OBID classic-pro

MANUAL

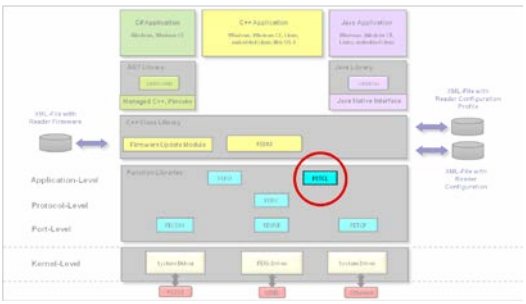
ID FETCL

Version 2.02.01

T=CL Interface

for

OBID® *classic-pro* Reader



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

Note

© Copyright 2005-2014 by FEIG ELECTRONIC GmbH
Lange Straße 4
D-35781 Weilburg-Waldhausen
Germany
Tel.: +49 6471 3109-0
http://www.feig.de

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

Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

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

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

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

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

OBID® and OBID *i-scan*® are registered Trademarks of FEIG ELECTRONIC GmbH.

Windows is a registered trademark of Microsoft Corporation in the United States and other countries.

Windows Vista is either a registered trademark or trademark of Microsoft Corporation in the United States and/or other countries.

Linux® is a registered Trademark of Linus Torvalds.

Apple, Mac, Mac OS, OS X, Cocoa and Xcode are trademarks of Apple Inc., registered in the U.S. and other countries.

I-CODE® and Mifare® are registered Trademarks of Philips Electronics N.V.

Tag-it (TM) is a registered Trademark of Texas Instruments Inc.

Licensing agreement for use of the software

This is an agreement between you and FEIG ELECTRONIC GmbH (hereafter "FEIG") for use of the ID FETCL 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 FETCL.DLL, FETCLCE.DLL, LIBFETCL.x.y.z.DYLIB[1] or LIBFETCL.SO.x.y.z[1] without licensing fees under the stipulation that these application programs or program libraries are used to control devices and/or systems which are developed and/or sold by FEIG.

§2. Protection of the licensing material

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

§3 Warranty and liability limitations

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

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

§4 Concluding provisions

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

Contents:

Contents: 5

1. Introduction 7

1.1. Shipment 8

1.1.1. Windows XP / Vista / 7 / 8 8
1.1.2. Windows CE 8
1.1.3. Linux 8
1.1.4. Mac OS X 8

2. Changes since the previous version 9

3. Installation 10

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

3.2. Windows CE 11

3.3. 32- and 64-Bit Linux 12

3.4. 64-Bit Mac OS X 13

4. Including into the application program 14

4.1. Supported Development Tools 14

4.2. Incorporating into Visual Studio 14

4.3. Incorporating into Xcode 14

5. Programming Interface 15

5.1. Overview 15

5.2. Thread security 17

5.3. Event flagging to applications 18

5.4. List of functions 20

Administration functions for Transonder-Objects 20

Query functions 20

Special communication functions 20

5.4.1. FETCL_NewTransponder 21
5.4.2. FETCL_DeleteTransponder 22
5.4.3. FETCL_GetTransponderList 23
5.4.4. FETCL_GetDLLVersion 24
5.4.5. FETCL_GetErrorText 24
5.4.6. FETCL_GetLastError 25

5.4.7. FETCL_APDU 26
5.4.8. FETCL_Ping 27
5.4.9. FETCL_Deselect 27
5.4.10. FETCL_GetResponseData 27

6. Appendix 28

6.1. Error codes 28

6.2. List of constants for the FEISC_EVENT_INIT structure 29

6.3. History 30

1. Introduction

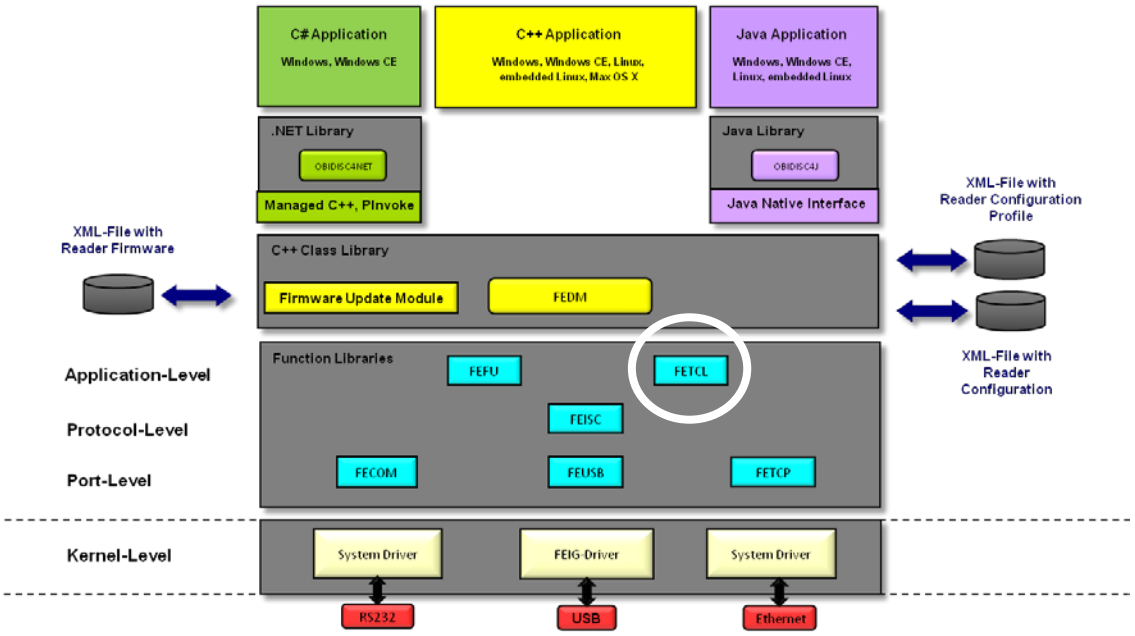
The support package ID FETCL is intended to support in programming T=CL based application software, integrate the OBID® *classic-pro* Reader, and supports ANSI-C, ANSI-C++ and essentially any other language which can invoke C functions.

The support package provides a simple T=CL function interface for easy APDU exchange and is designed to work together with the FEISC.DLL for the OBID i-*scan*® and OBID® *classic-pro* Reader.

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 FETCL is part of the third level of a hierarchical structured, multi-tier FEIG library stack. The following picture shows the multi-tier library stack.



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 XP / Vista / 7 / 8

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

1.1.2. Windows CE

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

1.1.3. Linux

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

1.1.4. Mac OS X

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

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

2. Changes since the previous version

- Bugfix for calculating Bock-ID for communication over Serial Port
- Improvements for receiving multiple frames over TCP/IP

- Linux:

  1st Version for 64-Bit

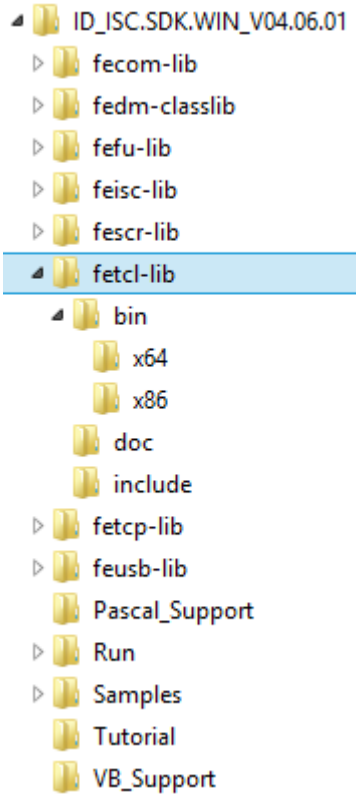
Please note also the revision history in the Appendix to this document.

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 fetcl-lib.

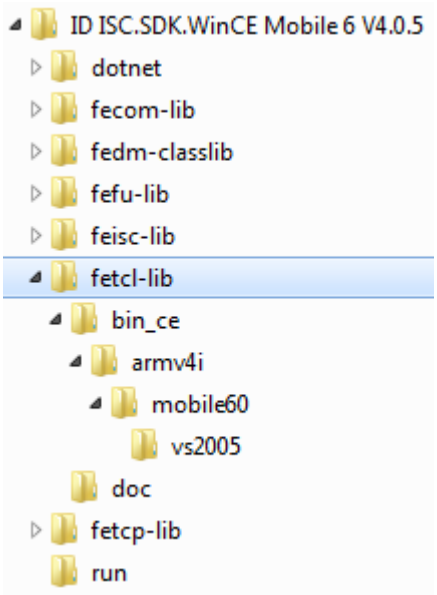
3.1. 32- and 64-Bit Windows XP/Vista/7/8



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

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

3.2. Windows CE

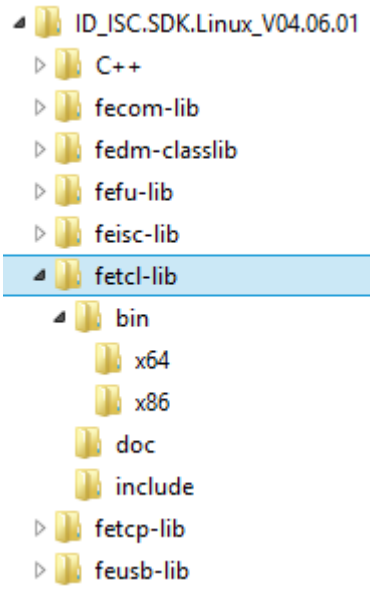


If you won't add your projects to the Samples path, we recommend the following steps:

- Copy FETCLCE.DLL into the application directory or system directory of the Windows CE system.
- Copy FETCLCE.LIB into the project or LIB directory.
- Copy FETCL.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 creates 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 libfetcl.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>/libfetcl.so.x.y.z libfetcl.so.x

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

ldconfig

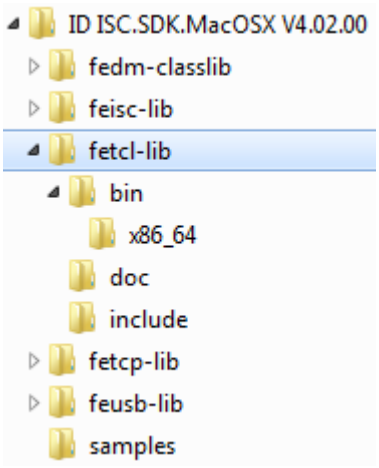
Note:

X86: The library is compiled under SuSE Linux 11.1 with the GNU Compiler Collection V4.3.2.

X64: The library is compiled under SuSE Linux 11.2 with the GNU Compiler Collection V4.4.1.

[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 creates 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 libfetcl.x.y.z.dylib[4] in the directory /usr/local/lib with the following calls:cd /usr/local/lib

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

ln –s libfetcl.x.dylib libfetcl.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.

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

4. Including into the application program

4.1. Supported Development Tools

Operating System	Development Tool	Supported
Windows XP / Vista / 7 / 8	Visual Studio	Yes
	Borland C++ Builder	Yes
	Embarcadero C++ Builder	Yes
Windows CE	eMbedded Visual C++ 4	Yes
	Visual Studio 2005 / 2008	Yes
Linux	GCC	Yes
Mac OS X	GCC	Yes, for projects with x86_64 architecture
	Xcode ≥ V4.3.2	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 fetcl.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 fetcl.dylib with drag'n drop to your project

ID FEISC and one of the packages ID FECOM and/or ID FEUSB and/or ID FETCP must also be incorporated into your project.

5. Programming Interface

5.1. Overview

The FETCL library encapsulates for the programmer all the functions and parameters necessary for simple T=CL based communication with ISO14443-4 compliant transponders, accessed by a reader of the OBID® *classic-pro* Reader Family. Together with the support packages ID FEISC and one of ID FECOM, ID FETCP or ID FEUSB, this makes it possible to handle complex T=CL commands by invoking of one function, which executes a communication cycle asynchonously to the application program. The end of the cycle is signaled by an user defined event flagging mechanism.

The functions in FETCL are responsible only for internal administration, T=CL protocol cycle handling, T=CL response data collection and any necessary error outputs. Every other ISO14443 commands, like Inventory, Select or Halt must be executed with FEISC.

The FETCL library alone is not enough to communicate with an OBID® *classic-pro* Reader. You can however initiate the communication process and use the FEISC and one of the port libraries (FECOM, FEUSB, FETCP) to communicate with an OBID® *classic-pro* Reader over an asynchronous serial interface or with a TCP/IP-Server or through the USB port. Other interface drivers can be integrated with the Plug-In mechanism of the FEISC.

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

The core elements of the library are the Object Manager and the Transponder objects generated during runtime. After an Inventory it is mandatory to create a Transponder object for every Transponder. Every Transponder object represents an ISO14443-4 compliant transponder. After remove of the transponder, the Transponder object must be deleted manually by the application.

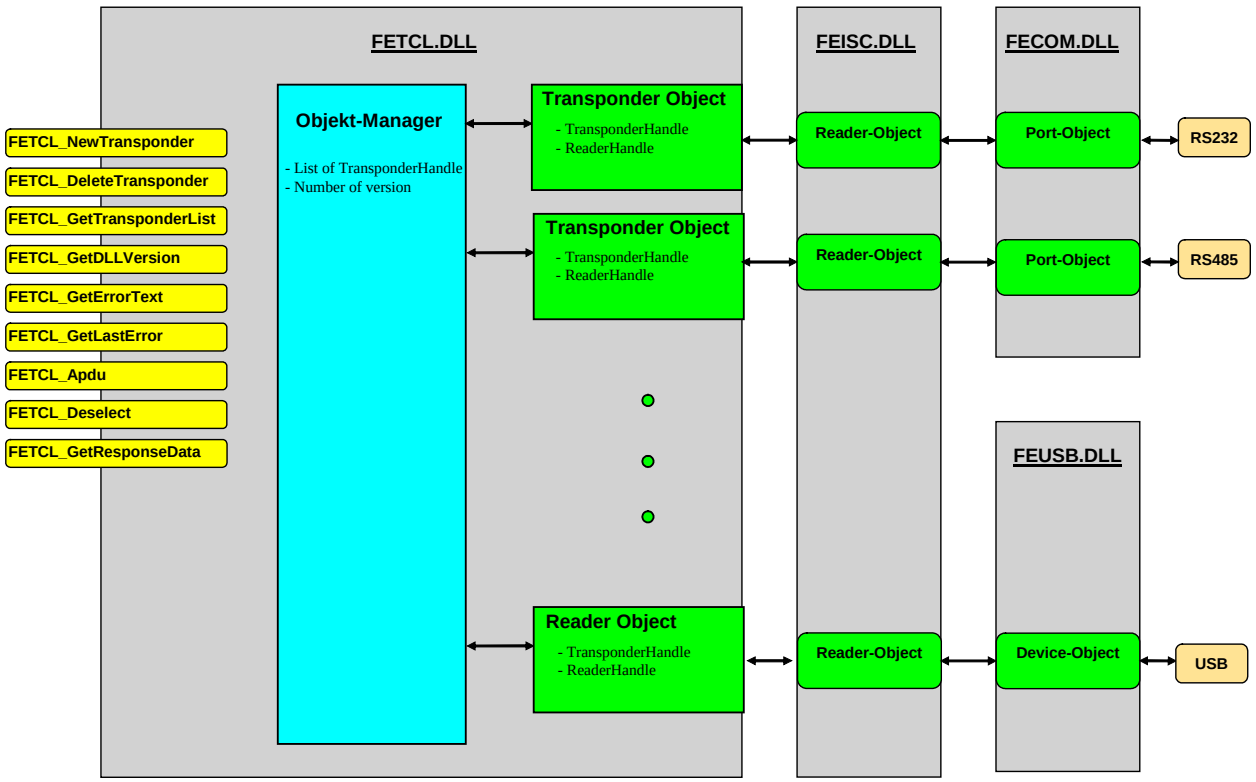
The TCL Object Manager implements self-administration which frees an application program from having to buffer any values, parameters or other settings: It keeps a list with all generated Transponder objects. The Transponder object is the central program section that carries out the communication cycles. Each Transponder object administers all the parameters relevant to its protocol tasks within its local memory and are independent of each other. However, for every OBID® *classic-pro* Reader the application can start only one T=CL communication cycle at the same time.

After an Inventory and before first using you must create a Transponder object using the **FETCL_NewTransponder** function. If this is done without error, the return value includes a handle which is used by the application program as an access number. This handle is required for unique identification of the generated Transponder object. If you are using self-administration, the Transponder Object List can be called up using the **FETCL_GetTransponderList** function.

A Transponder object generated using **FETCL_NewTransponder** must always be deleted from memory using the **FETCL_DeleteTransponder** function.

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

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



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 OBID® RFID-Reader works 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.

On the level of the protocol layer (FEISC), parallelism can be realized, when each Reader object represents exactly one physical Reader and is bound with an individual communication port. This is not true for the four specialized functions FEISC_BuildxxProtocol and FEISC_SplitxxProtocol, which use an internal global buffer for protocol data.

The library FETCL for ISO 14443-4 compliant Transponders is thread-safe, only when each Transponder object is connected with a different Reader object and only one APDU is exchanged with each Reader at the same time. Even if the function FETCL_Apdu can be called asynchronously, this means not, that multiple calls of FETCL_Apdu to the same Transponder object are allowed. APDUs are not stored in a stack.

5.3. Event flagging to applications

Event handling mechanisms can optionally be installed for the APDU communication function. As the schedule of the communication process is not predictable, it is recommended to install an event flagging mechanism to notify the application asynchronous to the program sequence. The event handling mechanism is executed to signal the end of the communication cycle.

An event handling mechanism can be installed with every APDU communication function and is valid only for one communication cycle. You may choose between three various flagging methods: Message to a calling process, message to a window or use of a callback function.

An already installed event handling mechanism is deleted automatically after the APDU communication cycle is finished.

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

```
typedef struct _FETCL_EVENT_INIT
{
    void* pAny;       // pointer to anything, which is reflected as the first parameter
                      // in the callback function cbFct2 (e.g. can be used to pass an object pointer)
    UINT uiUse;       // Defines the event (e.g. FETCL_xxx_EVENT)
    UINT uiMsg;       // Message Code for dwThreadID and hwndWnd (e.g. WM_USER_xyz)
    UINT uiFlag;      // Specifies use of the union (e.g. FETCL_WND_HWND)
    union
    {
        DWORD     dwThreadID;              // for Thread-ID
        HWND      hwndWnd;                 // for Window-Handle
        void      (*cbFct)(int, int, int); // for Callback-Function
        void      (*cbFct2)(void*, int, int, int); // for Callback-Function
    }Method[5];
} FETCL_EVENT_INIT;
```

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

When an event is arisen, the event handling method transports informations from the library to the application. The following table lists these function parameters:

Event Mechanism (uiUse)	Event Handling Method	Parameter
Message to a thread of an application (FETCL_THREAD_ID)	registration of event handling method with: ON_THREAD_MESSAGE(WM_USER_MY_ID, OnMyThreadMsg) event handling method: OnMyThreadMsg(UINT hnd, LPARAM err)	*hnd* is the Transponder Handle, returned by FETCL_NewTransponder. *err* is an error code as a result of the APDU communication process.

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

Event Mechanism (uiUse)	Event Handling Method	Parameter
Message to a window (FETCL_WND_HWND)	registration of event handling method with: ON_MESSAGE(WM_USER_MY_ID, OnMyMsg) event handling method: OnMyThreadMsg(WPARAM hnd, LPARAM err)	*hnd* is the Transponder Handle, returned by FETCL_NewTransponder. *err* is an error code as a result of the APDU communication process.
Invoke of a callback function (FETCL_CALLBACK)	MyCallback1(int hnd, int err, int len) or MyCallback2(void* pAny, int hnd, int err, int len)	*hnd* is the Transponder Handle, returned by FETCL_NewTransponder. *err* is an error code as a result of the APDU communication process. *len* is the length of the APDU response (number of bytes), which can be get with FETCL_GetResponseData *pAny* is a Pointer to anything, which is reflected as the first parameter (e.g. can be used to pass an object pointer)

5.4. List of functions

The support package contains functions for various tasks. They are divided into groups for better orientation.

Administration functions for Transonder-Objects

- **int FETCL_NewTransponder(** int iReaderHnd, UCHAR ucBusAdr, UCHAR ucCid, UCHAR ucNad, bool bUseCid, bool bUseNad **)**
- **int FETCL_DeleteTransponder(** int iTrpHnd **)**
- **int FETCL_GetTransponderList(** int iNext **)**

Query functions

- **void FETCL_GetDLLVersion(** char* cVersion **)**
- **int FETCL_GetErrorText(** int iErrorCode, char* cErrorText **)**
- **int FETCL_GetLastError(** int iTrpHnd , int* iErrorCode, char* cErrorText **)**

Special communication functions

- **int FETCL_Apdu(** int iTrpHnd, UCHAR* ucData, int iDataLen, FETCL_EVENT_INIT* pInit **)**
- **int FETCL_Ping(** int iTrpHnd **)**
- **int FETCL_Deselect(** int iTrpHnd **)**
- **int FETCL_GetResponseData(** int iTrpHnd, UCHAR* ucData, int iDataBufLen **)**

5.4.1. FETCL_NewTransponder

Function	Creates a Transponder Object.
Syntax	**int FETCL_NewTransponder(int iReaderHnd, UCHAR ucBusAdr, UCHAR ucCid, UCHAR ucNad, bool bUseCid, bool bUseNad)**
Description	A Transponder object is created. Communications based on APDUs require a Transponder Object in order to run. Multiple Transponder Objects can in principle carry out their communication over the same Reader Object, if the communication protocols are scheduled successive. A Transponder Object created with **FETCL_NewTransponder** must (!) be deleted from memory using the **FETCL_DeleteTransponder** function. Otherwise the memory reserved by the library is not freed up again. *iReadertHnd* is the handle of a Reader Object created from FEISC using the **FEISC_NewReader** function. *ucBusAdr* is the bus address of the Reader. For USB Reader, this parameter has no impact. *ucCid* is the card identifier and is used internally, if *bUseCid* is true. *ucNad* is the card identifier and is used internally, if *bUseNad* is true.
Return value	If a Transponder Object was created without error, a handle (>0) is returned. In case of error, the function returns a value less than zero. A list of error codes can be found in the Appendix.

Example	...
	#include "fecom.h"
	#include "feisc.h"
	#include "fetcl.h"
	...
	...
	char cPortNr[4];
	itoa(1, cPortNr, 10); // Convert Integer to Char
	
	int iPortHnd = FECOM_OpenPort(cPortNr); // COM:1 should be opened
	if(iPortHnd < 0)
	{
	// code here in case of error
	}
	else
	{ // Open Reader object
	int iReaderHnd = FEISC_NewReader(iPortHnd);
	if(iReaderHnd > 0)
	{
	int iTrpHnd = FETCL_NewTransponder(iReaderHnd, 255, 0, 0, false, false);
	}
	}

5.4.2. FETCL_DeleteTransponder

Function	Deletes a Transponder object
Syntax	**int FETCL_DeleteTransponder(int iTrpHnd)**
Description	The function deletes the Transponder Object indicated by the parameter *iTrprHnd* and frees up the reserved memory.
Return value	The return value is 0 if the action was successful. In case of error, the function returns a value less than zero. A list of error codes can be found in the Appendix.

Example	...
	#include "feisc.h"
	#include "fetcl.h"
	...
	...
	int iErr;
	int iReaderHnd = FEISC_NewReader(iPortHnd);
	if(iReaderHnd < 0)
	{
	  // code here in case of error
	}
	...
	...
	...
	if(iReaderHnd > 0)
	{
	  int iTrpHnd = FETCL_NewTransponder(iReaderHnd);
	  ...
	  iErr = FETCL_DeleteTransponder(iTrpHnd);
	}
	...
	...

5.4.3. FETCL_GetTransponderList

Function	Depending on the *iNext* parameter, gets the first or following Transponder handle from the internal list of the generated Transponder Objects.
Syntax	**int FETCL_GetTransponderList(int iNext)**
Description	The function returns a Transponder handle from the internal list of Transponder handles. If one transmits a 0 for *iNext*, the first entry in the list is returned. If you transmit a Transponder handle contained in the list with *iNext*, the function gets and returns the entry following the Transponder handle. In this way you can keep incrementing the return value to go through the list and call out all the entries.
Return value	When an entry is found, the Transponder handle is provided with the return value. When the end of the internal list is reached, in other words the transferred Transponder handle has no following entry, a 0 is returned. If there is no Transponder Object, FETCL_ERR_EMPTY_LIST is returned. In case of error, the function returns a value less than zero. A list of error codes can be found in the Appendix.

Example	...
	#include "fetcl.h"
	...
	// Example function for creating a list of Transponder objects
	void TransponderList(void)
	{
	int iNextHnd = FETCL_GetTransponderList(0); // get the first handle
	while(iNextHnd > 0)
	{
	// here for example code for collecting the handles and reading out parameters
	...
	iNextHnd = FETCL_GetTransponderList(iNextHnd); // get next handle
	}
	...
	// here for example code for displaying a list
	}

Tip	When closing all open created Transponder Objects it is convenient to use a loop such as in the example above. Bear in mind however than you cannot get the next in line from a deleted Transponder Object. The following code fragment gives you an idea of how to delete all created Transponder Objects in a loop:
	...
	int iNextHnd, iCloseHnd, iError;
	iNextHnd = FETCL_GetTransponderList(0); // get first handle
	while(iNextHnd > 0)
	{
	iCloseHnd = iNextHnd;
	iNextHnd = FETCL_GetTransponderList(iNextHnd); // get next handle
	iError = FETCL_DeleteTransponder(iCloseHnd); // only now delete Transponder Object
	}

5.4.4. FETCL_GetDLLVersion

Function	Gets the DLL/SO version number.
Syntax	**void FETCL_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.”02.02.01“). Newer versions may provide additional information.
Return value	none
Example	... #include "fetcl.h" ... ... char cVersion[256]; FETCL_GetDLLVersion(cVersion); // code here for displaying the version number ... ...

5.4.5. FETCL_GetErrorText

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

5.4.6. FETCL_GetLastError

Function	Gets the last error code and transmits error text.
Syntax	**int FETCL_GetLastError(int iReaderHnd , int* iErrorCode, char* cErrorText)**
Description	The function uses *iErrorCode* to send the last error code of the Reader object selected with*iReaderHnd* and transmits the associated error text in *cErrorText*. The buffer for *cErrorText* should be able to hold at least 256 characters.
Return value	If no error the function returns zero, and in case of error a value less than zero. A list of error codes can be found in the Appendix.

Example	...
	#include "fetcl.h"
	...
	...
	char cErrorText[256];
	int iErrorCode = 0;
	...
	
	int iBack = FETCL_GetLastError(iTrpHnd, &iErrorCode, cErrorText)
	// code here for displaying the text
	...
	...

5.4.7. FETCL_APDU

Function	Function executes an APDU.
Syntax	**int FETCL_APDU(int iTrpHnd,** **UCHAR* ucData,** **int iDataLen,** **FETCL_EVENT_INIT* pInit)**
Note	This function initiates an APDU communication process and returns immediately, if an event handler is passed to the function. In this case, the APDU communication process is handled internally by a thread and after finish of the communication, an event handler informs the application to read the APDU response data. In case of blocking behaviour, which is selected with passing NULL for pInit, the function handles the APDU communication process without a thread. One Transponder Object can handle only one APDU command at the same time. Multiple APDU communication processes can be started, if each APDU is handled by a different Reader, which implies the use of multiple Reader Objects in the FEISC library. *iTrpHnd* is the handle for the Transponder Object. *ucData* is a pointer to a buffer with the ADPU command. *iDataLen* contains the number of bytes in ucData. *pInit* is a Pointer to an initialized structure with event handling parameters.
Return value	If no error the function returns zero, and in case of error a value less than zero. A list of error codes can be found in the Appendix.

Example	...
	#include "fetcl.h"
	...
	// declaration of callback function
	void FEApduCallback(int iApduHnd, int iError, int iDataLength);
	
	// global APDU response buffer
	extern unsigned char g_ucApduData[FETCLDEMO_MAX_TCL_BUFFER];
	...
	int MyApdu()
	{
	unsigned char* pucApdu ;
	FETCL_EVENT_INIT Init;
	...
	// build the APDU, allocate menory and save it in pucApdu.
	…
	Init.uiFlag = FETCL_CALLBACK;
	Init.cbFct = &FEApduCallback;
	
	// execute the APDU. The function returns immediately
	return FETCL_APDU(iTrpHnd, pucApdu, iApduLen, &Init)
	}
	
	void FEApduCallback(int iApduHnd, int iError, int iDataLength)
	{
	if (iError != 0)
	return;
	
	if(iDataLength > FETCLDEMO_MAX_TCL_BUFFER)
	return;

```
int iErr = FETCL_GetResponseData( iApduHnd, g_ucApduData, FETCLDEMO_MAX_TCL_BUFFER );
if( iErr < 0)
    return;
// do anythig with the APDU response
}
```

5.4.8. FETCL_Ping

Function	Function tests a transponder.
Syntax	**int FETCL_Ping(int iTrpHnd)**
Description	This function tests the presence of selected a transponder with a ping command. *iTrpHnd* is the handle for the Transponder Object.
Return value	If no error the function returns zero, and in case of error a value less than zero. A list of error codes can be found in the Appendix.

5.4.9. FETCL_Deselect

Function	Function deselects a transponder.
Syntax	**int FETCL_Deselect(int iTrpHnd)**
Description	This function deselects a transponder. *iTrpHnd* is the handle for the Transponder Object.
Return value	If no error the function returns zero, and in case of error a value less than zero. A list of error codes can be found in the Appendix.

5.4.10. FETCL_GetResponseData

Function	Function for transfer of response data.
Syntax	**int FETCL_GetResponseData(int iTrpHnd, UCHAR* ucData, int iDataBufLen)**
Description	This function returns the APDU response data. The finish of the APDU process is signaled with an event handler or, in case of blocking behaviour, after the return of FETCL_Apdu. in case of non-blocking behaviour, the read of response data should happen inside this event handling function. *iTrpHnd* is the handle for the Transponder Object.
Return value	If no error the function returns the number of bytes inside ucData, and in case of error a value less than zero. A list of error codes can be found in the Appendix.

6. Appendix

6.1. Error codes

Error constants	Value	Description
FETCL_ERR_NEW_TRANSPONDER_FAILURE	-4200	Error in creating a new Transponder Object
FETCL_ERR_EMPTY_LIST	-4201	Transponder handle list is empty (no Transponder Objects stored)
FETCL_ERR_POINTER_IS_NULL	-4202	Pointer to transfer parameter is NULL
FETCL_ERR_NO_MORE_MEM	-4203	No more system memory
FETCL_ERR_NO_VALUE	-4210	No data value
FETCL_ERR_UNKNOWN_HND	-4220	The transferred Transponder handle is unknown
FETCL_ERR_HND_IS_NULL	-4221	The transferred Transponder handle is 0
FETCL_ERR_HND_IS_NEGATIVE	-4222	The transferred Transponder handle is negative
FETCL_ERR_NO_HND_FOUND	-4223	No Transponder handle found in Transponder handle list
FETCL_ERR_READER_HND_IS_NEGATIVE	-4226	The transferred Reader handle is negative
FETCL_ERR_THREAD_NOT_CREATED	-4227	The APDU thread could not be started
FETCL_ERR_UNKNOWN_PARAMETER	-4250	Transfer parameter is unknown
FETCL_ERR_PARAMETER_OUT_OF_RANGE	-4251	Transfer parameter too large or too small
FETCL_ERR_UNKNOWN_ERRORCODE	-4253	Unknown error code
FETCL_ERR_UNDERSIZED_RESPONSE_BUFFER	-4257	The response buffer is too small
FETCL_INVALID_ACKNOWLEDGEMENT_LENGTH	-4273	Too less response data from reader after APDU transmission
FETCL_LIST_COMPLETE_FAILURE	-4274	Internal error in front of APDU transmission
FETCL_INCOMPLETE_RESPONSE	-4275	Receive procedure interrupt causes too less response data
FETCL_INVALID_PROTOCOL	-4276	Unknown command or unvalid protocol data
FETCL_INVALID_TRANSMISSION	-4277	Internal error in FEISC library in front of transmission or invalid reader status after transmission.

6.2. List of constants for the FEISC_EVENT_INIT structure

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

Constants	Value	Use	Description
FETCL_THREAD_ID	1	uiFlag	Event flagging with thread message
FETCL_WND_HWND	2	uiFlag	Event flagging with window message
FETCL_CALLBACK	3	uiFlag	Event flagging with callback function

6.3. History

V2.01.00

- Improved thread safeness
- Bugfix for serial port connected Reader: calculation of timeout and assignment of PSTAT fixed

- Windows / Windows CE:
  1. Migration of the development environment 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

V2.00.00

- 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 _FETCL_EVENT_INIT*** concerning the calling convention. Thus, this version of FETCL is not compatible with the previous version and with applications compiled against the previous version of FETCL. Code modifications are not necessary, but re-compilation of applications is mandatory.

V1.00.06

- First Linux Release

V1.00.05

- First Linux Release

V1.00.00

- Extension in structure **FETCL_EVENT_INIT**: new element pAny and new callback function cbFct2

V0.05.03

- Modification in **FETCL_Apdu** for blocking behaviour

- Modification of the return value of **FETCL_GetRespoonseData**
- Remove of the function **FETCL_GetResponseDataLength**
- Modification for WTXM handling for serial communication

V0.05.00

- This is the first beta version.