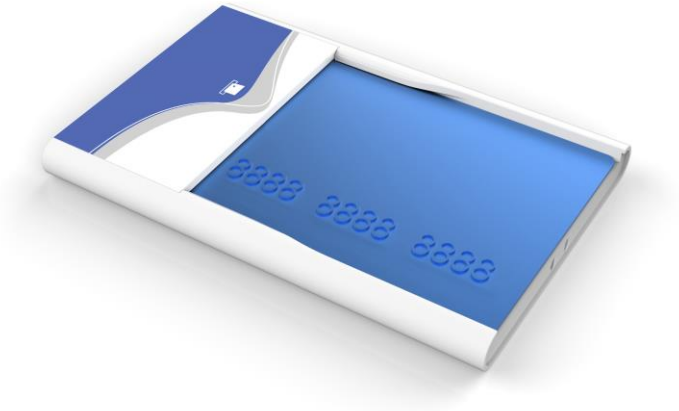
FEITIAN

bR500 Smartcard Reader iOS Developer Guide



Revision History:

Date	Revision	Description
Feb, 2016	V1.0	Release of the first version

Software Developer's Agreement

All Products of Feitian Technologies Co., Ltd. (Feitian) including, but not limited to, evaluation copies, diskettes, CD-ROMs, hardware and documentation, and all future orders, are subject to the terms of this Agreement. If you do not agree with the terms herein, please return the evaluation package to us, postage and insurance prepaid, within seven days of their receipt, and we will reimburse you the cost of the Product, less freight and reasonable handling charges.

1. Allowable Use – You may merge and link the Software with other programs for the sole purpose of protecting those programs in accordance with the usage described in the Developer's Guide. You may make archival copies of the Software.

2. Prohibited Use – The Software or hardware or any other part of the Product may not be copied, reengineered, disassembled, decompiled, revised, enhanced or otherwise modified, except as specifically allowed in item 1. You may not reverse engineer the Software or any part of the product or attempt to discover the Software's source code. You may not use the magnetic or optical media included with the Product for the purposes of transferring or storing data that was not either an original part of the Product, or a Feitian provided enhancement or upgrade to the Product.

3. Warranty – Feitian warrants that the hardware and Software storage media are substantially free from significant defects of workmanship or materials for a time period of twelve (12) months from the date of delivery of the Product to you.

4. Breach of Warranty – In the event of breach of this warranty, Feitian's sole obligation is to replace or repair, at the discretion of Feitian, any Product free of charge. Any replaced Product becomes the property of Feitian.

Warranty claims must be made in writing to Feitian during the warranty period and within fourteen (14) days after the observation of the defect. All warranty claims must be accompanied by evidence of the defect that is deemed satisfactory by Feitian. Any Products that you return to Feitian, or a Feitian authorized distributor, must be sent with freight and insurance prepaid.

EXCEPT AS STATED ABOVE, THERE IS NO OTHER WARRANTY OR REPRESENTATION OF THE PRODUCT, EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

5. Limitation of Feitian's Liability – Feitian's entire liability to you or any other party for any cause whatsoever, whether in contract or in tort, including negligence, shall not exceed the price you paid for the unit of the Product that caused the damages or are the subject of, or indirectly related to the cause of action. In no event shall Feitian be liable for any damages caused by your failure to meet your obligations, nor for any loss of data, profit or savings, or any other consequential and incidental damages, even if Feitian has been advised of the possibility of damages, or for any claim by you based on any third-party claim.

6. Termination – This Agreement shall terminate if you fail to comply with the terms herein. Items 2, 3, 4 and 5 shall survive any termination of this Agreement.

Contents

Chapter 1. Overview

This chapter describes how to develop bR500 reader applications, including the development interfaces supported by the product (bR500) and how to develop applications based on these interfaces.

FEITIAN bR500 is specially engineered to accommodate a range of smart card applications. Developers use it as a platform to generate and deploy related products and services. Moreover, FEITIAN bR500 is a terminal unit which is seamlessly integrated to all major systems of operation. Additional features such as the built-in inclusive support for different smart card interfaces has facilitated the wide scale and cross industry adoption of bR500.

bR500 suits customers where security concerns are the most salient and satisfies the demand for a flexible solution for ID authentication, e-commerce, e-payment, information security and access control.

BR500 and the rest of FEITIAN's line of smart card readers offer each customer a complete solution for all manner of utilizations.

Chapter 2. Definitions

2.1 Error codes

Below list down commonly used errors. All errors from different cards must map over to these error messages.

#define SCARD_S_SUCCESS 0x00000000 No error was encountered

#define SCARD_E_INVALID_HANDLE 0x80100003 Handle was invalid

#define SCARD_E_INVALID_PARAMETER 0x80100004 One or more of the supplied parameters could not be properly interpreted.

#define SCARD_E_INSUFFICIENT_BUFFER 0x80100008 The data buffer to receive returned data is too small for the returned data.

#define SCARD_E_UNKNOWN_READER 0x80100009 The specified reader name is not recognized.

#define SCARD_E_NO_SMARTCARD 0x8010000C The operation requires a Smart Card, but no Smart Card is currently in the device

#define SCARD_E_UNKNOWN_CARD 0x8010000D The specified smart card name is not recognized.

#define SCARD_E_INVALID_VALUE 0x80100011 One or more of the supplied parameters values could not be properly interpreted.

#define SCARD_F_COMM_ERROR 0x80100013 An internal communications error has been detected.

#define SCARD_F_UNKNOWN_ERROR 0x80100014 An internal error has been detected, but the source is unknown.

#define SCARD_E_INVALID_ATR 0x80100015 An ATR obtained from the registry is not a valid ATR string.

#define SCARD_E_NOT_TRANSACTED 0x80100016 An attempt was made to end a non-existent transaction

#define SCARD_E_READER_UNAVAILABLE 0x80100017 The specified reader is not currently available for use

#define SCARD_E_READER_UNSUPPORTED 0x8010001A The reader driver does not meet minimal requirements for support

#define SCARD_E_CARD_UNSUPPORTED 0x8010001C The smart card does not meet minimal requirements for support.

Chapter 3. API Reference

3.1 SCardEstablishContext

Synopsis:

```
#include <winscard.h>
LONG SCardEstablishContext(DWORD dwScope,
    /*@unused@*/ LPCVOID pvReserved1,
    /*@unused@*/ LPCVOID pvReserved2,
    LPSCARDCONTEXT phContext);
```

Parameters:

dwScope	IN	Scope of the establishment
pvReserved1		unused
pvReserved2		unused
phContext	OUT	Returned reference to this connection

Description:

This function creates a communication context to the PC/SC Resource Manager. This must be the first function called in a PC/SC application.

Value of dwScope Meaning

SCARD_SCOPE_USER	Not used
SCARD_SCOPE_TERMINAL	Not used
SCARD_SCOPE_GLOBAL	Not used
SCARD_SCOPE_SYSTEM	Services on the local machine

Example:

```
SCARDCONTEXT    hContext;
LONG            rv;
rv = SCardEstablishContext(SCARD_SCOPE_SYSTEM, NULL, NULL, &hContext);
```

Returns:

SCARD_S_SUCCESS	Successful
SCARD_E_INVALID_VALUE	Invalid scope type passed
SCARD_E_INVALID_PARAMETER	Invalid parameter

3.2 SCardReleaseContext

Synopsis:

#include <winscard.h>
LONG SCardReleaseContext(SCARDCONTEXT hContext);

Parameters:
hContext IN Connection context to be closed

Description:
This function destroys a communication context to the PC/SC Resource Manager. This must be the last function called in a PC/SC application.

Example:
SCARDCONTEXT hContext;
LONG rv;
rv = SCardEstablishContext(SCARD_SCOPE_SYSTEM, NULL, NULL, &hContext);
rv = SCardReleaseContext(hContext);

Returns:
SCARD_S_SUCCESS Successful
SCARD_E_INVALID_HANDLE Invalid hContext handle

3.3 SCardIsValidContext

Synopsis:
#include <winscard.h>
LONG SCardIsValidContext(SCARDCONTEXT hContext);

Parameters:
hContext IN Connection context to be checked

Description:
This function determines whether a smart card context handle is still valid. After a smart card context handle has been set by SCardEstablishContext(), it may become not valid if the resource manager service has been shut down.

Example:
SCARDCONTEXT hContext;
LONG rv;
rv = SCardEstablishContext(SCARD_SCOPE_SYSTEM, NULL, NULL, &hContext);
rv = SCardIsValidContext(hContext);

Returns:
SCARD_S_SUCCESS Successful
SCARD_E_INVALID_HANDLE Invalid hContext handle

3.4 SCardListReaders

Synopsis:

#include <winscard.h>

LONG SCardListReaders(SCARDCONTEXT hContext,

/*@null@*/ /*@out@*/ LPCSTR mszGroups,

/*@null@*/ /*@out@*/ LPSTR mszReaders,

/*@out@*/ LPDWORD pcchReaders);

Parameters:

hContext	IN	Connection context to the PC/SC Resource Manager
mszGroups	IN	List of groups to list readers (not used)
mszReaders	OUT	Multi-string with list of readers
pcchReaders	OUT	Size of multi-string buffer including NULL's

Description:

This function returns a list of currently available readers on the system. mszReaders is a pointer to a character string that is allocated by the application. If the application sends mszGroups and mszReaders as NULL then this function will return the size of the buffer needed to allocate in pcchReaders. The reader names is a multi-string and separated by a nul character ('\0') and ended by a double null character. "Reader A\0Reader B\0\0".

Example:

```
SCARDCONTEXT    hContext;
LPSTR           mszReaders;
DWORD           dwReaders;
LONG            rv;
rv = SCardEstablishContext(SCARD_SCOPE_SYSTEM, NULL, NULL, &hContext);
rv = SCardListReaders(hContext, NULL, NULL, &dwReaders);
mszReaders = malloc(sizeof(char)*dwReaders);
rv = SCardListReaders(hContext, NULL, mszReaders, &dwReaders);
```

Returns:

SCARD_S_SUCCESS	Successful
SCARD_E_INVALID_HANDLE	Invalid Scope Handle
SCARD_E_INSUFFICIENT_BUFFER	Reader buffer not large enough
SCARD_E_INVALID_PARAMETER	Invalid parameter

3.5 SCardConnect

Synopsis:

#include <winscard.h>

LONG SCardConnect(SCARDCONTEXT hContext,

LPCSTR szReader,
DWORD dwShareMode,
DWORD dwPreferredProtocols,
LPSCARDHANDLE phCard,
LPDWORD pdwActiveProtocol);

Parameters:

hContext	IN	Connection context to the PC/SC Resource Manager
szReader	IN	Reader name to connect to
dwShareMode	IN	Mode of connection type: exclusive or shared
dwPreferredProtocols	IN	Desired protocol use
phCard	OUT	Handle to this connection
pdwActiveProtocol	OUT	Established protocol to this connection.

Description:

This function establishes a connection to the friendly name of the reader specified in szReader. The first connection will power up and perform a reset on the card. Value of dwShareMode Meaning
SCARD_SHARE_SHARED This application will allow others to share the reader
SCARD_SHARE_EXCLUSIVE This application will NOT allow others to share the reader
SCARD_SHARE_DIRECT Direct control of the reader, even without a card
SCARD_SHARE_DIRECT can be used before using SCardControl() to send control commands to the reader even if a card is not present in the reader.
Value of dwPreferredProtocols Meaning
SCARD_PROTOCOL_T0 Use the T=0 protocol
SCARD_PROTOCOL_T1 Use the T=1 protocol
SCARD_PROTOCOL_RAW Use with memory type cards
dwPreferredProtocols is a bit mask of acceptable protocols for the connection. You can use (SCARD_PROTOCOL_T0 | SCARD_PROTOCOL_T1) if you do not have a preferred protocol.

Example:

```
SCARDCONTEXT     hContext;
SCARDHANDLE      hCard;
DWORD            dwActiveProtocol;
LONG             rv;
rv = SCardEstablishContext(SCARD_SCOPE_SYSTEM, NULL, NULL, &hContext);
rv = SCardConnect(hContext, "Reader X", SCARD_SHARE_SHARED,
SCARD_PROTOCOL_T0, &hCard, &dwActiveProtocol);
```

Returns:

SCARD_S_SUCCESS	Successful
SCARD_E_INVALID_HANDLE	Invalid hContext handle
SCARD_E_INVALID_PARAMETER	Invalid parameter
SCARD_E_NO_SMARTCARD	no smart card
SCARD_E_READER_UNAVAILABLE	Could not power up the reader or card

SCARD_E_UNSUPPORTED_FEATURE Protocol not supported

3.6 FtGetDeviceHID(private interface)

Synopsis:

#include <winscard.h>

LONG FtGetDeviceHID(SCARDCONTEXT hContext, unsigned int length,char * buffer);

Parameters:

hContext	IN	Context of reader
length	IN	length of buffer (>=8)
buffer	OUT	Serial number

Description:

This function used to get hardware serial number of reader.

Example:

```
SCARDCONTEXT    hContext;
SCARDHANDLE     hCard;
DWORD           dwActiveProtocol;
LONG            rv;
Char            buffer[20] = {0};
rv = SCardEstablishContext(SCARD_SCOPE_SYSTEM, NULL, NULL, &hContext);
rv = SCardConnect(hContext, "Reader X", SCARD_SHARE_SHARED,
SCARD_PROTOCOL_T0, &hCard, &dwActiveProtocol);
rv = FtGetDeviceHID (hContext, sizeof(buffer), buffer);
```

Returns:

SCARD_S_SUCCESS	Successful
SCARD_F_COMM_ERROR	Get serial Num failed
SCARD_E_INVALID_PARAMETER	Invalid parameter

3.7 FtWriteFlash (private interface)

Synopsis:

#include <winscard.h>

```
LONG FtWriteFlash(unsigned int reader_index,
    unsigned char bOffset,
    unsigned char blength,
    unsigned char buffer[]);
```

Parameters:

reader_index	IN	reader index

bOffset	IN	Offset of flash to write
blength	IN	The length of data
buffer	IN	The data for write

Description:

This function userd to write data to flash.

Example:

```
SCARDCONTEXT      hContext;
SCARDHANDLE       hCard;
DWORD             dwActiveProtocol;
LONG              rv;
unsigned char buffer[255] ={0};
rv = SCardEstablishContext(SCARD_SCOPE_SYSTEM, NULL, NULL, &hContext);
rv = SCardConnect(hContext, "Reader X", SCARD_SHARE_SHARED,SCARD_PROTOCOL_T0, &hCard,
&dwActiveProtocol);
for (int i=0; i< 255; i++) {
      buffer[i]= i;
}
rv = FtWriteFlash(0, 0,255, buffer);
```

Returns:

SCARD_S_SUCCESS	Successful
SCARD_F_COMM_ERROR	write data failed
SCARD_E_INVALID_PARAMETER	Invalid parameter

3.8 FtReadFlash(private interface)

Synopsis:

```
#include <winscard.h>
LONG FtReadFlash(unsigned int reader_index,
      unsigned char bOffset,
      unsigned char blength,
      unsigned char buffer[]);
```

Parameters:

reader_index	IN	reader index
bOffset	IN	Offset of flash to write
blength	IN	The length of read data
buffer	OUT	The read data

Description:

This function used to read data from flash.

Example:

```
SCARDCONTEXT    hContext;
SCARDHANDLE     hCard;
DWORD           dwActiveProtocol;
LONG            rv;
unsigned char buffer[255] ={0};
rv = SCardEstablishContext(SCARD_SCOPE_SYSTEM, NULL, NULL, &hContext);
rv = SCardConnect(hContext, "Reader X", SCARD_SHARE_SHARED, SCARD_PROTOCOL_T0, &hCard,
&dwActiveProtocol);
rv = FtReadFlash (0, 0,255, buffer);
```

Returns:

SCARD_S_SUCCESS	Successful
SCARD_F_COMM_ERROR	write data failed
SCARD_E_INVALID_PARAMETER	Invalid parameter

3.9 SCardReconnect

Synopsis:

```
#include <winscard.h>
LONG SCardReconnect(SCARDHANDLE hCard,
    DWORD dwShareMode,
    DWORD dwPreferredProtocols,
    DWORD dwInitialization,
    LPDWORD pdwActiveProtocol);
```

Parameters:

hCard	IN	Handle to a previous call to connect
dwShareMode	IN	Mode of connection type: exclusive/shared
dwPreferredProtocols	IN	Desired protocol use
dwInitialization	IN	Desired action taken on the card/reader
pdwActiveProtocol	OUT	Established protocol to this connection

Description:

This function reestablishes a connection to a reader that was previously connected to using SCardConnect(). In a multi application environment it is possible for an application to reset the card in shared mode. When this occurs any other application trying to access certain commands will be returned the value SCARD_W_RESET_CARD. When this occurs SCardReconnect() must be called in order to acknowledge that the card was reset and allow it to change it's state accordingly.

Value of dwShareMode Meaning

SCARD_SHARE_SHARED This application will allow others to share the reader

SCARD_SHARE_EXCLUSIVE This application will NOT allow others to share the reader

Value of dwPreferredProtocols Meaning
SCARD_PROTOCOL_T0 Use the T=0 protocol
SCARD_PROTOCOL_T1 Use the T=1 protocol
SCARD_PROTOCOL_RAW Use with memory type cards
dwPreferredProtocols is a bit mask of acceptable protocols for the connection. You can use (SCARD_PROTOCOL_T0 | SCARD_PROTOCOL_T1) if you do not have a preferred protocol.

Value of dwInitialization Meaning

SCARD_LEAVE_CARD	Do nothing
SCARD_RESET_CARD	Reset the card (warm reset)
SCARD_UNPOWER_CARD	Unpower the card (cold reset)
SCARD_EJECT_CARD	Eject the card

Example:

```
SCARDCONTEXT    hContext;
SCARDHANDLE     hCard;
DWORD   dwActiveProtocol, dwSendLength, dwRecvLength;
LONG    rv;
BYTE    pbRecvBuffer[10];
BYTE    pbSendBuffer[] = {0xC0, 0xA4, 0x00, 0x00, 0x02, 0x3F, 0x00};
rv = SCardEstablishContext(SCARD_SCOPE_SYSTEM, NULL, NULL, &hContext);
rv = SCardConnect(hContext, "Reader X", SCARD_SHARE_SHARED,
SCARD_PROTOCOL_T0, &hCard, &dwActiveProtocol);
dwSendLength = sizeof(pbSendBuffer);
dwRecvLength = sizeof(pbRecvBuffer);
rv = SCardTransmit(hCard, SCARD_PCI_T0, pbSendBuffer, dwSendLength,
&pioRecvPci, pbRecvBuffer, &dwRecvLength);
/* Card has been reset by another application */
if (rv == SCARD_W_RESET_CARD)
{
    rv = SCardReconnect(hCard, SCARD_SHARE_SHARED, SCARD_PROTOCOL_T0,
        SCARD_RESET_CARD, &dwActisveProtocol);
}
```

Returns:

SCARD_S_SUCCESS	Successful
SCARD_E_INVALID_HANDLE	Invalid hContext handle
SCARD_E_INVALID_PARAMETER	Invalid parameter
SCARD_E_NO_SMARTCARD	no smart card
SCARD_E_READER_UNAVAILABLE	Could not power up the reader or card
SCARD_E_UNSUPPORTED_FEATURE	Protocol not supported

3.10 SCardDisconnect

Synopsis:

#include <winscard.h>

LONG SCardDisconnect(SCARDHANDLE hCard,

DWORD dwDisposition);

Parameters:

hCard	IN	Connection made from SCardConnect
dwDisposition	IN	Reader function to execute

Description:

This function terminates a connection to the connection made through SCardConnect.

dwDisposition can have the following values:

Value of dwDisposition	Meaning
SCARD_LEAVE_CARD	Do nothing
SCARD_RESET_CARD	Reset the card (warm reset)
SCARD_UNPOWER_CARD	Unpower the card (cold reset)
SCARD_EJECT_CARD	Eject the card

Example:

```
SCARDCONTEXT hContext;
SCARDHANDLE hCard;
DWORD dwActiveProtocol;
LONG rv;
rv = SCardEstablishContext(SCARD_SCOPE_SYSTEM, NULL, NULL, &hContext);
rv = SCardConnect(hContext, "Reader X", SCARD_SHARE_SHARED,
SCARD_PROTOCOL_T0, &hCard, &dwActiveProtocol);
rv = SCardDisconnect(hCard, SCARD_UNPOWER_CARD);
```

Returns:

SCARD_S_SUCCESS	Successful
SCARD_E_INVALID_HANDLE	Invalid hCard handle
SCARD_E_INVALID_VALUE	Invalid dwDisposition

3.11 SCardStatus

Synopsis:

#include <winscard.h>

LONG SCardStatus(SCARDHANDLE hCard,

LPSTR mszReaderNames,

LPDWORD pcchReaderLen,

LPDWORD pdwState,
LPDWORD pdwProtocol,
LPBYTE pbAtr,
LPDWORD pcbAtrLen);

Parameters:

hCard	IN	Connection made from SCardConnect
mszReaderNames	IN OUT	Friendly name of this reader
pcchReaderLen	IN OUT	Size of the szReaderName multistring
pdwState	OUT	Current state of this reader
pdwProtocol	OUT	Current protocol of this reader
pbAtr	OUT	Current ATR of a card in this reader
pcbAtrLen	OUT	Length of ATR

Description:

This function returns the current status of the reader connected to by hCard. It's friendly name will be stored in mszReaderNames. pcchReaderLen will be the size of the allocated buffer for mszReaderNames, while pcbAtrLen will be the size of the allocated buffer for pbAtr. If either of these is too small, the function will return with SCARD_E_INSUFFICIENT_BUFFER and the necessary size in pcchReaderLen and pcbAtrLen. The current state, and protocol will be stored in pdwState and pdwProtocol respectively. pdwState is a DWORD possibly OR'd with the following values:

Value of pdwState	Meaning
SCARD_ABSENT	There is no card in the reader
SCARD_PRESENT	There is a card in the reader, but it has not been moved into position for use
SCARD_SWALLOWED	There is a card in the reader in position for use. The card is not powered
SCARD_POWERED	Power is being provided to the card, but the reader driver is unaware of the mode of the card
SCARD_NEGOTIABLE	The card has been reset and is awaiting PTS negotiation
SCARD_SPECIFIC	The card has been reset and specific communication protocols have been established

Value of pdwProtocol	Meaning
SCARD_PROTOCOL_T0	Use the T=0 protocol
SCARD_PROTOCOL_T1	Use the T=1 protocol

Example:

```
SCARDCONTEXT    hContext;
SCARDHANDLE     hCard;
DWORD           dwActiveProtocol;
DWORD           dwState, dwProtocol, dwAtrLen, dwReaderLen;
BYTE            pbAtr[MAX_ATR_SIZE];
rv = SCardEstablishContext(SCARD_SCOPE_SYSTEM, NULL, NULL, &hContext);
rv = SCardConnect(hContext, "Reader X", SCARD_SHARE_SHARED,
SCARD_PROTOCOL_T0, &hCard, &dwActiveProtocol);
dwAtrLen = sizeof(pbAtr);
rv=SCardStatus(hCard, NULL, &dwReaderLen, &dwState, &dwProtocol,pbAtr, &dwAtrLen);
```

Returns:

SCARD_S_SUCCESS	Successful
SCARD_E_INSUFFICIENT_BUFFER	Not enough allocated memory for mszReaderNames or for pbAtr

3.12 SCardGetAttrib

Synopsis:

```
#include <winscard.h>
LONG SCardGetAttrib(SCARDHANDLE hCard,
    DWORD dwAttrId,
    LPBYTE pbAttr,
    LPDWORD pcbAttrLen);
```

Parameters:

hCard	IN	Connection made from SCardConnect
dwAttrId	IN	Identifier for the attribute to get
pbAttr	OUT	Pointer to a buffer that receives the attribute
pcbAttrLen	IN/OUT	Length of the pbAttr buffer in bytes

Description:

This function get an attribute from the IFD Handler. The list of possible attributes is:

• SCARD_ATTR_ATR_STRING

Example:

```
LONG            rv;
SCARDCONTEXT    hContext;
SCARDHANDLE     hCard;
DWORD           dwActiveProtocol;
unsigned char   pbAtr[MAX_ATR_SIZE];
DWORD           dwAtrLen;
rv = SCardEstablishContext(SCARD_SCOPE_SYSTEM, NULL, NULL, &hContext);
rv = SCardConnect(hContext, "Reader X", SCARD_SHARE_SHARED,
SCARD_PROTOCOL_RAW &hCard, &dwActiveProtocol);
rv = SCardGetAttrib(hCard, SCARD_ATTR_ATR_STRING, pbAtr, &dwAtrLen);
```

Returns:

SCARD_S_SUCCESS	Successful
SCARD_E_INVALID_HANDLE	Invalid hCard handle
SCARD_E_INVALID_PARAMETER	Invalid parameter
SCARD_E_INSUFFICIENT_BUFFER	receive buffer not large enough
SCARD_E_NOT_TRANSACTED	Data exchange not successful
SCARD_E_SHARING_VIOLATION	Someone else has exclusive rights

SCARD_E_READER_UNAVAILABLE	The reader has been removed

3.13 SCardTransmit

Synopsis:

```
#include <winscard.h>
LONG SCardTransmit(SCARDHANDLE hCard,
    const SCARD_IO_REQUEST *pioSendPci,
    LPCBYTE pbSendBuffer,
    DWORD cbSendLength,
    SCARD_IO_REQUEST *pioRecvPci,
    LPBYTE pbRecvBuffer,
    LPDWORD pcbRecvLength);
```

Parameters:

hCard	IN	Connection made from SCardConnect
pioSendPci	IN/OUT	Structure of protocol information
pbSendBuffer	IN	APDU to send to the card
cbSendLength	IN	Length of the APDU
pioRecvPci	IN/OUT	Structure of protocol information
pbRecvBuffer	OUT	Response from the card
pcbRecvLength	IN/OUT	Length of the response

Description:

This function sends an APDU to the smart card contained in the reader connected to by SCardConnect(). The card responds from the APDU and stores this response in pbRecvBuffer and it's length in SpcbRecvLength. SSendPci and SRecvPci are structures containing the following:

```
typedef struct {
    DWORD dwProtocol; /* SCARD_PROTOCOL_T0 or SCARD_PROTOCOL_T1 */
    DWORD cbPciLength; /* Length of this structure - not used */
} SCARD_IO_REQUEST;
```

Value of pioSendPci Meaning

SCARD_PCI_T0	Pre-defined T=0 PCI structure
SCARD_PCI_T1	Pre-defined T=1 PCI structure

Example:

```
LONG            rv;
SCARDCONTEXT    hContext;
SCARDHANDLE     hCard;
DWORD           dwActiveProtocol, dwSendLength, dwRecvLength;
SCARD_IO_REQUEST    pioRecvPci;
BYTE    pbRecvBuffer[10];
```

```
BYTE        pbSendBuffer[] = { 0xC0, 0xA4, 0x00, 0x00, 0x02, 0x3F, 0x00 };
rv = SCardEstablishContext(SCARD_SCOPE_SYSTEM, NULL, NULL, &hContext);
rv = SCardConnect(hContext, "Reader X", SCARD_SHARE_SHARED,
        SCARD_PROTOCOL_T0, &hCard, &dwActiveProtocol);
dwSendLength = sizeof(pbSendBuffer);
dwRecvLength = sizeof(pbRecvBuffer);
rv = SCardTransmit(hCard, SCARD_PCI_T0, pbSendBuffer, dwSendLength,
&pioRecvPci, pbRecvBuffer, &dwRecvLength);
```

Returns:

SCARD_S_SUCCESS	Successful
SCARD_E_INVALID_HANDLE	Invalid hCard handle
SCARD_E_INSUFFICIENT_BUFFER	receive buffer not large enough
SCARD_E_NOT_**TRANSACTED**	Data exchange not successful
SCARD_E_INVALID_PARAMETER	invalid parameter
SCARD_E_INVALID_VALUE	Invalid Protocol, reader name, etc

3.14 SCardGetStatusChange

Synopsis:

```
#include <winscard.h>
LONG SCardGetStatusChange(SCARDCONTEXT hContext,
        DWORD dwTimeout,
        LPSCARD_READERSTATE rgReaderStates,
        DWORD cReaders);
```

Parameters:

hContext	IN	Connection context to the PC/SC Resource Manager
dwTimeout	IN	Maximum waiting time (in miliseconds) for status change, zero (or INFINITE) for infinite
rgReaderStates	IN/OUT	Structures of readers with current states
cReaders	IN	Number of structures

Description:

This function blocks execution until the current availability of the cards in a specific set of readers changes.

The caller supplies a list of readers to be monitored through an SCARD_READERSTATE array and the maximum amount of time, in seconds, that it is willing to wait for an action to occur on one of the listed readers. The function returns when there is a change in availability, having filled in the *dwEventState* members of the **SCARD_READERSTATE** structures appropriately.

Example:

```
SCARDCONTEXT            hContext;
SCARD_READERSTATE_A     rgReaderStates[1];
LONG    rv;
rv = SCardEstablishContext(SCARD_SCOPE_SYSTEM, NULL, NULL, &hContext);
rgReaderStates[0].szReader = "Reader X";
rgReaderStates[0].dwCurrentState = SCARD_STATE_UNAWARE;
rv = SCardGetStatusChange(hContext, INFINITE, rgReaderStates, 1);
printf("reader state: 0x%04X\n", rgReaderStates[0].dwEventState);
```

Returns:

SCARD_S_SUCCESS	Successful
SCARD_E_READER_UNAVAILABLE	The reader is unavailable

3.15 FtGenerateDeviceUID

Synopsis:

```
#include <winscard.h>

LONG FtGenerateDeviceUID(SCARDCONTEXT hContext, unsigned int    seedLength,unsigned char * seedBuffer);
```

Parameters:

hContext	IN	Connection context to the PC/SC Resource Manager
seedLength	IN	The seed length is 1-48 bytes
seedBuffer	IN/OUT	The seed which using generate UID

Description:

UID function is included generate the ID of User (UID). Only the User holding the legal UID can update the smart card reader. The software includes the following items: Seed File, Generate UID, Read UID, Erase UID.

When 'FtGenerateDeviceUID ' function is called, a UID will be generated in the smart card reader. If a UID (It is not 'FFFFFFFFFFFFFFFF') has already been existent in the smart card reader, it will show error. If the seed is empty, then you can input seed to generate a UID.

The seed can be any kind of ASCII character. The length of seed should be no more than 48bytes.

Example:

Please refer to IOS demo code

Returns:

SCARD_S_SUCCESS	Successful
SCARD_E_READER_UNAVAILABLE	The reader is unavailable
SCARD_F_COMM_ERR	communication error

3.16 FtGetDeviceUID

Synopsis:

#include <winscard.h>

LONG FtGetDeviceUID(SCARDCONTEXT hContext, unsigned int uidLength, char * uidBuffer);

Parameters:

hContext	IN	Connection context to the PC/SC Resource Manager
uidLength	IN	The length of UID, the UID is 8bytes
uidBuffer	OUT	The buffer stored UID

Description:

When 'FtGetDeviceUID' API is called, the UID which is stored in the smart card reader will be read out.

Example:

Please refer to IOS demo code

Returns:

SCARD_S_SUCCESS	Successful
SCARD_E_READER_UNAVAILABLE	The reader is unavailable
SCARD_F_COMM_ERR	communication error

3.17 FtEscapeDeviceUID

Synopsis:

#include <winscard.h>

LONG FtEscapeDeveviceUID(SCARDCONTEXT hContext, unsigned int seedLength,unsigned char * seedBuffer);

Parameters:

hContext	IN	Connection context to the PC/SC Resource Manager
seedLength	IN	The length of seed
seedBuffer	IN	The buffer stored seed, to erase UID need using seed

Description:

UID function is included generate the ID of User (UID). Only the User holding the legal UID can update the smart card reader. The software includes the following items: Seed File, Generate UID, Read UID, Erase UID.

A smart card reader in which a UID has been existent (The UID is not 'FFFFFFFFFFFFFFFF') cannot generate a new UID. The user must erase the old UID to generate a new UID. The old seed by which the user generated the UID

must be used to erase the UID. After the seed is imported and 'Erase UID' button is clicked, the old UID will be erased and become 'FFFFFFFFFFFFFFFF'.

Example:
Please refer to IOS demo code

Returns:

SCARD_S_SUCCESS	Successful
SCARD_E_READER_UNAVAILABLE	The reader is unavailable
SCARD_F_COMM_ERR	communication error

3.18 @interface ReaderInterface

Synopsis:
#include <winscard.h>

-(void) setDelegate:(id<ReaderInterfaceDelegate>)delegate;

Description:

API using set delegate, the delegate can monitor reader and card status, also the delegate allow user found peripheral reader.

Example:
Please refer to IOS demo code

Returns:
NULL

3.19 findPeripheralReader

Synopsis:
#include <winscard.h>

-(void)findPeripheralReader:(NSString *)readerName

Parameters:

readerName	OUT	device name which found

Description:

Before using this function, you will need create delegate "ReaderInterfaceDelegate", this API using to discover peripheral devices

Example:
Please refer to IOS demo code

Returns:
NULL

3.20 readerInterfaceDidChange

Synopsis:
#include <winscard.h>

-(void)readerInterfaceDidChange:(BOOL)attached

Parameters:
attached OUT TURE means the reader connected, FALSE means disconnect

Description:

Before using this function, you will need create delegate "ReaderInterfaceDelegate", this API using to monitor reader connection status event

Example:
Please refer to IOS demo code

Returns:

NULL

3.21 cardInterfaceDidDetach

Synopsis:
#include <winscard.h>

-(void) cardInterfaceDidDetach:(BOOL)attached

Parameters:
attached OUT TURE means the card present, FALSE means absent

Description:

Before using this function, you will need create delegate "ReaderInterfaceDelegate", this API using to monitor card slot event

Example:
Please refer to IOS demo code

Returns:

NULL

3.22 connectPeripheralReader

Synopsis:
#include <winscard.h>

-(void) connectPeripheralReader:(NSString*) readername

Parameters:
readername IN connect specified reader

Description:

API using to connect specified reader that found by findPeripheralReader

Example:
Please refer to IOS demo code

Returns:

NULL

3.23 disConnectCurrentPeripheralReader

Synopsis:
#include <winscard.h>

-(void) disConnectCurrentPeripheralReader:(NSString*) readername

Parameters:
readername IN disconnect specified reader

Description:

API using to disconnect specified reader that found by findPeripheralReader

Example:
Please refer to IOS demo code

Returns:

NULL

3.24 isReaderAttached

Synopsis:
#include <winscard.h>

-(BOOL) isReaderAttached

Parameters:
NULL

Description:

Once the reader connected to IOS, will call this API

Example:
Please refer to IOS demo code

Returns:

NULL

3.25 isCardAttached

Synopsis:
#include <winscard.h>

-(BOOL) isCardAttached

Parameters:
NULL

Description:

Once the card insert to reader, will call this API

Example:

Please refer to IOS demo code

Returns:

NULL

Sample code:

```
@interface mainViewController:UIViewController<ReaderInterfaceDelegate>
{
    ReaderInterface *reader;
}
@end

@implementation mainViewController
- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)nibBundleOrNil
{
    self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil];
    if (self) {
        // Custom initialization
    }
    return self;
}

- (void)viewDidLoad
{
    /*
     *查看卡槽状态用 ReaderInterfaceDelegate
     */
    SCARDCONTEXT cardContext;
```

```
    reader =[[ReaderInterface alloc] init];
    [reader setDelegate:self];
    SCardEstablishContext(SCARD_SCOPE_SYSTEM, NULL, NULL,
                                  &cardContext);
    [self test];
    [super viewDidLoad];
    // Do any additional setup after loading the view from its nib.
}

-(void)test
{
    BOOL ReaderStatus;
    BOOL   CardStatus;
    ReaderStatus = isReaderAttached;
    if (ReaderStatus) {
        NSLog(@"\nreader is attached>>>");
    }else{
        NSLog(@"\nreader is disattached>>>");
    }
    CardStatus = isCardAttached;
    if (CardStatus) {
        NSLog(@"\ncard is attached>>>");
    }else{
        NSLog(@"\ncard is disattached>>>");
    }
}

- (void)didReceiveMemoryWarning
{
```

```
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

#pragma ReaderInterfaceDelegate
-(void)readerInterfaceDidChange:(BOOL)attached
{
    if (attached) {
        NSLog(@"\nreader is attached>>>");
    }else{
        NSLog(@"\nreader is disattached>>>");
    }
}
-(void)cardInterfaceDidDetach:(BOOL)attached
{
    if (attached) {
        NSLog(@"\ncard is attached>>>");
    }else{
        NSLog(@"\ncard is disattached>>>");
    }
}
```