

SpringSmart Smartcard Reader User Manual Version 1.2

Silone Magcard Inc.
May 2001-November 2001

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

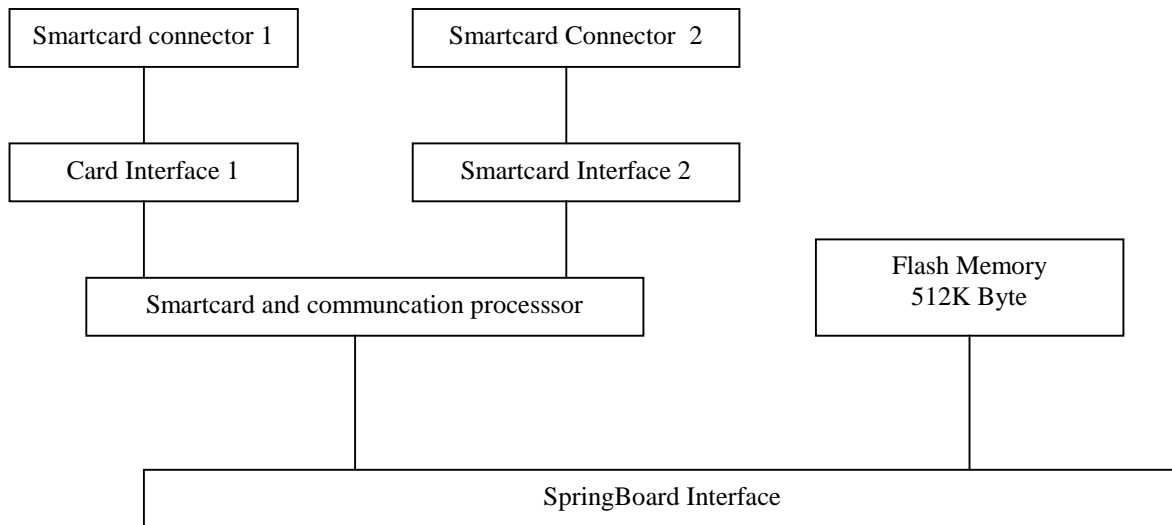
SpringSmart Smartcard User Manual Revision History

Revision	Issue Date	Comments
1.0 Draft 1	May 22, 2001	Original version
1.0 Draft 2	August 20, 2001	CPU card APIs added
1.2	November 1, 2001	SLE 4428 added SLE 4442 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.

SprintSmart 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 the is 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 and the Smartcard/Communication processor. This flash is used to store program code and/or program database. The program code stored in this 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.

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 can not moved. 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: This isn't installation program for the development kit. If you are going to design you own application, all you need are two files: mc2000.h and mc200.lib, and they are located in the Lib folder on the CD. Please setup your project and copy these two files into your project folder and add them into your project. Then you can make calls to the API to read/write the smartcard. If you want to run the sample program, just unzip the sample zips into your HD and build and run the project. There is also a document folder on the CD where you can find all the documents for SpringSmart. The folder makerom and the file flash.zip contains the programs needed to write you application code into the flash memory on the reader, this is explained in next few pages.

SpringSmart Smartcard Reader Development Guide

The suggested steps for developing applications using a SpringSmart Smartcard Reader

1. Choose a smartcard. The current version of SpringSmart smartcard reader, namely version 1.0, supports the follow card types:
 - AT24C64:** I2C type of memory card, memory size is 64K bits or 8K bytes. No security is supported by this card.
 - AT88SC1608:** Atmel's high security memory card. It's supports authentication process found only on microcontroller cards, and it's memory organization supports multiple applications on one card. The memory size is 16K bits or 2K bytes.
 - SLE4428:** Infineon's secure memory card. The memory size is 1K bytes. The memory is free to read, but the memory write operation is protected by a 2-byte password.
 - SLE4442:** Infineon's secure memory card. The memory size is 256 bytes. The memory is free to read, but the memory write operation is protected by a 3-byte password.

T=0 and T=1 microcontroller cards.
Support of more card types are planning to be added to the reader. You may check our web site at <http://www.magcard.com> to see if new cards are added or you may request us to add support of a specific card. Update of the firmware is free for the user of the development kit.
2. Purchase a CodeWarrior™ for PalmOS and install it.
3. Install the development kit and evaluate the demo program for the card. A demo project for each supported card type is provided in the development kit. The demo program will make use of all the functions associated with the card. The programmer can look into the source of the program and see how use the APIs.
4. Using CodeWarrior or GNU tools to development your own application. A .h file and a .lib file is provided in the development kit, just copy this two file into your project folder and add them into your project. Then you can make calls to the APIs to access the smartcard. At this stage the application is downloaded to the handheld by USB or serial interface and it runs in the handheld's RAM.
5. Prepare the welcome application to be program into the read's flash memory. A sample welcome application is provided in the development kit. You may modify this sample and add your special functions.
6. Use the utility program in the dev kit to make the ROM image to be programmed into the flash memory in the reader.
7. Now that the ROM image is ready, run the FlashUpdater project to program the ROM image into the flash memory on the reader. After this reset the handheld or reinsert

the reader, you will see the welcome application running. And your application is ready to be released.

Writing the application program into the Flash memory

When your application program is ready, you may want to put your application into the flash memory on the reader. This is important because this will give the end user a real plug and play experience. SpringSmart development kit provides the necessary utility program for programming the flash memory. The following are the steps to take to write your application into the flash.

1. Log on to http://www.handspring.com/developers/sw_dev.jhtml to download the PalmOS GNU tools and install the tools. We need the Palm-MakeROM.exe program to convert the .prc file into ROM image.
2. Copy the makerom folder in the CD into your HD. Unzip the flash.zip into HD.
3. Prepare your application to be burned into the flash. Normally you may want to write 2 application into the flash, one is the welcome application which will be running when the module is inserted into the handheld or when the handheld is reset. This welcome application will normally show some thing like module name, company name or module version on the screen. The other application is the application for the module. When the module is inserted, the PalmOS will show the icon of this application on the screen so you can start it by tapping on it.
4. Copy the two application into the makerom folder.
5. Start a command prompt and run the .bat file in the makerom folder with the two application's name (without extension name) as the parameter. The format is:
M2 name1 name2
The .bat will first invoke the palm-makerom program to convert the two .prc into one .bin file. This the binary ROM image to be flashed. Then a small program c.exe will convert the .bin file into a C source file, namely ss.c. This .c file is to be later compiled and linked into another project, which will be downloaded into the handheld and burned into the flash.
6. Copy the ss.c file into the folder where the "flash" project located and build and run the project on the handheld.
7. Press the Program Flash button and will until the flash is erased and updated. It's possible that on some handhelds you need to supply a 5V supply instead of the two cell 3V supply in order to burn the flash.
8. Delete the Flash application on your handheld and reset the handheld. You should see the new welcome application running. Then you can start your application.

API Reference

Return code of the functions

All of the function 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

Reader management functions

int OpenReader();

This function will setup the reader. **It MUST be called once before any other functions.**

int CloseReader();

This function will release any resource that the reader used and put the reader into low power mode. This function should be called before quitting the application program, in order to lower the power consumption by the reader.

int GetReaderVersion(int *ReaderVersion);

Call this one to get the firmware version of the reader. The return value is the error code indicating the success/failure of the function. The version number is save into the place pointer to by the parameter. For version 1.0 the number is 0x10.

int SelectInterface1();

int SelectInterface2();

These two functions will select which card interface following operations will be applied to. At any time only one interface can be operated. The other interface will just remain the state before it's deselected.

Int SetupCPUCard();

int SetupMemoryCard();

These two functions determine what kind of card is inserted in the current interface. If it a CPU (micro controller card), an asynchronous clock will be started to be later applied to the card. If it's a memory type card, the asynchronous clock will be stopped and it's ready to apply a sync clock to the card.

int Setup5V();

int Setup3V();

This two function will determine the working voltage of the card.

int PowerOnInterface1();

int PowerOffInterface1();

These two functions will connect/cut the power to the smartcard interface chip. The interface chip will not be functioning if it's power it cut, or it will recognize the control

signals applied to it if it's power is on. If you want to operate the card, power on the interface. If the program is to be stopped, power off the interface chip so the battery life can get longer.

```
int PowerOnInterface2( );  
int PowerOffInterface2( );
```

See previous.

```
int InterfaceModeHigh( );  
int InterfaceModeLow( );
```

This two function will put the interface to high/low power mode. In high mode, the interface chip can read/write the card, and the power consumption is high. In low mode, the interface can only check the card present state but it's not possible to read/write the card, but the power consumption can be much lower.

```
int ActivateInterface( );
```

This function will start the ISO standard activation sequence if the card is present. The power to the card is applied, and if it's a CPU card the asynchronous clock will be started and seen on the CLK pin of the card. If the card is absent, the activation sequence is not started and an error signal will be asserted. The error signal can be checked by CardPresent function. See the sample program pursecpu.

```
int DeactivateInterface( );
```

This function will start the ISO standard deactivate sequence. This power to the card will be cut off and the pin will be put to standby mode.

```
int CardPresent( ); // 1 card present, 0 no card, else error code
```

This function can be used to see if the card is already in the reader or not. If there is a card in the reader, it returns errCardPresent. If there is no card in the reader, the return value is errCardAbsent (=errSuccess, = 0). If there is any error occurred, the return value will be one of the error codes. The good practice is to always call this function before calling card read/write function. See the sample program pursecpu.

```
int GetMemoryCardATR( char *ATRBuffer );
```

This function only works for memory type card. It will reset the memory card, get the card's ATR bytes. **This function MUST be called one before reading/writing the memory card or the reading/writing will not work.**

API Reference

AT24C32/64 related functions

(a more generic I2C function set will be provided)

int Read_AT24C64(int Address, int NOB, char *Data);

This function reads NOB bytes of data starting from Address from the card. The result will be put into the place pointed to by Data. The return code will be one of the error codes.

int Write_AT24C64_Byte(int Address, char Data);

This function writes one byte to the card at Address. Check the return code to see if the write is done successfully.

int Write_AT24C64_Page(int Address, int NOB, char *Data);

This function writes an array of byte pointed by Data to the card start from Address. The number of bytes is indicated by NOB. This function uses the actual page write protocol of the card, please check the data sheet of the card to see if there is any limitations of the write.

API Reference

AT88SC1608 related functions

```
int Select_1608_User_Zone( int Address ); // ZoneAddr = 0 - 7  
int Read_1608_User_Zone( int ByteAddr, int NOB, char *Data );  
int Write_1608_User_Zone( int ByteAddr, int NOB, char *Data );  
int Read_1608_Configuration( int ByteAddr, int NOB, char *Data );  
int Write_1608_Configuration( int ByteAddr, int NOB, char *Data );  
int Read_1608_Fuses( char *Fuses );  
int Write_1608_Fuses( );  
int Verify_1608_Password( int ReadWrite, int SetNumber, char* Password );  
int Init_1608_Authentication( char *Q0 );  
int Verify_1608_Authentication( char *Q1 );  
int Compute_Challenge( char *Q0, char *GC, char *Ci, char *Q1, char *Q2 );
```

These functions are implemented exactly as described on the data sheet of the 1608 card. Please refer to the data sheet of the card and also the sample project 1608 for the usage of these functions.

API Reference

SLE4428 related functions

int Read_4428_With_PB(int StartPos, int NOB, char *Bfr, char *PB_Bfr);
// PB stands for protection bit, see 4428 datasheet for details

This function reads NOB bytes starting at address StartPos into Bfr, the protection bit associated with each byte is read into PB_Bfr.

int Read_4428_No_PB(int StartPos, int NOB, char *Bfr);

This function reads NOB bytes starting at address StartPos into Bfr.

int Write_4428(int StartPos, char DestByte, char PBSetFlag); // PBSetFlag == 1 set PB

This function writes one byte DestByte to address StartPos, if PBSetFlag == 1, the protection bit will be set, if PBSetFlag == 0, the protection bit will not be touched.

int Verify_4428_PSC(char *PSC);

This function verifies 4428 PSC (Password). Use the following function to read back the security memory (last 3 byte of the main memory) to see the result of the verification.

int Read_4428_SM(char *SM_Bfr, char *SM_PB_Bfr);

Read the Security Memory of the card. See 4428 datasheet for details.

These functions are implemented exactly as described in the data sheet of the 4428 card. Please refer to the data sheet of the card and also the sample project 4428 for the usage of these functions.

API Reference

SLE4442 related functions

int Read_4442_With_PB(int StartPos, int NOB, char *Bfr, char *PB_Bfr);

Use this function to read the main memory of 4442. The bytes that have a protecting bit is read into PB_Bfr.

int Read_4442_No_PB(int StartPos, int NOB, char *Bfr);

Use this function to read the main memory only.

int Read_4442_PB(char *PB_Bfr);

Use this function to read the protection bits. There are 32 protection bit in 4442, and they are put into 32-byte PB_Bfr.

int Write_4442_PB(int Address); // bit address !!! range : 0-31

Use this function to blow one protection bit. The address must be in the range 0-31.

int Write_4442(int Address, char DestByte, char PBSetFlag);

Use this function to write one byte to 4442 card. If PBSetFlag == 1 then the protection bit is set (protected) or it's not touched.

int Verify_4442_PSC(char *PSC);

Use this function to verify the card's 3-byte PSC (password). Use the following function Read_4442_SM to read the security memory of the card to see the result of the verification.

int Read_4442_SM(char *SM_Bfr);

Read the security memory of the card. The SM is 4 bytes, byte 0 is the error counter, byte 1-3 are the 3-byte password.

int Write_4442_SM(int SMAddress, char SMByte);

Use this function to change one byte in the 4-byte security memory. Address starts at 0.

API Reference

Microcontroller card related structures

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. Declare your variable of this type and fill it up, then call the ExchangeAPDU function with the address of this variable as the first parameter.

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 ];
};
```

See GetCPUParam function on the following page.

API Reference

Microcontroller card related functions

int ResetCPUCard();

This function will hard reset the micro controller card and then get the card's ATR (Answer To Reset) string. Then the ATR will be parsed and the operation parameters will be determined. The result operation parameter will be fill into a variable with the types CPUParamStruct, and the variable is allocated by this function. The address of the variable can be retrived by GetCPUParam function.

int ExchangeAPDU(struct CommandAPDUStruct *CAPDU, struct ResponseAPDUStruct *RAPDU);

After the CPU card is reset, this function can be called to exchange APDU with the card. If the APDU is successfully exchanged, the function will return errSuccess. Otherwise the function's return value indicates what kind of error was occurred.

int GetCPUParam(struct CPUParamStruct **CP);

Call this function to get a pointer to the Parameter Block of the CPU card. Normally it's not necessary to call this function, because the ExchangeAPDU will take care of all the parameters when access the CPU card.

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, SLE4442 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: 1.3 mA max

Memory size

Internal EEPROM: 128KByte to 4Mbyte, 512Mbyte 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