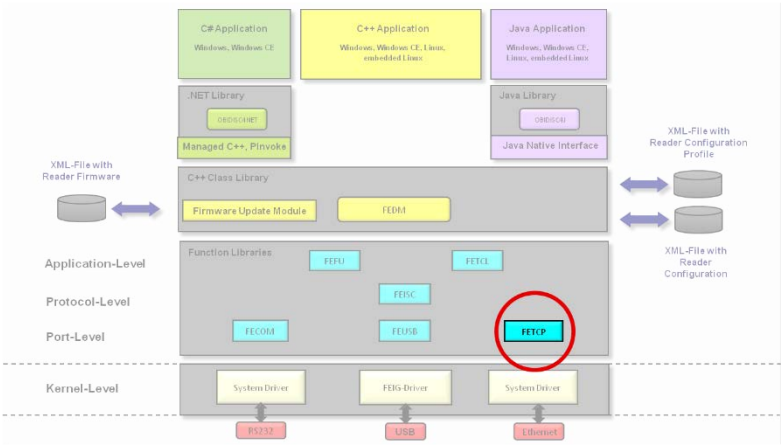
IDENTIFICATION

MANUAL

ID FETCP

Software Support for TCP/IP Interface

Version 3.00.00



Operating System	Target		Notes
	32-Bit	64-Bit	
Windows Vista / 7 / 8 / 10	X	X	
Windows CE 6	X	-	On request
Linux	X	X	
Android	X	X	On request
Apple Max OS X	-	X	On request

Note

FEIG ELECTRONIC GmbH
Lange Strasse 4
D-35781 Weilburg (Germany)
Tel.: +49 6471 3109-0
http://www.feig.de
identification-support@feig.de

With the edition of this manual, all previous editions become void. Indications made in this manual may be changed without previous notice.

Composition of the information in this manual has been done to the best of our knowledge.
FEIG ELECTRONIC GmbH does not guarantee the correctness and completeness of the details given in this manual and may not be held liable for damages ensuing from incorrect or incomplete information. Since, despite all our efforts, errors may not be completely avoided, we are always grateful for your useful tips.

The installation instructions given in this manual are based on advantageous boundary conditions.
FEIG ELECTRONIC GmbH does not give any guarantee promise for perfect function of an Identification system in cross environments.

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

All cited brand names, product names, or trademarks belong to their respective holders.

Licensing Agreement for Use of the Software

This is an agreement between you and FEIG ELECTRONIC GmbH (hereafter "FEIG") for use of the ID FETCP 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 FETCP.DLL, FETCPCE.DLL, , LIBFETCP.x.y.z.DYLIB[1] or LIBFETCP.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 it 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 calls explicit attention to the fact that 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 human health.
   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 current 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.

Content

1. Introduction

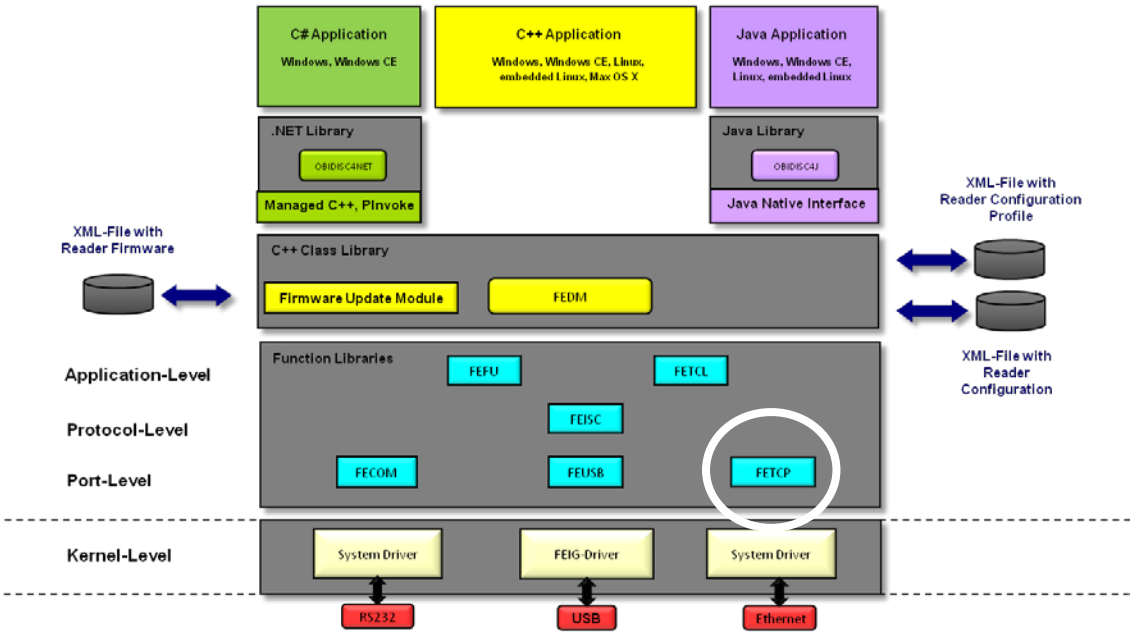
The support package ID FETCP serves as an aid in programming communication-oriented software and supports the languages ANSI-C, ANSI-C++ and in principle any other language that can invoke C functions.

The support package offers a simple function interface for the Socket API of the supported operating systems and has been specially developed for use together with other support packages (e.g., ID FEISC).

This library package can be used with the following operating systems:

Operating System	Target		Notes
	32-Bit	64-Bit	
Windows XP	X	(X)	with 64-Bit OS: only with 32-Bit Runtime Environment
Windows Vista / 7 / 8	X	X	
Windows CE	X	-	
Linux	X	X	
Apple Max OS X	-	X	OS X V10.7.3 or higher Architecture x86_64

The library FETCP is part of the first level of a hierarchical structured, multi-tier FEIG library stack. It is only designed for the data exchange between the TCP/IP driver of an operating system with an application. The following picture shows the multi-tier library stack.



Applications, based on the layer of FETCP have to implement the protocol handling (building/splitting of protocol frames, CRC check, check of protocol frame). Thus, the implementation complexity is extensive and every programmer should calculate the costs.

If the project forces to use only function libraries, the library FEISC from the next level should be chosen as the best API.

1.1. Shipment

This support package consists of files listed in the tables below. Normally, this package is shipped together with other libraries in a Software Development Kit (SDK) – e.g. ID ISC.SDK.Win.

1.1.1. Windows Vista / 7 / 8 / 10

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

1.1.2. Windows CE 6

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

1.1.3. Linux

File	Use
LIBFETCP.SO.x.y.z[2]	Function library
FETCP.H	Header file for C/C++ projects

1.1.4. Mac OS X

File	Use
LIBFETCP.x.y.z.dylib[2]	Function library
FETCP.H	Header file for C/C++ projects

[2] *x.y.z represents the version number*

2. Changes Since the Previous Version

- CharTimeout increased from 25ms to 250ms
-
- Discontinued support for Windows XP, Windows CE 4 and CE 5

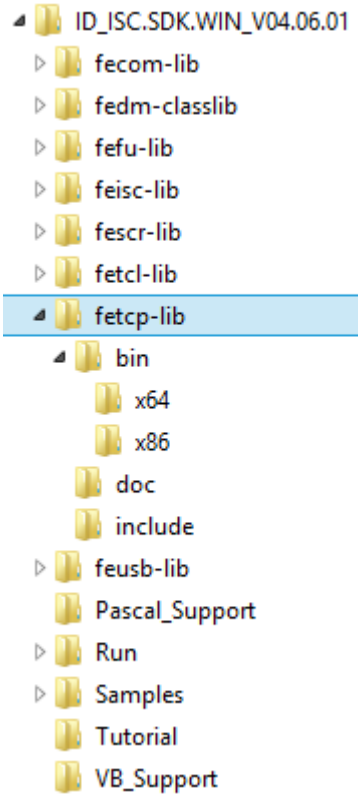
Please also note the revision history in the appendix.

3. Installation

Normally, this package is shipped together with other libraries in a Software Development Kit (SDK). Copy the SDK into a directory of your choice.

The files of this library package can be found in the sub-directory fetcp-lib.

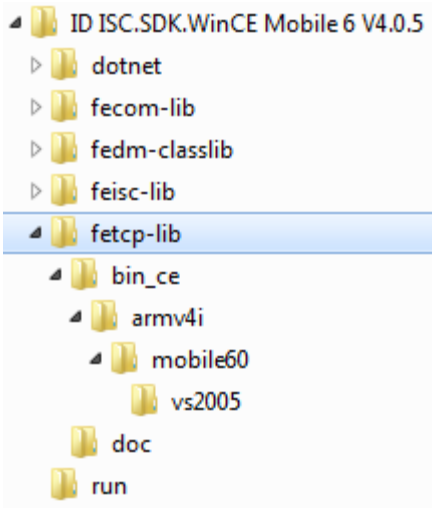
3.1. 32- and 64-Bit Windows Vista / 7 / 8 / 10



If you won't add your projects to the Samples path, we recommend the following steps:

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

3.2. Windows CE 6

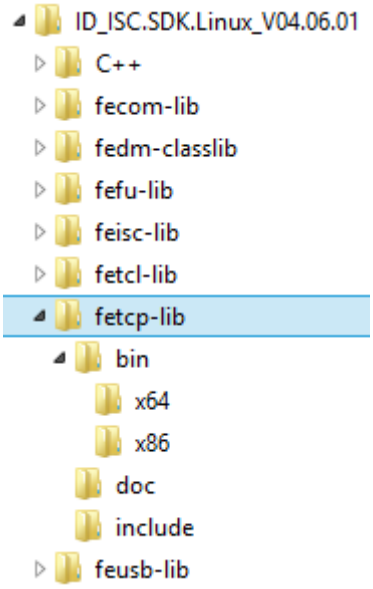


If you won't add your projects to the Samples path, we recommend the following steps:

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

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

3.3. 32- and 64-Bit Linux



Choose one option for installation:

Option 1: If an install.sh is shipped inside the SDK root directory, execute this install script. It will copy all library files into the directory /usr/lib resp. /usr/lib64 and create symbolic links for each library file. The header file can be copied into a directory of your choice.

Option 2: Copy all files of this support package into a directory of your choice and create symbolic links for libfetcp.so.x.y.z[3] in the directory /usr/lib resp. /usr/lib64 with the following calls:

cd /usr/lib (for 64 Bit : /usr/lib64)

ln –s /< your_directory>/libfetcp.so.x.y.z libfetcp.so.x

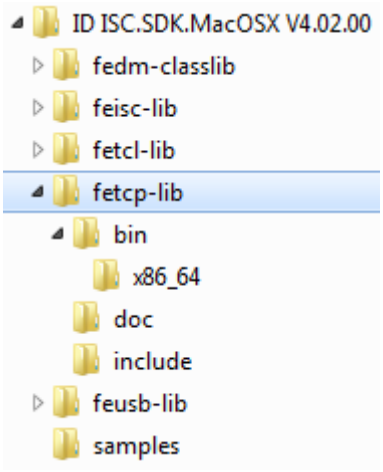
ln –s /< your_directory>/libfetcp.so.x libfetcp.so

ldconfig

The compiled library is linked against LibC V6 und LibStdC++ V6.

[3] *x.y.z represents the version number*

3.4. 64-Bit Mac OS X



Choose one option for installation:

Option 1: If an install.sh is shipped inside the SDK root directory, execute this install script. It will copy all library files into the directory /usr/local/lib and create symbolic links for each library file. The header file can be copied into a directory of your choice.

Option 2: Copy all files of this support package into a directory of your choice and create symbolic links for libfetcp.x.y.z.dylib[4] in the directory /usr/local/lib with the following calls:cd /usr/local/lib

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

ln –s libfetcp.x.dylib libfetcp.dylib

Note: The library is compiled under Mac OS X V10.7.3 with Xcode V4.3.2 and is compatible with the architecture x86_64. Newer compilations on request.

[4] *x.y.z represents the version number*

4. Incorporating Into the Application Program

4.1. Supported Development Tools

Operating System	Development Tool	Supported
Windows Vista / 7 / 8 / 10	Visual Studio	Yes
	Borland C++ Builder	Yes
	Embarcadero C++ Builder	Yes
Windows CE 6	eMbedded Visual C++ 4	No
	Visual Studio 2005	No
	Visual Studio 2008	Yes
Linux	GCC	Yes
Mac OS X	GCC	Yes, for projects with x86_64 architecture
	Xcode	Yes, for projects with x86_64 architecture

4.2. Incorporating into Visual Studio

1. Add Include path for the header file in project settings (category C/C++)
2. Add fetcp.lib (optional with path) in project settings (category Linker)

4.3. Incorporating into Xcode

1. Add path for the header file in project settings (User Header Search Paths in category Search Paths)
2. add fetcp.dylib with drag'n drop to your project

5. Programming Interface

5.1. Overview

The FETCP library encapsulates for the user all the necessary functions and parameters for managing one or more simultaneously opened TCP/IP channels (sockets). The object-oriented internal structure (see. Fig. 1) is intentionally brought out as a function interface. This gives it the virtue of being language-neutral.

The library has self-administration which frees the application program from having to buffer any values, settings, etc. The driver manager in FETCP keeps a list with all generated socket objects and each socket object manages all the settings relevant to its interface within its local memory.

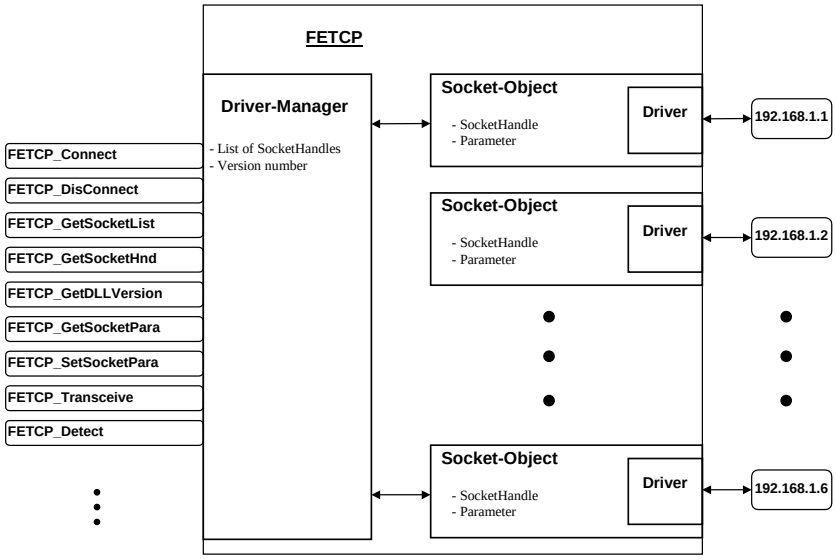


Fig. 1: Internal structure of FETCP

Before communicating for the first time, a socket object must be created. This is automatically done by the function **FETCP_Connect**. If this function was run without error, the return value includes a handle which can be managed by the application program. Only with this handle is unique identification of the opened socket object possible. The handle(s) do(es) not have to be saved in the application program, however, since the driver manager of the library manages internally a list of all opened sockets. This list can be opened with the function **FETCP_GetSocketList**. With the handles that are thus successively received you can then use the function **FETCP_GetSocketPara** to read all the settings pertaining to this socket, including the host address and the port number.

A socket object generated with **FETCP_Connect** must always deleted from the memory using the function **FETCP_DisConnect**, which also closes the socket.

If an application is opened multiple times, each program (instance) gets an empty socket list when **FETCP_GetSocketList** is used. This prevents mixing of access rights under various program instances.

If communication errors occurs with **FETCP_Transceive**, **FETCP_Transmit** or **FETCP_Receive,** the connection must be closed with **FETCP_DisConnect** and re-opened with **FETCP_Connect** again.

Each library function (with the exception of **FETCP_GetDLLVersion**) has a return value which in case of error is always negative.

The TCP/IP communication is not trivial. In particular the error handling may more complex as other port types. For basic information, the internet provides some good tutorials.

5.2. Thread Security

In principle, all FEIG libraries are not fully thread safe. But respecting some guidance, a practical thread security can be realized allowing parallel execution of communication tasks. One should keep in mind, that all FEIG RFID readers work synchronously and can perform commands only in succession.

On the level of the transport layer (FECOM, FEUSB, FETCP) the communication with each port must be synchronized in the application, as the reader works synchronously. Using multiple ports and so multiple readers from different threads simultaneously is possible, as the internal port objects acts independently from each other.

5.3. Error Handling for TCP/IP Communication

TCP/IP based communication is normally easy to realize. But in error cases, the handling is different from serial or USB based communication.

In the following, we discuss error handling for:

- Communication errors
- Errors while establish a connection
- Errors while closing the connection
- Problem with broken communication link

5.3.1. Communication Errors

In general, when a communication with the function `FETCP_Transceive` fails with error codes -1230 (Timeout), -1232 (Error in read process) or -1237 (error in send process), the connection must be closed at once and established again.

If a timeout is ignored and another FEIG protocol is sent afterwards, the timed out receive protocol may be received. After this, a displacement of the receive protocol is permanent existent. Only closing and opening of the connection can fix this situation.

The preset timeout is 3000ms and normally large enough for the most communication tasks. In rare cases it must be enlarged with the function `FETCP_SetPortPara`.

5.3.2. Errors While Establishing a Connection

Errors while establishing a connection with the FETCP_Connect function must be analyzed in detail depending on the error code.

Error code	Error handling
-1211	Timeout for establishing a connection to the TCP/IP server. Cause may be that another client is blocking the connection. This is a normal runtime problem. The repetitive call can be applied until the server (RFID reader) can be connected. Other reasons: the RFID reader is not powered on or not switched into the subnet or not configured properly concerning the TCP parameters. This must be analyzed by the installation team
-1212	The parameter cHostAdr in the function is structurally defective. This is a critical error and must be analyzed by the development team.
-1251	Pass parameter too large or too small, here: the transferred port number is out of range. This is a critical error and must be analyzed by the development team.

5.3.3. Errors While Closing the Connection

The closing of a connection is realized internally with a call of closesocket (Windows) or close (Linux, Mac OS X) and returns while the process of closing is not finished. Thus, although the disconnection from the application-side is finished, the final TCP status TIME_WAIT is probably not yet reached. To indicate this situation, the last TCP status is reflected to inform the application. With successive calls of the function `FETCP_GetSocketState(cHostAdr, iPortNr)` the closing process can be observed.

Only two important errors can occur with the function `FETCP_DisConnect`:

Error code	Error handling
-1213	The socket in the operating system cannot be closed and remains open. The disconnection must be repeated, until it is successful.
1 - 10	The socket is closed, but the last TCP status is returned as the final status TIME_WAIT is not reached. (see also: 5.7. List with TCP States)

5.3.4. Problem with Broken Communication Link – the Keep Alive Option

When the Ethernet cable gets broken while an active communication, the server-side application (reader) may not indicate an error while he is listening for new transmissions. On the other side, the host application will run in an error with the next transmission and can close and reopen the socket. But the close and reopen will never be noticed by the reader, as he is listening at a half-closed port.

The solution for this very realistic scenario is the activating of the Keep Alive option on the server-side. Every FEIG reader with Ethernet interface has parameters for Keep Alive and it is recommended to enable this option.

5.4. Functions

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

- **int FETCP_Connect(** char* cHostAdr, int iPortNr **)**
- **int FETCP_DisConnect(** int iSocketHnd **)**
- **int FETCP_GetSocketState(** char* cHostAdr, int iPortNr **)**
- **int FETCP_Detect(**char* cHostAdr, int iPortNr **)**
- **int FETCP_GetSocketList(** int iNext **)**
- **void FETCP_GetDLLVersion(** char* cVersion **)**
- **int FETCP_GetErrorText(** int iErrorCode, char* cErrorText **)**
- **int FETCP_GetLastError(** int iSocketHnd , int* iErrorCode, char* cErrorText **)**
- **int FETCP_GetSocketHnd(**char* cHostAdr, int iPortNr **)**
- **int FETCP_GetSocketPara(** int iSocketHnd, char* cPara, char* cValue **)**
- **int FETCP_SetSocketPara(** int iSocketHnd, char* cPara, char* cValue **)**
- **int FETCP_AddEventHandler**[5]**(** int iSocketHnd, FETCP_EVENT_INIT* pInit **)**
- **int FETCP_DelEventHandler**[2]**(** int iSocketHnd, FETCP_EVENT_INIT* pInit **)**
- **int FETCP_Transceive(** int iSocketHnd, UCHAR* cSendProt, int iSendLen, UCHAR* cRecProt, int iRecLen **)**
- **int FETCP_Transmit(** int iSocketHnd, UCHAR* cSendProt, int iSendLen **)**
- **int FETCP_Receive(** int iSocketHnd, UCHAR* cRecProt, int iRecLen **)**

[5] for future expansions

5.4.1. FETCP_Connect

Function	Opens a TCP/IP channel (socket) for communication with a FEIG reader (TCP/IP server).
Syntax	**int FETCP_Connect(char* cHostAdr, int iPortNr)**
Description	The function opens a TCP/IP channel, makes a connection to the TCP/IP server and internally creates an interface structure for managing the parameters. The function **FETCP_SetSocketPara** can be used to change these parameters after the fact. Use **FETCP_GetSocketPara** to read these parameters. The returned handle *iSocketHnd* identifies the interface from the outside. *cHostAdr* is a null-terminated string with the host address (e.g., "192.168.1.1"). *iPortNr* is the port number. The socket opened with **FETCP_Connect** must (!) be closed again using the function **FETCP_DisConnect**. Otherwise the memory reserved by the library is not released again. If communication errors occurs with **FETCP_Transceive**, **FETCP_Transmit** or **FETCP_Receive,** the connection must be closed with **FETCP_DisConnect** and re-opened with **FETCP_Connect** again.
Return value	If the socket was opened without error and a connection to the TCP/IP server established, a handle (>0) is returned. In case of error a value less than null is returned. The list of error codes can be found in the appendix.

Example	...
	#include "fetcp.h"
	...
	...
	int handle = FETCP_Connect("192.168.1.1", 4000);
	if(handle < 0)
	{
	// Code here for error
	}
	else
	{ // Communication possible
	// Code here for communication or other
	}

5.4.2. FETCP_DisConnect

Function	Ends a TCP/IP connection to a FEIG reader and closes the socket.
Syntax	**int FETCP_DisConnect(int iSocketHnd)**
Description	The function ends a TCP/IP connection to a FEIG reader, closes the socket indicated by the parameter *iSocketHnd* and releases the reserved memory. After the socket is closed, it is checked if the status TIME_WAIT is optained (s. 5.7. List with TCP States).
Return value	The return value is 0 if the socket was closed and the status TIME_WAIT is obtained. If the socket could be closed, but TIME_WAIT was not obtained after 500 ms, the last status (>0) is returned. In case of error the function returns a value less than null. The list of error codes can be found in the appendix.

Example	...
	#include "fetcp.h"
	...
	...
	int Err;
	...
	int handle = FETCP_Connect("192.168.1.1", 4000);
	if(handle < 0)
	{
	// Code here for error
	}
	...
	...
	if(handle > 0)
	{
	Err = FETCP_DisConnect(handle);
	if(Err == 0)
	{
	// OK
	}
	else if(Err > 0)
	{
	// status TIME_WAIT not obtained. Err contains last TCP status
	}
	else
	{
	// <0 means error
	}
	}
	...
	...

5.4.3. FETCP_Detect

Function	Checks whether a TCP/IP server can be reached.
Syntax	**int FETCP_Detect(char* cHostAdr, int iPortNr)**
Description	The function checks whether a TCP/IP server can be reached. If the connection is made, it is then immediately ended. *cHostAdr* is a null-terminated string with the host address (e.g. "192.168.1.1"). *iPortNr* is the port number.
Return value	If the connection test is successful, a 0 is returned, otherwise FETCP_ERR_SERVER_NOT_FOUND. In case of error the function returns a value less than null. The list of error codes can be found in the appendix.

Example	...
	#include "fetcp.h"
	...
	if(0 == FETCP_Detect("192.168.1.1", 4000);
	{
	// Connection to TCP/IP server possible
	}

5.4.4. FETCP_GetSocketState

Function	Request the status of a TCP/IP channel (socket).
Syntax	**int FETCP_GetSocketState(char* cHostAdr, int iPortNr)**
Description	The function checks for a TCP/IP channel, the obtained status of a connection to the TCP/IP server. After setup a connection with **FETCP_Connect**, the status should be ESTABLISHED (s 5.7. List with TCP States). After closing the connection with **FETCP_DisConnect**, the status should be TIME_WAIT. This function is helpful after communication problems to validate the connection. *cHostAdr* is a null-terminated string with the host address (e.g., "192.168.1.1"). *iPortNr* is the port number.
Return value	The status of the connection (>0). In case of error a value less than null is returned. The list of error codes can be found in the appendix.

Example	...
	#include "fetcp.h"
	...
	...
	int status = FETCP_GetSocketState("192.168.1.1", 4000);
	if(status < 0)
	{
	// Code here for error
	}
	else
	{ // status could be requested
	// a value between 1 and 12 defines the connection status
	}

5.4.5. FETCP_GetSocketList

Function	Uses parameter *iNext* to get the first or successive socket handle from the internal list of opened sockets.
Syntax	**int FETCP_GetSocketList(int iNext)**
Description	The function returns a socket handle from the internal list of socket handles. If you pass a 0 for *iNext* , the first entry from the list is returned. If you pass a socket handle contained in the list with *iNext* , the entry following the socket handle is returned. In this way you can go through the list from front to back and call up all the entries.
Return value	When an entry is found, the socket handle is returned with the return value. Once the end of the internal list is reached, i.e., the passed socket handle has no successor, a 0 is returned. If no socket is opened, FETCP_ERR_EMPTY_LIST is returned. In case of error the function returns a value less than null. The list of error codes can be found in the appendix.

Example	#include "fetcp.h"
	...
	// Function gets the settings of all opened sockets
	void TcpList(void)
	{ int iNextHnd = FETCP_GetSocketList(0); // get first handle
	while(iNextHnd > 0)
	{ // any code here
	...
	iNextHnd = FETCP_GetSocketList(iNextHnd); // Get next handle
	}
	...
	// Code here for example for displaying the list
	...
	}

Tip	When closing all opened sockets, it is helpful to use a loop similar to the example above. Bear in mind that you cannot get a successor to a closed socket. In the following code fragment is an example of how to close all opened sockets in a loop.
	...
	iNextHnd = FETCP_GetSocketList(0); // Get first handle
	while(iNextHnd > 0)
	{ iCloseHnd = iNextHnd;
	iNextHnd = FETCP_GetSocketList(iNextHnd); // Get only next handle
	iError = FETCP_DisConnect(iCloseHnd); // Only now close socket
	}
	...

5.4.6. FETCP_GetDLLVersion

Function	Gets the version number of the DLL/SO.
Syntax	**void FETCP_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. The string is filled with the current version number (e.g.”02.02.00”). Newer versions may be able to deliver more information.
Return value	none
Example	... #include "fetcp.h" ... ... char cVersion[256]; FETCP_GetDLLVersion(cVersion); // Code here for displaying version number ... ...

5.4.7. FETCP_GetErrorText

Function	Gets the error text associated with the error code
Syntax	**int FETCP_GetErrorText(int iErrorCode, char* cErrorText)**
Description	The function uses *cErrorText* to return the English error text associated with *iErrorCode*. The buffer for *cErrorText* should be able to hold at least 256 characters.
Return value	If no error, the function returns null and in case of error a value less than null. The list of error codes can be found in the appendix.
Example	... #include "fetcp.h" ... ... char cErrorText[256]; ... int iBack = FETCP_GetErrorText(FETCP_ERR_TIMEOUT, cErrorText) // Code here for displaying Text ... ...

5.4.8. FETCP_GetLastError

Function	Gets the last error code and passes the error text
Syntax	**int FETCP_GetLastError(int* iErrorCode, char* cErrorText)**
Description	The function uses *iErrorCode* to return the last error code and *cErrorText* to return the associated English text The buffer for *cErrorText* should be able to hold at least 256 characters.
Return value	If no error, the function returns null and in case of error a value less than null. The list of error codes can be found in the appendix.

Example	...
	#include "fetcp.h"
	...
	...
	char cErrorText[256];
	int iErrorCode = 0;
	...
	
	int iBack = FETCP_GetLastError(&iErrorCode, cErrorText)
	// Code here for displaying Text
	...
	...

5.4.9. FETCP_GetSocketHnd

Function	Gets the socket handle of an opened TCP/IP port
Syntax	**int FETCP_GetSocketHnd(char* cHostAdr, int iPortNr)**
Description	This function provides an easy way to get the socket handle of a previously opened TCP/IP port. *cHostAdr* is a null-terminated string with the host address (e.g. "192.168.1.1"). *iPortNr* is the port number.
Return value	If the specified TCP/IP port in the internal socket list was found, the socket handle (>0) is returned. If the desired TCP/IP port could not be found in the socket list, FETCP_ERR_NO_HND_FOUND is returned. In case of error the function returns a value less than null. The list of error codes can be found in the appendix.

Example	...
	#include "fetcp.h"
	...
	...
	int handle = FETCP_Connect("192.168.1.1", 4000);
	if(handle < 0)
	{
	// Code here for error
	}
	else
	{ // handle is gotten again using host address and port number
	handle = FETCP_GetSocketHnd("192.168.1.1", 4000);
	}

5.4.10. FETCP_GetSocketPara

Function	Gets a parameter for the TCP/IP port specified with *iSocketHnd*.
Syntax	**Int FETCP_GetSocketPara(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 ID. *cValue* is an empty, null-terminated string for returning the parameter value. The string should be able to hold at least 128 characters.
Parameter IDs	The parameter IDs are: HostAdr, PortNr, Timeout, CharTimeout, Language. Parameter Language sets the language in the DLL and is a global (not restricted to a socket handle) value. That means in this case you would set *iSocketHnd* to 0.
Return value	If there is no error the function returns a value of 0, and in case of error a value less than null. The list of error codes can be found in the appendix.
Cross-reference	For additional information see: 5.6. List of Parameter IDs.

Example	...
	#include "fetcp.h"
	...
	...
	char cValue[128];
	...
	if(!FETCP_GetSocketPara(handle, "HostAdr", cValue))
	{
	// Code here for displaying host address
	...
	}
	...
	...
	}

5.4.11. FETCP_SetSocketPara

<table>
<tr><td>Function</td><td colspan="4">Sets a parameter for a TCP/IP port to a new value.</td></tr>
<tr><td>Syntax</td><td colspan="4">int FETCP_SetSocketPara(int iPortHnd, char* cPara, char* cValue)</td></tr>
<tr><td>Description</td><td colspan="4">The function passes a new parameter to the TCP/IP port specified by iSocketHnd.
cPara is a null-terminated string with the parameter ID.
cValue is a null-terminated string with the new parameter value.</td></tr>
<tr><td></td><td>Parameter ID</td><td>Value range</td><td>Default value</td><td>Unit</td></tr>
<tr><td></td><td>Timeout</td><td>0...99999</td><td>3000</td><td>ms</td></tr>
<tr><td></td><td>CharTimeout</td><td>1…999</td><td>25</td><td>ms</td></tr>
<tr><td></td><td>Language</td><td>7, 9</td><td>9</td><td>-</td></tr>
<tr><td></td><td>UseOBID</td><td>0, 1</td><td>1</td><td>-</td></tr>
<tr><td>Return value</td><td colspan="4">If there is no error the function returns a value of 0, and in case of error a value less than null. The list of error codes can be found in the appendix.</td></tr>
<tr><td>Cross-reference</td><td colspan="4">For additional information see: 5.6. List of Parameter IDs.</td></tr>
<tr><td>Example</td><td colspan="4">...
#include "fetcp.h"
...
...
int Err;
...
int handle = FETCP_Connect("192.168.1.1", 4000);
if(handle > 0)
{
Err = FETCP_SetSocketPara(handle, "Timeout", "5000");
...
}
...
...</td></tr>
</table>

5.4.12. FETCP_Transceive

Function	Function for socket communication (Transmit and Receive).
Syntax	**int FETCP_Transceive(int iPortHnd, UCHAR* cSendProt, int iSendLen, UCHAR* cRecProt, int iRecLen)**
Description	The function sends the data contained in *cSendProt* 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 immediately. The characters received up to the point of the cancel are stored in *cRecProt*. Prior to communication the transmit and receive buffers are deleted.
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 “fetcp.h”
	...
	int iSendLen;
	int iRecProtLen;
	char* cHost = “192.168.1.1”;
	int iPortNr = 10001;
	UCHAR cSendBuf[256]; // Adjust buffer size to transmit data if needed
	UCHAR cRecBuf[256]; // Adjust buffer size to receive data if needed
	…
	int handle = FETCP_Connect(cHost, iPortNr);
	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 the socket, if successful, the receive data are located in RecBuf
	iRecProtLen = FETCP_Transceive(handle, cSendBuf, iSendLen, cRecBuf, 256);
	if(cRecProtLen < 0)
	{
	// Communication error
	}
	}

5.4.13. FETCP_Transmit

Function	Function for sending a protocol.
Syntax	**int FETCP_Transmit(int iPortHnd, UCHAR* cSendProt, int iSendLen)**
Description	The function sends the data contained in *cSendProt* 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 “fetcp.h”
	...
	...
	int iErr;
	int iSendLen;
	char* cHost = “192.168.1.1”;
	int iPortNr = 10001;
	UCHAR cSendBuf[256]; // Buffer size may need to be adjusted to the send data
	…
	int handle = FETCP_Connect(cHost, iPortNr);
	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 the socket
	iErr = FETCP_Transmit(handle, cSendBuf, iSendLen);
	if(iErr < 0)
	{
	// Communication error
	}
	}

5.4.14. FETCP_Receive

Function	Function for receiving a protocol.
Syntax	**int FETCP_Receive(int iPortHnd, UCHAR* cRecProt, int iRecLen)**
Description	The function expects data received through the socket within the timeout time (see 5.6. List of Parameter IDs), 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 returns immediately. 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 “fetcp.h”
	…
	…
	int iRecProtLen;
	char* cHost = “192.168.1.1”;
	int iPortNr = 10001;
	UCHAR cRecBuf[256]; // Buffer size may need to be adjusted to the receive data
	…
	int handle = FETCP_Connect(cHost, iPortNr);
	if(handle < 0)
	{
	  // code here for error condition
	}
	else
	{ // Communication through the socket, if successful the receive data will be located in RecBuf
	  iRecProtLen = FETCP_Receive(handle, cRecBuf, 256);
	  if(iRecProtLen < 0)
	  {
	  // Communication error
	  }
	}

APPENDIX

5.5. Error Codes

Error constants	Value	Description
FETCP_ERR_NEWSOCKET_FAILURE	-1200	Error in creating a new socket object. Insufficient memory is a possible cause.
FETCP_ERR_EMPTY_LIST	-1201	Socket handle is empty (no socket objects created).
FETCP_ERR_POINTER_IS_NULL	-1202	A point is null and therefore invalid.
FETCP_ERR_NO_MEMORY	-1203	Insufficient memory
FETCP_ERR_SERVER_NOT_FOUND	-1205	Returns FETCP_Detect if no connection to the specified server was possible.
FETCP_ERR_NO_CONNECTION	-1211	Timeout for establishing a connection to the TCP/IP server. Cause may also be that another client is blocking the connection.
FETCP_ERR_SERVER_ADDR	-1212	The parameter cHostAdr in the functions FETCP_Detect or FETCP_Connect is structurally defective.
FETCP_ERR_UNKNOWN_HND	-1220	The passed socket handle is unknown.
FETCP_ERR_HND_IS_NULL	-1221	The passed socket handle is 0
FETCP_ERR_HND_IS_NEGATIVE	-1222	The passed socket handle is negative
FETCP_ERR_NO_HND_FOUND	-1223	No socket handle found in the socket handle list
FETCP_ERR_TIMEOUT	-1230	Timeout when reading socket
FETCP_ERR_RECEIVE_PROCESS	-1232	Error in receive process
FETCP_ERR_CHANGE_PORT_PARA	-1236	Error in changing a port parameter
FETCP_ERR_TRANSMIT_PROCESS	-1237	Error in send process
FETCP_ERR_GET_CONNECTION_STATE	-1238	Error while reading the connection status
FETCP_ERR_UNKNOWN_PARAMETER	-1250	Pass parameter is unknown
FETCP_ERR_PARAMETER_OUT_OF_RANGE	-1251	Pass parameter too large or too small
FETCP_ERR_ODD_PARAMETERSTRING	-1252	An unsupported option was invoked by the pass parameter
FETCP_ERR_UNKNOWN_ERRORCODE	-1254	Unknown error code
FETCP_ERR_BUFFER_OVERFLOW	-1270	Receive buffer is too small

5.6. List of Parameter IDs

Parameter ID	Value range	Default	Unit	Description
HostAdr				Host address (e.g. "192.168.1.1") Address is read-only
PortNr	0…65535		-	Port number Port number is read-only
Timeout	0...99999	3000	ms	Maximum wait time for receive protocol
CharTimeout	1...999	250	ms	The character timeout determines after how much time after receipt of the last character the receive process is ended.
Language	7 - German 9 - English	9	-	Selects the language for internal text resources.
UseOBID	0, 1	1	-	activates internally a specialized receive algorithm adopted to OBID protocol frames to increase the communication performance

5.7. List with TCP States

TCP State	Value
FETCP_STATE_CLOSED	1
FETCP_STATE_LISTEN	2
FETCP_STATE_SYN_SENT	3
FETCP_STATE_SYN_RCVD	4
FETCP_STATE_ESTABLISHED	5
FETCP_STATE_FIN_WAIT1	6
FETCP_STATE_FIN_WAIT2	7
FETCP_STATE_CLOSE_WAIT	8
FETCP_STATE_CLOSING	9
FETCP_STATE_LAST_ACK	10
FETCP_STATE_TIME_WAIT	11
FETCP_STATE_DELETE_TCB	12

5.8. Revision History

V2.02.00

- Bug fix: avoiding buffer overflow with too small receive buffers
- New error code: FETCP_ERR_BUFFER_OVERFLOW
- Improved thread safeness
- Windows:
  1. Migration from Visual Studio 2008 to Visual Studio 2010.
  2. DLL without MFC
  3. First release of 64-Bit version
  4. Dynamic binding to log manager
- First release for Mac OS X, V10.7.3 or higher
- Linux:
  1. Version for 64-Bit

V2.00.00

- New function: **FETCP_GetSocketState**
- The function **FETCP_DisConnect** can now return with a positive value. This can imply code modifications in applications. Thus, it is recommended to view every code line with **FETCP_DisConnect**. More information can be found in **5.2.2. FETCP_DisConnect**
- New error code -1238 (Error while reading the connection state)
- Windows / Windows CE:
  1. Migration of the development environment from Visual Studio 6 to Visual Studio 2008.
  2. Adaptation of the callback declaration in ***struct _FETCP_EVENT_INIT*** concerning the calling convention. Thus, this version of FETCP is not compatible with the previous version and with applications compiled against the previous version of FETCP. Code modifications are not necessary, but re-compilation of applications is mandatory.

V1.02.05

- Internal modification

V1.02.04

- Version for Windows CE

- Improved communication performance with using of specialized receive algorithm adopted to FEIG protocol frames. Can be disabled with the new parameter UseOBID.

V1.02.03

- Modified licence agreement
- No length limits for **FETCP_Transmit**, **FETCP_Receive**
- **FETCP_Transmit**, **FETCP_Receive** for Visual Basic 6
- The Linux library is compiled with GCC 3.3.3 under SuSE Linux 9.1

V1.02.00

- New functions: **FETCP_Transmit**, **FETCP_Receive**
- First Linux release (SuSE Linux 8.2, GNU Compiler Collection V3.3-23, glibc V2.3.2-6)

V1.00.00

- This is the first release version.