

SpringSmart™
Smartcard Reader
User Manual
for SattliteForms™
Version 1.2

Silone Magcard Inc.
May 2001

Visit <http://www.magcard.com>

SpringSmart™ Smartcard User Manual

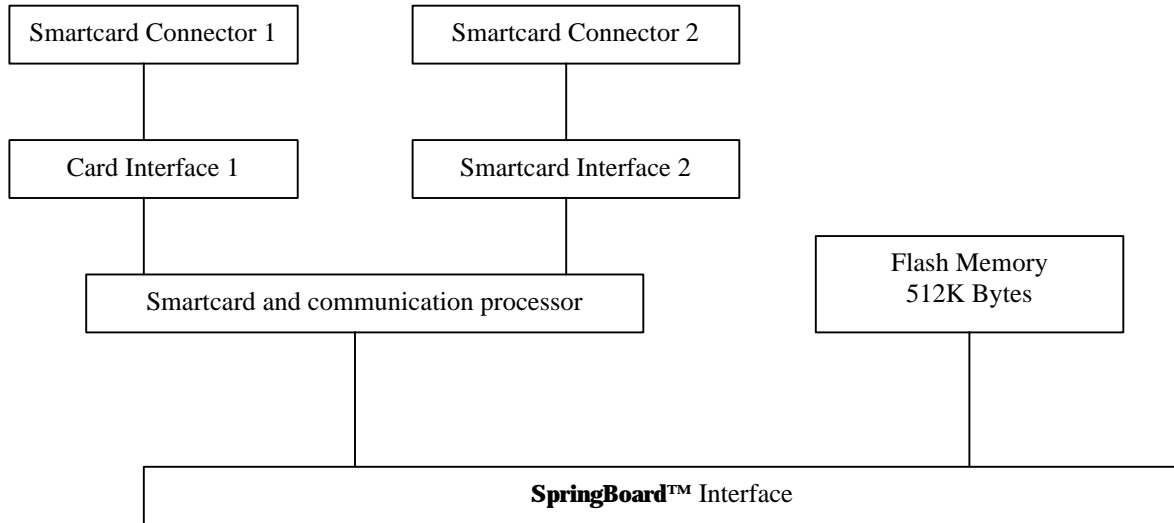
Revision History

Revision	Issue Date	Comments
1.0 Draft 1	May 22, 2001	Original version
1.0 Draft 2	Aug 20, 2001	CPU card APIs added
1.0 Draft 3	Oct 17, 2001	Satellite Forms APIs added
1.2 Draft 4	Dec 05, 2001	SLE4428 SLE4442 APIs added

This document contains material that is confidential to Magcard. Reproduction without the express written consent of Magcard is prohibited. All reasonable attempts were made to ensure the contents of this document are accurate, however no liability, expressed or implied is guaranteed. Magcard reserves the right to modify this document, without notification, at any time.

SpringSmart™ Module Description

SpringSmart™ smartcard reader is **Silone Magcard**'s newest modal of smartcard reader with a **SpringBoard™** interface. This reader can be plug in to a handheld with the **SpringBoard™** interface and turn the handheld into a mobile smartcard terminal. Following is the block diagram of the **SpringSmart™** smart card reader.



From the figure we can see that there are two smartcard interfaces in the reader. The outside one is interface1 and the little SAM card interface inside the reader is interface2. The Smartcard and communication processor is a microcontroller that communicates with the processor inside the **Visor™** handheld using the **SpringBoard™** interface. That microcontroller receives command from the **Visor™** processor and applies certain operation onto the smartcard and returns the result back to the **Visor™** processor. There is a 512K byte flash memory inside the reader; this memory has nothing to do with the Smartcard or the Smartcard/Communication processor. This flash memory is used to store program code and/or program database. The program code stored in the flash memory is to be running on the **Visor™**'s processor. The **SpringBoard™** bus supports plug and play, that is, when the module is plugged into the handheld, the OS in the handheld will try to find out if there is valid ROM memory in the module, if this is true, the OS will try to load and run the program in the ROM. This way a module can integrate all the hardware and software needed by the application into the module. The user does not have to download software into the handheld before using it. There is a utility program can be used to write the application software ROM image into the flash memory.

SpringSmart™ Smartcard Reader Development Guide for Satellite Forms™

Satellite Forms™ is a visual software development environment that makes it simple to create custom applications for Palm Computing Platform-compatible handheld devices such as the Palm OS™ software-compatible handheld devices manufactured by Palm Computing®, Symbol Technologies, IBM, TRG, and Handspring. For more information about **Satellite Forms™**, please visit their website <http://www.pumatech.com>.

We provide a Magcard proprietary extension, which is called **MC2000™** SFX plug-in, to extend the capabilities of **Satellite Forms™** to support our **SpringSmart™** smartcard reader.

Installation

Hardware: Just align the reader to the **SpingBoard™** slot on the back of the handheld and slowly push the reader module into the slot until it cannot move. On successful you will see the welcome application running and some animation is displayed on the screen, the welcome application will stop in a few seconds. If you want to uninstall the module, just pull it out. You don't have to power off the handheld before installing of uninstalling the reader module because the handheld support hot plug and play.

Software: The **MC2000™** extension is located in SDK CD at:

{CD-ROM}:\Satellite Forms\Extension\Magcard

Please copy the whole directory of \Magcard to a subdirectory of your Satellite Forms Extensions directory. By default, the locations of the Extensions directory is:

{Satellite Forms}\Extensions

When the Satellite Forms App Designer is started, it will search all the subdirectories of the Extensions directory for all extensions that you can load into Satellite Forms. Our proprietary extension is called MC2000.

MC2000™ Extension Requirement

Because the fundamental library of MC2000 Extension is written in C, so some basic knowledge of C programming is required for **Satellite Forms™** developers. These basic knowledge of C language covers:

- The definition and usage of Struct
- The usage of Dynamic Memory
- The variable definition of Char, Signed/Unsigned Integer, String and etc.
- The definition of Pointer variable

MC2000™ Extension API Brief

MC2000™ Extension APIs are divided into several categories:

Dynamic Memory Management Category:

These APIs are used to do the dynamic memory related manipulation. All of them are started with “Memory” string.

APIs Include:

- MemoryAllocate(Size)
- MemoryCompare(Memory1, Memory2, Count)
- MemoryCopy(Memory1, Memory2, Count)
- MemoryFree(Memory)
- MemoryGetByte(Memory, Offset)
- MemoryGetString(Memory, Offset, Count)
- MemoryGetSignedInteger(Memory, Offset, Size)
- MemoryGetUnsignedInteger(Memory, Offset, Size)
- MemoryReallocate(Memory, Size)
- MemoryReverse(Memory, Size)
- MemorySearch(Memory1, Size1, Memory2, Size2)
- MemorySet(Memory, Byte, Count)
- MemorySetByte(Memory, Offset, Byte)
- MemorySetInteger(Memory, Offset, Size, Value)
- MemorySetString(Memory, Offset, String)

Reader Management Category:

These APIs are used to control **SpringSmart™** Smartcard Reader’s electricity behavior.

APIs Include:

- OpenReader()
- CloseReader()
- GetReaderVersion(int *ReaderVersion)
- SelectInterface1()
- SelectInterface2()
- SetupCPUCard()
- SetupMemoryCard()
- Setup5V()
- Setup3V()
- PowerOnInterface1()
- PowerOffInterface1()
- PowerOnInterface2()
- PowerOffInterface2()
- InterfaceModeHigh()
- InterfaceModeLow()
- ActivateInterface()
- DeactivateInterface()
- CardPresent()

Memory Card Category:

This category’s API only works for Memory Type smart card.

APIs Include:

- GetMemoryCardATR(char *ATRBuffer)

ATMEL AT24C32/64 Memory Card Category:

AT24C32/64 smart card related manipulation functions.

APIs Include:

- ReadAT24C64(int Address, int NOB, char *Data)
- WriteAT24C64Byte(int Address, char Data)
- WriteAT24C64Page(int Address, int NOB, char *Data)

ATMEL AT88SC1608 Secure Memory Card Category:

AT88SC1608 smart card related manipulation functions.

APIs Include:

- Select1608UserZone(int Address)
- Read1608UserZone(int ByteAddr, int NOB, char *Data)
- Write1608UserZone(int ByteAddr, int NOB, char *Data)
- Read1608Configuration(int ByteAddr, int NOB, char *Data)
- Write1608Configuration(int ByteAddr, int NOB, char *Data)
- Read1608Fuses(char *Fuses)
- Write1608Fuses()
- Verify1608Password(int ReadWrite, int SetNumber, char *Password)
- Init1608Authentication(char *Q0)
- Verify1608Authentication(char *Q1)
- ComputeChallenge(char *Q0, char *GC, char *Ci, char *Q1, char *Q2)

Infineon SLE4428 Secure Memory Card Category:

SLE4428 smart card related manipulation functions.

APIs Include:

- Read4428WithPB(int StartPos, int NOB, char *Bfr, char *PB_Bfr)
- Read4428NoPB(int StartPos, int NOB, char *Bfr)
- Write4428(int StartPos, char DestByte, char PBSetFlag)
- Verify4428PSC(char *PSC)
- Read4428SM(char *SM_Bfr, char *SM_PB_Bfr)

Infineon SLE4442 Secure Memory Card Category:

SLE4442 smart card related manipulation functions.

APIs Include:

- Read4442WithPB(int StartPos, int NOB, char *Bfr, char *PB_Bfr)
- Read4442NoPB(int StartPos, int NOB, char *Bfr)
- Read4442PB(char *PB_Bfr)
- Write4442PB(int Address)
- Write4442(int StartPos, char DestByte, char PBSetFlag)
- Verify4442PSC(char *PSC)
- Read4442SM(char *SM_Bfr, char *SM_PB_Bfr)
- Write4442SM(int SMAddress, char SMByte)

CPU Card Category:

This category's API only works for Processor Type smart card.

APIs Include:

- `ResetCPUCard()`
- `GetCommandAPDUPointer(int MaxLength_Lc + MaxLength_Le)`
- `GetResponseAPDUPointer(CommandAPDUStructPtr CApduPtr)`
- `SetCommandAPDUPointer(CommandAPDUStructPtr CApduPtr, unsigned char CLA, unsigned char INS, unsigned char P1, unsigned char P2, unsigned char Lc, unsigned char Le)`
- `GetCommandAPDUPointer(unsigned char CLA, unsigned char INS, unsigned char P1, unsigned char P2, unsigned char Lc, unsigned char Le)`
- `ExchangeAPDU(CommandAPDUStructPtr CApduPtr, ResponseAPDUStructPtr RApduPtr)`
- `GetResponseAPDUMember(ResponseAPDUStructPtr RApduPtr, int MemberType)`
- `ReleaseAPDUPointer(CommandAPDUStructPtr CApduPtr, ResponseAPDUStructPtr RApduPtr)`
- `GetCPUParamMember(struct CPUParamStruct *pCPUParam, int MemberType)`

Related Structs:

- **struct CommandAPDUStruct**

```
{
    unsigned char CLA;
    unsigned char INS;
    unsigned char P1;
    unsigned char P2;
    unsigned char Lc;
    unsigned char *Data;
    int Le; // Le: 0 no Le, 1-255 Le, >256 Le = 0
};
```

This structure is used with the high level APDU exchange function `ExchangeAPDU`.

- **struct ResponseAPDUStruct**

```
{
    unsigned char *Data;
    unsigned char SW1;
    unsigned char SW2;
    int Lr;
};
```

The result structure of the `ExchangeAPDU` function. `Lr` indicates the total number of bytes that is in the `Data` field.

- struct CPUParamStruct


```

      {
          int Convention;
          int Protocol;           // 0: T=0, 1: T=1
          int N;                 // TC1
          int WI;                // TC2 only when T=0 is specified
          int IFSC;              // TA3
          int IFSD;              // reader defined
          int CWI, BWI;          // TB3 when T=1 is used
          int ATRLength;
          char ATR[ 32 ];
      };
      
```

Return code of the APIs

All of the APIs should return one of the following codes.

```

// error code and return values
//
// communcation related
//
#define errSuccess                0
#define errCardPresent           1
#define errCardAbsent            0

#define errFatal                 500
#define errCommunicationWithReader 501
#define errResponseFromReader   502
#define errSession              503

//
// card access related
//
#define errGetCPUATR             101
#define errParseATR              102
#define errInvalidProtocol      103
#define errResponseTPDU         104
#define errLENOTACCEPTED        105
#define errIFSDResponse         106
#define errResynchronize        107
#define errTimeOutStartBit      108
#define errParityError           109
#define errStartBitStartLow     110
#define errAttemptsCounter      111
#define errCardAbortion         112

```

API Reference

Dynamic Memory Management Category:

MemoryAllocate(Size)

Function : Allocates dynamic memory.

Usage : Memory = MemoryAllocate(Size)

Reference: **Size** is the byte amount you want to allocate.

Memory is a long integer which is the pointer to the allocated dynamic memory

MemoryCompare(Memory1, Memory2, Count)

Function : Compares dynamic memory.

Usage : Equal = MemoryCompare(Memory1, Memory2, Count)

Reference: **Memory1** & **Memory2** are the long integer which are the pointers of the comparing memory.

Count is the size you want to compare.

If it is equal, API returns 1, others return 0.

MemoryCopy(Memory1, Memory2, Count)

Function : Copies dynamic memory.

Usage : Memory1 = MemoryCopy(Memory1, Memory2, Count)

Reference: **Memory1** is the destination of memory copy result.

Memory2 is the memory source of the data.

Count is the size you want to copy.

MemoryFree(Memory)

Function : Frees dynamic memory.

Usage : Memory = MemoryFree(Memory)

Reference: **Memory** is the pointers of dynamic memory allocated by MemoryAllocate().

Returns 0 for success.

MemoryGetByte(Memory, Offset)

Function : Gets a byte from dynamic memory.

Usage : Byte = MemoryGetByte(Memory, Offset)

Reference: **Memory** is the start address of dynamic memory.

Offset is the offset from the start address.

MemoryGetString(Memory, Offset, Count)

Function : Gets a string from dynamic memory.

Usage : String = MemoryGetString(Memory, Offset, Count)

Reference: **Memory** is the start address of dynamic memory.

Offset is the offset from the start address.

Count is the length of the string.

MemoryGetSignedInteger(Memory, Offset, Size)

Function : Gets a signed dynamic size integer from dynamic memory.

Usage : Integer = MemoryGetSignedInteger(Memory, Offset, Size)

Reference: **Memory** is the start address of dynamic memory.

Offset is the offset from the start address.

Count is the length of the signed integer. Count 4

-32768 **Integer** 32767

MemoryGetUnsignedInteger(Memory, Offset, Size)

Function : Gets a unsigned dynamic size integer from dynamic memory.
Usage : `Integer = MemoryGetUnsignedInteger(Memory, Offset, Size)`
Reference: **Memory** is the start address of dynamic memory.
Offset is the offset from the start address.
Count is the length of the signed integer. `Count 4`
`0 Integer 65535`

MemoryReallocate(Memory, Size)

Function : Resizes a block of dynamic memory.
Usage : `Memory = MemoryReallocate(Memory, Size)`
Reference: **Memory** is the pointers of dynamic memory allocated by `MemoryAllocate()`.
Size is the byte amount you want to reallocate.

MemoryReverse(Memory, Size)

Function : Reverses a block of dynamic memory.
Usage : `Memory = MemoryReverse(Memory, Size)`
Reference: **Memory** is the start address of dynamic memory
Size is the byte amount you want to reverse.

MemorySearch(Memory1, Size1, Memory2, Size2)

Function : Searches for first occurrence of `Memory2` in `Memory1`.
Usage : `Offset = MemorySearch(Memory1, Size1, Memory2, Size2)`
Reference: Returns -1 if not found.

MemorySet(Memory, Byte, Count)

Function : Sets a block of dynamic memory with default value.
Usage : `Memory = MemorySet(Memory, Byte, Count)`
Reference: **Memory** is the start address of dynamic memory.
Byte is the default value.
Count is the bytes amount you want to set.

MemorySetByte(Memory, Offset, Byte)

Function : Sets a byte of dynamic memory.
Usage : `Memory = MemorySetByte(Memory, Offset, Byte)`
Reference: **Memory** is the start address of dynamic memory.
Offset is the offset from the start address.
Byte is the value.

MemorySetInteger(Memory, Offset, Size, Value)

Function : Sets a dynamic size integer of dynamic memory.
Usage : `Memory = MemorySetInteger(Memory, Offset, Size, Value)`
Reference: **Memory** is the start address of dynamic memory.
Offset is the offset from the start address.
Size is the length of the integer. `Size 4`
Value is the integer. `-32768 Value 65535`

MemorySetString(Memory, Offset, String)

Function : Sets a dynamic size string of dynamic memory.
Usage : `Memory = MemorySetString(Memory, Offset, String)`
Reference: **Memory** is the start address of dynamic memory.
Offset is the offset from the start address.

String is what you want to set.

Reader Management Category:

OpenReader()

Function : Begin to Setup Reader.

Usage : Result = OpenReader()

Reference: **Result** is one of the pre-defined error code or return values.

CloseReader()

Function : Release the resource and set the Reader in Low Power Mode.

Usage : Result = CloseReader()

Reference: **Result** is one of the pre-defined error code or return values.

GetReaderVersion(int *ReaderVersion)

Function : Get Reader's firmware version.

Usage : Result = GetReaderVersion(int *ReaderVersion)

Reference: **ReadVersion**.is a pointer to 2-byte integer which is the firmware version.

Result is one of the pre-defined error code or return values.

SelectInterface1 ()

Function : Select Reader's Interface1 as working interface.

Usage : Result = SelectInterface1 ()

Reference: **Result** is one of the pre-defined error code or return values.

SelectInterface2 ()

Function : Select Reader's Interface2 as working interface.

Usage : Result = SelectInterface2 ()

Reference: **Result** is one of the pre-defined error code or return values.

SetupCPUCard()

Function : Set Reader to CPU Card working mode.

Usage : Result = SetupCPUCard()

Reference: **Result** is one of the pre-defined error code or return values.

SetupMemoryCard()

Function : Set Reader to Memory Card working mode.

Usage : Result = SetupMemoryCard()

Reference: **Result** is one of the pre-defined error code or return values.

Setup5V()

Function : Set card's working voltage to 5V mode.

Usage : Result = Setup5V()

Reference: **Result** is one of the pre-defined error code or return values.

Setup3V()

Function : Set card's working voltage to 5V mode.

Usage : Result = Setup3V()

Reference: **Result** is one of the pre-defined error code or return values.

PowerOnInterface1()

Function : Connect the power to the smartcard interface1 chip.

Usage : Result = PowerOnInterface1()

Reference: **Result** is one of the pre-defined error code or return values.

PowerOffInterface1()

Function : Cut the power to the smartcard interface1 chip.

Usage : Result = PowerOffInterface1()

Reference: **Result** is one of the pre-defined error code or return values.

PowerOnInterface2()

Function : Connect the power to the smartcard interface2 chip.

Usage : Result = PowerOnInterface2()

Reference: **Result** is one of the pre-defined error code or return values.

PowerOffInterface2()

Function : Cut the power to the smartcard interface2 chip.

Usage : Result = PowerOffInterface2()

Reference: **Result** is one of the pre-defined error code or return values.

InterfaceModeHigh()

Function : Set the selected Interface to High Power mode.

Usage : Result = InterfaceModeHigh()

Reference: **Result** is one of the pre-defined error code or return values.

InterfaceModeLow()

Function : Set the selected Interface to Low Power mode.

Usage : Result = InterfaceModeLow()

Reference: **Result** is one of the pre-defined error code or return values.

ActivateInterface()

Function : Activate the selected Interface.

Usage : Result = ActivateInterface()

Reference: **Result** is one of the pre-defined error code or return values.

DeactivateInterface()

Function : Deactivate the selected Interface.

Usage : Result = DeactivateInterface()

Reference: **Result** is one of the pre-defined error code or return values.

CardPresent()

Function : Check the status on card's presence of the selected interface.

Usage : Result = CardPresent()

Reference: **Result** 1 : Card present 0: No card

Memory Card Category:

GetMemoryCardATR(char *ATRBuffer)

Function : Check the status on card's presence of the selected interface.

Usage : Result = CardPresent()

Reference: **ATRBuffer** is the char pointer of the returned ATR string.

Result is one of the pre-defined error code or return values.

ATMEL AT24C32/64 Memory Card Category:

ReadAT24C64(int Address, int NOB, char *Data)

Function : Read data from AT24C/3264 card.

Usage : Result = ReadAT24C64(int Address, int NOB, char *Data)

Reference: **Address** is the start address of data you want to read from the card

NOB is the bytes amount you want to read.

Data is the char pointer of the returned data string.

Result is one of the pre-defined error code or return values.

WriteAT24C64Byte(int Address, char Data)

Function : Write data to AT24C32/64 card in byte mode.

Usage : Result = WriteAT24C64Byte(int Address, char Data)

Reference: **Address** is the start address of data string you want to write to the card

Data is the value you want to write

Result is one of the pre-defined error code or return values.

WriteAT24C64Page(int Address, int NOB, char *Data)

Function : Write data to AT24C32/64 card in page mode.

Usage : Result = WriteAT24C64Page(int Address, int NOB, char *Data)

Reference: **Address** is the start address of data you want to write to the card

NOB is the bytes amount you want to write.

Data is the char pointer of the data string you want to write

Result is one of the pre-defined error code or return values.

ATMEL AT88SC1608 Secure Memory Card Category:

Select1608UserZone(int ZoneAddress)

Function : Select the operated user zone of AT88SC1608 Card.

Usage : Result = Select1608UserZone(int ZoneAddress)

Reference: **ZoneAddress** is the user zone number of card. 0 **ZoneAddress** 7

Result is one of the pre-defined error code or return values.

Read1608UserZone(int ByteAddr, int NOB, char *Data)

Function : Read data from the selected user zone of AT88SC1608 Card.

Usage : Result = Read1608UserZone(int ByteAddr, int NOB, char *Data)

Reference: **ByteAddr** is the start address of data you want to read from the card.

NOB is the bytes amount you want to read.

Data is the char pointer of the returned data string.

Result is one of the pre-defined error code or return values.

Write1608UserZone(int ByteAddr, int NOB, char *Data)

Function : Write data to the selected user zone of AT88SC1608 Card.

Usage : Result = Write1608UserZone(int ByteAddr, int NOB, char *Data)

Reference: **ByteAddr** is the start address of data you want to write to the card.
NOB is the bytes amount you want to write.
Data is the char pointer of the data string you want to write.
Result is one of the pre-defined error code or return values.

Read1608Configuration(int ByteAddr, int NOB, char *Data)

Function : Read data form the configuration zone of AT88SC1608 card.
Usage : Result = Read1608Configuration(int ByteAddr, int NOB, char *Data)
Reference: **ByteAddr** is the start address of data you want to read from the configuration zone.
NOB is the bytes amount you want to read.
Data is the char pointer of the returned data string.
Result is one of the pre-defined error code or return values.

Write1608Configuration(int ByteAddr, int NOB, char *Data)

Function : Write data to the configuration zone of AT88SC1608 card.
Usage : Result = Write1608Configuration(int ByteAddr, int NOB, char *Data)
Reference: **ByteAddr** is the start address of data you want to write to the configuration zone.
NOB is the bytes amount you want to write.
Data is the char pointer of the data string you want to write.
Result is one of the pre-defined error code or return values.

Read1608Fuses(char *Fuses)

Function : Read the Fuse status of AT88SC1608 card.
Usage : Result = Read1608Fuses(char *Fuses)
Reference: **Fuses** is the pointer of char that stores the card's fuses status.
Result is one of the pre-defined error code or return values.

Write1608Fuses()

Function : Change the Fuse status of AT88SC1608 card.
Usage : Result = Write1608Fuses()
Reference: **Result** is one of the pre-defined error code or return values.

Verify1608Password(int ReadWrite, int SetNumber, char *Password)

Function : Verify the password of AT88SC1608 card.
Usage : Result = Verify1608Password(int ReadWrite, int SetNumber, char *Password)
Reference: **ReadWrite** is the flag indicate the read/write mode.
0: Write Mode Other Value: Read Mode
SetNumber is the password set number.
Password is the char pointer of the password string.
Result is one of the pre-defined error code or return values.

Init1608Authentication(char *Q0)

Function : Initialize the authentication process of AT88SC1608 card.
Usage : Result = Init1608Authentication(char *Q0)
Reference: **Q0** is the pointer of random number generated by card reader.
The Result is one of the pre-defined error code or return values.

Verify1608Authentication(char *Q1)

Function : Verify AT88SC1608 card Authentication Result.
Usage : Result = Verify1608Authentication(char *Q1)

Reference: **Q1** is the pointer of first result generated by card.
The Result is one of the pre-defined error code or return values.

ComputeChallenge(char *Q0, char *GC, char *Ci, char *Q1, char *Q2)

Function : Compute Challenges for AT88SC1608 Card Authentication Process.
Usage : Result = ComputeChallenge(char *Q0, char *GC, char *Ci, char *Q1, char *Q2)
Reference: **Q0** is the pointer of random number generated by card reader.
GC is the seed in authentication process.
Ci is the pointer of random number generated by card.
Q1 is the pointer of first result generated by card.
Q2 is the pointer of second result generated by card.
Result is one of the pre-defined error code or return values.

Infineon SLE4428 Secure Memory Card Category:

Read4428WithPB(int StartPos, int NOB, char *Bfr, char *PB_Bfr)

Function : Read data and protection bit from SLE4428 card..
Usage : Result = Read4428WithPB(int StartPos, int NOB, char *Bfr, char *PB_Bfr)
Reference: **StartPos** is the start address of data you want to read from the card.
NOB is the bytes amount you want to read.
Bfr is the char pointer of the returned data string.
PB_Bfr is the char pointer of the returned protection bit array.
Result is one of the pre-defined error code or return values.

Read4428NoPB(int StartPos, int NOB, char *Bfr)

Function : Read data only from SLE4428 card.
Usage : Result = Read4428NoPB(int StartPos, int NOB, char *Bfr)
Reference: **StartPos** is the start address of data you want to read from the card.
NOB is the bytes amount you want to read.
Bfr is the char pointer of the returned data string.
Result is one of the pre-defined error code or return values.

Write4428(int StartPos, char DestByte, char PBSetFlag)

Function : Write data to SLE4428 card.
Usage : Result = Write4428(int StartPos, char DestByte, char PBSetFlag)
Reference: **StartPos** is the start address of data you want to write to the card.
DestByte is the char data which is going to be written.
PBSetFlag PBSetFlag = 1 Kill the Protection Byte.
PBSetFlag = 0 Leave the Protection Byte.
Result is one of the pre-defined error code or return values.

Verify4428PSC(char *PSC)

Function : Verify SLE4428 card password code.
Usage : Result = Verify4428PSC(char *PSC)
Reference: **PSC** is the char pointer of the password string.
Result is one of the pre-defined error code or return values.

Read4428SM(char *SM_Bfr, char *SM_PB_Bfr)

Function : Read SLE4428 card security memory.
Usage : Result = Read4428SM(char *SM_Bfr, char *SM_PB_Bfr)
Reference: **SM_Bfr** is the char pointer of the returned data string.

SM_PB_Bfr is the char pointer of the returned protection bit array.
Result is one of the pre-defined error code or return values.

Infineon SLE4442 Secure Memory Card Category:

Read4442WithPB(int StartPos, int NOB, char *Bfr, char *PB_Bfr)

Function : Read data and protection bit from SLE4442 card..
Usage : Result = Read4442WithPB(int StartPos, int NOB, char *Bfr, char *PB_Bfr)
Reference: **StartPos** is the start address of data you want to read from the card.
NOB is the bytes amount you want to read.
Bfr is the char pointer of the returned data string.
PB_Bfr is the char pointer of the returned protection bit array.
Result is one of the pre-defined error code or return values.

Read4442NoPB(int StartPos, int NOB, char *Bfr)

Function : Read data only from SLE4442 card.
Usage : Result = Read4442NoPB(int StartPos, int NOB, char *Bfr)
Reference: **StartPos** is the start address of data you want to read from the card.
NOB is the bytes amount you want to read.
Bfr is the char pointer of the returned data string.
Result is one of the pre-defined error code or return values.

Read4442PB(char *PB_Bfr)

Function : Read protection bit from SLE4442 card.
Usage : Result = Read4442PB(char *PB_Bfr)
Reference: **PB_Bfr** is the char pointer of the returned 4-Byte long ProtectionByte data.
Result is one of the pre-defined error code or return values.

Write4442PB(int Address)

Function : Write protection bit to SLE4442 card.
Usage : Result = Write4442PB(int Address)
Reference: **Address** is the address of data that you want to protect. Address < 32.
Result is one of the pre-defined error code or return values.

Write4442(int StartPos, char DestByte, char PBSetFlag)

Function : Write data to SLE4442 card.
Usage : Result = Write4442(int StartPos, char DestByte, char PBSetFlag)
Reference: **StartPos** is the start address of data you want to write to the card.
DestByte is the char data which is going to be written.
PBSetFlag PBSetFlag = 1 Kill the Protection Byte.
PBSetFlag = 0 Leave the Protection Byte.
Result is one of the pre-defined error code or return values.

Verify4442PSC(char *PSC)

Function : Verify SLE4442 card password code.
Usage : Result = Verify4442PSC(char *PSC)
Reference: **PSC** is the char pointer of the password string.
Result is one of the pre-defined error code or return values.

Read4442SM(char *SM_Bfr, char *SM_PB_Bfr)

Function : Read SLE4442 card security memory.

Usage : Result = Read4442SM(char *SM_Bfr, char *SM_PB_Bfr)
Reference: **SM_Bfr** is the char pointer of the returned data string.
SM_PB_Bfr is the char pointer of the returned protection bit array.
Result is one of the pre-defined error code or return values.

Write4442SM(int SMAddress, char SMByte)

Function : Write SLE4442 card security memory.
Usage : Result = Write4442SM(int SMAddress, char SMByte)
Reference: **SMAddress** is the address of the Security Memory.
SMByte is the data that you want to write to Security Memory.
Result is one of the pre-defined error code or return values.

CPU Card Category:

ResetCPUCard()

Function : Reset CPU card and get the ATR string.
Usage : Result = ResetCPUCard()
Reference: **Result** is one of the pre-defined error code or return values.

GetCommandAPDUPointer(int MaxLength_Lc + MaxLength_Le)

Function : Get the pointer of Command APDU Struct.
Usage : CommandAPDUStructPtr CApduPtr =
GetCommandAPDUPointer(int MaxLength_Lc + MaxLength_Le)
Reference: **MaxLength_Lc** is the max length of Lc you want to use in your future APDU
Exchange.
MaxLength_Le is the max length of Le you want to use in your future APDU
Exchange.

GetResponseAPDUPointer(CommandAPDUStructPtr CApduPtr)

Function : Get the pointer of Response APDU Struct.
Usage : ResponseAPDUStructPtr RApduPtr =
GetResponseAPDUPointer(CommandAPDUStructPtr CApduPtr)
Reference: **CApduPtr** is the pointer of Command APDU Struct that you get from
GetCommandAPDUPointer() function.

SetCommandAPDUPointer(CommandAPDUStructPtr CApduPtr, unsigned char CLA, unsigned char INS, unsigned char P1, unsigned char P2, unsigned char Lc, unsigned char Le)

Function : Setup a Command APDU.
Usage : CommandAPDUStructPtr CApdu =
SetCommandAPDUPointer(CommandAPDUStructPtr CApduPtr,
unsigned char CLA,
unsigned char INS,
unsigned char P1,
unsigned char P2,

unsigned char Lc,
unsigned char Le)

Reference: **CApduPtr** is the pointer of Command APDU Struct that you get from
GetCommandAPDUPointer() function.

CLA is the Class value.

INS is the Instruction value.

P1 is the P1 value.

P2 is the P2 value.

Lc is the Lc value, input 0 if it is null.

Le is the Le value, input 0 if it is null.

GetCommandAPDUDataPointer(CommandAPDUStructPtr CApduPtr)

Function : Get the pointer of Command APDU Struct's Data member.

Usage : char *Data = GetCommandAPDUDataPointer(CommandAPDUStructPtr CApduPtr)

Reference: **CApduPtr** is the pointer of Command APDU Struct that you get from
GetCommandAPDUPointer() function.

ExchangeAPDU(CommandAPDUStructPtr CApduPtr, ResponseAPDUStructPtr RApduPtr)

Function : Exchange the APDU between the Reader and the Card.

Usage : Result = ExchangeAPDU(CommandAPDUStructPtr CApduPtr,
ResponseAPDUStructPtr RApduPtr)

Reference: **CApduPtr** is the pointer of Command APDU Struct that you get from
GetCommandAPDUPointer() function.

RApduPtr is the pointer of Response APDU Struct that you get from
GetResponseAPDUPointer() function.

Result is one of the pre-defined error code or return values.

GetResponseAPDUMember(ResponseAPDUStructPtr RApduPtr, int MemberType)

Function : Get the member of the Response APDU Struct.

Usage : MemberType Member =
GetResponseAPDUMember(ResponseAPDUStructPtr RApduPtr, int MemberType)

Reference: **RApduPtr** is the pointer of Response APDU Struct that you get from previous
GetResponseAPDUPointer() function.

MemberType indicates which member of Response APDU Struct you want to get.

0: Data 1: SW1 2: SW2 3: Lr

Member is varied by the different member requesting.

ReleaseAPDUPointer(CommandAPDUStructPtr CApduPtr, ResponseAPDUStructPtr RApduPtr)

Function : Release the resource used in APDU Exchange.

Usage : Memory MemPtr = ReleaseAPDUPointer(CommandAPDUStructPtr CApduPtr,
ResponseAPDUStructPtr RApduPtr)

Reference: **CApduPtr** is the pointer of Command APDU Struct that you get from
GetCommandAPDUPointer() function.

RApduPtr is the pointer of Response APDU Struct that you get from
GetResponseAPDUPointer() function.

GetCPUParamMember(int MemberType)

Function : Get the member of the CPU Parameters Struct.

Usage : MemberType Member = GetCPUParamMember(int MemberType)

Reference: **MemberType** indicates which member of CPUParamStruct Struct you want to get.

0: Convention 1: Protocol 2: N 3: WI 4: IFSC 5: IFSD

6: CWI 7: BWI 8: ATRLength 9: ATR

Member is varied by the different member requesting.

Specifications

Card Interfaces

Features 2 smartcard interface

Card types supported

Microcontroller cards with T=0 and T=1 protocol

AT24C32, AT24C64 I2C cards

AT88SC1608 security memory card

SLE4428 security memory card

Support of more card types coming

Card operating voltage supported

Both 3V and 5V card are supported

Host Interface

HandSpring's SpringBoard™ interface

Power supply

Operating voltage: 2.7V to 3.3V

Operation current: depending on cards, typical 20mA

Standby current: 300uA max

Memory size

Internal EEPROM: 128KByte to 4Mbyte, 512Kbyte in standard configuration

Contact Information

Silone Magcard Inc.
1440 Koll Circle #109
San Jose, CA 95112
USA
Voice: 1-408-4418858
Fax: 1-408-4418878

<http://www.magcard.com>

Email: techsupport@magcard.com

Please visit www.magcard.com for product and software updates