OBID i-*scan*®

MANUAL

ID FEISC

Version 5.07.05

Software-Support for

OBID i-*scan*®

and

OBID® *classic-pro*

For 32-Bit Operating Systems

Windows 2000/XP/Vista

and Windows CE

and Linux

final
public (B)
2008-10-30
H9391-30e-ID-B.doc

FEIG ELECTRONIC

Note

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

The indications made in these mounting instructions may be altered without previous notice. With the edition of these instructions, all previous editions become void.

Composition of the information given in these mounting instructions has been done to the best of our knowledge. FEIG ELECTRONIC GmbH does not guarantee the correctness and completeness of the details given and may not be held liable for damages ensuing from incorrect installation.

Since, despite all our efforts, errors may not be completely avoided, we are always grateful for your useful tips.

FEIG ELECTRONIC GmbH assumes no responsibility for the use of any information contained in this manual and makes no representation that they free of patent infringement. FEIG ELECTRONIC GmbH does not convey any license under its patent rights nor the rights of others.

The installation-information recommended here relate to ideal outside conditions. FEIG ELECTRONIC GmbH does not guarantee the failure-free function of the OBID® -system in outside environment.

Licensing agreement for use of the software

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

§1 Object and scope of the agreement

1. FEIG grants you the right to install the licensing material provided and to use it under the following conditions.
2. You may install all components of the licensing material on a hard disk or other storage medium. The installation and use may also be done on a network fileserver. You may create backup copies of the licensing material.
3. FEIG grants you the right to use the documented program library for developing your own application programs or program libraries, and you may sell the runtime file FEISC.DLL, FEISCCE.DLL or LIBFEISC.SO.x.y.z[1] without licensing fees under the stipulation that these application programs or program libraries are used to control devices and/or systems which are developed and/or sold by FEIG.

§2. Protection of the licensing material

1. The licensing material is the intellectual property of FEIG and its suppliers. It is protected in accordance with copyright, international agreements and relevant national statutes where it is used. The structure, organization and code of the software are a valuable business secret and confidential information of FEIG and its suppliers.
2. You agree not to change, modify, translate, reverse develop, decompile, disassemble the program library or the documentation or in any way attempt to discover the source code of this software.
3. To the extent that FEIG has applied protection marks, such as copyright marks and other legal restrictions in the licensing material, you agree to keep these unchanged and to use them unchanged in all complete or partial copies which you make.
4. The transmission of licensing material in part or in full is prohibited unless there is an explicit agreement to the contrary between you and FEIG. Application programs or program libraries which are created and sold in accordance with §1 Par. 3 of this Agreement are excepted.

§3 Warranty and liability limitations

1. You agree with FEIG that is not possible to develop EDP programs such that they are error-free for all application conditions. FEIG explicitly makes you aware that the installation of a new program can affect already existing software, including such software that does not run at the same time as the new software. FEIG assumes no liability for direct or indirect damages, for consequential damages or special damages, including lost profits or lost savings. If you want to ensure that no already installed program will be affected, you should not install the present software.
2. FEIG explicitly notes that this software makes it possible for irreversible settings and adaptations to be made on devices which could destroy these devices or render them unusable. FEIG assumes no liability for such actions, regardless of whether they are carried out intentionally or unintentionally.
3. FEIG provides the software „as is“ and without any warranty. FEIG cannot guarantee the performance or the results you obtain from using the software. FEIG assumes no liability or guarantee that the protection rights of third parties are not violated, nor that the software is suitable for a particular purpose.
4. FEIG call explicit attention the licensed material is not designed with components and testing for a level of reliability suitable for use in or in connection with surgical implants or as critical components in any life support systems whose failure to perform can reasonably be expected to cause significant injury to a human.
   To avoid damage, injury, or death, the user or application designer must take reasonably prudent steps to protect against system failures.

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

§4 Concluding provisions

1. This Agreement contains the complete licensing terms and conditions and supercedes any prior agreements and terms. Changes and additions must be made in writing.
2. If any provision this agreement is declared to be void, or if for any reason is declared to be invalid or of no effect, the remaining provisions shall be in no manner affected thereby but shall remain in full force and effect. Both parties agree to replace the invalid provision with one which comes closest to its original intention.
3. This agreement is subject to the laws of the Federal Republic of Germany. Place of jurisdiction is Frankfurt a. M.

Contents:

1. Introduction

The support package ID FEISC is intended to support in programming application software, integrate the OBID i-*scan*® Reader, and supports ANSI-C, ANSI-C++ und essentially any other language which can invoke C functions. This support package may be used only under 32-Bit MS Windows 2000//XP/Vista and Linux. On request, a version for Windows CE can be delivered

The support package provides a simple function interface for the OBID i-*scan*® Reader. Each protocol documented in the system manual for the OBID i-*scan*® Reader Family has its own function. The support package ID FECOM can be used for communication through a serial port on the PC. To implement your own interface functions, use the ID FEISC function collection to build and split protocols (see also 5.1. Overview).

Communication through a USB port requires the support package ID FEUSB[2].

Communication through a Ethernet port (TCP/IP) requires the support package ID FETCP.

The support package for 32-Bit Windows 2000/XP/Vista consists of the following components:

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

The support package for Windows CE consists of the following components:

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

[2] only for Windows 2000/XP/Vista and Linux

The support package for 32-Bit Linux consists of the following components:

File	Use
LIBFEISC.SO.x.y.z[3]	Function library
FEISC.H	Header file for C/C++ projects

Note: The library is compiled under SuSE Linux 9.1 with the GNU Compiler Collection V3.3.3.

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

2. Installation

2.1. 32-Bit Windows 2000/XP/Vista

Installation is quite simple: just copy the files described below to the corresponding directories.

- Copy FEISC.DLL into the Windows program directory (recommended) or system directory.
- Copy FEISC.LIB into the project or LIB directory
- Copy FEISC.H into the project or INCLUDE directory

2.2. Windows CE

Installation is quite simple: just copy the files described below to the corresponding directories.

- Copy FEISCCE.DLL into the system directory of the Windows CE system.
- Copy FEISCCE.LIB into the project or LIB directory.
- Copy FEISC.H into the project or INCLUDE directory

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

2.3. 32-Bit Linux

Installation is quite simple:

- copy the files described below to the corresponding directories.
- create a symbolic link to the library file libfeisc.so.x.y.z[4] in the directory /usr/lib:

cd /usr/lib

ln –sf /<Directory>/libfeisc.so.x.y.z libfeisc.so.x

ln –sf /<Directory>/libfeisc.so.x libfeisc.so

ldconfig

Note: The library is compiled under SuSE Linux 9.1 with the GNU Compiler Collection V3.3.3.

3. Including into the application program

If the LIB file (only Windows) was made known to the development tool, any function may be immediately used. This presumes of course the declaration of the DLL/SO functions with an INCLUDE instruction within each source file that invokes FEISC functions.

ID FECOM and/or ID FEUSB and/or ID FETCP must also be incorporated into your project if you want to invoke functions from them.

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

4. Changes since the previous version

- Check of receive protocol frame in **FEISC_SendTransparent**
- New functions: **FEISC_0x8A_ReadConfiguration, FEISC_0x8B_WriteConfiguration, FEISC_0x8C_ResetConfiguration,**

Please note also the revision history in the Appendix to this document.

5. Programming Interface

5.1. Overview

The FEISC library encapsulates for the programmer all the functions and parameters necessary for simple communication with readers in the OBID i-*scan*® Reader Family. Together with the support package ID FECOM, ID FETCP or ID FEUSB, this makes it possible to run all the protocols in the system manual of the OBID i-*scan*® Reader Family directly by invoking a function.

The functions in FEISC are responsible only for internal administration, protocol building, protocol splitting and any necessary error outputs. The FEISC library alone is not enough to communicate with an OBID i-*scan*® Reader. You can however initiate the output of a protocol and use the FECOM to communicate with an OBID i-*scan*® Reader over an asynchronous serial interface or the FETCP to communicate with a TCP/IP-Server or the FEUSB to communicate through the USB port. Other interface drivers can be integrated with the Plug-In mechanism.

Use of the FEUSB for communicating with OBID® USB devices is mandatory.

The core elements of the library are the Object Manager and the Reader objects generated during runtime.

The Object Manager implements self-administration which frees an application program from having to buffer any values, parameters or other settings: It keeps a list with all generated Reader objects. The Reader object is the central program section that carries out the protocol functions and is assigned a connection to the serial interface when using the FECOM or a channel to a USB device when using the FEUSB or a TCP/IP-Server when using the FETCP. Each Reader object administers all the parameters relevant to its protocol tasks within its local memory.

Before first using you must create a Reader object using the **FEISC_NewReader** function. If this done without error, the return value includes a handle which is used by the application program as an access number. This handle is required for unique identification of the generated Reader object. If you are using self-administration, the Object List can be called up using the **FEISC_GetReaderList** function. The successive handles which you then get can be used to read out all the parameters pertaining to this object using the **FEISC_GetReaderPara** function.

A Reader object generated using **FEISC_NewReader** must always be deleted from memory using the **REISC_DeleteReader** function.

If an application program is opened multiple times, each program (instance) gets an empty object list by invoking **FEISC_GetReaderList**. This prevents mixing up access rights under different program instances.

The object-oriented internal structure (see Fig. 1) is externally visible as a function interface, making it language-neutral.

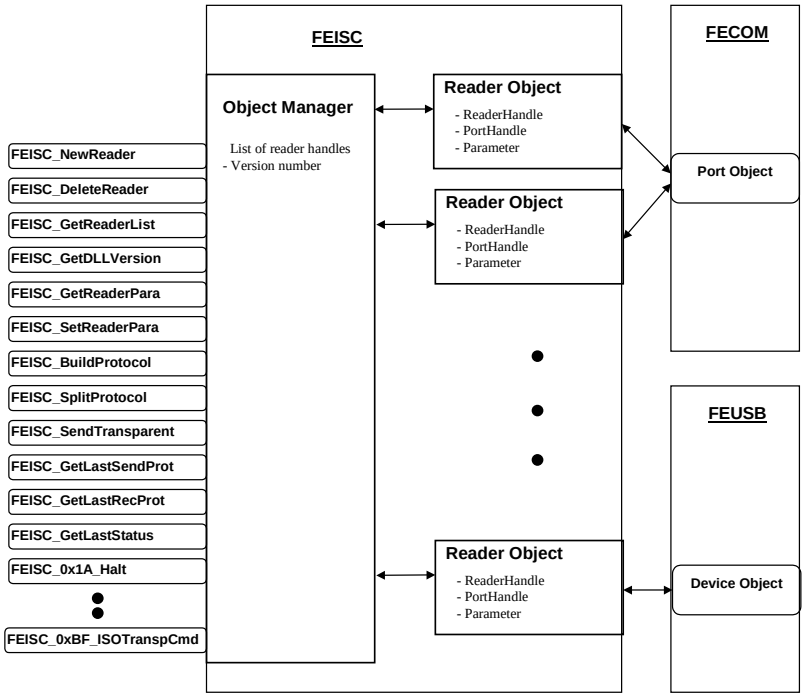

Fig. 1: Internal structure of FEISC

Fig. 1 shows how several Reader objects can share a common serial interface in FECOM or a common channel in FEUSB. No conflicts will occur as long as access to the port object takes place sequentially within a work thread. In a multi-reading or multi-process environment however appropriate measures have to be taken. These are not implemented in FECOM, FETCP, FEUSB or FEISC.

Nearly all the library functions have a return value which is negative in case of error.

5.2. Parameter transfer

Some functions support parameter transfer both as a null-terminated string and as an array of hex numbers. Transfer as data type UCHAR★ is possible for both data types. Interpretation of the transfer value is indicated by the function parameter *iDataType*.

iDataType	Parameter transfer	interpreted as a pointer to
0	0x23, 0x56, 0xFA, 0xA6 (internally 0x23 corresponds to the character "#"; 0x56 to the character "V"; etc.)	an array of UCHAR
1	"2356FAA6" (each two characters are interpreted as a hex value: Example: "23" -> 0x23)	a null-terminated string

All other parameters to be transferred as UCHAR must be given as a hex value (e.g. 0x23). It is not possible to transfer by strings!

Note: UCHAR is used as an abbreviation (#define) for „unsigned char“.

5.3. Asynchronous tasks for relieving the load on applications

A recurring task of applications is inventorying transponders in the antenna field of the reader. Ideally this should run in the background and then tell the application when transponders are in the field or when the notification has arrived.

This is precisely the functionality you can implement using the **FEISC_StartAsyncTask** function. Internally a thread is started which waits for the reply protocol of the reader and provides the reply data to the application using a callback function.

Asynchronous tasks are defined for two types of applications: for inventory in host mode or for receiving Buffered-Read-Mode data in Notification Mode.

Asynchronous tasks can be specified for multiple Readers at the same time as long as they were given their own object in the DLL using **FEISC_NewReader**. Readers on an RS485 bus are problematic. In this case you can only "monitor" one Reader at a time, since they are all connected on the same interface.

The features of the tasks are described in the table below:

Task	TaskID	Remarks
One-time Inventory	FEISC_TASKID_FIRST_NEW_TAG	A task can only started if the following option is integrated in the Reader's firmware: the Reader protocol [0xB0][0x01] Inventory must support an optional NOTIFY flag in its Mode byte. After receiving the Reader protocol within the specified time, the task automatically closes itself. If the time is exceeded, the callback function is invoked and the status 0x01 (No transponder in read field) send and the task ended. In case of error the task is always ended immediately and the callback function transmits the error code. Serial, USB and TCP/IP interfaces are supported, whereby the ports must be open before starting the task. Autonomous opening of the connection via TCP/IP by the Reader or a suitable converter for sending the data is not possible.
Repeating Inventory	FEISC_TASKID_EVERY_NEW_TAG	The same conditions as for one-time inventory apply, with the following difference: Repeating inventory defines a cyclical task which can only be cancelled by **FEISC_CancelAsyncTask**. A cycle corresponds to a one-time inventory and ends on a wait loop until the next cycle has been triggered by the application using **FEISC_TriggerAsyncTask**. Application-side triggering ensures that an application has time for receiving and processing the inventory data.

Receiving notifications	FEISC_TASKID_NOTIFICATION	A task should only be started if notification mode is integrated and activated in the Reader's firmware. Only TCP/IP communication is supported. Possible connection options are (see system manual for the Reader): - Temporary opening of the connection by the Reader for the duration of data transmission - Continuous opening of the connection by the Reader (in development) - Continuous opening of the connection by the host (in development) The task defines an endless task which can only be cancelled using **FEISC_CancelAsyncTask** or in case of error during the initialization phase is ended immediately after invoking the callback function. The task waits for reception of the Buffered-Read-Mode data and then invokes the callback function. After the callback function returns data can immediately be received again by the Reader. In case of transmission errors the callback function is invoked with the error code and the receiving procedure then resumed. Note: Depending on the Reader setting large quantities of data may be sent by the Reader in very short time intervals. Without use of a handshake procedure (see system manual for the Reader) data may be lost if the host is not appropriate for the quantity of notifications.
SAM communication	FEISC_TASKID_SAM_COMMAND	A single task for communication with a SAM (Security Application Module) inside an OBID® *classic-pro* Reader with SAM-Socket is executed with the function **FEISC_0xC0_SAMCmd**. After receiving the Reader protocol within the specified time, the task automatically closes itself. If the time is exceeded, the callback function is invoked with the error code -4082 (FEISC_ERR_TASK_TIMEOUT) and the task ended. In case of error the task is always ended immediately and the callback function transmits the error code. Serial and USB interfaces are supported, whereby the ports must be open before starting the task.
Command Queue	FEISC_TASKID_COMMAND_QUEUE	A single task for launching a [0xBC] Command Queue inside an OBID® *classic-pro* Reader is executed with the function **FEISC_0xBC_CmdQueue**. After receiving the Reader protocol within the specified time, the task automatically closes itself. If the time is exceeded, the callback function is invoked with the error code -4082 (FEISC_ERR_TASK_TIMEOUT) and the task ended. In case of error the task is always ended immediately and the callback function transmits the error code. Serial and USB interfaces are supported, whereby the ports must be open before starting the task.

The internal behavior is determined essentially by the structure **FEISC_TASK_INIT**, which is sent using **FEISC_StartAsyncTask**. Among other things it contains the necessary parameters for the callback function:

```
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
    UINT            uiFlag;             // specifies the use of the union (e.g. FEISC_TASKCB_1)
    union
    {
        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

        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
    }Method[5];

    uinion
    {
        int iNotifyWithAck;     // 0: notification without acknowledge
                                // 1: notification with acknowledge
    }InData[6]

} FEISC_TASK_INIT;
```

The core element of the structure is the *union* (method), which contains one or more function pointers. Selection of the callback function is handled by the parameter *uiFlag*. The parameter *pAny* can be used for any data and is returned in the first parameter of the callback function. C++ programmers can thus have a pointer for the invoking object sent to the static declared callback

[5] Naming of the union with Method is only for C programmers. C++ programmers access the union directly through the structure.

function and in this way access class functions. *uiTimeout* defines the timeout for an inventory cycle. The value depends on the specifications in the system manual for the reader for the protocol [0xB0][0x01] Inventory.

The structure variables *cClientIP* and *iPortAdr* are intended only for the Notification task. When using the TCP/IP channel for the inventory task the socket must already be opened before starting the asynchronous task.

5.4. Event flagging to applications[6]

Event handling mechanisms can be installed for some events. As soon as a send protocol for example is output over the interface, you can also notify the application of the event asynchronous to the program sequence. The application must already contain a corresponding function for this (s. 5.6.10. FEISC_AddEventHandler). These event handling mechanism must not mistake with the handling of events, triggered by starting of asynchronous tasks.

An event handling mechanism must be installed using the **FEISC_AddEventHandler** function. You may choose between five various flagging methods: Message to a calling process, message to a window use one of two possible callback function, or flagging with a Windows-API event.

An already installed event handling mechanism must be deleted using the **FEISC_DelEventHandler** function.

The structure **FEISC_EVENT_INIT** contains the parameters required for flagging:

```
typedef struct _FEISC_EVENT_INIT
{
    UINT uiUse;     // Defines the event (e.g. FEISC_PRT_EVENT)
    UINT uiMsg;     // Message Code for dwThreadID and hwndWnd (e.g. WM_USER_xyz)
    UINT uiFlag;    // Specifies use of the union (e.g. FEISC_WND_HWND)
    union
    {
        DWORD     dwThreadID;              // for Thread-ID
        HWND      hwndWnd;                 // for Window-Handle
        void      (*cbFct)(int, int);      // for first Callback-Function
        void      (*cbFct2)(BSTR, int, int); // for second Callback-Function
        HANDLE    hEvent;                  // for Event-Handle
    }Method[7];

} FEISC_EVENT_INIT;
```

The core element of the structure is the *union*, which contains either the ID of a process, the handle of a window, a function pointer or the handle of an Windows-API event. The *uiFlag* parameter is used to select the flag form. You use the *uiUse* parameter to store a designator for the event for assigning the handling method. To use the message methods you must store the message code in *uiMsg*.

You may install more than one handling mechanism for a single event. However, each *dwThreadID*, *hwndWnd*, *cbFct*, *cbFct2* or *hEvent* can be used only once per event.

[6] Can be used only with limitations for Linux C/C++ projects

[7] Naming of the union with method is only for C-programmers. C++ programmers access the union directly through the structure.

5.5. Integration of user-defined port drivers with the Plug-In mechanism[8]

This chapter is in preparation.

[8] Only for C/C++ projects

5.6. List of functions

The support package contains a large number of functions for various tasks. They are divided into groups for better orientation. Please note that most of the functions can be used only in conjunction (direct or indirect) with the ID FECOM or FETCP or ID FEUSB support package.

Administration functions for Reader Objects

- **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 **)**
- **int FEISC_InstallPlugIn(** int iReaderHnd, int iPlugInID, void* pPlugIn **)**
- **int FEISC_RemovePlugIn(** int iReaderHnd, int iPlugInID **)**

Protocol functions

- **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 **)**

Query functions

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

General communication functions

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

Special communication functions

- **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_0x21_ReadBuffer(** int iReaderHnd, UCHAR cBusAdr, UCHAR cSets, UCHAR* cTrData, UCHAR* cRecSets, UCHAR* cRecDataSets, int iDataType **)**
- **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 iDataType **)**
- **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_0x65_SoftVersion(** int iReaderHnd, UCHAR cBusAdr, UCHAR* cVersion, int iDataType **)**
- **int FEISC_0x66_ReaderInfo(** int iReaderHnd, UCHAR cBusAdr, UCHAR cMode, UCHAR* cVersion, int iDataType **)**
- **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 iDataType **)**
- **int FEISC_0x6D_GetNoiseLevel(** int iReaderHnd, UCHAR cBusAdr, UCHAR* cLevel, int iDataType **)**
- **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 iDataType **)**
- **int FEISC_0x80_ReadConfBlock(** int iReaderHnd, UCHAR cBusAdr, UCHAR cConfAdr, UCHAR* cConfBlock, int iDataType **)**
- **int FEISC_0x81_WriteConfBlock(** int iReaderHnd, UCHAR cBusAdr, UCHAR cConfAdr, UCHAR* cConfBlock, int iDataType **)**
- **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 iDataType **)**
- **int FEISC_0x86_GetSysTimer(** int iReaderHnd, UCHAR cBusAdr, UCHAR* cTime, int iDataType **)**
- **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* cReqData**)**
- **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_0xA0_RdLogin(** int iReaderHnd, UCHAR cBusAdr, UCHAR* cRd_PW, int iDataType **)**
- **int FEISC_0xA2_WriteMifareKeys(** int iReaderHnd, UCHAR cBusAdr, UCHAR cType, UCHAR cAdr, UCHAR* cKey, int iDataType **)**
- **int FEISC_0xB0_ISOCmd**(int iReaderHnd, UCHAR cBusAdr, UCHAR* cReqData, int iReqLen, UCHAR* cRspData, int* iRspLen, int iDataType **)**
- **int FEISC_0xB1_ISOCustAndPropCmd**(int iReaderHnd, UCHAR cBusAdr, UCHAR cMfr, UCHAR* cReqData, int iReqLen, UCHAR* cRspData, int* iRspLen, int iDataType **)**
- **int FEISC_0xB2_ISOCmd**(int iReaderHnd, UCHAR cBusAdr, UCHAR* cReqData, int iReqLen, UCHAR* cRspData, int* iRspLen, int iDataType **)**
- **int FEISC_0xB3_EPCCmd**(int iReaderHnd, UCHAR cBusAdr, UCHAR* cReqData, int iReqLen, UCHAR* cRspData, int* iRspLen, int iDataType **)**
- **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, UCHAR ucMode, 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 iDataType **)**
- **int FEISC_0xBE_ISOTranspCmd**(int iReaderHnd, UCHAR cBusAdr, int iMode, int iRspLength, UCHAR* cReqData, int iReqLen, UCHAR* cRspData, int* iRspLen, int iDataType **)**
- **int FEISC_0xBF_ISOTranspCmd**(int iReaderHnd, UCHAR cBusAdr, int iMode, int iRspLength, UCHAR* cReqData, int iReqLen, UCHAR* cRspData, int* iRspLen, int iDataType **)**
- **int FEISC_0xC0_SAMCmd**(int iReaderHnd, int iSlot, UCHAR* cReqData, int iReqLen, FEISC_TASK_INIT* pInit **)**

Special functions for asynchronous tasks

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

5.6.1. Which function for which OBID i-*scan*® and OBID® *classic-pro* Reader

For OBID classic-pro Reader please have a look to the system manuals which contains a reader command matrix.

Function	ISC.M01	ISC.M02	ISC.PR ISC.MR	ISC.PRH	ISC.LR	ISC.MRU	ISC.LRU	CPR
FEISC_0x18_Destroy		X	X	X	X			
FEISC_0x1A_Halt	X	X						
FEISC_0x1B_ResetQuietBit	X	X			X			
FEISC_0x1C_EASRequest	X	X			X			
FEISC_0x21_ReadBuffer					X			
FEISC_0x22_ReadBuffer						X	X	
FEISC_0x31_ReadDataBufferInfo					X	X	X	
FEISC_0x32_ClearDataBuffer					X	X	X	
FEISC_0x33_InitBuffer					X	X	X	
FEISC_0x34_ForceNotifyTrigger					X		X	
FEISC_0x52_GetBaud	X	X	X	X	X	X	X	X
FEISC_0x55_StartFlashLoader			X	X	X	X	X	X
FEISC_0x55_StartFlashLoaderEx		X	X	X	X	X	X	X
FEISC_0x63_CPUReset	X	X	X	X	X	X	X	X
FEISC_0x64_SystemReset						X	X	
FEISC_0x65_SoftVersion	X	X	X	X	X	X		X
FEISC_0x66_ReaderInfo						X	X	
FEISC_0x69_RFReset	X	X	X	X	X	X	X	X
FEISC_0x6A_RFOnOff	X	X	X	X	X	X	X	X
FEISC_0x6B_CentralizedRFSync						X	X	
FEISC_0x6C_SetNoiseLevel					X		X	
FEISC_0x6D_GetNoiseLevel					X		X	
FEISC_0x6E_RdDiag					X	X	X	
FEISC_0x6F_AntennaTuning					X			
FEISC_0x71_SetOutput	X	X	X	X	X	X	X	X
FEISC_0x72_SetOutput						X	X	
FEISC_0x74_ReadInput	X			X	X	X	X	X
FEISC_0x75_AdjAntenna	X							
FEISC_0x80_ReadConfBlock	X	X	X	X	X	X	X	X
FEISC_0x81_WriteConfBlock	X	X	X	X	X	X	X	X
FEISC_0x82_SaveConfBlock	X	X	X	X	X	X	X	X
FEISC_0x83_ResetConfBlock	X	X	X	X	X	X	X	X
FEISC_0x85_SetSysTimer					X			
FEISC_0x86_GetSysTimer					X			
FEISC_0x87_SetSystemDate						X		

Function	ISC.M01	ISC.M02	ISC.PR ISC.MR	ISC.PRH	ISC.LR	ISC.MRU	ISC.LRU	CPR
FEISC_0x88_GetSystemDate						X		
FEISC_0xA0_RdLogin					X	X		
FEISC_0xA2_WriteMifareKeys					X			
FEISC_0xB0_ISOCmd	X	X	X	X	X	X	X	X
FEISC_0xB1_ISOCustAndPropCmd		X						
FEISC_0xB2_ISOCmd		X	X	X	X			X
FEISC_0xB3_EPCCmd						X	X	
FEISC_0xB4_EPC_UHF_Cmd						X	X	
FEISC_0xBB_C1G2_TranspCmd						X	X	
FEISC_0xBC_CmdQueue								X
FEISC_0xBD_ISOTranspCmd								X
FEISC_0xBE_ISOTranspCmd						X		
FEISC_0xBF_ISOTranspCmd		X	X	X	X			X
FEISC_0xC0_SAMCmd								X

5.6.2. FEISC_NewReader

Function	Creates a Reader object.
Syntax	**int FEISC_NewReader(int iPortHnd)**
Description	A Reader object is created. Protocol functions require a Reader object in order to run. *iPortHnd*[9] is the handle of a port object created from FECOM using the **FECOM_OpenPort** function or a device object using the **FEUSB_OpenDevice** function or a TCP/IP socket object using the **FETCP_Connect** function. This handle allows protocols to be directly passed on to FECOM or FETCP or FEUSB. Transfer of a 0 is also permitted. Multiple Reader objects can in principle carry out their communication over the same serial COM port, the same TCP/IP socket or the same USB channel. *iPortHnd* uses the first byte (MSB) of the PortHandle to distinguish between protocol output to FECOM or FEUSB: *iPortHnd* = 0x0XXXXXXX[10] indicates output to FECOM.DLL/SO *iPortHnd* = 0x1XXXXXXX indicates output to FEUSB.DLL/SO *iPortHnd* = 0x2XXXXXXX indicates output to FETCP.DLL/SO You may change the value of the PortHandle stored in the Reader object after the fact using the **FEISC_SetReaderPara** function. A Reader object created with **FEISC_NewReader** must (!) be deleted from memory using the **FEISC_DeleteReader** function. Otherwise the memory reserved by the library is not freed up again.
Return value	If a Reader object was created without error, a handle (>0) is returned. In case of error, the function returns a value less than zero. A list of error codes can be found in the Appendix.
Example	... #include "feisc.h" #include "fecom.h" ... ... char cPortNr[4]; itoa(1, cPortNr, 10); // Convert Integer to Char int iPortHnd = FECOM_OpenPort(cPortNr); // COM:1 should be opened if(iPortHnd < 0) { // code here in case of error } else { // Open Reader object int iReaderHnd = FEISC_NewReader(iPortHnd); }

[9] iPortHnd is used in this document throughout to mean iDevHnd or iSocketHnd as well

[10] X represents any hex value

5.6.3. FEISC_DeleteReader

Function	Deletes a Reader object
Syntax	**int FEISC_DeleteReader(int iReaderHnd)**
Description	The function deletes the Reader object indicated by the parameter *iReaderHnd* and frees up the reserved memory.
Return value	The return value is 0 if the action was successful. In case of error, the function returns a value less than zero. A list of error codes can be found in the Appendix.
Example	... #include "feisc.h" ... ... int iErr; int iReaderHnd = FEISC_NewReader(0); if(iReaderHnd < 0) { // code here in case of error } ... ... if(iReaderHnd > 0) { iErr = FEISC_DeleteReader(iReaderHnd); ... } ... ...

5.6.4. FEISC_GetReaderList

Function	Depending on the *iNext* parameter, gets the first or following Reader handle from the internal list of the generated Reader objects.
Syntax	**int FEISC_GetReaderList(int iNext)**
Description	The function returns a Reader handle from the internal list of Reader handles. If one transmits a 0 for *iNext*, the first entry in the list is returned. If you transmit a Reader handle contained in the list with *iNext*, the function gets and returns the entry following the Reader handle. In this way you can keep incrementing the return value to go through the list and call out all the entries.
Return value	When an entry is found, the Reader handle is provided with the return value. When the end of the internal list is reached, in other words the transferred Reader handle has no following entry, a 0 is returned. If there is no Reader object, FEISC_ERR_EMPTY_LIST is returned. In case of error, the function returns a value less than zero. A list of error codes can be found in the Appendix.
Example	... #include "feisc.h" ... // Example function for creating a list of Reader objects void ReaderList(void) { int iNextHnd = FEISC_GetReaderList(0); // get the first handle while(iNextHnd > 0) { // here for example code for collecting the handles and reading out parameters ... iNextHnd = FEISC_GetReaderList(iNextHnd); // get next handle } ... // here for example code for displaying a list }
Tip	When closing all open created Reader objects it is convenient to use a loop such as in the example above. Bear in mind however than you cannot get the next in line from a deleted Reader object. The following code fragment gives you an idea of how to delete all created Reader objects in a loop: ... int iNextHnd, iCloseHnd, iError; iNextHnd = FEISC_GetReaderList(0); // get first handle while(iNextHnd > 0) { iCloseHnd = iNextHnd; iNextHnd = FEISC_GetReaderList(iNextHnd); // get next handle iError = FEISC_DeleteReader(iCloseHnd); // only now delete Reader object }

5.6.5. FEISC_GetDLLVersion

Function	Gets the DLL/SO version number.
Syntax	**void FEISC_GetDLLVersion(char* cVersion)**
Description	The function returns the version number of the DLL/SO. *cVersion* is an empty, null-terminated string for returning the version number. The string should be able to hold at least 256 characters. In the current version the string is filled with „05.07.05“. Newer versions may provide additional information.
Return value	none
Example	... #include "feisc.h" ... ... char cVersion[256]; FEISC_GetDLLVersion(cVersion); // code here for displaying the version number ... ...

5.6.6. FEISC_GetErrorText

Function	Gets error text for error code
Syntax	**int FEISC_GetErrorText(int iErrorCode, char* cErrorText)**
Description	This function uses *cErrorText* to send a short error text associated with the *iErrorCode*. The buffer for *cErrorText* should be able to hold at least 256 characters.
Return value	If there is no error the function returns zero, and if error a value less than zero. The list of error codes can be found in the Appendix.
Example	... #include "feisc.h" ... ... char cErrorText[256]; ... int iBack = FEISC_GetErrorText(FEISC_ERR_PROTLEN, cErrorText) // code here for displaying the text ... ...

5.6.7. FEISC_GetStatusText

Function	Gets a short text for status byte
Syntax	**int FEISC_GetStatusText(UCHAR ucStatus, char* cStatusText)**
Description	This function uses *cStatusText* to send a short text associated with the *ucStatus*. The buffer for *cStatusText* should be able to hold at least 128 characters.
Return value	If there is no error the function returns zero, and if error a value less than zero. The list of error codes can be found in the Appendix.
Example	... #include "feisc.h" ... ... char cStatusText[128]; ... int iBack = FEISC_GetStatusText(0x01, cStatusText) // code here for displaying the text ... ...

5.6.8. FEISC_GetReaderPara

Function	Gets a parameter from a Reader object
Syntax	**int FEISC_GetReaderPara(int iReaderHnd, char* cPara, char* cValue)**
Description	The function gets the current value of a parameter. *cPara* is a null-terminated string with the variable. *cValue* is an empty, null-terminated string for returning the parameter value. The string should be able to hold at least 128 characters. *iReaderHnd* is the handle for the Reader object.
Variables	The variables are: PortHnd[11], LogProt, LogFile, LogFilename, RecBusAdr, Language, ChkRecBusAdr, ConvHexToString, SendStr, RecStr, IsProtToAppLocked and FrameSupport
Cross-reference	For more information see: 5.7. Support for multithreading and 6.2. List of variables
Return value	If no error the function returns a value of 0, and in case of error a value less than zero. A list of error codes can be found in the Appendix.
Example	... #include "feisc.h" ... ... char cValue[128]; int iPortHnd; ... if(!FEISC_GetReaderPara(handle, "PortHnd", cValue)) { // Convert Char to Integer iPortHnd = atoi(cValue); // here for example code for using the PortHandle ... } ... ... }

[11] Note here the remarks concerning the PortHandle in 5.6.2. FEISC_NewReader

5.6.9. FEISC_SetReaderPara

Function	Sets a Reader object parameter to a new value.
Syntax	**int FEISC_SetReaderPara(int iReaderHnd, char* cPara, char* cValue)**
Description	The function gives a new parameter to a Reader object. The Reader object stores the new value and immediately turns it into the current parameter. *cPara* is a null-terminated string with the variable. *cValue* is a null-terminated string with the new parameter value. *iReaderHnd* is the handle for the Reader object.
Variables	The variables are: PortHnd[12], LogProt, LogFile, LogFilename, Language, ChkRecBusAdr, ConvHexToString, LockProtToApp, UnlockProtToApp and FrameSupport
Cross-reference	For more information see: 5.7. Support for multithreading and 6.2. List of variables
Return value	If the Reader object with the new parameter value was successfully (error-free) installed, a 0 is returned. In case of error, the function returns a value less than zero. A list of error codes can be found in the Appendix.
Example	// the example shows that a new PortHandle can be assigned to a Reader object after the fact. // after this assignment, communication is through the new port . ... #include "feisc.h" #include "fecom.h" ... ... int iErr; char cPortHnd[9]; char cPortNr[4]; itoa(1, cPortNr, 10); // Convert Integer to Char ... int iPortHnd = FECOM_OpenPort(cPortNr); // COM:1 should be opened if(iPortHnd > 0) { itoa(iPortHnd, cPortHnd, 10); // Convert Integer to Char iErr = FEISC_SetReaderPara(iReaderHnd, "PortHnd", cPortHnd); // from here on communication through the new port is possible ... } ... ...

[12] Note here the remarks concerning the PortHandle in 5.6.2. FEISC_NewReader

5.6.10. FEISC_AddEventHandler

<table>
<tr><td>Function</td><td>Installs an event handling mechanism</td></tr>
<tr><td>Syntax</td><td>int FEISC_AddEventHandler(int iReaderHnd, FEISC_EVENT_INIT* pInit)</td></tr>
<tr><td>Description</td><td>
The function installs one of four possible event handling methods. This method is used when an event occurs for which the method was installed. This allows asynchronous response to events in an application program.

The event handling method is established only for the port identified by iReaderHnd. This means that if necessary you may have to repeat this installation for each Reader object.
<table>
<tr><td>Event</td><td>Description</td></tr>
<tr><td>FEISC_PRT_EVENT</td><td>One event each for the send and receive protocol [13]</td></tr>
<tr><td>FEISC_SNDPRT_EVENT</td><td>Event for send protocol [10]</td></tr>
<tr><td>FEISC_RECPRT_EVENT</td><td>Event for receive protocol [10]</td></tr>
<tr><td>FEISC_SCANNER_EVENT</td><td>Event for received protocol when reader in scan mode[14]
(no support in Linux)</td></tr>
</table>
<u>1st Method: Message to thread (not for Linux)</u>
This method is used for exchanging messages between threads [15]. The thread uses the Windows-API function GetCurrentThreadID() to get the thread identifier and transfers this as the parameter dwThreadID in the FEISC_EVEN_INIT structure.
The thread must provide a message handling function for receiving the message that was sent by FEISC with the Windows-API function PostThreadMessage(..). The message code is freely selectable.
The FEISC_EVENT_INIT structure is filled as follows:
uiFlag = FEISC_THREAD_ID
uiUse = FEISC_xyz_EVENT // see Defines FEISC.H
uiMsg = WM_USER + ... // freely selectable, but higher than WM_USER [16]
dwThreadID = GetCurrentThreadID()
The MessageMap function in the application is given in the 1st parameter (WPARAM) the pointer to the string and in the 2nd parameter the status byte of the receive protocol. Note that the string pointer is cast with int, so that it needs to be converted back using the cast operator (LPCTSTR) when allocating to a CString data type or (char*) when allocating to a C-String.

<u>2nd Method: Message to window (not for Linux)</u>
This method is used when the message needs to be sent directly to a window. The corresponding window uses the Windows-API function GetWindow (..)[17] to get the
</td></tr>
</table>

[13] Event is only generated if the parameter LogProt is set to 1 (default: 0)
[14] See description to parameter ConvHexToString in: 6.2. List of variables
[15] Parallel execution path independent of the application program. The application program itself is a thread.
[16] See Windows documentation for the SDK platform
[17] When using MFC CWnd you can also use the GetSafeHwnd() method

handle and transfer it as the parameter hwndWnd in the **FEISC_EVENT_INIT** structure. The window must provide a message handling function for receiving the message that was sent by FEISC with the Windows-API function PostMessage(..). The message code is freely selectable.

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

```
uiFlag = FEISC_WND_HWND
uiUse = FEISC_xyz_EVENT          // see Defines FEISC.H
uiMsg = WM_USER + ...            // freely selectable, but higher than WM_USER [18]
hwndWnd = GetWindow(...)
```

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

3rd method: Invoking the first callback function

The first callback method installs a function pointer for an event. When the event occurs, FEISC calls the function. The contents of the function can be freely determined. The transfer parameters are described above for the 1st method.

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

```
uiUse = FEISC_xyz_EVENT          // see Defines FEISC.H
uiMsg not needed
uiFlag = FEISC_CALLBACK
cbFct = (void*)&YourFunctionName[19]
```

4th method: Invoking the second callback function (not for Linux)

The first callback method installs a function pointer for an event. When the event occurs, FEISC calls the function. The contents of the function can be freely determined. The transfer parameters are as follows:

```
BSTR - pointer to a Unicode string
int  - number of characters in string
int  - statusbyte or errorcode
```

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

```
uiUse = FEISC_xyz_EVENT          // see Defines FEISC.H
uiMsg not needed
uiFlag = FEISC_CALLBACK_2
cbFct = (void*)&YourFunctionName[20]
```

5th method: Setting an event (not for Linux)

With the event method an event handle is installed for an event. When an event occurs, FEISC sets the event with the Windows-API function SetEvent(…). On the application side you wait for the event with the Windows-API function WaitForSingleObject(…). Since no parameters can be received, you must query the desired parameter with an appropriate function. The set event must be reset again by the application program with the Windows-API function ResetEvent(…).

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

```
uiUse = FEISC_xyz_EVENT          // see Defines FEISC.H
uiMsg not needed
uiFlag = FEISC_EVENT
```

[18] See Windows documentation for the SDK platform

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

[20] The function has the prototype: void YourFunctionName(BSTR, int, int)

	hEvent = CreateEvent(..) An installed event handling method can only be deleted using the function **FEISC_DelEventHandler**. When removing a Reader object, all event handling methods installed for that object are lost.
Cross-reference	For more information see: 5.4. Event flagging to applications and 5.7. Support for multithreading
Return value	If no error the function returns zero, and in case of error a value less than zero. A list of error codes can be found in the Appendix.

5.6.11. FEISC_DelEventHandler

Function	Deletes an event handling mechanism
Syntax	**int FEISC_DelEventHandler(int iReaderHnd, FEISC_EVENT_INIT* pInit)**
Description	The function deletes an event handling mechanism which was previously installed using **FEISC_AddEventHandler**. The **FEISC_EVENT_INIT** structure is where you specify in detail the event handling mechanism to be deleted. Deleting the 1st method: Message to Thread (not for Linux) The **FEISC_EVENT_INIT** structure is filled as follows: uiFlag = FEISC_THREAD_ID uiUse = FEISC_xyz_EVENT // see Defines in FEISC.H uiMsg is not needed dwThreadID = GetCurrentThreadID() Deleting the 2nd method: Message to Window (not for Linux) The **FEISC_EVENT_INIT** structure is filled as follows: uiFlag = FEISC_WND_HWND uiUse = FEISC_xyz_EVENT // see Defines in FEISC.H uiMsg is not needed hwndWnd = GetWindow(...) Deleting the 3rd method: Invoking the first callback function The **FEISC_EVENT_INIT** structure is filled as follows: uiFlag = FEISC_CALLBACK uiUse = FEISC_xyz_EVENT // see Defines FEISC.H uiMsg is not needed cbFct = (void*)&YourFunctionName Deleting the 4th method: Invoking the second callback function (not for Linux) The **FEISC_EVENT_INIT** structure is filled as follows: uiFlag = FEISC_CALLBACK_2 uiUse = FEISC_xyz_EVENT // see Defines FEISC.H uiMsg is not needed cbFct = (void*)&YourFunctionName Deleting the 5th method: Setting an event (not for Linux) The **FEISC_EVENT_INIT** structure is filled as follows: uiFlag = FEISC_EVENT uiUse = FEISC_xyz_EVENT // see Defines FEISC.H uiMsg is not needed hEvent = hYourEventHandle
Cross-reference	For more information see: 5.4. Event flagging to applications and5.7. Support for multithreading.
Return value	If no error the function returns zero, and in case of error a value less than zero. A list of error codes can be found in the Appendix.

5.6.12. FEISC_StartAsyncTask

Function	An inventory or notification task is started asynchronous to the application	
Syntax	**int FEISC_StartAsyncTask(int iReaderHnd, int iTaskID, FEISC_TASK_INIT* pInit, void* pInput)**	
Description	This function starts an asynchronous task. An asynchronous task is an internal thread which e.g. sends an inventory command to the reader and waits for the reply for a time up to the timeout. Signaling of the reply data or the cancel condition to the application is done by invoking a callback function. The task behavior is specified in the parameter *iTaskID*. Three tasks are currently defined:	
	FEISC_TASKID_FIRST_NEW_TAG	starts a one-time inventory task
	FEISC_TASKID_EVERY_NEW_TAG	starts a repeating inventory task
	FEISC_TASKID_NOTIFICATION	starts a task prepared for receiving notifications
	All the data relevant to the callback function are contained in the structure FEISC_TASK_INIT. This structure is described in greater detail in section 5.3. Asynchronous tasks for relieving the load on applications. The last parameter *pInput* is not currently considered. You should always send NULL (vbNull). *iReaderHnd* is the handle for the reader object.	
Cross-references	Additional information about asynchronous tasks can be found in the section 5.3. Asynchronous tasks for relieving the load on applications. 5.6.13. FEISC_CancelAsyncTask 5.6.14. FEISC_TriggerAsyncTask	
Note	Asynchronous inventory tasks use protocol [0xB0][0x01] Inventory with the NOTIFY option in the mode byte. Readers not supporting this option can not be used for asynchronous tasks. More detailed information about the protocol [0xB0][0x01] Inventory can be found in the manual for the OBID *i-scan*® or OBID® *classic-pro* Reader family.	
Return value	In case of no error a 0 is returned. A value less than 0 indicates an error. The list of error codes can be found in the Appendix.	

5.6.13. FEISC_CancelAsyncTask

Function	Cancels an inventory or notification task.
Syntax	**int FEISC_CancelAsyncTask(int iReaderHnd)**
Description	This function cancels an asynchronous task. You should not normally use one-time inventory (started with TaskID = FEISC_TASKID_FIRST_NEW_TAG) to quit this function. You should end repeating inventory (started with TaskID = FEISC_TASKID_EVERY_NEW_TAG) using this function if the callback function was ended and the internal thread is waiting for the next trigger. This ensures that the task in the Reader is ended and it can again process reader tasks. Notification tasks must always be canceled with thuis function. The cancellation of the task is locked if the task execution is just inside the callback function. This prevents deadlocks. In this case this funktion returns directly with the return value FEISC_ERR_TASK_BUSY (-4084) and the application must invoke FEISC_CancelAsyncTask again. *iReaderHnd* is the handle for the reader object.
Cross-references	Additional information about asynchronous tasks can be found in the section 5.3. Asynchronous tasks for relieving the load on applications. 5.6.12. FEISC_StartAsyncTask 5.6.14. FEISC_TriggerAsyncTask
Return value	In case of no error a 0 is returned. A value less than 0 indicates an error. The list of error codes can be found in the Appendix.

5.6.14. FEISC_TriggerAsyncTask

Function	Triggers the next cycle in the inventory task.
Syntax	**int FEISC_TriggerAsyncTask(int iReaderHnd)**
Description	This function is used to trigger the next inventory cycle in the asynchronous task. The asynchronous task must have been previously started with the TaskID = FEISC_TASKID_EVERY_NEW_TAG. This function is always invoked after the callback function has been exited. Without this invoke a task with repeating function hangs up in a wait loop. *iReaderHnd* is the handle for the reader object.
Cross-references	Additional information about asynchronous tasks can be found in the section 5.3. Asynchronous tasks for relieving the load on applications. 5.6.12. FEISC_StartAsyncTask 5.6.13. FEISC_CancelAsyncTask
Return value	In case of no error a 0 is returned. A value less than 0 indicates an error. The list of error codes can be found in the Appendix.

5.6.15. FEISC_BuildSendProtocol

Function	The transmitted parameters and data are used to build a send protocol with a protocol frame.
Syntax	**int FEISC_BuildSendProtocol(int iReaderHnd, UCHAR cBusAdr, UCHAR cCmdByte, UCHAR* cSendData, int iDataLen, UCHAR* cSendProt, int iDataType)**
Description	This function uses the transmitted parameters bus address (*cBusAdr*), command byte (*cCmdByte*), send data (*cSendData*) and the information about the length of the send data (*iDataLen*) to build a complete send protocol with protocol frame. The protocol string is stored in *cSendProt* as a hex array (*iDataType=0*) or string (*iDataType=1*). The buffer for *cSendProt* must be longer by a factor of one than the expected protocol length, since a NUL character is appended. For more information about the protocol frame, see the system manual for the ISC Reader family. *iReaderHnd* is the handle for the Reader object. The constructed protocol is not passed along to a port driver (like FECOM).
Cross-reference	For more information on *iDataType* see Section 5.2. Parameter transfer
Note	This function does not yet support the USB protocols.
Return value	In case of no errors the length of *cSendProt* is indicated in the return value. In case of errors a negative value is returned. A list of error codes can be found in the Appendix.
Example	... 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; // Build send protocol iErr = FEISC_BuildProtocol(iReaderHnd, cBusAdr, cCmdByte, cSendData, iDataLen, cSendProt, 0); ... }

5.6.16. FEISC_BuildRecProtocol

Function	The transmitted parameters and data are used to build a receive protocol with a protocol frame.
Syntax	**int FEISC_BuildRecProtocol(int iReaderHnd, UCHAR cBusAdr, UCHAR cCmdByte, UCHAR cStatus, UCHAR* cRecData, int iDataLen, UCHAR* cRecProt, int iDataFormat)**
Description	This function uses the transmitted parameters bus address (*cBusAdr*), command byte (*cCmdByte*), status byte (*cStatus*), receive data (*cRecData*) and the information about the length of the receive data (*iDataLen*) to build a complete receive protocol with protocol frame. The protocol string is stored in *cRecProt* as a hex array (*iDataType=0*) or string (*iDataType=1*). The buffer for *cRecProt* must be longer by a factor of one than the expected protocol length, since a NUL character is appended. For more information about the protocol frame, see the system manual for the OBID *i-scan*® or OBID® *classic-pro* Reader family. *iReaderHnd* is the handle for the Reader object. The constructed protocol is not passed along to a port driver (like FECOM).
Cross-reference	For more information on *iDataType* see Section 5.2. Parameter transfer
Note	This function does not yet support the USB protocols.
Return value	In case of no errors the length of *cRecProt* is indicated in the return value. In case of errors a negative value is returned. A list of error codes can be found in the Appendix.
Example	Ananlog zu FEISC_BuildSendProt

5.6.17. FEISC_SplitSendProtocol

Function	Splits the transmitted protocol string.
Syntax	**int FEISC_SplitSendProtocol(int iReaderHnd, UCHAR* cSendProt, int iSendLen, UCHAR* cBusAdr, UCHAR* cCmdByte, UCHAR* cSendData, int* iDataLen, int iDataFormat)**
Description	This function splits the data contained in *cSendProt* into bus address (*cBusAdr*), command byte (*cCmdByte*), send data (*cSendData*) and the information about the length of the send data (*iDataLen*). The protocol string in *cSendProt* must be transmitted as a hex array (*iDataType=0*) or string (*iDataType=1*) with a length indication in *iSendLen*. *cSendData* is interpreted as a hex array (*iDataType*=0) or string (*iDataType*=1). For more information about the protocol frame, see the system manual for the OBID *i-scan®* or OBID® *classic-pro* Reader family. *iReaderHnd* is the handle for the Reader object. This function depends not of a port driver (like FECOM).
Cross-reference	For more information on *iDataType* see Section 5.2. Parameter transfer
Note	This function does not yet support the USB protocols.
Return value	If no error the function returns zero, and in case of error a value less than zero. A list of error codes can be found in the Appendix.
Example	Analog to FEISC_SplitRecProt

5.6.18. FEISC_SplitRecProtocol

Function	Splits the transmitted protocol string.
Syntax	**int FEISC_SplitRecProtocol(int iReaderHnd, UCHAR* cRecProt, int iRecLen, UCHAR* cBusAdr, UCHAR* cCmdByte, UCHAR* cRecData, int* iDataLen, int iDataType)**
Description	This function splits the data contained in *cRecProt* into bus address (*cBusAdr*), command byte (*cCmdByte*), receive data (*cRecData*) and the information about the length of the receive data (*iDataLen*). The protocol string in *cRecProt* must be transmitted as a hex array (*iDataType=0*) or string (*iDataType=1*) with a length indication in *iRecLen*. *cRecData* is interpreted as a hex array (*iDataType*=0) or string (*iDataType*=1). For more information about the protocol frame, see the system manual for the ISC Reader family. *iReaderHnd* is the handle for the Reader object. This function depends not of a port driver (like FECOM).
Cross-reference	For more information on *iDataType* see Section 5.2. Parameter transfer
Note	This function does not yet support the USB protocols.
Return value	In case of no errors the status byte of the receive protocol is returned. A value greater than 0x00 indicates an exception condition for the reader. A list of error codes can be found in the Appendix.
Example	// the following code fragment presupposes initialized port and Reader objects. #include "feisc.h" #include "fecom.h" ... int iStatus, iRecLen; UCHAR cBusAdr, cCmdByte; UCHAR cSendProt[256], cRecProt[256], cRecData[256]; int iDataLen = 0; // Build send protocol FEISC_BuildProtocol(iReaderHnd, cBusAdr, cCmdByte, cSendData, cDataLen, cSendProt, 0); // Send and receive protocol iRecLen = FECOM_Transceive(iPortHnd, cSendProt, cSendProt[0], cRecProt, 256); if(iRecLen > 0) { // Split receive protocol iStatus = FEISC_SplitProtocol(iReaderHnd, cRecProt, iRecLen, &cBusAdr, &cCmdByte, cRecData, &iDataLen, 0); if(iStatus == 0) // Statusbyte == 0x00 { // Process receive data ... } }

5.6.19. FEISC_SendTransparent

Function	Outputs a protocol string directly over the interface; the receive protocol is returned.
Syntax	**int FEISC_SendTransparent(int iReaderHnd, UCHAR* cSendProt, int iSendLen, UCHAR* cRecProt, int iMaxRecLen, int iCheckSum, int iDataType)**
Description	This function can be used to send protocol strings created using editors to a Reader. This presupposes thorough knowledge of protocol frames. The protocol with protocol frame contained in *cSendProt* is optionally expanded with the checksum (*iCheckSum = 1*) and the receive protocol is stored in *cRecProt*. Both buffers should be interpreted as hex array (*iDataType=0*) or string (*iDataType=1*). The length of the protocol (number of characters in *cSendProt*) must be indicated in the *iSendLen* parameter. The receive protocol buffer should as a precaution be able to hold 256 characters (*iDataType*=0) or 512 characters (*iDataType*=1). This buffer size must be indicated in *iMaxRecLen*. The buffer must be increased for Advanced Protocol Frames. *iReaderHnd* is the handle for the Reader object.
Cross-reference	For more information on *iDataType* see Section 5.2. Parameter transfer
Note	This function supports all FEIG port drivers.
Return value	In case of no errors the number of characters contained in *cRecProt* is sent. A list of error codes can be found in the Appendix.
Example	int outLen, inLen; UCHAR cSendProt[256]; UCHAR cRecProt[256]; ... // Define send protocol cSendProt[0] = 0x05; // Length byte cSendProt[1] = 0xFF; // Address byte cSendProt[2] = 0x80; // Control byte cSendProt[3] = 0x00; // Configuration address in Reader cSendProt[4] = 0x00; // Placeholder for checksum cSendProt[5] = '\0'; outLen = 5; ... // Send protocol, first calculating and appending checksum inLen = FEISC_SendTransparent(iReaderHnd, cSendProt, outLen, cRecProt, 256, 1, 0); if(inLen > 0) { // starting here code for processing the receive data ... }

5.6.20. FEISC_Transmit

Function	Outputs a protocol string directly over the interface.
Syntax	**int FEISC_Transmit(int iReaderHnd, UCHAR* cSendProt, int iSendLen, int iCheckSum, int iDataType)**
Description	This function can be used to send protocol string created using editors to a Reader. This presupposes thorough knowledge of protocol frames. There is no waiting for a reply protocol after sending the *cSendProt* protocol. The protocol with protocol frame contained in *cSendProt* is optionally expanded with the checksum (*iCheckSum = 1*) and the receive protocol is stored in *cRecProt*. Both buffers should be interpreted as hex array (*iDataType=0*) or string (*iDataType=1*). The length of the protocol (number of characters in *cSendProt*) must be indicated in the *iSendLen* parameter. If *iDataType*=1, then *iSendLen* is twice as large as in the case of *iDataType*=0. *iReaderHnd* is the handle for the Reader object.
Cross-reference	For more information on *iDataType* see Section 5.2. Parameter transfer
Note	This function supports all FEIG port drivers.
Return value	In case of error a 0 is transferred. A list of error codes can be found in the Appendix.
Example	int outLen; UCHAR cSendProt[256]; ... // Define send protocol cSendProt[0] = 0x05; // Length byte cSendProt[1] = 0xFF; // Address byte cSendProt[2] = 0x80; // Command byte for Read Configuration cSendProt[3] = 0x00; // Configuration address in Reader cSendProt[4] = '\0'; outLen = 4; // Send protocol, first calculating and appending checksum FEISC_Transmit(iReaderHnd, cSendProt, outLen, 1, 0); ...

5.6.21. FEISC_Receive

Function	Receives a protocol string directly from the interface.
Syntax	**int FEISC_Receive(int iReaderHnd, UCHAR* cRecProt, int iRecLen, int iDataType)**
Description	This function reads a protocol directly out of the receive buffer and stores it in *cRecProt*. If an ISC Reader has already send several protocols, the function reads in all the protocols. In this case *cRecProt* contains all protocols. A maximum of 256 ASCII characters can be taken from the receive buffer. The receive protocol buffer should as a precaution be able to hold 256 characters (*iDataType*=0) or 512 characters (*iDataType*=1). This buffer size must be indicated in *iRecLen*. The buffer must be increased for Advanced Protocol Frames. *iReaderHnd* is the handle for the Reader object.
Cross-reference	For more information on *iDataType* see Section 5.2. Parameter transfer
Note	This function supports all FEIG port drivers.
Return value	In case of no errors the number of characters contained in cRecProt is transmitted. If *iDataType*=1, then *iSendLen* is twice as large as in the case of *iDataType*=0. A list of error codes can be found in the Appendix.
Example	int inLen; UCHAR cRecProt[256]; ... // Receive protocol inLen = FEISC_Receive(iReaderHnd, cRecProt, 256, 0); ...

5.6.22. FEISC_GetLastSendProt

Function	Returns the last send protocol string.
Syntax	**int FEISC_GetLastSendProt(int iReaderHnd, UCHAR* cSendProt, int iDataType)**
Description	This function can be used to get the last sent send protocol from a Reader object. All functions which begin with **FEISC_0x…** as well as the function **FEISC_SendTransparent** store this protocol in the Reader object. The send protocol buffer *cSendProt* should as a precaution be able to hold 256 characters (*iDataType*=0) or 512 characters (*iDataType*=1). *cSendProt* should be interpreted as a hex array (*iDataType=0*) or string (*iDataType=1*). The buffer must be increased for Advanced Protocol Frames. *iReaderHnd* is the handle for the Reader object.
Cross-reference	For more information on *iDataType* see Section 5.2. Parameter transfer
Return value	In case of no errors the return value contains the number of characters contained in *cSendProt*. A list of error codes can be found in the Appendix.

5.6.23. FEISC_GetLastRecProt

Function	Returns the last received protocol string.
Syntax	**int FEISC_GetLastRecProt(int iReaderHnd, UCHAR* cRecProt, int iDataType)**
Description	This function can be used to get the last receive protocol from a Reader object. All functions which begin with **FEISC_0x…** as well as the function **FEISC_SendTransparent** store this protocol in the Reader object. The receive protocol buffer *cRecProt* should as a precaution be able to hold 256 characters (*iDataType*=0) or 512 characters (*iDataType*=1). *cRecProt* should be interpreted as a hex array (*iDataType=0*) or string (*iDataType=1*). The buffer must be increased for Advanced Protocol Frames. *iReaderHnd* is the handle for the Reader object.
Cross-reference	For more information on *iDataType* see Section 5.2. Parameter transfer
Return value	In case of no errors the return value contains the number of characters contained in *cRecProt*. A list of error codes can be found in the Appendix.

5.6.24. FEISC_GetLastState

Function	Returns the status byte contained in the last receive protocol.
Syntax	**int FEISC_GetLastStatus(int iReaderHnd, char* cStatusText)**
Description	This function can be used to get the status byte from a Reader object and a short text for the status byte of the last receive protocol. All functions which begin with **FEISC_0x…** as well as the function **FEISC_SendTransparent** store this protocol in the Reader object. The buffer for the short text *cStateText* should be able to hold at least 256 characters. *iReaderHnd* is the handle for the Reader object.
Return value	In case of no errors the return value contains the status byte. A list of error codes can be found in the Appendix.

5.6.25. FEISC_GetLastRecProtLen

Function	Gets the length of the last receive protocol.
Syntax	**int FEISC_GetLastRecProtLen(int iReaderHnd)**
Description	Sometimes it is helpful to be able to get the length of the data contained in it from the protocol length. This protocol length is what this function gets. Example: The function **FEISC_0x21_ReadBuffer** provides some data records for a data structure. You could get the total length of the data by analyzing the data sets, but it is much simpler to use the protocol length and deduct 6 bytes for the protocol frame and another 2 bytes for the parameters *cTrData* and *cRecDataSets*. *iReaderHnd* is the handle for the Reader object.
Return value	In case of no errors the return value contains the protocol length. A list of error codes can be found in the Appendix.

5.6.26. FEISC_GetLastError

Function	Gets the last error code and transmits error text.
Syntax	**int FEISC_GetLastError(int iReaderHnd , int* iErrorCode, char* cErrorText)**
Description	The function uses *iErrorCode* to send the last error code of the Reader object selected with*iReaderHnd* and transmits the associated error text in *cErrorText*. The buffer for *cErrorText* should be able to hold at least 256 characters.
Return value	If no error the function returns zero, and in case of error a value less than zero. A list of error codes can be found in the Appendix.
Example	... #include "feisc.h" ... ... char cErrorText[256]; int iErrorCode = 0; ... int iBack = FEISC_GetLastError(iReaderHnd, &iErrorCode, cErrorText) // code here for displaying the text ... ...

5.6.27. FEISC_0x18_Destroy

Function	Function destroys an Transponder.
Syntax	**int FEISC_0x18_Destroy(int iReaderHnd, UCHAR cBusAdr, UCHAR cMode, UCHAR* cEPC, UCHAR* cPW)**
Note	This function will render an Transponder permanently unable to give any replies. *cMode* is the mode byte. *cEPC* is a pointer to the buffer with the EPC or UID. The length of the EPC or UID is calculated internally based on the mode byte and the EPC header. *cPW* is a pointer to the buffer with the 3 byte password. *iReaderHnd* ist der Handle zum Leser-Objekt. *cBusAdr* ist die im multijob-Leser eingestellte Busadresse.
Return value	If there was no error, the return value contains the status byte of the reply protocol. A list of error codes can be found in the Appendix.

5.6.28. FEISC_0x1A_Halt

Function	Function for turning off transponders.
Syntax	**int FEISC_0x1A_Halt(int iReaderHnd, UCHAR cBusAdr)**
Description	This function turns off a previously selected transponder. The **FEISC_0x69_RFReset** function can be used to reactivate all the transponders which are turned off. *iReaderHnd* is the handle for the Reader object. *cBusAdr* is the bus address set in the Reader.
Return value	If there was no error, the return value contains the status byte of the reply protocol. A list of error codes can be found in the Appendix.

5.6.29. FEISC_0x1B_ResetQuietBit

Function	Function for resetting the Quiet bit.
Syntax	**int FEISC_0x1B_ResetQuietBit(int iReaderHnd, UCHAR cBusAdr)**
Description	The function resets the Quiet bit in the transponder Type I-Code. *iReaderHnd* is the handle for the Reader object. *cBusAdr* is the bus address set in the Reader.
Return value	If there was no error, the return value contains the status byte of the reply protocol. A list of error codes can be found in the Appendix.

5.6.30. FEISC_0x1C_EASRequest

Function	Function for sending the EAS Request
Syntax	**int FEISC_0x1C_EASRequest(int iReaderHnd, UCHAR cBusAdr)**
Description	The function sends an EAS Request to the transponder Type I-Code. *iReaderHnd* is the handle for the Reader object. *cBusAdr* is the bus address set in the Reader.
Return value	If there was no error, the return value contains the status byte of the reply protocol. A list of error codes can be found in the Appendix.

5.6.31. FEISC_0x21_ReadBuffer

Function	Function for data transfer with a transponder
Syntax	**int FEISC_0x21_ReadBuffer(int iReaderHnd, UCHAR cBusAdr, UCHAR cSets, UCHAR* cTrData, UCHAR* cRecSets, UCHAR* cRecDataSets, int iDataType)**
Description	The function reads the number of data sets *cSets* from the internal data table and stores the data in *cRecDataSets*. *cTrData* defines the structure of a data set in *cRecDataSets*. The number of returned data sets in *cRecDataSets* is indicated in *cRecSets*. The parameter *iDataType* determines whether the receive data in *cRecDataSets* are to be interpreted as a hex array or as a string. *cRecSets* and *cTrData* always consist of 1 hex character. The *cRecDataSets* buffer should be dimensioned as follows: • iDataType=0: 256 characters (incl. 1 NUL character) • iDataType=1: 512 characters (incl. 1 NUL character) The data contained in *cRecDataSets* are inserted in the order described in the system manual for the OBID *i-scan*® family. *iReaderHnd* is the handle for the Reader object. *cBusAdr* is the bus address set in the multijob-Reader.
Note	The function does not check the data in *cRecDataSets* based on the data structure indicated in *cTrData*.
Cross-reference	For more information on *iDataType* see Section 5.2. Parameter transfer **FEISC_0x33_InitBuffer, FEISC_0x31_ReadDataBufferInfo, FEISC_0x32_ClearDataBuffer**
Return value	If there was no error, the return value contains the status byte of the reply protocol. A list of error codes can be found in the Appendix.

5.6.32. FEISC_0x22_ReadBuffer

Function	Function for data transfer with a transponder
Syntax	**int FEISC_0x22_ReadBuffer(int iReaderHnd, UCHAR cBusAdr, int iSets, UCHAR* cTrData, UCHAR* cRecSets, int* iRecDataSets, int iDataType)**
Description	The function reads the number of data sets *iSets* from the internal data table and stores the data in *cRecDataSets*. *cTrData* defines the structure of a data set in *cRecDataSets*. The number of returned data sets in *cRecDataSets* is indicated in *iRecSets*. The parameter *iDataType* determines whether the receive data in *cRecDataSets* are to be interpreted as a hex array or as a string. *cRecSets* and *cTrData* always consist of 1 hex character. The *cRecDataSets* buffer should be dimensioned for containing all Transponder data. If iDataFormat=1, then the buffer cRecDataSets must be redoubled. The data contained in *cRecDataSets* are inserted in the order described in the system manual for the OBID *i-scan*® family. *iReaderHnd* is the handle for the Reader object. *cBusAdr* is the bus address set in the multijob-Reader.
Note	The function does not check the data in *cRecDataSets* based on the data structure indicated in *cTrData*.
Cross-reference	For more information on *iDataType* see Section 5.2. Parameter transfer **FEISC_0x33_InitBuffer, FEISC_0x31_ReadDataBufferInfo, FEISC_0x32_ClearDataBuffer**
Return value	If there was no error, the return value contains the status byte of the reply protocol. A list of error codes can be found in the Appendix.

5.6.33. FEISC_0x31_ReadDataBufferInfo

Function	Function gets table parameters for the internal data buffer.
Syntax	**int FEISC_0x31_ReadDataBufferInfo(int iReaderHnd, UCHAR cBusAdr, UCHAR* cTabSize, UCHAR* cTabStart, UCHAR* cTabLen, int iDataType)**
Description	The function reads the table parameters from the internal buffer table and stores them in *cTabSize* , *cTabStart* and *cTabLen*. The parameter *iDataType* determines whether the table parameters are to be interpreted as a hex array or as a string. The *cTabSize* , *cTabStart* and *cTabLen* buffers must be dimensioned as follows: • iDataType=0: 3 Characters (incl. 1 NUL character) • iDataType=1: 5 Characters (incl. 1 NUL character) *iReaderHnd* is the handle for the Reader object. *cBusAdr* is the bus address set in the multijob-Reader.
Cross-reference	For more information on *iDataType* see Section 5.2. Parameter transfer **FEISC_0x21_ReadBuffer, FEISC_0x22_ReadBuffer, FEISC_0x33_InitBuffer, FEISC_0x32_ClearDataBuffer,**
Return value	If there was no error, the return value contains the status byte of the reply protocol. A list of error codes can be found in the Appendix.

5.6.34. FEISC_0x32_ClearDataBuffer

Function	Function clears entries read from the internal data buffer.
Syntax	**int FEISC_0x32_ClearDataBuffer(int iReaderHnd, UCHAR cBusAdr)**
Description	The function clears the entries read out from the Reader-internal data buffer by **FEISC_0x21_ReadBuffer**. *iReaderHnd* is the handle for the Reader object. *cBusAdr* is the bus address set in the multijob-Reader.
Cross-reference	**FEISC_0x21_ReadBuffer, FEISC_0x22_ReadBuffer, FEISC_0x33_InitBuffer, FEISC_0x31_ReadDataBufferInfo**
Return value	If there was no error, the return value contains the status byte of the reply protocol. A list of error codes can be found in the Appendix.

5.6.35. FEISC_0x33_InitBuffer

Function	Function for initializing the Reader-internal data table.
Syntax	**int FEISC_0x33_InitBuffer(int iReaderHnd, UCHAR cBusAdr)**
Description	The function initializes the internal data table for the Buffered Read Mode. *iReaderHnd* is the handle for the Reader object. *cBusAdr* is the bus address set in the Reader.
Cross-reference	**FEISC_0x21_ReadBuffer**, **FEISC_0x21_ReadBuffer**, **FEISC_0x31_ReadDataBufferInfo**
Return value	If there was no error, the return value contains the status byte of the reply protocol. A list of error codes can be found in the Appendix.

5.6.36. FEISC_0x34_ForceNotifyTrigger

Function	Function to trigger a notification
Syntax	**int FEISC_0x34_ForceNotifyTrigger(int iReaderHnd, UCHAR cBusAdr, UCHAR ucMode)**
Description	This function triggers at once a notification, which transfers data records from the internal Buffered Read Mode table to the Host. The function returns immediately after the execution and in front of the notification. This function is only usefull, if a background task is prepated with **FEISC_StartAsyncTask** to receive notifications. The parameter *ucMode* is actually unused and should be contain 0x00. *iReaderHnd* is the handle for the Reader object. *cBusAdr* is the bus address set in the Reader.
Cross-reference	5.3. Asynchronous tasks for relieving the load on applications 5.6.12. FEISC_StartAsyncTask
Return value	If there was no error, the return value contains the status byte of the reply protocol. A list of error codes can be found in the Appendix.

5.6.37. FEISC_0x52_GetBaud

Function	Test function for getting baud rate and parity.
Syntax	**int FEISC_0x52_GetBaud(int iReaderHnd, UCHAR cBusAdr)**
Description	If the reply telegram can be received, the configured baud rate and parity are the same as for the Reader. *iReaderHnd* is the handle for the Reader object. *cBusAdr* is the bus address set in the Reader.
Return value	If there was no error, the return value contains the status byte of the reply protocol. A list of error codes can be found in the Appendix.

5.6.38. FEISC_0x55_StartFlashLoader

Function	The function starts the flash loader.
Syntax	**int FEISC_0x55_StartFlashLoader(int iReaderHnd)**
Description	The function starts the Reader flash loader. The Reader must have bus address 0. *iReaderHnd* is the handle for the Reader object.
Return value	If there was no error, the return value contains the status byte of the reply protocol. A list of error codes can be found in the Appendix.

5.6.39. FEISC_0x55_StartFlashLoaderEx

Function	The function starts the flash loader.
Syntax	**int FEISC_0x55_StartFlashLoaderEx(int iReaderHnd, UCHAR cBusAdr)**
Description	The function starts the Reader flash loader. This advanced function supports any busaddress. *iReaderHnd* is the handle for the Reader object. *cBusAdr* is the bus address set in the Reader.
Return value	If there was no error, the return value contains the status byte of the reply protocol. A list of error codes can be found in the Appendix.

5.6.40. FEISC_0x63_CPUReset

Function	Function initiates a reset in the Reader's CPU
Syntax	**int FEISC_0x63_CPUReset(int iReaderHnd, UCHAR cBusAdr)**
Description	Function initiates a reset in the Reader's CPU *iReaderHnd* is the handle for the Reader object. *cBusAdr* is the bus address set in the Reader.
Return value	If there was no error, the return value contains the status byte of the reply protocol. A list of error codes can be found in the Appendix.

5.6.41. FEISC_0x64_SystemReset

Function	Function initiates a reset in a part of the Reader.
Syntax	**int FEISC_0x64_SystemReset(int iReaderHnd, UCHAR cBusAdr, UCHAR cMode)**
Note	Function initiates a reset in a part of the Reader *cMode* defines the Controller to be reset. *iReaderHnd* is the handle for the Reader object. *cBusAdr* is the bus address set in the Reader.
Return value	If there was no error, the return value contains the status byte of the reply protocol. A list of error codes can be found in the Appendix.

5.6.42. FEISC_0x65_SoftVersion

Function	Function reads out the Reader version number.
Syntax	**int FEISC_0x65_SoftVersion(int iReaderHnd, UCHAR cBusAdr, UCHAR* cVersion, int iDataType)**
Description	The Reader version number is gotten and stored in *cVersion*. The parameter *iDataType* specifies whether the version number in *cVersion* is to be interpreted as a hex array or as a string. The buffer for the version must be able to hold at least 8 bytes (iDataType=0) or 15 bytes (iDataType=1). One byte is intended for the NUL character. *iReaderHnd* is the handle for the Reader object. *cBusAdr* is the bus address set in the Reader.
Cross-reference	For more information on *iDataType* see Section 5.2. Parameter transfer
Return value	If there was no error, the return value contains the status byte of the reply protocol. A list of error codes can be found in the Appendix.

5.6.43. FEISC_0x66_ReaderInfo

Function	Function reads out informations of a part of the Reader.
Syntax	**int FEISC_0x66_ReaderInfo(int iReaderHnd, UCHAR cBusAdr, UCHAR cMode, UCHAR* cInfo, int iDataType)**
Description	The information of a part of the Reader is gotten and stored in *cInfo*. *cMode* defines the part of the Reader. The parameter *iDataType* specifies whether the information in *cInfo* is to be interpreted as a hex array or as a string. The buffer for *cInfo* must be able to hold all bytes. One byte is intended for the NUL character. For detailed informations, please refer to the system manual of the OBID *i-scan®* family *iReaderHnd* is the handle for the Reader object. *cBusAdr* is the bus address set in the Reader.
Cross-reference	For more information on *iDataType* see Section 5.2. Parameter transfer
Return value	If there was no error, the return value contains the status byte of the reply protocol. A list of error codes can be found in the Appendix.

5.6.44. FEISC_0x69_RFReset

Function	Function initiates a reset for the antenna field.
Syntax	**int FEISC_0x69_RFReset(int ReaderHnd, UCHAR cBusAdr)**
Description	Function initiates a reset for the Reader's antenna field. All transponders previously turned off by **FEISC_0x1A_Halt** are reactivated. *iReaderHnd* is the handle for the Reader object. *cBusAdr* is the bus address set in the Reader.
Return value	If there was no error, the return value contains the status byte of the reply protocol. A list of error codes can be found in the Appendix.

5.6.45. FEISC_0x6A_RFOnOff

Function	Function for turning the antenna field on/off.
Syntax	**int FEISC_0x6A_RFOnOff(int iReaderHnd, UCHAR cBusAdr, UCHAR cRF)**
Description	A 0 in *cRF* turns the antenna field off. A 1 in *cRF* turns the antenna field on. *iReaderHnd* is the handle for the Reader object. *cBusAdr* is the bus address set in the Reader.
Return value	If there was no error, the return value contains the status byte of the reply protocol. A list of error codes can be found in the Appendix.

5.6.46. FEISC_0x6B_CentralizedRFSync

Function	Function to synchronize antennas.
Syntax	**int FEISC_0x6B_CentralizedRFSync (int iReaderHnd, UCHAR cBusAdr, UCHAR cMode, UCHAR cTxChannel, int iTxPeriod, UCHAR cRes1, UCHAR cRes2)**
Description	The parameters are described in the system manual of the reader. *iReaderHnd* is the handle for the Reader object. *cBusAdr* is the bus address set in the Reader.
Return value	If there was no error, the return value contains the status byte of the reply protocol. A list of error codes can be found in the Appendix.

5.6.47. FEISC_0x6C_SetNoiseLevel

Function	Function for setting the noise level.
Syntax	**int FEISC_0x6C_SetNoiseLevel(int iReaderHnd, UCHAR cBusAdr, UCHAR* cLevel, int iDataType)**
Description	*cLevel* contains the 3 level values which are sent as a hex array with a total of 6 bytes (iDataType=0) or as a string with a total of 12 bytes (iDataType=1). *iReaderHnd* is the handle for the Reader object. *cBusAdr* is the bus address set in the Reader.
Cross-reference	For more information on *iDataType* see Section 5.2. Parameter transfer
Return value	If there was no error, the return value contains the status byte of the reply protocol. A list of error codes can be found in the Appendix.

5.6.48. FEISC_0x6D_GetNoiseLevel

Function	Function for getting the noise level.
Syntax	**int FEISC_0x6D_GetNoiseLevel(int iReaderHnd, UCHAR cBusAdr, UCHAR* cLevel, int iDataType)**
Description	The 3 level values are stored in *cLevel*. The buffer for *cLevel* must be dimensioned as follows: 1. iDataType=0: 7 bytes (incl. NUL character) *2.* iDataType=1: 13 bytes (incl. NUL character) *iReaderHnd* is the handle for the Reader object. *cBusAdr* is the bus address set in the Reader.
Cross-reference	For more information on *iDataType* see Section 5.2. Parameter transfer
Return value	If there was no error, the return value contains the status byte of the reply protocol. A list of error codes can be found in the Appendix.

5.6.49. FEISC_0x6E_RdDiag

Function	Function for Reader diagnostics.
Syntax	**int FEISC_0x6E_RdDiag(int iReaderHnd, UCHAR cBusAdr, UCHAR cMode, UCHAR* cData)**
Description	The function returns diagnostics values for the handle stored in *cMode*. The buffer for the receive data *cData* must be sufficiently dimensioned. *iReaderHnd* is the handle for the Reader object. *cBusAdr* is the bus address set in the Reader.
Cross-reference	For more information on *iDataType* see Section 5.2. Parameter transfer
Return value	If there was no error, the return value contains the status byte of the reply protocol. A list of error codes can be found in the Appendix.

5.6.50. FEISC_0x6F_AntennaTuning

Function	Function enables a special mode in the reader.
Syntax	**int FEISC_0x6F_AntennaTuning(int ReaderHnd, UCHAR cBusAdr)**
Description	This function enables a special tuning mode in the reader. The reader must be reset for disabling this mode. *iReaderHnd* is the handle for the Reader object. *cBusAdr* is the bus address set in the Reader.
Return value	If there was no error, the return value contains the status byte of the reply protocol. A list of error codes can be found in the Appendix.

5.6.51. FEISC_0x71_SetOutput

Function	Function activates the Reader's outputs.
Syntax	**int FEISC_0x71_SetOutput(int iReaderHnd, UCHAR cBusAdr, int iOS, int iOSF, int iOSTime, int iOutTime)**
Description	The function activates the Reader's outputs. All times are multiplied internally in the Reader by 100 and are to be interpreted in units of ms. The value ranges indicated in the system manual for the ISC Reader family are applicable. *iReaderHnd* is the handle for the Reader object. *cBusAdr* is the bus address set in the Reader.
Return value	If there was no error, the return value contains the status byte of the reply protocol. A list of error codes can be found in the Appendix.

5.6.52. FEISC_0x72_SetOutput

Function	Function activates the Reader's outputs.
Syntax	**int FEISC_0x72_SetOutput(int iReaderHnd, UCHAR cBusAdr, UCHAR cMode, UCHAR cOutN, UCHAR* pRecords)**
Description	The function activates the Reader's outputs. The number of outputs to be activated is set with *cOutN*. The activation parameters of each output must be collected in a buffer. *pRecords* is the pointer to this buffer. The parameter *cMode* is the mode byte of the protocol. *iReaderHnd* is the handle for the Reader object. *cBusAdr* is the bus address set in the Reader.
Return value	If there was no error, the return value contains the status byte of the reply protocol. A list of error codes can be found in the Appendix.

5.6.53. FEISC_0x74_ReadInput

Function	Function reads the status of the digital inputs.
Syntax	**int FEISC_0x74_ReadInput(int iReaderHnd, UCHAR cBusAdr, UCHAR* cInput)**
Description	The function reads the digital inputs and stores the status in *cInput*. The length of *cInput* is 1. *iReaderHnd* is the handle for the Reader object. *cBusAdr* is the bus address set in the Reader.
Return value	If there was no error, the return value contains the status byte of the reply protocol. A list of error codes can be found in the Appendix.

5.6.54. FEISC_0x75_AdjAntenna

Function	Function for reading the antenna level.
Syntax	**int FEISC_0x75_AdjAntenna(int iReaderHnd, UCHAR cBusAdr, UCHAR* cLevel, int iDataType)**
Description	The read level value is stored in *cLevel*. The buffer for *cLevel* must be dimensioned as follows: 3. iDataType=0: 3 bytes (incl. NUL character) *4.* iDataType=1: 5 bytes (incl. NUL character) *iReaderHnd* is the handle for the Reader object. *cBusAdr* is the bus address set in the Reader.
Cross-reference	For more information on *iDataType* see Section 5.2. Parameter transfer
Return value	If there was no error, the return value contains the status byte of the reply protocol. A list of error codes can be found in the Appendix.

5.6.55. FEISC_0x80_ReadConfBlock

Function	Function reads a configuration block from the Reader.
Syntax	**int FEISC_0x80_ReadConfBlock(int iReaderHnd, UCHAR cBusAdr, UCHAR cConfAdr, UCHAR* cConfBlock, int iDataType)**
Description	This function allows you to read a configuration block from address *cConfAdr* of the Reader. The data read out in *cConfBlock* are to be interpreted as a hex array (*iDataType*=0) or as a string (*iDataType*=1). The buffer for the configuration data *cConfBlock* must be dimensioned as follows: 1. *iDataType*=0: 15 bytes (incl. 1 NUL character) 2. *iDataType*=1: 29 bytes (incl. 1 NUL character) *iReaderHnd* is the handle for the Reader object. *cBusAdr* is the bus address set in the Reader.
Cross-reference	For more information on *iDataType* see Section 5.2. Parameter transfer
Return value	If there was no error, the return value contains the status byte of the reply protocol. A list of error codes can be found in the Appendix.

5.6.56. FEISC_0x81_WriteConfBlock

Function	Function writes a configuration block to the Reader.
Syntax	**int FEISC_0x81_WriteConfBlock(int iReaderHnd, UCHAR cBusAdr, UCHAR cConfAdr, UCHAR* cConfBlock, int iDataType)**
Description	This function lets you write a configuration block to address *cConfAdr* of the Reader. The configuration data must be stored in *cConfBlock* as a hex array (*iDataType*=0) or string (*iDataType*=1). The buffer with the configuration data must contain 14 bytes (*iDataType*=0) or 28 bytes (*iDataType*=1). *iReaderHnd* is the handle for the Reader object. *cBusAdr* is the bus address set in the Reader.
Cross-reference	For more information on *iDataType* see Section 5.2. Parameter transfer
Return value	If there was no error, the return value contains the status byte of the reply protocol. A list of error codes can be found in the Appendix.

5.6.57. FEISC_0x82_SaveConfBlock

Function	Function saves a configuration block in the Reader.
Syntax	**int FEISC_0x82_SaveConfBlock(int iReaderHnd, UCHAR cBusAdr, UCHAR cConfAdr)**
Description	This function allows you to write a configuration block for address *cConfAdr* from RAM memory to the EEPROM (non-volatile memory) and save it for a longer period. *iReaderHnd* is the handle for the Reader object. *cBusAdr* is the bus address set in the Reader.
Return value	If there was no error, the return value contains the status byte of the reply protocol. A list of error codes can be found in the Appendix.

5.6.58. FEISC_0x83_ResetConfBlock

Function	Function loads the factory setting into a configuration block in the Reader.
Syntax	**int FEISC_0x83_ResetConfBlock(int iReaderHnd, UCHAR cBusAdr, UCHAR cConfAdr)**
Description	This function allows you to load the parameters for the factory default settings into a configuration block for address *cConfAdr*. *iReaderHnd* is the handle for the Reader object. *cBusAdr* is the bus address set in the Reader.
Return value	If there was no error, the return value contains the status byte of the reply protocol. A list of error codes can be found in the Appendix.

5.6.59. FEISC_0x85_SetSysTimer

Function	Sets the system time in the Reader.
Syntax	**int FEISC_0x85_SetSysTimer(int iReaderHnd, UCHAR cBusAdr, UCHAR* cTime, int iDataType)**
Description	The function initializes the system time in the Reader. The buffer *cTime* must contain 4 bytes (*iDataType*=0) or be a string with 8 characters (*iDataType*=1). *iReaderHnd* is the handle for the Reader object. *cBusAdr* is the bus address set in the Reader.
Reference	For more information on *iDataType* see Section 5.2. Parameter transfer
Return value	If there was no error, the return value contains the status byte of the reply protocol. A list of error codes can be found in the Appendix.

5.6.60. FEISC_0x86_GetSysTimer

Function	Reads the system time from the Reader.
Syntax	**int FEISC_0x86_GetSysTimer(int iReaderHnd, UCHAR cBusAdr, UCHAR* cTime, int iDataType)**
Description	This function gets the system time from the Reader. The buffer for *cTime* must be dimensioned as follows: 5. *iDataType*=0: 5 Characters (incl. 1 NUL character)) *6.* *iDataType*=1: 9 Characters (incl. 1 NUL character)) *iReaderHnd* is the handle for the Reader object. *cBusAdr* is the bus address set in the Reader.
Reference	For more information on *iDataType* see Section 5.2. Parameter transfer
Return value	If there was no error, the return value contains the status byte of the reply protocol. A list of error codes can be found in the Appendix.

5.6.61. FEISC_0x87_SetSystemDate

Function	Sets the system date and time in the Reader.
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)**
Description	The function initializes the system date and time in the Reader. *cCentury* : century (e.g. 20) *cYear* : year (e.g. 4) *cMonth* : month (e.g. 10) *cDay* : day (e.g. 5) *cTimezone* : timezone (actually unused) *cHour* : hour (e.g. 15) *cMinute* : minute (e.g. 13) *iMilliSecond* : milliseconds, containing also the seconds (e.g. 1234 for 1s and 234ms) *iReaderHnd* is the handle for the Reader object. *cBusAdr* is the bus address set in the Reader.
Return value	If there was no error, the return value contains the status byte of the reply protocol. A list of error codes can be found in the Appendix.

5.6.62. FEISC_0x88_GetSystemDate

Function	Reads the system date and time from the Reader.
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)**
Description	This function gets the system date and time from the Reader. The function parameters are described in 5.6.61. FEISC_0x87_SetSystemDate. *iReaderHnd* is the handle for the Reader object. *cBusAdr* is the bus address set in the Reader.
Return value	If there was no error, the return value contains the status byte of the reply protocol. A list of error codes can be found in the Appendix.

5.6.63. FEISC_0xA0_RdLogin

Function	Function performs a login in the Reader.
Syntax	**int FEISC_0xA0_RdLogin(int iReaderHnd, UCHAR cBusAdr, UCHAR* cRd_PW, int iDataType)**
Description	The function uses the password *cRd_PW* to login to the Reader. The parameter *iDataType* specifies whether the password in *cRd_PW* is to be interpreted as a hex array (*iDataType*=0) or as a string (*iDataType*=1). *iReaderHnd* is the handle for the Reader object. *cBusAdr* is the bus address set in the Reader.
Cross-reference	For more information on *iDataType* see Section 5.2. Parameter transfer
Return value	If there was no error, the return value contains the status byte of the reply protocol. A list of error codes can be found in the Appendix.

5.6.64. FEISC_0xA2_WriteMifareKeys

Funktion	Function writes authentication key into the reader.
Syntax	**int FEISC_0xA2_WriteMifareKeys(int iReaderHnd, UCHAR cBusAdr, UCHAR cType, UCHAR cAdr, UCHAR* cKey, int iDataType)**
Hinweis	Be careful with this function. You cannot read back the authentication key from the reader.
Beschreibung	This function writes the authentication key for a Mifare-Transponder into the EEPROM of the reader. *cType* defines the key type, *cAdr* specifies the EEPROM address of the key in the reader. The parameter *iDataType* specifies whether the authentication key in *cKey* is to be interpreted as a hex array (*iDataType*=0) or as a string (*iDataType*=1). *iReaderHnd* is the handle for the Reader object. *cBusAdr* is the bus address set in the Reader.
Querverweis	For more information on *iDataType* see Section 5.2. Parameter transfer
Rückgabewert	If there was no error, the return value contains the status byte of the reply protocol. A list of error codes can be found in the Appendix.

5.6.65. FEISC_0xB0_ISOCmd

Function	Function initiates data transfer with ISO15693 or ISO14443 transponders.
Syntax	**int FEISC_0xB0_ISOCmd(int iReaderHnd, UCHAR cBusAdr, UCHAR* cReqData, int iReqLen, UCHAR* cRspData, int* iRspLen, int iDataType)**
Description	The function initiates a data transfer for multiple ISO15693 or ISO14443 transponders located in the active zone of the ISC Reader. The data necessary for the data transfer are to be stored in *cReqData*. The number of characters contained in *cReqData* must be indicated in *iReqLen*. The data read from the ISO15693 oder ISO14443 transponder are contained in *cRspData*. *iRspLen* indicates the number of characters in *cRspData*. The parameter *iDataType* specifies whether *cReqData* and *cRspData* are to be interpreted as a hex array or as a string. Note the following: The buffer for the receive data *cRspData* must be dimensioned such that all the receive data can be stored. This means in the case of *iDataType*=1 that the size of the buffer *cRspData* is twice as large as in the case of *iDataType*=0. The length of the send data (number of characters in *cReqData*) must be indicated in *iReqLen*. If *iDataType*=1, then *iReqLen* is twice as large as in the case of *iDataType*=0. The length indication for the receive buffer (*cRspData*) is to be handled analogously. *iReaderHnd* is the handle for the Reader object. *cBusAdr* is the bus address set in the Reader.
Cross-reference	For more information on *iDataType* see Section 5.2. Parameter transfer
Return value	If there was no error, the return value contains the status byte of the reply protocol.

5.6.66. FEISC_0xB1_ ISOCustAndPropCmd

Function	Function initiates data transfer with an ISO15693 transponder.
Syntax	**int FEISC_0xB1_ISOCustAndPropCmd(int iReaderHnd, UCHAR cBusAdr, UCHAR cMfr, UCHAR* cReqData, int iReqLen, UCHAR* cRspData, int* iRspLen, int iDataType)**
Description	The function initiates a data transfer for multiple ISO15693 transponders located in the active zone of the ISC Reader. The parameter *cMfr* contains the manufacturer code and specifies the structure of send data *cReqData* and receive data *cRspData*. The data necessary for the data transfer are to be stored in *cReqData*. The number of characters contained in *cReqData* must be indicated in *iReqLen*. The data read from the ISO15693 transponder are contained in *cRspData*. *iRspLen* indicates the number of characters in *cRspData*. The parameter *iDataType* specifies whether *cReqData* and *cRspData* are to be interpreted as a hex array or as a string. Note the following: The buffer for the receive data *cRspData* must be dimensioned such that all the receive data can be stored. This means in the case of *iDataType*=1 that the size of the buffer *cRspData* is twice as large as in the case of *iDataType*=0. The length of the send data (number of characters in *cReqData*) must be indicated in *iReqLen*. If *iDataType*=1, then *iReqLen* is twice as large as in the case of *iDataType*=0. The length indication for the receive buffer (*cRspData*) is to be handled analogously. *iReaderHnd* is the handle for the Reader object. *cBusAdr* is the bus address set in the Reader.
Cross-reference	For more information on *iDataType* see Section 5.2. Parameter transfer
Return value	If there was no error, the return value contains the status byte of the reply protocol.

5.6.67. FEISC_0xB2_ISOCmd

Function	Function initiates data transfer with an ISO14443 transponder.
Syntax	**int FEISC_0xB2_ISOCmd(int iReaderHnd, UCHAR cBusAdr, UCHAR* cReqData, int iReqLen, UCHAR* cRspData, int* iRspLen, int iDataType)**
Description	The function initiates a data transfer for multiple ISO14443 transponders located in the active zone of the ISC Reader. The data necessary for the data transfer are to be stored in *cReqData*. The number of characters contained in *cReqData* must be indicated in *iReqLen*. The data read from the ISO14443 transponder are contained in *cRspData*. *iRspLen* indicates the number of characters in *cRspData*. The parameter *iDataType* specifies whether *cReqData* and *cRspData* are to be interpreted as a hex array or as a string. Note the following: The buffer for the receive data *cRspData* must be dimensioned such that all the receive data can be stored. This means in the case of *iDataType*=1 that the size of the buffer *cRspData* is twice as large as in the case of *iDataType*=0. The length of the send data (number of characters in *cReqData*) must be indicated in *iReqLen*. If *iDataType*=1, then *iReqLen* is twice as large as in the case of *iDataType*=0. The length indication for the receive buffer (*cRspData*) is to be handled analogously. *iReaderHnd* is the handle for the Reader object. *cBusAdr* is the bus address set in the Reader.
Cross-reference	For more information on *iDataType* see Section 5.2. Parameter transfer
Return value	If there was no error, the return value contains the status byte of the reply protocol.

5.6.68. FEISC_0xB3_EPCCmd

Function	Function initiates data transfer with an UHF EPC-Transponder.
Syntax	**int FEISC_0xB3_EPCCmd(int iReaderHnd, UCHAR cBusAdr, UCHAR* cReqData, int iReqLen, UCHAR* cRspData, int* iRspLen, int iDataType)**
Description	The function initiates a data transfer with an UHF EPC-Transponders located in the active zone of the Reader. The data necessary for the data transfer are to be stored in *cReqData*. The number of characters contained in *cReqData* must be indicated in *iReqLen*. The data read from the transponder are contained in *cRspData*. *iRspLen* indicates the number of characters in *cRspData*. The parameter *iDataType* specifies whether *cReqData* and *cRspData* are to be interpreted as a hex array or as a string. Note the following: The buffer for the receive data *cRspData* must be dimensioned such that all the receive data can be stored. This means in the case of *iDataType*=1 that the size of the buffer *cRspData* is twice as large as in the case of *iDataType*=0. The length of the send data (number of characters in *cReqData*) must be indicated in *iReqLen*. If *iDataType*=1, then *iReqLen* is twice as large as in the case of *iDataType*=0. The length indication for the receive buffer (*cRspData*) is to be handled analogously. *iReaderHnd* is the handle for the Reader object. *cBusAdr* is the bus address set in the Reader.
Cross-reference	For more information on *iDataType* see Section 5.2. Parameter transfer
Return value	If there was no error, the return value contains the status byte of the reply protocol.

5.6.69. FEISC_0xB4_EPC_UHF_Cmd

Function	Function initiates data transfer with an 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 iDataType)**
Description	The function initiates a data transfer with an UHF EPC-Transponders located in the active zone of the Reader. The parameter *cMfr* contains the manufacturer code and specifies the structure of send data *cReqData* and receive data *cRspData*. The data necessary for the data transfer are to be stored in *cReqData*. The number of characters contained in *cReqData* must be indicated in *iReqLen*. The data read from the transponder are contained in *cRspData*. *iRspLen* indicates the number of characters in *cRspData*. The parameter *iDataType* specifies whether *cReqData* and *cRspData* are to be interpreted as a hex array or as a string. Note the following: The buffer for the receive data *cRspData* must be dimensioned such that all the receive data can be stored. This means in the case of *iDataType*=1 that the size of the buffer *cRspData* is twice as large as in the case of *iDataType*=0. The length of the send data (number of characters in *cReqData*) must be indicated in *iReqLen*. If *iDataType*=1, then *iReqLen* is twice as large as in the case of *iDataType*=0. The length indication for the receive buffer (*cRspData*) is to be handled analogously. *iReaderHnd* is the handle for the Reader object. *cBusAdr* is the bus address set in the Reader.
Cross-reference	For more information on *iDataType* see Section 5.2. Parameter transfer
Return value	If there was no error, the return value contains the status byte of the reply protocol.

5.6.70. FEISC_0xBB_C1G2_ TranspCmd

Function	Function initiates data transfer with a 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)**
Description	The function initiates a data transfer for oneI Class 1 generation 2 UHF transponder located in the active zone of the Reader. The parameter *ucMode* contains the mode for the reader. The parameters *ucTxPara*, *ucRxPara* and *uiTs* controlles the timing of the RF communication. The parameter *iRspLength* contains the requested length (number of bits) of receive data *cRspData*. The data necessary for the data transfer are to be stored in *cReqData*. The number of characters contained in *cReqData* must be indicated in *iReqLen*. The data read from the UHF transponder are contained in *cRspData*. *iRspLen* indicates the number of characters in *cRspData*. Note the following: The buffer for the receive data *cRspData* must be dimensioned such that all the receive data can be stored. *iReaderHnd* is the handle for the Reader object. *cBusAdr* is the bus address set in the Reader.
Cross-reference	
Return value	If there was no error, the return value contains the status byte of the reply protocol.

5.6.71. FEISC_0xBC_CmdQueue

Function	A queue command task is started asynchronous to the application
Syntax	**int FEISC_0xBC_CmdQueue(int iReaderHnd, int iMode, int iCmdCount, UCHAR* ucCmdQueue, int iCmdQueueLen, FEISC_TASK_INIT* pInit)**
Description	This function starts the queue command as an asynchronous task. An asynchronous task is an internal thread which sends the queue command to the reader and waits for the reply for a time up to the timeout. Signaling of the reply data or the cancel condition to the application is done by invoking a callback function. The parameter *iMode* contains mode values. *iCmdCount* contains the number of commands in the queue. The queue data necessary for the data transfer are to be stored in *ucCmdQueue*. The number of characters contained in *ucCmdQueue* must be indicated in *iCmdQueueLen*. All the data relevant to the callback function are contained in the structure FEISC_TASK_INIT. This structure is described in greater detail in section 5.3. Asynchronous tasks for relieving the load on applications. The following setting is recommended: `FEISC_TASK_INIT Init;` `Init.cbFct1 = this->cbsTaskRsp1;  // callback function` `Init.ucBusAdr = 255;  // every reader will respond` `Init.uiFlag = FEISC_TASKCB_1;` `Init.uiTimeout = m_uiTimeout;  // individual timeout` `Init.pAny = this;  // optional: This-Pointer` *iReaderHnd* is the handle for the reader object.
Cross-references	Additional information about asynchronous tasks can be found in the section 5.3. Asynchronous tasks for relieving the load on applications. 5.6.13. FEISC_CancelAsyncTask
Note	More detailed information about the protocol [0xBC] Command Queue can be found in the manual for the OBID® *classic-pro* Reader family.
Return value	In case of no error a 0 is returned. A value less than 0 indicate an error. The list of error codes can be found in the Appendix.

5.6.72. FEISC_0xBD_ ISOTranspCmd

Function	Function initiates data transfer with an ISO14443A transponder.
Syntax	**int FEISC_0xBD_ISOTranspCmd(int iReaderHnd, UCHAR cBusAdr, int iMode, int iRspLength, UCHAR* cReqData, int iReqLen, UCHAR* cRspData, int* iRspLen, int iDataType)**
Description	The function initiates a data transfer for multiple ISO14443A transponders located in the active zone of the ISC Reader. The parameter *iMode* contains the mode for the reader. The parameter *iRspLength* contains the requested length (number of bits) of receive data *cRspData*. The data necessary for the data transfer are to be stored in *cReqData*. The number of characters contained in *cReqData* must be indicated in *iReqLen*. The data read from the ISO14443A transponder are contained in *cRspData*. *iRspLen* indicates the number of characters in *cRspData*. The parameter *iDataType* specifies whether *cReqData* and *cRspData* are to be interpreted as a hex array or as a string. Note the following: The buffer for the receive data *cRspData* must be dimensioned such that all the receive data can be stored. This means in the case of *iDataType*=1 that the size of the buffer *cRspData* is twice as large as in the case of *iDataType*=0. The length of the send data (number of characters in *cReqData*) must be indicated in *iReqLen*. If *iDataType*=1, then *iReqLen* is twice as large as in the case of *iDataType*=0. The length indication for the receive buffer (*cRspData*) is to be handled analogously. *iReaderHnd* is the handle for the Reader object. *cBusAdr* is the bus address set in the Reader.
Cross-reference	For more information on *iDataType* see Section 5.2. Parameter transfer
Return value	If there was no error, the return value contains the status byte of the reply protocol.

5.6.73. FEISC_0xBE_ ISOTranspCmd

Function	Function initiates data transfer with an ISO14443B transponder.
Syntax	**int FEISC_0xBE_ISOTranspCmd(int iReaderHnd, UCHAR cBusAdr, int iMode, int iRspLength, UCHAR* cReqData, int iReqLen, UCHAR* cRspData, int* iRspLen, int iDataType)**
Description	The function initiates a data transfer for multiple ISO14443B transponders located in the active zone of the ISC Reader. The parameter *iMode* contains the mode for the reader. The parameter *iRspLength* contains the requested length (number of bits) of receive data *cRspData*. The data necessary for the data transfer are to be stored in *cReqData*. The number of characters contained in *cReqData* must be indicated in *iReqLen*. The data read from the ISO14443B transponder are contained in *cRspData*. *iRspLen* indicates the number of characters in *cRspData*. The parameter *iDataType* specifies whether *cReqData* and *cRspData* are to be interpreted as a hex array or as a string. Note the following: The buffer for the receive data *cRspData* must be dimensioned such that all the receive data can be stored. This means in the case of *iDataType*=1 that the size of the buffer *cRspData* is twice as large as in the case of *iDataType*=0. The length of the send data (number of characters in *cReqData*) must be indicated in *iReqLen*. If *iDataType*=1, then *iReqLen* is twice as large as in the case of *iDataType*=0. The length indication for the receive buffer (*cRspData*) is to be handled analogously. *iReaderHnd* is the handle for the Reader object. *cBusAdr* is the bus address set in the Reader.
Cross-reference	For more information on *iDataType* see Section 5.2. Parameter transfer
Return value	If there was no error, the return value contains the status byte of the reply protocol.

5.6.74. FEISC_0xBF_ ISOTranspCmd

Function	Function initiates data transfer with an ISO15693 transponder.
Syntax	**int FEISC_0xBF_ISOTranspCmd(int iReaderHnd, UCHAR cBusAdr, int iMode, int iRspLength, UCHAR* cReqData, int iReqLen, UCHAR* cRspData, int* iRspLen, int iDataType)**
Description	The function initiates a data transfer for multiple ISO15693 transponders located in the active zone of the ISC Reader. The parameter *iMode* contains the mode for the reader. The parameter *iRspLength* contains the requested length (number of bits) of receive data *cRspData*. The data necessary for the data transfer are to be stored in *cReqData*. The number of characters contained in *cReqData* must be indicated in *iReqLen*. The data read from the ISO15693 transponder are contained in *cRspData*. *iRspLen* indicates the number of characters in *cRspData*. The parameter *iDataType* specifies whether *cReqData* and *cRspData* are to be interpreted as a hex array or as a string. Note the following: The buffer for the receive data *cRspData* must be dimensioned such that all the receive data can be stored. This means in the case of *iDataType*=1 that the size of the buffer *cRspData* is twice as large as in the case of *iDataType*=0. The length of the send data (number of characters in *cReqData*) must be indicated in *iReqLen*. If *iDataType*=1, then *iReqLen* is twice as large as in the case of *iDataType*=0. The length indication for the receive buffer (*cRspData*) is to be handled analogously. *iReaderHnd* is the handle for the Reader object. *cBusAdr* is the bus address set in the Reader.
Cross-reference	For more information on *iDataType* see Section 5.2. Parameter transfer
Return value	If there was no error, the return value contains the status byte of the reply protocol.

5.6.75. FEISC_0xC0_SAMCmd

Function	Function initiates a data transfer with a SAM (Secure Access Module).
Syntax	**int FEISC_0xC0_SAMCmd(int iReaderHnd, int iSlot, UCHAR* cReqData, int iReqLen, FEISC_TASK_INIT* pInit)**
Description	This function starts the SAM command as an asynchronous task. An asynchronous task is an internal thread which sends the SAM command to the reader and waits for the reply for a time up to the timeout. Signaling of the reply data or the cancel condition to the application is done by invoking a callback function. The parameter *iSlot* identifies the SAM slot. The queue data necessary for the data transfer are to be stored in *cReqData*. The number of characters contained in *cReqData* must be indicated in *iReqDataLen*. All the data relevant to the callback function are contained in the structure FEISC_TASK_INIT. This structure is described in greater detail in section 5.3. Asynchronous tasks for relieving the load on applications. The following setting is recommended: `FEISC_TASK_INIT Init;` `Init.cbFct1 = this->cbsTaskRsp1;  // callback function` `Init.ucBusAdr = 255;  // every reader will respond` `Init.uiFlag = FEISC_TASKCB_1;` `Init.uiTimeout = m_uiTimeout;  // individual timeout` `Init.pAny = this;  // optional: This-Pointer` *iReaderHnd* is the handle for the Reader object.
Cross-reference	Additional information about asynchronous tasks can be found in the section 5.3. Asynchronous tasks for relieving the load on applications. 5.6.13. FEISC_CancelAsyncTask
Note	More detailed information about the protocol [0xC0] SAM Command can be found in the manual for the OBID® *classic-pro* Reader family.
Return value	If there was no error, the return value contains the status byte of the reply protocol.

5.7. Support for multithreading

The functions in FEISC are essentially thread-safe, meaning function calls from several threads to the library are possible as long as a communications procedure in a thread is never interrupted by another communications procedure from another thread.

There are no protection mechanisms within the library which preclude a preemptive procedure from another thread. This protection must be implemented on the application level.

A problem does occur when a callback function implemented using the **FEISC_AddEventHandler** function is used to transfer a protocol string to the application and represent it in a protocol window. Attempting to display the string in the window from out of the thread can cause the program to crash (e.g. when using MFC in C++). The remedy is to buffer store and send a Windows message with the API function SendMessage(..) to the window. This will serve to decouple the threads. Even better in such cases is to select the **FEISC_AddEventHandler** message methods from right at the outset.

Closing a window while a protocol is being represented can also cause a program crash. The FEISC offers some help here in that the protocol output in the library can be specifically stopped in all Reader objects. This is done by invoking **FEISC_SetReaderPara**(0, „LockProtToApp“, „“). Next continue checking using the function **FEISC_GetReaderPara**(0, „IsProtToAppLocked“, „“) until all the protocol outputs from the library are finished. If the function returns a 0, the protocol output is not yet finished. If a 1 is returned, the window may be closed. Contrary to convention, the return values are selected so that you can check them (in any case using C) for true.

C++ Example with MFC:

The member function OnClose is called when you want to close the window (View) by clicking with the mouse on the close icon. The class FELogChildFrame derived from CMDIChildWnd is the frame window of the Doc/View pair for the protocol output window. Cyclically recalling with a WM_CLOSE message to yourself will cause a time loop which gives the FEISC time to close the protocol outputs. Only when the function **FEISC_GetReaderPara**(0, „IsProtToAppLocked“, „“) no longer returns a 0 may the window be closed using CMDIChildWnd::OnClose().

```
void FELogChildFrame::OnClose()
{
    // Message to library that all further protocol outputs are to be locked
    FEISC_SetReaderPara(0, "LockProtToApp", "");

    // Query to library whether all protocol outputs are already finished
    int iBack = FEISC_GetReaderPara(0, "IsProtToAppLocked", "");

    if(iBack==0)
    {
        // No, therefore with message to this repeated call from OnClose
        this->SendMessage(WM_CLOSE, 0, 0);
        return;
    }

    // If you are here then all protocol outputs are finished
    // and there is no longer a risk of crashing when the Doc/View pair is closed
    CMDIChildWnd::OnClose();
}
```

6. Appendix

6.1. Error codes

Error constants	Value	Description
FEISC_ERR_NEWREADER_FAILURE	-4000	Error in creating a new Reader object
FEISC_ERR_EMPTY_LIST	-4001	Reader handle list is empty (no Reader objects stored)
FEISC_ERR_POINTER_IS_NULL	-4002	Pointer to transfer parameter is NULL
FEISC_ERR_NO_MORE_MEM	-4003	No more system memory
FEISC_ERR_UNKNOWN_COMM_PORT	-4004	Unknown COM port
FEISC_ERR_UNSUPPORTED_FUNCTION	-4005	Unsupported function
FEISC_ERR_NO_USB_SUPPORT	-4006	No USB support (e.g. under NT4)
FEISC_ERR_OLD_FECOM	-4007	Old FECOM.DLL detected
FEISC_ERR_NO_VALUE	-4010	No data value
FEISC_ERR_UNKNOWN_HND	-4020	The transferred Reader handle is unknown
FEISC_ERR_HND_IS_NULL	-4021	The transferred Reader handle is 0
FEISC_ERR_HND_IS_NEGATIVE	-4022	The transferred Reader handle is negative
FEISC_ERR_NO_HND_FOUND	-4023	No Reader handle found in Reader handle list
FEISC_ERR_PORTHND_IS_NEGATIVE	-4024	The transferred port handle is negative
FEISC_ERR_HND_UNVALID	-4025	Invalid port handle; the first byte (MSB) in the port handle is invalid
FEISC_ERR_PROTLEN	-4030	Protocol length error
FEISC_ERR_CHECKSUM	-4031	Checksum error
FEISC_ERR_BUSY_TIMEOUT	-4032	Timeout after continuous busy messages
FEISC_ERR_UNKNOWN_STATUS	-4033	Unknown status byte
FEISC_ERR_NO_RECPROTOCOL	-4034	No USB receive protocol arrived
FEISC_ERR_CMD_BYTE	-4035	Wrong command byte in receive protocol
FEISC_ERR_TRANSCEIVE	-4036	General USB communications error
FEISC_ERR_REC_BUS_ADR	-4037	False bus address in receive protocol
FEISC_ERR_UNKNOWN_PARAMETER	-4050	Transfer parameter is unknown
FEISC_ERR_PARAMETER_OUT_OF_RANGE	-4051	Transfer parameter too large or too small
FEISC_ERR_ODD_PARAMETERSTRING	-4052	The transferred string contains an uneven number of characters
FEISC_ERR_UNKNOWN_ERRORCODE	-4053	Unknown error code
FEISC_ERR_UNSUPPORTED_OPTION	-4054	Unsupported option
FEISC_ERR_UNKNOWN_EPC_TYPE	-4055	Unknown EPC type
FEISC_ERR_NO_PLUGIN	-4060	Installation of Plug-In object in reader object is missing

Error constants	Value	Description
FEISC_ERR_PLUGIN_PRESENT	-4061	Error while installation of a second Plug-In object to a reader object
FEISC_ERR_UNKNOWN_PLUGIN_ID	-4062	Unknown Plug-In ID
FEISC_ERR_PI_BUILD_DATA	-4063	Return value for an error in the Plug-In function build_datastream
FEISC_ERR_PI_BUILD_FRAME	-4064	Return value for an error in the Plug-In function build_protocol
FEISC_ERR_PI_SPLIT_FRAME	-4065	Return value for an error in the Plug-In function split_protocol
FEISC_ERR_PI_SPLIT_DATA	-4066	Return value for an error in the Plug-In function split_datastream
FEISC_ERR_BUFFER_OVERFLOW	-4070	Databuffer is too small
FEISC_ERR_TASK_STILL_RUNNING	-4080	Asynchronous task is still running
FEISC_ERR_TASK_NOT_STARTED	-4081	Start of asynchronous task failed
FEISC_ERR_TASK_TIMEOUT	-4082	Asynchronous task timed out: the reader has sent no reply
FEISC_ERR_TASK_SOCKET_INIT	-4083	The socket for the task couldn't be initialized.
FEISC_ERR_TASK_BUSY	-4084	Asynchronous task executes the callback function and is just busy. The application must repeat the function.
FEISC_ERR_THREAD_CANCEL_ERROR	-4085	Cancellation of internal thread failed.

6.2. List of variables

Variable	Value range	Default	Unit	Description
PortHnd[21]	0 ... 4294967295	0		PortHandle for communication with ID FECOM, ID FETCP or ID FEUSB
LogProt	0, 1	0		If 1, then protocol are output through event flagging
LogFile	0, 1	0		If 1, then writing all protocol strings into Logfile feisc_log.txt
LogFilename	Max. 256 chars	feisc_log.txt		Filename for LogFile
Language	7 - german 9 - english	9	-	language selection for internal strings.
RecBusAdr	0 ... 255	-	-	bus address from last receive protocol. Read-only value.
ConvHexToString	0, 1	0	-	If 1, then all received bytes in scan option are converted to a string. Parameter is only useful, if the readers scan data output is set to *unformatted hex data*.
FrameSupport	„Standard“, „Advanced“	“Standard”	-	Selection of the protocol frame of the send protocol. The frame of the received protocol is detected automatically.
SendStr	-	-	-	Provides last send protocol with preceding date and time of day
RecStr	-	-	-	Provides last receive protocol with preceding date and time of day
ChkRecBusAdr	0, 1	0	-	If 1, then check of received bus address with the bus address of send protocol. If bus addresses are unequal, an error code is responded. Exceptions: bus addresses 254 and 255.
LockProtToApp	none	-		Multithreading support: Locks the protocol output through event flagging in all Reader objects s. 5.7. Support for multithreading
UnlockProtToApp	none	-		Multithreading support: Unlocks protocol output through event flagging s. 5.7. Support for multithreading
IsProtToAppLocked	none	-		Multithreading support: Asks whether all Reader objects are finished with protocol output through event flagging s. 5.7. Support for multithreading

[21] Note the remarks in Section 5.6.2. FEISC_NewReader

6.3. List of constants for the FEISC_EVENT_INIT structure

The constants definitions are contained in the file FEISC.H or FEISC.BAS or FEISC.PAS.

Constants	Value	Use	Description
FEISC_THREAD_ID	1	uiFlag	Event flagging with thread message
FEISC_WND_HWND	2	uiFlag	Event flagging with window message
FEISC_CALLBACK	3	uiFlag	Event flagging with callback function
FEISC_EVENT	4	uiFlag	Event flagging with Windows-API event
FEISC_CALLBACK_2	5	uiFlag	Event flagging with 2. callback function
FEISC_PRT_EVENT	1	uiUse	Flagging for send and receive protocols
FEISC_SNDPRT_EVENT	2	uiUse	Flagging for send protocols
FEISC_RECPRT_EVENT	3	uiUse	Flagging for receive protocols
FEISC_SCANNER_EVENT	4	uiUse	Flagging for received scanner protocols

6.4. List of constants for TaskID and for the FEISC_TASK_INIT structure

The constants definitions are contained in the file FEISC.H or FEISC.BAS or FEISC.PAS.

Constants	Value	Use	Description
FEISC_TASKID_FIRST_NEW_TAG	1	iTaskID	one-time inventory
FEISC_TASKID_EVERY_NEW_TAG	2	iTaskID	repeating inventory
FEISC_TASKID_NOTIFICATION	3	iTaskID	unlimited task for receiving of notifications
FEISC_TASKCB_1	1	uiFlag	select of callback function cbFct1
FEISC_TASKCB_2	2	uiFlag	select of callback function cbFct2
FEISC_TASK_CHANNEL_TYPE_AS_OPEN	1	uiChannelType	for all inventary tasks
FEISC_TASK_CHANNEL_TYPE_NEW_TCP	5	uiChannelType	for notification task

6.5. History

V5.06.03

- New functions **FEISC_0xC0_SAMCmd**, **FEISC_0xBC_CmdQueue**, **FEISC_0xBB_C1G2_TranspCmd**

V5.05.05

- Optimizations in internal Notification-Thread (activated with **FEISC_StartAsyncTask**) for communication channels with higher error rate, like GPRS.
- New parameter for **FEISC_GetReaderPara** and **FEISC_SetReaderPara**: LogFilename

V5.05.01

- USB support for Linux
- New functions: **FEISC_0xB4_EPC_UHF_Cmd**, **FEISC_0x6B_CentralizedRFSync**
- New status bytes: 0x86, 0x18
- The Linux library is compiled with GCC 3.3.3 under SuSE Linux 9.1

V5.04.11

- Modified licence agreement
- New error code -4085

V5.04.10

- New Task: Notification for Reader with Notification Mode.
- Modifications in the structur **FEISC_TASK_INIT**. This structure is not compatible to the previous version.
- New function: **FEISC_0x34_ForceNotifyTrigger**
- All threads available under Linux
- Support for new status bytes: 0xF1, 0xF2, 0xF8
- New error codes: FEISC_ERR_TASK_SOCKET_INIT, FEISC_ERR_TASK_BUSY

V5.04.00

- New functions **FEISC_StartAsyncTask**, **FEISC_CancelAsyncTask** and **FEISC_TriggerAsyncTask**.
- New error codes

V5.03.09

- New function **FEISC_0x72_SetOutput**.
- **FEISC_0x22_ReadBuffer** supports extended features (TR-DATA, INPUT, STATUS).

V5.03.03

- New function **FEISC_0xB3_EPCCmd**.
- **FEISC_Transmit** and **FEISC_Receive** can be used with all port types.
- New status byte: 0x96 (ISO14443-Error)

V5.03.00

- The new version is not 100% downward compatible with the previous version because of rename of function **FEISC_0x66_FirmwareVersion**. The new name is **FEISC_0x66_ReaderInfo**.

V5.02.00

- Prepared for comming soon new USB protocols.
- The new version is not 100% downward compatible with the previous version because of rename and modification of the parameter list of function **FEISC_0x18DestroyEPC**. The new name is **FEISC_0x18Destroy**.
- New error code: -4055.
- Some minor bug fixes.

V5.01.19

- Support of Transponder I-CODE UID in protocol [0x18] Destroy.
- First Linux Release (SuSE Linux 8.2, GNU Compiler Collection V3.3-23, glibc V2.3.2-6)

V5.01.17

- Plug-In mechanism for integration of user-defined protocol drivers.
- All functions, except of **FEISC_BuildProtocol** and **FEISC_SplitProtocol**, are 100% downward compatible with the previous version.
- **FEISC_BuildProtocol** is renamed in **FEISC_BuildSendProtocol** and has changes in the function parameters.

- **FEISC_SplitProtocol** is renamed in **FEISC_SplitRecProtocol** and has changes in the function parameters.
- New functions:
  FEISC_BuildRecProtocol
  FEISC_SplitSendProtocol
  FEISC_Conv2StdProtocol
  FEISC_Conf2AdvProtocol
  FEISC_InstallPlugIn
  FEISC_RemovePlugIn
- New protocol functions:
  FEISC_0x22_ReadBuffer
  FEISC_0x18_DestroyEPC
  FEISC_0x87_SetSystemDate
  FEISC_0x88_GetSystemDate
  FEISC_0x64_SystemReset.
- Support of the protocol [0x74] Read Input for ID ISC.PRH-A and -U Reader.
- Support of Advanced Protocol Frames with two length bytes.
- Thread-Security for created Reader-Objects.
- Support of multithreading: every created Reader-Object has an own internal buffer. This enables the simultaneous operation of multiple readers if every reader is connected on different ports.
- New error codes:

Error constants	Value	Description
FEISC_ERR_NO_VALUE	-4010	Error in the function FEISC_GetReaderPara
FEISC_ERR_NO_PLUGIN	-4060	Installation of Plug-In object in reader object is missing
FEISC_ERR_PLUGIN_PRESENT	-4061	Error while installation of a second Plug-In object to a reader object
FEISC_ERR_UNKNOWN_PLUGIN_ID	-4062	Unknown Plug-In ID
FEISC_ERR_PI_BUILD_DATA	-4063	Return value for an error in the Plug-In function build_datastream
FEISC_ERR_PI_BUILD_FRAME	-4064	Return value for an error in the Plug-In function build_protocol
FEISC_ERR_PI_SPLIT_FRAME	-4065	Return value for an error in the Plug-In function split_protocol
FEISC_ERR_PI_SPLIT_DATA	-4066	Return value for an error in the Plug-In function split_datastream
FEISC_ERR_BUFFER_OVERFLOW	-4070	Databuffer is too small

V5.01.00

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

- New functions: **FEISC_0xBD_ISOTranspCmd**, **FEISC_0xBE_ISOTranspCmd**
- Integration of TCP/IP support if the support package ID FETCP is used
- Bug-fix in in **FEISC_0xBF_ISOTranspCmd** for parameter iDataType=1
- Bus address of last receive protocol is saved and can be read out with **FEISC_GetReaderPara**
- new error code: -4054 (FEISC_ERR_UNSUPPORTED_OPTION)

V5.00.00

- The new version is 100% downward compatible with the previous version.
- New functions: **FEISC_0xA2_WriteMifareKeys**, **FEISC_0xB2_ISOCmd**
- First Windows CE Version

V4.09.00

- Move of all constants from the file ferwdef.h to the file ferw.h. The file ferwdef.h is now dispensable.
- new function: **FEISC_0x55_StartFlashLoaderEx** which supports any busaddress and replaces **FEISC_0x55_StartFlashLoader**.
- Internal check of received bus address (normally disabled).
- new parameter ChkRecBusAdr for the functions **FEISC_SetReaderPara** and **FEISC_GetReaderPara** to activate the check of the received busaddress.
- new parameter ConvHexToString for the functions **FEISC_SetReaderPara** and **FEISC_GetReaderPara** to activate the conversion of raw hex data received from reader in scan mode.
- new error code FEISC_ERR_REC_BUS_ADR
- new uiFlag constant for the structure FEISC_EVENT_INIT: FEISC_CALLBACK_2
- new uiUse constant for the structure FEISC_EVENT_INIT: FEISC_SCANNER_EVENT

V5.06.00 – V4.08.00

- New function **FEISC_0x6F_AntennaTuning**
- Removed functions:

  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

- Internal Versions.

V4.03.00

- Change of the function parameters in **FEISC_0xBF_ISOTranspCmd.**

V4.02.00

- Check of the command byte in the response protocol
- Error correction for USB-Protocols
- Correction of small errors

V4.01.00

- New functions: **FEISC_GetStatusText**, **FEISC_0xB1_ISOCustAndPropCmd**, **FEISC_0xBF_ISOTranspCmd**.

V4.00.00

This is the official Release Version. No changes.

V3.01.00

- FEISC.DLL only works together with FECOM.DLL in Version 2.00.00 and higher. Older versions of FECOM.DLL will not allow communication to take place.
- Event flagging now also supports Windows API events.

V3.00.00

- Support of OBID® USB devices
- New functions: **FEISC_GetErrorText**, **FEISC_GetLastError**, **FEISC_AddEventHandler**, **FEISC_DelEventHandler**.
- Limiting the port handle (transfer parameter *iPortHnd* in **FEISC_NewReader**) to 0x0FFFFFFF. The first byte (MSB) is reserved for distinguishing between the communication channels (asynchronous/USB).

V2.01.00

- New parameters for **FEISC_GetReaderPara**: ERRCODE, ERRSTR, SENDSTR, RECSTR
- Renaming of the functions **FEISC_0x85_SetTime** in **FEISC_0x85_SetSysTimer** and **FEISC_0x86_GetTime** in **FEISC_0x86_GetSysTimer**.
- New functions: **FEISC_0x55_StartFlashLoader**, **FEISC_0x6E_RdDiag** and **FEISC_0xA0_RdLogin**.

Version 2.00.03

- Eliminates errors in **FEISC_0x01_MultiJobPoll**, **FEISC_0x01_MultiJobPollAndState** and **FEISC_0x03_MultiJobState**.
- New addition of command parameters for supporting multithreading: 5.7. Support for multithreading see and the function **FEISC_0x75_AdjAntenna**.

Version 2.00.01

- Renaming of the function **FEISC_0x23_InitBuffer** to **FEISC_0x33_InitBuffer**, since the command byte of the protocol has changed.

V2.00.00

New functions for the 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**

In addition the following functions were added to the parameter list:

1. **FEISC_BuildProtocol**: The parameter iDataType is new
2. **FEISC_SplitProtocols**: The parameter iDataType is new
3. **FEISC_GetLastSendProt**: The parameter iDataType is new
4. **FEISC_GetLastRecProt**: The parameter iDataType is new
5. **FEISC_SendTransparent**: The parameter iDataType is new

6. **FEISC_Transmit**: The parameter iDataType is new
7. **FEISC_Receive**: The parameter iDataType is new
8. **FEISC_0x80_ReadConfBlock**: The parameter iDataType is new
9. **FEISC_0x81_WriteConfBlock**: The parameter iDataType is new

We have done this for the sake of Visual Basic programmers.

New query function added:

FEISC_GetLastRecProtLen