Card Manipulation Related APIs Reference

Any kinds of smart card could be used by MC998 handheld smart card reader. New card support can be added upon request.
Please send your request to our technical support man at techsupport@magcard.com .

1 Current Card Types Supported

Memory Card:
- ATMEL AT24C01
- ATMEL AT24C16
- ATMEL AT24C64
- ATMEL AT24C256
- ATMEL AT45D series

Secure Memory Card:
- ATMEL AT88SC102
- ATMEL AT88SC153
- ATMEL AT88SC1608
- INFINEON SLE4428
- INFINEON SLE4442

Microcontroller Card:
- ALL microcontroller cards with T=0 and T=1 protocols

2 General Steps for Memory & Secure Memory Card Manipulation

For the historical reasons, there're two different ways to do the MC998 card interface initiation and cleanup.

I The card manipulation procedure described as below is often used when user will only work on the main card interface:

```
Card_init(CARD_INIT_NUM_1);            // initialize card interface
                                       // the CARD_INIT_NUM_1 is kept for the
                                       // historical reason; no actual meaning
Card_ATR(typ_Card_ATR_param *);        // activates IC card and waits for the ATR
                                       // please refer to "API Manual" about the
                                       // parameters setting
Card Driver APIs for the specific card;
Card_deactivate(CARD_INIT_NUM_2);      // deactivate the whole card subsystem
                                       // the CARD_INIT_NUM_2 is kept for the
                                       // historical reason; no actual meaning
```

Sample code for card manipulation (Based on I^2C card)

```
{
    typ_Card_ATR_param stru_cardInfo;
    char atrBuffer[32];
```

```
        stru_cardInfo.card_num = 2;        // outside socket is #2
                                                   // inside    socket is #0
        stru_cardInfo.card_type_async = 0;     // 0 stands for synchronous card
                                                   // 1 stands for asynchronous card
        stru_cardInfo.ATR_buff = atrBuffer;
        Card_ATR(&stru_cardInfo);
        Read_IIC_Page32(0x00, dataSize, dataBuffer);
        Write_IIC_Page32(0x00, dataSize, data);
        Card_deactivate(CARD_INIT_NUM_2);
}
```

II The card manipulation procedure described as below is often used when user will work on both the main card interface and SAM interface. In this case, you can control the main card interface or SAM interface individually:

```
Card_power_on();                           // power the card i/f up but no card is
                                           // powered on
Card_select_sock()                         // select socket
Card_ATR(typ_Card_ATR_param *);   //power a certain socket on ( better fill in
                                           // the parameter so that it's the current on);
Card Driver APIs for the specific card;
Card_select_sock()                         // hop around
Card Driver APIs for the specific card;
…………….
Card_deact_curr_sock()                     // power down current selected socket
Card_deactivate()                          // deactivate the whole card subsystem
```

3. APIs reference for Memory & Secure Memory cards

** These functions are implemented exactly as described on the data sheet of the cards. It is recommended to read the datasheets carefully before using these functions.

3.1 Memory Cards

3.1.1 ATMEL AT24C01

int Read_AT24C01A(unsigned int StartPos, unsigned int NOB, unsigned char *Bfr);

Function : Read data from AT24C/3264 card.

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.

int Write_AT24C01A_Byte(unsigned int Address, unsigned char Data);

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

Reference: **Addresss** is the address of data you want to write to the card
Data is the value you want to write

int Write_AT24C01A_Page(unsigned int Address,unsigned int NOB,unsigned char *Data);

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

Reference : **Addresss** is the start address of data you want to write to the card
NOB is the bytes amount you want to write. 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.
Data is the char pointer of the data string you want to write

3.1.2 ATMEL AT24C16

int Read_AT24C16(unsigned int StartPos, unsigned int NOB, unsigned char *Bfr);

Function : Read data from AT24C/3264 card.

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.

int Write_AT24C16_Byte(unsigned int Address, unsigned char Data);

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

Reference: **Addresss** is the address of data you want to write to the card
Data is the value you want to write

int Write_AT24C16_Page(unsigned int Address, unsigned int NOB, unsigned char *Data);

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

Reference : **Addresss** is the start address of data you want to write to the card
NOB is the bytes amount you want to write. 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.
Data is the char pointer of the data string you want to write

3.1.3 ATMEL AT24C64

int Read_AT24C64(unsigned int StartPos, unsigned int NOB, unsigned char *Bfr);

Function : Read data from AT24C/3264 card.

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.

int Write_AT24C64_Byte(unsigned int Address, unsigned char Data);

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

Reference: **Addresss** is the address of data you want to write to the card
Data is the value you want to write

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

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

Reference : **Addresss** is the start address of data you want to write to the card
NOB is the bytes amount you want to write. 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.
Data is the char pointer of the data string you want to write

3.1.4 ATMEL AT24C256

int Read_AT24C256(unsigned int StartPos, unsigned int NOB, unsigned char *Bfr);

Function : Read data from AT24C/3264 card.

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.

int Write_AT24C256_Byte(unsigned int Address, unsigned char Data);

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

Reference: **Addresss** is the address of data you want to write to the card
Data is the value you want to write

int Write_AT24C256_Page(unsigned int Address, unsigned int NOB, unsigned char *Data);

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

Reference : **Addresss** is the start address of data you want to write to the card
NOB is the bytes amount you want to write. 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.
Data is the char pointer of the data string you want to write

3.1.5 ATMEL AT45D series

int Read_Main_AT45D(int Page, int ByteAddress, int NOB, char *Data);

Function : Main memory page read.

Reference : **Page** is the specified page address in main memory 0 ≤ Page < 2048
ByteAddresss is the specified byte address 0 ≤ ByteAddress < 264
NOB is the bytes amount you want to read.
Data is the char pointer of the returned data string

int Read_Buffer_AT45D(int Buffer, int ByteAddress, int NOB, char *Data);

Function : Buffer read.

Reference : **Buffer** is the specified buffer number 1- Buffer1 2-Buffer
ByteAddresss is the specified byte address 0 ≤ ByteAddress < 264
NOB is the bytes amount you want to read.
Data is the char pointer of the returned data string

int Transfer_AT45D(int Page, int Buffer);

Function : Main memory page to buffer transfer.

Reference : **Page** is the specified page address in main memory 0 ≤ Page < 2048
Buffer is the specified buffer number 1- Buffer1 2-Buffer

int Compare_AT45D(int Page, int Buffer);

Function : Main memory page to buffer compare.

Reference : **Page** is the specified page address in main memory 0 ≤ Page < 2048
Buffer is the specified buffer number 1- Buffer1 2-Buffer

int Buffer_Write_AT45D(int Buffer, int ByteAddress, int NOB, char *Data);

Function : Buffer write.

Reference : **Buffer** is the specified buffer number 1- Buffer1 2-Buffer
ByteAddresss is the specified byte address 0 ≤ ByteAddress < 264
NOB is the bytes amount you want to write.
Data is the char pointer of the data string you want to write

int Program_Erase_AT45D(int Buffer, int Page);

Function : Buffer to main memory page program with built-in erase.

Reference : **Buffer** is the specified buffer number 1- Buffer1 2-Buffer
Page is the specified page address in main memory 0 ≤ Page < 2048

int Program_No_Erase_AT45D(int Buffer, int Page);

Function : Buffer to main memory page program without built-in erase.

Reference : **Buffer** is the specified buffer number 1- Buffer1 2-Buffer
Page is the specified page address in main memory 0 ≤ Page < 2048

int Program_Main_AT45D(int Buffer, int Page, int ByteAddress, int NOB, char *Data);

Function : Main memory page program through buffer.

Reference : **Buffer** is the specified buffer number 1- Buffer1 2-Buffer
Page is the specified page address in main memory 0 ≤ Page < 2048
ByteAddresss is the specified byte address 0 ≤ ByteAddress < 264
NOB is the bytes amount you want to write.
Data is the char pointer of the data string you want to write

int Rewrite_AT45D(int Buffer, int Page);

Function : Auto page rewrite.

Reference : **Buffer** is the specified buffer number 1- Buffer1 2-Buffer
Page is the specified page address in main memory 0 ≤ Page < 2048

3.2 Secure Memory Cards

3.2.1 ATMEL AT88SC102

int Read_102_Byte(int StartAddress, char *Data)
int Read_102_Word(int StartAddress, char *Data)
int Write_102_Word(int StartAddress, char *Data)
int Erase_102_Word(unsigned int WordAddress)
int Fuse_High_102(void)
int Fuse_Low_102(void)
int Verify_102_SC(char *PSC)
int Erase_102_AZ1(char *PSC)
int Erase_102_AZ2(char *PSC)

int Erase_102_AZ2_EC(char *PSC)
int Blow_Manufacturer_Fuse_102(void)
int Blow_EC2EN_Fuse_102(void)
int Blow_Issuer_Fuse_102(void)

3.2.2 ATMEL AT88SC153

int Read_153(int ZoneNumber, int StartAddress, int NOB, char *Data)

Function : Read data from the specified user zone of AT88SC153 Card.

Reference : **ZoneNumber** is the user zone number of card.
0 ≤ ZoneNumber ≤ 3 Configuration Zone Number is 3.
StartAddress 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.

int Write_153(int ZoneNumber, int StartAddress, int NOB, char *Data)

Function : Write data to the specified zone of AT88SC153 Card.

Reference : **ZoneNumber** is the user zone number of card.
0 ≤ ZoneNumber ≤ 3 Configuration Zone Number is 3.
StartAddress 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.

int Read_Fuse_153(char *Fuses)

Function : Read the Fuse status of AT88SC153 card.

Reference : **Fuses** is the pointer of char that stores the card's fuses status.

int Blow_Fuse_153(char Fuses)

Function : Change the Fuse status of AT88SC153 card.

Reference : **Fuses** is the setting data to the card's fuses status.

int Verify_Password_153(int ReadWrite, int SetNumber, char *Password)

Function : Verify the password of AT88SC153 card.

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.

int Init_Authentication_153(char *Q0)

Function : Initialize the authentication process of AT88SC153 card.

Reference : **Q0** is the pointer of random number generated by card reader.

int Verify_Authentication_153(char *Q1)

Function : Verify AT88SC153 card Authentication Result.

Reference : **Q1** is the pointer of first result generated by card.

int Compute_Challenge(char *Q0, char *GC, char *Ci, char *Q1, char *Q2)

Function : Compute Challenges for AT88SC153 Card Authentication Process.

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.

3.2.3 ATMEL AT88SC1608

int Select_1608_User_Zone(int ZoneAddress)

Function : Select the operated user zone of AT88SC1608 Card.

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

int Read_1608_User_Zone(int ByteAddr, int NOB, char *Data)

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

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.

int Write_1608_User_Zone(int ByteAddr, int NOB, char *Data)

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

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.

int Read_1608_Configuration(int ByteAddr, int NOB, char *Data)

Function : Read data form the configuration zone of AT88SC1608 card.

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.

int Write_1608_Configuration(int ByteAddr, int NOB, char *Data)

Function : W rite data to the configuration zone of AT88SC1608 card.

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.

int Read_1608_Fuses(char *Fuses)

Function : Read the Fuse status of AT88SC1608 card.

Reference : **Fuses** is the pointer of char that stores the card's fuses status.

int Write_1608_Fuses(char *Fuses)

Function : Change the Fuse status of AT88SC1608 card.

Reference : **Fuses** is the pointer of char that set the card's fuses status.

int Verify_1608_Password(int ReadWrite, int SetNumber, char *Password)

Function : Verify the password of AT88SC1608 card.

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.

int Init_1608_Authentication(char *Q0)

Function : Initialize the authentication process of AT88SC1608 card.

Reference : **Q0** is the pointer of random number generated by card reader.

int Verify_1608_Authentication(char *Q1)

Function : Verify AT88SC1608 card Authentication Result.

Reference : **Q1** is the pointer of first result generated by card.

int Compute_Challenge(char *Q0, char *GC, char *Ci, char *Q1, char *Q2)

Function : Compute Challenges for AT88SC1608 Card Authentication Process.

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.

3.2.4 INFINEON SLE4428

int Read_4428_With_PB(int StartPos, int NOB, char *Bfr, char *PB_Bfr)

Function : Read data and protection bit from SLE4428 card..

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.

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

Function : Read data only from SLE4428 card.

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.

int Write_4428(int StartPos, char DestByte, char PBSetFlag)

Function : Write data to SLE4428 card.

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 writen.

PBSetFlag PBSetFlag = 1 Kill the Protection Byte.

PBSetFlag = 0 Leave the Protection Byte.

int Verify_4428_PSC(char PSC1, char PSC2)

Function : Verify SLE4428 card password code.

Reference : **PSC1** is the first byte of the 3-byte pass code

PSC2 is the second byte of the 3-byte pass code

int Read_4428_SM(char *SM_Bfr, char *SM_PB_Bfr)

Function : Read SLE4428 card security memory.

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.

3.2.5 INFINEON SLE4442

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

Function : Read data only from SLE4442 card.

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.

int Read_4442_PB(char *PB_Bfr)

Function : Read protection bit from SLE4442 card.

Reference : **PB_Bfr** is the char pointer of the returned 4-Byte long ProtectionByte data.

int Write_4442(int StartPos, char DestByte, char PBSetFlag)

Function : Write data to SLE4442 card.

Reference : **StartPos** is the start address of data you want to write to the card.
DestByte is the char data that is going to be written.
PBSetFlag PBSetFlag = 1 Kill the Protection Byte.
PBSetFlag = 0 Leave the Protection Byte.

int Verify_4442_PSC(unsigned char PSC1, unsigned char PSC2, unsigned char PSC3)

Function : Verify SLE4442 card password code.

Reference : **PSC1** is the first byte of the 3-byte pass code
PSC2 is the second byte of the 3-byte pass code
PSC3 is the third byte of the 3-byte pass code

int Read_4442_SM(char *SM_Bfr)

Function : Read SLE4442 card security memory.

Reference : **SM_Bfr** is the char pointer of the returned data string.

int Write_4442_SM(int SMAddress, char SMByte)

Function : Write SLE4442 card security memory.

Reference : **SMAddress** is the address of the Security Memory.
SMByte is the data that you want to write to Security Memory.

4 General Steps for Microcontroller Card Manipulation

The card manipulation procedure described as below is often used when user will work on both the main card interface and SAM interface with CPU card. In this case, you can control the main card interface or SAM interface individually:

```
Card_power_on();                    // power the card i/f up but no card is
                                    // powered on
Card_select_sock()                  // select socket
ResetCPUCard(n)                // power a certain socket on and reset the
                                    // CPU card in the selected socket; all
                                    // security status of the card in the other
                                    // socket would not be lost after the card
                                    // socket switching.
                                    // n=1 Inside i/f    n=2 Outside i/f
GetCPUParam()                       // Get CPU card information if it is needed
ExchangeAPDU()                      // Exchange APUD between IFC and IFD
Card_select_sock()                  // hop around
ResetCPUCard(n)                // power another certain socket on and
                                    // reset    the CPU card in the selected
                                    // socket; all security status of the card in
                                    // the other socket would not be lost after
                                    // the card socket switching.
                                    // n=1 Inside i/f   n=2 outside i/f
GetCPUParam()                       // Get CPU card information if it is needed
ExchangeAPDU()                      // Exchange APUD between IFC and IFD
…………..
Card_get_curr_sock()           // return the active reader
DeactiveCPUCard(n)                  // power down current selected socket
                                    // n=1 Inside i/f   n=2 Outside i/f
DeactiveCPUCard(0)                  // deactivate the whole card subsystem
```

5. APIs reference for Microcontroller cards

5.1 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.

5.2 Microcontroller card related functions

int ResetCPUCard(int in_cardNum);

Function : Power a certain socket on and reset the CPU card in the selected socket; all security status of the card in the other socket would not be lost after the card socket switching.

Reference : **in_cardNum** is the card socket number.
n=1 Inside socket n=2 Outside socket.

Return : 0 Function success
1 Not enough memory
2 CPU Card reset failed
3 Invalid input parameter

int GetCPUParam(struct CPUParamStruct **CP);

Function : Get CPU card information.

Reference : **CP** is the pointer to the pointer of CPUParamStruct struct

Sample **:** Sample usage for GetCPUParam()

```
void demo() {
     struct CPUParamStruct *CP;
      int protocol, ATRLen;
```

```
            // Active the IC card & wait for the ATR
            if (ResetCPUCard(2)) //Card socket number in (2)
                proc_messageBox("Reset Card Failed", 2);
            else {
                GetCPUParam(&CP);
                protocol = CP->Protocol;
                ATRLen = CP->ATRLength;
            }
        }
```

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

Function	: After the CPU card is reset, this function can be called to exchange APDU with the card. Now we don’t support the case that Lc and Le are both non-zero with T=0 CPU card.
Reference	: **CAPDU** is the pointer to CommandAPDUStruct struct **RAPDU** is the pointer to ResponseAPDUStruct struct
Return	: 0 Function success 1 Not enough memory 2 T=0 Call type not support 3 T=0 TPDU Exchange Failed 4 Invalid Protocols 5 T=1 Invalid Response packet 6 T=1 Attempts over count 7 T=1 Error Card Abortion

int DeactivateCPUCard (int in_cardNum);

Function	: deactivate the specified card socket.
Reference	: **in_cardNum** is the card socket number. n=0 the whole card system n=1 Inside socket n=2 Outside socket.
Return	: 0 Function success 3 Invalid input parameter

5.3 Sample Code

```
void CPUDemo() {
    struct CommandAPDUStruct CAPDU;
    struct ResponseAPDUStruct RAPDU;
    char data[50], displayString[30];

    Card_power_on();
    Card_select_sock(2);        //Card socket number in (2) Outside socket
    if (ResetCPUCard(2)) {  //Card socket number in (2) Outside socket
        goto_xy(0, 6);
        Disp_write_str("Reset CPU Card Failed");
        return;
    }
```

```
// Select MF, it's not necessary
CAPDU.CLA = 0x00;
CAPDU.INS = 0xa4;
CAPDU.P1 = 0x00;
CAPDU.P2 = 0x00;
CAPDU.Lc = 2;
CAPDU.Le = 0;
CAPDU.Data = data;
RAPDU.Data = data;
data[0] = 0x3f;
data[1] = 0x00;
if (status = ExchangeAPDU(&CAPDU, &RAPDU)) {
        sprintf(displayString, "      APDU Error: %d", status);
        goto_xy(0, 6);
        Disp_write_str(displayString);
        return;
}

Card_select_sock(1);            //Card socket number in (1) Inside socket
if (ResetCPUCard(1)) {  //Card socket number in (1) Inside socket
        goto_xy(0, 6);
        Disp_write_str("Reset CPU Card Failed");
        return;
}

// Select MF, it's not necessary
CAPDU.CLA = 0x00;
CAPDU.INS = 0xa4;
CAPDU.P1 = 0x00;
CAPDU.P2 = 0x00;
CAPDU.Lc = 2;
CAPDU.Le = 0;
CAPDU.Data = data;
RAPDU.Data = data;
data[0] = 0x3f;
data[1] = 0x00;

if (status = ExchangeAPDU(&CAPDU, &RAPDU)) {
        sprintf(displayString, "      APDU Error: %d", status);
        goto_xy(0, 6);
        Disp_write_str(displayString);
        return;
}
```

```
        Card_select_sock(1);
        DeactivateCPUCard(1);
        Card_select_sock(2);     // Also you can deactiveate the socket
        DeactivateCPUCard(2); // as soon as you finished the manipulation
                                        // on the specified socket
        DeactivateCPUCard(0); // Also you can deactivate the whole card system
                                        // without deactivating them individually
}
```