Winscard manual

04/2010

Overview

The interface for MR-800 rfid reader is PC/SC SmartCard USB. The card and peripherals can be operated via using the windows built-in drive. So before writing an application, make sure it contains two files--*winscard.lib* and *winscard.h*. For the user convinient operation, the MR-800 card reader inherited most of the PC / SC specification, and also made some necessary process changes. MR-800 can both read the contact/contactless smartcard and memorycard/peripherals (such as LCD, BEEP, the LED, the RTC, etc.). So in the connected funtion (***SCardConnect***) is used to connect the reader(Generally in PC/SC, it is to establish a connection between the application and the smart card. In this situation, the problem is if there is no smart card, users can not operate memory card and peripheralbe conveniently.) As long as the reader is connected, ***SCardConnect*** always return the correct connection information. And the original smartcard ATR information is a fixed ATR information not the card's ATR information. If you want to operate the card, you can via ***GetData APDU*** to get the card (smart card or memory card) reset informations or UID informations. The present note is a mere description of the operation of commonly used functions in ***winscard.lib***. For other functions, please refer to the ***MSDN.***

1. SCardEstablishContext

This function establishes the resource manager contexts (the scope) within which database operations are performed.

LONG SCardEstablishContext (
IN DWORD *dwScope*,
IN LPCVOID *pvReserved1*,
IN LPCVOID *pvReserved2*,
OUT LPSCARDCONTEXT *phContext* **);**

Parameters

dwScope

Specifies the scope of the resource manager context. The following list shows the possible values for *dwScope*.

Value	Description
SCARD_SCOPE_SYSTEM	Database operations are performed within the domain of the system. The calling application must have appropriate access permissions for any database actions.
SCARD_SCOPE_USER	Unsupported. If specified, Windows CE assumes the SCARD_SCOPE_SYSTEM value.

pvReserved1

Void pointer reserved for future use; must be NULL. Reserved to enable a suitably privileged management application to act on behalf of another user.

pvReserved2

Void pointer reserved for future use; must be NULL. Reserved to enable a suitably privileged management application to act on behalf of another terminal.

phContext

Pointer to a handle to the established resource manager context. This handle can now be supplied to other functions attempting to do work within this context.

Return Values

Value	Description
Succeeds	SCARD_S_SUCCESS

Fails	An error value (see Smart Card Error values for a list of all error values)

Remarks

The context handle returned by **SCardEstablishContext** can be used by database query and management functions.

To release an established resource manager context, see **SCardReleaseContext**.

Requirements

Runs on	Versions	Defined in	Include	Link to
Windows CE OS	3.0 and later	Winscard.h		Winscard.lib

Note This API is part of the complete Windows CE OS package as provided by Microsoft. The functionality of a particular platform is determined by the original equipment manufacturer (OEM) and some devices may not support this API.

See Also

SCardReleaseContext

Example Code

```
SCARDCONTEXT       hSC;
LONG               lReturn;
// Establish the context.
lReturn = SCardEstablishContext(SCARD_SCOPE_USER,NULL,NULL,&hSC);
if (SCARD_S_SUCCESS != lReturn )
        printf("Failed SCardEstablishContext\n");
else
{
        // Use the context as needed. When done,
        // free the context by calling SCardReleaseContext.
        // …
}
```

2. SCardListReaders

The **SCardListReaders** function provides the list of readers within a set of named reader groups, eliminating duplicates.

The caller supplies a list of reader groups, and receives the list of readers within the named groups. Unrecognized group names are ignored.

```
LONG SCardListReaders(
   IN SCARDCONTEXT hContext,
   IN LPCTSTR mszGroups,
   OUT LPTSTR mszReaders,
   IN OUT LPDWORD pcchReaders
);
```

Parameters

hContext

Supplies the handle that identifies the resource manager context for the query. The resource manager context can be set by a previous call to **SCardEstablishContext.** On Windows XP and later, NULL can be passed to query in system scope.

mszGroups

Supplies the names of the reader groups defined to the system, as a multi-string. Use a NULL value to list all readers in the system (that is, the SCard$AllReaders group).

mszReaders

Receives a multi-string that lists the card readers within the supplied reader groups. If this value is NULL, **SCardListReaders** ignores the buffer length supplied in *pcchReaders*, writes the length of the buffer that would have been returned if this parameter had not been NULL to *pcchReaders*, and returns a success code.

pcchReaders

Supplies the length of the *mszReaders* buffer in characters, and receives the actual length of the multi-string structure, including all trailing Null characters. If the buffer length is specified as SCARD_AUTOALLOCATE, then *mszReaders* is converted to a pointer to a string pointer, and receives the address of a block of memory containing the multi-string structure. This block of memory must be deallocated with **SCardFreeMemory**.

Return Values

This function returns different values depending on whether it succeeds or fails.

Outcome	**Return value**
Success	SCARD_S_SUCCESS.
Group contains no readers	SCARD_E_NO_READERS_AVAILABLE
Other	An error code (see Error Codes for a list of all error codes).

Remarks

SCardListReaders is a database query function. For a description of other database query functions, see Smart Card Database Query Functions.

Example Code

```
LPTSTR              pmszReaders = NULL;
LPTSTR              pReader;
LONG                 lReturn, lReturn2;
DWORD                 cch = SCARD_AUTOALLOCATE;

// Retrieve the list the readers.
// hSC was set by a previous call to SCardEstablishContext.
lReturn = SCardListReaders(hSC,
                                NULL,
                                (LPTSTR)&pmszReaders,
                                &cch );
switch( lReturn )
{
    case SCARD_E_NO_READERS_AVAILABLE:
        printf("Reader is not in groups.\n");
        // Take appropriate action.
        // ...
        break;

    case SCARD_S_SUCCESS:
        // Do something with the multi string of readers.
        // Here, we'll merely output the values.
```

```
            // A double-null terminates the list of values.
            pReader = pmszReaders;
            while ( '\0' != *pReader )
            {
                // Display the value.
                printf("Reader: %S\n", pReader );
                // Advance to the next value.
                pReader = pReader + wcslen(pReader) + 1;
            }
            // Free the memory.
            lReturn2 = SCardFreeMemory( hSC,
                                        pmszReaders );
            if ( SCARD_S_SUCCESS != lReturn2 )
                printf("Failed SCardFreeMemory\n");
            break;

default:
            printf("Failed SCardListReaders\n");
            // Take appropriate action.
            // ...
            break;
}
```

3. SCardConnect

This function establishes a connection, using a specific resource manager context, between the calling application and a smart card contained by a specific reader. If no card exists in the specified reader, an error is returned.

LONG SCardConnect(
IN SCARDCONTEXT ***hContext*****,**
IN LPCTSTR ***szReader*****,**
IN DWORD ***dwShareMode*****,**
IN DWORD ***dwPreferredProtocols*****,**
OUT LPSCARDHANDLE ***phCard*****,**
OUT LPDWORD ***pdwActiveProtocol*** **);**

Parameters

hContext

Handle that identifies the resource manager context. The resource manager context is set by a previous call to **SCardEstablishContext**.

szReader

Null-terminated string that specifies the name of the reader containing the target card.

dwShareMode

Specifies a flag that indicates whether other applications can form connections to the card. The following list shows the possible values for *dwShareMode*.

Value	Description
SCARD_SHARE_SHARED	Unsupported.
SCARD_SHARE_EXCLUSIVE	This application is not willing to share the card with other applications.
SCARD_SHARE_DIRECT	Unsupported.

dwPreferredProtocols

Specifies a bit mask of acceptable protocols for the connection. The following list shows the possible values, which may be combined with the **OR** operation, for *dwPreferredProtocols*.

Value	Description
SCARD_PROTOCOL_T0	T=0 is an acceptable protocol.
SCARD_PROTOCOL_T1	T=1 is an acceptable protocol.
0	This parameter may be 0 only if *dwShareMode* is set to SCARD_SHARE_DIRECT. In this case, no protocol negotiation is performed by the drivers until an IOCTL_SMARTCARD_SET_PROTOCOL control directive is sent with **SCardControl**.

phCard

Pointer to a handle that identifies the connection to the smart card in the designated reader.

pdwActiveProtocol

Pointer to a **DWORD** that receives a flag indicating the established active protocol. The following list shows the possible values for *pdwActiveProtocol*.

Value	Description
SCARD_PROTOCOL_T0	T=0 is the active protocol.
SCARD_PROTOCOL_T1	T=1 is the active protocol.

SCARD_PROTOCOL_UNKNOWN	SCARD_SHARE_DIRECT has been specified, so that no protocol negotiation has occurred. It is possible that there is no card in the reader.

Return Values

Value	Description
Succeeds	SCARD_S_SUCCESS
Fails	An error value (see Smart Card Error values for a list of all error values)

Remarks

SCardConnect is a smart card and reader access function.

Only one active connection is allowed per context. If a second connection is required, a different context needs to be established and supplied to **SCardConnect**.

Requirements

Runs on	Versions	Defined in	Include	Link to
Windows CE OS	3.0 and later	Winscard.h		Winscard.lib

Note This API is part of the complete Windows CE OS package as provided by Microsoft. The functionality of a particular platform is determined by the original equipment manufacturer (OEM) and some devices may not support this API.

Example Code

```
SCARDHANDLE        hCardHandle;
LONG               lReturn;
DWORD              dwAP;

// Connect to the reader.
// hContext is a SCARDCONTEXT previously set by
// a call to SCardEstablishContext.
```

```
lReturn = SCardConnect( hContext,
                        L"Rainbow Technologies SCR3531 0",
                        SCARD_SHARE_SHARED,
                        SCARD_PROTOCOL_T0 | SCARD_PROTOCOL_T1,
                        &hCardHandle,
                        &dwAP );
if ( SCARD_S_SUCCESS != lReturn )
{
    printf("Failed SCardConnect\n");
    exit(1);    // Or other appropriate action.
}

// Use the connection; here we will merely display the
// active protocol.
switch ( dwAP )
{
    case SCARD_PROTOCOL_T0:
        printf("Active protocol T0\n");
        break;

    case SCARD_PROTOCOL_T1:
        printf("Active protocol T1\n");
        break;

    case SCARD_PROTOCOL_UNDEFINED:
    default:
        printf("Active protocol unnegotiated or unknown\n");
        break;
}

// Remember to disconnect (by calling SCardDisconnect).
// …
```

Note: In the JINMUYU rfid reader set, this function main is used to establish a connection between the application and the reader. Generally it is to establish a connection between the application and the smart card. In order to be compatible with more cards and to rich users' operations, only the reader's application software correctly connects with the reader. Then it all will return the correct information. So that users, without a smart card, can freely operate LCD, Beep and the memory card like MifareS50/S70.

4. SCardTransmit

The **SCardTransmit** function sends a service request to the smart card, and expects to receive data back from the card.

LONG SCardTransmit(
  IN SCARDHANDLE *hCard*,
  IN LPCSCARD_I0_REQUEST *pioSendPci*,
  IN LPCBYTE *pbSendBuffer*,
  IN DWORD *cbSendLength*,
  IN OUT LPSCARD_IO_REQUEST *pioRecvPci*,
  OUT LPBYTE *pbRecvBuffer*,
  IN OUT LPDWORD *pcbRecvLength*
);

Parameters

hCard

Supplies the reference value returned from **SCardConnect**.

pioSendPci

Pointer to the protocol header structure for the instruction. This buffer is in the format of an SCARD_IO_REQUEST structure, followed by the specific protocol control information (PCI).

For the T=0, T=1, and Raw protocols, the PCI structure is constant. The smart card subsystem supplies a global T=0, T=1, or Raw PCI structure, which you can reference by using the symbols SCARD_PCI_T0, SCARD_PCI_T1, and SCARD_PCI_RAW respectively.

pbSendBuffer

Pointer to the actual data to be written to the card.

T=0 Note For T=0, the data parameters are placed into the *pbSendBuffer* according to the following structure:

```
struct {
      BYTE
            bCla,    // The instruction class
            bIns,    // The instruction code
            bP1,     // Parameter to the instruction
            bP2,     // Parameter to the instruction
            bP3;     // Size of I/O Transfer
} CmdBytes;
```

Members

bCla
The T=0 instruction class
bIns
An instruction code in the T=0 instruction class
bP1, bP2
Reference codes completing the instruction code
bP3
The number of data bytes which are to be transmitted during the command, per ISO 7816-4, Section 8.2.1.

The data sent to the card should immediately follow the send buffer. In the special case where no data is sent to the card and no data is expected in return, **bP3** is not sent.

cbSendLength

Supplies the length (in bytes)of the *pbSendBuffer* parameter.

T=0 Note For T=0, in the special case where no data is sent to the card and no data expected in return, this length must reflect that the **bP3** member is not being sent: the length should be sizeof(CmdBytes) – sizeof(BYTE).

pioRecvPci

Pointer to the protocol header structure for the instruction, followed by a buffer in which to receive any returned protocol control information (PCI) specific to the protocol in use. This parameter may be NULL if no returned PCI is desired.

pbRecvBuffer

Pointer to any data returned from the card.

T=0 Note For T=0, the data is immediately followed by the SW1 and SW2 status bytes. If no data is returned from the card, then this buffer will only contain the SW1 and SW2 status bytes.

pcbRecvLength

Supplies the length of the *pbRecvBuffer* parameter (in bytes) and receives the actual number of bytes received from the smart card.

T=0 Note For T=0, the receive buffer must be at least two bytes long, in order to receive the SW1 and SW2 status bytes.If this buffer length is specified as SCARD_AUTOALLOCATE, then *pbReceiveBuffer* is converted to a pointer to a string pointer and receives the address of a block of memory containing the structure.

Return Values

This function returns different values depending on whether it succeeds or fails.

Outcome	**Return value**
Success	SCARD_S_SUCCESS.
Failure	An error code (see Error Codes for a list of all error codes).

Remarks

SCardTransmit is a smart card and reader access function. For a description of other access functions, see Smart Card and Reader Access Functions.

T=0 Protocol Remarks

For the T=0 protocol, the data received back are the SW1 and SW2 status codes, possibly preceded by response data. The following paragraphs provide information on the send and receive buffers used to transfer data and issue a command.

Sending Data to the Card

To send *n* bytes of data to the card, where *n*>0, the send and receive buffers must be formatted as follows.

The first four bytes of the *pbSendBuffer* buffer contain the CLA, INS, P1, and P2 values for the T=0 operation. The fifth byte shall be set to *n*: the size (in bytes) of the data to be transferred to the card. The next *n* bytes shall contain the data to be sent to the card.

The *cbSendLength* parameter shall be set to the size of the T=0 header information (CLA, INS, P1 and P2) plus a byte containing the length of the data to be transferred (*n*), plus the size of data to be sent. In this example, this is *n*+5.

The *pbRecvBuffer* will receive the SW1 and SW2 status codes from the operation.

The *pcbRecvLength* should be at least 2, and will be set to 2 upon return.

Obtaining Data from the Card

To receive *n*>0 bytes of data from the card, the send and receive buffers must be formatted as follows.

The first four bytes of the *pbSendBuffer* buffer contain the CLA, INS, P1, and P2 values for the T=0 operation. The fifth byte shall be set to *n*: the size (in bytes) of the data to be transferred from the card. If 256 bytes are to be transferred from the card, then this byte shall be set to zero.

The *cbSendLength* parameter shall be set to 5, the size of the T=0 header information.

The *pbRecvBuffer* will receive the data returned from the card, immediately followed by the SW1 and SW2 status codes from the operation.

The *pcbRecvLength* should be at least *n*+2, and will be set to *n*+2 upon return.

Issuing a Command Without Exchanging Data

To issue a command to the card that does not involve the exchange of data (either sent or received), the send and receive buffers must be formatted as follows.

The *pbSendBuffer* buffer shall contain the CLA, INS, P1, and P2 values for the T=0 operation. The P3 value is not sent. (This is to differentiate the header from the case where 256 bytes are expected to be returned.)

The *cbSendLength* parameter shall be set to 4, the size of the T=0 header information (CLA, INS, P1, and P2).

The *pbRecvBuffer* will receive the SW1 and SW2 status codes from the operation.

The *pcbRecvLength* should be at least 2, and will be set to 2 upon return.

Example Code

```
// Transmit the request.
// lReturn is of type LONG.
// hCardHandle was set by a previous call to SCardConnect.
// pbSend points to the buffer of bytes to send.
// dwSend is the DWORD value for the number of bytes to send.
// pbRecv points to the buffer for returned bytes.
// dwRecv is the DWORD value for the number of returned bytes.
lReturn = SCardTransmit(hCardHandle,
                        SCARD_PCI_T0,
                        pbSend,
                        dwSend,
                        NULL,
                        pbRecv,
                        &dwRecv );
```

```
if ( SCARD_S_SUCCESS != lReturn )
{
	printf("Failed SCardTransmit\n");
	exit(1);	// Or other appropriate error action.
}
```

5. SCardDisconnect

The **SCardDisconnect** function terminates a connection previously opened between the calling application and a smart card in the target reader.

```
LONG SCardDisconnect(
  IN SCARDHANDLE hCard,
  IN DWORD dwDisposition
);
```

Parameters

hCard

Supplies the reference value obtained from a previous call to **SCardConnect**.

dwDisposition

Indicates what to do with the card in the connected reader on close. The following are the possible values.

Value	Meaning
SCARD_LEAVE_CARD	Don't do anything special.
SCARD_RESET_CARD	Reset the card.
SCARD_UNPOWER_CARD	Power down the card.
SCARD_EJECT_CARD	Eject the card.

Return Values

This function returns different values depending on whether it succeeds or fails.

Outcome	Return value
Success	SCARD_S_SUCCESS.
Failure	An error code (see Error Codes for a list of all error codes).

Remarks

If an application (which previously called **SCardConnect**) exits without calling **SCardDisconnect**, the card is automatically reset.

SCardDisconnect is a smart card and reader access function. For a description of other access functions, see Smart Card and Reader Access Functions.

Example Code

```
// Terminate the connection.
// lReturn is a LONG variable.
// hCardHandle was set by a previous call to SCardConnect.
lReturn = SCardDisconnect(hCardHandle,
                          SCARD_LEAVE_CARD);
if ( SCARD_S_SUCCESS != lReturn )
{
    printf("Failed SCardDisconnect\n");
    exit(1);    // Or other appropriate action.
}
```

Note: The JINMUYU rfid readers connect not a smart card but the card reader. If to set the reader, this option is invalid.

6. SCardReleaseContext

The **SCardReleaseContext** function closes an established resource manager context, freeing any resources allocated under that context, including SCARDHANDLE objects and memory allocated using the SCARD_AUTOALLOCATE length designator.

LONG SCardReleaseContext(
IN SCARDCONTEXT *hContext*
);

Parameters

hContext

Supplies the handle that identifies the resource manager context. The resource manager context is set by a previous call to **SCardEstablishContext**.

Return Values

This function returns different values depending on whether it succeeds or fails.

Outcome	**Return value**
Success	SCARD_S_SUCCESS.
Failure	An error code (see Error Codes for a list of all error codes).

Example Code

```
// Free the context.
// lReturn is of type LONG.
// hSC was set by an earlier call to SCardEstablishContext.
lReturn = SCardReleaseContext(hSC);
if ( SCARD_S_SUCCESS != lReturn )
        printf("Failed SCardReleaseContext\n");
```