MUSCLE PC/SC Lite API
Toolkit API Reference Documentation

David Corcoran & Ludovic Rousseau
corcoran@linuxnet.com, ludovic.rousseau@free.fr

May 26, 2004

Abstract

This toolkit and documentation is provided on an *as is* basis. The authors shall not be held responsible for any mishaps caused by the use of this software.
For more information please visit http://www.musclecard.com/.

Document history:

0.8.7	March 8, 2001	latest PDF only version
0.9.0	May 26, 2004	reformat using LaTeX, correct bugs and add parts 4 and 5
0.9.1	Jan 10, 2007	add SCardIsValidContext()

Contents

1 Introduction/Overview

This document contains the reference API calls for communicating to the MUSCLE PC/SC Smart Card Resource Manager. PC/SC is a standard proposed by the PC/SC workgroup [3] which is a conglomerate of representative from major smart card manufacturers and other companies. This specification tries to abstract the smart card layer into a high level API so that smart cards and their readers can be accessed in a homogeneous fashion.

This toolkit was written in ANSI C that can be used with most compilers and does NOT use complex and large data structures such as vectors, *etc.* The C API emulates the winscard API that is used on the Windows platform. It is contained in the library `libpcsclite.so` that is linked to your application.

I would really like to hear from you. If you have any feedback either on this documentation or on the MUSCLE project please feel free to email me at: `corcoran@musclecard.com`.

2 Definitions

2.1 Defined types

The following is a list of commonly used type definitions in the following API. These definitions and more can be found in the `include/pcsclite.h` file.

PC/SC type	C type
BOOL	short
BYTE	unsigned char
DWORD	unsigned long
LONG	long
LPBYTE	unsigned char *
LPCBYTE	const unsigned char *
LPCSTR	const char *
LPCVOID	const void *
LPCWSTR	char *
LPDWORD	unsigned long *
LPSCARDCONTEXT	unsigned long *
LPSCARDHANDLE	unsigned long *
LPSTR	char *
LPVOID	void *
PSCARDCONTEXT	unsigned long *
PSCARDHANDLE	unsigned long *
RESPONSECODE	long
SCARDCONTEXT	unsigned long
SCARDHANDLE	unsigned long
ULONG	unsigned long

USHORT	unsigned short
WORD	unsigned long

2.2 Error codes

The following is a list of commonly used errors. Since different cards produce different errors they must map over to these error messages.

SCARD_S_SUCCESS
SCARD_E_CANCELLED
SCARD_E_CANT_DISPOSE
SCARD_E_CARD_UNSUPPORTED
SCARD_E_DUPLICATE_READER
SCARD_E_INSUFFICIENT_BUFFER
SCARD_E_INVALID_ATR
SCARD_E_INVALID_HANDLE
SCARD_E_INVALID_PARAMETER
SCARD_E_INVALID_TARGET
SCARD_E_INVALID_VALUE
SCARD_E_NO_MEMORY
SCARD_E_NO_SERVICE
SCARD_E_NO_SMARTCARD
SCARD_E_NOT_READY
SCARD_E_NOT_TRANSACTED
SCARD_E_PCI_TOO_SMALL
SCARD_E_PROTO_MISMATCH
SCARD_E_READER_UNAVAILABLE
SCARD_E_READER_UNSUPPORTED
SCARD_E_SERVICE_STOPPED
SCARD_E_SHARING_VIOLATION
SCARD_E_SYSTEM_CANCELLED
SCARD_E_TIMEOUT
SCARD_E_UNKNOWN_CARD
SCARD_E_UNKNOWN_READER
SCARD_F_COMM_ERROR
SCARD_F_INTERNAL_ERROR
SCARD_F_UNKNOWN_ERROR
SCARD_F_WAITED_TOO_LONG
SCARD_W_UNSUPPORTED_CARD
SCARD_W_UNRESPONSIVE_CARD
SCARD_W_UNPOWERED_CARD
SCARD_W_RESET_CARD
SCARD_W_REMOVED_CARD

3 API Routines

These routines specified here are winscard routines like those in the winscard API provided under Windows®. These are compatible with the Microsoft® API calls. This list of calls is mainly an abstraction of readers. It gives a common API for communication to most readers in a homogeneous fashion.

Since all functions can produce a wide array of errors, please refer to § 2.2 on the preceding page for a list of error returns.

For a human readable representation of an error the function `pcsc_stringify_error()` is declared in `pcsclite.h`. This function is not available on Microsoft® winscard API and is pcsc-lite specific.

3.1 SCardEstablishContext

Synopsis:

```
#include <winscard.h>

LONG SCardEstablishContext(DWORD dwScope,
    LPCVOID pvReserved1,
    LPCVOID pvReserved2,
    LPSCARDCONTEXT phContext);
```

Parameters:

`dwScope`	IN	Scope of the establishment This can either be a local or remote connection
`pvReserved1`	IN	Reserved for future use. Can be used for remote connection
`pvReserved2`	IN	Reserved for future use
`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

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,
    LPCSTR mszGroups,
    LPSTR mszReaders,
    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`	INOUT	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 nul 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

3.5 SCardListReaderGroups

Synopsis:

```
#include <winscard.h>

LONG SCardListReaderGroups(SCARDCONTEXT hContext,
    LPSTR mszGroups,
    LPDWORD pcchGroups);
```

Parameters:

`hContext`	IN	Connection context to the PC/SC Resource Manager
`mszGroups`	OUT	List of groups to list readers
`pcchGroups`	INOUT	Size of multi-string buffer including NULL's

Description:

This function returns a list of currently available reader groups on the system. `mszGroups` is a pointer to a character string that is allocated by the application. If the application sends `mszGroups` as `NULL` then this function will return the size of the buffer needed to allocate in `pcchGroups`.

The group names is a multi-string and separated by a nul character (`'\0'`) and ended by a double nul character. `"SCard$DefaultReaders\0Group 2\0\0"`.

Example:

```
SCARDCONTEXT hContext;
LPSTR mszGroups;
DWORD dwGroups;
LONG rv;

rv = SCardEstablishContext(SCARD_SCOPE_SYSTEM, NULL, NULL, &hContext);
rv = SCardListReaderGroups(hContext, NULL, &dwGroups);
mszGroups = malloc(sizeof(char)*dwGroups);
rv = SCardListReaderGroups(hContext, mszGroups, &dwGroups);
```

Returns:

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

3.6 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_VALUE	Invalid sharing mode, requested protocol, or reader name
SCARD_E_NOT_READY	Could not allocate the desired port
SCARD_E_READER_UNAVAILABLE	Could not power up the reader or card
SCARD_E_SHARING_VIOLATION	Someone else has exclusive rights
SCARD_E_UNSUPPORTED_FEATURE	Protocol not supported

3.7 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, &dwActiveProtocol);
}
```

Returns:

SCARD_S_SUCCESS	Successful
SCARD_E_INVALID_HANDLE	Invalid hCard handle
SCARD_E_NOT_READY	Could not allocate the desired port
SCARD_E_INVALID_VALUE	Invalid sharing mode, requested protocol, or reader name
SCARD_E_READER_UNAVAILABLE	The reader has been removed
SCARD_E_UNSUPPORTED_FEATURE	Protocol not supported
SCARD_E_SHARING_VIOLATION	Someone else has exclusive rights

3.8 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

3.9 SCardBeginTransaction

Synopsis:

```
#include <winscard.h>

LONG SCardBeginTransaction(SCARDHANDLE hCard);
```

Parameters:

hCard	IN	Connection made from SCardConnect

Description:

This function establishes a temporary exclusive access mode for doing a series of commands or transaction. You might want to use this when you are selecting a few files and then writing a large file so you can make sure that another application will not change the current file. If another application has a lock on this reader or this application is in SCARD_SHARE_EXCLUSIVE there will be no action taken.

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 = SCardBeginTransaction(hCard);

/* Do some transmit commands */
```

Returns:

SCARD_S_SUCCESS	Successful
SCARD_E_INVALID_HANDLE	Invalid hCard handle
SCARD_E_SHARING_VIOLATION	Someone else has exclusive rights
SCARD_E_READER_UNAVAILABLE	The reader has been removed

3.10 SCardEndTransaction

Synopsis:

```
#include <winscard.h>

LONG SCardEndTransaction(SCARDHANDLE hCard,
    DWORD dwDisposition);
```

Parameters:

hCard	IN	Connection made from SCardConnect
dwDisposition	IN	Action to be taken on the reader

Description:

This function ends a previously begun transaction. The calling application must be the owner of the previously begun transaction or an error will occur. dwDisposition can have the following values: The disposition action is not currently used in this release.

Value of dwDisposition	Meaning
SCARD_LEAVE_CARD	Do nothing
SCARD_RESET_CARD	Reset the card
SCARD_UNPOWER_CARD	Unpower the card
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 = SCardBeginTransaction(hCard);
```

```
/* Do some transmit commands */

rv = SCardEndTransaction(hCard, SCARD_LEAVE_CARD);
```

Returns:

SCARD_S_SUCCESS	Successful
SCARD_E_INVALID_HANDLE	Invalid hCard handle
SCARD_E_SHARING_VIOLATION	Someone else has exclusive rights
SCARD_E_READER_UNAVAILABLE	The reader has been removed

3.11 SCardTransmit

Synopsis:

```
#include <winscard.h>

LONG SCardTransmit(SCARDHANDLE hCard,
    LPCSCARD_IO_REQUEST pioSendPci,
    LPCBYTE pbSendBuffer,
    DWORD cbSendLength,
    LPSCARD_IO_REQUEST pioRecvPci,
    LPBYTE pbRecvBuffer,
    LPDWORD pcbRecvLength);
```

Parameters:

hCard	IN	Connection made from SCardConnect
pioSendPci	INOUT	Structure of protocol information
pbSendBuffer	IN	APDU to send to the card
cbSendLength	IN	Length of the APDU
pioRecvPci	INOUT	Structure of protocol information
pbRecvBuffer	OUT	Response from the card
pcbRecvLength	INOUT	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_NOT_TRANSACTED`	APDU exchange not successful
`SCARD_E_PROTO_MISMATCH`	Connect protocol is different than desired
`SCARD_E_INVALID_VALUE`	Invalid Protocol, reader name, etc
`SCARD_E_READER_UNAVAILABLE`	The reader has been removed
`SCARD_W_RESET_CARD`	The card has been reset by another application
`SCARD_W_REMOVED_CARD`	The card has been removed from the reader

3.12 SCardControl

Synopsis:

```
#include <winscard.h>
```

```
LONG SCardControl(SCARDHANDLE hCard,
    DWORD dwControlCode,
    LPCVOID pbSendBuffer,
    DWORD cbSendLength,
    LPVOID pbRecvBuffer,
    DWORD pcbRecvLength,
    LPDWORD lpBytesReturned);
```

Parameters:

`hCard`	IN	Connection made from `SCardConnect`
`dwControlCode`	IN	Control code for the operation
`pbSendBuffer`	IN	Command to send to the reader
`cbSendLength`	IN	Length of the command
`pbRecvBuffer`	OUT	Response from the reader
`pcbRecvLength`	IN	Length of the response buffer
`lpBytesReturned`	OUT	Length of the response

Description:

This function sends a command directly to the IFD Handler to be processed by the reader. This is useful for creating client side reader drivers for functions like PIN pads, biometrics, or other extensions to the normal smart card reader that are not normally handled by PC/SC.

Note: the API of this function changed. In pcsc-lite 1.2.0 and before the API was not Windows® PC/SC compatible. This has been corrected.

see § 5 for a list of supported commands by some drivers.

Example:

```
LONG rv;
SCARDCONTEXT hContext;
SCARDHANDLE hCard;
DWORD dwActiveProtocol, dwSendLength, dwRecvLength;
BYTE pbRecvBuffer[10];
BYTE pbSendBuffer[] = { 0x06, 0x00, 0x0A, 0x01, 0x01, 0x10 0x00 };

rv = SCardEstablishContext(SCARD_SCOPE_SYSTEM, NULL, NULL, &hContext);
rv = SCardConnect(hContext, "Reader X", SCARD_SHARE_SHARED,
         SCARD_PROTOCOL_RAW &hCard, &dwActiveProtocol);
dwSendLength = sizeof(pbSendBuffer);
dwRecvLength = sizeof(pbRecvBuffer);
```

```
rv = SCardControl(hCard, 0x42000001, pbSendBuffer, dwSendLength,
         pbRecvBuffer, sizeof(pbRecvBuffer), &dwRecvLength);
```

Returns:

SCARD_S_SUCCESS	Successful
SCARD_E_NOT_TRANSACTED	Data exchange not successful
SCARD_E_INVALID_HANDLE	Invalid hCard handle
SCARD_E_INVALID_VALUE	Invalid value was presented
SCARD_E_READER_UNAVAILABLE	The reader has been removed
SCARD_W_RESET_CARD	The card has been reset by another application
SCARD_W_REMOVED_CARD	The card has been removed from the reader

3.13 SCardStatus

Synopsis:

```
#include <winscard.h>

LONG SCardStatus(SCARDHANDLE hCard,
    LPSTR szReaderName,
    LPDWORD pcchReaderLen,
    LPDWORD pdwState,
    LPDWORD pdwProtocol,
    LPBYTE pbAtr,
    LPDWORD pcbAtrLen);
```

Parameters:

hCard	IN	Connection made from SCardConnect
szReaderName	INOUT	Friendly name of this reader
pcchReaderLen	INOUT	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 szReaderName. pcchReaderLen will be the size of the allocated buffer for szReaderName, 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_INVALID_HANDLE`	Invalid `hCard` handle
`SCARD_E_INSUFFICIENT_BUFFER`	Not enough allocated memory for `szReaderName` or for `pbAtr`
`SCARD_E_READER_UNAVAILABLE`	The reader has been removed

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`	INOUT	Structures of readers with current states
`cReaders`	IN	Number of structures

Description:

This function receives a structure or list of structures containing reader names. It then blocks for a change in state to occur on any of the OR'd values contained in `dwCurrentState` for a maximum blocking time of `dwTimeout` or forever if `INFINITE` is used. The new event state will be contained in `dwEventState`. A status change might be a card insertion or removal event, a change in ATR, *etc.*

This function will block for reader availability if `cReaders` is equal to zero and `rgReaderStates` is `NULL`.

```
typedef struct {
    LPCSTR szReader;     /* Reader name                        */
    LPVOID pvUserData;    /* User defined data                  */
    DWORD dwCurrentState; /* Current state of reader            */
    DWORD dwEventState;   /* Reader state after a state change */
    DWORD cbAtr;          /* ATR Length, usually MAX_ATR_SIZE  */
    BYTE rgbAtr[MAX_ATR_SIZE]; /* ATR Value                     */
} SCARD_READERSTATE;

typedef SCARD_READERSTATE *PSCARD_READERSTATE, **LPSCARD_READERSTATE;
```

Value of dwCurrentState and dwEventState	Meaning
SCARD_STATE_UNAWARE	The application is unaware of the current state, and would like to know. The use of this value results in an immediate return from state transition monitoring services. This is represented by all bits set to zero
SCARD_STATE_IGNORE	This reader should be ignored
SCARD_STATE_CHANGED	There is a difference between the state believed by the application, and the state known by the resource manager. When this bit is set, the application may assume a significant state change has occurred on this reader
SCARD_STATE_UNKNOWN	The given reader name is not recognized by the resource manager. If this bit is set, then SCARD_STATE_CHANGED and SCARD_STATE_IGNORE will also be set

Value of dwCurrentState and dwEventState	Meaning
SCARD_STATE_UNAVAILABLE	The actual state of this reader is not available. If this bit is set, then all the following bits are clear
SCARD_STATE_EMPTY	There is no card in the reader. If this bit is set, all the following bits will be clear
SCARD_STATE_PRESENT	There is a card in the reader
SCARD_STATE_ATRMATCH	There is a card in the reader with an ATR matching one of the target cards. If this bit is set, SCARD_STATE_PRESENT will also be set. This bit is only returned on the SCardLocateCards function
SCARD_STATE_EXCLUSIVE	The card in the reader is allocated for exclusive use by another application. If this bit is set, SCARD_STATE_PRESENT will also be set
SCARD_STATE_INUSE	The card in the reader is in use by one or more other applications, but may be connected to in shared mode. If this bit is set, SCARD_STATE_PRESENT will also be set
SCARD_STATE_MUTE	There is an unresponsive card in the reader

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_INVALID_VALUE`	Invalid States, reader name, etc
`SCARD_E_INVALID_HANDLE`	Invalid `hContext` handle
`SCARD_E_READER_UNAVAILABLE`	The reader is unavailable

3.15 SCardCancel

Synopsis:

```
#include <winscard.h>

LONG SCardCancel(SCARDCONTEXT hContext);
```

Parameters:

`hContext`	IN	Connection context to the PC/SC Resource Manager

Description:

This function cancels all pending blocking requests on the `GetStatusChange()` function.

Example:

```
SCARDCONTEXT hContext;
DWORD cReaders;
SCARD_READERSTATE rgReaderStates;
LONG rv;

rv = SCardEstablishContext(SCARD_SCOPE_SYSTEM, NULL, NULL, &hContext);

rgReaderStates.szReader = strdup("Reader X");
rgReaderStates.dwCurrentState = SCARD_STATE_EMPTY;

/* Spawn off thread for following function */
rv = SCardGetStatusChange(hContext, 0, rgReaderStates, cReaders);

rv = SCardCancel(hContext);
```

Returns:

`SCARD_S_SUCCESS`	Successful
`SCARD_E_INVALID_HANDLE`	Invalid `hContext` handle

3.16 SCardSetTimeout

Synopsis:

```
#include <winscard.h>

LONG SCardSetTimeout(SCARDCONTEXT hContext,
    DWORD dwTimeout);
```

Parameters:

`hContext`	IN	Connection context to the PC/SC Resource Manager
`dwTimeout`	IN	New timeout value

Description:

This function is not in Microsoft® WinSCard API and is deprecated in pcsc-lite API.

The function does not do anything except returning `SCARD_S_SUCCESS`.

Returns:

`SCARD_S_SUCCESS`	Successful

3.17 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 available in the file `pcsclite.h`.

- `SCARD_ATTR_ASYNC_PROTOCOL_TYPES`
- `SCARD_ATTR_ATR_STRING`
- `SCARD_ATTR_CHANNEL_ID`
- `SCARD_ATTR_CHARACTERISTICS`
- `SCARD_ATTR_CURRENT_BWT`
- `SCARD_ATTR_CURRENT_CLK`
- `SCARD_ATTR_CURRENT_CWT`
- `SCARD_ATTR_CURRENT_D`
- `SCARD_ATTR_CURRENT_EBC_ENCODING`
- `SCARD_ATTR_CURRENT_F`
- `SCARD_ATTR_CURRENT_IFSC`
- `SCARD_ATTR_CURRENT_IFSD`
- `SCARD_ATTR_CURRENT_IO_STATE`
- `SCARD_ATTR_CURRENT_N`
- `SCARD_ATTR_CURRENT_PROTOCOL_TYPE`
- `SCARD_ATTR_CURRENT_W`
- `SCARD_ATTR_DEFAULT_CLK`
- `SCARD_ATTR_DEFAULT_DATA_RATE`
- `SCARD_ATTR_DEVICE_FRIENDLY_NAME_A`
- `SCARD_ATTR_DEVICE_FRIENDLY_NAME_W`

- SCARD_ATTR_DEVICE_IN_USE
- SCARD_ATTR_DEVICE_SYSTEM_NAME_A
- SCARD_ATTR_DEVICE_SYSTEM_NAME_W
- SCARD_ATTR_DEVICE_UNIT
- SCARD_ATTR_ESC_AUTHREQUEST
- SCARD_ATTR_ESC_CANCEL
- SCARD_ATTR_ESC_RESET
- SCARD_ATTR_EXTENDED_BWT
- SCARD_ATTR_ICC_INTERFACE_STATUS
- SCARD_ATTR_ICC_PRESENCE
- SCARD_ATTR_ICC_TYPE_PER_ATR
- SCARD_ATTR_MAX_CLK
- SCARD_ATTR_MAX_DATA_RATE
- SCARD_ATTR_MAX_IFSD
- SCARD_ATTR_MAXINPUT
- SCARD_ATTR_POWER_MGMT_SUPPORT
- SCARD_ATTR_SUPRESS_T1_IFS_REQUEST
- SCARD_ATTR_SYNC_PROTOCOL_TYPES
- SCARD_ATTR_USER_AUTH_INPUT_DEVICE
- SCARD_ATTR_USER_TO_CARD_AUTH_DEVICE
- SCARD_ATTR_VENDOR_IFD_SERIAL_NO
- SCARD_ATTR_VENDOR_IFD_TYPE
- SCARD_ATTR_VENDOR_IFD_VERSION
- SCARD_ATTR_VENDOR_NAME

Not all the `dwAttrId` values listed above may be implemented in the IFD Handler you are using. And some `dwAttrId` values not listed here may be implemented.

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_NOT_TRANSACTED`	Data exchange not successful
`SCARD_E_INSUFFICIENT_BUFFER`	Reader buffer not large enough

3.18 SCardSetAttrib

Synopsis:

```
#include <winscard.h>

LONG SCardSetAttrib(SCARDHANDLE hCard,
    DWORD dwAttrId,
    LPCBYTE pbAttr,
    DWORD cbAttrLen);
```

Parameters:

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

Description:

This function set an attribute of the IFD Handler. The list of attributes you can set is dependent on the IFD Handler you are using.

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 = SCardSetAttrib(hCard, 0x42000001, "\x12\x34\x56", 3);
```

Returns:

SCARD_S_SUCCESS	Successful
SCARD_E_NOT_TRANSACTED	Data exchange not successful

3.19 pcsc_stringify_error

Synopsis:

```
#include <pcsclite.h>

char *pcsc_stringify_error(long error);
```

Description:

This function return a human readable text for the given PC/SC error code.

Example:

```
SCARDCONTEXT hContext;
LONG rv;

rv = SCardEstablishContext(SCARD\_SCOPE\_SYSTEM, NULL, NULL, &hContext);
if (rv != SCARD_S_SUCCESS)
    printf("SCardEstablishContext: %s (0x%lX)\n",
        pcsc_stringify_error(rv), rv);
```

3.20 log_msg and log_xxd

The `pcscd` daemon (part of pcsc-lite) provides two functions that can be used to log debug messages. You should not use `log_msg()` directly but use the macros defined in `/usr/include/PCSC/debuglog.h`.

These logging functions are used by some IFD handlers (smart card driver) like the CCID driver `http://pcsclite.alioth.debian.org/ccid.html` to benefit from the log framework offered by `pcscd` (the daemon). With these functions it is easy to change the log level (debug, info, error or critical) and the log output (syslog or stderr) without recompiling the driver.

4 Multithreading and contexts

From version 1.2.0 pcsc-lite is much more multithreading friendly.

You have to follow some rules:

- For security reasons, a context can only be released (using `SCardReleaseContext()`) by the thread that created it.
- To access different readers (*i.e.* cards) in different threads, each thread must use a different context (not necessarily created by this thread itself).

Each thread should create his own context with `SCardEstablishContext()` and should release it with `SCardReleaseContext()` when the context is not necessary any more.

If different threads share a same context, the calls to different functions of the pcsc-lite API are stored in a queue and the executions serialised for this context because there is a mutex shared for all the (critical) operations of this context.

Note: The SCF (Smart Card Framework) used by Solaris has not been updated. So if you compile pcsc-lite using `./configure -enable-scf` you will still have a global lock mechanism.

5 Some SCardControl commands

The commands described here may not be implemented by all the drivers. They are implemented by the CCID driver available at `http://pcsclite.alioth.debian.org/ccid.html` and maybe some other.

The tag names used by these functions are `IOCTL_SMARTCARD_VENDOR_*`. They are vendor (driver) specific.

5.1 IFD_EXCHANGE

This command is used to send a proprietary command to a reader.

The CCID specification [1] describes a `PC_to_RDR_Escape` command to send proprietary commands to the reader.

Example:

```
#include <winscard.h>
#include <reader.h>

#define IOCTL_SMARTCARD_VENDOR_IFD_EXCHANGE SCARD_CTL_CODE(1)

SCARDHANDLE hCard;
unsigned char bSendBuffer[MAX_BUFFER_SIZE];
unsigned char bRecvBuffer[MAX_BUFFER_SIZE];
DWORD length;

/* get firmware */
bSendBuffer[0] = 0x02; /* proprietary code for Gemplus CCID readers */
rv = SCardControl(hCard, IOCTL_SMARTCARD_VENDOR_IFD_EXCHANGE,
    bSendBuffer, 1, bRecvBuffer, sizeof(bRecvBuffer), &length);

printf(" Firmware: ");
for (i=0; i<length; i++)
printf("%02X ", bRecvBuffer[i]);
printf("\n");
```

5.2 VERIFY_PIN and MODIFY_PIN

The CCID specification [1] describes a `PC_to_RDR_Secure` command to perform a PIN verification or PIN modification without sending the PIN to the host. The reader must have a keyboard, and optionnaly a display.

The command format is described in the PCSCv2 part 10 specifications [2].

The `bSendBuffer` to pass to `SCardControl()` contains:

- the CCID `abPINDataStructure`

  This is the CCID structure used to parameter the PIN verification command.

- the VERIFY APDU

  That is the APDU sent to the card with the PIN code values replaced by the actually entered PIN code. See the CCID specification [1] for a more precise descruption.

Example:

```
#include <winscard.h>
#include <reader.h>

LONG rv;
SCARDHANDLE hCard;
unsigned char bSendBuffer[MAX_BUFFER_SIZE];
unsigned char bRecvBuffer[MAX_BUFFER_SIZE];
DWORD verify_ioctl = 0;
DWORD modify_ioctl = 0;
PIN_VERIFY_STRUCTURE *pin_verify;

/* does the reader support PIN verification? */
rv = SCardControl(hCard, CM_IOCTL_GET_FEATURE_REQUEST, NULL, 0,
    bRecvBuffer, sizeof(bRecvBuffer), &length);

/* get the number of elements instead of the complete size */
length /= sizeof(PCSC_TLV_STRUCTURE);

pcsc_tlv = (PCSC_TLV_STRUCTURE *)bRecvBuffer;
for (i = 0; i < length; i++)
{
    if (pcsc_tlv[i].tag == FEATURE_VERIFY_PIN_DIRECT)
        verify_ioctl = pcsc_tlv[i].value;
    if (pcsc_tlv[i].tag == FEATURE_MODIFY_PIN_DIRECT)
        modify_ioctl = pcsc_tlv[i].value;
}

if (0 == verify_ioctl)
{
    printf("Reader %s does not support PIN verification\n",
        readers[reader_nb]);
    return;
}

pin_verify = (PIN_VERIFY_STRUCTURE *)bSendBuffer;

/* PC/SC v2.0.2 Part 10 PIN verification data structure */
pin_verify -> bTimerOut = 0x00;
pin_verify -> bTimerOut2 = 0x00;
pin_verify -> bmFormatString = 0x82;
pin_verify -> bmPINBlockString = 0x04;
pin_verify -> bmPINLengthFormat = 0x00;
pin_verify -> wPINMaxExtraDigit = HOST_TO_CCID_16(0x0408); /* Min Max */
pin_verify -> bEntryValidationCondition = 0x02; /* validation key pressed */
```

```
pin_verify -> bNumberMessage = 0x01;
pin_verify -> wLangId = HOST_TO_CCID_16(0x0904);
pin_verify -> bMsgIndex = 0x00;
pin_verify -> bTeoPrologue[0] = 0x00;
pin_verify -> bTeoPrologue[1] = 0x00;
pin_verify -> bTeoPrologue[2] = 0x00;
/* pin_verify -> ulDataLength = 0x00; we don't know the size yet */

/* APDU: 00 20 00 00 08 30 30 30 30 00 00 00 00 */
offset = 0;
pin_verify -> abData[offset++] = 0x00;  /* CLA */
pin_verify -> abData[offset++] = 0x20;  /* INS: VERIFY */
pin_verify -> abData[offset++] = 0x00;  /* P1 */
pin_verify -> abData[offset++] = 0x00;  /* P2 */
pin_verify -> abData[offset++] = 0x08;  /* Lc: 8 data bytes */
pin_verify -> abData[offset++] = 0x30;  /* '0' */
pin_verify -> abData[offset++] = 0x30;  /* '0' */
pin_verify -> abData[offset++] = 0x30;  /* '0' */
pin_verify -> abData[offset++] = 0x30;  /* '0' */
pin_verify -> abData[offset++] = 0x00;  /* '\0' */
pin_verify -> abData[offset++] = 0x00;  /* '\0' */
pin_verify -> abData[offset++] = 0x00;  /* '\0' */
pin_verify -> abData[offset++] = 0x00;  /* '\0' */
pin_verify -> ulDataLength = HOST_TO_CCID_32(offset);   /* APDU size */

length = sizeof(PIN_VERIFY_STRUCTURE) + offset -1;
    /* -1 because PIN_VERIFY_STRUCTURE contains the first byte of abData[] */

printf("Enter your PIN: ");
fflush(stdout);

rv = SCardControl(hCard, verify_ioctl, bSendBuffer,
    length, bRecvBuffer, sizeof(bRecvBuffer), &length);
```

References

[1] Universal Serial Bus, Device Class Specification for USB Chip/Smart Card Interface Devices, 20 March 2001. Revision 1.00, http://www.usb.org/developers/devclass_docs/ccid_classspec_1_00a.pdf.

[2] Interoperability Specification for ICCs and Personal Computer Systems, Part 10 IFDs with Secure Pin Entry Capabilities. http://www.pcscworkgroup.com/specifications/specdownload.php.

[3] PC/SC workgroup. http://www.pcscworkgroup.com/.