnd, UCHAR cBusAdr, UCHAR cType, UCHAR cAdr, UCHAR* cKey, int iDataFormat)**
Hinweis	Der Authentifikationsschlüssel kann nicht zurückgelesen werden. Deshalb muß mit dieser Funktion sehr sorgfältig umgegangen werden!
Beschreibung	Die Funktion schreibt Authentifikationsschlüssel für Mifare-Transponder in den Leser. *cType* definiert den Schlüsseltyp, *cAdr* spezifiziert die EEPROM-Adresse des Keys im Leser. Der Parameter *iDataFormat* bestimmt, ob der Schlüssel in *cKey* als Hex-Array (*iDataFormat*=0) oder als Zeichenkette (*iDataFormat*=1) zu interpretieren ist. *iReaderHnd* ist der Handle zum Leser-Object. *cBusAdr* ist die im Leser eingestellte Busadresse.
Querverweis	Weitere Informationen zu *iDataFormat* in Kapitel 5.3. Parameterübergabe
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.72. FEISC_0xA3_Write_DES_AES_Keys

Funktion	Funktion schreibt Authentifikationsschlüssel in den Leser.
Syntax	**int FEISC_0xA3_Write_DES_AES_Keys(int iReaderHnd, UCHAR cBusAdr, UCHAR cMode, UCHAR cReaderKeyIndex, UCHAR cAuthentMode, UCHAR cKeyLen, UCHAR* cKey, int iDataFormat)**
Hinweis	Der Authentifikationsschlüssel kann nicht zurückgelesen werden. Deshalb muß mit dieser Funktion sehr sorgfältig umgegangen werden!
Beschreibung	Die Funktion schreibt Authentifikationsschlüssel für Mifare-Transponder in den Leser. Alle Parameter sind im Detail im Systemhandbuch des jeweiligen Lesers erklärt. Der Parameter *iDataFormat* bestimmt, ob der Schlüssel in *cKey* als Hex-Array (*iDataFormat*=0) oder als Zeichenkette (*iDataFormat*=1) zu interpretieren ist. *iReaderHnd* ist der Handle zum Leser-Object. *cBusAdr* ist die im Leser eingestellte Busadresse.
Querverweis	Weitere Informationen zu *iDataFormat* in Kapitel 5.3. Parameterübergabe
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.73. FEISC_0xAD_WriteReaderAuthentKey

Funktion	Funktion schreibt Authentifikationsschlüssel in den Leser.
Syntax	**int FEISC_0xAD_WriteReaderAuthentKey(int iReaderHnd, UCHAR cBusAdr, UCHAR cMode, UCHAR cKeyType, UCHAR cKeyLen, UCHAR* cKey, int iDataFormat)**
Hinweis	Der Authentifikationsschlüssel kann nicht zurückgelesen werden. Deshalb muß mit dieser Funktion sehr sorgfältig umgegangen werden!
Beschreibung	Die Funktion schreibt Authentifikationsschlüssel für den Aufbau einer verschlüsselten Datenübertragung in den Leser. Alle Parameter sind im Detail im Systemhandbuch des jeweiligen Lesers erklärt. Der Parameter *iDataFormat* bestimmt, ob der Schlüssel in *cKey* als Hex-Array (*iDataFormat*=0) oder als Zeichenkette (*iDataFormat*=1) zu interpretieren ist. *iReaderHnd* ist der Handle zum Leser-Object. *cBusAdr* ist die im Leser eingestellte Busadresse.
Querverweis	Grundlagen zur Datenverschlüsselung in 5.6. Sicherheit in der Datenübertragung. Weitere Informationen zu *iDataFormat* in Kapitel 5.3. Parameterübergabe
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.74. FEISC_0xAE_ReaderAuthent

Funktion	Authentifikationsfunktion
Syntax	**int FEISC_0xAE_ReaderAuthent(int iReaderHnd, UCHAR cBusAdr, UCHAR cMode, UCHAR cKeyType, UCHAR cKeyLen, UCHAR* cKey, int iDataFormat)**
Beschreibung	Authentifikationsfunktion zur Einleitung einer verschlüsselten Datenübertragung. Alle Parameter sind im Detail im Systemhandbuch des jeweiligen Lesers erklärt. Der Parameter *iDataFormat* bestimmt, ob der Schlüssel in *cKey* als Hex-Array (*iDataFormat*=0) oder als Zeichenkette (*iDataFormat*=1) zu interpretieren ist. *iReaderHnd* ist der Handle zum Leser-Object. *cBusAdr* ist die im Leser eingestellte Busadresse.
Querverweis	Grundlagen zur Datenverschlüsselung in 5.6. Sicherheit in der Datenübertragung. Weitere Informationen zu *iDataFormat* in Kapitel 5.3. Parameterübergabe
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.75. FEISC_0xB0_ISOCmd

Funktion	Funktion initiiert Datentransfer mit ISO15693 oder ISO14443-Transpondern
Syntax	**int FEISC_0xB0_ISOCmd(int iReaderHnd, UCHAR cBusAdr, UCHAR* cReqData, int iReqLen, UCHAR* cRspData, int* iRspLen, int iDataFormat)**
Beschreibung	Die Funktion initiiert einen Datentransfer für mehrere im Lesefeld des Lesers befindliche Transponder. Die für den Datentransfer notwendigen Daten sind in *cReqData* zu hinterlegen. In *iReqLen* muß die Anzahl der in *cReqData* enthaltenen Zeichen angegeben werden. Die vom Transponder gelesenen Daten sind in *cRspData* enthalten. *iRspLen* gibt die Anzahl der Zeichen in *cRspData* an. Der Parameter *iDataFormat* bestimmt, ob *cReqData* und *cRspData* als Hex-Array oder als Zeichenkette zu interpretieren sind. Dabei ist folgendes zu beachten: der Puffer für die Empfangsdaten *cRspData* muß so dimensioniert werden, dass alle Empfangsdaten gespeichert werden können. Das bedeutet im Fall *iDataFormat*=1, dass die Größe des Puffers *cRspData* doppelt so groß ist, als im Fall *iDataFormat*=0. Im Parameter *iReqLen* muß die Länge der Sendedaten (Anzahl Zeichen in *cReqData*) angegeben werden. Wenn *iDataFormat*=1 ist, dann ist *iReqLen* also genau doppelt so groß, als im Fall *iDataFormat*=0. Analog dazu ist die Längenangabe für den Empfangspuffer (*iRspLen*) auszuwerten. *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im Leser eingestellte Busadresse.
Querverweis	Weitere Informationen zu *iDataFormat* in Kapitel 5.3. Parameterübergabe
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls.

5.7.76. FEISC_0xB1_ ISOCustAndPropCmd

Funktion	Funktion initiiert Datentransfer mit einem ISO15693-Transponder
Syntax	**int FEISC_0xB1_ISOCustAndPropCmd(int iReaderHnd, UCHAR cBusAdr, UCHAR cMfr, UCHAR* cReqData, int iReqLen, UCHAR* cRspData, int* iRspLen, int iDataFormat)**
Beschreibung	Die Funktion initiiert einen Datentransfer für mehrere im Lesefeld des ISC-Lesers befindliche ISO15693-Transponder Der Parameter *cMfr* enthält den Manufacturer Code und bestimmt die Strukturen der Sendedaten *cReqData* und Empfangsdaten *cRspData*. Die für den Datentransfer notwendigen Daten sind in *cReqData* zu hinterlegen. In *iReqLen* muß die Anzahl der in *cReqData* enthaltenen Zeichen angegeben werden. Die vom ISO15693-Transponder gelesenen Daten sind in *cRspData* enthalten. *iRspLen* gibt die Anzahl der Zeichen in *cRspData* an. Der Parameter *iDataFormat* bestimmt, ob *cReqData* und *cRspData* als Hex-Array oder als Zeichenkette zu interpretieren sind. Dabei ist folgendes zu beachten: der Puffer für die Empfangsdaten *cRspData* muß so dimensioniert werden, dass alle Empfangsdaten gespeichert werden können. Das bedeutet im Fall *iDataFormat*=1, dass die Größe des Puffers *cRspData* doppelt so groß ist, als im Fall *iDataFormat*=0. Im Parameter *iReqLen* muß die Länge der Sendedaten (Anzahl Zeichen in *cReqData*) angegeben werden. Wenn *iDataFormat*=1 ist, dann ist *iReqLen* also genau doppelt so groß, als im Fall *iDataFormat*=0. Analog dazu ist die Längenangabe für den Empfangspuffer (*iRspLen*) auszuwerten.*iReaderHnd* ist der Handle zum Leser-Objekt. *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im Leser eingestellte Busadresse.
Querverweis	Weitere Informationen zu *iDataFormat* in Kapitel 5.3. Parameterübergabe
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls.

5.7.77. FEISC_0xB2_ISOCmd

Funktion	Funktion initiiert Datentransfer mit einem ISO14443-Transponder
Syntax	**int FEISC_0xB2_ISOCmd(int iReaderHnd, UCHAR cBusAdr, UCHAR* cReqData, int iReqLen, UCHAR* cRspData, int* iRspLen, int iDataFormat)**
Beschreibung	Die Funktion initiiert einen Datentransfer für mehrere im Lesefeld des ISC-Lesers befindliche ISO14443-Transponder Die für den Datentransfer notwendigen Daten sind in *cReqData* zu hinterlegen. In *iReqLen* muß die Anzahl der in *cReqData* enthaltenen Zeichen angegeben werden. Die vom ISO14443-Transponder gelesenen Daten sind in *cRspData* enthalten. *iRspLen* gibt die Anzahl der Zeichen in *cRspData* an. Der Parameter *iDataFormat* bestimmt, ob *cReqData* und *cRspData* als Hex-Array oder als Zeichenkette zu interpretieren sind. Dabei ist folgendes zu beachten: der Puffer für die Empfangsdaten *cRspData* muß so dimensioniert werden, dass alle Empfangsdaten gespeichert werden können. Das bedeutet im Fall *iDataFormat*=1, dass die Größe des Puffers *cRspData* doppelt so groß ist, als im Fall *iDataFormat*=0. Im Parameter *iReqLen* muß die Länge der Sendedaten (Anzahl Zeichen in *cReqData*) angegeben werden. Wenn *iDataFormat*=1 ist, dann ist *iReqLen* also genau doppelt so groß, als im Fall *iDataFormat*=0. Analog dazu ist die Längenangabe für den Empfangspuffer (*iRspLen*) auszuwerten. *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im Leser eingestellte Busadresse.
Querverweis	Weitere Informationen zu *iDataFormat* in Kapitel 5.3. Parameterübergabe
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls.

5.7.78. FEISC_0xB3_EPCCmd

Funktion	Funktion initiiert Datentransfer mit UHF EPC-Transponder
Syntax	**int FEISC_0xB3_EPCCmd(int iReaderHnd, UCHAR cBusAdr, UCHAR* cReqData, int iReqLen, UCHAR* cRspData, int* iRspLen, int iDataFormat)**
Beschreibung	Die Funktion initiiert einen Datentransfer für einen im Lesefeld des Lesers befindliche UHF EPC-Transponder Die für den Datentransfer notwendigen Daten sind in *cReqData* zu hinterlegen. In *iReqLen* muß die Anzahl der in *cReqData* enthaltenen Zeichen angegeben werden. Die vom Transponder gelesenen Daten sind in *cRspData* enthalten. *iRspLen* gibt die Anzahl der Zeichen in *cRspData* an. Der Parameter *iDataFormat* bestimmt, ob *cReqData* und *cRspData* als Hex-Array oder als Zeichenkette zu interpretieren sind. Dabei ist folgendes zu beachten: der Puffer für die Empfangsdaten *cRspData* muß so dimensioniert werden, dass alle Empfangsdaten gespeichert werden können. Das bedeutet im Fall *iDataFormat*=1, dass die Größe des Puffers *cRspData* doppelt so groß ist, als im Fall *iDataFormat*=0. Im Parameter *iReqLen* muß die Länge der Sendedaten (Anzahl Zeichen in *cReqData*) angegeben werden. Wenn *iDataFormat*=1 ist, dann ist *iReqLen* also genau doppelt so groß, als im Fall *iDataFormat*=0. Analog dazu ist die Längenangabe für den Empfangspuffer (*iRspLen*) auszuwerten. *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im Leser eingestellte Busadresse.
Querverweis	Weitere Informationen zu *iDataFormat* in Kapitel 5.3. Parameterübergabe
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls.

5.7.79. FEISC_0xB4_EPC_UHF_Cmd

Funktion	Funktion initiiert Datentransfer mit UHF EPC-Transponder
Syntax	**int FEISC_0xB4_EPC_UHF_Cmd(int iReaderHnd, UCHAR cBusAdr, UCHAR cMfr, UCHAR* cReqData, int iReqLen, UCHAR* cRspData, int* iRspLen, int iDataFormat)**
Beschreibung	Die Funktion initiiert einen Datentransfer für einen im Lesefeld des Lesers befindliche UHF EPC-Transponder Der Parameter *cMfr* enthält den Manufacturer Code und bestimmt die Strukturen der Sendedaten *cReqData* und Empfangsdaten *cRspData*. Die für den Datentransfer notwendigen Daten sind in *cReqData* zu hinterlegen. In *iReqLen* muß die Anzahl der in *cReqData* enthaltenen Zeichen angegeben werden. Die vom Transponder gelesenen Daten sind in *cRspData* enthalten. *iRspLen* gibt die Anzahl der Zeichen in *cRspData* an. Der Parameter *iDataFormat* bestimmt, ob *cReqData* und *cRspData* als Hex-Array oder als Zeichenkette zu interpretieren sind. Dabei ist folgendes zu beachten: der Puffer für die Empfangsdaten *cRspData* muß so dimensioniert werden, dass alle Empfangsdaten gespeichert werden können. Das bedeutet im Fall *iDataFormat*=1, dass die Größe des Puffers *cRspData* doppelt so groß ist, als im Fall *iDataFormat*=0. Im Parameter *iReqLen* muß die Länge der Sendedaten (Anzahl Zeichen in *cReqData*) angegeben werden. Wenn *iDataFormat*=1 ist, dann ist *iReqLen* also genau doppelt so groß, als im Fall *iDataFormat*=0. Analog dazu ist die Längenangabe für den Empfangspuffer (*iRspLen*) auszuwerten. *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im Leser eingestellte Busadresse.
Querverweis	Weitere Informationen zu *iDataFormat* in Kapitel 5.3. Parameterübergabe
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls.

5.7.80. FEISC_0xBB_C1G2_ TranspCmd

Funktion	Funktion initiiert Datentransfer mit einem Class 1 Gen 2 UHF-Transponder.
Syntax	**int FEISC_0xBB_C1G2_TranspCmd(int iReaderHnd, UCHAR cBusAdr, UCHAR ucMode, UCHAR ucTxPara, UCHAR ucRxPara, unsigned int uiTs, int iRspLength, UCHAR* cReqData, int iReqLen, UCHAR* cRspData, int* iRspLen)**
Beschreibung	Die Funktion initiiert einen Datentransfer für einen im Lesefeld des Lesers befindlichen Class 1 Generation 2 UHF-Transponder. Der Parameter *ucMode* enthält das Mode-Byte. Die Parameter *ucTxPara*, *ucRxPara* und *uiTs* steuern das zeitliche Verhalten der RF-Kommunikation. Der Parameter *iRspLength* enthält die erwartete Länge (Anzahl Bits) der Empfangsdaten *cRspData*. Die für den Datentransfer notwendigen Daten sind in *cReqData* zu hinterlegen. In *iReqLen* muß die Anzahl der in *cReqData* enthaltenen Zeichen angegeben werden. Die vom UHF-Transponder gelesenen Daten sind in *cRspData* enthalten. *iRspLen* gibt die Anzahl der Zeichen in *cRspData* an. Dabei ist folgendes zu beachten: der Puffer für die Empfangsdaten *cRspData* muß so dimensioniert werden, dass alle Empfangsdaten gespeichert werden können. *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im Leser eingestellte Busadresse.
Querverweis	
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls.

5.7.81. FEISC_0xBC_CmdQueue

Funktion	Ein Queue Command Task wird asynchron zur Applikation gestartet
Syntax	**int FEISC_0xBC_CmdQueue(int iReaderHnd, int iMode, int iCmdCount, UCHAR* ucCmdQueue, int iCmdQueueLen, FEISC_TASK_INIT* pInit)**
Beschreibung	Mit dieser Funktion wird der Queue Command als ein asynchroner Task gestartet. Ein asynchroner Task ist ein interner Thread, der das Queue Kommando an den Leser sendet und für eine Zeit Timeout auf die Antwort wartet. Die Signalisierung der Antwortdaten bzw. der Abbruchbedingung an die Applikation erfolgt mit dem Aufruf einer Callback-Funktion. Der Parameter *iMode* enthält einen Steuerungswert. *iCmdCount* enthält die Anzahl der Transponder-Kommandos in der Queue. Die für den Datentransfer notwendigen Daten sind in *ucCmdQueue* zu hinterlegen. In *iCmdQueueLen* muß die Anzahl der in *ucCmdQueue* enthaltenen Zeichen angegeben werden. Alle für den Task und für die Callback-Funktion relevanten Daten sind in der Struktur FEISC_TASK_INIT zusammengefasst. Diese Struktur wird im Kapitel 5.4. Asynchrone Tasks zur Entlastung von Applikationen genauer beschrieben. Folgende Verwendung wird empfohlen: `FEISC_TASK_INIT Init;` `Init.cbFct1 = this->cbsTaskRsp1;  // Callback-Funktion` `Init.ucBusAdr = 255;  // jeder Leser antwortet` `Init.uiFlag = FEISC_TASKCB_1;` `Init.uiTimeout = m_uiTimeout;  // individuelle Timeout` `Init.pAny = this;  // optional: This-Pointer` *iReaderHnd* ist der Handle zum Leser-Objekt.
Querverweis	Weitere Informationen zu asynchronen Tasks finden sich im Kapitel 5.4. Asynchrone Tasks zur Entlastung von Applikationen. 5.7.13. FEISC_CancelAsyncTask
Anmerkung	Nähere Informationen zum Protokoll [0xBC] Command Queue finden sich im Systemhandbuch zur OBID® *classic-pro* Leserfamilie.
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls.

5.7.82. FEISC_0xBD_ ISOTranspCmd

Funktion	Funktion initiiert Datentransfer mit einem ISO14443A-Transponder
Syntax	**int FEISC_0xBD_ISOTranspCmd(int iReaderHnd, UCHAR cBusAdr, int iMode, int iRspLength, UCHAR* cReqData, int iReqLen, UCHAR* cRspData, int* iRspLen, int iDataFormat)**
Beschreibung	Die Funktion initiiert einen Datentransfer für mehrere im Lesefeld des ISC-Lesers befindliche ISO14443A-Transponder Der Parameter *iMode* enthält den Modus für den Leser. Der Parameter *iRspLength* enthält die erwartete Länge (Anzahl Bits) der Empfangsdaten *cRspData*. Die für den Datentransfer notwendigen Daten sind in *cReqData* zu hinterlegen. In *iReqLen* muß die Anzahl der in *cReqData* enthaltenen Zeichen angegeben werden. Die vom ISO14443-Transponder gelesenen Daten sind in *cRspData* enthalten. *iRspLen* gibt die Anzahl der Zeichen in *cRspData* an. Der Parameter *iDataFormat* bestimmt, ob *cReqData* und *cRspData* als Hex-Array oder als Zeichenkette zu interpretieren sind. Dabei ist folgendes zu beachten: der Puffer für die Empfangsdaten *cRspData* muß so dimensioniert werden, dass alle Empfangsdaten gespeichert werden können. Das bedeutet im Fall *iDataFormat*=1, dass die Größe des Puffers *cRspData* doppelt so groß ist, als im Fall *iDataFormat*=0. Im Parameter *iReqLen* muß die Länge der Sendedaten (Anzahl Zeichen in *cReqData*) angegeben werden. Wenn *iDataFormat*=1 ist, dann ist *iReqLen* also genau doppelt so groß, als im Fall *iDataFormat*=0. Analog dazu ist die Längenangabe für den Empfangspuffer (*iRspLen*) auszuwerten. *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im Leser eingestellte Busadresse.
Querverweis	Weitere Informationen zu *iDataFormat* in Kapitel 5.3. Parameterübergabe
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls.

5.7.83. FEISC_0xBE_ ISOTranspCmd

Funktion	Funktion initiiert Datentransfer mit einem ISO14443B-Transponder
Syntax	**int FEISC_0xBE_ISOTranspCmd(int iReaderHnd, UCHAR cBusAdr, int iMode, int iRspLength, UCHAR* cReqData, int iReqLen, UCHAR* cRspData, int* iRspLen, int iDataFormat)**
Beschreibung	Die Funktion initiiert einen Datentransfer für mehrere im Lesefeld des ISC-Lesers befindliche ISO14443B-Transponder Der Parameter *iMode* enthält den Modus für den Leser. Der Parameter *iRspLength* enthält die erwartete Länge (Anzahl Bits) der Empfangsdaten *cRspData*. Die für den Datentransfer notwendigen Daten sind in *cReqData* zu hinterlegen. In *iReqLen* muß die Anzahl der in *cReqData* enthaltenen Zeichen angegeben werden. Die vom ISO14443B-Transponder gelesenen Daten sind in *cRspData* enthalten. *iRspLen* gibt die Anzahl der Zeichen in *cRspData* an. Der Parameter *iDataFormat* bestimmt, ob *cReqData* und *cRspData* als Hex-Array oder als Zeichenkette zu interpretieren sind. Dabei ist folgendes zu beachten: der Puffer für die Empfangsdaten *cRspData* muß so dimensioniert werden, dass alle Empfangsdaten gespeichert werden können. Das bedeutet im Fall *iDataFormat*=1, dass die Größe des Puffers *cRspData* doppelt so groß ist, als im Fall *iDataFormat*=0. Im Parameter *iReqLen* muß die Länge der Sendedaten (Anzahl Zeichen in *cReqData*) angegeben werden. Wenn *iDataFormat*=1 ist, dann ist *iReqLen* also genau doppelt so groß, als im Fall *iDataFormat*=0. Analog dazu ist die Längenangabe für den Empfangspuffer (*iRspLen*) auszuwerten. *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im Leser eingestellte Busadresse.
Querverweis	Weitere Informationen zu *iDataFormat* in Kapitel 5.3. Parameterübergabe
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls.

5.7.84. FEISC_0xBF_ ISOTranspCmd

Funktion	Funktion initiiert Datentransfer mit einem ISO15693-Transponder
Syntax	**int FEISC_0xBF_ISOTranspCmd(int iReaderHnd, UCHAR cBusAdr, int iMode, int iRspLength, UCHAR* cReqData, int iReqLen, UCHAR* cRspData, int* iRspLen, int iDataFormat)**
Beschreibung	Die Funktion initiiert einen Datentransfer für mehrere im Lesefeld des ISC-Lesers befindliche ISO15693-Transponder Der Parameter *iMode* enthält den Modus für den Leser. Der Parameter *iRspLength* enthält die erwartete Länge (Anzahl Bits) der Empfangsdaten *cRspData*. Die für den Datentransfer notwendigen Daten sind in *cReqData* zu hinterlegen. In *iReqLen* muß die Anzahl der in *cReqData* enthaltenen Zeichen angegeben werden. Die vom ISO15693-Transponder gelesenen Daten sind in *cRspData* enthalten. *iRspLen* gibt die Anzahl der Zeichen in *cRspData* an. Der Parameter *iDataFormat* bestimmt, ob *cReqData* und *cRspData* als Hex-Array oder als Zeichenkette zu interpretieren sind. Dabei ist folgendes zu beachten: der Puffer für die Empfangsdaten *cRspData* muß so dimensioniert werden, dass alle Empfangsdaten gespeichert werden können. Das bedeutet im Fall *iDataFormat*=1, dass die Größe des Puffers *cRspData* doppelt so groß ist, als im Fall *iDataFormat*=0. Im Parameter *iReqLen* muß die Länge der Sendedaten (Anzahl Zeichen in *cReqData*) angegeben werden. Wenn *iDataFormat*=1 ist, dann ist *iReqLen* also genau doppelt so groß, als im Fall *iDataFormat*=0. Analog dazu ist die Längenangabe für den Empfangspuffer (*iRspLen*) auszuwerten. *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im Leser eingestellte Busadresse.
Querverweis	Weitere Informationen zu *iDataFormat* in Kapitel 5.3. Parameterübergabe
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls.

5.7.85. FEISC_0xC0_SAMCmd, FEISC_0xC0_SAMCmd_Sync

Funktion	Ein SAM Command Task wird synchron oder asynchron zur Applikation gestartet. SAM = Secure Access Module
Syntax	**(1) int FEISC_0xC0_SAMCmd(int iReaderHnd, int iSlot, UCHAR* cReqData, int iReqLen, FEISC_TASK_INIT* pInit)** **(2) int FEISC_0xC0_SAMCmd_Sync(int iReaderHnd, UCHAR cBusAdr, int iSlot, int iTimeout, UCHAR* cReqData, int iReqLen, UCHAR* cRspData, int* iRspLen)**
Beschreibung	Mit der Funktion (1) wird der SAM Command als ein asynchroner Task gestartet. Ein asynchroner Task ist ein interner Thread, der das SAM Kommando an den Leser sendet und für eine Zeit Timeout auf die Antwort wartet. Die Signalisierung der Antwortdaten bzw. der Abbruchbedingung an die Applikation erfolgt mit dem Aufruf einer Callback-Funktion. Die Funktion (2) führt das SAM Command synchron aus. Die Antwortdaten sind in *cRspData* und die Anzahl empfangener Bytes in *iRspLen* enthalten. Der Parameter *iSlot* enthält die Nummer des Einsteckplatzes (Slot). Der Parameter *iTimeout* definiert die maximale Wartezeit im Leser. Die maximale Wartezeit im Host sollte etwas höher eingestellt werden. Die für den Datentransfer notwendigen Daten sind in *cReqData* zu hinterlegen. In *iReqDataLen* muß die Anzahl der in *cReqData* enthaltenen Zeichen angegeben werden. Alle für den Task in (1) und für die Callback-Funktion relevanten Daten sind in der Struktur FEISC_TASK_INIT zusammengefasst. Diese Struktur wird im Kapitel 5.4. Asynchrone Tasks zur Entlastung von Applikationen genauer beschrieben. Folgende Verwendung wird empfohlen: `FEISC_TASK_INIT Init;` `Init.cbFct1 = this->cbsTaskRsp1; // Callback-Funktion` `Init.ucBusAdr = 255; // jeder Leser antwortet` `Init.uiFlag = FEISC_TASKCB_1;` `Init.uiTimeout = m_uiTimeout; // individuelle Timeout` `Init.pAny = this; // optional: This-Pointer` *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im Leser eingestellte Busadresse.
Querverweis	Weitere Informationen zu asynchronen Tasks finden sich im Kapitel 5.4. Asynchrone Tasks zur Entlastung von Applikationen. 5.7.13. FEISC_CancelAsyncTask
Anmerkung	Nähere Informationen zum Protokoll [0xC0] SAM Command finden sich im Systemhandbuch zur OBID® *classic-pro* Leserfamilie.
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls.

5.7.86. FEISC_0xC1_DESFireCmd

Funktion	Funktion initiiert einen Datentransfer mit einem ISO 14443-4, Type A DESFire Transponder.
Syntax	**int FEISC_0xC1_DESFireCmd(int iReaderHnd, UCHAR cBusAdr, UCHAR cSubCmd, UCHAR cMode, UCHAR* cAppID, UCHAR cReaderKeyIndex, UCHAR* cReqData, int iReqLen, UCHAR* cRspData, int* iRspLen, int iDataFormat)**
Beschreibung	Die Funktion führt einen DESFire-spezifischen Befehl aus. Alle Parameter sind im Detail im Systemhandbuch des jeweiligen Lesers erklärt. Der Parameter *iDataFormat* bestimmt, ob die Sendedaten in *cReqData* und die Empfangsdaten in *cRspData* als Hex-Array (*iDataFormat*=0) oder als Zeichenkette (*iDataFormat*=1) zu interpretieren sind. *iReaderHnd* ist der Handle zum Leser-Object. *cBusAdr* ist die im Leser eingestellte Busadresse.
Querverweis	Weitere Informationen zu *iDataFormat* in Kapitel 5.3. Parameterübergabe
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.87. FEISC_0xC2_MifarePlusCmd

Funktion	Funktion initiiert einen Datentransfer mit einem ISO 14443, Type A MIFARE Plus Transponder.
Syntax	**int FEISC_0xC2_MifarePlusCmd(int iReaderHnd, UCHAR cBusAdr, UCHAR cSubCmd, UCHAR cMode, UCHAR* cReqData, int iReqLen, UCHAR* cRspData, int* iRspLen, int iDataFormat)**
Beschreibung	Die Funktion führt einen MIFARE Plus spezifischen Befehl aus. Alle Parameter sind im Detail im Systemhandbuch des jeweiligen Lesers erklärt. Der Parameter *iDataFormat* bestimmt, ob die Sendedaten in *cReqData* und die Empfangsdaten in *cRspData* als Hex-Array (*iDataFormat*=0) oder als Zeichenkette (*iDataFormat*=1) zu interpretieren sind. *iReaderHnd* ist der Handle zum Leser-Object. *cBusAdr* ist die im Leser eingestellte Busadresse.
Querverweis	Weitere Informationen zu *iDataFormat* in Kapitel 5.3. Parameterübergabe
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.7.88. FEISC_0xC3_DESFireCmd

Funktion	Funktion initiiert einen Datentransfer mit einem ISO 14443-4, Type A DESFire Transponder.
Syntax	**int FEISC_0xC3_DESFireCmd(int iReaderHnd, UCHAR cBusAdr, UCHAR cSubCmd, UCHAR cMode, UCHAR* cReqData, int iReqLen, UCHAR* cRspData, int* iRspLen, int iDataFormat)**
Beschreibung	Die Funktion führt einen DESFire-spezifischen Befehl aus. Alle Parameter sind im Detail im Systemhandbuch des jeweiligen Lesers erklärt. Der Parameter *iDataFormat* bestimmt, ob die Sendedaten in *cReqData* und die Empfangsdaten in *cRspData* als Hex-Array (*iDataFormat*=0) oder als Zeichenkette (*iDataFormat*=1) zu interpretieren sind. *iReaderHnd* ist der Handle zum Leser-Object. *cBusAdr* ist die im Leser eingestellte Busadresse.
Querverweis	Weitere Informationen zu *iDataFormat* in Kapitel 5.3. Parameterübergabe
Rückgabewert	Der Rückgabewert enthält im fehlerfreien Fall das Statusbyte des Antwortprotokolls. Die Liste der Fehlercodes findet sich im Anhang.

5.8. Unterstützung für Multithreading

Die Funktionen in FEISC sind prinzipiell Thread-sicher, d.h. dass Funktionsaufrufe aus mehreren Threads an die Bibliothek möglich sind, solange ein Kommunikationsvorgang eines Threads nie durch einen weiteren Kommunikationsvorgang eines anderen Threads unterbrochen wird.

Innerhalb der Bibliothek sind keine Schutzmaßnahmen vorhanden, die eine verdrängende Vorgehensweise eines anderen Threads unterbinden. Dieser Schutz muß in der Anwendungsebene implementiert werden.

Ein Problem stellt sich, wenn mit der mittels der Funktion **FEISC_AddEventHandler** implementierten Ereignisbehandlungsmethode ein Protokollstring an die Anwendung übergeben und in einem Protokollfenster dargestellt wird. Dabei kann die Darstellung des Strings im Fenster aus dem Thread heraus zum Programmabsturz führen (z.B. mit Verwendung der MFC in C++). Abhilfe schafft die Zwischenspeicherung in einem Puffer und die Versendung einer Windows-Nachricht mit der API-Funktion SendMessage(..) an das Fenster. Damit erreicht man eine Entkopplung der Threads. Besser ist es, in solchen Fällen gleich die Message-Methoden von **FEISC_AddEventHandler** zu wählen.

Jedoch kann das Schließen des Fensters während einer Protokolldarstellung wiederum zum Programmabsturz führen. Hierfür bietet die FEISC Unterstützung in der Weise, dass gezielt die Protokollausgabe in der Bibliothek in allen Reader-Objekten gestoppt werden kann. Dies wird mit dem Funktionsaufruf **FEISC_SetReaderPara**(0, „LockProtToApp“, „“) eingeleitet. Anschließend prüft man mit der Funktion **FEISC_GetReaderPara**(0, „IsProtToAppLocked“, „“) solange, bis alle Protokollausgaben aus der Bibliothek heraus beendet sind. Liefert die Funktion eine 0 zurück, ist die Protokollausgabe noch nicht abgeschlossen. Wird eine 1 zurückgegeben, kann das Fenster geschlossen werden kann. Die Rückgabewerte sind entgegen der Konvention so gewählt, damit man (jedenfalls mit C) auf true prüfen kann.

C++ Beispiel mit MFC:

Die Memberfunktion OnClose wird aufgerufen, wenn man das Fenster (View) mit einem Mausklick auf das Schließsymbol schließen will. Die von CMDIChildWnd abgeleitete Klasse FELogChildFrame ist das Rahmenfenster des Doc/View-Paares zum Protokollausgabefenster. Durch zyklisches Wiederaufrufen mit einer WM_CLOSE-Nachricht an sich selbst wird eine Zeitschleife realisiert, die der FEISC Zeit zum Abschließen der Protokollausgaben gibt. Erst wenn die Funktion **FEISC_GetReaderPara**(0, „IsProtToAppLocked“, „“) keine 0 mehr zurückliefert, darf das Fenster mit CMDIChildWnd::OnClose() geschlossen werden.

```
void FELogChildFrame::OnClose()
{
    // Mitteilung an Bibliothek, dass alle weiteren Protokollausgaben zu sperren sind
    FEISC_SetReaderPara(0, "LockProtToApp", "");

    // Anfrage an Bibliothek, ob alle Protokollausgaben schon beendet wurden
    int iBack = FEISC_GetReaderPara(0, "IsProtToAppLocked", "");

    if(iBack==0)
    {
        // nein, also mit Message an this erneuter Aufruf von OnClose
        this->SendMessage(WM_CLOSE, 0, 0);
        return;
    }

    // wenn hier angekommen, dann sind alle Protokollausgaben der DLL beendet
    // somit besteht keine Absturzgefahr mehr, wenn das Doc/View-Paar geschlossen wird
    CMDIChildWnd::OnClose();
}
```

6. Anhang

6.1. Fehlercodes

Fehlerkonstante	Wert	Beschreibung
FEISC_ERR_NEWREADER_FAILURE	-4000	Fehler beim Erzeugen eines neuen Reader-Objekts
FEISC_ERR_EMPTY_LIST	-4001	Reader-Handleliste ist leer (keine Reader-Objekte angelegt)
FEISC_ERR_POINTER_IS_NULL	-4002	Zeiger auf Übergabeparameter ist NULL
FEISC_ERR_NO_MORE_MEM	-4003	Kein Systemspeicher mehr
FEISC_ERR_UNKNOWN_COMM_PORT	-4004	unbekannter COM-Port
FEISC_ERR_UNSUPPORTED_FUNCTION	-4005	nicht unterstützte Funktion
FEISC_ERR_NO_USB_SUPPORT	-4006	keine USB-Unterstützung (z.B. unter NT4)
FEISC_ERR_OLD_FECOM	-4007	alte FECOM.DLL detektiert
FEISC_ERR_NO_VALUE	-4010	Kein Datenwert vorhanden
FEISC_ERR_UNKNOWN_HND	-4020	der übergebene Reader-Handle ist unkekannt
FEISC_ERR_HND_IS_NULL	-4021	der übergebene Reader-Handle ist 0
FEISC_ERR_HND_IS_NEGATIVE	-4022	der übergebene Reader-Handle ist negativ
FEISC_ERR_NO_HND_FOUND	-4023	kein Reader-Handle in Reader-Handleliste gefunden
FEISC_ERR_PORTHND_IS_NEGATIVE	-4024	der übergebene Port-Handle ist negativ
FEISC_ERR_HND_UNVALID	-4025	ungültiger Port-Handle; das erste Byte (MSB) im Port-Handle ist ungültig
FEISC_ERR_PROTLEN	-4030	Protokolllängenfehler
FEISC_ERR_CHECKSUM	-4031	Checksummenfehler
FEISC_ERR_BUSY_TIMEOUT	-4032	Timeout nach dauerhaften Busy-Meldungen
FEISC_ERR_UNKNOWN_STATUS	-4033	unbekanntes Statusbyte
FEISC_ERR_NO_RECPROTOCOL	-4034	kein USB-Empfangsprotokoll eingetroffen
FEISC_ERR_CMD_BYTE	-4035	falsches Steuerbyte im Empfangsprotokoll
FEISC_ERR_TRANSCEIVE	-4036	allgemeiner USB-Kommunikationsfehler
FEISC_ERR_REC_BUS_ADR	-4037	falsche Busadresse im Empfangsprotokoll
FEISC_ERR_UNKNOWN_PARAMETER	-4050	Übergabeparameter ist nicht bekannt
FEISC_ERR_PARAMETER_OUT_OF_RANGE	-4051	Übergabeparameter zu groß oder zu klein
FEISC_ERR_ODD_PARAMETERSTRING	-4052	Die übergebene Zeichenkette enthält eine ungerade Anzahl Zeichen
FEISC_ERR_UNKNOWN_ERRORCODE	-4053	unbekannter Fehlercode
FEISC_ERR_UNSUPPORTED_OPTION	-4054	Option wird nicht unterstützt
FEISC_ERR_UNKNOWN_EPC_TYPE	-4055	Unbekannter EPC Typ
FEISC_ERR_NO_PLUGIN	-4060	Es ist kein PlugIn installiert

Fehlerkonstante	Wert	Beschreibung
FEISC_ERR_PLUGIN_PRESENT	-4061	Es ist bereits ein PlugIn installiert
FEISC_ERR_UNKNOWN_PLUGIN_ID	-4062	unbekannte PlugIn-ID
FEISC_ERR_PI_BUILD_DATA	-4063	Fehler beim Zusammenbau der Protokolldaten
FEISC_ERR_PI_BUILD_FRAME	-4064	Fehler beim Zusammenbau desProtokollrahmens
FEISC_ERR_PI_SPLIT_FRAME	-4065	Fehler beim Zerlegen des Protokollrahmens
FEISC_ERR_PI_SPLIT_DATA	-4066	Fehler beim Zerlegen der Protokolldaten
FEISC_ERR_BUFFER_OVERFLOW	-4070	Überlauf in Daten- oder Protokollpuffer
FEISC_ERR_TASK_STILL_RUNNING	-4080	Asynchroner Task ist bereits gestartet
FEISC_ERR_TASK_NOT_STARTED	-4081	Asynchroner Task konnte nicht gestartet werden
FEISC_ERR_TASK_TIMEOUT	-4082	Asynchroner Task Timeout: Leser antwortet nicht mehr
FEISC_ERR_TASK_SOCKET_INIT	-4083	Der Socket für den Task konnte nicht initialisiert werden.
FEISC_ERR_TASK_BUSY	-4084	Der Ausführungspfad des asynchronen Tasks liegt gerade innerhalb der Callback-Funktion. Deshalb kann keine Aktion ausgeführt werden. Die Funktion muß ernneut aufgerufen werden.
FEISC_ERR_THREAD_CANCEL_ERROR	-4085	Das Beenden des asynchronen Task war nicht möglich.
FEISC_ERR_CRYPT_LOAD_LIBRARY	-4090	Die Bibliothek openSSL konnte nicht geladen werden.
FEISC_ERR_CRYPT_INIT	-4091	Fehler bei der Initialisierung des Kryptomoduls.
FEISC_ERR_CRYPT_AUTHENT_PROCESS	-4092	Fehler im Authentifizierungsprozess.
FEISC_ERR_CRYPT_ENCYPHER	-4093	Fehler beim Verschlüsseln.
FEISC_ERR_CRYPT_DECYPHER	-4094	Fehler beim Entschlüsseln.

6.2. Liste der Parameterkennungen

Parameterkennung	Wertebereich	Default	Einheit	Beschreibung
PortHnd[21]	0 ... 4294967295	0		PortHandle für Kommunikation mit ID FECOM, ID FETCP bzw. ID FEUSB
LogProt	0, 1	0		wenn 1, dann Übergabe von Protokollstrings an Applikation über Ereignissignalisierung möglich[22]
LogFile	0, 1	0		wenn 1, dann Speichern von Protokollstrings in Logfile feisc_log.txt
LogFileName	Max 256 Zeichen	feisc_log.txt		Dateiname für LogFile
Language	7 - Deutsch 9 - Englisch	9	-	Auswahl der Sprache für interne Textresourcen.
RecBusAdr	0 ... 255	-	-	mit dem letzten Protokoll empfangene Busadresse. Wert kann nur gelesen werden.
ChkRecBusAdr	0, 1	0	-	wenn 1, dann wird die empfangene Busadresse mit der gesendeten Busadresse verglichen und bei Ungleichheit ein Fehler erzeugt. Ausgenommen sind die Busadressen 254 und 255.
ConvHexToString	0, 1	0	-	Konvertiert (wenn 1) alle im Scanmodus empfangenen Bytes in einen String. Parameter wird nur benötigt, wenn im Leser die Datenausgabe im Scannermodus auf *unformatierte Hexdaten* eingestellt ist.
FrameSupport	„Standard“, „Advanced“	“Standard”	-	Wahl des Protokollrahmens von Sendeprotokollen. Die Anpassung für Empfangsprotokolle erfolgt automatisch.
SendStr	-	-	-	liefert letztes Sendprotokoll mit vorangestelltem Datum und Uhrzeit
RecStr	-	-	-	liefert letztes Empfangsprotokoll mit vorangestelltem Datum und Uhrzeit
LockProtToApp	ohne	-		Unterstützung für Multithreading: sperrt die Protokollausgabe mittels Ereignissignalisierung in allen Reader-Objekten s. 5.8. Unterstützung für Multithreading
UnlockProtToApp	ohne	-		Unterstützung für Multithreading: hebt die Sperre für die Protokollausgabe wieder auf s. 5.8. Unterstützung für Multithreading
IsProtToAppLocked	ohne	-		Unterstützung für Multithreading: fragt ab, ob alle Reader-Objekte mit der Protokollausgabe fertig sind s. 5.8. Unterstützung für Multithreading

[21] Man beachte die Hinweise in 5.7.2. FEISC_NewReader

[22] Siehe Kapitel 5.5. Ereignissignalisierung an Applikationen und 5.7.10. FEISC_AddEventHandler

6.3. Liste der Konstanten für die FEISC_EVENT_INIT-Struktur

Die Konstantendefinitionen sind in der Datei FEISC.H enthalten.

Konstante	Wert	Verwendung	Beschreibung
FEISC_THREAD_ID	1	uiFlag	Ereignissignalisierung mit Thread-Nachricht
FEISC_WND_HWND	2	uiFlag	Ereignissignalisierung mit Window-Nachricht
FEISC_CALLBACK	3	uiFlag	Ereignissignalisierung mit 1. Callback-Funktion
FEISC_EVENT	4	uiFlag	Ereignissignalisierung mit Windows-API-Event
FEISC_CALLBACK_2	5	uiFlag	Ereignissignalisierung mit 2. Callback-Funktion
FEISC_CALLBACK_4	6	uiFlag	Ereignissignalisierung mit 4. Callback-Funktion
FEISC_PRT_EVENT	1	uiUse	Signalisierung bei Sende- und Empfangsprotokollen
FEISC_SNDPRT_EVENT	2	uiUse	Signalisierung bei Sendeprotokollen
FEISC_RECPRT_EVENT	3	uiUse	Signalisierung bei Empfangsprotokollen
FEISC_SCANNER_EVENT	4	uiUse	Signalisierung bei Empfang eines Scannerprotokolls

6.4. Liste der Konstanten für TaskID und die FEISC_TASK_INIT-Struktur

Die Konstantendefinitionen sind in der Datei FEISC.H enthalten.

Konstante	Wert	Verwendung	Beschreibung
FEISC_TASKID_FIRST_NEW_TAG	1	iTaskID	einmaliger Inventarisierungstask
FEISC_TASKID_EVERY_NEW_TAG	2	iTaskID	repetierender Inventarisierungstask
FEISC_TASKID_NOTIFICATION	3	iTaskID	unendlicher Task zum Empfang von Notifications
FEISC_TASKID_SAM_COMMAND	4	iTaskID	einmaliger Task zum Empfang der SAM-Antwort
FEISC_TASKID_COMMAND_QUEUE	5	iTaskID	einmaliger Task zum Empfang der Antwort eines Queue-Commands
FEISC_TASKID_MAX_EVENT	6	iTaskID	unendlicher Task zum Empfang von Access-Notifications
FEISC_TASKID_PEOPLE_COUNTER	7	iTaskID	unendlicher Task zum Empfang von People Counter Events
FEISC_TASKCB_1	1	uiFlag	Auswahl von cbFct1
FEISC_TASKCB_2	2	uiFlag	Auswahl von cbFct2
FEISC_TASKCB_3	3	uiFlag	Auswahl von cbFct3
FEISC_TASK_CHANNEL_TYPE_AS_OPEN	1	uiChannelType	für alle Inventarisierungstasks

FEISC_TASK_CHANNEL_TYPE_NEW_TCP	5	uiChannelType	für Notification-Task

6.5. Historie

V7.02.02

- Modifikationen für **FEISC_StartAsyncTask**:

  a) Der Listener-Port muss bei der Initialisierung des asynchronen Tasks systemweit frei sein. Andernfalls wird der neue Fehlercode -4086 zurückgegeben.

  b) Listener-Port für Notification-Mode nimmt nur noch eine Verbindung zur gleichen Zeit an. Alle weiteren Verbindungsversuche werden abgelehnt.

- Linux:

  Version für 64-Bit

V7.01.06

- Erweiterungen für Notifications bei verschlüsselter Datenübertragung
- Interne Erweiterung für Mode 0x21 des Commands [0x6E] Reader Diagnostic

V7.01.04

- Verbesserungen für gesicherte Datenverbindung:

  1. FEISC_0x52_GetBaud erweitert
  2. Wiederholung eines Protokolls nach einem Crypto Processing Error

- Verbesserungen für FEISC_0xC0_SAMCmd_Sync bzgl. Timeout-Verhalten

V7.01.00

- Verbesserte Threadsicherheit
- FEISC_StartAsyncTask gibt einen Fehlercode zurück, wenn der interne Thread nicht gestartet werden konnte.

- Windows:

  1. Migration der Entwicklungsumgebung von Visual Studio 2008 zu Visual Studio 2010.
  2. DLL jetzt ohne MFC
  3. Erstes Release der 64-Bit Version
  4. Anbindung an Log-Manager

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

V7.00.01

- Fehlerkorrektur für Keep-Alive im Notification-Task.

V7.00.00

- Diese Version ist aus den nachfolgenden Gründen nicht vollständig kompatibel mit Vorversionen. Modifikationen am Quellcode von Applikationen können notwendig sein.
- Erweiterung in der Struktur ***struct _FEISC_TASK_INIT*** um Keep-Alive Parameter für den Notification-Task. Bedingt dadurch muss entweder der neue Parameter *bKeepAlive* auf false oder besser, die gesamte Struktur mit 0 initialisert werden (z.B. mit memset). Es wird empfohlen, die Codezeilen genau zu untersuchen, die diese Struktur erzeugen und initialisieren.
- Neues Plug-in API für die Anbindung alternativer Schnittstellen
- Entfernte Funktionen: **FEISC_InstallPlugIn and FEISC_RemovePlugIn**
- Windows / Windows CE:
  1. Migration der Entwicklungsumgebung von Visual Studio 6 zu Visual Studio 2008.
  2. Anpassung der Callback Funktionsdeklaration in den beiden Strukturen ***struct _FEISC_EVENT_INIT*** und ***struct _FEISC_TASK_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.

V6.02.01

- Fehlerkorrektur für automatische Abschaltung des Kryptomodes

V6.02.00

- Neue Funktionen: **FEISC_0xC3_DESFireCmd** und **FEISC_0xC0_SAMCmd_Sync**

V6.01.00

- Unterstützung für People Counter ID ISC.ANTGPC
- Neue Funktion:

  FEISC_0x9F_Piggyback_Command
- Erweiterungen in der Struktur FEISC_EVENT_INIT für Ereignissignalisierung

V6.00.00

- Option zur Verschlüsselung der Protokolle mit Hilfe der openSSL Bibliothek in der Version 0.9.8l (s. 5.6. Sicherheit in der Datenübertragung).
- Neue Funktionen:

  FEISC_0x8A_ReadConfiguration

  FEISC_0x8B_WriteConfiguration

  FEISC_0x8C_ResetConfiguration

  FEISC_0xAD_WriteReaderAuthentKey

  FEISC_0xAE_ReaderAuthent

- Neue Fehlerkonstanten

Fehlerkonstante	Wert	Beschreibung
FEISC_ERR_CRYPT_INIT	-4091	Fehler bei der Initialisierung des Kryptomoduls.
FEISC_ERR_CRYPT_AUTHENT_PROCESS	-4092	Fehler im Authentifizierungsprozess.
FEISC_ERR_CRYPT_ENCYPHER	-4093	Fehler beim Verschlüsseln.
FEISC_ERR_CRYPT_DECYPHER	-4094	Fehler beim Entschlüsseln.

V5.07.13

- Neue Funktionen: **FEISC_0x1F_MAXDataExchange, FEISC_0x76_CheckAntennas, FEISC_0xC2_MifarePlusCmd**

V5.07.10

- Die Kommunikationsbibliothek FECOM wird so parametriert, dass der Empfangsalgorithmus für OBID Protokollrahmen optimiert ist. Diese Option wird automatisch für den internen Scannerthread abgeschaltet.
- Neue Funktionen: **FEISC_0xC1_DESFireCmd, FEISC_0xA3_Write_DES_AES_Keys**

V5.07.05

- Verifikation des empfangenen Protokollrahmens in der Funktion **FEISC_SendTransparent**
- Neue Funktionen: **FEISC_0x8A_ReadConfiguration, FEISC_0x8B_WriteConfiguration, FEISC_0x8C_ResetConfiguration,**

V5.06.03

- Neue Funktionen: **FEISC_0xC0_SAMCmd**, **FEISC_0xBC_CmdQueue**, **FEISC_0xBB_C1G2_TranspCmd**

V5.05.05

- Optimierung für internen Notification-Thread (aktiviert mit **FEISC_StartAsyncTask**) für Kommunikationskanäle mit hoher Fehlerrate, wie z.B. GPRS.
- Neuer Parameter für **FEISC_GetReaderPara** bzw. **FEISC_SetReaderPara**: LogFilename

V5.05.01

- USB-Unterstützung für Linux
- Neue Funktionen: **FEISC_0xB4_EPC_UHF_Cmd**, **FEISC_0x6B_CentralizedRFSync**
- Neue Statusbytes: 0x86, 0x18
- Die Linux-Version wurde mit dem Compiler GCC 3.3.3 unter SuSE Linux 9.1 erstellt

V5.04.11

- Modifizierte Lizenzbestimmung
- Neuer Fehlercode -4085

V5.04.10

- Neuer Task: Notification für Leser mit Notification Mode.
- Änderungen in der Struktur **FEISC_TASK_INIT**, die dadurch nicht mehr kompatibel zur Vorversion ist.
- Neue Funktion: **FEISC_0x34_ForceNotifyTrigger**
- Alle Threads sind unter Linux verfügbar
- Neue Statusbytes: 0xF1, 0xF2, 0xF8
- Neue Fehlercodes: FEISC_ERR_TASK_SOCKET_INIT, FEISC_ERR_TASK_BUSY

V5.04.00

- Neue Funktionen **FEISC_StartAsyncTask**, **FEISC_CancelAsyncTask** und **FEISC_TriggerAsyncTask**.
- Neue Fehlercodes

V5.03.09

- Neue Funktion **FEISC_0x72_SetOutput**.
- **FEISC_0x22_ReadBuffer** unterstützt erweiterte Optionen (TR-DATA, INPUT, STATUS).

V5.03.03

- Neue Funktion **FEISC_0xB3_EPCCmd**.
- **FEISC_Transmit** und **FEISC_Receive** für alle Schnittstellentypen verwendbar.
- Neues Statusbyte: 0x96 (ISO14443-Error)

V5.03.00

- Die neue Version ist nicht zu 100% rückwärtskompatibel zur Vorversion, weil die Funktion **FEISC_0x66_FirmwareVersion** umbenannt wurde. Der neue Funktionsname ist **FEISC_0x66_ReaderInfo**.

V5.02.00

- Vorbereitet für kommende neue USB-Protokolle
- Die neue Version ist nicht zu 100% rückwärtskompatibel zur Vorversion, weil die Funktion **FEISC_0x18DestroyEPC** umbenannt wurde und eine neue Parameterliste bekam. Der neue Funktionsname ist **FEISC_0x18Destroy**.
- Neuer Fehlercode: -4055.
- Kleinere Fehlerkorrekturen

V5.01.19

- Unterstützung für den Transponder I-CODE UID im Protokoll [0x18] Destroy.
- Erstes Linux Release (SuSE Linux 8.2, GNU Compiler Collection V3.3-23, glibc V2.3.2-6)

V5.01.17

- Plug-In Mechanismus zur benutzerdefinierten Erweiterung der Protokollausgabe.
- Alle Funktionen, mit Ausnahme von **FEISC_BuildProtocol** und **FEISC_SplitProtocol**, sind zu 100% rückwärtskompatibel zur Vorversion.
- **FEISC_BuildProtocol** ist umbenannt worden in **FEISC_BuildSendProtocol** und hat Änderungen in den Aufrufparametern.
- **FEISC_SplitProtocol** ist umbenannt worden in **FEISC_SplitRecProtocol** und hat Änderungen in den Aufrufparametern.
- Neue Funktionen:

FEISC_BuildRecProtocol
FEISC_SplitSendProtocol
FEISC_Conv2StdProtocol

FEISC_Conf2AdvProtocol
FEISC_InstallPlugIn
FEISC_RemovePlugIn

- Neue Protokollfunktionen:

FEISC_0x22_ReadBuffer
FEISC_0x18_DestroyEPC
FEISC_0x87_SetSystemDate
FEISC_0x88_GetSystemDate
FEISC_0x64_SystemReset.

- Unterstützung des Protokolls [0x74] Read Input für ID ISC.PRH-A und -U Leser.
- Unterstützung für Protokolle mit 2 Längenbytes.
- Thread-Sicherheit für jedes erzeugte Reader-Objekt.
- Unterstützung für echtes Multithreading: jedes erzeugte Reader-Objekt hat jetzt eigene interne Puffer. Somit können mehrere Leser parallel und gleichzeitig bedient werden, sofern diese an verschiedenen Schnittstellen angeschlossen sind. Eine gegenseitige Blockade ist ausgeschlossen.

- neue Fehlercodes:

Fehler-Konstante	Wert	Beschreibung
FEISC_ERR_NO_VALUE	-4010	Fehler in der Funktion FEISC_GetReaderPara
FEISC_ERR_NO_PLUGIN	-4060	Fehler, weil kein Plug-In Objekt installiert wurde
FEISC_ERR_PLUGIN_PRESENT	-4061	Fehler, weil schon ein Plug-In Objekt installiert ist
FEISC_ERR_UNKNOWN_PLUGIN_ID	-4062	Unbekannte Plug-In ID
FEISC_ERR_PI_BUILD_DATA	-4063	Fehler in der Plug-In Funktion build_datastream
FEISC_ERR_PI_BUILD_FRAME	-4064	Fehler in der Plug-In Funktion build_protocol
FEISC_ERR_PI_SPLIT_FRAME	-4065	Fehler in der Plug-In Funktion split_protocol
FEISC_ERR_PI_SPLIT_DATA	-4066	Fehler in der Plug-In Funktion split_datastream
FEISC_ERR_BUFFER_OVERFLOW	-4070	Datenpuffer ist zu klein

V5.01.00

- Alle Funktionen sind zu 100% rückwärtskompatibel zur Vorversion

- Neue Funktionen: **FEISC_0xBD_ISOTranspCmd**, **FEISC_0xBE_ISOTranspCmd**
- Integration der TCP/IP-Unterstützung bei Verwendung des Support-Pakets ID FETCP
- Fehlerkorrekturen in **FEISC_0xBF_ISOTranspCmd** für Parameter iDataFormat=1
- empfangene Busadresse wird gespeichert und kann mit **FEISC_GetReaderPara** abgefragt werden
- neuer Fehlercode: -4054 (FEISC_ERR_UNSUPPORTED_OPTION)

V5.00.00

- Alle Funktionen sind zu 100% rückwärtskompatibel zur Vorversion

- Neue Funktionen: **FEISC_0xA2_WriteMifareKeys**, **FEISC_0xB2_ISOCmd**
- Erste Version für Windows CE

V4.09.00

- Alle Funktionen sind zu 100% rückwärtskompatibel zur Vorversion

- Alle Konstanten der Headerdatei feiscdef.h sind in die Datei feisc.h verschoben worden. feiscdef.h ist damit überflüssig.
- neue Funktion: **FEISC_0x55_StartFlashLoaderEx** erlaubt die Übergabe einer Busadresse und ersetzt die Funktion **FEISC_0x55_StartFlashLoader**.
- Interne Überprüfung der empfangenen Busadresse (ist standardmäßig deaktiviert)
- neue Parameter-Konstante ChkRecBusAdr für die Funktionen **FEISC_SetReaderPara** und **FEISC_GetReaderPara** zur Aktivierung der Überprüfung der empfangenen Busadresse
- neue Parameter-Konstante Language für die Funktionen **FEISC_SetReaderPara** und **FEISC_GetReaderPara** zur Auswahl der Landessprache für interne Texte
- neuer Fehlercode FEISC_ERR_REC_BUS_ADR
- neue Flag-Konstante für Struktur FEISC_EVENT_INIT: FEISC_CALLBACK_2
- neue Use-Konstante für Struktur FEISC_EVENT_INIT: FEISC_SCANNER_EVENT

V4.06.00 – V4.08.00

- Neue Funktion **FEISC_0x6F_AntennaTuning**
- Nicht mehr enthaltene Funktionen:

  FEISC_0x01_MultiJobPoll
  FEISC_0x01_MultiJobPollAndState
  FEISC_0x03_MultiJobState
  FEISC_0x11_GetSerNr
  FEISC_0x14_WritePData
  FEISC_0x15_ReadPData
  FEISC_0x16_WriteCData
  FEISC_0x17_ReadCData
  FEISC_0x6B_InitNoiseLevel

V4.04.00 – V4.05.00

- interne Versionen

V4.03.00

- Änderung der Aufrufparameter in **FEISC_0xBF_ISOTranspCmd**

V4.02.00

- Überprüfung des Steuerbytes im Antwortprotokoll
- Fehlerkorrekturen für Protokolltransfer über USB
- Beseitigung kleinerer Fehler

V4.01.00

- Neue Funktionen: **FEISC_GetStatusText**, **FEISC_0xB1_ISOCustAndPropCmd**, **FEISC_0xBF_ISOTranspCmd**.

V4.00.00

- Offizielle Release-Version. Ohne Änderungen gegenüber V3.01.00.

V3.01.00

- FEISC.DLL kann nur noch mit FECOM.DLL ab Version 2.00.00 zusammenarbeiten. Mit älteren Versionen von FECOM.DLL kann keine Kommunikation ausgeführt werden.
- Die Ereignissignalisierung unterstützt jetzt auch Events des Windows-API

V3.00.00

- Unterstützung von OBID® USB-Geräten
- neue Funktionen: **FEISC_GetErrorText**, **FEISC_GetLastError**, **FEISC_AddEventHandler**, **FEISC_DelEventHandler**.
- Limitierung des Port-Handle (Übergabeparameter *iPortHnd* in **FEISC_NewReader**) auf 0x0FFFFFFF. Das erste Byte (MSB) ist reserviert für die Unterscheidung des Kommunikations-Kanals (Asynchron, USB).

V2.01.00

- neue Parameter für **FEISC_GetReaderPara**: ERRCODE, ERRSTR, SENDSTR, RECSTR
- Umbenennung der Funktionen **FEISC_0x85_SetTime** in **FEISC_0x85_SetSysTimer** und **FEISC_0x86_GetTime** in **FEISC_0x86_GetSysTimer**.
- neue Funktionen: **FEISC_0x55_StartFlashLoader**, **FEISC_0x6E_RdDiag** und **FEISC_0xA0_RdLogin**.

V2.00.03

Fehlerbeseitigung in **FEISC_0x01_MultiJobPoll**, **FEISC_0x01_MultiJobPollAndState** und **FEISC_0x03_MultiJobState**.

Neu hinzugekommen sind Steuerparameter zur Unterstützung von Multithreading (s. 5.8. Unterstützung für Multithreading)

) und die Funktion **FEISC_0x75_AdjAntenna**.

Version 2.00.01

Umbenennung der Funktion **FEISC_0x23_InitBuffer** in **FEISC_0x33_InitBuffer**, da sich das Steuerbyte des Protokolls geändert hat.

V2.00.00

Neue Funktionen für den Long-Range-Reader ID ISCLR:

1. **FEISC_0x01_MultiJobPoll**
2. **FEISC_0x01_MultiJobPollAndState**
3. **FEISC_0x03_MultiJobState**
4. **FEISC_0x21_ReadBuffer**
5. **FEISC_0x23_InitBuffer**
6. **FEISC_0x31_ReadDataBufferInfo**
7. **FEISC_0x32_ClearDataBuffer**
8. **FEISC_0x6B_InitNoiseThreshold**
9. **FEISC_0x6C_SetNoiseLevel**
10. **FEISC_0x6D_GetNoiseLevel**
11. **FEISC_0x84_SetCFGMemLoc**
12. **FEISC_0x85_SetTime**
13. **FEISC_0x86_GetTime**

Zusätzlich wurde in folgenden Funktionen die Parameterliste erweitert:

1. **FEISC_BuildProtocol**: der Parameter iDataFormat ist neu
2. **FEISC_SplitProtocols**: der Parameter iDataFormat ist neu
3. **FEISC_GetLastSendProt**: der Parameter iDataFormat ist neu
4. **FEISC_GetLastRecProt**: der Parameter iDataFormat ist neu
5. **FEISC_SendTransparent**: der Parameter iDataFormat ist neu
6. **FEISC_Transmit**: der Parameter iDataFormat ist neu
7. **FEISC_Receive**: der Parameter iDataFormat ist neu
8. **FEISC_0x80_ReadConfBlock**: der Parameter iDataFormat ist neu
9. **FEISC_0x81_WriteConfBlock**: der Parameter iDataFormat ist neu

Dies haben wir im Interesse der Visual Basic Programmierer getan.

Als neue Abfragefunktion ist hinzugekommen: **FEISC_GetLastRecProtLen**