MANUAL

ID FECOM

Version 2.08.04

Software-Support for the

Serial Interface

For 32-Bit Operating Systems

Windows 2000/XP/Vista

and Windows CE

and Linux

final
public (B)
2008-12-16
H80592-20e-ID-B.doc

Note

 FEIG ELECTRONIC GmbH
Lange Straße 4
D-35781 Weilburg-Waldhausen
Germany
eMail: obid@feig.de

This manual supercedes all previous editions.
The information contained in this manual is subject to change without notice.

The information contained in this manual has been gathered with all due care and to the best of our knowledge. FEIG ELECTRONIC GmbH assumes no liability for the accuracy and completeness of the data in this manual. In particular, FEIG ELECTRONIC GmbH cannot be held liable for consequential damages resulting from inaccurate or incomplete information. Since even with our best efforts this document may still contain mistakes, please contact us should you find any errors.

Licensing agreement for use of the software

This is an agreement between you and FEIG ELECTRONIC GmbH (hereafter "FEIG") for use of the ID 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 FEIG ELECTRONIC GmbH and is protected by international copyright.

§1 Object and scope of the agreement

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

§2. Protection of the licensing material

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

§3 Warranty and liability limitations

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

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

§4 Concluding provisions

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

Contents:

1. Introduction

The ID FECOM support package is intended to assist in programming communications-oriented software and supports the languages ANSI-C, ANSI-C++ as well as any other language which can invoke C functions.

The support package provides a simple function interface for the serial interface of a PC running under 32-Bit Windows 2000/XP/Vista and Windows CE and 32-Bit Linux, and has been especially designed for use together with other support packages (e.g. ID FERW, ID FERWA, ID FEISC).

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

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

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

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

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

File	Use
LIBFECOM.SO.x.y.z[1]	Function library
FECOM.H	Header file for C/C++ projects

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

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

2. Installation

2.1. 32-Bit Windows 2000/XP/Vista

Installation must be performed manually:

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

2.2. Windows CE

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

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

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

2.3. 32-Bit Linux

Installation is quite simple:

- copy the files described below to the corresponding directories.
- create a symbolic link to the library file libfecom.so.x.y.z[1] in the directory /usr/lib:

```
cd /usr/lib
ln –s /<Directory>/libfecom.so.x.y.z   libfecom.so.x
ln –s /<Directory>/libfecom.so.x   libfecom.so
ldconfig
```

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

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

3. Incorporating into the application program

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

4. Changes since the previous version

- Windows: Improvements for remote access with Windows Server
- Linux:

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

5. Programming Interface

5.1. Overview

The FECOM library encapsulates all the functions and parameters which the user needs in order to manage one or more serial ports open at the same time. The object-oriented internal structure (see Fig. 1) is intentionally configured as a function interface to the outside world. This gives it the advantage of being language-neutral.

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

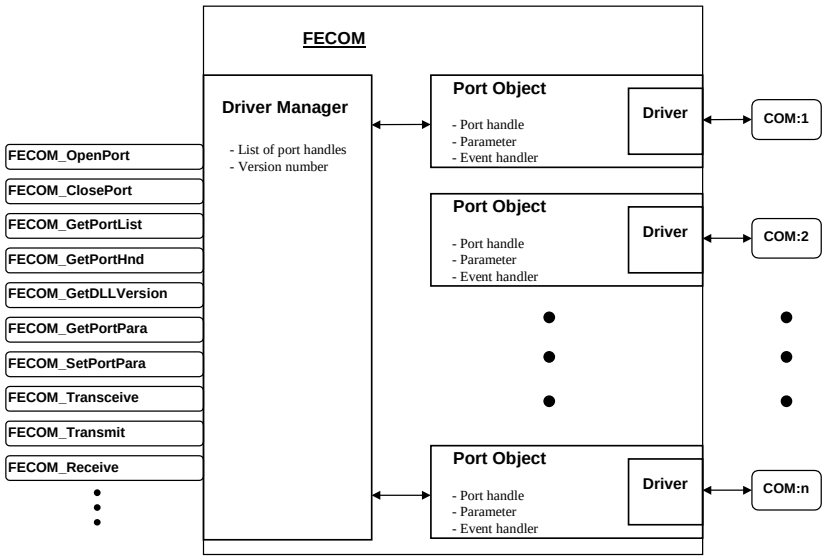

Fig. 1: Internal structure of FECOM

Before you can first communicate, a port object must be created. This is automatically done by the **FECOM_OpenPort** function. If this function was executed 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) do not however have to be saved in the application program, since the library driver manager maintains a list internally of all opened COM Ports. This list can be called with the **FECOM_GetPortList** function. The handles that one thereby receives successively can be used with the **FEDCOM_GetPortPara** function to read out all the parameters pertaining to this 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 multiple 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. Of course a COM port can only be opened once, since it is physically present only once.

Every library function (exception: **FECOM_GetDLLVersion**) has a return value which in case of error is always negative.

5.2. 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_DetectPort(** int iPortNr **)**
- **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 **)**

5.3. Event flagging for control lines

Event handling mechanisms can be installed individually for each control lines 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 signaling is generated. This is a way to notify an application of the event asynchronous to 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 callback function.

An 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 (e.g. FECOM_CTS_EVENT)
    UINT uiMsg;      // Message code for dwThreadID and hwndWnd (e.g. WM_USER_xyz)
    UINT uiFlag;     // Specifies use of the union (e.g. FECOM_WND_HWND)
    union

    {
        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" of the union is only for C programmers. C++ programmers access the union directly through the structure.

5.3.1. FECOM_OpenPort

Function	Opens a serial port for communicating with an OBID 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 (e.g. "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 the Appendix.
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 }

5.3.2. 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 the Appendix.
Example	... #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); ... } ... ...

5.3.3. FECOM_DetectPort

Function	Checks whether a serial port is physically present.
Syntax	**int FECOM_DetectPort(int iPortNr)**
Description	The function checks the serial port having the number iPortNr to see whether it is physically present. If the port is found, a 0 is returned, otherwise FECOM_ERR_PORT_NOT_FOUND. This function is ideal for application programs that can offer the user a list of the possible serial ports.
Return value	If the port is found, a 0 is returned, otherwise FECOM_ERR_PORT_NOT_FOUND. In case of error, the function returns a value less than zero. The error code list can be found in the Appendix.
Example	... #include "fecom.h" ... for(int iPortNr=1; iPortNr<257, ++iPortNr) { if(0 == FECOM_DetectPort(iPortNr); { // Port is physically present } }

5.3.4. 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 one enters a 0 for *iNext*, the first entry in the list is returned. If one enters a PortHandle contained in the list for *iNext*, the entry following that PortHandle is gotten 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.
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 e.g. code for reading out the COM parameters using FECOM_GetPortPara(...) ... iNextHnd = FECOM_GetPortList(iNextHnd); // get next handle } ... // here e.g. 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 } ...

5.3.5. FECOM_GetDLLVersion

Function	Gets the version number of the DLL/SO
Syntax	**void FECOM_GetDLLVersion(char* cVersion)**
Description	The function returns the version number of the DLL/SO. *cVersion* is an empty, null-terminated string for returning the version number. The string should be able to hold at least 256 characters. In the current version the string is filled with „02.08.03“. Newer versions may provide additional information.
Return value	without
Example	... #include "fecom.h" ... ... char cVersion[256]; FECOM_GetDLLVersion(cVersion) // code here for displaying version number ... ...

5.3.6. 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 the Appendix.
Example	... #include "fecom.h" #include "fecomdef.h" ... ... char cErrorText[256]; ... int iBack = FECOM_GetErrorText(FECOM_ERR_EMPTY_LIST, cErrorText) // code here for displaying the text ... ...

5.3.7. 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 the Appendix.
Example	... #include "fecom.h" ... ... char cErrorText[256]; int iErrorCode = 0; ... int iBack = FECOM_GetLastError(iPortHnd, &iErrorCode, cErrorText) // code here for displaying the text ... ...

5.3.8. 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 (e.g. "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 the Appendix.
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 gotten again using PortNr handle = FECOM_GetPortHnd(cPortNr); }

5.3.9. 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, Performance, Language and PortNr. The latter returns the physical number of the serial port. The parameter Language sets the language inside the DLL and is a global parameter. This means, that iPortHnd is insignificant and should be set to 0.
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 the Appendix.
Cross-reference	For additional information see: 6.2. List of Parameter Codes.
Example	... #include "fecom.h" ... ... char cValue[128]; ... if(!FECOM_GetPortPara(handle, "Baud", cValue)) { // code here for displaying the COM parameter ... } ... ... }

5.3.10. FECOM_SetPortPara

<table>
<tr><td>Function</td><td>Sets a serial port parameter to a new value.</td></tr>
<tr><td>Syntax</td><td>int FECOM_SetPortPara(int iPortHnd, char* cPara, char* cValue)</td></tr>
<tr><td>Description</td><td>The function transfers a new parameter to the serial port specified in iPortHnd. This reinitializes 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.
<table>
<tr><th>Parameter code</th><th>Value range</th><th>Default value</th><th>Units</th></tr>
<tr><td>Baud</td><td>300...115200</td><td>9600</td><td>bit/s</td></tr>
<tr><td>Frame</td><td>7N1, 7E1, 7O1,
7N2, 7E2, 7O2,
8N1, 8E1, 8O1</td><td>8E1</td><td></td></tr>
<tr><td>Timeout</td><td>0...99999</td><td>600</td><td>ms</td></tr>
<tr><td>TxTimeControl</td><td>0, 1</td><td>1</td><td>-</td></tr>
<tr><td>TxDelayTime</td><td>0...999</td><td>5</td><td>ms</td></tr>
<tr><td>CharTimeoutMpy</td><td>1...99</td><td>1</td><td>-</td></tr>
<tr><td>Performance</td><td>0, 1</td><td>1</td><td>-</td></tr>
<tr><td>Language</td><td>7, 9</td><td>9</td><td>-</td></tr>
<tr><td>UseOBID (only Linux)</td><td>0, 1</td><td>0</td><td>-</td></tr>
</table></td></tr>
<tr><td>Return value</td><td>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 the Appendix.</td></tr>
<tr><td>Cross-reference</td><td>For additional information see: 6.2. List of Parameter Codes.</td></tr>
<tr><td>Example</td><td>…
#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”);
  …
}
…
…</td></tr>
</table>

5.3.11. FECOM_DoPortCmd

<table>
<tr><td>Function</td><td>Executes a command on a serial port.</td></tr>
<tr><td>Syntax</td><td>int FECOM_DoPortCmd(int iPortHnd, char* cCmd, char* cValue)</td></tr>
<tr><td>Description</td><td>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 (e.g., status of a control line).

If a return value is expected in cValue, the buffer should be able to hold at least 16 characters.
<table>
<tr><th>Command</th><th>Function</th><th>cValue</th></tr>
<tr><td>FlushInQ</td><td>Flushes input buffer</td><td>not used</td></tr>
<tr><td>FlushOutQ</td><td>Flushes output buffer</td><td>not used</td></tr>
<tr><td>SetDTR</td><td>Sets DRT line „ON“ or „OFF“</td><td>„ON“ or „OFF“</td></tr>
<tr><td>SetRTS</td><td>Sets RTS line „ON“ or „OFF“</td><td>„ON“ or „OFF“</td></tr>
<tr><td>GetDTR</td><td>Gets DTR-Status</td><td>Status {„ON“, „OFF“}</td></tr>
<tr><td>GetRTS</td><td>Gets RTS-Status</td><td>Status {„ON“, „OFF“}</td></tr>
<tr><td>GetCTS</td><td>Gets CTS-Status</td><td>Status {„ON“, „OFF“}</td></tr>
<tr><td>GetDCD</td><td>Gets DCD-Status</td><td>Status {„ON“, „OFF“}</td></tr>
<tr><td>GetDSR</td><td>Gets DSR-Status</td><td>Status {„ON“, „OFF“}</td></tr>
</table></td></tr>
<tr><td>Return value</td><td>If the command was executed without error, a 0 is returned. In case of error the function returns a value less than 0.</td></tr>
<tr><td>1. Example</td><td>

```
#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”, “” );
      …
}
…
```

</td></tr>
<tr><td>2. Example</td><td>

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

</td></tr>
</table>

5.3.12. FECOM_AddEventHandler

Function	Installs an event handling mechanism
Syntax	**int FECOM_AddEventHandler(int iPortHnd, FECOM_EVENT_INIT* pInit)**
Description	The 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 Linux) This method is used for exchanging messages between Threads[1]. 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[2] 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 Linux) This method is used when the message needs to be sent directly to a window. The corresponding window uses the API function GetWindow(..)[3] 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_USER[2] 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).

[1] Parallel execution path independent of the application program. The application program itself is a thread.
[2] See Windows documentation for the SDK platform
[3] When using MFC CWnd you can also use the GetSafeHwnd() method.

	3rd method: Invoking a callback function In the callback 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[1] 4th method: Setting an event (not for Linux) 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: 5.3. Event flagging for control lines, 5.3.13. FECOM_DelEventHandler, 6.3. List of constants for the FECOM_EVENT_INIT structure.
Return value	If there is no error the function returns zero, and if error a value less than zero. The list of error codes can be found in the Appendix.

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

5.3.13. 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. Deleting the 1st method: Message to Thread (not for Linux) 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() Deleting the 2nd method: Message to Window (not for Linux) Die **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(…) Deleting the 3rd method: Invoking a callback 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 Deleting the 4th method: Setting an event (not for Linux) 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: 5.3. Event flagging for control lines, 5.3.12. FECOM_AddEventHandler, 6.3. List of constants for the FECOM_EVENT_INIT structure.
Return value	If there is no error the function returns zero, and if error a value less than zero. The list of error codes can be found in the Appendix.

5.3.14. 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	The 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[1] 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 the Appendix.
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 gotten 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, iSendLen, cRecBuf, 256); if(cRecProtLen < 0) { // Communication error } }

[1] See Section 5.3.10. FECOM_SetPortPara

5.3.15. 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 not 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 the Appendix.
Example	… #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 gotten 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 } }

5.3.16. FECOM_Receive

Function	Function for receiving a protocol through the port.
Syntax	**int FECOM_Receive(int iPortHnd, UCHAR* cRecProt, int iRecLen)**
Description	The function expects data received through the serial port within the Timeout time (see 6.2. List of Parameter Codes), 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 not 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 the Appendix.
Example	… #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 … } } }

6. Appendix

6.1. 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_PORT_NOT_FOUND	-1005	Return value of FECOM_DetectPort, if the transferred port is not existent.
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	-1020	The transferred port handle is unknown
FECOM_ERR_HND_IS_NULL	-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_RECEIVE_NOISE_DATA	-1038	Checksum error and/or parity error or not identifiable data stream
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

6.2. 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, 7°1, 7N2, 7E2, 7°2, 8N1, 8E1, 8°1	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…99	1	-	Since Version 2.00.00 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.
Performance	0, 1	1	-	Determines the performance of the internal communication process. In normal case, Performance is set to 1. For maximum of communication power, Performance is set to 0. But in this case, other threads and applications will be delayed.
Language	7 - german 9 - english	9	-	Sets the language for text resources
UseOBID	0, 1	0	-	**only for Linux**: activates internally a specialized receive algorithm adopted to OBID protocol frames to increase the communication performance
Instead of the following parameter codes the function **FECOM_GetLastError** should be used				
ErrCode	-	-	-	Returns the last error code
ErrStr	-	-	-	Returns text for the last error

Obsolete parameters				
CharTimeout	0…99999	20	ms	Maximum wait time for next character in the receive protocol
SleepTime	0...999	0	ms	Wait time after the send protocol and before reading the receive protocol[1]
PortOpenTimeout	0...99999	5000	ms	Maximum wait time for opening a COM port

[1] See 5.3.14. FECOM_Transceive

6.3. List of constants for the FECOM_EVENT_INIT structure

The constant definitions are contained in the file FECOM.H, FECOM.BAS or FECOM.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 callback 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

6.4. Revision history

V2.08.02

- Windows: Improved thread security while opening and closing of ports.
- Linux: Improved communication performance when using of specialized receive algorithm adopted to OBID protocol frames by enabling with the parameter UseOBID.
- Increase of the upper range of the parameter CharTimeoutMpy to 99.

V2.08.00

- The Linux library is compiled with GCC 3.3.3 under SuSE Linux 9.1

V2.07.00

- New baudrates 230400 and 460800.

V2.06.08

- Fixing of small bugs.

V2.06.06

- First Linux Release (SuSE Linux 8.2, GNU Compiler Collection V23.3-23, glibc V2.3.2-6)

V2.06.00

- Integration of a Java interface for the OBID® Java library
- new error code: -1006

V2.05.00

- The new version is 100% downward compatible with the previous version.
- First Windows CE Version

V2.04.04

- The new version is 100% downward compatible with the previous version.
- Move of all constants from the file fecomdef.h to the file fecom.h. The file fecomdef.h is now dispensable.
- The function **FECOM_GetPortHnd** returns for port number 256 a port handle.

V2.04.00

- The new version is 100% downward compatible with the previous version.
- New function: **FECOM_DetectPort**.

V2.03.00

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

V2.02.00

- The new version is 100% downward compatible with the previous version.
- New parameters: Performance and Language
- The problem with the port number 256 is removed

V2.00.01

- The new version is 100% backward compatible with the previous version.
- New parameter CharTimeoutMpy

V2.00.00

- The new version is 100% backward compatible with the previous version.
- FECOM is no longer dependent on other files.
- The performance of the receive routine in the DLL was significantly increased: The functions FECOM_Receive and FECOM_Transceive now revert with a minimum delay after receipt of the protocol.
- New control parameters: TxTimeControl and TxDelayTime for targeted delay of transmission protocols.
- The parameters CharTimeout, PortOpenTimeout and SleepTime are obsolete.
- It is now possible to open ports having a number greater than 9.
- Event handling for events has been expanded.
- FECOM can by dynamically linked under C/C++. The function declarations are found in the header file FECOM.H.
- Renaming of the error codes:
  FECOM_ERR_READ_PROTOCOL in FECOM_ERR_RECEIVE_PROCESS

V1.01.00

- The function **FECOM_DoPortCmd** permits setting and querying of the following control lines:

Control line	Set	Query status
DTR	X	X
RTS	X	X
CTS	-	X
DCD	-	X
DSR	-	X

- In addition you can link the status change of a control line with a flag. For more detailed information, see Section 5.3. Event flagging for control lines.
- The function **FECOM_GetLastError** returns the last error code and an error text.
- The function **FECOM_GetErrorText** returns the error text for any desired error code.

V1.00.09

- New parameters for **FECOM_GetPortPara**: ERRCODE, ERRSTR

Versions 1.00.07 and 1.00.08 were internal preliminary versions

V1.00.06

- Version 1.00.06 was modified internally only with respect to better performance and stability.
- Version 1.00.06 is in all functions invoke-compatible with the previous versions 1.00.02 – 1.00.05
- Version 1.00.06 is no longer invoke-compatible with Version 1.00.01 in the following functions:

  1. FECOM_OpenPort
  2. FECOM_GetPortHnd

  Both functions now expect a pointer to a string, whereas the previous version expects a byte value!