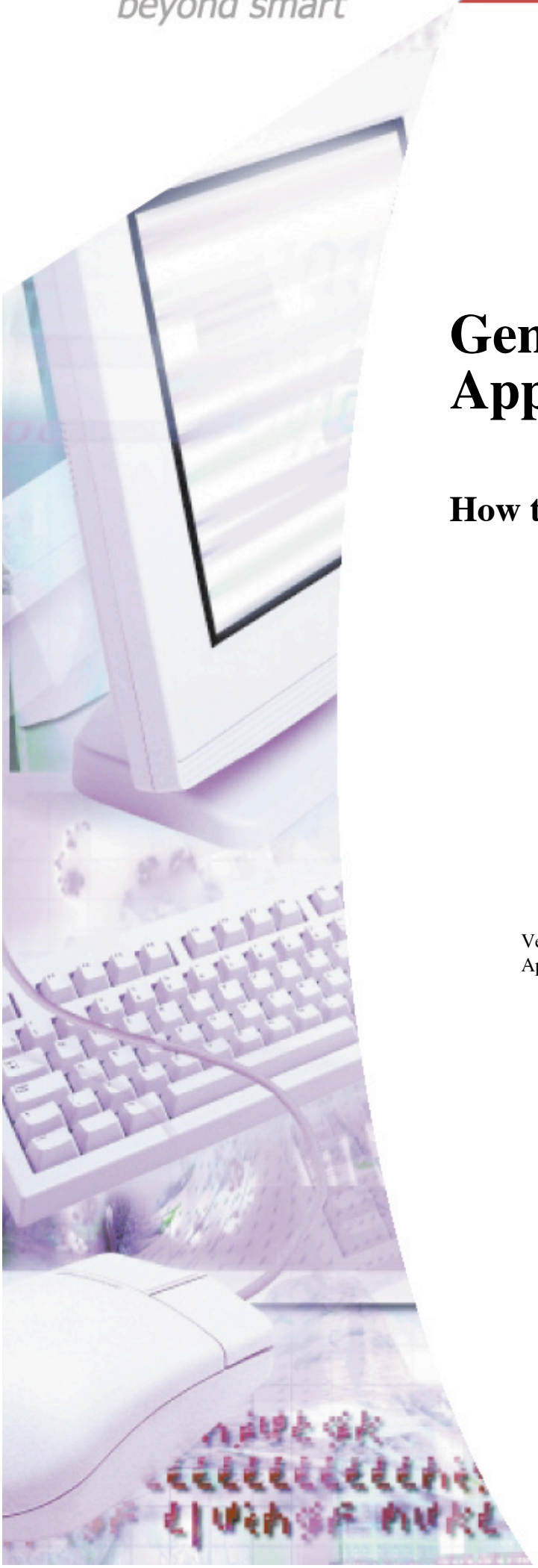
GEMPLUS
beyond smart

GemPC Twin



Gemcore Twin Pro Application Note

How to use Synchronous Cards

Preliminary

Version V0.93
April 16, 2004

GEMPLUS Technology

All information herein is either public information or is property of and owned solely by Gemplus who shall have and keep the sole right to file patent application or any kind of intellectual property protection in connection with such information.

Nothing herein shall be construed as implying or granting to you any rights, by license, grant or otherwise, under any intellectual and/or industrial property rights of or concerning any of Gemplus' information.

This document can be used for informational, non-commercial, internal and personal use only provided that:

- The copyrights notice below, the confidentiality and proprietary legend and this full warning notice appear in all copies.
- This document shall not be posted on any network computer or broadcast in any media and no modification of any part of this document shall be made.

Use for any other purpose is expressly prohibited and may result in severe civil and criminal liabilities.

The information contained in this document is provided "AS IS" without any warranty of any kind. Unless otherwise expressly agreed in writing, Gemplus makes no warranty as to the value or accuracy of information contained herein. The document could include technical inaccuracies or typographical errors. Changes are periodically added to the information herein. Furthermore, Gemplus reserves the rights to make any change or improvement in the specifications data, information, and the like described herein, at any time.

Gemplus hereby disclaims all warranties and conditions with regard to the information contained herein, including all implied warranties of merchantability, fitness for a particular purpose, title and non-infringement. In no event shall Gemplus be liable, whether in contract, tort or otherwise, for any indirect, special or consequential damages or any damages whatsoever including but not limited to damages resulting from loss of use, data, profits, revenues, or customers, arising out of or in connection with the use or performance of information contained in this document.

Gemplus does not and shall not warrant that this product will be resistant to all possible attacks and shall not incur, and disclaims, any liability in this respect. Even if each product is compliant with current security standards in force on the date of their design, security mechanisms' resistance necessarily evolves according to the state of the art in security and notably under the emergence of new attacks. Under no circumstances, shall Gemplus be held liable for any third party actions and in particular in case of any successful attack against systems or equipment incorporating Gemplus products. Gemplus disclaims any liability with respect to security for direct, indirect, incidental or consequential damages that result from any use of its products. It is further stressed that independent testing and verification by the person using the product is particularly encouraged, especially in any application in which defective, incorrect or insecure functioning could result in damage to person or property, denial of service or loss of privacy.

Content

Preface **4**
Scope **4**
Introduction **4**
Proprietary Commands **5**
Command chaining **5**
Description of the common input parameters **6**
Description of the commands **7**
GCSInit 7
GCSClose 7
GSCPowerUp 7
GSCPowerDown 7
GSCWrite 8
GSCRead 8
GSCWriteProtectionBits 9
GSCReadProtectionBits 9
GSCWriteSecurityMemory 10
GSCReadSecurityMemory 10
GSCVerify 10
Description of the output parameter **11**
Application notes **12**
Synchronous card short datasheet **13**
Protocol 2-wire : SLE 4432/42 (GPM2k) **14**
GSCPowerUp 14
GSCWrite 14
GSCRead 14
GSCWriteProtectionBits 15
GSCReadSecurityMemory (SLE4442 only) 16
GSCVerify (SLE4442 only) 16
Protocol 3-wire : SLE 4418/28 (GPM8k) **17**
GSCPowerUp 17
GSCWrite 17
GSCRead 17
GSCReadProtectionBits 18
GSCWriteSecurityMemory (SLE4428 only) 18
GSCVerify (SLE4428 only) 19
Protocol SDA : GFM2k … 32k **20**
GSCPowerUp 20
GSCWrite 20
GSCRead 20
Protocol 4403 : GPM103 ..... **21**
GSCPowerUp 21
GSCRead 21
Protocol 4433 : GAM226/326 ..... **21**
GSCPowerUp 21
GSCRead 21
ANNEXE 1: Extracts of GSC_DEF.H **22**
ANNEXE 2: Examples **23**

Preface

This document describes the use of the Gemplus-proprietary functions to manage the synchronous cards with the readers using the GemCore Twin Pro chipset.
All these functions are in the API for synchronous cards GSC.DLL version 1.0.0.7.
Some values of common parameters are defined in GSC_DEF.H.

For additional information about the PC/SC API, refer to the Software Development Kit for Microsoft WIN32.

Audience

This document is intended for developers of PC/SC-based applications. It requires familiarity with the Microsoft PC/SC application-programming interface (API).
See at http://support.gemplus.com for downloading then installing the GemPC PC/SC reader drivers on your Windows Platform.

Scope

This document applies to the readers implementing the GemCore Twin Pro chipset:
GemPC Twin
GemPC USB
GemPC Key
GemPC Serial
GemPC Card

For the support of the following synchronous cards families:
2 wire card type: **SLE 4432/42, GPM2k**
3 wire card type: **SLE 4418/28, GPM8k**
SDA wire card type: **GFM 2k, 4k … 32k**
4403 card type: **GPM 103**
4433 card type: **GAM 226/326**

Introduction

Any PC/SC application designed for the synchronous cards should follow the steps below:

1. Establish a PC/SC context with a call to SCardEstablishContext.
2. Get the reader identifier from a list of installed readers with SCardListReaders.
3. Connect to that reader with a call to SCardConnect.
4. Handle communication with your synchronous card by sending commands directly to that reader through calls to the SCardControl function.
5. Terminate the connection to the reader with a call to SCardDisconnect.
6. Close an established context with a call to SCardReleaseContext.

In the user application, the access to the reader and to the smart card is handled by a resource manager (ICC RM) that uses the Windows PC/SC API.
The smart card reader driver translates the requests coming from the smart card resource manager into the reader' s language. It also chooses and sets the appropriate USB or serial channel to communicate with the smart card reader.

The ICC Resource Manager is a component of the Windows system that has the following tasks:
Manage database information about the installed readers, the installed cards, the cards inserted and the ATR of these cards.
Allow the resources for the applications.
Manage the access to the cards.

The Microsoft **winscard.h** module contains the ICC RM functions prototypes, the constants and the structures declaration. To import the resource manager functions, link the **winscard.lib** library to your project. For more information see at **http://msdn.microsoft.com**

Proprietary Commands

In the PC/SC application, the following commands make internal calls to the SCardControl function. So, they need a handle returned by a previous call to the SCardConnect function.
This handle is the 1st parameter of ScardControl. The 2nd parameter internal used in ScardControl is dwControlCode = SCARD_CTL_CODE(2048).

Function	Operation
GCSInit	Initialise synchronous card mode
GCSClose	Close synchronous card mode
GSCPowerUp	Power up the card
GSCPowerDown	Power down the card
GSCWrite	Write to Main Memory
GSCRead	Read from Main Memory
GSCWriteProtectionBits	Write to Protection Memory
GSCReadProtectionBits	Read from Protection Memory
GSCWriteProtectionMemory	Write to Security Memory
GSCReadProtectionMemory	Read from Security Memory
GSCVerify	Check Secret Code

The calling convention used by the DLL is __stdcall, so it can be interfaced with different languages (C/C++, Visual Basic …)
To avoid warnings during the compilation of a C application, rename the main() function in :
void __cdecl main()
Because any function in C with a variable argument list must use the __cdecl calling convention.

Command chaining

These commands can be chained as in the following example :
The commands into brackets are optional.

GCSInit
GSCPowerUp
[GSCRead]
[GSCWrite]
[GSCVerify]
…
GSCPowerDown

[GSCPowerUp]
[GSCRead]
[GSCWrite]
[GSCVerify]
…
[GSCPowerDown]
…
GCSClose

Description of the common input parameters

Parameter	Description
IN SCARDHANDLE hCard	Handle returned from ScardConnect (with dwShareMode = SCARD_SHARE_DIRECT)
IN BYTE bCardType	Type of synchronous card. The values are read-only.
IN BYTE bSubCardType	Sub type of synchronous card.

bCardType value	**Description**
ICC_2_WIRE	2 wire card type (SLE 4432/42, GPM2k)
ICC_3_WIRE	3 wire card type (SLE 4418/28, GPM8k)
ICC_SDA	SDA wire card type (GFM 2k, 4k … 32k)
ICC_4403	4403 card type (GPM 103)
ICC_4433	4433 card type (GAM 226/326)

bSubCardType value	**Description**
ADDR_1_BYTE	1 byte address for 2 wire card type
ADDR_2_BYTES	2 bytes address for 2 wire card type
ADDR_3_BYTES	3 bytes address for 3 wire card type
PAGE_8_BYTES	8 bytes page write for SDA card type
PAGE_16_BYTES	16 bytes page write for SDA card type
PAGE_32_BYTES	32 bytes page write for SDA card type
PAGE_64_BYTES	64 bytes page write for SDA card type
PAGE_128_BYTES	128 bytes page write for SDA card type
NO_SUB_CARD_TYPE	For type of card don't have any sub card type

Note that vendors may not support all attributes.

For the moment, about the read operation and the write operation only the following combinations are authorized between bCardType and bSubCardType :

bCardType value	**bSubCardType** value(s) associated	Size of the main memory of the smartcard in bytes
ICC_2_WIRE	ADDR_1_BYTE	256
ICC_3_WIRE	NO_SUB_CARD_TYPE	1024
ICC_4403	NO_SUB_CARD_TYPE	13
ICC_4433	NO_SUB_CARD_TYPE	48
ICC_SDA	PAGE_8_BYTES (for GFM 2k) PAGE_16_BYTES (for GFM4k…16k) PAGE_32_BYTES (for GFM32k)	(256) (512…2048) (4096)

Description of the commands

These commands use common parameters defined above.
All the commands return a **LONG** as output parameter described further.

GCSInit

Initialise the library and select 5 Volt synchronous cards mode.

	Parameter	Description
P1	**IN SCARDHANDLE** hCard	handle

Can be used to create new library connection with synchronous smart card and initialise card specific context. Card handle can be received from Resource Manager ScardConnect() call. This function should be called for each newly inserted smart card. By initialising new library connection, the client requests the library and IFD handler to initialise specific synchronous card support. This support will be valid only for current smart card and will be automatically reset by the IFD handler upon new card insertion/removal notification. It is still recommended to provide corresponding GCSClose() call to close synchronous card session. If any of synchronous card commands will be sent while smart card is absent, the library will report error GEM_E_NO_SMARTCARD.

GCSClose

This function terminates the library connection with synchronous smart card, invalidates card context, and then restores the asynchronous driver mode.

	Parameter	Description
P1	**IN SCARDHANDLE** hCard	handle

GSCPowerUp

This function provides a reset of the card. If parameter pbRecvBuffer is equal to NULL, the function provides card power up without any attempt to read specific smart card, else the 7816-10 reset for synchronous card type 1 (as for SLE4432/42 or SLE4418/28) is performed.

	Parameter	Description
P1	**IN SCARDHANDLE** hCard	handle
P2	**IN OUT LPBYTE** pbRecvBuffer	Pointer to any data returned from the card after reset. Can be NULL. In this case the caller is not interesting with card reply on Reset and this function just powers up the card, without attempt to read it.
P3	**IN OUT LPDWORD** pcbRecvLength	Supplies the length of the *pbRecvBuffer* parameter (in bytes) and receives the actual number of bytes received from the smart card.

Caution: The output buffer must have been allocated with P3 or more bytes before calling the function.

GSCPowerDown

This function performs a power down of synchronous card.

	Parameter	Description
P1	**IN SCARDHANDLE** hCard	handle

GSCWrite

This function writes a stream of data bytes to the main memory of a synchronous card.
The data are written without any verification.

	Parameter	Description
P1	**IN SCARDHANDLE** hCard	handle
P2	**IN BYTE** bCardType	Card Type
P3	**IN BYTE** bSubCardType	Sub Card Type
P4	**IN DWORD** dwAddressToWrite	Address in smart-card to write data
P5	**IN LPBYTE** pbByteToWrite	Pointer to the data to send to the card
P6	**IN DWORD** dwLengthToWrite	Length in bytes of data to write to synchronous card. The max length is guaranteed until 256 for the upper than 256-bytes synchronous cards.

GSCRead

This function reads a stream of data bytes from the main memory of a synchronous card.

	Parameter	Description
P1	**IN SCARDHANDLE** hCard	handle
P2	**IN BYTE** bCardType	Card Type
P3	**IN BYTE** bSubCardType	Sub Card Type
P4	**IN DWORD** dwAddressToRead	Origin address in synchronous card to read data.
P5	**IN DWORD** dwLengthToRead	Number of bytes expected to read
P6	**OUT LPBYTE** pbRecvBuffer	Pointer to the data received
P7	**IN OUT LPDWORD** pdRecvBufferLength	Pointer on buffer size on call Pointer of the length received on exit

Caution: The output buffer must have been allocated with P5 or more bytes before calling the function.

bCardType value	**dwLengthToRead** max value in bytes	**Cause of the limitation**
ICC_2_WIRE	256	synchronous card mapping
ICC_3_WIRE	1024	synchronous card mapping
ICC_SDA	256 guaranteed	DLL's buffer
ICC_4403	13	synchronous card mapping
ICC_4433	48	synchronous card mapping

(dwAddressToRead + dwLengthToRead) must be less than or equal the memory size of the card, expressed in bytes. Idem for (dwAddressToWrite + dwLengthToWrite).

GSCWriteProtectionBits

This function blows a bit in the Protection Memory of a synchronous card, to definitively protect the associated byte located in the main memory against any writing.

The execution of this command contains a comparison of the entered data byte with the assigned byte in the EEPROM. When this comparison is correct, the protection bit is written (made "0" for the GPM2K/8K), making the data unchangeable. If the comparison results in a difference, the protection bit cannot be written.

	Parameter	**Description**
P1	**IN SCARDHANDLE** hCard	handle
P2	**IN BYTE** bCardType	Card Type
P3	**IN BYTE** bSubCardType	Sub Card Type
P4	**IN DWORD** dwAddressToWrite	Address of the assigned byte to protect
P5	**IN LPBYTE** pbByteToWrite	Pointer to the data to protect to send to the card
P6	**IN DWORD** dwLengthToWrite	Data number to write (1 recommended)

GSCReadProtectionBits

This function reads a stream of 8-bits packets from the Protection Memory of a synchronous card.

	Parameter	**Description**
P1	**IN SCARDHANDLE** hCard	handle
P2	**IN BYTE** bCardType	Card Type
P3	**IN BYTE** bSubCardType	Sub Card Type
P4	**IN DWORD** dwAddressToRead	Address A of a 8-bits block in the protection memory
P5	**IN DWORD** dwLengthToRead	Number N of 8-bits blocks to read
P6	**OUT LPBYTE** pbRecvBuffer	Pointer to the N * 8 bits of protection memory
P7	**IN OUT LPDWORD** pdRecvBufferLength	Pointer on buffer size on call Pointer of the length received on exit

Caution: The output buffer must have been allocated with P5 or more bytes before calling the function.

A Main Memory's byte is definitively protected when the associated protection bit is blown (that is, 0 for the GPM2K/8K).

The LSB of the first response byte is the protection bit of the byte at (8 * A) in the main memory, the MSB of the first response byte is the protection bit of the byte at (8 * A) + 7 in the main memory, etcetra.

The LSB of the Nth response byte is the protection bit of the byte at 8 * (A + N-1) in the main memory, the MSB of the Nth response byte is the protection bit of the byte at 8 * (A + N-1) + 7 in the main memory.

GSCWriteSecurityMemory

This function writes a stream of data bytes to the Security Memory of a synchronous card.

	Parameter	Description
P1	**IN SCARDHANDLE** hCard	handle
P2	**IN BYTE** bCardType	Card Type
P3	**IN BYTE** bSubCardType	Sub Card Type
P4	**IN DWORD** dwAddressToWrite	Address in smart-card security memory to write data
P5	**IN LPBYTE** pbByteToWrite	Pointer to the data to send to the card
P6	**IN DWORD** dwLengthToWrite	Length in bytes of data to write to synchronous card. The max length is guaranteed until 256 for the upper than 256-bytes synchronous cards.

GSCReadSecurityMemory

This function reads a stream of data bytes from the Security Memory of a synchronous card.

	Parameter	Description
P1	**IN SCARDHANDLE** hCard	handle
P2	**IN BYTE** bCardType	Card Type
P3	**IN BYTE** bSubCardType	Sub Card Type
P4	**IN DWORD** dwAddressToRead	Origin address in synchronous card to read data.
P5	**IN DWORD** dwLengthToRead	Number of bytes expected to read
P6	**OUT LPBYTE** pbRecvBuffer	Pointer of the data receive
P7	**IN OUT LPDWORD** pdRecvBufferLength	Pointer on buffer size on call Pointer of the length received on exit

Caution : The output buffer must have been allocated with P5 or more bytes before calling the function.

GSCVerify

This function verifies a secret code of a synchronous card.

	Parameter	Description
P1	**IN SCARDHANDLE** hCard	handle
P2	**IN BYTE** bCardType	Card Type
P3	**IN DWORD** dwCodeLength	Length in bytes of code to verify
P4	**IN LPBYTE** pbCodeToVerify	Pointer to code to verify

Description of the output parameter

These functions return a LONG with different values depending on whether they succeed or fail.

Outcome with success	**Return value**
Success	GEM_SUCCESS

Outcome with Failure	**Return value**
Incorrect smart card handle (handle NULL)	GEM_E_INVALID_HANDLE
Incorrect parameters (pointer NULL …)	GEM_E_INVALID_PARAMETERS
Incorrect length	GEM_E_INVALID_LENGTH
Incorrect address	GEM_E_INVALID_ADDRESS
Incorrect length versus address …	GEM_E_EARLY_END_OF_FILE
Protocol not supported.	GEM_E_PROTOCOL_NOT_SUPPORTED
No reader available.	GEM_E_NO_READERS_AVAILABLE
No ATR pursuant to ISO 7816-3 could be read. And no protocol was recognized. (first character received is 00h or 0FFh).	GEM_E_UNKNOWN_CARD
No error can be retrieved from hardware	GEM_E_UNKNOWN_ERROR
Incorrect card type or function not supported for this card type, incorrect sub card type or function not supported for this sub card type.	GEM_E_CARD_UNSUPPORTED
Buffer length too small	GEM_E_INSUFFICIENT_BUFFER
No ICC in the contact unit.	GEM_E_NO_SMARTCARD
Access not possible (No acknowledge with SDA; Card locked …)	GEM_E_NO_ACCESS
Writing unsuccessful (Bad acknowledge with SDA, incomplete writing with 2 & 3-wire protocol)	GEM_E_WRITE_ERROR
Wrong password.	GEM_E_WRONG_PWD
Resource Manager unavailable	GEM_E_NO_SERVICE

Return values implemented for each command
They are listed at the x-marked intersections :

Return values ↓ Command →	GCS Init	GCS Close	GSC Power Up	GSC Power Down	GSC Write …	GSC Read …	GSC Verify
GEM_SUCCESS	x	x	x	x	x	x	x
GEM_E_INVALID_HANDLE	x	x	x	x	x	x	x
GEM_E_INVALID_PARAMETERS	x	x	x	x	x	x	x
GEM_E_INVALID_LENGTH					x	x	x
GEM_E_INVALID_ADDRESS					x	x	
GEM_E_EARLY_END_OF_FILE					x	x	
GEM_E_PROTOCOL_NOT_SUPPORTED					x	x	
GEM_E_NO_READERS_AVAILABLE	x		x	x	x	x	x
GEM_E_UNKNOWN_CARD			x				
GEM_E_UNKNOWN_ERROR			x				
GEM_E_CARD_UNSUPPORTED					x	x	x
GEM_E_INSUFFICIENT_BUFFER			x			x	x
GEM_E_NO_SMARTCARD			x	x	x	x	x
GEM_E_NO_ACCESS						x	x
GEM_E_WRITE_ERROR					x		
GEM_E_WRONG_PWD							x
GEM_E_NO_SERVICE	x		x	x	x	x	x

Application notes

The DLL does not manage the card withdrawals. This task shall be handled by the PC/SC application.

Card Detection: the application should use the function ScardGetStatusChange() to know if a card is inserted or not. If smart card is not present while card handle is still valid, the library will report error GEM_E_NO_SMARTCARD from each function except GCSClose().

ScardConnect: to get smart card handle the application should use a ScardConnect in SCARD_SHARE_DIRECT mode. The library to get communications with IFD handler through Resource Manager will use this handle.

Error Codes: library reports error codes defined in GSC_DEF.H.

Synchronous card short datasheet

All the operations implemented for each synchronous card are summarized below:

Protocol 2-wire : SLE 4432/42

Function	SLE4432/42 ‘s Operation used
GSCWrite	Update Main Memory
GSCRead	Read Main Memory
GSCWriteProtectionBits	Write Protection Memory
GSCReadProtectionBits	Read Protection Memory
GSCWriteSecurityMemory	Update Security Memory
GSCReadSecurityMemory	Read Security Memory
GSCVerify	Read Security Memory, Update Security Memory and Compare Verification Data operations of the SLE4442 (only)

Protocol 3-wire : SLE 4418/28

Function	SLE4418/28 ‘s Operation used
GSCWrite	Write Erase Without Protect Bit
GSCRead	Read 8 Bit Data Without Protect Bit
GSCWriteProtectionBits	Data Comparison and Write Protect Bit
GSCReadProtectionBits	Read 9 Bits, Data with Protect Bit
GSCWriteSecurityMemory	Write Erase Without Protect Bit
GSCReadSecurityMemory	Read 8 Bit Data Without Protect Bit
GSCVerify	Read 8 Bit Data Without Protect Bit, Write Error Counter, Verify 1st PSC byte, Verify 2nd PSC byte, and Write Erase Without Protect Bit operations of the SLE4428 (only)

Protocol SDA

Function	SDA ‘s Operation used
GSCWrite	Byte Write
GSCRead	Random Read followed by Sequential Read

Protocol 4403 : GPM103

Function	SDA ‘s Operation used
GSCRead	Read sequence of the GPM103

Protocol 4433 : GAM226/326

Function	SDA ‘s Operation used
GSCRead	Read sequence of the GAM226

Protocol 2-wire : SLE 4432/42 (GPM2k)

Notation :
the arrow "→" means "pointer to"
PSC means Programmable Security Code

GSCPowerUp

	Parameter	Value
P1	**IN SCARDHANDLE** hCard	(≠ NULL)
P2	**IN OUT LPBYTE** pbRecvBuffer	(≠ NULL) → the P3 data received
P3	**IN OUT LPDWORD** pcbRecvLength	(≠ NULL) IN → length of the data expected : **4** bytes for ATR OUT → length of the data really received

GSCWrite

	Parameter	Value
P1	**IN SCARDHANDLE** hCard	(≠ NULL)
P2	**IN BYTE** bCardType	ICC_2_WIRE
P3	**IN BYTE** bSubCardType	ADDR_1_BYTE
P4	**IN DWORD** dwAddressToWrite	00h … FFh
P5	**IN LPBYTE** pbByteToWrite	(≠ NULL) → the P6 data
P6	**IN DWORD** dwLengthToWrite	1 to 256 (with P4+P6 ≤ 256)

GSCRead

	Parameter	Value
P1	**IN SCARDHANDLE** hCard	(≠ NULL)
P2	**IN BYTE** bCardType	ICC_2_WIRE
P3	**IN BYTE** bSubCardType	ADDR_1_BYTE
P4	**IN DWORD** dwAddressToRead	00h … FFh
P5	**IN DWORD** dwLengthToRead	1 to 256 (with P4+P5 ≤ 256)
P6	**OUT LPBYTE** pbRecvBuffer	(≠ NULL) → the P7 data received (P5 data expected)
P7	**IN OUT LPDWORD** pdRecvBufferLength	(≠ NULL) IN → buffer size ≥ P5 OUT → length received (P5 data expected)

GSCWriteProtectionBits

	Parameter	Value
P1	**IN SCARDHANDLE** hCard	(≠ NULL)
P2	**IN BYTE** bCardType	ICC_2_WIRE
P3	**IN BYTE** bSubCardType	ADDR_1_BYTE
P4	**IN DWORD** dwAddressToWrite	00h … 1Fh
P5	**IN LPBYTE** pbByteToWrite	(≠ NULL) → the P6 data
P6	**IN DWORD** dwLengthToWrite	1 at once recommended (but until 32 is possible)

GSCReadProtectionBits

The reading of all the 4*8 protection bits of the GPM2K is obtained with the values in **bold** :

	Parameter	Value
P1	**IN SCARDHANDLE** hCard	(≠ NULL)
P2	**IN BYTE** bCardType	ICC_2_WIRE
P3	**IN BYTE** bSubCardType	ADDR_1_BYTE
P4	**IN DWORD** dwAddressToRead	**00h** … 03h
P5	**IN DWORD** dwLengthToRead	1 to **4** (with P4+P5 ≤ 4)
P6	**OUT LPBYTE** pbRecvBuffer	(≠ NULL) → the P7 * 8 bits of protection memory (P7=P5 expected)
P7	**IN OUT LPDWORD** pdRecvBufferLength	(≠ NULL) IN → buffer size ≥ P5 OUT → length received (P5 data expected)

GSCWriteSecurityMemory (SLE4442 only)

Error Counter, CSC1, CSC2 and CSC3 are physically written at addresses 00h, 01h, 02h and 03h of the Security Memory.
Caution : Avoid writing in the Error Counter and in particular clearing its 3 lowest significant bits which authorize the 3 presentations of the secret codes before locking the card.

	Parameter	Value
P1	**IN SCARDHANDLE** hCard	(≠ NULL)
P2	**IN BYTE** bCardType	ICC_2_WIRE
P3	**IN BYTE** bSubCardType	ADDR_1_BYTE
P4	**IN DWORD** dwAddressToWrite	00h Error Counter 01h PSC1 02h PSC2 03h PSC3
P5	**IN LPBYTE** pbByteToWrite	(≠ NULL) → 1 to 4 bytes : [ERROR_COUNTER] [PSC1] [PSC2] [PSC3]
P6	**IN DWORD** dwLengthToWrite	1 to 4 (with P4+P6 ≤ 4)

GSCReadSecurityMemory (SLE4442 only)

The ERROR_COUNTER is always readable. PSC1, PSC2 and PSC3 are replaced with 0 if the 3-byte PSC has not been presented.
The number of the Error Counter bits at level 1 determines the number of possible attempts to present the secret codes. Only the 3 low bits are significant.
If the Error Counter returns 0 the card is definitively write locked.

	Parameter	**Value**
P1	**IN SCARDHANDLE** hCard	(≠ NULL)
P2	**IN BYTE** bCardType	ICC_2_WIRE
P3	**IN BYTE** bSubCardType	ADDR_1_BYTE
P4	**IN DWORD** dwAddressToRead	**00h** Error Counter 01h PSC1 02h PSC2 03h PSC3
P5	**IN DWORD** dwLengthToRead	1 to **4** (with P4+P5 ≤ 4)
P6	**OUT LPBYTE** pbRecvBuffer	(≠ NULL) → 1 to 4 bytes expected : **[ERROR_COUNTER] [PSC1] [PSC2] [PSC3]**
P7	**IN OUT LPDWORD** pdRecvBufferLength	(≠ NULL) IN → buffer size ≥ P5 OUT → length received (P5 data expected)

GSCVerify (SLE4442 only)

This function performs all the verification procedure: write one bit in Error Counter, send the three COMPARE VERIFICATION DATA commands, erase the counter error and read the Error Counter to check if verify is successful.

	Parameter	**Value**
P1	**IN SCARDHANDLE** hCard	(≠ NULL)
P2	**IN BYTE** bCardType	ICC_2_WIRE
P3	**IN DWORD** dwCodeLength	**3**
P4	**IN LPBYTE** pbCodeToVerify	(≠ NULL) → 3-byte PSC to verify

Protocol 3-wire : SLE 4418/28 (GPM8k)

GSCPowerUp

	Parameter	Value
P1	**IN SCARDHANDLE** hCard	(≠ NULL)
P2	**IN OUT LPBYTE** pbRecvBuffer	(≠ NULL) → the P3 data received
P3	**IN OUT LPDWORD** pcbRecvLength	(≠ NULL) IN → length of the data expected : **4** bytes for ATR OUT → length of the data really received

GSCWrite

	Parameter	Value
P1	**IN SCARDHANDLE** hCard	(≠ NULL)
P2	**IN BYTE** bCardType	ICC_3_WIRE
P3	**IN BYTE** bSubCardType	NO_SUB_CARD_TYPE
P4	**IN DWORD** dwAddressToWrite	0000h … 03FFh
P5	**IN LPBYTE** pbByteToWrite	(≠ NULL) → the P6 data
P6	**IN DWORD** dwLengthToWrite	1 to 1024 (with P4+P6 ≤ 1024)

Warning :
With the GPM8K ie SLE4428 only, the addresses 03FDh…03FFh are in fact the physical addresses of the Security Memory. These addresses of the Main Memory correspond respectively to the logical addresses 00…03 of the Security Memory.

GSCRead

	Parameter	Value
P1	**IN SCARDHANDLE** hCard	(≠ NULL)
P2	**IN BYTE** bCardType	ICC_3_WIRE
P3	**IN BYTE** bSubCardType	NO_SUB_CARD_TYPE
P4	**IN DWORD** dwAddressToRead	0000h … 03FFh
P5	**IN DWORD** dwLengthToRead	1 to 1024 (with P4+P5 ≤ 1024)
P6	**OUT LPBYTE** pbRecvBuffer	(≠ NULL) → the P7 data received (P5 data expected)
P7	**IN OUT LPDWORD** pdRecvBufferLength	(≠ NULL) IN → buffer size ≥ P5 OUT → length received (P5 data expected)

GSCWriteProtectionBits

	Parameter	Value
P1	**IN SCARDHANDLE** hCard	(≠ NULL)
P2	**IN BYTE** bCardType	ICC_3_WIRE
P3	**IN BYTE** bSubCardType	NO_SUB_CARD_TYPE
P4	**IN DWORD** dwAddressToWrite	0000h … 03FFh for the SLE4418 0000h … 03FCh for the SLE4428 (GPM8K)
P5	**IN LPBYTE** pbByteToWrite	(≠ NULL) → the P6 data
P6	**IN DWORD** dwLengthToWrite	1 at once recommended (but until 1024 is possible)

GSCReadProtectionBits

The reading of all the 128*8 protection bits of the GPM8K is obtained with the values in **bold** :

	Parameter	Value
P1	**IN SCARDHANDLE** hCard	(≠ NULL)
P2	**IN BYTE** bCardType	ICC_3_WIRE
P3	**IN BYTE** bSubCardType	NO_SUB_CARD_TYPE
P4	**IN DWORD** dwAddressToRead	**00h** … 7Fh
P5	**IN DWORD** dwLengthToRead	1 to **128**
P6	**OUT LPBYTE** pbRecvBuffer	(≠ NULL) → the P7 * 8 bits of protection memory (P7=P5 expected)
P7	**IN OUT LPDWORD** pdRecvBufferLength	(≠ NULL) IN → buffer size ≥ P5 OUT → length received (P5 data expected)

GSCWriteSecurityMemory (SLE4428 only)

Error Counter, PSC1 and PSC2 are written to the logical addresses 00h, 01h and 02h of the Security Memory. In fact, they are physically written at addresses FDh, FEh and FFh of the Main Memory.
Caution : Avoid writing in the Error Counter and in particular clearing its bits which authorize the 8 presentations of the secret codes before locking the card.

	Parameter	Value
P1	**IN SCARDHANDLE** hCard	(≠ NULL)
P2	**IN BYTE** bCardType	ICC_3_WIRE
P3	**IN BYTE** bSubCardType	NO_SUB_CARD_TYPE
P4	**IN DWORD** dwAddressToWrite	00h Error Counter 01h PSC1 02h PSC2
P5	**IN LPBYTE** pbByteToWrite	(≠ NULL) → 1 to 3 bytes : [ERROR_COUNTER] [PSC1] [PSC2]
P6	**IN DWORD** dwLengthToWrite	1 to 3 (with P4+P6 ≤ 3)

GSCReadSecurityMemory (SLE4428 only)

The ERROR_COUNTER is always readable. PSC1 and PSC2 are replaced with 0 if the 2-byte PSC has not been presented.
The number of the Error Counter bits at level 1 determines the number of possible attempts to present the secret codes. All the 8 bits are significant.
If the Error Counter returns 0 the card is definitively write locked.

The 3 bytes of the Security Memory are read in fact at the addresses 3FDh, 3FEh and 3FFh of the Main Memory of the card.

	Parameter	**Value**
P1	**IN SCARDHANDLE** hCard	(≠ NULL)
P2	**IN BYTE** bCardType	ICC_3_WIRE
P3	**IN BYTE** bSubCardType	NO_SUB_CARD_TYPE
P4	**IN DWORD** dwAddressToRead	**00h** Error Counter 01h PSC1 02h PSC2
P5	**IN DWORD** dwLengthToRead	1 to **3** (with P4+P5 ≤ 3)
P6	**OUT LPBYTE** pbRecvBuffer	(≠ NULL) → 1 to 3 bytes expected : **[ERROR_COUNTER] [PSC1] [PSC2]**
P7	**IN OUT LPDWORD** pdRecvBufferLength	(≠ NULL) IN → buffer size ≥ P5 OUT → length received (P5 data expected)

GSCVerify (SLE4428 only)

This function performs all the verification procedure: write one bit in Error Counter, verify 1st PSC byte, verify 2nd PSC byte, erase the counter error and read the Error Counter to check if verify is successful.

	Parameter	**Value**
P1	**IN SCARDHANDLE** hCard	(≠ NULL)
P2	**IN BYTE** bCardType	ICC_3_WIRE
P3	**IN DWORD** dwCodeLength	**2**
P4	**IN LPBYTE** pbCodeToVerify	(≠ NULL) → 2-byte PSC to verify

Protocol SDA : GFM2k … 32k

GSCPowerUp

	Parameter	Value
P1	**IN SCARDHANDLE** hCard	(≠ NULL)
P2	**IN OUT LPBYTE** pbRecvBuffer	NULL
P3	**IN OUT LPDWORD** pcbRecvLength	NULL

GSCWrite

	Parameter	Value
P1	**IN SCARDHANDLE** hCard	(≠ NULL)
P2	**IN BYTE** bCardType	ICC_SDA
P3	**IN BYTE** bSubCardType	PAGE_8_BYTES (for GFM 2k) PAGE_16_BYTES (for GFM4k…16k) PAGE_32_BYTES (for GFM32k)
P4	**IN DWORD** dwAddressToWrite	0000h … 0FFFh (for GFM32k)
P5	**IN LPBYTE** pbByteToWrite	(≠ NULL) → the P6 data
P6	**IN DWORD** dwLengthToWrite	1 to 256 (with P4+P6 ≤ 1000h for GFM32k)

GSCRead

	Parameter	Value
P1	**IN SCARDHANDLE** hCard	(≠ NULL)
P2	**IN BYTE** bCardType	ICC_SDA
P3	**IN BYTE** bSubCardType	PAGE_8_BYTES (for GFM 2k) PAGE_16_BYTES (for GFM4k…16k) PAGE_32_BYTES (for GFM32k)
P4	**IN DWORD** dwAddressToRead	0000h … 0FFFh (for GFM32k)
P5	**IN DWORD** dwLengthToRead	1 to 256 (with P4+P5 ≤ 1000h for GFM32k)
P6	**OUT LPBYTE** pbRecvBuffer	(≠ NULL) → the P7 data received (P5 data expected)
P7	**IN OUT LPDWORD** pdRecvBufferLength	(≠ NULL) IN → buffer size ≥ P5 OUT → length received (P5 data expected)

Protocol 4403 : GPM103

GSCPowerUp

	Parameter	Value
P1	**IN SCARDHANDLE** hCard	(≠ NULL)
P2	**IN OUT LPBYTE** pbRecvBuffer	NULL
P3	**IN OUT LPDWORD** pcbRecvLength	NULL

GSCRead

	Parameter	Value
P1	**IN SCARDHANDLE** hCard	(≠ NULL)
P2	**IN BYTE** bCardType	ICC_4403
P3	**IN BYTE** bSubCardType	NO_SUB_CARD_TYPE
P4	**IN DWORD** dwAddressToRead	00h … 0Ch
P5	**IN DWORD** dwLengthToRead	1 to 13 (with P4+P5 ≤ 13)
P6	**OUT LPBYTE** pbRecvBuffer	(≠ NULL) → the P7 data received (P5 data expected)
P7	**IN OUT LPDWORD** pdRecvBufferLength	(≠ NULL) IN → buffer size ≥ P5 OUT → length received (P5 data expected)

Protocol 4433 : GAM226/326

GSCPowerUp

	Parameter	Value
P1	**IN SCARDHANDLE** hCard	(≠ NULL)
P2	**IN OUT LPBYTE** pbRecvBuffer	NULL
P3	**IN OUT LPDWORD** pcbRecvLength	NULL

GSCRead

	Parameter	Value
P1	**IN SCARDHANDLE** hCard	(≠ NULL)
P2	**IN BYTE** bCardType	ICC_4433
P3	**IN BYTE** bSubCardType	NO_SUB_CARD_TYPE
P4	**IN DWORD** dwAddressToRead	00h … 2Fh
P5	**IN DWORD** dwLengthToRead	1 to 48 (with P4+P5 ≤ 48)
P6	**OUT LPBYTE** pbRecvBuffer	(≠ NULL) → the P7 data received (P5 data expected)
P7	**IN OUT LPDWORD** pdRecvBufferLength	(≠ NULL) IN → buffer size ≥ P5 OUT → length received (P5 data expected)

ANNEXE 1: Extracts of GSC_DEF.H

```
// For bCardType
#define  ICC_2_WIRE                    0x01
#define  ICC_3_WIRE                    0x02
#define  ICC_SDA                       0x03
#define  ICC_4403                      0x05
#define  ICC_4433                      0x06

// For bSubCardType
#define ADDR_1_BYTE                    0x01
#define ADDR_2_BYTES                   0x02
#define ADDR_3_BYTES                   0x03
#define PAGE_8_BYTES                   0x04
#define PAGE_16_BYTES                  0x05
#define PAGE_32_BYTES                  0x06
#define PAGE_64_BYTES                  0x07
#define PAGE_128_BYTES                 0x08
#define NO_SUB_CARD_TYPE               0x09

//// Error code section

#define  GEM_SUCCESS                                0x00000000
#define  GEM_E_INVALID_HANDLE                       0x00000001
#define  GEM_E_NO_SERVICE                           0x00000002
#define  GEM_E_INVALID_PARAMETERS                   0x00000003
#define  GEM_E_NO_READERS_AVAILABLE                 0x00000004
#define  GEM_E_NO_SMARTCARD                         0x00000005
#define  GEM_E_CARD_UNSUPPORTED                     0x00000006
#define  GEM_E_UNKNOWN_CARD                         0x00000007
#define  GEM_E_INSUFFICIENT_BUFFER                  0x00000008
#define  GEM_E_PROTOCOL_NOT_SUPPORTED               0x00000009
#define  GEM_E_WRITE_ERROR                          0x0000000A
#define  GEM_E_INVALID_ADDRESS                      0x0000000B
#define  GEM_E_INVALID_LENGTH                       0x0000000C
#define  GEM_E_NO_ACCESS                            0x0000000D
#define  GEM_E_WRONG_PWD                            0x0000000E
#define  GEM_E_UNKNOWN_ERROR                        0x0000000F
#define  GEM_E_INVALID_FIRMWARE_ANSWER              0x00000010
#define  GEM_E_EARLY_END_OF_FILE                    0x00000011
```

ANNEXE 2: Examples

Function	This function introduces the use of
TestInit()	GCSInit
TestClose()	GCSIose
TestPowerUp_WithType1ATR()	GSCPowerUp for the SLE4418/28 & 4432/42
TestPowerUp_WithoutATR()	GSCPowerUp for the other synchronous cards
TestPowerDown()	GSCPowerDown
TestReadSLE4418()	GSCRead for the SLE4418/28
TestWriteSLE4418()	GSCWrite for the SLE4418/28
TestVerifySLE4418()	GSCVerify for the SLE4418/28

```
DWORD                   lRet;
DWORD                   m_Ret;
SCARDCONTEXT            m_hContext;
SCARDHANDLE             m_hCard;
Cstring             m_csSATR;

*****************************************************************************
void TestInit()
*****************************************************************************
{
if(m_hContext)
{
if(m_hCard)
lRet = SCardDisconnect(m_hCard, SCARD_UNPOWER_CARD);

        lRet = SCardReleaseContext(m_hContext);
        m_hContext = NULL;
        m_hCard    = NULL;
}
lRet = SCardEstablishContext(SCARD_SCOPE_SYSTEM, NULL, NULL, &m_hContext);
if (lRet != SCARD_S_SUCCESS)
{
        // Display ("Error! No RM context!");
        m_Ret = GEM_E_NO_SERVICE;
        return;
}
// Get size of reader list
DWORD           dwReadersLength = 0;
LPSTR           szListReaders = NULL;
LPTSTR          szSCReader    = NULL;
lRet = SCardListReaders(m_hContext, NULL, NULL, &dwReadersLength);
if (lRet != SCARD_S_SUCCESS)
{
        // Display ("No readers present!");
        lRet = SCardReleaseContext(m_hContext);
        m_Ret = GEM_E_NO_READERS_AVAILABLE;
        return;
}
szListReaders = (char*)malloc(dwReadersLength);
lRet = SCardListReaders(m_hContext, NULL, (LPTSTR)szListReaders, &dwReadersLength);
if (lRet != SCARD_S_SUCCESS)
{
if(szListReaders) free(szListReaders);
lRet = SCardReleaseContext(m_hContext);
        // Display ("No readers present!");
        m_Ret = GEM_E_NO_READERS_AVAILABLE;
```

```
            return;
}
szSCReader = (LPTSTR)szListReaders;
if(!szSCReader || *(szSCReader) == '\0')
{
if(szListReaders) free(szListReaders);
            lRet = SCardReleaseContext(m_hContext);
            // Display ("No readers present!");
            m_Ret = GEM_E_NO_READERS_AVAILABLE;
            return;
}

// Connect to the card :
DWORD           dwShareMode = SCARD_SHARE_DIRECT;
DWORD           dwPreferedProtocol = SCARD_PROTOCOL_UNDEFINED;
DWORD           dwProtocol;

lRet = SCardConnect(m_hContext, (LPCTSTR)szSCReader, dwShareMode,
      dwPreferedProtocol, &m_hCard, &dwProtocol);

if (lRet != SCARD_S_SUCCESS)
{
lRet = SCardReleaseContext(m_hContext);
            if(szListReaders) free(szListReaders);
            m_hCard = NULL;
            // Display ("Failed SHARE_DIRECT!");
            m_Ret = GEM_E_NO_READERS_AVAILABLE;
            return;
}

if(szListReaders) free(szListReaders);

// Init the library and select 5 Volt synchronous cards mode :

m_Ret = GCSInit(m_hCard);

if(m_Ret==GEM_SUCCESS)
// Display ("Smartcard subsystem initialized!");
else
// Display ("Error initializing smartcard subsystem!");
return;
}

*****************************************************************************
Void TestClose()
*****************************************************************************
{
if(m_hCard)
{
            m_Ret = GCSClose(m_hCard);
            SCardDisconnect(m_hCard,SCARD_UNPOWER_CARD);
            m_hCard = NULL;
}
if(m_hContext)
{
            SCardReleaseContext(m_hContext);
            m_hContext = NULL;
}
            // Display ("Smartcard connection closed!");
}
```

```
*********************************************************************************
void TestPowerUp_WithoutATR()
*********************************************************************************
{
if(!m_hCard)
{
// Display ("Smartcard is not initialized!");
        m_Ret = GEM_E_INVALID_PARAMETERS;
        return;
}
        m_Ret = GSCPowerUp(m_hCard, NULL, NULL);

        PUCHAR pOutBuffer = (PUCHAR)m_csSATR.GetBufferSetLength(0);
        m_csSATR.ReleaseBuffer( );

        if(m_Ret!=GEM_SUCCESS)
// Display ("SC subsystem error!");
        else
                // Display ("Success!");
}

*********************************************************************************
void TestPowerUp_WithType1ATR()      // for the SLE4418/28 & 4432/42
*********************************************************************************
{
if(!m_hCard)
{
// Display ("Smartcard is not initialized!");
        m_Ret = GEM_E_INVALID_PARAMETERS;
        return;
}

PUCHAR pOutBuffer = (PUCHAR)m_csSATR.GetBufferSetLength(10); //  Length ≥ ATR size
memset(pInBuffer, 0x00, 10);
DWORD  Length = 4;  // expected ATR length in bytes

m_Ret = GSCPowerUp(m_hCard, pOutBuffer, &Length);

// Reset buffer length...
m_csSATR.ReleaseBuffer( );
pOutBuffer = (PUCHAR)m_csSATR.GetBufferSetLength(Length);

// The conversion of the sting to HEX should be done before releasing buffer,
// Otherwise the reported string length can be not correct in case of zero byte in buffer
ConvertStringToHex(m_csSATR);
m_csSATR.ReleaseBuffer( );

if(m_Ret!=GEM_SUCCESS)
        // Display ("SC subsystem error!");
else
        // Display ("Success!");
}

*********************************************************************************
void TestPowerDown()
*********************************************************************************
{
if(!m_hCard)
```

```
{
        // Display ("Smartcard is not initialized!");
        m_Ret = GEM_E_INVALID_PARAMETERS;
        return;
}

m_Ret = GSCPowerDown(m_hCard);

PUCHAR pOutBuffer = (PUCHAR)m_csSATR.GetBufferSetLength(0);
m_csSATR.ReleaseBuffer( );

if (m_Ret!=GEM_SUCCESS)
// Display ("SC subsystem error!");
else
        // Display ("Success!");
}

*****************************************************************************
void TestReadSLE4418()  // case ICC_3_WIRE
*****************************************************************************
// DWORD m_OpLength  is the number of bytes to read.
// DWORD m_OpAddress is the address of the 1st byte to read.

{
if(!m_hCard)
{
        // Display ("Smartcard is not initialized!");
        m_Ret = GEM_E_INVALID_PARAMETERS;
        return;
}
if (m_OpLength == 0)
{
        // Display ("Invalid operation length!");
        m_Ret = GEM_E_INVALID_PARAMETERS;
        return;
}

PUCHAR pOutBuffer = (PUCHAR)m_csOUT.GetBufferSetLength(m_OpLength);
memset(pOutBuffer,0x00,m_OpLength);

DWORD m_iOUTLength = 0x1000;   // Max buffer size in the reader

m_Ret = GSCRead(m_hCard, ICC_3_WIRE, NO_SUB_CARD_TYPE, m_OpAddress,
m_OpLength, pOutBuffer, &m_iOUTLength);

if(m_Ret!=GEM_SUCCESS)
{
        // Display ("SC subsystem error!");
        m_csOUT.ReleaseBuffer( );
        pOutBuffer = (PUCHAR)m_csOUT.GetBufferSetLength(0);
        return;
}
else
        // Display ("Success!");

if(m_iOUTLength<m_OpLength)
{
        // Reset buffer length...
        m_csOUT.ReleaseBuffer( );
        pOutBuffer = (PUCHAR)m_csOUT.GetBufferSetLength(m_iOUTLength);
```

```
}
// Convert string before releasing buffer!
ConvertStringToHex(m_csOUT);
m_csOUT.ReleaseBuffer( ); // This will fix buffer length
}

*****************************************************************************
void TestWriteSLE4418()  // case ICC_3_WIRE
*****************************************************************************
// DWORD m_OpLength  is the number of bytes to write.
// DWORD m_OpAddress is the address of the 1st byte to write.
// UCHAR pInBuffer[256] is the buffer of the data to write
{
if(!m_hCard)
{
        // Display ("Smartcard is not initialized!");
        m_Ret = GEM_E_INVALID_PARAMETERS;
        return;
}
if (m_OpLength == 0)
{
        // Display ("Invalid length! Nothing to write!");
        m_Ret = GEM_E_INVALID_PARAMETERS;
        return;
}
m_Ret =GSCWrite(hCard, ICC_3_WIRE , NO_SUB_CARD_TYPE, m_OpAddress,
    pInBuffer, m_OpLength);

if(m_Ret!=GEM_SUCCESS)
        // Display ("SC subsystem error!");
else
        // Display ("Success!");
}

*****************************************************************************
void TestVerifySLE4418()  // case ICC_3_WIRE
*****************************************************************************
// DWORD m_OpLength = 2  is the number of code bytes for the verify operation.
// UCHAR pInBuffer[2] is the buffer with the 2 code bytes

{
if(!m_hCard)
{
        // Display ("Smartcard is not initialized!");
        m_Ret = GEM_E_INVALID_PARAMETERS;
        return;
}
ret = GSCVerify(hCard, ICC_3_WIRE, m_OpLength, pInBuffer);

if(m_Ret!=GEM_SUCCESS)
// Display ("SC subsystem error!");
else
        // Display ("Success!");
}
```