Texas Instruments

HF Reader System Series 6000

S6500/S6550 Program Library FEISC

Reference Guide

11-06-21-062 June 2001

Edition One - June 2001

This is the first edition of this manual. It describes the S6500/S6550 Program Library FEISC.

Read This First

About This Manual

This reference guide for the S6500/S6550 Program Library FEISC is designed for use by TI partners who are experienced with software development.

Conventions

WARNING:

A WARNING IS USED WHERE CARE MUST BE TAKEN, OR A CERTAIN PROCEDURE MUST BE FOLLOWED IN ORDER TO PREVENT INJURY OR HARM TO YOUR HEALTH.

CAUTION:

This indicates information on conditions which must be met, or a procedure which must be followed, which if not heeded could cause permanent damage to the equipment or software.

Note:

Indicates conditions which must be met, or procedures which must be followed, to ensure proper functioning of the equipment or software.

Information:

Indicates information which makes usage of the equipment or software easier

If You Need Assistance

For more information, please contact the sales office or distributor nearest you. This contact information can be found on our web site at:

http://www.ti-rfid.com

Licensing agreement for use of the software

This is an agreement between you and Texas Instruments Deutschland GmbH (hereafter “Texas Instruments”) for use of the 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 Texas Instruments and its suppliers, and is protected by international copyright.

Object and scope of the agreement

1. Texas Instruments 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 file server. You may create backup copies of the licensing material.
3. Texas Instruments 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 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 Texas Instruments.

§2. Protection of the licensing material

1. The licensing material is the intellectual property of Texas Instruments 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 Texas Instruments 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 Texas Instruments 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 Texas Instruments. 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 Texas Instruments that is not possible to develop EDP programs such that they are error-free for all application conditions. Texas Instruments 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. Texas Instruments 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. Texas Instruments 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. Texas Instruments assumes no liability for such actions, regardless of whether they are carried out intentionally or unintentionally.
3. Texas Instruments provides the software “as is” and without any warranty. Texas Instruments cannot guarantee the performance or the results you obtain from using the software. Texas Instruments 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 Concluding provisions

1. This Agreement contains the complete licensing terms and conditions and supersedes 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.

Document Overview

List of Figures

CHAPTER 1

Introduction

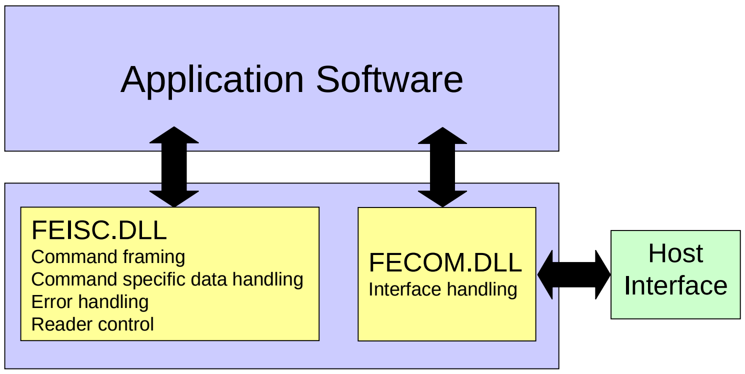
This chapter introduces you to the S6500/S6550 Program Library FEISC and tells you how to install it.

Topic **Page**

1.1 Introduction

The Program Library FEISC assists you in programming application software and integrating the S6500/S6550 reader into your system. It supports the functionality of the reader. Together with the Program Library FECOM (described in document number: 11-06-21-063) which supports the serial interface communication it makes it possible to run all the protocols described in the S6500/S6550 Configuration and Host Protocol Reference Guide (document number: 11-06-21-064) by directly invoking a function. You can use this program library to build and split protocols to implement your own interface functions (refer to section 2.1 “Overview” for further explanations).

Figure 1: Software System Overview



It supports the languages ANSI-C, ANSI-C++ and Microsoft Visual Basic[1], as well as any other language which can invoke C functions. This support package can only be used under MS Windows 9x/ME/2000 and NT (3.51 or higher).

The Program Library FEISC package consists of the components listed in the following table and is available at the software download section on the TI-RFID homepage:

http:/www.ti-rfid.com

File	Use
FEISC.DLL	DLL with all functions
FEISC.LIB	LIB file for linking with C/C++ projects
FEISCBOR.LIB	LIB file for linking with C/C++ projects using Borland compilers
FEISC.H	Header file for C/C++ projects
FEISCDEF.H	Header file with error codes for C/C++ projects (optional)
FEISC.BAS	Declare file for Visual Basic projects
FEISC.PAS	Declare file for Delphi Projects

1. Visual Basic 4.0 or higher.

1.2 Installation

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

- Copy FEISC.DLL into the Windows program or system directory (the latter is recommended).

C/C++ programmers:

- Copy FEISC.LIB into the project or LIB directory, or add FEISCBOR.LIB to the project.

Copy FEISC.H and (optionally) FEISCDEF.H into the project or INCLUDE directory.

Visual Basic programmers:

- Copy FEISC.BAS into the project directory.

Delphi programmers:

- Copy FEISC.PAS to the project directory.

1.3 Incorporating into the application program

C/C++ programmers:

As soon as the LIB file is made known to the development tool, any function may be used immediately. This presumes of course the declaration of the DLL functions with an INCLUDE instruction within each source file that invokes FEISC functions.

For Delphi programmers:

Add “FEISC” to the USES statement in each source file of your project that invokes FEISC.DLL functions.

For Visual Basic programmers:

Add the file FEISC.BAS to your project.

The S6500/S6550 Program Library FEISC must also be incorporated into your project.

CHAPTER 2

Programming Interface

This chapter introduces you to the S6500/S6550 Program Library FEISC.

Topic **Page**

Continued on next Page

Topic **Page**

2.1 Overview

The Program Library FEISC comprises all the functions and parameters necessary for simple communication with the S6500/S6550 Readers for the programmer. Together with the Program Library FECOM it makes it possible to run all the protocols described in the S6500/S6550 Host Protocol Reference Guide 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.DLL alone is not enough to communicate with an S6500/S6550 Reader. You can however initiate the output of a protocol and use the FECOM.DLL to communicate with an S6500/ S6550 Reader over an asynchronous serial interface. No other interface drivers are supported. If you are restricted to using your own asynchronous serial interface driver it will considerably reduce the functional usability of the FEISC.DLL, additionally you will have to code each protocol traffic through the asynchronous serial interface. In this case the FEISC.DLL provides the support for protocol building (**FEISC_BuildProtocol**) and splitting (**FEISC_SplitProtocol**).

The core elements of the DLL 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 of all the reader objects that are generated. 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.DLL. Each reader object administers all the parameters relevant to its protocol tasks within its local memory.

If you have included the DLL in your project, the DLL will be automatically opened when the application is started. This gives you immediate access to all the DLL's functions.

Before you use the system for the first time 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 **FEISC_DeleteReader** function.

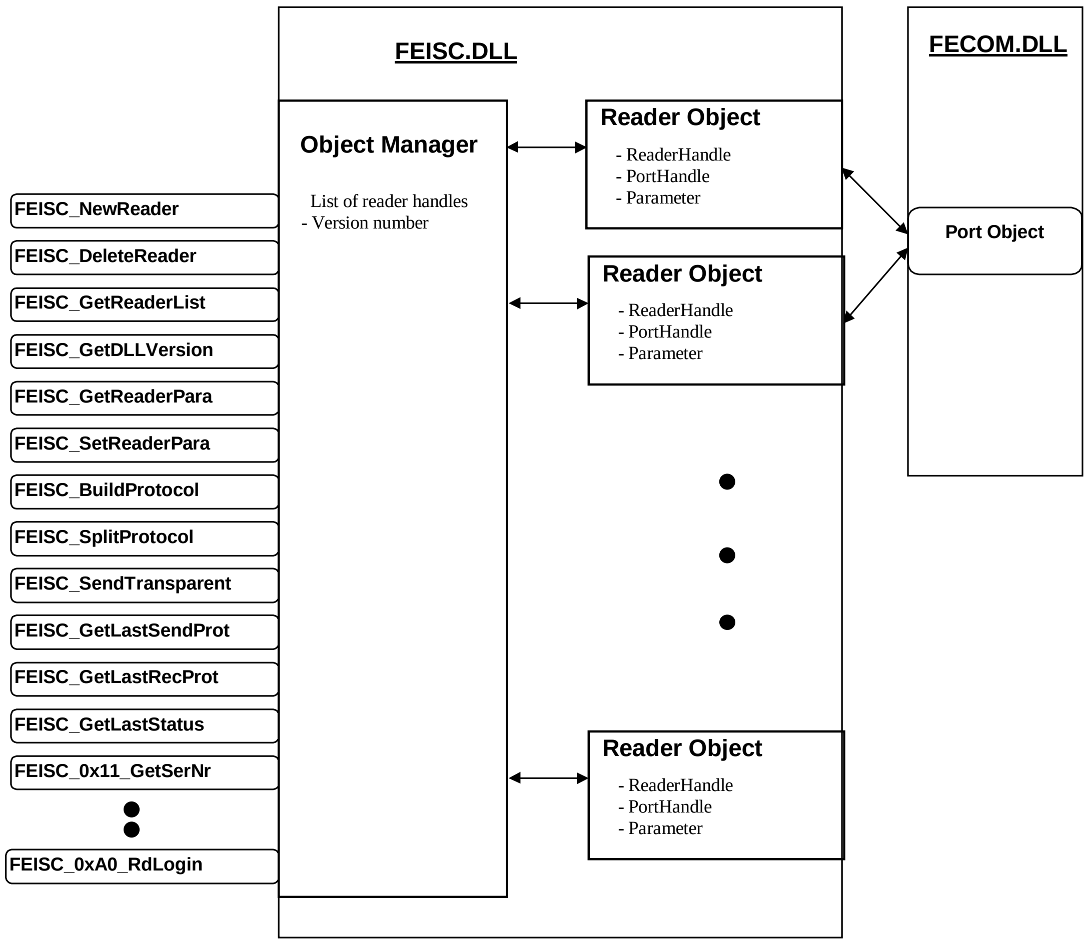
If an application program is opened more than once, each program (instance) gets an empty object list by invoking **FEISC_GetReaderList**. This prevents access rights being confused under different program instances.

The object-oriented internal structure (see Figure 2) is externally visible as a function interface, making it language-neutral.

Figure 2 shows how several *reader objects* can share a common serial interface in FECOM. 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 or FEISC.

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

Figure 2: FEISC.DLL Internal structure



2.2 Parameter transfer

Some functions support parameter transfer both as a null-terminated string and as an array of hexadecimal 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”; and so on)	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 (for example: 0x23). It is not possible to transfer by strings!

Note:

UCHAR is used as an abbreviation (#define) for “unsigned char”. In Visual Basic and Delphi the compatible data type is the Byte (see contents of FEISC.BAS/ FEISC.PAS).

2.3 Event flagging to applications

When used with Visual Basic projects there are some limitations.

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. The function gets a pointer to the protocol string and can then for example display this string in a window.

An event handling mechanism must be installed using the **FEISC_AddEventHandler** function. You can choose between four various flagging methods:

- Message to a calling process
- Message to a window
- Use of a callback function
- Flagging with a Windows-API event

An 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
                     (for example: WM_USER_xyz)
  UINT uiFlag;   //  Specifies use of the union (for example:
  union              FEISC_WND_HWND)
  {
    DWORD  dwThreadID;           // for Thread-ID
    HWND   hwndWnd;              // for Window-Handle
    void   (*cbFct)(int, int);   // for Callback-Function
    HANDLE hEvent;               // for Event-Handle
  }Method;[1]
} FEIS_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 a Windows-API event. The flag form is selected using the *uiFlag* parameter. You use the *uiUse* parameter to store a designator for assigning the handling method. For message methods you must store the message code in *uiMsg*.

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

1. The name “Method” for the union is only for C programmers. C++ programmers access the union directly through the structure.

2.4 List of functions

The Program Library FEISC contains a large number of functions for various tasks. They are divided into groups for better overview.

Note:

Most of the functions can be used only in conjunction (directly or indirectly) with the Program Library FECOM.

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)

Protocol functions

- **int FEISC_BuildProtocol** (int iReaderHnd, UCHAR cBusAdr, UCHAR cCmdByte, UCHAR* cSendData, int iDataLen, UCHAR* cSendProt, int iDataType)
- **int FEISC_SplitProtocol** (int iReaderHnd, UCHAR* cRecProt, int iRecLen, UCHAR* cBusAdr, UCHAR* cCmdByte, UCHAR* cData, int* iDataLen, int iDataType)

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 iSend-Len, 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_0x11_GetSerNr** (int iReaderHnd, UCHAR cBusAdr, UCHAR* cTR_TYP, UCHAR* cSNr, int iDataType)
- **int FEISC_0x21_ReadBuffer** (int iReaderHnd, UCHAR cBusAdr, UCHAR cSets, UCHAR* cTrData, UCHAR* cRecSets, UCHAR* cRecDataSets, int iDa-taType)
- **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_0x52_GetBaud** (int iReaderHnd, UCHAR cBusAdr)
- **int FEISC_0x55_StartFlashLoader** (int iReaderHnd)
- **int FEISC_0x63_CPUReset** (int iReaderHnd, UCHAR cBusAdr)
- **int FEISC_0x65_SoftVersion** (int iReaderHnd, UCHAR cBusAdr, 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_InitNoiseThreshold** (int iReaderHnd, UCHAR cBusAdr)
- **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_0x71_SetOutput** (int iReaderHnd, UCHAR cBusAdr, UCHAR cOut-put)
- **int FEISC_0x74_ReadInput** (int iReaderHnd, UCHAR cBusAdr, UCHAR* cIn-put)
- **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_0xB0_ISOCmd** (int iReaderHnd, UCHAR cBusAdr, UCHAR* cReq-Data, 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_0xBF_ISOTRanspCmd** (int iReaderHnd, UCHAR cBusAdr, int iRspLen, int UCHAR* cReqData, int iReqLen, UCHAR* cRspData, int* iRspLen, int iDataType)

2.4.1 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*[a] is the handle of a port object created from FECOM.DLL using the **FECOM_OpenPort** function. This handle allows protocols to be directly passed on to FECOM.DLL. Transfer of a 0 is also permitted. Multiple Reader objects can in principle carry out their communication over the same serial COM port. iPortHnd uses the first byte (MSB) of the PortHandle to specify protocol output to FECOM: iPortHnd = 0x0XXXXXXX[b] indicates output to FECOM.DLL You can 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 DLL is not freed up again.
Return value	If a Reader object was created without error, a handle (>0) is returned. If an error occurs, the function returns a value less than zero. A list of error codes can be found in Appendix A.

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);`
	`}`

a. iPortHnd is also used throughout this Reference Guide to indicate iDevHnd.
b. 0x0XXXXXXX represents any hex value.

2.4.2 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. If an error occurs, the function returns a value less than zero. A list of error codes can be found in Appendix A.

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);
     ...
}
...
...
```

2.4.3 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 you transmit 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. If an error occurs, the function returns a value less than zero. A list of error codes can be found in Appendix A.

Example	`#include "feisc.h"`
	`...`
	// Example function to readout 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
	`}`

2.4.4 FEISC_GetDLLVersion

Function	Gets the DLL version number.
Syntax	**void FEISC_GetDLLVersion (char* cVersion)**
Description	The function returns the version number of the DLL. *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 ”04.01.00”. 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
	`...`
	`...`

2.4.5 FEISC_GetErrorText

Function	Gets error text for error code
Syntax	**int FEISC_GetErrorText (int iErrorCode, char* cErrorText)**
Description	This function uses cErrorText to send the English 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. A list of error codes can be found in Appendix A.

Example	`#include “feisc.h”`
	`#include “feiscdef.h”`
	`...`
	`...`
	`char cErrorText[256];`
	`...`
	
	`int iBack = FEISC_GetErrorText(FEISC_ERR_PROTLEN, cErrorText)`
	        // code here for displaying the text
	`...`
	`...`

2.4.6 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 the short English text associated with the *ucStatus*[a]. 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 an error occurs, the function returns a value less than zero. A list of error codes can be found in Appendix A.

Example	`#include “feisc.h”`
	`...`
	`...`
	`char cStatusText[128];`
	`...`
	
	`int iBack = FEISC_GetStatusText(0x01, cStatusText)`
	        // code here for displaying the text
	`...`
	`...`

a. The list of status bytes can be found in the S6500/S6550 Host Protocol Reference Guide.

2.4.7 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[a], LogProt, BusyTO, RepCnt, ErrCode, ErrStr, SendStr, RecStr and IsProtToAppLocked
Cross-reference	For more information refer to: section 2.5 “Support for multi-threading” and Appendix B “List of variables”
Return value	If no error occurs, the function returns a value of 0, and if an error occurs, the function returns a value less than zero. A list of error codes can be found in Appendix A.

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
	`...`
	`     }`
	`...`
	`...`
	`}`

a. Note here the comments about the PortHandle in section 2.4.1.

2.4.8 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[a], LogProt, BusyTO, RepCnt, LockProtToApp and UnlockProtToApp
Cross-reference	For more information refer to section 2.5 “Support for multi-threading” and Appendix B “List of variables”
Return value	If the Reader object with the new parameter value was successfully (error-free) installed, a 0 is returned. If an error occurs, the function returns a value less than zero. A list of error codes can be found in Appendix A.

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
	`     ...`
	`}`
	`...`
	`...`

a. Note here the comments about the PortHandle in section 2.4.1.

2.4.9 FEISC_AddEventHandler

Function	Installs an event handling mechanism
Syntax	**int FEISC_AddEventHandler (int iReaderHnd, FEISC_EVENT_INIT* pInit)**
Description	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.

Event	*Description*
FEISC_PRT_EVENT	One event each for the send and receive protocol[a]
FEISC_SNDPRT_EVENT	Event for send protocol
FEISC_RECPRT_EVENT	Event for receive protocol

1st Method: Message to thread (not for Visual Basic)

This method is used for exchanging messages between Threads[b]. 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 FECOM 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
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.

Description (continued)	2nd Method: Message to window (not for Visual Basic)
	This method is used when the message needs to be sent directly to a window. The corresponding window uses the Windows-API function GetWindow (..)[c] to get the 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 PostThreadMessage(..). 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
	`hwndWnd = GetWindow(...)`
	The MessageMap function gets the same parameters as in the first method.
	
	3rd method: Invoking a callback function
	The 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`[d]
	
	4th method: Setting an event
	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`
	`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 refer to sections 2.3 and 2.5.
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 Appendix A.

a. Event is only generated if the parameter LogProt is set to 1 (default 0)
b. Parallel execution path independent of the application program. The program itself is a thread.
c. When using MFC CWnd you can also use the GetSafeHwnd() method
d. The function has the prototype: void YourFunctionName(int, int)

2.4.10 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.
	1st method of deleting: Message to Thread (not for Visual Basic)
	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()`
	2nd method of deleting: Message to Window (not for Visual Basic)
	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(...)`
	3rd method of deleting: Invoking a 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`
	4th method of deleting: Setting an event
	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 refer to sections 2.3 and 2.5.
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 Appendix A.

2.4.11 FEISC_BuildProtocol

Function	The transmitted parameters and data are used to build a protocol with a protocol frame.
Syntax	**int FEISC_BuildProtocol (int iReaderHnd, UCHAR cBusAdr, UCHAR cCmdByte, UCHAR* cSendData, UCHAR cDataLen, 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 (*cDataLen*) 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 S6500/S6550 Host Protocol Reference Guide. *iReaderHnd* is the handle for the Reader object. The constructed protocol is not passed along to FECOM, making it independent of FECOM.
Cross-reference	For more information on *iDataType* see Section 2.2. **FEISC_SplitProtocol**
Return value	If no errors occur, the length of *cSendProt* is indicated in the return value. If an error occurs, the function returns a value less than zero. A list of error codes can be found in Appendix A.
Example	see below

```
int BuildTestProtocol (int iReaderHnd)
{
   int iErr;
   UCHAR cDataLen, cSendData[32], cSendProt[256];
   UCHAR cBusAdr = 0xFF;
   UCHAR cCmdByte= 0x6A;
   ...
   cSendData[0]           = 0x01;
   cSendData[1]           = '\0';
   cDataLen               = 0x01;

   // Build send protocol
   iErr = FEISC_BuildProtocol (iReaderHnd, cBusAdr, cCmdByte,
                               cSendData,  cDataLen,  cSend-
                               Prot, 0);
...
}
```

2.4.12 FEISC_SplitProtocol

Function	Splits the transmitted protocol string.
Syntax	**int FEISC_SplitProtocol (int iReaderHnd, UCHAR* cRecProt, UCHAR cRecLen, UCHAR* cBusAdr, UCHAR* cCmdByte, UCHAR* cRecData, UCHAR* cDataLen, 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 (*cDataLen*). The protocol string in *cRecProt* must be transmitted as a hex array (*iDataType=0*) or string (*iDataType=1*) with a length indication in *cRecLen*. cRecData is interpreted as a hex array (*iDataType=0*) or string (*iDataType=1*). For more information about the protocol frame, see the S6500/S6550 Host Protocol Reference Guide. *iReaderHnd* is the handle for the Reader object. The constructed protocol is not passed to the Interface Control (FECOM).
Cross-reference	For more information about *iDataType* refer to section 2.2 "Parameter Transfer" **FEISC_BuildProtocol**
Return value	In case of no errors occur 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 Appendix A.
Example	see below

```
// the following code fragment presupposes initialized port and Reader objects.
#include "feisc.h"
#include "fecom.h"
...
     int iStatus, iRecLen;
     UCHAR cBusAdr, cCmdByte, cDataLen;
     UCHAR cSendProt[256], cRecProt[256], cRecData[256];
     // Build send protocol
     FEISC_BuildProtocol(iReaderHnd,    cBusAdr,    cCmdByte,
                         cSendData, cDataLen, cSendProt, 0);
     // Send and receive protocol
     iRecLen = FECOM_Transceive(iPortHnd,  cSendProt,  cSend-
                                Prot[0], cRecProt, 256);
     if (iRecLen > 0)
     {                    // Split receive protocol
     iStatus = FEISC_SplitProtocol(iReaderHnd,     cRecProt,
                                   (UCHAR)iRecLen,   &cBusAdr,
                                   &cCmdByte, cRecData, &cDa-
                                   taLen, 0);
     if (iStatus == 0)  // Statusbyte == 0x00
                          {// Process receive data
                          ...
                          }
     }
```

2.4.13 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 iRecLen, 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 *iRecLen*. *iReaderHnd* is the handle for the Reader object.
Cross-reference	For more information about *iDataType* refer to section 2.2 “Parameter Transfer”
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 Appendix A.
Example	see below

```
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
                                 ...
      }
```

2.4.14 FEISC_Transmit

Function	Outputs a protocol string directly over the interface.
Syntax	**int FEISC_Transmit (int iReaderHnd, UCHAR* cSendProt, int iSendLen, int iCheckSum[a], 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 about *iDataType* refer to section 2.2 “Parameter Transfer”
Return value	In case of error a 0 is transferred. A list of error codes can be found in Appendix A.

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
	FEIC_Transmit (iReaderHnd, cSendProt, outLen, 1, 0);
	`...`

a. A description of the CRC calculation can be found in the S6500/S6550 Host Protocol Reference Guide.

2.4.15 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 the S6500/S6550 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*. *iReaderHnd* is the handle for the Reader object.
Cross-reference	For more information about *iDataType* refer to section 2.2 “Parameter Transfer”
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 Appendix A.

Example	`int inLen;`
	`UCHAR cRecProt[256];`
	`...`
	// Receive protocol
	`inLen = FEISC_Receive (iReaderHnd, cRecProt, 256, 0);`
	`...`

2.4.16 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 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). *iReaderHnd* is the handle for the Reader object.
Cross-reference	For more information about *iDataType* refer to section 2.2 “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 Appendix A.

2.4.17 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). **iReaderHnd** is the handle for the Reader object.
Cross-reference	For more information about *iDataType* refer to section 2.2 “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 Appendix A.

2.4.18 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 English 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 Appendix A.

2.4.19 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 Appendix A.

2.4.20 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 English 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 Appendix A.

Example	`#include "feisc.h"`
	`...`
	`...`
	`char cErrorText[256];`
	`int iErrorCode = 0;`
	`...`
	
	`int iBack = FEISC_GetLastError (iReaderHnd, &iErrorCode, cEr-rorText)`
	// code here for displaying the text
	`...`
	`...`

2.4.21 FEISC_0x11_GetSerNr

Function	Function for selecting a transponder and reading the serial number.
Syntax	**int FEISC_0x11_GetSerNr (int iReaderHnd, UCHAR cBusAdr, UCHAR* cTR_TYP, UCHAR* cSNr, int iDataType)**
Description	The function selects a transponder located in the active zone of the S6500/S6550 Reader. If the transponder was able to be selected, the transponder type (*cTR_TYP*) and the serial number (*cSNr*) are retrieved. The parameter *iDataType* specifies whether the serial number in *cSNr* is to be interpreted as a hex array (*iDataType*=0) or as a string (*iDataType*=1). The length of *cTR_TYP* is 1. The buffer for the serial number *cSNr* must be able to hold at least 9 bytes (*iDataType*=0) or 17 bytes (*iDataType*=1). Note that 1 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 about *iDataType* refer to section 2.2 “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 Appendix A.

2.4.22 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 S6500/S6550 Host Protocol Reference Guide. *iReaderHnd* is the handle for the Reader object. *cBusAdr* is the bus address set in the Reader.
Note	The function does not check the data in cRecDataSets based on the data structure indicated in cTrData.
Cross-reference	For more information about *iDataType* refer to section 2.2 “Parameter Transfer” **FEISC_0x33_InitBuffer, FEISC_0x31_ReadDataBufferInfo, FEISC_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 Appendix A.

2.4.23 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 Reader.
Cross-reference	For more information about *iDataType* refer to section 2.2 “Parameter Transfer” **FEISC_0x21_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 Appendix A.

2.4.24 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 Reader.
Cross-reference	**FEISC_0x21_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 Appendix A.

2.4.25 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_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 Appendix A.

2.4.26 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 Appendix A.

2.4.27 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 Appendix A.

2.4.28 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 Appendix A.

2.4.29 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 fetched 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 about *iDataType* refer to section 2.2 “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 Appendix A.

2.4.30 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 Appendix A.

2.4.31 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 Appendix A.

2.4.32 FEISC_0x6B_InitNoiseThreshold

Function	Function for initializing the noise threshold.
Syntax	**int FEISC_0x6B_InitNoiseThreshold (int iReaderHnd, UCHAR cBusAdr)**
Description	*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 Appendix A.

2.4.33 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 about *iDataType* refer to section 2.2 “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 Appendix A.

2.4.34 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 about *iDataType* refer to section 2.2 “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 Appendix A.

2.4.35 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 about *iDataType* refer to section 2.2 “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 Appendix A.

2.4.36 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 S6500/S6550 Host Protocol Reference Guide 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 Appendix A.

2.4.37 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 Appendix A.

2.4.38 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 about *iDataType* refer to section 2.2 “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 Appendix A.

2.4.39 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 about *iDataType* refer to section 2.2 “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 Appendix A.

2.4.40 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 Appendix A.

2.4.41 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 Appendix A.

2.4.42 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.
Cross-reference	For more information about *iDataType* refer to section 2.2 “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 Appendix A.

2.4.43 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.
Cross-reference	For more information about *iDataType* refer to section 2.2 “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 Appendix A.

2.4.44 FEISC_0xB0_ISOCmd

Function	Function initiates data transfer with an ISO transponder.
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 ISO transponders located in the active zone of the S6500/S6550 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 ISO 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 about *iDataType* refer to section 2.2 “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 Appendix A.

2.4.45 FEISC_0xB1_ISOCustAndPropCmd

Function	Function initiates data transfer with an ISO 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 ISO transponders located in the active zone of the S6500/S6550 Reader. The parameter *cMfr* contains the Manufacturers Code and specifies the structure of the request 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 ISO 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 about *iDataType* refer to section 2.2 “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 Appendix A.

2.4.46 FEISC_0xBF_ISOTranspCmd

Function	Function initiates data transfer with an ISO transponder.
Syntax	**int FEISC_0xBF_ISOTRanspCmd (int iReaderHnd, UCHAR cBusAdr, int iRspLen, UCHAR* cReqData, int iReqLen, UCHAR* cRspData, int* iRspLen, int iDataType)**
Description	The function initiates a data transfer for multiple ISO transponders located in the active zone of the S6500/S6550 Reader. The parameter *iRspLength* specifies the expected length of the received data to be stored in *cRspData*. The parameter *iCmdRspDly* specifies response delay time. 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 ISO 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 about *iDataType* refer to section 2.2 "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 Appendix A.

2.5 Support for Multi-threading

The functions in FEISC.DLL are essentially thread-safe, meaning function calls from several threads to the DLL 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 DLL which preclude a pre-emptive 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 (for example: 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.DLL offers some help here in that the protocol output in the DLL 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 DLL 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.

<u>C++ Example with MFC:</u>

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.DLL 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 DLL that all further protocol outputs are to be locked
    FEISC_SetReaderPara(0, "LockProtToApp", "");

    // Query to DLL 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();
}
```

APPENDIX A

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_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
EISC_ERR_CHECKSUM	-4031	Checksum error
FEISC_ERR_BUSY_TIMEOUT	-4032	Timeout after continuous busy messages
FEISC_ERR_NO_RECPROTOCOL	-4033	Unknown status byte
FEISC_ERR_CMD_BYTE	-4035	Wrong command byte 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

APPENDIX B

List of Variables

Variable	Value range	Default	Units	Description
PortHnd[a]	0 ... 4294967295	0		PortHandle for communication with FECOM
LogProt	0, 1	0		If 1, then protocol are output through callback function cbPtr
BusyTO	0 ... 60	5	s	Timeout for continuously busy status (Reader)
RepCnt	1 ... 99	1		Number of repetitions after Status 0x01 in FEISC_0x11_GetSerNr
SendStr	-	-	-	Provides last end protocol with preceding date and time of day
RecStr	-	-	-	Provides last receive protocol with preceding date and time of day
LockProtToApp	none	-		Multithreading support: Locks the protocol output through callback function in all Reader objects see 2.5. Support for multithreading
UnlockProtToApp	none	-		Multithreading support: Unlocks protocol output through callback function see 2.5. Support for multithreading
IsProtToAppLocked	none2.5	-		Multithreading support: Asks whether all Reader objects are finished with protocol output see 2.5. Support for multithreading

a. Note the remarks in 2.4.1

FEISC_EVENT_INIT Structure

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

Constant	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_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