PMDZ061-AE
24/09/2009

PC/SC

Partial documentation of the API

Headquarters, Europa

SpringCard
13 Voie la Cardon
Parc Gutenberg
91120 Palaiseau
FRANCE

Phone : +33 (0) 1 64 53 20 10
Fax : +33 (0) 1 64 53 20 18

Americas

SpringCard
694 Fifth Avenue
Suite 235
San Diego, CA 92101
USA

Phone : +1 (619) 544 1450
Fax : +1 (619) 573 6867

www.springcard.com

DOCUMENT INFORMATION

Category : Developer's manual
Group : General - PC/SC
Reference : PMDZ061
Version : AE
Status : draft

pmdz061-ae.doc
saved 24/09/09 - printed 24/09/09

Keywords :
PC/SC, smartcard, SCardConnect, SCardTransmit, SCardControl

Abstract :
As an introduction to Microsoft's "official" PC/SC documentation, this manual gives a good overview of all interesting functions in Winscard and equivalent PC/SC middlewares.

REVISION HISTORY

Ver.	Date	Author	Valid. by Tech.	Valid. by Qual.	Approv. by	Remarks :
AA	17/01/08	JDA				Early draft
AB	30/05/08	JDA				Almost complete
AC	05/09/08	JDA				New SpringCard template
AD	01/04/09	ECL	LTX	LTX		Adding SCardGetGCRFamily
AE	24/09/09	ECL				Adding SCardPresent and SCardSetReaderPower

TABLE OF CONTENT

1. INTRODUCTION

1.1. ABSTRACT

PC/SC is the de-facto standard to interface Personal Computers with Smart Cards (and smartcard readers of course). This high-level and standardized API allows the developer to focus on the smartcard itself, without dealing with various –and proprietary– aspects of every smartcard reader.

PC/SC is available on Windows computers (PC/SC stack integrated in the OS, available through `winscard.dll` library). Thanks to open-source PCSC-Lite project, PC/SC is also implemented on numerous UNIX platforms (including GNU/Linux and Mac OS X).

On the PocketPC platform (lightweight build of Windows CE that lacks the PC/SC stack), SpringCard's "SpringCard API" (`springcard.dll` library available in our SDK) provides the same features in an handy manner.

This document is a quick reference of the most useful functions provided by the PC/SC API. Pay attention that there're may be a few differences from one implementation and another. Always refer to the documentation of the PC/SC stack you are actually working with.

1.2. SUPPORTED PRODUCTS

At the date of writing, this document refers to

- Products in the CSB6 Family (CSB6, Prox'N'Roll, CrazyWriter, EasyFinger) running a firmware version ≥ 1.47 ;
- Products in the SpringCard Family (SpringCard-CF, SpringCard-WAP) using SpringCard "PC/SC Like" API.

Please review the datasheet of each product for accurate specification and a detailed list of features.

1.3. AUDIENCE

This manual is designed for use by application developers. He assumes that the reader has expert knowledge of computer development.

1.4. SUPPORT AND UPDATES

Interesting related materials (product datasheets, application notes, sample software, HOWTOs and FAQs…) are available at SpringCard's web site :

www.springcard.com

Updated versions of this document and others will be posted on this web site as soon as they are made available.

For technical support enquiries, please refer to SpringCard support page, on the web at address www.springcard.com/support .

1.5. USEFUL LINKS

- Microsoft's PC/SC reference documentation is included in most Visual Studio help system, and available online at http://msdn.microsoft.com . Enter "winscard" or "SCardTransmit" keywords in the search box.
- MUSCLE PCSC-Lite project : http://www.musclecard.com (direct link to PC/SC stack : http://pcsclite.alioth.debian.org)
- PC/SC workgroup : http://www.pcscworkgroup.com

1.6. ACKNOWLEDGEMENTS

Most explanations and examples written here have been inspired by PCSC-Lite documentation. The original document is located at :

http://pcsclite.alioth.debian.org/pcsc-lite/pcsc-lite.html

We thank the authors for this great job.

2. LIST OF PC/SC FUNCTIONS

Here's the list of "interesting" PC/SC functions[1].

For each function, a "compatibility" block shows whether the function is available under the 3 different platforms, according to the convention below :

Yes	Function is implemented and fully supported
NO	Function is missing
Dummy	Function is implemented for convenience, but does nothing
Limited	Function is implemented, but has strong limitations or a specific behaviour

All functions return a LONG. On success, value is SCARD_S_SUCCESS. Please refer to chapter 3.1 for a list of error (or warning) return codes.

[1] We've made the choice to list only the functions a developer must understand to build his smartcard-aware application. The full PC/SC API (as specified by Microsoft and as implemented in pcsc-lite) contains a lot of other functions, related to smartcard service providers and to the management of the smartcard subsystem. In 90% of the applications, they are never used.

SCardEstablishContext

Compatibility	
Winscard	**Yes**
pcsc-lite	**Yes**
SpringCard	**Yes**

The **SCardEstablishContext** function instantiates a context for the application within the PC/SC Resource Manager. This must be the first function called in a PC/SC application.

Synopsis

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

Parameters

dwScope	in	See below
pvReserved1	in	Must be NULL.
pvReserved2	in	Must be NULL.
PhContext	out	Handle to the PC/SC resource manager

Values for dwScope

Winscard specific

Parameter dwScope can take one of the following values :

- SCARD_SCOPE_USER
- SCARD_SCOPE_SYSTEM

PCSC-lite specific

Parameter dwScope must be set to SCARD_SCOPE_SYSTEM.

SpringCard API specific

Parameter dwScope is ignored, set to SCARD_SCOPE_SYSTEM for compatibility.

Side effects

SpringCard API specific

Calling the SCardEstablishContext function implicitly powers the SpringCard reader ON, and let it powered until

- SCardReleaseContext is called,
- SCardDisconnect is called, with dwDisposition parameter set to SCARD_UNPOWER_CARD (or SCARD_EJECT_CARD), and the card was the only powered one (on a multi-slot reader),
- A communication timeout occurs.

SCardReleaseContext

Compatibility	
Winscard	**Yes**
pcsc-lite	**Yes**
SpringCard	**Yes**

The **SCardReleaseContext** function destroys the application context within the PC/SC Resource Manager. This must be the last function called in a PC/SC application.

Synopsis

```
LONG SCardReleaseContext(SCARDCONTEXT hContext);
```

Parameters

hContext	in	Connection context to the be released

Side effects

SpringCard API specific

Calling the SCardReleaseContext function implicitly powers the SpringCard reader OFF.

SCardIsValidContext

Compatibility	
Winscard	**Yes**
pcsc-lite	**Yes**
SpringCard	**NO**

The **ScardIsValidContext** 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.

Synopsis

```
LONG SCardIsValidContext(SCARDCONTEXT hContext);
```

Parameters

hContext	in	Connection context to the be tested

Return value

- SCARD_E_INVALID_HANDLE when hContext is invalid.

SCardListReaders

Compatibility	
Winscard	**Yes**
pcsc-lite	**Yes**
SpringCard	**Yes**

The **ScardListReaders** functions returns a list of currently available readers on the system, optionally filtered by a set of named reader groups.

Synopsis

```
LONG SCardListReaders(SCARDCONTEXT hContext,
                      LPCSTR mszGroups,
                      LPSTR mszReaders,
                      LPDWORD pcchReaders);
```

Details

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 *szReaders* as NULL then this function returns the size of the buffer needed to allocate in *pcchReaders*.

If value pointed by *pcchReaders* is specified as SCARD_AUTOALLOCATE, then *mszReaders* is casted as a pointer to a pointer, and receives the address of a block of memory containing the character string, allocated by the PC/SC API. This block of memory must be deallocated by the calling application with *SCardFreeMemory*.

On success, *mszReaders* is a multi-string and separated by a null character ('\0') and ended by a double null character (e.g. "`Reader A\0Reader B\0\0`").

Parameters

`hContext`	in	Connection context to the PC/SC Resource Manager
`mszGroups`	in	List of groups to list readers
`MszReaders`	out	Multi-string to receive the list of readers
`pcchReaders`	in	Max. length of mszReaders
	out	Actual length of mszReaders including all NULL's characters

PCSC-lite specific

Parameter mszGroups is ignored.

mszReaders must be allocated by caller ; value SCARD_AUTOALLOCATE for pcchReaders is forbidden.

SpringCard API specific

Parameter mszGroups is ignored.

SCardConnect

Compatibility	
Winscard	**Yes**
pcsc-lite	**Yes**
SpringCard	**Yes**

The **SCardConnect** function establishes a connection between the calling application and a smartcard contained inserted into a specific reader.

The first connection powers up and resets of the card. If there's no card in the specified reader, an error is returned.

Synopsis

```
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	The name of the reader that contains the target card.
dwShareMode	in	Exclusive or shared connection
dwPreferredProtocols	in	Bit-mask specifying the list of acceptable card protocols
phCard	out	Handle to the smartcard
pdwActiveProtocol	out	Actual protocol selected by the reader

Values for dwShareMode

- SCARD_SHARE_SHARED : the application allows others to share the smartcard.
- SCARD_SHARE_EXCLUSIVE : the application requests exclusive access to the smartcard.
- SCARD_SHARE_DIRECT : the application requests direct (and exclusive) control of the reader, even without a smartcard in it.

SpringCard API specific

dwShareMode must be set to SCARD_SHARE_EXCLUSIVE. Specifying another value is unsupported and returns an error.

Values for dwPreferredProtocols

- SCARD_PROTOCOL_T0 : use the T=0 protocol.
- SCARD_PROTOCOL_T1 : use the T=1 protocol.
- SCARD_PROTOCOL_T0| SCARD_PROTOCOL_T1 : use either protocol.
- SCARD_PROTOCOL_RAW : for memory cards.
- 0 : allowed only if *dwShareMode* is set to SCARD_SHARE_DIRECT. In this case, there no communication with the smartcard.

Winscard specific

Value SCARD_PROTOCOL_RAW is no longer available.

PCSC-Lite specific

Value SCARD_PROTOCOL_RAW is no longer available.

SpringCard API specific

SpringCard API allows other values for dwPreferredProtocols. Please refer to SpringCard-CF or SpringCard-WAP specific documentation for details.

Values for pdwActiveProtocol

- SCARD_PROTOCOL_T0 : T=0 protocol activated.
- SCARD_PROTOCOL_T1 : T=1 protocol activated.
- SCARD_PROTOCOL_RAW : for memory cards.
- SCARD_PROTOCOL_UNDEFINED : SCARD_SHARE_DIRECT has been specified. It is possible that there is no card in the reader.

SpringCard specific

Calling the SCardConnect function implicitly powers the SpringCard reader ON (if it wasn't).

SCardReconnect

Compatibility	
Winscard	**Yes**
pcsc-lite	**Yes**
SpringCard	**Yes**

The **SCardReconnect** function re-establishes an existing connection between the calling application and smartcard. This function can be useful to move a card handle from direct access to general access, or to acknowledge and clear an error condition that is preventing further access to a smartcard.

Synopsis

```
LONG SCardReconnect(SCARDHANDLE hCard,
                    DWORD dwShareMode,
                    DWORD dwPreferredProtocols,
                    DWORD dwInitialization,
                    LPDWORD pdwActiveProtocol);
```

Parameters

hCard	in	Card context as returned by SCardConnect
dwShareMode	in	Exclusive or shared connection
dwPreferredProtocols	in	Bit-mask specifying the list of acceptable card protocols
dwInitialization	in	Type of initialization that should be performed on the card.
PdwActiveProtocol	out	Actual protocol selected by the reader

Values for dwShareMode

See SCardConnect paragraph.

Values for dwPreferredProtocols

See SCardConnect paragraph.

Values for dwInitialization

- SCARD_LEAVE_CARD : Do nothing.
- SCARD_RESET_CARD : Keep the card powered, and reset it (warm reset).
- SCARD_UNPOWER_CARD : Power down the card, then reset it (cold reset).

Values for pdwActiveProtocol

See SCardConnect paragraph.

Side effects

SpringCard specific

Calling the SCardConnect function implicitly powers the SpringCard reader ON (if it wasn't).

SCardDisconnect

Compatibility	
Winscard	**Yes**
pcsc-lite	**Yes**
SpringCard	**Yes**

The **SCardDisconnect** function terminates a previously opened connection between the calling application and a smartcard.

Synopsis

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

Parameters

hCard	in	Card context as returned by SCardConnect
dwDisposition	in	Action to perform on the smartcard

Values for dwDisposition

- SCARD_LEAVE_CARD : Do nothing (the card remains powered).
- SCARD_RESET_CARD : Reset the card (warm reset).
- SCARD_UNPOWER_CARD : Power down the card.
- SCARD_EJECT_CARD : Physically eject the card *(if reader is able to do so ☺).*

Side effects

SpringCard API specific

Calling the SCardDisconnect function with dwDisposition parameter set to SCARD_UNPOWER_CARD (or SCARD_EJECT_CARD) will implicitly power the SpringCard reader OFF if the card is the only powered one (on a multi-slot reader).

SCardStatus

Compatibility	
Winscard	**Yes**
pcsc-lite	**Yes**
SpringCard	**Yes**

The **SCardStatus** function returns the current status of a smartcard previously connected with SCardConnect.

Synopsis

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

Parameters

hCard	in	Card context as returned by SCardConnect
szReaderName	out	Friendly name of the reader the card is in
pcchReaderLen	in	Length of szReaderName
	out	Actual length of szReaderName, including terminating NULL character
pdwState	out	Current state of the reader
pdwProtocol	out	Current protocol of the smartcard
pbAtr	in	Current ATR of the smartcard[2]
pcbAtrLen	in	Max. length of pbAtr
	out	Actual length of pbAtr

Winscard and SpringCard API specific

If the value pointed by pcchReaderLen is specified as SCARD_AUTOALLOCATE, then szReaderName is casted as a pointer to a pointer, and receives the address of a block of memory containing the character string, allocated by the PC/SC API. This block of memory must be deallocated by the calling application with SCardFreeMemory.

Values for pdwState

- SCARD_ABSENT : No card in the reader.
- SCARD_PRESENT : There is a card in the reader, but it has not been moved into position for use *(if reader does support it ☺)*..
- 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.

[2] ISO 7816-3 says that the ATR has 32 bytes or less, there's no need to allocate more.

- 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.

Values for pdwProtocol

See SCardConnect paragraph. Value is accurate only if pdwState value is SCARD_SPECIFIC.

SCardSetTimeout

Compatibility	
Winscard	**NO**
pcsc-lite	**Dummy**
SpringCard	**Yes**

The **SCardSetTimeout** function defines the timeout value for SCardTransmit. This is useful for instance when card operation takes a long time, during which the reader may appear as "frozen".

Note that this function has never been supported by Winscard, and has been deprecated from PCSC-lite. We continue to support it in SpringCard API.

Synopsis

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

Parameters

hCard	in	Card context as returned by SCardConnect
dwTimeout	In	New value for SCardTransmit timeout (in seconds)

SCardTransmit

Compatibility	
Winscard	**Yes**
pcsc-lite	**Yes**
SpringCard	**Yes**

The **SCardTransmit** function sends a command APDU to the smartcard, and retrieve its response.

Synopsis

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

Parameters

hCard	in	Card context as returned by SCardConnect
pioSendPci	in/out	See below
pbSendBuffer	in	APDU to send to the card
cbSendLength	in	Length of the APDU
pioRecvPci	in/out	See below
pbRecvBuffer	out	*Response from the card*
pcbRecvLength	in	*Max. length of pbRecvBuffer*
	out	*Actual length of response in pbRecvBuffer*

Values for pioSendPci and pioRecvPci

Both parameters are pointer to structure telling the driver which protocol shall be used to send the APDU, and receive card's response.

In most cases, you may use one of the pre-defined global values for both parameters :

- SCARD_PCI_T0 if current card protocol T=0.
- SCARD_PCI_T1 if current card protocol T=1.

Alternatively, you may set both values to NULL, and let the PC/SC API verify which protocol is actually in use.

SpringCard API specific

Parameters pioSendPci and pioRecvPci are ignored. Dummy defines of SCARD_PCI_T0 and SCARD_PCI_T1 are provided to ease code portability.

SCardControl

Compatibility	
Winscard	**Yes**
pcsc-lite	**Yes**
SpringCard	**Limited**

The **SCardControl** function gives you direct control on the reader (even when there's no card in it).

Synopsis

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

Parameters

hCard	in	Card context as returned by SCardConnect
dwControlCode	in	*Control code for the operation to be performed*
pbSendBuffer	in	Data required to perform the operation. This parameter can be NULL if the operation takes no data.
cbSendLength	in	Length of the data
pbRecvBuffer	out	*Response from the reader. This parameter can be NULL if the operation returns nothing.*
pcbRecvLength	in	*Max. length of pbRecvBuffer*
lpBytesReturned	out	*Actual length of response in pbRecvBuffer*

Remark 1

The list of available commands, together with their control codes, are proprietary (e.g. vendor-specific and most of the time product-specific).

Always refer to the vendor-supplied documentation of the reader to get the list of actually supported operations (plus specification of parameters and responses).

SpringCard API specific

The dwControlCode parameter is ignored, and pbSendBuffer is forwarded "as is" to the GemCore chipset that is inside the SpringCard-CF or SpringCard-WAP reader.

Please refer to Gemalto's GemCore documention in this cas.

Remark 2

Depending on the implementation, calling SCardControl may require that the hCard connection has been created with either the SCARD_SHARE_DIRECT or

SCARD_SHARE_EXCLUSIVE value for SCardConnect or SCardReconnect dwShareMode parameter.

SCardGetStatusChange

Compatibility	
Winscard	**Yes**
pcsc-lite	**Yes**
SpringCard	**NO**

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

SpringCard API specific

Please use **SCardPresent** for SpringCard API

Synopsis

```
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	Array of SCARD_READERSTATE structures (one for each reader)
cReader	in	Number of SCARD_READERSTATE structures in rgReaderStates

Usage note

The caller supplies a list of readers to be monitored by an SCARD_READERSTATE array and the maximum amount of time (in milliseconds) 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 (e.g. card inserted or card removed), having filled in the dwEventState members of rgReaderStates appropriately.

If rgReaderStates is NULL, and cReader is equal to 0, then the function will block until a reader is added to the system (requires hot-plupging support).

SCardGetStatusChange uses the user-supplied value in the dwCurrentState members of the rgReaderStates SCARD_READERSTATE array as the definition of the current state of the readers. Providing an invalid value will prevent ScardGetStatusChange to work as expected.

SCARD_READERSTATE structure

```
typedef struct {
```

```
    LPCSTR szReader;      /* Reader name */
    LPVOID pvUserData;    /* User defined data */
    DWORD dwCurrentState; /* Current state of reader */
    DWORD dwEventState;   /* New state after a state change */
    DWORD cbAtr;          /* ATR length */
    BYTE rgbAtr[MAX_ATR_SIZE]; /* ATR of the card */
} SCARD_READERSTATE;

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

Values of dwCurrentState and dwEventState

- 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.
- 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_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.

SCardPresent

Compatibility	
Winscard	NO
pcsc-lite	NO
SpringCard	Yes

This function gives information on card's presence for one reader.

This function is available in SpringCardAPI.dll only.

Synopsis

```
LONG SCardPresent(SCARDCONTEXT hContext,
                  LPCTSTR Reader);
```

Parameters

hContext	in	Connection context to the PC/SC Resource Manager
Reader	in	Reader name.

Sample code

```
uint state = SCARD.Present(SCardContext, readerName);
string ReaderState = SCARD.StateToStr(state);
```

Return values

Use StateToStr function of the SpringCardAPI.dll to get a string that describes card's state.

- SCARD_UNKNOWN : Unknown state
- SCARD_ABSENT : Card is absent
- SCARD_PRESENT : Card is present
- SCARD_SWALLOWED : Card not powered
- SCARD_POWERED : Card is powered
- SCARD_NEGOTIABLE : Ready for PTS
- SCARD_SPECIFIC : PTS has been set

SCardCancel

Compatibility	
Winscard	**Yes**
pcsc-lite	**Yes**
SpringCard	**NO**

The **SCardCancel** function cancels all pending blocking requests on the `SCardGetStatusChange()` function.

Synopsis

```
LONG SCardCancel(SCARDCONTEXT hContext);
```

Parameters

`hContext`	in	Connection context

SCardBeginTransaction

Compatibility	
Winscard	**Yes**
pcsc-lite	**Yes**
SpringCard	**Dummy**

The **SCardBeginTransaction** function establishes a temporary exclusive access mode, for doing a series of commands or transaction into the card.

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 already started a transaction with the same smartcard, then function will block until the transaction is finished.

SpringCard API specific

This function does nothing, as only one application can access the reader in the same time. In a multi-threaded application, transaction isolation must be implemented by the developer.

Synopsis

```
LONG SCardBeginTransaction(SCARDHANDLE hCard);
```

Parameters

hCard	in	Card context as returned by SCardConnect

SCardEndTransaction

Compatibility	
Winscard	**Yes**
pcsc-lite	**Yes**
SpringCard	**Dummy**

The **SCardEndTransaction** exits the exclusive access mode gained after a successful call to SCardBeginTransaction.

Synopsis

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

Parameters

hCard	in	Card context as returned by SCardConnect
dwDisposition	in	Action to perform on the smartcard

Values for dwDisposition

See SCardDisconnect paragraph.

PCSC-Lite specific

dwDisposition must be set to SCARD_LEAVE_CARD only.

SCardGetGCRFamily

Compatibility	
Winscard	**NO**
pcsc-lite	**NO**
SpringCard	**Yes**

The **SCardGetGCRFamily** returns the version of the GemCore chipset.

Synopsis

```
BYTE SCardGetGCRFamily(void);
```

SpringCard specific

This function is available for Springcard API only, and returns one of the following values:

10 stands for GCR_ORIGINAL

13 stands for GCR_LITE

14 stands for GCR_LITE_PRO

15 stands for GCR_POS

SCardSetReaderPower

Compatibility	
Winscard	NO
pcsc-lite	NO
SpringCard	Yes

This function sets Reader power control.

This function is available in SpringCardAPI.dll only.

Synopsis

```
LONG SCardSetReaderPower(SCARDCONTEXT context,
                                    DWORD mode);
```

Parameters

hContext	in	Connection context to the PC/SC Resource Manager
mode	in	Reader power control mode (see below).

Values for mode

- POWER_AUTO :

  the default mode, the reader power is managed by the dll.
- POWER_ALWAYS_ON :

  forces the reader to be always powered.
- POWER_FORCE_OFF :

  forces the power to switch off, then the mode is set at automatic mode.

Note

When working for Pocket Pc, you don't need to pass the context, the only parameter you have to set is mode.

Sample code

For Pocket PC :

```
SCARD.ReaderMode(SCARD.POWER_FORCE_OFF); //Forces the reader to turn off
```

Other :

```
SCARD.ReaderMode(SCardContext, SCARD.POWER_FORCE_OFF); //Forces the reader to turn off
```

3. LIST OF PC/SC CONSTANTS

3.1. RETURN CODES

Value	Meaning
SCARD_E_BAD_SEEK	An error occurred in setting the smart card file object pointer.
SCARD_E_CANCELLED	The action was canceled by an SCardCancel request.
SCARD_E_CANT_DISPOSE	The system could not dispose of the media in the requested manner.
SCARD_E_CARD_UNSUPPORTED	The smart card does not meet minimal requirements for support.
SCARD_E_CERTIFICATE_UNAVAILABLE	The requested certificate could not be obtained.
SCARD_E_COMM_DATA_LOST	A communications error with the smart card has been detected.
SCARD_E_DIR_NOT_FOUND	The specified directory does not exist in the smart card.
SCARD_E_DUPLICATE_READER	The reader driver did not produce a unique reader name.
SCARD_E_FILE_NOT_FOUND	The specified file does not exist in the smart card.
SCARD_E_ICC_CREATEORDER	The requested order of object creation is not supported.
SCARD_E_ICC_INSTALLATION	No primary provider can be found for the smart card.
SCARD_E_INSUFFICIENT_BUFFER	The data buffer for returned data is too small for the returned data.
SCARD_E_INVALID_ATR	An ATR string obtained from the registry is not a valid ATR string.
SCARD_E_INVALID_CHV	The supplied PIN is incorrect.
SCARD_E_INVALID_HANDLE	The supplied handle was not valid.
SCARD_E_INVALID_PARAMETER	One or more of the supplied parameters could not be properly interpreted.
SCARD_E_INVALID_TARGET	Registry startup information is missing or not valid.
SCARD_E_INVALID_VALUE	One or more of the supplied parameter values could not be properly interpreted.
SCARD_E_NO_ACCESS	Access is denied to the file.
SCARD_E_NO_DIR	The supplied path does not represent a smart card directory.
SCARD_E_NO_FILE	The supplied path does not represent a smart card file.
SCARD_E_NO_MEMORY	Not enough memory available to complete this command.
SCARD_E_NO_READERS_AVAILABLE	No smart card reader is available.
SCARD_E_NO_SERVICE	The smart card resource manager is not running.

SCARD_E_NO_SMARTCARD	The operation requires a smart card, but no smart card is currently in the device.
SCARD_E_NO_SUCH_CERTIFICATE	The requested certificate does not exist.
SCARD_E_NOT_READY	The reader or card is not ready to accept commands.
SCARD_E_NOT_TRANSACTED	An attempt was made to end a nonexistent transaction.
SCARD_E_PCI_TOO_SMALL	The PCI receive buffer was too small.
SCARD_E_PROTO_MISMATCH	The requested protocols are incompatible with the protocol currently in use with the card.
SCARD_E_READER_UNAVAILABLE	The specified reader is not currently available for use.
SCARD_E_READER_UNSUPPORTED	The reader driver does not meet minimal requirements for support.
SCARD_E_SERVICE_STOPPED	The smart card resource manager has shut down.
SCARD_E_SHARING_VIOLATION	The smart card cannot be accessed because of other outstanding connections.
SCARD_E_SYSTEM_CANCELLED	The action was canceled by the system, presumably to log off or shut down.
SCARD_E_TIMEOUT	The user-specified time-out value has expired.
SCARD_E_UNEXPECTED	An unexpected card error has occurred.
SCARD_E_UNKNOWN_CARD	The specified smart card name is not recognized.
SCARD_E_UNKNOWN_READER	The specified reader name is not recognized.
SCARD_E_UNKNOWN_RES_MNG	An unrecognized error code was returned from a layered component.
SCARD_E_UNSUPPORTED_FEATURE	This smart card does not support the requested feature.
SCARD_E_WRITE_TOO_MANY	An attempt was made to write more data than would fit in the target object.
SCARD_F_COMM_ERROR	An internal communications error has been detected.
SCARD_F_INTERNAL_ERROR	An internal consistency check failed.
SCARD_F_UNKNOWN_ERROR	An internal error has been detected, but the source is unknown.
SCARD_F_WAITED_TOO_LONG	An internal consistency timer has expired.
SCARD_P_SHUTDOWN	The operation has been aborted to allow the server application to exit.
SCARD_S_SUCCESS	No error was encountered.
SCARD_W_CANCELLED_BY_USER	The action was canceled by the user.
SCARD_W_CHV_BLOCKED	The card cannot be accessed because the maximum number of PIN entry attempts has been reached.
SCARD_W_EOF	The end of the smart card file has been reached.
SCARD_W_REMOVED_CARD	The smart card has been removed, so further communication is not possible.
SCARD_W_RESET_CARD	The smart card has been reset, so any shared state information is not valid.
SCARD_W_SECURITY_VIOLATION	Access was denied because of a security violation.

SCARD_W_UNPOWERED_CARD	Power has been removed from the smart card, so that further communication is not possible.
SCARD_W_UNRESPONSIVE_CARD	The smart card is not responding to a reset.
SCARD_W_UNSUPPORTED_CARD	The reader cannot communicate with the card, due to ATR string configuration conflicts.
SCARD_W_WRONG_CHV	The card cannot be accessed because the wrong PIN was presented.

DISCLAIMER

This document is provided for informational purposes only and shall not be construed as a commercial offer, a license, an advisory, fiduciary or professional relationship between PRO ACTIVE and you. No information provided in this document shall be considered a substitute for your independent investigation.

The information provided in document may be related to products or services that are not available in your country.

This document is provided "as is" and without warranty of any kind to the extent allowed by the applicable law. While PRO ACTIVE will use reasonable efforts to provide reliable information, we don't warrant that this document is free of inaccuracies, errors and/or omissions, or that its content is appropriate for your particular use or up to date. PRO ACTIVE reserves the right to change the information at any time without notice.

PRO ACTIVE does not warrant any results derived from the use of the products described in this document. PRO ACTIVE will not be liable for any indirect, consequential or incidental damages, including but not limited to lost profits or revenues, business interruption, loss of data arising out of or in connection with the use, inability to use or reliance on any product (either hardware or software) described in this document.

These products are not designed for use in life support appliances, devices, or systems where malfunction of these product may result in personal injury. PRO ACTIVE customers using or selling these products for use in such applications do so on their own risk and agree to fully indemnify PRO ACTIVE for any damages resulting from such improper use or sale.

COPYRIGHT NOTICE

All information in this document is either public information or is the intellectual property of PRO ACTIVE and/or its suppliers or partners.

You are free to view and print this document for your own use only. Those rights granted to you constitute a license and not a transfer of title : you may not remove this copyright notice nor the proprietary notices contained in this documents, and you are not allowed to publish or reproduce this document, either on the web or by any mean, without written permission of PRO ACTIVE.

EDITOR'S INFORMATION

PRO ACTIVE SAS company with a capital of 227 000 €
RCS EVRY B 429 665 482
Parc Gutenberg, 13 voie La Cardon
91120 Palaiseau – France