SpringCard

Developers' Toolkit Reference

This toolkit and documentation is provided on an as is basis. Pro-Active shall not be held responsible for any mishaps caused by the use of this software. For more information please visit www.springcard.com

Copyrights and disclaimers

The **SpringCard developers' toolkit** is copyright (c) 2000 - 2002 Pro-Active

Redistribution and use in source (source code) and binary (object code) forms, with or without modification, are permitted provided that the following conditions are met :
1. Redistributed source code must retain the above copyright notice, this list of conditions and the disclaimer below,
2. Redistributed object code must reproduce the above copyright notice, this list of conditions and the disclaimer below in the documentation and/or other materials provided with the distribution,
3. The name of Pro-Active may not be used to endorse or promote products derived from this software or in any other form without specific prior written permission from Pro-Active,
4. Redistribution of any modified code must be labeled "Code derived from original Pro-Active copyrighted source code".

THIS SOFTWARE IS PROVIDED BY PRO-ACTIVE "AS IS" EITHER FREE OF CHARGE OR AS PART OF A COMMERCIAL BUNDLE. PRO-ACTIVE SHALL NOT BE LIABLE FOR INFRINGEMENTS OF THIRD PARTIES RIGHTS BASED ON THIS SOFTWARE. ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. PRO-ACTIVE DOES NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THIS SOFTWARE WILL MEET THE USER'S REQUIREMENTS OR THAT THE OPERATION OF IT WILL BE UNINTERRUPTED OR ERROR-FREE. IN NO EVENT, UNLESS REQUIRED BY APPLICABLE LAW, SHALL PRO-ACTIVE BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. ALSO, PRO-ACTIVE IS UNDER NO OBLIGATION TO MAINTAIN, CORRECT, UPDATE, CHANGE, MODIFY, OR OTHERWISE SUPPORT THIS SOFTWARE.

The **SpringCard developers' toolkit** and the **SpringCard APIs** include software and documentation developed by or wrote by **David Corcoran** in the MUSCLE project.
corcoran@linuxnet.com
www.linuxnet.com
Please read the file "credit-muscle.txt" for copyright and disclaimer.

The **SpringCard developers' toolkit** and the **SpringCard APIs** include software and documentation developed by or wrote by **GemPlus**.
http://www.gemplus.com/techno/tlp_drivers/
Please read the file "credit-gemplus.txt" for copyright and disclaimer.

Trademarks

SpringCard and SpringCard-CF are registered trademarks of Pro-Active, France.
Visor, SpringBoard and HandSpring are registered trademarks of HandSpring, USA.
Microsoft, Windows, PocketPC are registered trademarks of Microsoft, USA.
GemPlus and GemCore are registered trademarks of GemPlus, France.

SpringCard' technical data

Common

Dual slot smartcard reader :

- Slot A is full size ISO slot;
- Slot B is an internal SIM/SAM slot.
- GemPlus GemCore chipset.
- Supported cards : ISO7816-1,2,3,4 (T=0, T=1) in 3V.

SpringCard-CF

- For use in the PocketPC handled family through the Compact-Flash expansion slot[1].
- API implementation as a Windows CE DLL for PocketPC 2000 or 2002.
- Plug and play detection as a serial port under Windows CE for PocketPC 2000 or 2002.
- Powered by the PocketPC.

SpringCard-VS

- For use in the HandSpring Visor PDA family through the SpringBoard expansion slot.
- API implementation as a PalmOS static library.
- Embedded 2Mb flash memory.
- Powered by the Visor.

[1] Suitable for use with most of the PocketPC handleds featuring a Compact-Flash slot. Check mechanical data before buying.

The SpringCard API

The PC/SC API

This document contains the reference API calls for communicating to the SpringCard readers in “PC/SC like” mode.

PC/SC is a standard proposed by the PC/SC workgroup which is a conglomerate of representative from major smartcard manufacturers and other companies. This specification tries to abstract the smartcard layer into a high level API so that smartcards and their readers can be accessed in a homogeneous fashion.

Two well known PC/SC implementation are the WinsCard API provided by Microsoft under Windows ®, and the “PC/SC lite” provided by the MUSCLE project under Linux.

The SpringCard routines are compatible with the Microsoft ® API calls and with the MUSCLE project “PC/SC lite” API calls. It gives also a common API for communication with the SpringCard readers under PalmOS or Windows CE as if they were regular PC/SC readers under Windows 95/98/ME/2000/XP.

Differences between the SpringCard API and the PC/SC standard

As the SpringCard family is targeted for light embedded systems, the goal is not to provide a complete PC/SC implementation, but to give developers the ability to access smart cards the same way under many OS.

- PC/SC is designed as a three layers model :
  - The bottom layer (IFD Handlers) is provided by the reader manufacturer;
  - The middle layer (ICC Resource Manager) –or PC/SC middleware– is provided by the OS itself;
  - The top layer (Service Provider) is developed in parallel with the embedded card application. The applications can access a card through this Service Provider or directly through the PC/SC middleware (ICC Aware Application).

  The SpringCard API includes both bottom and middle layers in a monolithic library and doesn’t provide support for the Service Provider layer.
  ICC Aware Application can be directly bound to the SpringCard through standards PC/SC calls, otherwise the Service Provider must be replaced by external helper libraries.

- PC/SC is designed for very large systems (supports of many readers from multiple manufactures in the same system, logical or geographical groups of readers…).

Through the SpringCard API, the application can access only two readers: the SpringCard main slot and the SpringCard SIM/SAM slot if available.

- PC/SC is designed for multi users, multi tasking systems. Both PalmOS and PocketPC are single user systems, and multi tasking is very limited. The SpringCard API also gives only a exclusive access to the smart cards and doesn't provide the transaction mechanism.

- SpringCard must remain an easy way to access smart cards from a handled or a PDA. The SpringCard API provides many helper functions, primarily dedicated to ease the use of binary buffers under Embedded Visual Basic, but also very useful under other development tools.

Working with the SpringCard API

Under PalmOS (Visor), the SpringCard API is contained in the static library "springcard.lib". Use the supplied C/C++ header file "springcard.h" for type definition and function prototypes.

You can debug your PalmOS SpringCard application with POSE (the PalmOS emulator) through the SpringCard Starter Kit which can be bought at Pro-Active.

Under Windows CE (PocketPC), the SpringCard API is contained in the library springcard.dll[2].

- In an Embedded Visual C++ project, use the supplied C/C++ header file "springcard.h" and the related wrapper "springcard.lib";
- In an Embedded Visual Basic, use the supplied VB modules "springcard_api.bas" and "springcard_err.h".

You can't emulate the SpringCard reader in the PocketPC environment, but you can either

- Design a VB or Visual C++ application featuring smart cards access through the Windows PC/SC API (winscard.h), then port it to eVB or eVC and move references from "winscard.dll" to "springcard.dll";
- Debug your application inside your target PocketPC handled through Microsoft provided tools.

Type definitions

The following is a list of commonly used type definitions.

```
BYTE            unsigned char
USHORT          unsigned short
ULONG           unsigned long
BOOL            short
DWORD           unsigned long
WORD            unsigned long
LONG            long
RESPONSECODE    long
LPCSTR          const char *
SCARDCONTEXT    unsigned long
```

[2] Under Windows CE the API is compiled using UNICODE for every strings. Be sure to deploy the springcard.dll suitable for your target (i.e., PocketPC version and processor family).

```
PSCARDCONTEXT    unsigned long *
LPSCARDCONTEXT   unsigned long *
SCARDHANDLE      unsigned long
PSCARDHANDLE     unsigned long *
LPSCARDHANDLE    unsigned long *
LPCVOID          const void *
LPVOID           void *
LPCBYTE          const unsigned char *
LPBYTE           unsigned char *
LPDWORD          unsigned long *
LPSTR            char *
LPCWSTR          char *
```

Error codes

The following is a list of commonly used errors.

SCARD_E_UNSUPPORTED_INTERFACE	SCARD_E_NOT_TRANSACTED
SCARD_E_UNSUPPORTED_FEATURE	SCARD_E_READER_UNAVAILABLE
SCARD_E_NOTIMPL	SCARD_F_UNKNOWN_ERROR
SCARD_E_UNSUPPORTED_FUNCTION	SCARD_W_UNSUPPORTED_CARD
SCARD_E_INSUFFICIENT_BUFFER	SCARD_W_UNRESPONSIVE_CARD
SCARD_E_INVALID_ATR	SCARD_W_UNPOWERED_CARD
SCARD_E_INVALID_HANDLE	SCARD_W_RESET_CARD
SCARD_E_INVALID_PARAMETER	SCARD_W_REMOVED_CARD
SCARD_E_INVALID_TARGET	SCARD_W_INSERTED_CARD
SCARD_E_INVALID_VALUE	SCARD_E_UNKNOWN_READER
SCARD_F_COMM_ERROR	SCARD_E_TIMEOUT
SCARD_F_INTERNAL_ERROR	SCARD_E_NO_SMARTCARD
SCARD_E_UNKNOWN_READER	SCARD_E_UNKNOWN_CARD
SCARD_E_TIMEOUT	SCARD_E_PROTO_MISMATCH
SCARD_E_SHARING_VIOLATION	SCARD_E_SYSTEM_CANCELLED
SCARD_E_NO_SMARTCARD	SCARD_E_PCI_TOO_SMALL
SCARD_E_UNKNOWN_CARD	SCARD_E_READER_UNSUPPORTED
SCARD_E_NOT_READY	SCARD_E_DUPLICATE_READER
SCARD_E_SYSTEM_CANCELLED	SCARD_E_CARD_UNSUPPORTED
	SCARD_E_NO_SERVICE
	SCARD_E_SERVICE_STOPPED

For a human readable representation of an error code, call the API function SCardErrorToString.

SpringCard functions – PC/SC compliant

SCardEstablishContext

SCardReleaseContext

SCardListReaders

SCardConnect

SCardDisconnect

SCardControl

SCardStatus

SCardTransmit

SCardFreeMemory

SCardEstablishContext

Synopsis:

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

Parameters:

dwScope	Not used. Set to SCARD_SCOPE_SYSTEM for compatibility.
pvReserved1	Not used. Set to NULL.
pvReserved2	Not used. Set to NULL.
phContext	Returned reference to this connection.

Description:

This function creates a communication context to the SpringCard readers. This must be the first function called in a SpringCard application.

Returns:

SCARD_S_SUCCESS	Successful.
SCARD_E_INVALID_VALUE	Invalid scope type passed.
SCARD_E_READER_UNAVAILABLE	Could not find the SpringCard readers.

Example:

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

SCardReleaseContext

Synopsis:

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

Parameters:

hContext	Connection context to be closed.

Description:

This function destroys a communication context to the SpringCard readers. This must be the last function called in a SpringCard application.

Returns:

SCARD_S_SUCCESS	Successful.
SCARD_E_INVALID_HANDLE	Invalid hContext handle.

Example:

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

SCardListReaders

Synopsis:

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

Parameters:

hContext	Connection context to the SpringCard readers.
mszGroups	Not used, set to NULL.
mszReaders	Multi-string with list of readers.
pcchReaders	Size of multi-string buffer including NULLs.

Description:

This function returns a list of currently available readers.

mszReaders is a pointer to a character string which will be allocated by the application. If the application sends mszReaders as NULL then this function will return the size of the buffer needed to allocate in pcchReaders.

Remark:

The reader names will be a multi-string and separated by a NULL character and ended by a double NULL, for example,

```
SpringCard GCR-1.20-0M7 slot A\0 SpringCard GCR-1.20-0M7 B\0\0
```

The “GCR-xxx” pattern is the firmware release of the embedded GemCore chipset.
The first reader name returned is always the SpringCard main slot, the second is the SpringCard internal SIM/SAM slot.

Returns:

SCARD_S_SUCCESS	Successful.
SCARD_E_INVALID_HANDLE	Invalid hContext handle.
SCARD_E_INSUFFICIENT_BUFFER	Reader buffer not large enough.
SCARD_E_READER_UNAVAILABLE	The reader has been removed.

Example:

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

SCardConnect

Synopsis:

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

Parameters:

hContext	Connection context to the PC/SC Resource Manager.
szReader	Reader name to connect to. Use SCardListReaders to retrieve the available reader names. You can also use shortcut “A” for the SpringCard main-slot, and shortcut “B” for the internal SIM/SAM slot.
dwShareMode	Not used. Set to SCARD_SHARE_EXCLUSIVE.
dwPreferredProtocols	Desired protocol use.
phCard	Handle to this connection.
pdwActiveProtocol	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.

Values of dwPreferredProtocols:

SCARD_PROTOCOL_T0	Select the T=0 protocol.
SCARD_PROTOCOL_T1	Select the T=1 protocol.
SCARD_PROTOCOL_Tx	Select the T=0 or T=1 protocol.

Those values of dwPreferredProtocols may be combined (OR) with the following flags:

SCARD_PROTOCOL_HIS	Force T=0 or T=1 at double clock speed.
SCARD_PROTOCOL_PTS	Force the card baudrate to 9600 bauds (otherwise, the SpringCard reader will ask the card to run at the highest available baudrate).
SCARD_PROTOCOL_EMV	Force EMV compliant T=0 or T=1 protocol.

The specific value “SCARD_PROTOCOL_RAW” allows to work with synchronous cards. This is not a PC/SC standard. See chapter to “synchronous smartcards” for further information.

Remark:

This function powers-up the SpringCard and the card. Call SCardDisconnect as soon as possible to reduce power consumption.

Returns:

SCARD_S_SUCCESS	Successful
SCARD_E_INVALID_VALUE	Invalid sharing mode, 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.
SCARD_E_INVALID_HANDLE	Invalid hContext handle.

The if not NULL, the dwActiveProcol variable receives the value of the selected protocol:

SCARD_PROTOCOL_T0	T=0 protocol has been selected.
SCARD_PROTOCOL_T1	T=1 protocol has been selected.
SCARD_PROTOCOL_RAW	No protocol selected, reserved for synchronous cards.

Example:

```
SCARDCONTEXT hContext;
SCARDHANDLE hCard
DWORD dwActiveProtocol;
LONG rc;
rc = SCardEstablishContext( SCARD_SCOPE_SYSTEM,
                            NULL,
                            NULL,
                           &hContext );
rc = SCardConnect( hContext,
                   "A",
                   SCARD_SHARE_EXCLUSIVE,
                   SCARD_PROTOCOL_T0,
                  &hCard,
                  &dwActiveProtocol );
```

SCardDisconnect

Synopsis:

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

Parameters:

hCard	Connection made from SCardConnect.
dwDisposition	Action to perform with the card and the reader.

Description:

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

Depending on the value of dwDisposition, the SpringCard API does the following:

SCARD_LEAVE_CARD	The handle to the card is closed, but the card remains physically in the same state as before.
SCARD_RESET_CARD	The card is reset, but remains powered.
SCARD_UNPOWER_CARD	The card is powered-down. The reader remains powered.
SCARD_EJECT_CARD	The card is powered-down. If both slots are in power-down state, the reader is also powered-down. It will automatically be powered-up again on the next call to SCardConnect.

Returns:

SCARD_S_SUCCESS	Successful.
SCARD_E_INVALID_HANDLE	Invalid hCard handle.
SCARD_E_READER_UNAVAILABLE	The reader has been removed.

Example:

```
SCARDCONTEXT hContext;
SCARDHANDLE hCard
DWORD dwActiveProtocol;
LONG rc;
rc = SCardEstablishContext( SCARD_SCOPE_SYSTEM,
                            NULL,
                            NULL,
                           &hContext );
rc = SCardConnect( hContext,
                   "A",
                   SCARD_SHARE_EXCLUSIVE,
                   SCARD_PROTOCOL_Tx,
                  &hCard,
                  &dwActiveProtocol );
rc = SCardDisconnect( hCard,
                      SCARD_UNPOWER_CARD );
```

SCardTransmit

Synopsis:

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

Parameters:

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

Description:

This function sends an APDU to the smartcard contained in the reader connected to by SCardConnect. The card responds from the APDU and stores this response in pbRecvBuffer and its length in pcbRecvLength.

SendPci and RecvPci 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;
```

Values of pioSendPci:

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

Remark:

As the card protocol is known through hCard handle, pioSendPci and RecvPci can also be set to NULL.

Returns:

SCARD_S_SUCCESS	Successful
SCARD_E_NOT_TRANSACTED	APDU exchange not successful.
SCARD_E_INVALID_HANDLE	Invalid hCard handle.
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_REMOVED_CARD	The card has been removed from the reader.

Example:

```
LONG rc;
SCARDCONTEXT hContext; SCARDHANDLE hCard;
DWORD dwActiveProtocol, dwRecvLength;
SCARD_IO_REQUEST pioRecvPci;
BYTE pbRecvBuffer[10];
BYTE pbSendBuffer = { 0xC0, 0xA4, 0x00, 0x00, 0x02, 0x3F, 0x00 };
rc = SCardEstablishContext( SCARD_SCOPE_SYSTEM,
                            NULL,
                            NULL,
                           &hContext );
rc = SCardConnect( hContext,
                   "A",
                   SCARD_SHARE_EXCLUSIVE,
                   SCARD_PROTOCOL_T0,
                  &hCard,
                  &dwActiveProtocol);
dwRecvLength = sizeof(pbRecvBuffer);
rc = SCardTransmit( hCard,
                    SCARD_PCI_T0,
                    pbSendBuffer,
                    sizeof(pbSendBuffer),
                   &pioRecvPci,
                    pbRecvBuffer,
                   &dwRecvLength );
```

SCardControl

Synopsis:

```
LONG SCardControl( IN SCARDHANDLE hCard,
                   IN LPCBYTE pbSendBuffer,
                   IN DWORD cbSendLength,
                   OUT LPBYTE pbRecvBuffer,
                   INOUT LPDWORD pcbRecvLength );
```

Parameters:

hCard	Connection made from SCardConnect.
pbSendBuffer	Command to send to the reader.
cbSendLength	Length of the command.
pbRecvBuffer	Response from the reader
pcbRecvLength	Length of the response.

Description:

This function sends a command directly to the GemCore chipset into reader. This allows you to:

- Run an interpreted synchronous driver, for memory cards handling (see chapter to "synchronous smartcards" for further information);
- Access (read/write) GemCore internal configuration registers;
- Power-up your card with manual PTS negociation;
- …

For more details regarding the GemCore chipset command set, please refer to GemPlus documentation.

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_REMOVED_CARD	The card has been removed from the reader.

Example:

```
LONG rc;
SCARDCONTEXT hContext; SCARDHANDLE hCard;
DWORD dwActiveProtocol, dwRecvLength;
BYTE pbRecvBuffer[10];
BYTE pbSendBuffer = { 0x06, 0x00, 0x0A, 0x01, 0x01, 0x10 0x00 };
rc = SCardEstablishContext( SCARD_SCOPE_SYSTEM, NULL, NULL, &hContext );
rc = SCardConnect( hContext,
                   "A",
                   SCARD_SHARE_EXCLUSIVE,
                   SCARD_PROTOCOL_RAW,
                  &hCard,
                  &dwActiveProtocol);
dwRecvLength = sizeof(pbRecvBuffer);
rc = SCardControl( hCard,
                   pbSendBuffer,
                   sizeof(pbSendBuffer),
                   pbRecvBuffer,
                  &dwRecvLength );
```

SCardStatus

Synopsis:

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

Parameters:

hCard	Connection made from SCardConnect
szReaderName	Friendly name of this reader.
pcchReaderLen	Size of the szReaderName multi-string
pdwState	Current state of this reader
pdwProtocol	Current protocol of this reader
pbAtr	Current ATR of a card in this reader
pcbAtrLen	Length of ATR

Description:

This function returns the current status of the reader connected to by hCard. Its friendly name will be stored in szReaderName. pcchReaderLen will be the size of the allocated buffer for szReaderName. If this is too small the function will return with the necessary size in pcchReaderLen. The current state, and protocol will be stored in pdwState and pdwProtocol respectively. pdwState is a DWORD possibly OR ,d with the following values:

Values of pdwState:

SCARD_ABSENT	There is no card in the reader.
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_NEGOTIABLEMODE	The card has been reset and is awaiting PTS negotiation.
SCARD_SPECIFICMODE	The card has been reset and specific communication protocols have been established.

Value of pdwProtocols	Meaning
SCARD_PROTOCOL_T0	T=0 protocol has been selected.
SCARD_PROTOCOL_T1	T=1 protocol has been selected.
SCARD_PROTOCOL_RAW	No protocol selected, reserved for synchronous cards.

Returns:

SCARD_S_SUCCESS	Successful
SCARD_E_INVALID_HANDLE	Invalid hCard handle
SCARD_E_INSUFFICIENT_BUFFER	Not enough allocated memory for szReaderName
SCARD_E_READER_UNAVAILABLE	The reader has been removed.

Example:

```
SCARDCONTEXT hContext;
SCARDHANDLE hCard;
DWORD dwActiveProtocol, cReaders;
DWORD dwState, dwProtocol, dwAtrLen;
BYTE pbAtr[MAX_ATR_SIZE]
LPSTR mszReaders;
LONG rc;
rc = SCardEstablishContext( SCARD_SCOPE_SYSTEM, NULL, NULL, &hContext );
rc = SCardConnect( hContext, iCReader XlR, SCARD_SHARE_SHARED,
                   SCARD_PROTOCOL_T0, &hCard, &dwActiveProtocol );
mszReaders = (LPSTR)malloc(sizeof(char)*50);
rc=SCardStatus( hCard, mszReaders, 50, &dwState, &dwProtocol, pbAtr,
                &dwAtrLen );
```

SpringCard functions – not PC/SC

SCardAllocMemory

SCardStringToBin

SCardHexToString

SCardBinToString

SCardErrorToString

SCardISOErrorToString

SCardMultiStringItem

SCardMultiStringCount

SCardAllocMemory

Synopsis:

```
SCardAllocMemory( IN SCARDCONTEXT hContext,
                  INOUT LPVOID pvMem,
                  IN DWORD dwSize);
```

Parameters:

hContext	Connection context.
pvMem	Pointer on the buffer.
dwSize	Size of the buffer to allocate.

Description:

This function allocates a buffer whose size is worth dwSize, and return the beginning of this buffer in pvMem.

Returns:

SCARD_S_SUCCESS Succesful
SCARD_E_NO_MEMORY Failure

SCardStringToBin

Synopsis:

```
LONG SCardStringToBin( IN SCARDCONTEXT hContext,
                       INOUT LPVOID pvResult,
                       IN DWORD dwResultLen,
                       IN LPCTSTR szText);
```

Parameters:

hContext	Connection context.
pvResult	Pointer to memory area that receive the result of the translation
dwResultLen	Length of the memory area pointed to be pvResult
szText	Pointer to string

Description:

This function translate an HEX encoded string into binary. The size of the string to translate must be smaller than dwResultLen.

Returns:

SCARD_S_SUCCESS	Successful.
SCARD_E_INSUFFICIENT_BUFFER	The size of the buffer is insufficient.
SCARD_E_INVALID_PARAMETER	pvResult or/and szText is/are NULL.

SCardHexToByte

Synopsis:

```
LONG SCardHexToByte( IN SCARDCONTEXT hContext,
                     INOUT LPBYTE pbResult,
                     IN LPCTSTR szText);
```

Parameters:

hContext	Connection context.
pbResult	Pointer to a byte that receive the result of the translation.
szText	String to convert.

Description:

This function translate an HEX encoded string into one byte.

Returns:

SCARD_S_SUCCESS	Succesful
SCARD_E_INVALID_PARAMETER	pbResult or/and szText is/are NULL.

SCardHexToString

Synopsis:

```
LONG SCardHexToString( IN SCARDCONTEXT hContext,
                       INOUT LPTSTR szResult,
                       IN DWORD dwResultLen,
                       IN LPCTSTR szText);
```

Parameters:

hContext	Connection context.
szResult	Pointer to string that receive the result of the translation.
dwResultLen	Length of the memory area pointed to be pvResult.
szText	String to convert.

Description:

This function translate an HEX encoded string into an ASCII string.

Returns:

SCARD_S_SUCCESS	Succesful
SCARD_E_INSUFFICIENT_BUFFER	The size of the buffer is insufficient.
SCARD_E_INVALID_PARAMETER	pvResult or/and szText is/are NULL.

SCardBinToString

Synopsis:

```
LONG SCardBinToString( IN SCARDCONTEXT hContext,
                       INOUT LPTSTR szResult,
                       IN DWORD dwResultLen,
                       INOUT LPVOID pvMem,
                       IN DWORD dwSize);
```

Parameters:

hContext	Connection context.
szResult	Pointer to string that receive the result of the translation
dwResultLen	Length to the string pointed by szResult
pvMem	Pointer to the memory area to be translated
dwSize	Length of the memory area to be translated

Description:

This function translate a buffer into a HEX encoded string.

Returns:

SCARD_S_SUCCESS	Succesful
SCARD_E_INSUFFICIENT_BUFFER	The size of the buffer is insufficient.
SCARD_E_INVALID_PARAMETER	pvResult or/and szText is/are NULL.

SCardErrorToString

Synopsis:

```
LPTSTR SCardErrorToString( IN SCARDCONTEXT hContext,
                           DWORD dwErrCode);
```

Parameters:

hContext	Connection context.
dwErrCode	Error code to translate

Description:

This function translate a PC/SC error code to a human readable explanation string.

Returns:

Pointer to a string.

SCardISOErrorToString

Synopsis:

```
LPTSTR SCardISOErrorToString( IN SCARDCONTEXT hContext,
                              BYTE sw1,
                              BYTE sw2);
```

Parameters:

hContext	Connection context.
sw1	ISO return code SW1
sw2	ISO return code SW2

Description:

This function translate an ISO SW pair to a human readable explanation string.

Returns:

Pointer to a string.

SCardMultiStringCount

Synopsis:

```
DWORD SCardMultiStringCount( IN SCARDCONTEXT hContext,
                             IN LPCTSTR szMulti);
```

Parameters:

hContext	Connection context.
szMulti	A multi-string character string (double '\0' ended).

Description:

This function counts the strings in a multi-string buffer (for example, in the buffer returned by SCardListReaders).

Returns:

The number of strings found.

SCardMultiStringItem

Synopsis:

```
LPCTSTR SCardMultiStringCount( IN SCARDCONTEXT hContext,
                               IN LPCTSTR szMulti,
                               IN DWORD dwItem);
```

Parameters:

hContext	Connection context.
szMulti	A multi-string character string (double '\0' ended).
dwItem	The index of the string to be returned.

Description:

This function select and returns one string in a multi-string buffer (for example, in the buffer returned by SCardListReaders).

Returns:

A pointer to the selected string, NULL if error.

PC/SC functions not available under the SpringCard API

SCardListReaderGroups

SCardListCards

SCardListInterfaces

SCardGetProviderId

SCardGetCardTypeProviderName

SCardIntroduceReaderGroup

SCardForgetReaderGroup

SCardIntroduceReader

SCardForgetReader

SCardAddReaderToGroup

SCardRemoveReaderFromGroup

SCardIntroduceCardType

SCardSetCardTypeProviderName

SCardForgetCardType

SCardLocateCards

SCardGetStatusChange

SCardCancel

SCardReconnect

SCardBeginTransaction

SCardCancelTransaction

SCardControl

SCardGetReaderCapabilities / SCardGetAttrib

SCardSetReaderCapabilities / SCardSetAttrib

Using the SpringCard-VS with synchronous cards

Thanks to the GemCore chipset, the SpringCard-VS can access synchronous memory card as easily as they where T=0 or T=1 cards.

Please refer to GemPlus documentation “GemCore Chipset Controller Software version 2.0”, chapter “Using the GemCore chipset controller with memory cards” for more information.

Example

This sample shows how to read 64 bytes, starting at offset 32, from an S=9 memory card with the SpringCard-VS.

```
LONG rc;
SCARDCONTEXT hContext; SCARDHANDLE hCard;
DWORD dwActiveProtocol, dwRecvLength;
BYTE pbRecvBuffer[64+2];         // length = 64 + sw1,sw2
BYTE pbSendBuffer = { 0xB0,
                      0x00,
                      0x00,
                      32,        // offset = 32
                      64         // length = 64
                    };
rc = SCardEstablishContext( SCARD_SCOPE_SYSTEM,
                            NULL,
                            NULL,
                           &hContext );
rc = SCardConnect( hContext,
                   "A",
                   SCARD_SHARE_EXCLUSIVE,
                   SCARD_PROTOCOL_RAW | 0x08,  // S=9 is referred as GPM8K,
                                               // this is internal function
                                               // DEFINE_CARD_TYPE( 0x08 )
                  &hCard,
                  &dwActiveProtocol);
dwRecvLength = sizeof(pbRecvBuffer);
rc = SCardTransmit( hCard,
                    NULL,
                    pbSendBuffer,
                    sizeof(pbSendBuffer),
                    NULL,
                    pbRecvBuffer,
                   &dwRecvLength );
```

Using the SpringCard-CF with synchronous cards

Due to the lack of synchronous driver in the GemCore Lite chipset, the SpringCard-CF requires more work to access those memory cards.

Please refer to GemPlus documentation “GemCore Chipset Controller Software version 2.0”, chapter “Interpreted synchronous smartcard driver” for more information.

Example

This sample shows how to read 64 bytes, starting at offset 32, from an S=10 memory card with the SpringCard-CF.

```
/* S=10 Power-Up command in 8051 code */
static BYTE S10_POWER_UP_CMD[] =
{
  0x16,
  0x00, 0x12, 0x00, 0x00, 0x00, 0x00,
  0x02, 0x11, 0x12
};

/* S=10 Read command in 8051 code */
static BYTE S10_READ_CMD[] =
{
  0x16, /* Synchronous interpreter command */
  0x00, /* don't care */
  0xB0, /* don't care */
  0xCC, /* Will receive offset H */
  0xCC, /* Will receive offset L */
  0x00, /* don't care */
  0xCC, /* Will receive length */
  0x3E, 0x21, 0xBE, 0x00, 0x14, 0x51, 0x71, 0x41, 0x74, 0x30,
  0x93, 0xEF, 0x93, 0x74, 0xFF, 0x93, 0x41, 0x71, 0x82, 0xF6,
  0x08, 0xDB, 0xFB, 0xA2, 0x42, 0xBE, 0x80, 0x02, 0x80, 0x03,
  0xBE, 0xC0, 0x1B, 0x51, 0x71, 0x41, 0xBE, 0xC0, 0x04, 0x74,
  0x31, 0x80, 0x02, 0x74, 0x34, 0x93, 0x74, 0xFF, 0x93, 0x93,
  0x41, 0x71, 0x7B, 0x04, 0x82, 0xF6, 0x08, 0xDB, 0xFB, 0x42,
  0x62, 0x6D, 0x00
};
LONG rc;
SCARDCONTEXT hContext; SCARDHANDLE hCard;
DWORD dwActiveProtocol, dwRecvLength;
SCARD_IO_REQUEST pioRecvPci;
BYTE pbRecvBuffer[64+2];         // length = 64 + sw1,sw2

rc = SCardEstablishContext( SCARD_SCOPE_SYSTEM,
                            NULL,
                            NULL,
                           &hContext );
```

.../...

```
rc = SCardConnect( hContext,
                   "A",
                   SCARD_SHARE_EXCLUSIVE,
                   SCARD_PROTOCOL_RAW,    // The GemCore Lite doesn't care
                                          // of the card type here.
                  &hCard,
                  &dwActiveProtocol);

/*
   At this point, the reader has checked that there is a card in the slot;
   it has turn the slot mode to ISO + synchronous interpreted driver.
   We must at first power-up the card !
*/

dwRecvLength = sizeof(pbRecvBuffer);
rc = ScardControl( hCard,
                   S10_POWER_UP_CMD,
                   sizeof(S10_POWER_UP_CMD),
                   pbRecvBuffer,
                  &dwRecvLength );

/*
   We can't expect any return code, so we don't care for the result !
   Now, let's build the read command.
*/

S10_READ_CMD[3] = 0;   // offset H = 0
S10_READ_CMD[4] = 32;  // offset L = 32
S10_READ_CMD[6] = 64;  // length   = 64

dwRecvLength = sizeof(pbRecvBuffer);
rc = ScardControl( hCard,
                   S10_READ_CMD,
                   sizeof(S10_READ_CMD),
                   pbRecvBuffer,
                  &dwRecvLength );

/*
   We can check sw1, sw2, to make sure we have supplied a "valid" 8051
   code, but there is no way to know for sure if the returned data are
   valid or not (if the card is not an S=10, or hasn't been powered
   before, we should receive 0xFF, 0xFF, 0xFF, ...)
*/
```

Direct access to the SpringCard

For developers who don't want to use the PC/SC API, it is still possible to use the SpringCard as a regular GemCore based serial reader. GemPlus TLP driver or over specific software can use the serial port to access both slots A et B.

Under PalmOS (Visor)

The SpringCard is seen as an UART mapped into the CPU memory. Source code supplied under the Visor directory can provided serial port access in a way compatible with the (old) SerialManager API.

Under Windows CE (PocketPC)

The SpringCard is seen as a serial port. As the serial port number is assigned by the plug'n'play subsystem and can't be guessed, you must lookup into the registry to find the current number (see example in LibSrc/reader.c).
You can use the Windows API (CreateFile, ReadFile, WriteFile) to get direct access to the GemCore. Opening the serial port powers up the GemCore, closing the serial port powers it down.

NOTES .....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

PRO ACTIVE
PARC GUTENBERG
13 VOIE LA CARDON
F-91120 PALAISEAU
FRANCE

Phone +33 164 532 010
Fax. +33 164 532 018

Web site
http://www.pro-active.fr

e-Mail address
info@pro-active.fr

SpringCard web site
http://www.springcard.com