RFID1356*i*™ SDK

Software Developer's Kit

For use with the
RFID1356i ™ Contactless Read/Write Reader

RF IDEAS INC.

www.RFIDeas.com

LICENSE AGREEMENT

End-User License Agreement for RF IDeas™ SOFTWARE and HARDWARE – RFID1356, RFID1356i, RF IDeas' AIR ID® , AIR ID® LT, pcProx USB, and pcProx® Contactless and Proximity Activated Readers, Software Developer's Kit, and Proximity Reader Protocol(s).

IMPORTANT-READ CAREFULLY: This End-User License Agreement ("EULA") is a legal agreement between you (either an individual or a single entity) and the manufacturer RF IDeas ("Manufacturer") with which you acquired the RF IDeas software and hardware product(s) identified above ("PRODUCT"). The PRODUCT includes the RFID1356, pcProx Base Unit, AIR ID Badge, AIR ID Base Unit or AIR ID LT Badge, AIR ID LT Base, computer software, including DLL and OCX files, the associated media, any printed materials, and any "online" or electronic documentation. By installing, copying or otherwise using the PRODUCT, you agree to be bound by the terms of this EULA. The SOFTWARE PORTION OF THE PRODUCT includes the computer software, the associated media, any printed materials, and any "online" or electronic documentation. By installing, copying or otherwise using the PRODUCT, you agree to be bound by the terms of this EULA. If you do not agree to the terms of this EULA, RF IDeas is unwilling to license the PRODUCT to you. In such event, you may not use or copy the SOFTWARE PORTION OF THE PRODUCT, and you should promptly contact the vendor you obtained this PRODUCT from for instructions on return of the unused product(s) for a refund.

The products described in this publication are intended for consumer applications. AIR ID and AIR ID LT operate on shared radio channels. Radio interference can occur in any place at any time, and thus the communications link may not be absolutely reliable. AIR ID and AIR ID LT must be used so that a loss of communications due to radio interference or otherwise will not endanger either people or property, and will not cause the loss of valuable data. RF IDeas assumes no liability for the performance of product. **RF IDeas products are not suitable for use in life-support applications, biological hazard applications, nuclear control applications, or radioactive areas. None of these products or components, software or hardware, are intended for applications that provide life support or any critical function necessary for the support of protection of life, property or business interests.** The user assumes responsibility for the use of any of these products in any such application. RF IDeas, Inc. shall not be liable for losses due to failure of any of these products, or components of these products, beyond the RF IDeas commercial warranty, limited to the original purchase price.

SOFTWARE PRODUCT LICENSE

The PRODUCT is protected by copyright laws and international copyright treaties, as well as other intellectual property laws and treaties. The SOFTWARE PORTION OF THE PRODUCT is licensed, not sold.

1. GRANT OF LICENSE. This EULA grants you the following rights:

* Software. You may install and use one copy of the SOFTWARE PORTION OF THE PRODUCT on the COMPUTER.
* Network Services. If the SOFTWARE PORTION OF THE PRODUCT includes functionality that enables the COMPUTER to act as a network server, any number of computers or workstations may access or otherwise utilize the basic network services of that server. The basic network services are more fully described in the printed materials accompanying the SOFTWARE PORTION OF THE PRODUCT.

Storage/Network Use. You may also store or install a copy of the computer SOFTWARE PORTION OF THE PRODUCT on the COMPUTER to allow your other computers to use the SOFTWARE PORTION OF THE PRODUCT over an internal network, and distribute the SOFTWARE PORTION OF THE PRODUCT to your other computers over an internal network. However, you must acquire and dedicate a license for the SOFTWARE PORTION OF THE PRODUCT for each computer on which the SOFTWARE PORTION OF THE PRODUCT is used or to which it is distributed. A license for the SOFTWARE PORTION OF THE PRODUCT may not be shared or used concurrently on different computers.

1.1 General License Grant RF IDeas grants to you as an individual, a personal, nonexclusive license to make and use copies of the SOFTWARE PRODUCT for the sole purposes of designing, developing, and testing your software product(s) that are designed to operate in conjunction with any RF IDeas designed proximity reader product. You may install copies of the SOFTWARE PRODUCT on an unlimited number of computers provided that you are the only individual using the SOFTWARE PRODUCT. If you are an entity, RF IDeas grants you the right to designate one individual within your organization to have the sole right to use the SOFTWARE PRODUCT in the manner provided above.

1.2 Documentation. This EULA grants you, as an individual, a personal, nonexclusive license to make and use an unlimited number of copies of any documentation, provided that such copies shall be used only for personal purposes and are not to be republished or distributed (either in hard copy or electronic form) beyond the user's premises and with the following exception: you may use documentation identified in the SOFTWARE PRODUCT as the file format specification for RF IDeas' proximity readers solely in connection with your development of software product(s) or an integrated work or product suite whose components include one or more general purpose software products. Note:

1.3 Storage/Network Use. You may also store or install a copy of the SOFTWARE PRODUCT on a storage device, such as a network server, used only to install or run the SOFTWARE PRODUCT on computers used by a licensed end user in accordance with Section 1.1. A single license for the SOFTWARE PRODUCT may not be shared or used concurrently by other end users.

1.4 Sample Code. RF IDeas grants you the right to use and modify the source code version of those portions of the SOFTWARE PRODUCT identified as "Samples in the SOFTWARE PRODUCT ("Sample Code") for the sole purposes of designing, developing, and testing your software product(s), and to reproduce and distribute the Sample Code, along with any modifications thereof, only in object code form.

2. DESCRIPTION OF OTHER RIGHTS AND LIMITATIONS.

* Limitations on Reverse Engineering, Decompilation and Disassembly. You may not reverse engineer, decompile, or disassemble the PRODUCT, except and only to the extent that such activity is expressly permitted by applicable law notwithstanding this limitation.

* You may not reproduce or otherwise emulate, in whole or in part, any form the protocol(s) defined within this PRODUCT for use without a RF IDeas PRODUCT.
* Redistributable Code. If you are authorized and choose to redistribute Sample Code (“Redistributables”) as described in Section 1.4, you agree to: (a) distribute the Redistributables in object code only in conjunction with and as a part of a software application product developed by you using the product accompanying this EULA that adds significant and primary functionality to the SOFTWARE PRODUCT ("Licensed Product"); (b) not use RF IDeas' name, logo, or trademarks to market the Licensed Product; (c) include a valid copyright notice on the Licensed Product; (d) indemnify, hold harmless, and defend RF IDeas from and against any claims or lawsuits, including attorney's fees, that arise or result from the use or distribution of the Licensed Product; (e) otherwise comply with the terms of this EULA; and (g) agree that RF IDeas reserves all rights not expressly granted. You also agree not to permit further distribution of the Redistributables by your end users except: (1) you may permit further redistribution of the Redistributables by your distributors to your end-user customers if your distributors only distribute the Redistributables in conjunction with, and as part of, the Licensed Product and you and your distributors comply with all other terms of this EULA; and (2) in the manner described in Section 1.4.
* Separation of Components. The PRODUCT is licensed as a single product. Its component parts may not be separated for use on more than one computer.
* Single COMPUTER. The PRODUCT is licensed with the COMPUTER as a single integrated product. The PRODUCT may only be used with the COMPUTER.
* Rental. You may not rent or lease the PRODUCT without permission from RF IDeas.
* Software Transfer. You may permanently transfer all of your rights under this EULA only as part of a sale or transfer of the COMPUTER, provided you retain no copies, you transfer all of the PRODUCT (including all component parts, the media and printed materials, any upgrades, this EULA and, if applicable, the Certificate(s) of Authenticity), AND the recipient agrees to the terms of this EULA. If the PRODUCT is an upgrade, any transfer must include all prior versions of the PRODUCT.
* Termination. Without prejudice to any other rights, RF IDeas may terminate this EULA if you fail to comply with the terms and conditions of this EULA. In such event, you must destroy all copies of the SOFTWARE PORTION OF THE PRODUCT and all of its component parts.

3. UPGRADES. If the SOFTWARE PORTION OF THE PRODUCT is an upgrade from another product, whether from RF IDeas or another supplier, you may use or transfer the PRODUCT only in conjunction with that upgraded product, unless you destroy the upgraded product. If the SOFTWARE PORTION OF THE PRODUCT is an upgrade of a RF IDeas product, you now may use that upgraded product only in accordance with this EULA. If the SOFTWARE PORTION OF THE PRODUCT is an upgrade of a component of a package of software programs which you licensed as a single product, the SOFTWARE PORTION OF THE PRODUCT may be used and transferred only as part of that single product package and may not be separated for use on more than one computer.

4. OEM COPYRIGHT. All title and copyrights in and to the PRODUCT (including but not limited to any images, photographs, animations, video, audio, music, text and "applets," incorporated into the PRODUCT), the accompanying printed materials, and any copies of the SOFTWARE PORTION OF THE PRODUCT, are owned by RF IDeas or its suppliers. The PRODUCT and SOFTWARE PORTION OF THE PRODUCT is protected by copyright laws and international treaty provisions. You may not copy the printed materials accompanying the PRODUCT.

5. DUAL-MEDIA SOFTWARE. You may receive the SOFTWARE PORTION OF THE PRODUCT in more than one medium. Regardless of the type or size of medium you receive, you may use only one medium that is appropriate for your single computer. You may not use or install the other medium on another computer. You may not loan, rent, lease, or otherwise transfer the other medium to another user, except as part of the permanent transfer (as provided above) of the SOFTWARE PORTION OF THE PRODUCT.

FOR THE LIMITED WARRANTIES AND SPECIAL PROVISIONS PERTAINING TO YOUR PARTICULAR JURISDICTION, PLEASE REFER TO YOUR WARRANTY BOOKLET INCLUDED WITH THIS PACKAGE OR PROVIDED WITH THE SOFTWARE PRODUCT PRINTED MATERIALS.

Limited Warranty

RF IDeas, Inc. warrants to the original buyer of this product, that the hardware and related disk(s) are free of defects in material and workmanship for a period of one year from date of purchase from RF IDeas or from an authorized RF IDeas dealer. Should the RF IDeas products fail to be in good working order at any time during the one-year period, RF IDeas will, at its option, repair or replace the product at no additional charge, provided that the product has not been abused, misused, repaired or modified. This warranty shall be limited to repair or replacement and in no event shall RF IDeas be liable for any loss of profit or any commercial or other damages, including but not limited to special, incidental, consequential or other similar claims.

No dealer, distributor, company, or person has been authorized to change or add to the terms of this agreement, and RF IDeas will not be bound by any representation to the contrary. RF IDeas SPECIFICALLY DISCLAIMS ALL OTHER WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS OF PURPOSE. Since some states do not allow such exclusion of limitation of incidental or consequential damages for consumer products, check the statute of the state in which your business resides. This warranty gives you the specific legal rights in addition to any rights that you have under the laws of the state in which your business resides or operates.

Returns

RF IDeas products which require Limited Warranty service during the warranty period shall be delivered to the nearest authorized dealer or sent directly to RF IDeas at the address below with proof of purchase and a Return Materials Authorization (RMA) Number provided by RF IDeas technical support Dept. Replacement parts or complete boards become the property of RF IDeas.

If the returned board or unit is sent by mail, the purchaser agrees to pre-pay the shipping charges and insure the board or unit or assume the risk of loss or damage which may occur in transit. The purchaser is expected to employ a container equivalent to the original packaging.

Copyright

Trademarks

All RF IDeas products are trademarks of RF IDeas, Inc. All other product names or names are trademarks or registered trademarks of their respective holders.

Disclaimer

This Reference Guide is printed in the U.S.A. Any resemblance mentioned in the Reference Guide to persons living or dead, or to actual corporations or products is purely coincidental. RF IDeas believes that the information contained in this manual is correct. However, RF IDeas does not assume any responsibility for the accuracy of the content of this User Manual, nor for any patent infringements or other rights of third parties. RF IDeas reserves the right to make any modifications in either product or the manual without giving prior written notification.

*Congratulations on the purchase of your RFID1356i Software Developer's Kit. We at **RF IDeas** hope you enjoy using your new Smart Card Contactless Identification System as much as we enjoyed creating and developing it! Please share your comments and suggestions!*

Thank you,
*The Staff at **RF IDeas***

Need Assistance?

Call: 847-870-1723
Fax: 847-483-1129
E-mail: TechSupp@RFIDeas.com

Mail to:

RF IDeas Inc.
4238B Arlington Heights Rd.
#244
Arlington Heights, IL 60004

FCC Compliance Statements

AIR ID LT Badge - FCC ID M9MMU100

This device complies with Part 15 of the FCC rules. Operation is subject to the following two conditions: (1) This device may not cause harmful interference, and (2) This device must accept any interference that may cause undesired operation.

AIR ID LT Base Unit - FCC ID M9MBU100

This device complies with Part 15 of the FCC rules. Operation is subject to the following two conditions: (1) This device may not cause harmful interface, and (2) This device must accept any interference that may cause undesired operation.

pcProx Base Unit FCC ID

HID version M9MBUPCPROXH100,
AWID version M9MBUPCPROXA100, or
Motorola version M9MBUPCPROXM100
Casi-Rusco version M9MBUPCPROXC100

pcProx USB Base Unit FCC ID

HID version M9MPCPROXHUSB100,
Motorola version M9MPCPROXMUSB100
Casi-Rusco version M9MPCPROXCUSB100

RFID1356i Base Unit FCC ID

iCLASS version M9MRFID1356IS100,

This device complies with Part 15 of the FCC rules. Operation is subject to the following two conditions: (1) This device may not cause harmful interface, and (2) This device must accept any interference that may cause undesired operation.

Note: Changes not expressly approved by RF IDeas could void the user's authority to operate the equipment.

Table of Contents

Overview

This document will guide the developer through the various documents, specifications, and sample code contents of the ***RFID1356i*** ***TM*** ***Contactless Read/Write SDK*** (Software Developer's Kit). The ***RFID1356i*** serial port contactless read/write reader is fully supported in this SDK. If you are interested in the USB port SDK, please call RF IDeas for more information.

The ***RFID1356i*** uses drivers supplied with the operating system, however there is a DLL library required that is supplied by RF IDeas. The SDK makes use of this DLL, **"iClassAPI.dll"**, which is required for use with the device.

The "iClassAPI.dll" DLL manages all operations necessary to

1. Read and Write device configuration items and retrieve a card ID when a card is presented to the reader,
2. It also provides a means to program HID iClass smart cards with application data and read it back, and
3. To change master keys in the reader and cards.

There is a supplemental **RFIDProxFilter.ocx** ActiveX control for additional ID data filtering. The OCX assists the developer with interpreting the data into facility and ID codes.

The reader's configuration items are grouped according to their general function and are handled through structures defined in this document.

Before the configuration items are accessible, the DLL must be instructed to 'Connect' to the device and told to 'Read' all configuration items. Once the items have been read into local storage, the user may then access them through the four groups of 'Get' and 'Set' functions, using the defined structures. The configuration items are sent back to the device for permanent storage only when the 'Write' function is called.

The device may be reset to factory default values by calling 'ResetFactoryDflts'. This is a fast method to bring the device to a known state. Before returning to the caller, this function calls 'ReadCfg' to load the restored configuration items into local storage.

To read an ID from the device, it is only necessary to 'Connect' to the device. No configuration is necessary if the user is confident that the device is already configured appropriately.

The 'GetActiveID' function may be called at any time. It is independent of configuration activity except that it should not be called directly (within 1 second) after calling the 'Write' function.

When the user application is finished, the 'Disconnect' function should be called before the DLL is discarded. This is mostly in the spirit of 'Good Programming Practice' as both the DLL and the operating system will [probably] close the open handle to the device upon exit.

The 'GetLibVersion' function does not require that a 'USBConnect' be performed first. It only requires that the Library DLL was successfully loaded into memory.

All exported functions are of type 'APIENTRY' which specifies the Pascal Calling Convention. This makes the Library compatible with both 'C' (when APIENTRY is explicitly used in the declaration) and Basic (which assumes APIENTRY) Applications.

The configuration items are grouped according to their general function and are handled through structures defined in iClassAPI.h.

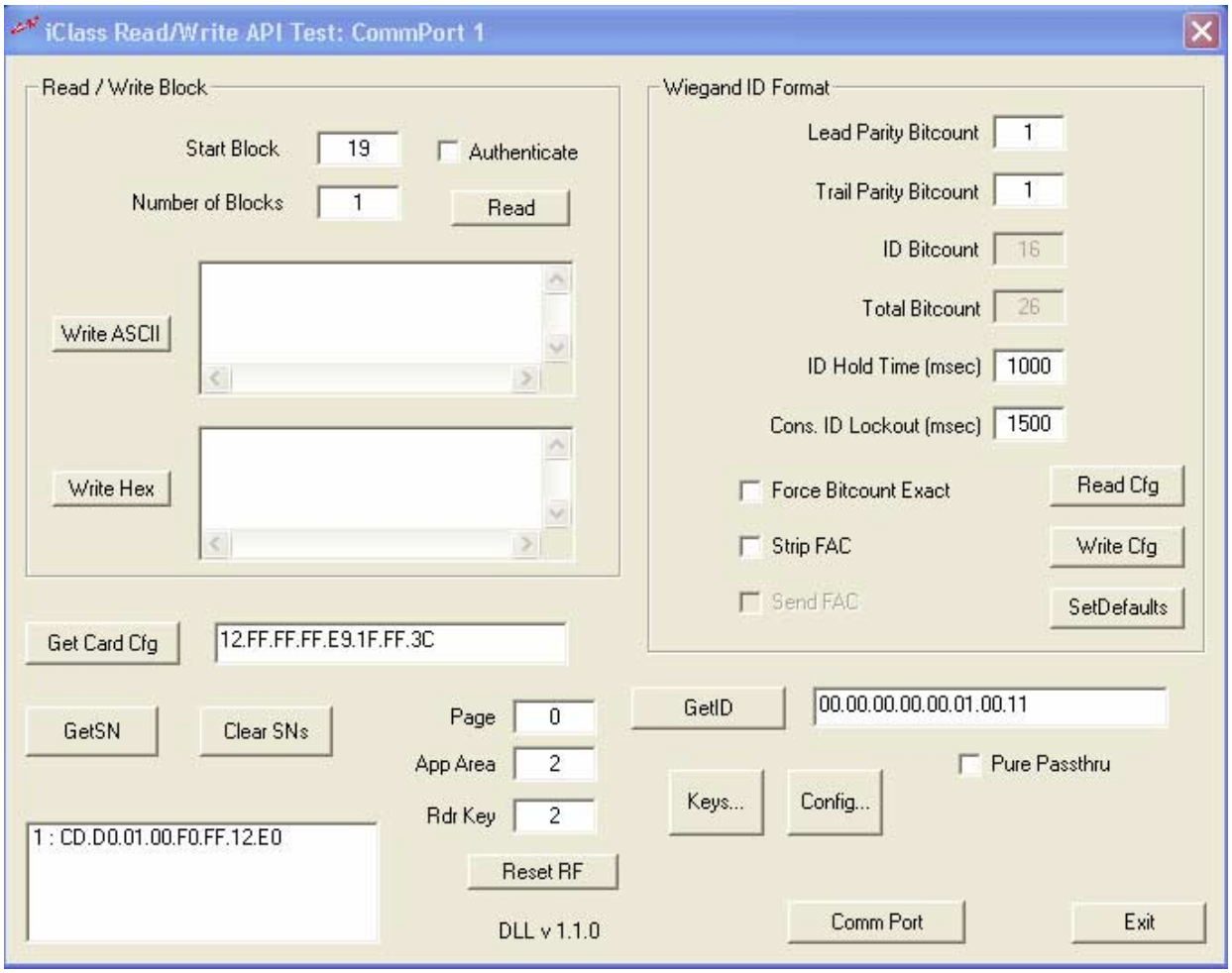
RFID1356*i* API Release Information

Version 1.1.0 - April. 19, 2002
Further pre-release refinement.
Version 1.0.0 - April. 12, 2002

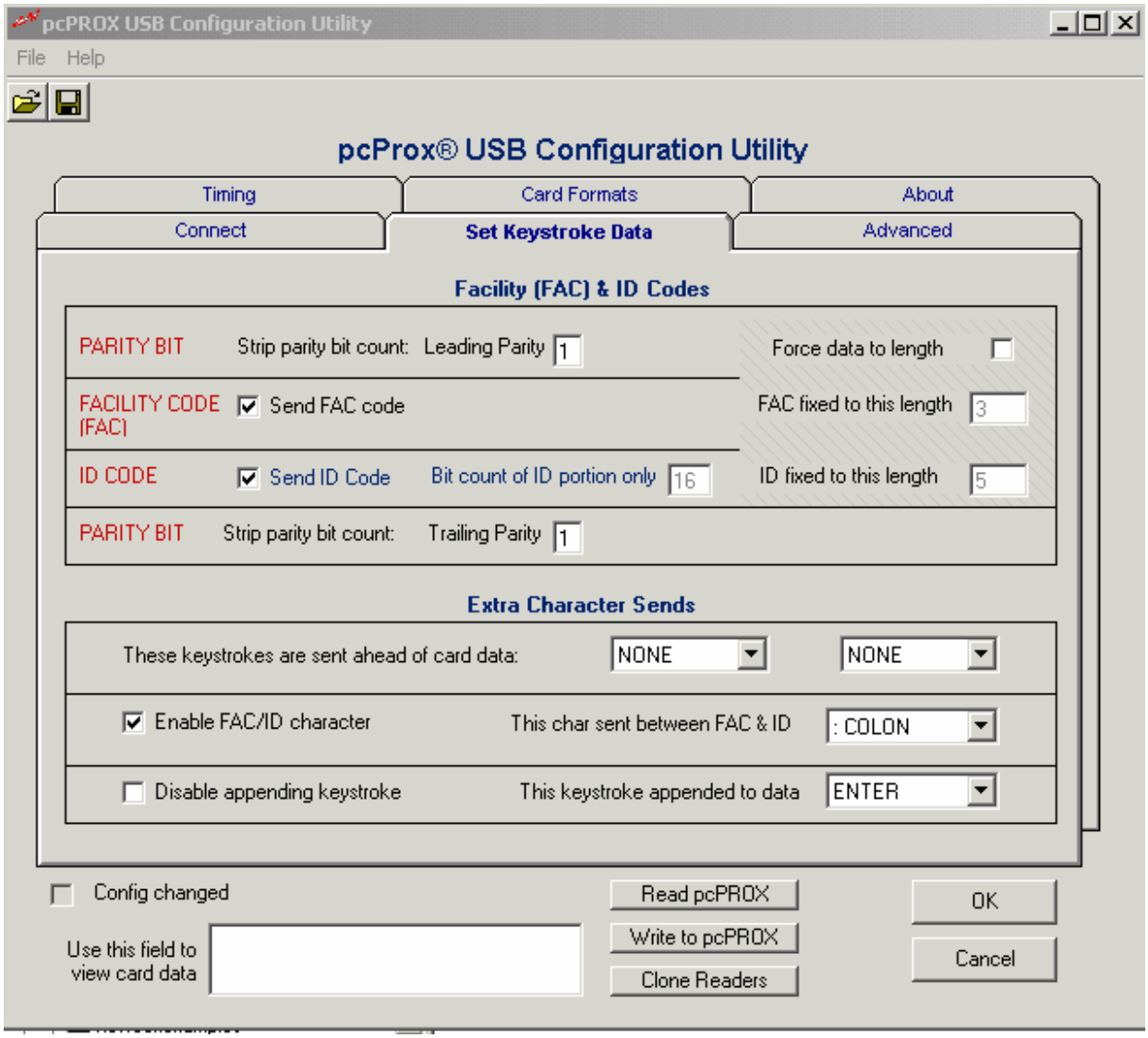
This document describes the interface to the RFID1356*i* "iClassAPI.dll" Dynamic Link Library. This library provides a RS-232c serial interface to the pcProx1356 'device' which is equipped with a HID iClass Reader capable of reading and writing the HID iClass 2Kbit and 16Kbit cards and reading the serial number from a MIFARE card.

The configuration of the RFID1356*i* generally provides formatting of the iClass card 'ID' and timing parameters associated with the retrieval of the ID. The ic_xxxx() routines provide an interface to the on-board HID iClass reader (and cards) and are, for the most part, independent if the card ID formatting and timing.

Disk Contents



C++ Command Exerciser Application



Summary

BOOL APIENTRY GetLibVersion(short* piVerMaj, short* piVerMin, short* piVerDev);

BOOL APIENTRY USBConnect(long* plDID);

BOOL APIENTRY USBDisconnect(void);

long APIENTRY GetLastLibErr(void);

BOOL APIENTRY ReadCfg(void);

BOOL APIENTRY WriteCfg(void);

BOOL APIENTRY ResetFactoryDflts(void);

BOOL APIENTRY GetFlags(tsCfgFlags* psCfgFlgs);

BOOL APIENTRY SetFlags(tsCfgFlags* psCfgFlgs);

BOOL APIENTRY GetIDBitCnts(tsIDBitCnts* psIDBitCnts);

BOOL APIENTRY SetIDBitCnts(tsIDBitCnts* psIDBitCnts);

BOOL APIENTRY GetIDDispParms(tsIDDispParms* psIDDispParms);

BOOL APIENTRY SetIDDispParms(tsIDDispParms* psIDDispParms);

BOOL APIENTRY GetTimeParms(tsTimeParms* psTimeParms);

BOOL APIENTRY SetTimeParms(tsTimeParms* psTimeParms);

BOOL APIENTRY GetFlags2(tsCfgFlags2* psCfgFlgs2);

BOOL APIENTRY SetFlags2(tsCfgFlags2* psCfgFlgs2);

BOOL APIENTRY GetIDDispParms2(tsIDDispParms2* psIDDispParms2);

BOOL APIENTRY SetIDDispParms2(tsIDDispParms2* psIDDispParms2);

short APIENTRY GetActiveID(BYTE* pBuf, short wBufMaxSz);

BOOL APIENTRY GetLEDCtrl(tsLEDCtrl *psLEDCtrl);

BOOL APIENTRY SetLEDCtrl(tsLEDCtrl *psLEDCtrl);

short APIENTRY GetDevCnt(void);

short APIENTRY GetActDev(void);

BOOL APIENTRY SetActDev(short iNdx);

short APIENTRY GetLUID(void);

BOOL APIENTRY SetLUID(short LUID);

short APIENTRY GetDID(void);

Function Details

```
BSHRT APIENTRY GetLibVersion( short* piVerMaj, short* piVerMin, short*
piVerDev );
```

Parameters:

piVerMaj – pointer (reference) to integer to receive the Major version
piVerMin – pointer (reference) to integer to receive the Minor version
piVerDev – pointer (reference) to integer to receive the Development version

Returns non-zero always.
The intended interpretation of the Version is ‘v VerMaj.VerMin.VerDev’.

```
[future release] BSHRT APIENTRY USBConnect( long* plDID );
```

```
[future release] BSHRT APIENTRY USBDisconnect( void );
```

```
long APIENTRY GetLastLibErr( void );
```

Parameters: [none]
Returns the last Library Error code (see Library Error Codes).
The Last Error Code is valid until another Library call is made. This call is the only Library call that does not reset the Last Library Error Code.

```
BSHRT APIENTRY ReadCfg( void );
```
Parameters: [none]
Returns non-zero on Success, zero otherwise.
A call to this function pulls all configuration information from the current active device to local Library memory space where it may be manipulated by the user application using the Getxxx() and Setxxx() functions.

```
BSHRT APIENTRY WriteCfg( void );
```
Parameters: [none]
Returns non-zero on Success, zero otherwise.
A call to this function writes all configuration information in local Library memory space to the current active device for non-volatile storage. Any changed parameters take effect immediately after the write. The actual write internal to the device is not done until all critical pending actions are complete. This may take up to 1 second.

```
BSHRT APIENTRY ResetFactoryDflts( void );
```
Parameters: [none]
Returns non-zero on Success, zero otherwise.
A call to this function instructs the current active device to set all configuration information to the Factory Default values. It should be treated as a WriteCfg() call except local values are not used in the transaction. Before returning to the caller, this function calls ReadCfg() to reload the configuration information (which may have changed) so on failure, the GetLastLibErr() return will reflect either the errored call to set the defaults or an error condition encountered in the ReadCfg() call.

```
BSHRT APIENTRY GetFlags( tsCfgFlags* psCfgFlgs );
Parameters:
   psCfgFlgs – pointer to structure to receive Configuration Flags info.
Returns non-zero on Success, zero otherwise.

BSHRT APIENTRY SetFlags( tsCfgFlags* psCfgFlgs );
Parameters:
   psCfgFlgs – pointer to structure containing new Configuration Flags info.
Returns non-zero on Success, zero otherwise.
This call does NOT write the configuration items to the device, just to local
Library storage.

BSHRT APIENTRY GetIDBitCnts( tsIDBitCnts* psIDBitCnts );
Parameters:
   psIDBitCnts – pointer to structure to receive Bit Count info.
Returns non-zero on Success, zero otherwise.

BSHRT APIENTRY SetIDBitCnts( tsIDBitCnts* psIDBitCnts );
Parameters:
   psIDBitCnts – pointer to structure containing new Bit Count info.
Returns non-zero on Success, zero otherwise.
This call does NOT write the configuration items to the device, just to local
Library storage.

BSHRT APIENTRY GetIDDispParms( tsIDDispParms* psIDDispParms );
Parameters:
   psIDDispParms – pointer to structure to receive ID Display info.
Returns non-zero on Success, zero otherwise.

BSHRT APIENTRY SetIDDispParms( tsIDDispParms* psIDDispParms );
Parameters:
   psIDDispParms – pointer to structure containing new ID Display info.
Returns non-zero on Success, zero otherwise.
This call does NOT write the configuration items to the device, just to local
Library storage.

BSHRT APIENTRY GetTimeParms( tsTimeParms* psTimeParms );
Parameters:
   psTimeParms – pointer to structure to receive Timing info.
Returns non-zero on Success, zero otherwise.

BSHRT APIENTRY SetTimeParms( tsTimeParms* psTimeParms );
Parameters:
   psTimeParms – pointer to structure containing new Timing info.
Returns non-zero on Success, zero otherwise.
This call does NOT write the configuration items to the device, just to local
Library storage.
```

```
BSHRT APIENTRY GetFlags2( tsCfgFlags2* psCfgFlgs2 );
```
Parameters:
   psCfgFlgs – pointer to structure to receive Configuration Flags (2) info.
Returns non-zero on Success, zero otherwise.

```
BSHRT APIENTRY SetFlags2( tsCfgFlags2* psCfgFlgs2 );
```
Parameters:
   psCfgFlgs – pointer to structure containing new Configuration Flags (2) info.
Returns non-zero on Success, zero otherwise.
This call does NOT write the configuration items to the device, just to local Library storage.

```
BSHRT APIENTRY GetIDDispParms2( tsIDDispParms2* psIDDispParms2 );
```
Parameters:
   psIDDispParms – pointer to structure to receive ID Display (2) info.
Returns non-zero on Success, zero otherwise.

```
BSHRT APIENTRY SetIDDispParms2( tsIDDispParms2* psIDDispParms2 );
```
Parameters:
   psIDDispParms – pointer to structure containing new ID Display (2) info.
Returns non-zero on Success, zero otherwise.
This call does NOT write the configuration items to the device, just to local Library storage.

```
BSHRT APIENTRY GetLEDCtrl( tsLEDCtrl *psLEDCtrl );
```
Parameters:
   psLEDCtrl – pointer to structure to receive LED control info.
Returns non-zero on Success, zero otherwise.

```
BSHRT APIENTRY SetLEDCtrl( tsLEDCtrl *psLEDCtrl );
```
Parameters:
   psLEDCtrl – pointer to structure containing new LED control info.
Returns non-zero on Success, zero otherwise.
This call does NOT write the configuration items to the device, just to local Library storage.

```
short APIENTRY GetActiveID( BYTE* pBuf, short wBufMaxSz );
```
Parameters:
   pBuf – pointer (reference) to character buffer to receive the ID of the card currently within reader range.
   wBufMaxSz – specifies the size of the character buffer. A max of 8 will be used so this is useful only to limit the buffer transfer to < 8 characters. Values > 8 will still only transfer 8 characters with no error.
Returns the number of bits received from the reader representing the ID.
The return count represents how many bits in the buffer are valid ID bits, it does NOT include the parity bits that may have been stripped from the ID through the use of the Leading and/or Trailing parity bit counts. A return of zero means that there is either no card within range or that there was another error encountered. GetLastLibErr() may be used to differentiate between the two possibilities.

```
short APIENTRY GetDevCnt( void );
```
Parameters: [none]
Returns the number of pcProxUSB devices found on this machine in USBConnect().

```
short APIENTRY GetActDev( void );
```
Parameters: [none]
Returns the current Active Device, an index into a list, 0 – (GetDevCnt()-1).

```
BSHRT APIENTRY SetActDev( short iNdx );
```
Parameters:
   iNdx – selects a new Active Device for processing.
Returns non-zero on Success, zero otherwise.
This call does not require a 'Write' to be performed as it is only selecting the device to which we are communicating with thru the library, not changing any device parameters.

```
short APIENTRY GetLUID( void );
```
Parameters: [none]
Returns the Locally Unique ID of the current Active Device.

```
BSHRT APIENTRY SetLUID( short LUID );
```
Parameters:
   LUID – a user defined 16-bit ID to be associated with the current Active Device.
Returns non-zero on Success, zero otherwise.
This call does NOT write the configuration items to the device, just to local Library storage.

```
short APIENTRY GetDID( void );
```
Parameters: [none]
Returns the Firmware version (Device ID) of the current Active Device. A return of Zero indicates that the call failed.
This device ID may be, and probably is, the same for all of the enumerated devices.

[future release] BSHRT APIENTRY GetHIDGuid)(GUID *pGuid);

Parameters:

pGuid – pointer to a GUID structure that will receive the GUID of this particular interface class. It may be treated as an 8-byte array.

Returns non-zero on Success, zero otherwise.

This call is required if your App will register for Device Interface change notifications.

[future release] BSHRT APIENTRY ChkAddArrival(char *szName);

Parameters:

szName – The 'friendly' name of the device as reported in a device notification.

Returns non-zero if the device 'checks' as one of ours and is added, zero otherwise.

This call is used when your App receives a Device Interface change notification for a device that has been added. This is part of an application controlled mechanism to dynamically manage devices as they are added and removed.

[future release] BSHRT APIENTRY ChkDelArrival(char *szName);

Parameters:

szName – The 'friendly' name of the device as reported in a device notification.

Returns non-zero if the device 'checks' as one of ours and is found and deleted, zero otherwise.

This call is used when your App receives a Device Interface change notification for a device that has been added. This is part of an application controlled mechanism to dynamically manage devices as they are added and removed.

```
long APIENTRY GetLastICSWErr( long lBufSz, char *szErrBuf );
```
Parameters:
   lBufSz – specifies the maximum size in bytes of szErrBuf.
   szErrBuf – contains NULL terminated string describing the last Status Word.
Return Value:
Returns the HID iClass reader Status Word which defines the text in szErrBuf.

```
BSHRT APIENTRY CommCnct( tsCommParms* pCommParms );
```
Opens a Comm port for communication to the device. The sCommParms structure contains information on baud rate, parity, and number of stop bits. This should always be set to 57600 baud, no parity, and 1 stop bit (see structure definition).
Parameters:
   pCommParms – pointer to a sCommParms structure containing the comm data.
Return Value:
Returns non-zero if successful, zero on failure.

```
void APIENTRY CommDiscnct( void );
```
Closes the open HANDLE to the active Comm port.
Parameters: [none]
Return Value: [none]

```
BSHRT APIENTRY ic_Select_Current_Key( long lRKey );
```
Parameters:
   lRKey - Reader key location for card authentication, valid for locations 1 thru 7.
Return Value:
   Returns non-zero if successful, zero on failure.
   Call GetLastICSWErr() for more detail on failure.

```
BSHRT APIENTRY ic_Select_Card( long lCAA,
                               long lRKey,
                               long lCTypes,
                               long *pCType,
                               BYTE *pCSNBuf
                             );
```
Parameters:
   lCAA – Card App area (1 or 2) to authenticate against (0 for no authentication).
   lRKey – Reader key to use in authentication (if lCAA != 0).
   lCTypes – Bitmap of Card Types. [iClass == 2, MIFARE = 8, iClass OR MIFARE = 10]
   pCType – pointer to return variable for card type found.
   pCSNBuf – Return buffer for the globally unique Card Serial Number (CSN).
Return Value:
   Returns non-zero if successful (CSN valid), zero on failure.
   Call GetLastICSWErr() for more detail on failure.

```
BSHRT APIENTRY ic_Page_Select( BSHRT bAuth,
                               long lCAA,
                               long lCPage,
                               long lRKey,
                               BYTE *pRetBuf
                             );
```
Used with HID iClass 16Kbit / 16 Application Cards ONLY!
Parameters:
   bAuth – Boolean, if non-zero, authenticate w/ current reader key against wCAA.
   lCAA – Card Application area (1 or 2) to authenticate against (if bAuth TRUE).
   lCPage – Card Page to select (0 thru 7).
   lRKey – Reader key to use in authentication (if bAuth TRUE).
   pRetBuf – pointer to buffer to receive either the CSN or config block 1 data.
Return Value:
   Returns non-zero if successful, zero on failure.
   Call GetLastICSWErr() for more detail on failure.

```
BSHRT APIENTRY ic_Read( long lBlkLen, long lBlkAddr, BYTE *pRetBuf );
Parameters:
   lBlkLen – Number of Blocks to read (bytes = blocks x 8).
   lBlkAddr – Starting Block address.
   pRetBuf – pointer to buffer (size = lBlkLen x 8) to receive the card data.
Return Value:
   Returns non-zero if successful, zero on failure.
   Call GetLastICSWErr() for more detail on failure.

BSHRT APIENTRY ic_Write( long lBlkLen, long lBlkAddr, BYTE *pSndBuf );
Parameters:
   lBlkLen – Number of Blocks to write (bytes = blocks x 8).
   lBlkAddr – Starting Block address.
   pRetBuf – pointer to buffer (size = lBlkLen x 8) with the card data to
      write.
Return Value:
   Returns non-zero if successful, zero on failure.
   Call GetLastICSWErr() for more detail on failure.

BSHRT APIENTRY ic_Reset_RF( void );
Reset the RF field of the reader/writer. Should be done after the writing of
Configuration Block 1.
Parameters: [none]
Return Value:
   Returns non-zero if successful, zero on failure.
   Call GetLastICSWErr() for more detail on failure.
```

```
BSHRT APIENTRY ic_LoadKeyInRdr( long lRKey, BYTE *pNewKey );
This function will perform necessary steps to load a new key into iClass
reader EEPROM at the specified location.  A call to ic_Select_Current_Key()
is then used to select this key as the authentication key against card reads
and writes.
Parameters:
   lRKey – Reader key location, valid for locations 1 thru 7.
   pNewKey – pointer to the new 8-byte key to be loaded.
Return Value:
   Returns non-zero if successful, zero on failure.
   Call GetLastICSWErr() for more detail on failure.
```

```
BSHRT APIENTRY ic_WriteKeyToCard( long lRKey,
                                  long lCAA,
                                  long lCPg,
                                  long lSwpRKey,
                                  BSHRT bCrdPrg,
                                  BYTE *pNewKey
                                );
This function will perform necessary steps to write a new key into an iClass
card EEPROM at the specified location.  A call to ic_LoadKeyInRdr() is then
used to store this key into a reader EEPROM location.
Parameters:
   lRKey – Reader key location of current authentication key that may be used in ic_Write() to the card
      whose key will be changed.
   lCAA – Card Application Area key location to be written (1 or 2).
   lCPg – Card Page to be written (0-7).  Applies to 16Kbit / 16 App cards
      only.  Set to Zero for other cards.
   lSwpRKey – Unused reader key location for temporary use (commonly 3-7).
   bCrdPrg – Boolean, is card already programmed.  This is obtained from bit
      7 of byte 7 of card block 1 using ic_Read(), and is TRUE if this bit
      equals zero.
   pNewKey – pointer to the new 8-byte key to be loaded.
Return Value:
   Returns non-zero if successful, zero on failure.
Call GetLastICSWErr
```

Factory Default Settings

sCfgFlags:

All Flags are set to Zero

sIDBitCnts:

iLeadParityBitCnt = 1
iTrailParityBitCnt = 1
iIDBitCnt = 16
iTotalBitCnt = 26

sIDDispParms:

iFACIDDelim = 0x33 + 0x80 (shifted Semicolon = Colon) = 0xB3
iELDelim = 0x28 (ENTER – Carriage Return)

iIDDispLen = 5
iFACDispLen = 3

sTimeParms:
iBitStrmTO = 80 msec
iIDHoldTO = 1500 msec
iIDLockOutTm = 2500 msec
iUSBKeyPrsTm = 20 msec
iUSBKeyRlsTm = 20 msec

sCfgFlags2:
All Flags are set to Zero

sIDDispParms2:
iLeadChrCnt = 0
iLeadChr0 = 0
iLeadChr1 = 0
iLeadChr2 = 0

‘C’ Reference

See file: pcProxUSBAPI.h for Structure and Function Prototypes.

```
//----------------------
// Typical ‘C’ Program Flow:
//----------------------
#include “pcProxUSBAPI.h”

// Static declarations of function pointers...
GetLibVersion fnGetLibVersion;
USBConnect fnUSBConnect;
ReadCfg fnReadCfg;
GetFlags fnGetFlags;
SetFlags fnSetFlags;
WriteCfg fnWriteCfg;
GetLastLibErr fnGetLastLibErr;
GetActiveID fnGetActiveID;
USBDisconnect fnUSBDisconnect;
GetDevCnt fnGetDevCnt;
SetActDev fnSetActDev;
SetLUID fnSetLUID;

// The DLL module handle...
HMODULE hWejAPIDLL;

//----------------------
BOOL InitUSBProxLib()
{
  hWejAPIDLL = (HMODULE)LoadLibrary("pcProxUSBAPI.DLL");
  if( NULL == hWejAPIDLL ) return FALSE;

  fnGetLibVersion = (GetLibVersion)GetProcAddress( hWejAPIDLL, "GetLibVersion" );
  fnUSBConnect = (USBConnect)GetProcAddress( hWejAPIDLL, "USBConnect" );
  fnUSBDisconnect = (USBDisconnect)GetProcAddress( hWejAPIDLL, "USBDisconnect" );
  fnGetLastLibErr = (GetLastLibErr)GetProcAddress( hWejAPIDLL, "GetLastLibErr" );
  fnReadCfg = (ReadCfg)GetProcAddress( hWejAPIDLL, "ReadCfg" );
  fnWriteCfg = (WriteCfg)GetProcAddress( hWejAPIDLL, "WriteCfg" );
  fnGetFlags = (GetFlags)GetProcAddress( hWejAPIDLL, "GetFlags" );
  fnSetFlags = (SetFlags)GetProcAddress( hWejAPIDLL, "SetFlags" );
  fnGetActiveID = (GetActiveID)GetProcAddress( hWejAPIDLL, "GetActiveID" );
  fnGetDevCnt = (GetDevCnt)GetProcAddress( hWejAPIDLL, "GetDevCnt" );
  fnSetActDev = (SetActDev)GetProcAddress( hWejAPIDLL, "SetActDev" );
  fnSetLUID = (SetLUID)GetProcAddress( hWejAPIDLL, "SetLUID" );

  if( (fnGetLibVersion == NULL) ||
      (fnUSBConnect == NULL) ||
      (fnUSBDisconnect == NULL) ||
      (fnGetLastLibErr == NULL) ||
      (fnReadCfg == NULL) ||
      (fnWriteCfg == NULL) ||
      (fnGetFlags == NULL) ||
      (fnSetFlags == NULL) ||
      (fnGetActiveID == NULL) ||
      (fnGetDevCnt == NULL) ||
      (fnSetActDev == NULL) ||
      (fnSetLUID == NULL)
    )
  {
    FreeLibrary( hWejAPIDLL );
    hWejAPIDLL = NULL;
    return FALSE;
  }

  return TRUE;
}
```

```
//----------------------
void main()
{
  short iVerMaj, iVerMin, iVerDev;
  BOOL brc;
  long lrc, lDeviceID;
  short iIDBitCnt, i, iDevCnt;
  tsCfgFlags devFlags;
  BYTE IDBuf[16];      // really only need 8 byte storage but what the hey

  if( !InitUSBProxLib() ) exit(1);

  brc = fnGetLibVersion( &iVerMaj, &iVerMin, &iVerDev );
  if( brc )
  {
    // display / squirrel away the Library version info...
  }

  brc = fnUSBConnect( &lDeviceID ); ' connect to the device
  if( !brc )
  {
    // Device is not on the USB or other scarry thoughts...
    FreeLibrary( hWejAPIDLL );
    exit(1); // or whatever user wants to do to terminate or wait around or whatever
  }
  // lDeviceID has ASCII-coded decimal USB Device version in lower 2 bytes

  iDevCnt = fnGetDevCnt();  // get number of devices we found in connection

  for( i = 0; i < iDevCnt; i++ )
  {
    fnSetActDev( i );  // make sure we know who we're talking to
    brc = fnReadCfg();               // pull ALL configurable items from device to local storage
    brc = fnGetFlags( &devFlags );    // pull local copy of configurable 'Flags'
    if( brc )
    {
      fnSetLUID( i );    // program the LUID for this device (this would be done just once
typically)
      devFlags.bHaltKBSnd = True     // make an adjustment
      brc = fnSetFlags( &devFlags );  // push local copy of configurable 'Flags'
      brc = fnWriteCfg();            // commit local changes to device non-volatile storage
    }
    else
    {
      lrc = fnGetLastLibErr();
      // then see what lrc tells yu
    }
  }

  iIDBitCnt = fnGetActiveID( IDBuf, 8 ); // read an active ID – ALL 8 Bytes
  if( iIDBitCnt )
  {
    // got a valid read in IDBuf...
    // IDBuf(0) has LSByte, IDBuf(7) has MSByte
  }

  brc = fnUSBDisconnect();
  FreeLibrary( hWejAPIDLL );
}
```

Basic Reference

See file: VB Project for example and pcProxUSBAPI.bas for Type and Function Declarations.

```
'---------------------
' Typical Basic Program Flow:
'---------------------
```

```
Sub Main()
  Dim iVerMaj, iVerMin, iVerDev As Integer
  Dim brc As Boolean
  Dim lrc, lDeviceID As Long
  Dim iIDBitCnt As Integer
  Dim devFlags As tsCfgFlags
  Dim IDBuf As tsIDBuf      ' 8-byte ID return buffer

  brc = GetLibVersion( iVerMaj, iVerMin, iVerDev )
  If brc <> False Then
    ' display / squirrel away the Library version info...
  End If

  brc = USBConnect( lDeviceID ) ' connect to the device
  If brc = False Then
    ' Device is not on the USB or other scarry thoughts...
    Exit ' or whatever user wants to do to terminate or wait around or whatever
  End If
 ' lDeviceID has ASCII-coded decimal USB Device version in lower 2 bytes

  brc = ReadCfg()               ' pull ALL configurable items from device to local storage
  brc = GetFlags( devFlags )    ' pull local copy of configurable 'Flags'
  If brc <> False Then
    devFlags.bHaltKBSnd = True    ' make an adjustment
    brc = SetFlags( devFlags )    ' push local copy of configurable 'Flags'
    brc = WriteCfg()              ' commit local changes to device non-volatile storage
  Else
    lrc = GetLastLibErr()
    ' then see what lrc tells yu
  End If

  iIDBitCnt = GetActiveID( IDBuf, 8 ) ' read an active ID – ALL 8 Bytes
  If iIDBitCnt <> 0 Then
    ' got a valid read in IDBuf...
    ' IDBuf.Byte0 has LSByte, IDBuf.Byte7 has MSByte
  End If

  brc = USBDisconnect()

End Sub
```

Library Error Codes

Returned from GetLastLibErr()

Public Interface Error bits:

USBConnect	0x0001xxxx
ReadCfg	0x0002xxxx
WriteCfg	0x0004xxxx
ResetFactoryDflts	0x0008xxxx
Get(Structure)	0x0010xxxx
NULL Pointe	0x00100001
ReadCfg not called	0x00100002
Set(Structure)	0x0020xxxx
NULL Pointer	0x00200001
GetActiveID	0x0100xxxx

where ‘xxxx’ represents Private Interface Error Bits.

Private Interface Error bits:

USBDeviceConnect 0x00xx, xx=
1:Couldn't open SETUPAPI.DLL
2:Unresolved SETUPAPI.DLL entry point
3:Couldn't open HID.DLL
4:Unresolved HID.DLL entry point
5:Unresolved DLL entry point
6:SetupDiGetClassDevs returned INVALID_HANDLE_VALUE
7:SetupDiEnumDeviceInterfaces failed or ran out of devices
8:SetupDiGetDeviceInterfaceDetail: ERROR_INSUFFICIENT_BUFFER != GetLastError()
9:Failed pDevIFDetail LocalAlloc
10:VendorID and/or ProductID not found
11:CreateFile failed
12:SetupDiGetDeviceInterfaceDetail returned 0
GetUSBDevFeatureRep 0x01xx, xx=
0:device not open
1:NULL module call reference
2:module call returned FALSE
SetUSBDevFeatureRep 0x02xx, xx=
0:device not open
1:NULL module call reference
2:module call returned FALSE
CheckUserFlags 0x100x, x= [will never fail]
CheckUserBitCnts 0x101x, x=
0:iLeadParityBitCnt > 15
1:iTrailParityBitCnt > 15
2:(iIDBitCnt < 1) OR (iIDBitCnt > 64)
3:(iTotalBitCnt < 26) OR (iTotalBitCnt > 64)
CheckUserDispParms 0x102x, x=
0:iFACIDDelim > 255
1:iELDelim > 255
2:iIDDispLen > 10
3:iFACDispLen > 10
CheckUserTimeParms 0x103x, x=
0:iBitStrmTO > 1020
1:iIDHoldTO > 12750
2:iIDLockOutTm > 12750
3:iUSBKeyPrsTm > 1020
4:iUSBKeyRlsTm > 1020
CheckUserFlags2 0x0000104x, x= [will never fail]
CheckUserDispParms2 0x0000105x, x=
0:iLeadChrCnt > 3

Error Recovery

The device not being plugged into the USB port is the normal cause of error encountered in the 'USBConnect' call. There is nothing that needs to be done (or that can be done) in the application, plugging the device in or un-plugging then re-plugging the device in should fix this problem.

Errors encountered in the 'ReadCfg', 'WriteCfg', 'SetFactoryDflts', and 'GetActiveID' calls are usually caused by the device being un-plugged after the call to 'USBConnect' is done. The 'handle' to the device becomes invalid in this case and the application should call 'USBDisconnect' and 'USBConnect' to get a new 'handle' (after the device is confirmed to be on the USB).

The 'Getxxx' calls will fail if the user does not first call 'ReadCfg'.

The 'Setxxx' calls will fail if any of the parameters are outside of acceptable values. See the 'Private Interface Error Bits' in the 'Library Error Codes' section for details on acceptable values.

USB Keys as Used by the Device

The iFACIDDelim and iELDelim are set to one of the following sUSBKey::ucKeyCode values.
The USB key codes do not have the ‘shifted’ state as part of the key code. Rather, this information is included in a ‘modifier’ byte that travels with the key code. Since the USB key codes (of interest) are all < 128, the Library uses the high bit of the key code byte to extend the definitions to ‘shifted’ keys.

```
//---------------------------------------------------------------------
// The USB Key translater structure and static array...
//---------------------------------------------------------------------
typedef struct sUSBKey {
  WORD ucKeyCode; // this code is used for parameters involving FAC-ID & EndLn Delims
  LPSTR pTxt;     // a visual text helper as to what they represent
} tsUSBKey;

static tsUSBKey USBKeyTable[] = {
  {0x00, "[NONE]"},
  //...
  {0x04, "a"},
  {0x05, "b"},
  {0x06, "c"},
  {0x07, "d"},
  {0x08, "e"},
  {0x09, "f"},
  {0x0A, "g"},
  {0x0B, "h"},
  {0x0C, "i"},
  {0x0D, "j"},
  {0x0E, "k"},
  {0x0F, "l"},
  {0x10, "m"},
  {0x11, "n"},
  {0x12, "o"},
  {0x13, "p"},
  {0x14, "q"},
  {0x15, "r"},
  {0x16, "s"},
  {0x17, "t"},
  {0x18, "u"},
  {0x19, "v"},
  {0x1A, "w"},
  {0x1B, "x"},
  {0x1C, "y"},
  {0x1D, "z"},
  {0x04 | 0x80, "A"},
  {0x05 | 0x80, "B"},
  {0x06 | 0x80, "C"},
  {0x07 | 0x80, "D"},
  {0x08 | 0x80, "E"},
  {0x09 | 0x80, "F"},
  {0x0A | 0x80, "G"},
  {0x0B | 0x80, "H"},
  {0x0C | 0x80, "I"},
  {0x0D | 0x80, "J"},
  {0x0E | 0x80, "K"},
  {0x0F | 0x80, "L"},
  {0x10 | 0x80, "M"},
  {0x11 | 0x80, "N"},
  {0x12 | 0x80, "O"},
  {0x13 | 0x80, "P"},
  {0x14 | 0x80, "Q"},
  {0x15 | 0x80, "R"},
  {0x16 | 0x80, "S"},
  {0x17 | 0x80, "T"},
  {0x18 | 0x80, "U"},
  {0x19 | 0x80, "V"},
```

```
{0x1A | 0x80, "W"},
{0x1B | 0x80, "X"},
{0x1C | 0x80, "Y"},
{0x1D | 0x80, "Z"},
//...
{0x1E, "1"},
{0x1F, "2"},
{0x20, "3"},
{0x21, "4"},
{0x22, "5"},
{0x23, "6"},
{0x24, "7"},
{0x25, "8"},
{0x26, "9"},
{0x27, "0"},
{0x1E | 0x80, "!"},
{0x1F | 0x80, "@"},
{0x20 | 0x80, "#"},
{0x21 | 0x80, "$"},
{0x22 | 0x80, "%"},
{0x23 | 0x80, "^"},
{0x24 | 0x80, "&"},
{0x25 | 0x80, "*"},
{0x26 | 0x80, "("},
{0x27 | 0x80, ")"},
//...
{0x28, "ENTER"},
{0x29, "ESC"},
{0x2A, "BKSPC"},
{0x2B, "TAB"},
{0x2C, "SPACE"},
//...
{0x2D, "- MINUS"},
{0x2E, "="},
{0x2F, "["},
{0x30, "]"},
{0x31, "\\"},
{0x33, "; SEMICOLON"},
{0x34, "' SNGLQUOTE"},
{0x35, "`"},
{0x36, ", COMMA"},
{0x37, ". PERIOD"},
{0x38, "/"},
{0x2D | 0x80, "_ UNDERSCORE"},
{0x2E | 0x80, "+"},
{0x2F | 0x80, "{"},
{0x30 | 0x80, "}"},
{0x31 | 0x80, "|"},
{0x33 | 0x80, ": COLON"},
{0x34 | 0x80, "\" DBLQUOTE"},
{0x35 | 0x80, "~"},
{0x36 | 0x80, "<"},
{0x37 | 0x80, ">"},
{0x38 | 0x80, "?"},
//...
{0x3A, "F1"},
{0x3B, "F2"},
{0x3C, "F3"},
{0x3D, "F4"},
{0x3E, "F5"},
{0x3F, "F6"},
{0x40, "F7"},
{0x41, "F8"},
{0x42, "F9"},
{0x43, "F10"},
{0x44, "F11"},
{0x45, "F12"},
{0x46, "PRNTSCRN"},
{0x49, "INSERT"},
{0x4A, "HOME"},
```

```
  {0x4B, "PGUP"},
  {0x4C, "DELETE"},
  {0x4D, "END"},
  {0x4E, "PGDN"},
  {0x4F, "RIGHT"},
  {0x50, "LEFT"},
  {0x51, "DOWN"},
  {0x52, "UP"},
  {0x54, "KP /"},
  {0x55, "KP *"},
  {0x56, "KP - MINUS"},
  {0x57, "KP +"},
  {0x58, "KP ENTER"},
  {0xFF, NULL}, // cut this list short
};
```

USB Keyboard ID Display Format

The following discussion pertains to the Display Properties when pcProx is used in the keystroke send mode of operation, i.e. the HaltKBSends configuration flag is cleared thereby enabling keyboard sends of the ID. The Library call 'GetActiveID' will return a binary 8-byte ID which is only affected by the leading / trailing parity bit counts. The binary ID is stripped of the parity bits if these settings are non-zero. Other display properties do NOT affect the binary ID.

The card ID can be thought of as a bit stream that usually has leading parity bit(s), the card ID, and trailing parity bit(s). The card ID may be logically broken down into a Facility Code (FAC) and Card ID. It is the Card ID (without FAC) that is usually printed on the card. As an example, one 'standard' 26-bit Wiegand format has 1 leading parity bit, 8 bits of FAC, 16 bits of Card ID, and 1 trailing parity bit.

Example using a 'standard' 26-bit card printed with ID = '05559':

API demo using GetActiveID() with HaltKBSends = TRUE,
Leading parity = Trailing parity = 0:
Returns:
Hex **00.00.00.00.00.02.2B.6F** (26 bits) = binary 00000000100010101101101111

while Leading parity = Trailing parity = 1:
Returns:
Hex **00.00.00.00.00.01.15.B7** (24 bits) = binary 000000010001010110110111

API demo with HaltKBSends = FALSE to display the keyboard characters,
Leading parity = Trailing parity = 0:
Displays:
142191 (= hex 22B6F)

while Leading parity = Trailing parity = 1:
Displays:
71095 (= hex 115B7)

With Leading parity = Trailing parity = 1, IDBitcount = 16,
FixedLengthDisplay = True, Strip FAC = True, Send FAC = True, UseFACIDDelim = True,
ID Length = 5, FAC Length = 3:
Displays:
001:05559 (which starts to resemble the ID on the card)
The same read with Send FAC = False, UseFACIDDelim = False,
Displays:
05559 (which is exactly the ID printed on the card albeit not all of the info in the card ID)

The Library allows the user to tailor this display to fit the requirements of the receiving application.

The general format for the card ID as sent to the PC as keystrokes is:
[lead parity][card ID][trail parity]<EndLineDelimiter>

Which may be further refined to be:
[lead parity][FAC]<FACIDDelimiter>[Card ID][trail parity]<EndLineDelimiter>

The purest 'raw' read of the card ID would use the 'GetActiveID' library call and include the Wiegand parity bits, leading and trailing, and not limit the ID bit length. This is accomplished with the settings:

```
  CfgFlags.bFrcBitCntEx = 0;   // Force Rx'd bit count to be exact to be valid
  CfgFlags.bStripFac = 0;      // Strip the FAC from the ID (not discarded)
  CfgFlags.bHaltKBSnd = 1;     // Don't Send keys to USB (GetActiveID mechanism may be
used always)
  IDBitCnts.iLeadParityBitCnt = 0;  // Wiegand Leading Parity bit count to be stripped
  IDBitCnts.iTrailParityBitCnt = 0; // Wiegand Trailing Parity bit count to be stripped
```

Your card supplier should be able to tell you what format the ID is in: whether the entire ID is composed of a FAC and Card ID or just one huge ID, etc. This is the best way to determine the ID format as a trial-and-error approach using the API Demo program and existing cards will not always guarantee that the line between the FAC and Card ID can actually be seen. Given a card with '1' printed on it, it could be impossible to find out how many bits are actually part of the ID space.

Your card supplier will probably NOT be able to tell you how many parity bits are included in the bit stream as these bits are not necessarily part of the card data but may be added by the reader in pcProx. A trial approach using the API Demo program is effective in determining at least the trailing bit count.

If any displayable segment of the entire card ID exceeds 32 bits, the decimal display cannot be taken literally as an integer. The pcProx reader will convert the low 32 bits to an unsigned integer and pre-pend the remaining bits as another independent unsigned integer. The largest unsigned 32-bit integer is 4294967295 which displays as 10 decimal digits. A displayed ID of 154294967295 is actually indicating 15:4294967295 which would have been converted from an ID (in hexadecimal notation) of 'FFFFFFFF'. The displayed 154294967295 would literally convert to hexadecimal '23ECB25BFF' which is obviously not the same integer number. Regardless of which interpretation is taken as the 'card ID', it is no less unique to the organization – as long as the interpretation is consistent across all card reads within the target application.

In some cases (card ID bit counts > 32), it may be desirable to artificially define the Card ID as 32 bits and use the FAC-ID delimiter to partition off the remainder of the bits to make for a more easily manageable display.

OCX Filter

Included with the SDK is an ActiveX control to aid software developers in splitting the bit-stream read from the proximity card into parity, facility, and ID codes. Note: As with other OCX ActiveX routines you will need to register the OCX using a Windows registry tool such as REGSVR32.EXE.

> Example to register under Windows 2000: 'regsvr32 rfidProxfilter.ocx'
> Example to de-register under Windows 2000: 'regsvr32 rfidProxfilter.ocx -u'

The OCX, called **RFIDProxFilter,** includes the following routines:

RFIDProxFilter.DoTheSplit(Filter, BufHex, FAC, ID)

> **Overview**
> This subroutines splits the supplied BufHex into facility code (FAC), and ID code (ID) using the supplied filter.
>
> **Parameters**
>
> *Filter* – String. Describes which binary bit locations will be combined into parity, facility or ID codes. Example for a popular 26 bit Wiegand format is: "PFFFFFFFFIIIIIIIIIIIIIIIIP". This has 2 parity bits, 8 facility code bits, and 16 ID code bits.
>
> *BufHex* – String. Proximity card's data supplied as a string in hex format. Example: The cards data of the form "00.00.00.02.00.20.07.DC" is submitted as "00000002002007DC". Note that a byte must be 2 character positions long so a hex 2 is sent as hex '02'.
>
> *FAC*– String. Holds the resulting facility code in a binary format.
>
> *ID* – String. Holds the resulting ID code in a binary format.

Example: Call RFIDProxFilter.DoTheSplit(Filter$, BufHex$, FAC$, ID$) (subroutine in VB syntax)

RFIDProxFilter.BinaryToDecimal(Binary) (function call in VB syntax)

> **Overview**
> This function converts the supplied binary data, returning a decimal string
>
> **Parameters**
> *Binary* - String of the form "010011…0010"
>
> Example: Text5.Text = RFIDProxFilter1.BinaryToDecimal(FAC$)