TEXAS INSTRUMENTS

HF Reader System Series 6000

S6500/S6550 Program Library FECOM

Reference Guide

11-06-21-063 June 2001

Edition One - June 2001

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

PREFACE

Read This First

About This Manual

This reference guide for the S6500/S6550 Program Library FECOM 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 FECOM 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 FECOM.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

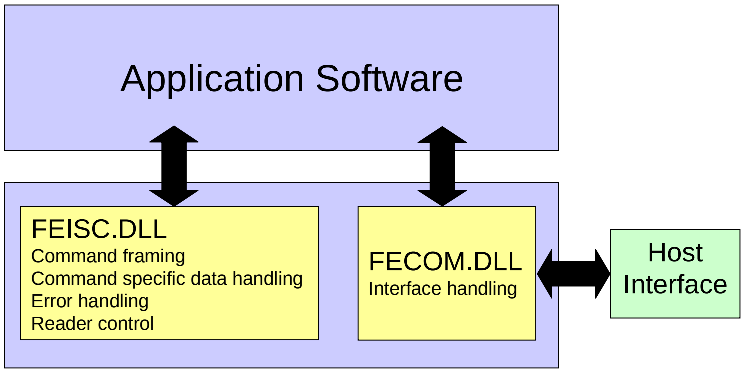
This chapter introduces you to the S6500/S6550 Program Library FECOM and tells you how to install it.

Topic **Page**

1.1 Introduction

The Program Library FECOM assists you in programming communications oriented software. It supports the serial interface communication. Together with the Program Library FEISC (described in document number: 11-06-21-062) which supports the functionality of the reader 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.

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.

The Program Library FECOM provides a simple function interface for the serial interface of a PC running under Windows 95/98/ME/NT[2] and 2000.

The Program Library 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
FECOM.DLL	DLL with all functions
FECOM.LIB	LIB file for linking with C/C++ projects
FECOMBOR.LIB	LIB file for linking with C/C++ projects using the Borland compiler
FECOM.H	Header file for C/C++ projects
FECOMDEF.H	Header file with error codes for C/C++ projects (optional)
FECOM.BAS	Declare file for Visual Basic projects
FECOM.PAS	Declare file for Delphi projects

1. Visual Basic 4.0 or higher.
2. NT Version 3.51 or higher

1.2 Installation

Installation must be performed manually:

- Copy FECOM.DLL into the project directory or the Windows system directory.

C/C++ programmers:

- Copy FECOM.LIB or FECOMBOR.LIB into the project or LIB directory.
- Copy FECOM.H and (optional) FECOMDEF.H into the project directory or INCLUDE directory.

Visual Basic programmers:

- Copy FECOM.BAS into the project directory.

Delphi programmers:

- Copy FECOM.PAS into 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 FECOM functions.

For Delphi programmers:

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

For Visual Basic programmers:

Add the file FECOM.BAS to your project.

CHAPTER 2

Programming Interface

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

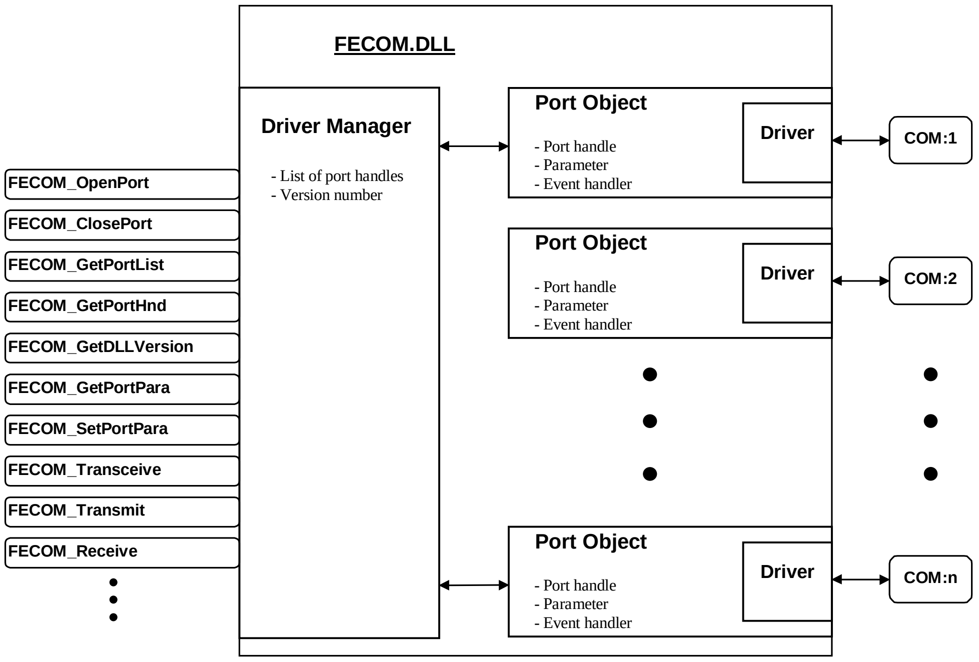
Topic **Page**

2.1 Overview

The FECOM DLL encapsulates all the functions and parameters which you need in order to manage one or more serial ports open at the same time. The object-oriented internal structure (see Figure 2) is intentionally configured as a function interface to the outside world. This gives it the advantage of being language-neutral.

The DLL has self-administration, thereby freeing the application program from having to buffer store any values, settings or other parameters. The driver manager in FECOM.DLL contains a list of all port objects generated, and each port object administers within its local memory all the settings relevant to its port.

Figure 2: FECOM.DLL - Internal Structure



Once you have embedded the DLL into your project it will be automatically opened when the application is started. This gives you immediate access to all DLL functions.

A port object must be created before you can first communicate. This is automatically performed by the **FECOM_OpenPort** function. If this function executes without error, a return value is received with a handle that can be administered by the application program. Unambiguous identification of the opened port object is only possible with this handle. The handle(s) does not however have to be saved in the application program, since the DLL driver manager maintains an internal list of all opened COM Ports. This list can be called with the **FECOM_GetPortList** function. The handles that you receive successively can be used with the **FEDCOM_GetPortPara** function to read out all the parameters pertaining to the port, including the port number.

A port object generated with **FECOM_OpenPort** must always be deleted from memory using the **FECOM_ClosePort** function, which also closes the COM port.

If an application is opened several times, each program (instance) receives an empty port list with the function call **FECOM_GetPortList**. This prevents a mixing of access rights under different program instances. A COM port can only be opened once, since it is a single physical object.

Every DLL function (exception: **FECOM_GetDLLVersion**) has a return value which is negative when an error occurs.

2.2 List of functions

Note:

UCHAR is used as an abbreviation (#define) for “unsigned char”.

In Visual Basic and Delphi the byte is the compatible data type (see contents of FECOM.BAS/FECOM.PAS).

- **int FECOM_OpenPort**(char* cPortNr)
- **int FECOM_ClosePort**(int iPortHnd)
- **int FECOM_GetPortList**(int iNext)
- **void FECOM_GetDLLVersion**(char* cVersion)
- **int FECOM_GetErrorText**(int iErrorCode, char* cErrorText)
- **int FECOM_GetLastError**(int iPortHnd, int* iErrorCode, char* cErrorText)
- **int FECOM_GetPortHnd**(char* cPortNr)
- **int FECOM_GetPortPara**(int iPortHnd, char* cPara, char* cValue)
- **int FECOM_SetPortPara**(int iPortHnd, char* cPara, char* cValue)
- **int FECOM_DoPortCmd**(int iPortHnd, char* cCmd, char* cValue)
- **int FECOM_AddEventHandler**(int iPortHnd, FECOM_EVENT_INIT* pInit)
- **int FECOM_DelEventHandler**(int iPortHnd, FECOM_EVENT_INIT* pInit)
- **int FECOM_Transceive**(int iPortHnd, UCHAR* cSendProt, int iSendLen, UCHAR* cRecProt, int iRecLen)
- **int FECOM_Transmit(**int iPortHnd, UCHAR* cSendProt, int iSendLen)
- **int FECOM_Receive**(int iPortHnd, UCHAR* cRecProt, int iRecLen)

2.3 Event flagging for control lines

Event handling mechanisms can be installed individually for each control line of any opened port for the control lines DTR, RTS, CTS, DCD and DSR. As soon as a control line changes its state, the appropriate signal is generated. This is a way of notifying an application of the event asynchronously with the program sequence.

An event handling mechanism must be installed using the **FECOM_AddEventHandler** function. You may select from among three various flagging methods:

Message to a calling process,
message to a window,
or use of a call back function.

Any installed event handling mechanism must be deleted using the **FECOM_DelEventHandler** function.

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

```
typedef struct _FECOM_EVENT_INIT
{
   UINT uiUse;    //  Defines the event (for example: FECOM_CTS_EVENT)
   UINT uiMsg;    //  Message code for dwThreadID and hwndWnd
                      (for example: WM_USER_xyz)
   UINT uiFlag;   //  Specifies use of the union (for example:
   union              FECOM_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]
} FECOM_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. In the *uiUse* parameter you store an ID for the control line to which you want to assign the handling method. For message methods you must store the message code in *uiMsg*.

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

Independent of the event flag you may query the status of any control line using the **FECOM_DoPortCmd** function.

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

2.3.1 FECOM_OpenPort

Function	Opens a serial port for communicating with an S6500/S6550 Reader
Syntax	**int FECOM_OpenPort(char* cPortNr)**
Description	The function uses standard parameters to open a serial port and internally stores a port structure for administering the parameters. For later changes to these parameters you can use the **FECOM_SetPortPara** function. Use **FECOM_GetPortPara** to read out these parameters. The returned handle iPortHnd identifies the port from the outside. *cPortNr* is a null-terminated string with the address of the serial port (for example: “1” for COM:1). Values between “1” and “256” are allowed. The serial port opened by **FECOM_OpenPort** must (!) be closed using the **FECOM_ClosePort** function. Otherwise the memory reserved by the DLL is not freed up.
Return value	If the serial port could be opened without error, a handle (>0) is returned. In case of error the function returns a value less than 0. The list of error codes can be found in Appendix A.
Standard-parameters	The standard parameters for the serial interface are: Baud: 9600; Frame: 8E1; Timeout: 600ms

Example

```
...
#include “fecom.h”
...
...
char cPortNr[4];
itoa(1, cPortNr, 10);// Convert Integer to Char
...
int handle = FECOM_OpenPort(cPortNr);// COM:1 should be opened
if(handle < 0)
{
    // code here for error condition
}
else
{   // Communication through COM:1, if successful received data are in RecBuf
    // code here for communication or other
}
```

2.3.2 FECOM_ClosePort

Function	Closes a serial port.
Syntax	**int FECOM_ClosePort(int iPortHnd)**
Description	The function closes the port defined in iPortHnd and frees up the reserved memory.
Return value	The return value is 0 if the serial port was closed. In case of error the function returns a value less than 0. The list of error codes can be found in Appendix A.
Example	see below

```
#include "fecom.h"
...
...
int Err;
char cPortNr[4];
...
itoa(1, cPortNr, 10);      // Convert Integer to Char
int handle = FECOM_OpenPort(cPortNr);// COM:1 should be opened
if(handle < 0)
{
        // code here for error condition
}
...
...
...
      if(handle > 0)
{     Err = FECOM_ClosePort(handle);
      ...
}
...
...
```

2.3.3 FECOM_GetPortList

Function	Uses the iNext parameter to get the first or succeeding PortHandle from the internal list of opened serial ports
Syntax	**int FECOM_GetPortList(int iNext)**
Description	Returns a PortHandle from the internal list of PortHandles. If you enter 0 for iNext, the first entry in the list is returned. If you enter a PortHandle contained in the list for iNext, the entry following that PortHandle is picked and returned. In this way you can scroll through the list from front to back and call up all entries.
Return value	If an entry was found, the PortHandle is returned with the return value. Once the end of the internal list is reached, i.e. the entered PortHandle has no successor, a 0 is returned. If no port is opened, FECOM_ERR_EMPTY_LIST is returned. In case of error the function returns a value less than 0. The list of error codes can be found in the Appendix A.

Example	`#include “fecom.h”`
	`...`
	`// Function gets the parameters of all open COM-Ports`
	`void COMList(void)`
	`{   int iNextHnd = FECOM_GetPortList(0);// get the first handle`
	`    while(iNextHnd > 0)`
	`    {  // here for example: code for reading out the COM parameters using FECOM_GetPortPara(...)`
	`       ...`
	`       iNextHnd = FECOM_GetPortList(iNextHnd);// get next handle`
	`    }`
	`...`
	`    // here for example: code for displaying the list`
	`...`
	`}`

Tip	When closing all open COM ports 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 closed port:
	`...`
	`iNextHnd = FECOM_GetPortList(0);// get the first handle`
	`while(iNextHnd > 0)`
	`{   iCloseHnd = iNextHnd;`
	`    iNextHnd = FECOM_GetPortList(iNextHnd);// get next handle only`
	`    iError = FECOM_ClosePort(iCloseHnd);// now close port`
	`}`
	`...`

2.3.4 FECOM_GetDLLVersion

Function	Gets the version number of the DLL
Syntax	**void FECOM_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 “02.00.01”. Newer versions may provide additional information.
Return value	none

Example	`#include “fecom.h”`
	`...`
	`...`
	`char cVersion[256];`
	`FECOM_GetDLLVersion(cVersion)`
	        // code here for displaying version number
	`...`
	`...`

2.3.5 FECOM_GetErrorText

Function	Gets error text for error code
Syntax	**int FECOM_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. The list of error codes can be found in Appendix A.

Example	`...`
	`#include “fecom.h”`
	`#include “fecomdef.h”`
	`...`
	`...`
	`char cErrorText[256];`
	`...`
	
	`int iBack = FECOM_GetErrorText(FECOM_ERR_EMPTY_LIST, cError-Text)`
	// code here for displaying the text
	`...`
	`...`

2.3.6 FECOM_GetLastError

Function	Gets the last error code and transfers error text.
Syntax	**int FECOM_GetLastError(int iPortHnd, int* iErrorCode, char* cErrorText)**
Description	The function uses *iErrorCode* to transfer the last error code of the port selected by *iPortHnd* and uses *cErrorText* to transfer the associated English-language error text. The buffer for *cErrorText* should to 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 Appendix A.

Example	`...`
	`#include “fecom.h”`
	`...`
	`...`
	`char cErrorText[256];`
	`int iErrorCode = 0;`
	`...`
	
	`int iBack = FECOM_GetLastError(iPortHnd, &iErrorCode, cError-Text)`
	    // code here for displaying the text
	`...`
	`...`

2.3.7 FECOM_GetPortHnd

Function	Gets the port handle of a serial port opened with FECOM.DLL.
Syntax	**int FECOM_GetPortHnd(char* cPortNr)**
Description	As a rule you set the COM port number in a program, whereas internally the program uses a handle. This function can be used to easily get the PortHandle of an already open serial port. This function is an “inverse” of **FECOM_GetPortPara**(iPortHnd, “PortNr”, Value), which gets the number of the COM port for the PortHandle. *cPortNr* is a null-terminated string with the address of the serial port (for example: “1” for COM:1). Values between 1 and 256 are allowed.
Return value	If the serial port for the transmitted *cPortNr* is found, the PortHandle (>0) is returned. If the searched for port number *cPortNr* could not be found in the port list, a 0 is returned. In case of error the function returns a value less than 0. The list of error codes can be found in Appendix A.

Example	`...`
	`#include “fecom.h”`
	`...`
	`char cPortNr[4];`
	`...`
	`itoa(1, cPortNr, 10);// Convert Integer to Char`
	`int handle = FECOM_OpenPort(cPortNr);// COM:1 should be opened`
	`if(handle < 0)`
	`{`
	`        // code here for error condition`
	`}`
	`else`
	`{       // handle is fetched again using PortNr`
	`        handle = FECOM_GetPortHnd(cPortNr);`
	`}`

2.3.8 FECOM_GetPortPara

Function	Gets a parameter from the serial port specified in iPortHnd.
Syntax	**Int FECOM_GetPortPara(int iPortHnd, char* cPara, char* cValue)**
Description	The function gets the current value of a parameter. *cPara* is a null-terminated string with the parameter code. *cValue* is an empty, null-terminated string for returning the parameter value. The string should be able to hold at least 128 characters.
Parameter codes	The parameter codes are: Baud, Frame, Timeout, ErrCode, ErrStr, TxTimeControl, TxDelayTime, CharTimeoutMpy, PortNr. The latter returns the physical number of the serial port.
Return value	If there is no error the function returns the value 0 and in case of error a value less than 0. The list of error codes can be found in Appendix A.
Cross-reference	For additional information refer to Appendix B.
Example	see below

```
..
#include "fecom.h"
...
...
char cValue[128];
...
if(!FECOM_GetPortPara(handle, "Baud", cValue))
     {
                              // code here for displaying the COM parameter
                              ...
     }
...
...
}
```

2.3.9 FECOM_SetPortPara

Function	Sets a serial port parameter to a new value.
Syntax	**int FECOM_SetPortPara(int iPortHnd, char* cPara, char* cValue)**
Description	The function transfers a new parameter to the serial port specified in iPortHnd. This re initializes the serial port in question and deletes the send and receive buffers. *cPara* is a null-terminated string with the parameter code. *cValue* is a null-terminated string with the new parameter value.

Parameter code	**Value range**	**Default value**	**Units**
Baud	300...115200	9600	bit/s
Frame	7N1, 7E1, 7O1, 7N2, 7E2, 7O2, 8N1, 8E1, 8O1	8E1	
Timeout	0...99999	600	ms
TxTimeControl	0, 1	1	
TxDelayTime	0..999	5	ms
CharTimeoutMpy	1...10	1	

Return value	If the serial port was able to be successfully initialized with the new parameter value, a 0 is returned. In case of error the function returns a value less than 0. The list of error codes can be found in Appendix A.
Cross-reference	For additional information refer to Appendix B.
Example	(see below)

```
...
#include “fecom.h”
...
...
int Err;
char cPortNr[4];
...
itoa(1, cPortNr, 10);      // Convert Integer to Char
int handle = FECOM_OpenPort(cPortNr);// COM:1 should be opened
if(handle > 0)
{     Err = FECOM_SetPortPara(handle, “Baud”, “4800”);
      ...
}
...
...
```

2.3.10 FECOM_DoPortCmd

Function	Executes a command on a serial port.
Syntax	**int FECOM_DoPortCmd(int iPortHnd, char* cCmd, char* cValue)**
Description	The function executes a command at the serial port named in *iPortHnd*. *cCmd* is a null-terminated string with the command code. *cValue* is a null-terminated string with the new parameter value or for the return value (for example: status of a control line). If a return value is expected in cValue, the buffer should be able to hold at least 16 characters.

Command	**Function**	**cValue**
FlushInQ	Flushes input buffer	not used
FlushOutQ	Flushes output buffer	not used
SetDTR	Sets DRT line “ON” or “OFF	“”ON” or “OFF”
SetRTS	Sets RTS line “ON” or “OFF”	“ON” or “OFF”
GetDTR	Gets DTR-Status	Status {“ON”, “OFF”}
GetRTS	Gets RTS-Status	Status {“ON”, “OFF”}
GetCTS	Gets CTS-Status	Status {“ON”, “OFF”}
GetDCD	Gets DCD-Status	Status {“ON”, “OFF”}
GetDSR	Gets DSR-Status	Status {“ON”, “OFF”}

Return value	If the command was executed without error, a 0 is returned. In case of error the function returns a value less than 0.

Example 1	`#include “fecom.h”`
	`...`
	`...`
	`int Err;`
	`char cPortNr[4];`
	`...`
	`itoa(1, cPortNr, 10);` // Convert Integer to Char
	`int handle = FECOM_OpenPort(cPortNr);`// COM:1 should be opened
	`if(handle > 0)`
	`{ Err = FECOM_DoPortCmd(handle, “FlushInQ”, “”);`
	`    ...`
	`}`
	`...`

Example 2	`#include “fecom.h”`
	`...`
	`int Err;`
	`char cValue[16];`
	`...`
	`Err = FECOM_DoPortCmd(handle, “GetCTS”, cValue);`
	`if(strcmp(cValue, “ON”)==0)`// Compares strings
	`{`
	    // CTS is set
	`}`
	`...`

2.3.11 FECOM_AddEventHandler

Function	Installs an event handling mechanism
Syntax	**int FECOM_AddEventHandler(int iPortHnd, FECOM_EVENT_INIT* pInit)**

Description	This function installs one of four possible event handling methods. This method is used when the state of the control line for which the method was installed changes. This allows asynchronous response to events in an application program.
	The event handling method is established only for the port identified by iPortHnd. This means that if necessary you may have to repeat this installation for each opened port.
	1st Method: Message to thread (not for Visual Basic)
	This method is used for exchanging messages between Threads[a]. The thread uses the API function GetCurrentThreadID() to get the thread identifier and transfers this as the parameter dwThreadID in the **FECOM_EVEN_INIT** structure.
	The thread must provide a message handling function for receiving the message that was sent by FECOM with the API function PostThreadMessage(..). The message code is freely selectable.
	The **FECOM_EVENT_INIT** structure is filled as follows:
	`uiUse = FECOM_xyz_EVENT`// see Defines FECOM.H
	`uiMsg = WM_USER + ...`// freely selectable, but higher than WM_USER [b]
	`uiFlag = FECOM_THREAD_ID`
	`dwThreadID = GetCurrentThreadID()`
	The MessageMap function in the application is given in the 1st parameter (WPARAM) the port number and in the 2nd parameter (LPARAM) the status of the control line (0 = not set; 1 = set).
	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 API function GetWindow(..)[c] to get the handle and transfer it as the parameter hwndWnd in the **FECOM_EVENT_INIT** structure. The window must provide a message handling function for receiving the message that was sent by FECOM with the API function PostThreadMessage(..). The message code is freely selectable.
	The **FECOM_EVENT_INIT** structure is filled as follows:
	`uiUse = FECOM_xyz_EVENT`// see Defines FECOM.H
	`uiMsg = WM_USER + ... //` freely selectable, but higher than WM_USER2 [b]
	`uiFlag = FECOM_WND_HWND`
	`hwndWnd = GetWindow(...)`
	The MessageMap function in the application is given in the 1st parameter (WPARAM) the port number and in the 2nd parameter (LPARAM) the status of the control line (0 = not set; 1 = set).

Description	<u>3rd method: Invoking a call back function</u>
	In the call back method a function pointer for an event is installed. When the status of an appropriate control line changes, FECOM invokes the function. The content of the function can be freely determined. The transfer parameters are however specified: In the first parameter the port number is transferred and in the 2nd parameter the status of the control line (0 = not set, 1 = set).
	Die **FECOM_EVENT_INIT** structure is filled as follows:
	`uiUse = FECOM_xyz_EVENT`// see Defines FECOM.H
	`uiMsg not needed`
	`uiFlag = FECOM_EVENT`
	`cbFct = (void*)&YourFunctionName`[d]
	<u>4th method: Setting an event</u>
	For the event method an event handle is installed for an event. When the state of an affected control line changes, the event is set by FECOM using the API-Function SetEvent(…). On the application side you wait for the event with the API-Function WaitForSingleObject(…). Since you cannot distinguish how the state of the affected control line changed, you must use the function **FECOM_DoPortCmd** to query the state. The set event must be reset again by the application program using the API-Function ResetEvent(…).
	The **FECOM_EVENT_INIT** structure is filled in as follows:
	`uiUse = FECOM_xyz_EVENT`// see Defines FECOM.H
	`uiMsg is not needed`
	`uiFlag = FECOM_EVENT`
	`hEvent = CreateEvent(...);`
	An installed event handling method is deleted using the function **FECOM_DelEventHandler**.
	When closing a port, all the event handling methods stored for this port are lost.

Cross-reference	For additional information see: sections 2.3, 2.3.12 and Appendix B.
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 Appendix A.

a. Parallel execution path independent of the application program. The application program itself is a thread.
b. See Windows documentation for SDK platform.
c. When using MFC CWind you can also use the GetSafeHwnd() method.
d. The function has the prototype: void YourFunctionName(int, int).

2.3.12 FECOM_DelEventHandler

Function	Deletes an event handling mechanism
Syntax	**int FECOM_DelEventHandler(int iPortHnd, FECOM_EVENT_INIT* pInit)**

Description	The function deletes an event handling mechanism which was previously installed using **FECOM_AddEventHandler**. The **FECOM_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 **FECOM_EVENT_INIT** structure is filled as follows:
	`uiUse = FECOM_xyz_EVENT`// see Defines in FECOM.H
	`uiMsg not needed`
	`uiFlag = FECOM_THREAD_ID`
	`dwThreadID = GetCurrentThreadID()`
	2nd method of deleting: Message to Window (not for Visual Basic)
	The **FECOM_EVENT_INIT** structure is filled as follows:
	`uiUse = FECOM_xyz_EVENT`// see Defines in FECOM.H
	`uiMsg not needed`
	`uiFlag = FECOM_WND_HWND`
	`hwndWnd = GetWindow(...)`
	3rd method of deleting: Invoking a call back function
	The **FECOM_EVENT_INIT** structure is filled as follows:
	`uiUse = FECOM_xyz_EVENT`// see Defines FECOM.H
	`uiMsg not needed`
	`uiFlag = FECOM_CALLBACK`
	`cbFct = (void*)&YourFunctionName`
	4th method of deleting: Setting an event
	The **FECOM_EVENT_INIT** structure is filled as follows:
	`uiUse = FECOM_xyz_EVENT`// see Defines FECOM.H
	`uiMsg not needed`
	`uiFlag = FECOM_EVENT`
	`hEvent = IhrEventHandle;`

Cross-reference	For additional information see: sections 2.3, 2.3.11 and Appendix B.
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 Appendix A.

2.3.13 FECOM_Transceive

Function	Function for communication (Transmit and Receive) through the port.
Syntax	**int FECOM_Transceive(int iPortHnd, UCHAR* cSendProt, int iSendLen, UCHAR* cRecProt, int iRecLen)**
Description	This function sends the data contained in *cSendProt* through the serial port to an attached device and stores the received data in *cRecProt*. The number of characters in *cSendProt* must be transferred in the *iSendLen* parameter. The *iRecLen* parameter must be used to indicate the maximum length of the *cRecProt* buffer. If the number of characters received exceeds the value transferred in *iRecLen*, the function is ended with an error. The characters received up to the point of the cancel are stored in *cRecProt*. Prior to communication the transmit and receive buffers are deleted. The parameter TxDelayTime can be used to delay the send protocol until the time TxDelayTime has elapsed since the last receive protocol.
Return value	If there are no errors, the function returns the length of the receive protocol, and in case of error it returns a value less than 0. The list of error codes can be found in Appendix A.

Example

```
#include “fecom.h”
...
int iSendLen;
int iRecProtLen;
char cPortNr[4];
...
itoa(1, cPortNr, 10);      // Convert Integer to Char
UCHAR cSendBuf[256];       // Adjust buffer size to transmit data if needed
UCHAR cRecBuf[256];        // Adjust buffer size to receive data if needed
...
int handle = FECOM_OpenPort(cPortNr);// COM:1 should be opened
if(handle < 0)
{
     // code here for error condition
}
else
{
     // the transmit protocol is fetched, for example with a function and stored in
       SendBuf
     iSendLen = GetSendProtocol(cSendBuf);
     // Communication through COM:1, if successful, the receive data are located
     in RecBuf
     iRecProtLen = FECOM_Transceive(handle, cSendBuf, iSend-
Len, cRecBuf, 256);
     if(cRecProtLen < 0)
     {
                           // Communication error
     }
}
```

2.3.14 FECOM_Transmit

Function	Function for sending a protocol through the port.
Syntax	**int FECOM_Transmit(int iPortHnd, UCHAR* cSendProt, int iSendLen)**
Description	The function sends the data contained in *cSendProt* through the serial port to an attached device and does <u>not</u> wait for a reply protocol. The number of characters in *cSendProt* must be indicated in the iSendLen parameter. Before the protocol is sent the transmit buffer is deleted. Any characters which are still waiting for the output are lost. The function does not revert until all the characters have been output through the port.
Return value	In case of error the Function returns 0, or in case of error a value less than 0. The list of error codes can be found in Appendix A.
Example	see below

```
...
#include “fecom.h”
...
...
int iErr;
int iSendLen;
char cPortNr[4];
...
itoa(1, cPortNr, 10);       // Convert Integer to Char
UCHAR cSendBuf[256];        // Buffer size may need to be adjusted to
                            the send data
...
int handle = FECOM_OpenPort(cPortNr);//  COM:1 should be
                                         opened
if(handle < 0)
{
     // code here for error condition
}
else
{    // the transmit protocol is fetched for example with a function and
     stored in SendBuf
     iSendLen = GetSendProtocol(cSendBuf);
     // Communication through COM:1
     iErr = FECOM_Transmit(handle, cSendBuf, iSendLen);
     if(iErr < 0)
     {
                            // Communication error
     }
}
```

2.3.15 FECOM_Receive

Function	Function for receiving a protocol through the port.
Syntax	**Syntaxint FECOM_Receive(int iPortHnd, UCHAR* cRecProt, int iRecLen)**
Description	This function expects data received through the serial port within the Timeout time (see Appendix B), reads them out and stores them in the receive buffer *cRecProt*. The *iRecLen* parameter must be used to indicate the maximum length of the *cRecProt* buffer. If the number of characters received exceeds the value transferred in *iRecLen*, the function is ended with an error. The characters received up to the point of the cancel are stored in *cRecProt*. The function does <u>not</u> delete the receive buffer. This ensures that characters which arrived previously are not lost.
Return value	If there is not error the function returns the length of the receive protocol, or in case of error a value less than 0. The list of error codes can be found in Appendix A.
Example	See code below

```
..
#include "fecom.h"
#include "fecomdef.h"
...
...
int iRecProtLen;
char cPortNr[4];
...
itoa(1, cPortNr, 10);        // Convert Integer to Char
UCHAR cRecBuf[256];          // Buffer size may need to be adjusted to
                             the receive data
...
int handle = FECOM_OpenPort(cPortNr);// COM:1 should be
                                        opened
if(handle < 0)
{
     // code here for error condition
}
else
{    // Communication through COM:1, if successful the receive data will
     be located in RecBuf
     iRecProtLen = FECOM_Receive(handle, cRecBuf, 256);
     if(iRecProtLen < 0)
     {
               // Communication error or buffer overflow
     if(iRecProtLen == FECOM_ERR_OVL_RECBUF)
                             {// Buffer overflow: Data in RecBuf are
                             valid receive data
                             ...
                             }
     }
}
```

Error Codes

Error constants	Value	Description
FECOM_ERR_NEWPORT_FAILURE	-1000	Error in generating a new port object. Lack of memory may be the cause.
FECOM_ERR_EMPTY_LIST	-1001	Port handle list is empty (no port objects stored)
FECOM_ERR_POINTER_IS_NULL	-1002	A pointer is null, thus invalid
FECOM_ERR_NO_MEMORY	-1003	Lack of memory
FECOM_ERR_UNSUPPORTED_HARDWARE	-1004	Unsupported hardware. The error is reported whenever the hardware used does not support a counter with high resolution
FECOM_ERR_NO_PORT	-1010	Port could not be opened
FECOM_ERR_NO_CONNECT	-1011	Timeout when opening the port. Port was not opened
FECOM_ERR_LINK_ID	-1012	The parameter cPortNr in the FECOM OpenPort function is defective
FECOM_ERR_PORT_IS_OPEN	-1013	The port is already open
FECOM_ERR_UNKNOWN_HND	-1013	The transferred port handle is unknown
FECOM_ERR_HND_IS_NUL	-1021	The transferred port handle is 0
FECOM_ERR_HND_IS_NEGATIVE	-1022	The transferred port handle is negative
FECOM_ERR_NO_HND_FOUND	-1023	No port handle found in the port handle list
FECOM_ERR_TIMEOUT	-1030	Timeout when reading from the port
FECOM_ERR_NO_SENDPROTOCOL	-1031	No send protocol transferred
FECOM_ERR_RECEIVE_PROCESS	-1032	Error in receive process
FECOM_ERR_INIT_COMM_PROCESS	-1033	Error in initializing the port
FECOM_ERR_FLUSH_INPUT_BUFFER	-1034	Error in flushing the input buffer
FECOM_ERR_FLUSH_OUTPUT_BUFFER	-1035	Error in flushing the output buffer
FECOM_ERR_CHANGE_PORT_PARA	-1036	Error in changing a port parameter
FECOM_ERR_TRANSMIT_PROCESS	-1037	Error in the transmit process
FECOM_ERR_UNKNOWN_PARAMETER	-1050	Transfer parameter unknown
FECOM_ERR_PARAMETER_OUT_OF_RANGE	-1051	Transfer parameter too large or too small

Error constants	Value	Description
FECOM_ERR_ODD_PARAMETERSTRING	-1052	An unsupported option was invoked by a transfer parameter
FECOM_ERR_PORTNR_OUT_OF_RANGE	-1053	The transferred port number is not within the allowed range of 1 to 256
FECOM_ERR_UNKNOWN_ERRORCODE	-1054	Unknown error code
FECOM_ERR_OVL_RECBUF	-1070	Receive buffer overflow

List of Parameter Codes

Parameter code	Value range	Default	Units	Description
Baud	300...115200	9600	bit/s	Baud rate for the port
Frame	7N1, 7E1, 7O1, 7N2, 7E2, 7O2, 8N1, 8E1, 8O1	8E1		Character frame (data bits, parity, stop bits)
Timeout	0...99999	600	ms	Maximum wait time for receive protocol
PortNr	1...256	0		Number of the COM port
TxTimeControl	0, 1	1		When set (1), there is an internal delay before the next send protocol is sent at least until TxDelayTime (mx) has elapsed since the last receive protocol. If not set (0), the send protocol is always output as soon as possible.
TxDelayTime	0...999	5	ms	Minimum time span between the last receive and the next send protocol. Only applies if TxTimeControl=1
CharTimeoutMpy	1...10	1		The character timeout is calculated internally. The character timeout specifies after how much time after receipt of the last character the receive process is ended. With some PCs there may be repeated protocol length errors because the wait time is too short. In this case you may use this parameter to multiply the wait time.

APPENDIX C

FECOM_EVENT_INIT Constants

The constant definitions are contained in the file FECOM.H, FECOM.BAS or FE-COM.PAS.

Constant	Value	Use	Description
FECOM_THREAD_ID	1	uiFlag	Event flag with thread message
FECOM_WND_HWND	2	uiFlag	Event flag with window message
FECOM_CALLBACK	3	uiFlag	Event flag with call back function
FECOM_EVENT	4	uiFlag	Event flag with Windows API event
FECOM_CTS_EVENT	1	uiUse	Flag for CTS change
FECOM_DCD_EVENT	2	uiUse	Flag for DCD change
FECOM_DSR_EVENT	3	uiUse	Flag for DSR change
FECOM_RTS_EVENT	4	uiUse	Flag for RTS change
FECOM_DTR_EVENT	5	uiUse	Flag for DTR change