MagCard
Card Expert

MC900Micro
Smartcard Terminal

Manual
V1.0

Silone Magcard, Inc. Visit http://www.magcard.com

Silone Magcard, Inc.
http://www.magcard.com

MC900Micro Manual

Revision History

Revision	Issue Date	Comments
1.0 Draft 1	June 6, 1999	Original version

Table of Contents

Features & List of Return Values 1

Initialization Functions

OpenPort 2
ClosePort 3
SelectPort 4
PowerOn 5
PowerOff 6
CardPresent 7
CardChanged 8
GetCardType 9
GetCardType Return Values 10
GetCardName 11

4428 Functions

Read_4428_With_PB 12
Read_4428_NO_PB 13
Write_4428 14
Verify_4428_PSC 15
Read_4428_SM 16

4442 functions

Read_4442_With_PB 17
Read_4442_NO_PB 18
Read_4442_PB 19
Write_4442_PB 20
Write_4442 21
Write_4442_Array 22
Verify_4442_PSC 23
Read_4442_SM 24
Write_4442_SM 25

AT88SC101/102 Functions

Read_102_Bit 26
Write_102_Bit 27
Read_102_Byte 28
Write_102_Byte 29

Read_102_Byte_Array 30
Write_102_Byte_Array 31
Read_102_Word 32
Write_102_Word 33
Erase_102_Word 34
Erase_102_Word_Array 35
Verify_102_SC 36
Verify_102_EZ1 37
Verify_102_EZ2 38
Erase_102_Global 39
Erase_102_AZ1 40
Erase_102_AZ2 41
Fuse_High_102 42
Fuse_Low_102 43

AT24C01A/02 Functions

Read_AT24C01A 44
Write_AT24C01A_Byte 45
Write_AT24C01A_Page 46

AT24C04/08/16 Functions

Read_AT24C16 47
Write_AT24C16_Byte 48
Write_AT24C16_Page 49

AT24C32/64 Functions

Read_AT24C64 50
Write_AT24C64_Byte 51
Write_AT24C64_Page 52

AT24C128/256 Functions

Read_AT24C256 53
Write_AT24C256_Byte 54
Write_AT24C256_Page 55

AT88SC1608 Functions

Select_1608_User_Zone 56
Read_1608_User_Zone 57
Write_1608_User_Zone 58
Read_1608_Configuration 59
Write_1608_Configuration 60
Read_1608_Fuses 61

Write_1608_Fuses 62
Verify_1608_Password 63
Init_1608_Authentication 64
Verify_1608_Authentication 65
Compute_Challenge 66

AT88SC153 Functions
Write_153 67
Read_153 68
Verify_Password_153 69
InitializeAuthentication_153 70
VerifyAuthentication_153 71
Compute_Challenge 72
ReadFuse_153 73
BlowFuse_153 74

The example card read/write procedure..75

API Manual

Together with the demo program, an API library in DLL format is provided for Microsoft Windows programmers. The name of the DLL is Micro900.dll, and the associated library module for DLL dynamic linking is Micro900.lib. This manual explains the usage of the API function in the DLL. Further information can be found in the sample source code.

The functions usually have return values indicating the result of the calling. Here is the list of the return values:

ERR_SUCCESS = 0
ERR_NO_CARD = 1
ERR_TIMEOUT = 2
ERR_COUNTER = 3
ERR_PSC1 = 4
ERR_PCS2 = 5
ERR_FORMAT = 6,

ERR_NOT_ENOUGH_MEMORY = 7,
ERR_CREATE_PORT_FILE = 8,
ERR_GET_COMM_STATE = 9,
ERR_SET_COMM_STATE = 10,
ERR_SET_COMM_MASK = 11,
ERR_SET_COMM_TIMEOUTS = 12,
ERR_CREATE_THREAD = 13,
ERR_NULL_PORT_HANDLE = 14,

The programmer need only know that a return value of 0 indicates the calling is successful. Return value other than 0 indicates the calling is terminated because of some error, which is most likely caused by the incorrect or unreliable cable connection between the card reader and the host computer. The programmer can use the other return values for diagnostic purposes, and in this case the explanation of the values can be obtained from Magcard technical support. See http://www.magcard.com for contact information.

OpenPort

Prototype

int WINAPI OpenPort(char *PortName, int *PortHandle);

Function

Initialize the communication port and the reader

Parameters

PortName: Points to the buffer holding the port name. The port name can be one of the following: COM1, COM2. It's must be the name of the port to which the reader is connected. This parameter is an input parameter.

PortHandle: This is a 4-byte integer holding the handle of the opened port. Application can use this handle to distinguish from different port in use. For example: the SelectPort function must use this handle to select the port for the following operations. The handle of an opened port can be any value other than NULL(0), if this handle is NULL after calling the function, the application must not continue operation. This is a output Parameter.

Return value

Refer to the list of return values on Page.1.

Remarks

Before any other operation to the reader, this function must be called first. This function works in couple with the CloseReader function. After calling this function, the application can check the return value and check the PortHandle to see if the open process is successful. On successful opening of the port, the return value must be 0, and the PortHandle must not be 0.

See also

SelectPort, ClosePort, PowerOn, PowerOff
List of return values

ClosePort

Prototype

```
int WINAPI ClosePort( );
```

Function

Close the communication port which is previously open by OpenPort.

Parameters

No parameters

Return value

Refer to the list of return values on Page.1.

Remarks

After all the reader operations and before end of the applicaton this function must be called to close the previously opened communication port. This function must work in couple with the OpenPort function.

See also

OpenPort, SelectPort, PowerOn, PowerOff,
List of return values

SelectPort

Prototype

```
int WINAPI SelectPort( int PortHandle );
```

Function

Select the communication port to which the following operation is applied.

Parameters

PortHandle is the handle of the port to be selected. This handle must be previously set be calling OpenPort function and shall not be NULL(0).

Return value

Refer to the list of return values on Page.1.

Remarks

If more than one reader is installed on the host, and the application want all of them to operate simultaneously, the application must call this function to select the reader (port) to which the following operations is applied. After calling this function, all the following operations before the next calling of SelectPort are applied to the reader (port) selected.

See also

OpenPort, ClosePort, PowerOn, PowerOff
List of return values

PowerOn

Prototype

int WINAPI PowerOn( );

Function

Reset the reader and put it in running state.

Parameters

No parameters.

Return value

Refer to the list of return values in Page.1.

Remarks

This function works in conjunction with the PowerOff function to reset the reader. The PowerOff function bring the reader into RESET state and holds it in this state before PowerOn function is called. PowerOn release the reader from it's reset state. After reset state the PWR pin in the card connector will be in IDLE state.

See also

OpenReader, CloseReader, PowerOff
List of return values

PowerOff

Prototype

int WINAPI PowerOff( );

Function

Reset the reader and hold it in reset state.

Parameters

No parameter.

Return value

Refer to the list of return value on Page.1.

Remarks

This function works in conjunction with the PowerOn function to reset the reader. The PowerOff function bring the reader into RESET state and holds it in this state before PowerOn function is called. PowerOn release the reader from it's reset state. After reset state the PWR pin in the card connector will be in IDLE state.

See also

OpenReader, CloseReader, PowerOn
List of return values

CardPresent

Prototype

int WINAPI CardPresent( );

Function

Test the presence of card in the reader.

Parameters

No parameters.

Return value

Return 1 indicating card present, 0 absent.

Remarks

Before apply any operation to the reader this function must be called to see if the card is present in the reader. If card is present, continue the operation. If card is absent, the application must notify the user to insert a card.

See also

CardChanged, GetCardType

CardChanged

Prototype

int WINAPI CardChanged( );

Function

Test if the card in the reader has been changed by the user since the last call of this function.

Parameters

No parameters.

Return value

Return 1 indicating the card has been changed, 0 indicating the card has not been changed since the last call of this function.

Remarks

Before apply operation to the card, the application must know if the card is inserted into the reader. Besides this, sometimes the application must also get to know if the card has been changed since the last operation of the card. This function tells the changed status of the card since last call of this function.

See also

CardPresent, GetCardType

GetCardType

Prototype

int WINAPI GetCardType(char *ATRBuffer);

Function

Power up the card and reset the card, initiate it's ATR sequence and return the ATR string to the caller

Parameters

ATRBuffer: this is the pointer to the 4-byte buffer holding the return ATR string, this buffer is allocated by the caller.

Return value

The return value indicating the detected card type. The possible card type this reader can detect is listed on the following page.

Remarks

Before applying any operation to the card, it's power on reset sequence must be initiated. This includes power on the card, reset the card and issue 32 clock pulses to obtain the 4-byte ATR string of the card. This function must be called after the caller makes sure the card is present by calling CardPresent. After calling GetCardType function, the card type can be determined. The method used for determined the card type is by comparing the return ATR with the known ATRs. The card types determined with this function is limited. This function does not check the validity of the ATRBuffer. So before calling this function, the caller must allocate enough memory for ATRBuffer and make sure this pointer is valid. Pass invalid pointer to the function may cause unpredictable results.

See also

CardPresent, CardChanged, PowerOn, PowerOff

GetCardType return values

```
CT_UNKNOWN = 0,

CT_SLE4432 = 1,
CT_SLE4442 = 2,

CT_SLE4418 = 3,
CT_SLE4428 = 4,

CT_SLE4406 = 5,
CT_SLE4436 = 6,

CT_AT88SC101 = 7,
CT_AT88SC102 = 8,

CT_X76F640 = 9,

CT_AT24C01A = 10,
CT_AT24C02 = 11,
CT_AT24C04 = 12,
CT_AT24C08 = 13,
CT_AT24C16 = 14,
CT_AT24C32 = 15,
CT_AT24C64 = 16,
CT_AT24C128 = 17,
CT_AT24C256 = 18,
CT_AT24C512 = 19,

CT_AT88SC1604 = 20,
CT_AT88SC1608 = 21,
CT_AT88SC153 = 22,

CT_AT45D041 = 23,
```

GetCardName

Prototype

```
char * WINAPI GetCardName( int CardType );
```

Function

Return the name of the Card associated with the CardType value returned from the GetCardType function

Parameters

CardType: the CardType value returned from the GetCardType function

Return value

A string pointer pointing to the name string

Remarks

The name of the card is the name defined by the card chip manufacturer, for example "SLE4428". This function return the pointer pointing to the C string holding the name of the card, which is terminated by a 0.

See also

GetCardType

4428 function: Read_4428_With_PB

Prototype

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

Function

Read data bytes of 4428 and their associated protection bits.

Parameters

int StartPos: ths start offset of the bytes to be read, range: 0 to 3FFH
int NOB: number of bytes to be read, limited to 16 bytes maximum.
char *Bfr: pointer to the buffer holding the returned data bytes
char *PB_Bfr: pointer to the buffer holding the returned protection bits. In this buffer, one byte is corresponding to one protection bit with the same order. A 0 byte denotes the the protection bit is 0 which means the data byte is write protected, A 1 byte denotes the data bytes is not protected.

Return value

See list of return values.

Remarks

PB denotes Protection Bit. When calling this function the caller must make sure that the parameter is valid: StartPos and NOB are in valid range, Bfr and PB_Bfr is allocated enough memory by the caller. This function makes no check of the validity of the parameters. Invalid parameters passed to this function may cause unpredictable results.

See also

Read_4428_NO_PB

4428 function: Read_4428_NO_PB

Prototype

int WINAPI Read_4428_NO_PB(int StartPos, int NOB, char *Bfr);

Function

Read data bytes of 4428 card.

Parameters

int StartPos: ths start offset of the bytes to be read, range: 0 to 3FFH
int NOB: number of bytes to be read, limited to 16 bytes maximum.
char *Bfr: pointer to the buffer holding the returned data bytes

Return value

See list of return values.

Remarks

PB denotes Protection Bit. When calling this function the caller must make sure that the parameter is valid: StartPos and NOB are in valid range, Bfr and is allocated enough memory by the caller. This function makes no check of the validity of the parameters. Invalid parameters passed to this function may cause unpredictable results.

See also

Read_4428_With_PB

4428 function: Write_4428

Prototype

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

Function

Write one byte of data and associated protection bit (if required) into 4428.

Parameters

int StartPos: the offset of the byte to be written, range: 0 to 3FFH

char DestByte: the value of the byte to be written

char PBSetFlag: 1 write the associated protection bit, 0 do not write the protection bit.

Return value

See list of return values.

Remarks

Before calling this function the caller must make sure that the parameters are valid, that is StartPos and NOB are in valid range and Bfr is allocated enough size of memory. This function makes no validity check of the parameter, so invalid parameters may cause unpredictable results.

See also

4428 function: Verify_4428_PSC

Prototype

int WINAPI Verify_4428_PSC(char PSC1, char PSC2);

Function

Verify the 2-byte PSC of 4428

Parameters

char PSC1, char PSC2: 2 PSC bytes

Return value

See list of return values

Remarks

According to 4428 data sheet, PSC denotes Programmable Security Code, which is also called password, PIN etc. Before any write operation to the card, the 2-byte PSC must be verified. After 8 continual failure of PSC Verification the card becomes deadlock.

See also

Read_4428_SM

4428 function: Read_4428_SM

Prototype

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

Function

Read the security memory (SM) area of 4428

Parameters

char *SM_Bfr: pointer to the buffer holding the return 3-byte security memory

char *SM_PB_Bfr: pointer to the buffer holding the return 3-byte protection bits of the security memory. Every byte in this buffer corresponds to one protection bit with the same order. A 0 byte denoted the protection bit is written and the associated SM byte is protected, A 1 byte means the protection bit is not written and the SM byte is not protected.

Return value

See list of return values.

Remarks

SM denotes Security Memory. The last 3 bytes in 4428, namely Error Counter, PSC1, PSC2, are called SM. This function is to read back these 3 bytes. The 1st byte read back is Error Counter, then PSC1, then PSC2. Before calling this function the caller must make sure the parameter is valid, that is SM_Bfr and SM_PB_Bfr is allocated enough size of memory. This function makes not validity check of the parameters thus invalid parameter may cause unpredictable results.

See also

Verify_4428_PSC

4442 function: Read_4442_With_PB

Prototype

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

Function

Read data bytes of 4442 and their associated protection bits.

Parameters

int StartPos: ths start offset of the bytes to be read, range: 0 to 0FFH
int NOB: number of bytes to be read, limited to 16 bytes maximum.
char *Bfr: pointer to the buffer holding the returned data bytes
char *PB_Bfr: pointer to the buffer holding the returned protection bits. In this buffer, one byte is corresponding to one protection bit with the same order. A 0 byte denotes the the protection bit is 0 which means the data byte is write protected, A 1 byte denotes the data bytes is not protected.

Return value

See list of return values.

Remarks

PB denotes Protection Bit. When calling this function the caller must make sure that the parameter is valid: StartPos and NOB are in valid range, Bfr and PB_Bfr is allocated enough memory by the caller. This function makes no check of the validity of the parameters. Invalid parameters passed to this function may cause unpredictable results. Because only the beginning 32 bytes of 4442 have protection bits, only the first 32 protected bits read back is useful. Other PB_Bfr bytes is meaningless and their value are not predictable.

See also

Read_4442_NO_PB

4442 function: Read_4442_NO_PB

Prototype

Read_4442_NO_PB(int StartPos, int NOB, char *Bfr);

Function

Read data bytes of 4442 card without protection bits.

Parameters

int StartPos: ths start offset of the bytes to be read, range: 0 to 0FFH
int NOB: number of bytes to be read, limited to 16 bytes maximum.
char *Bfr: pointer to the buffer holding the returned data bytes

Return value

See list of return values.

Remarks

PB denotes Protection Bit. When calling this function the caller must make sure that the parameter is valid: StartPos and NOB are in valid range, Bfr and is allocated enough memory by the caller. This function makes no check of the validity of the parameters. Invalid parameters passed to this function may cause unpredictable results.

See also

Read_4442_With_PB

4442 function: Read_4442_PB

Prototype

int WINAPI Read_4442_PB(char *PB_Bfr);

Function

Read all the protection bits of 4442.

Parameters

char *PB_Bfr: the pointer pointing to the 32-byte buffer holding the read back 32 protection-bit bytes associated with the first 32 bytes of 4442.

Return value

See list of return values.

Remarks

This function reads back the 32 protection bits and put them into 32 bytes, one bit in one byte.The 1 byte indicates the corresponding protection bit is written and the data byte is protected, 0 byte the correcponding protection bit is not written and the data byte is not protected. Before calling this function, the caller must make sure the parameter is valid. This function make no validity check of the parameter values, so invalid parameter may cause unpredictable results.

See also

Write_4442_PB

4442 function: Write_4442_PB

Prototype

int WINAPI Write_4442_PB(int Address); // bit address

Function

Write the specified the protection bit.

Parameters

int Address: the byte address of the byte whose protection bit is to be written, range: 0 to 31

Return value

See list of return values.

Remarks

This function writes the protection bit with the given address to 0 thus make the accociated data byte write protected. If the given address is out of the valid reange this function does nothing but return immediately. The return value will indicating the parameter error.

See also

Read_4442_PB

4442 function: Write_4442

Prototype

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

Function

Write 1 byte data to 4442. If required write protection bit.

Parameters

int StartPos: the byte address to write, ranging 0 to 255

char DestByte: the data to be written

char PBSetFlag: the protection bit write flag, 1 write protection bit, 0 do not write protection bit

Return value

See list of return values.

Remarks

Write 1 byte of data to 4442 with the given address. If the PBSetFlag is 1 the write the associated protection bit. If the address is greater than 31 then the PBSetFlag is omitted.

See also

Write_4442_PB, Write_4442_Array

4442 function: Write_4442_Array

Prototype

int WINAPI Write_4442_Array
(int StartPos, int NOB, char *DestByte, char *PBSetFlag);

Function

Write multiple bytes of data and protection bits to 4442.

Parameters

int StartPos: the start address of the bytes, ranging 0 to 255.
int NOB: Number of Bytes to be written to the card
char *DestByte: the pointer pointing to the buffer holding the bytes to be written to the card
char *PBSetFlag: the pointer pointing to the buffer holding the protection bits to be written to the card. In this buffer every byte corresponds to one protection bit, the byte with a value of 1 means the corresponding protection bit is written, 0 means do not write protection bit.

Return value

See list of return values.

Remarks

Write multiple bytes of data and protection bits to 4442. The PBSetFlag with an address greater than 31 is meaningless and is omitted.

See also

Write_4442_PB, Write_4442

4442 function: Verify_4442_PSC

Prototype

int WINAPI Verify_4442_PSC(char PSC1, char PSC2, char PSC3);

Function

Verify the 3-byte PSC of 4442

Parameters

The 2 parameter PSC1, PSC2 and PSC3 are the 3-byte PSC to be verified.

Return value

See list of return values.

Remarks

PSC denotes to Programmable Security Code according to the 4442 data sheet. Before any write operation the 3-byte PSC must be verified. 3 continual failure of PSC verification will cause the card deadlock.

See also

Read_4442_SM

4442 function: Read_4442_SM

Prototype

```
int WINAPI Read_4442_SM( char *SM_Bfr );
```

Function

Read back the 4-byte Security Memory (SM) of 4442.

Parameters

char *SM_Bfr: the pointer pointing to the buffer holding the read back 4 bytes SM.

Return value

See list of return values.

Remarks

SM denotes Security Memory. There is a 4-byte SM in 4442, namely Error Counter, PSC1, PSC2, PSC4. This function is to read back this 4-byte SM. The 1st byte is Error Counter, 2nd is PSC1, 3rd is PSC2 and 4th PSC3. Before calling this function the caller must make sure the parameters are valid, that is SM_Bfr must be allocated enough size of memory. This function makes no validity check of the parameters so invalid parameter may cause unpredictable results.

See also

Verify_4442_PSC

4442 function: Write_4442_SM

Prototype

int WINAPI Write_4442_SM(int SMAddress, char SMByte);

Function

Write one byte of data to 4442 SM with the given address.

Parameters

int SMAddress: the address of the data byte to be written, 0 for Error Counter, 1 PSC1, 2, PSC2, 3 PSC3.

char SMByte: the data byte to be written.

Return value

See list of return values.

Remarks

This function writes one byte to 4442 SM. SMAddress ranges from 0 to 3, other value is meaningless and will cause the function do nothing but return immediately.

See also

Read_4442_SM

AT88SC101/102 function: Read_102_Bit

Prototype

int WINAPI Read_102_Bit(int StartBitAddress, int NOBit, char *Data);

Function

Read data bits of AT88SC102.

Parameters

int StartBitAddress: start bit address to be read. The first physical bit in 102 is addressed 0.

int NOB: number of bits to be read, limited to 256 bits maximum.

char *Data: the pointer pointing to the buffer holding the data read. This buffer is allocated by the caller. The data bits read is packed bit by bit into bytes, the size of the buffer is (NOB div 8) + 1 bytes.

Return value

See list of return values.

Remarks

Before calling this function the caller must guarantee the validity of the parameters, that is StartBitAddress and NOBit must be in the valid range, Data is allocated enough size of memory. This function assumes the parameters are valid so invalid parameter may cause unpredictable results.

See also

Write_102_Bit

AT88SC101/102 function: Write_102_Bit

Prototype

int WINAPI Write_102_Bit(int StartBitAddress, int NOBit, char *Data);

Function

Write data bits into 102 card.

Parameters

int StartBitAddress: start bit address. The 1st physical bit in 102 is addressed 0.

int NOB: the number of bits to be written, limited to 256 bits maximum.

char *Data: the pointer pointing to the buffer holding the data to be written. This buffer is set up by the caller before calling this function.

Return values

See list of return values.

Remarks

Before calling this function the caller must guarantee the validity the parameter, StartBitAddress and NOBit must be in the valid range. This functin makes no validity check of the parameters. Invalid parameter may cause unpredictable results.

See also

Read_102_Bit

AT88SC101/102 function: Read_102_Byte

Prototype

int WINAPI Read_102_Byte(int ByteAddress, char *Data);

Function

Read 1 data byte of 102.

Parameters

int ByteAddress: the address of the byte to be read.
char *Data: the pointer pointing to the buffer holding the data read back.

Return values

See list of return values.

Remarks

Before calling this function the caller must guatantee the validity of the parameters, the ByteAddress must be in the valid range, Data is defined a pointer to the valid buffer. This function make no validity check of the parameters so invalid parameter may cause unpredictable results. The data read back with invalid parameter is meaningless.

See also

Write_102_Byte

AT88SC101/102 function: Write_102_Byte

Protetype

int WINAPI Write_102_Byte(int ByteAddress, char Data);

Function

Write 1 byte data to 102.

Parameters

int ByteAddress: the byte address to be written.
char Data: the data byte to be written to the card.

Return values

See list of return values.

Remarks

Before calling this function the caller must guarantee the validity of the parameters, ByteAddress must be in the valid range. This function makes no validity check of the parameters so invalid paramter may cause unpredictable results.

See also

Read_102_Byte

AT88SC101/102 function: Read_102_Byte_Array

Prototype

int WINAPI Read_102_Byte_Array(int ByteAddress, int NOB, char *Data);

Function

Read multiple bytes of data from 102.

Parameters

int ByteAddress: start byte address. The 1st physical byte of 102 is addressed 0.

int NOB: the number of bytes to be read back, limited to 16 bytes maximum.

char *Data: the pointer of the buffer holding the data read back. This buffer is allocated by the caller with the size of NOB bytes. The data is put into this buffer byte by byte.

Return values

See list of return values.

Remarks

Before calling this function the caller must guarantee the validity of the paramters, that is StartByteAddress and NOB must be in the valid range, the size of the Data buffer is big enough to hold the data read back. This function makes no validity check of the parameters so invalid parameter may cause unpredictable results. The data read back with the invalid parameter is meaningless.

See also

Write_102_Byte_Array

AT88SC101/102 function: Write_102_Byte_Array

Prototype

int WINAPI Write_102_Byte_Array(int ByteAddress, int NOB, char *Data);

Function

Write multiple byte of data to 102 card.

Parameters

int ByteAddress: start byte address to be written. The 1st physical byte in 102 is addressed 0.

int NOB: the number of byte to be written to the card, limited to 16 bytes maximum.

char *Data: the pointer of the buffer holding the data to be written. The size of the buffer is NOB bytes.

Return values

See list of return values.

Remarks

Before calling this function the caller must guarantee the validity of the paramters, that is StartByteAddress and NOB must be in the valid range. This function makes no validity check of the parameters so invalid parameter may cause unpredictable results.

See also

Read_102_Byte_Array

AT88SC101/102 function: Read_102_Word

Prototype

int WINAPI Read_102_Word(int WordAddress, char *Data);

Function

Read 1 word of data from 102 card.

Paramters

int WordAddress: the word address to be read. The word address is different from the byte address. The strart byte address of the card is 0, the start word address is also 0. The byte with a byte address 1 does not have a word address, the byte with a byte address 2 has the word address 1, the byte with a byte address 2n has the word address n, the byte with a byte address 2n+1 does not have a word address.

char *Data: the pointer of the buffer holding the data read back.

Return values

See list of return values.

Remarks

Before calling this function the caller must guarantee the validity of the parameters: WordAddress is in the valid range, Data pointer to the valid word buffer. This function make no validity check of the parameters so invalid parameter may cause unpredictable results.

See also

Write_102_Word

AT88SC101/102 function: Write_102_Word

Prototype

int WINAPI Write_102_Word(int WordAddress, char *Data);

Function

Write 1 word of data to 102 card.

Parameters

int WordAddress: the word address to be written. The word address is different from the byte address. The strart byte address of the card is 0, the start word address is also 0. The byte with a byte address 1 does not have a word address, the byte with a byte address 2 has the word address 1, the byte with a byte address 2n has the word address n, the byte with a byte address 2n+1 does not have a word address.

char *Data: the pointer of the buffer holding the data to be written.

Return values

See list of return values.

Remarks

Before calling this function the caller must guarantee the validity of the parameters: WordAddress is in the valid range, Data pointer to the valid word buffer. This function make no validity check of the parameters so invalid parameter may cause unpredictable results.

See also

Read_102_Word

AT88SC101/102 function: Erase_102_Word

Prototype

int WINAPI Erase_102_Word(int WordAddress);

Function

Erase one word of 102 card

Parameters

int WordAddress: the word address to be written. The word address is different from the byte address. The strart byte address of the card is 0, the start word address is also 0. The byte with a byte address 1 does not have a word address, the byte with a byte address 2 has the word address 1, the byte with a byte address 2n has the word address n, the byte with a byte address 2n+1 does not have a word address.

Return values

See list of return values.

Remarks

Before calling this function the caller must guarantee the validity of the parameters: WordAddress is in the valid range. This function make no validity check of the parameters so invalid parameter may cause unpredictable results.

See also

Read_102_Word, Write_102_Word, Erase_102_Word_Array

AT88SC101/102 function: Erase_102_Word_Array

Prototype

int WINAPI Erase_102_Word_Array(int WordAddress, int WordCount);

Function

Erase multiple words of 102 card.

Parameters

int WordAddress: the word address to be written. The word address is different from the byte address. The strart byte address of the card is 0, the start word address is also 0. The byte with a byte address 1 does not have a word address, the byte with a byte address 2 has the word address 1, the byte with a byte address 2n has the word address n, the byte with a byte address 2n+1 does not have a word address.

int WordCount: the number of words to be erased.

Return values

See list of return values.

Remarks

Before calling this function the caller must guarantee the validity of the parameters: WordAddress and WordCount are in the valid range. This function make no validity check of the parameters so invalid parameter may cause unpredictable results.

See also

Erase_102_Word

AT88SC101/102 function: Verify_102_SC

Prototype

int WINAPI Verify_102_SC(char *SCArray);

Function

Verify the Security Code (SC) of 102 card.

Parameters

char *SCArray: the pointer of the 2-byte SC buffer to be verified.

Return values

See list of return values.

Remarks

SC denotes Security Code (Defined in 102 data sheet). Before any operation to the card this 2-byte Security Code must be verified. 4 continual failure of the SC verification will cause the card deadlock.

See also

AT88SC101/102 function: Verify_102_EZ1

Prototype

int WINAPI Verify_102_EZ1(char *EZ1Array);

Function

Verify EZ1 of 102 card.

Parameters

Char *EZ1Array: the pointer to the 6-byte EZ1 buffer to be verified.

Return values

See list of return values.

Remarks

EZ1 denotes Application Zone 1 Erase Key of 102 card. This is defined in 102 data sheet. Refer to 102 data sheet for detailed explanation of the function of this Erase Key.

AT88SC101/102 function: Verify_102_EZ2

Prototype

int WINAPI Verify_102_EZ2(char *EZ1Array);

Function

Verify EZ2 of 102 card.

Parameters

Char *EZ2Array: the pointer to the 6-byte EZ2 buffer to be verified.

Return values

See list of return values.

Remarks

EZ2 denotes Application Zone 2 Erase Key of 102 card. This is defined in 102 data sheet. Refer to 102 data sheet for detailed explanation of the function of this Erase Key.

AT88SC101/102 function: Erase_102_Global

Prototype

int WINAPI Erase_102_Global( );

Function

Erase the global 102 card.

Parameters

No parameters.

Return values

See list of return values.

Remarks

According to 102 data sheet, when the FUSE2 fuse of 102 is 1, calling this function will erase the whole card. Refer to 102 data sheet for detailed explanation of the fuses and erase global function.

See also

AT88SC101/102 functions: Erase_102_AZ1

Prototype

int WINAPI Erase_102_AZ1(char *EZArray);

Function

Erase the AZ1 data zone of 102 card.

Parameters

Char *EZArray: the pointer to the buffer holding the 6-byte caller supplied erase key EZ1

Return values

See list of return values.

Remarks

Refer to 102 data sheet for detailed explanation of the 102 AZ1 data zone and this erase function.

See also

Erase_102_AZ2

AT88SC101/102 functions: Erase_102_AZ2

Prototype

int WINAPI Erase_102_AZ2(char *EZArray);

Function

Erase the AZ2 data zone of 102 card.

Parameters

Char *EZArray: the pointer to the buffer holding the 4-byte caller supplied erase key EZ2

Return values

See list of return values.

Remarks

Refer to 102 data sheet for detailed explanation of the 102 AZ2 data zone and this erase function.

See also

Erase_102_AZ1

AT88SC101/102 function: Fuse_High_102

Prototype

int WINAPI Fuse_High_102( );

Fuction

Set the 102 FUS pin to 5V to blow the FUSE2 fuse of 102.

Parameters

No parameters.

Return values

See list of return values.

Remarks

Set FUS pin to 5V will blow the FUSE2 fuse in 102 card. After operation of the card the application must call Fuse_Low_102 to set the FUS pin to 0V in order to operate another card. See 102 data sheet for detailed explanation of the fuses and their functions.

See also

Fuse_Low_102

AT88SC101/102 function: Fuse_Low_102

Prototype

int WINAPI Fuse_Low_102( );

Function

Set the FUS pin of 102 to 0V after blow the FUSE2 fuse.

Parameters

No parameters.

Return values

See list of return values.

Remarks

Set FUS pin to 5V will blow the FUSE2 fuse in 102 card. After operation of the card the application must call Fuse_Low_102 to set the FUS pin to 0V in order to operate another card. See 102 data sheet for detailed explanation of the fuses and their functions.

See also

Fuse_High_102

AT24C01A/02 function: Read_AT24C01A

Prototype

int WINAPI Read_AT24C01A(int Address, int NOB, char *Data);

Function

Read bytes from 24C01A or AT24C02.

Parameters

int Address: the start byte address of the read operation. For AT24C01A, the value should be in the range of 0 to 7FH. For AT24C02, the value should be in the range of 0 to FFH.

int NOB: number of bytes to read.

char *Data: the pointer to the buffer holding the data read back. This buffer is allocated by the caller with the size of NOB bytes. The data is put into this buffer byte by byte.

Return values

See list of return values.

Remarks

Before calling this function the caller must guarantee the validity of the paramters, that is Address and NOB must be in the valid range, the size of the Data buffer is big enough to hold the data read back. This function makes no validity check of the parameters so invalid parameter may cause unpredictable results. The data read back with the invalid parameter is meaningless.

See also

Write_AT24C01A_Byte, Write_AT24C01A_Page

AT24C01A/02 function: Write_AT24C01A_Byte

Prototype

int WINAPI Write_AT24C01A_Byte(int Address, char Data);

Function

Write one byte to AT24C01A/02.

Parameters

int Address: the byte address of the write operation. For AT24C01A, the value of Address must be in the range of 0 to 7FH. For AT24C02, the value of Address must be in the range of 0 to FFH.

Char Data: the one byte data to be written to the card.

Return values

See list of return values.

Remarks

This function writes only one byte to AT24C01A. The address must be in the range of 0 to 7FH for AT24C01A or 0 to FFH for AT24C02, otherwise the data byte will be written to unpredicted address of the card.

See also

Write_AT24C01A_Page

AT24C01A/02 function: Write_AT24C01A_Page

Prototype

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

Function

Write bytes to AT24C01A card.

Parameters

int Address: the start byte address of the write operation. For AT24C01A, the Address must be in the range of 0 to 7FH. For AT24C02, the Address must be in the range of 0 to FFH.

int NOB: the number of bytes to be written to the card. The value must be in the range of 1 to 8.

char *Data: the pointer to the buffer holding the data byte to be written to the card.

Return values

See list of return values.

Remarks

This function write multiple bytes to AT24C01A/02. The address must be in the range of 0 to 7FH for AT24C01A and 0 to FFH for AT24C02, and NOB must Address + NOB must not exceed the boundary of the card, otherwise unpredicted bytes will be written to the card.

The page size of AT24C01A/02 is 8 bytes, and partial writes are allowed. NOB must not exceed 8.

See also

Write_AT24C01A_Byte

AT24C04/08/16 function: Read_AT24C16

Prototype

int WINAPI Read_AT24C16(int Address, int NOB, char *Data);

Function

Read bytes from AT24C04/08/16.

Parameters

int Address: the start byte address of the read operation. For AT24C04, the value must be in the range of 0 to 1FFH. For AT24C08, the value must be in he range of 0 to 3FFH. For AT24C16, the value must be in the range of 0 to 7FF.

int NOB: number of bytes to read.

char *Data: the pointer to the buffer holding the data read back. This buffer is allocated by the caller with the size of NOB bytes. The data is put into this buffer byte by byte.

Return values

See list of return values.

Remarks

Before calling this function the caller must guarantee the validity of the paramters, that is Address and NOB must be in the valid range, the size of the Data buffer is big enough to hold the data read back. This function makes no validity check of the parameters so invalid parameter may cause unpredictable results. The data read back with the invalid parameter is meaningless.

See also

Write_AT24C16_Byte, Write_AT24C16_Page

AT24C04/08/16 function: Write_AT24C16_Byte

Prototype

int WINAPI Write_AT24C16_Byte(int Address, char Data);

Function

Write one byte to AT24C04/08/16.

Parameters

int Address: the start byte address of the read operation. For AT24C04, the value must be in the range of 0 to 1FFH. For AT24C08, the value must be in he range of 0 to 3FFH. For AT24C16, the value must be in the range of 0 to 7FF.

Char Data: the one byte data to be written to the card.

Return values

See list of return values.

Remarks

This function writes only one byte to AT24C/04/08/16. The address must be in the range of 0 to 1FFH for AT24C04 or 0 to 3FFH for AT24C08 or 0 to 7FFH for AT24C16, otherwise the data byte will be written to unpredicted address of the card.

See also

Write_AT24C16_Page

AT24C04/08/16 function: Write_AT24C16_Page

Prototype

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

Function

Write bytes to AT24C04/08/16 card.

Parameters

int Address: the start byte address of the write operation. For AT24C04, the Address must be in the range of 0 to 1FFH. For AT24C08, the Address must be in the range of 0 to 3FFH. For AT24C16, the address must be in the range of 0 to 7FFH.

int NOB: the number of bytes to be written to the card. The value must be in the range of 1 to 16.

char *Data: the pointer to the buffer holding the data byte to be written to the card.

Return values

See list of return values.

Remarks

This function write multiple bytes to AT24C04/08/16. The address must be in the range of 0 to 1FFH for AT24C04 and 0 to 3FFH for AT24C08 or 0 to 7FFH for AT24C16, and NOB must Address + NOB must not exceed the boundary of the card, otherwise unpredicted bytes will be written to the card.

The page size for AT24C04/08/16 is 16 bytes, and partial page writes are allowed. NOB must not exceed 16.

See also

Write_AT24C16_Byte

AT24C32/64 function: Read_AT24C64

Prototype

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

Function

Read bytes from AT24C32/64.

Parameters

int Address: the start byte address of the read operation. For AT24C32, the value must be in the range of 0 to FFFH. For AT24C64, the value must be in he range of 0 to 1FFFH.

int NOB: number of bytes to read.

char *Data: the pointer to the buffer holding the data read back. This buffer is allocated by the caller with the size of NOB bytes. The data is put into this buffer byte by byte.

Return values

See list of return values.

Remarks

Before calling this function the caller must guarantee the validity of the paramters, that is Address and NOB must be in the valid range, the size of the Data buffer is big enough to hold the data read back. This function makes no validity check of the parameters so invalid parameter may cause unpredictable results. The data read back with the invalid parameter is meaningless.

See also

Write_AT24C64_Byte, Write_AT24C64_Page

AT24C32/64 function: Write_AT24C64_Byte

Prototype

int WINAPI Write_AT24C64_Byte(int Address, char Data);

Function

Write one byte to AT24C32/64.

Parameters

int Address: the start byte address of the read operation. For AT24C32, the value must be in the range of 0 to FFFH. For AT24C64, the value must be in he range of 0 to 1FFFH.

Char Data: the one byte data to be written to the card.

Return values

See list of return values.

Remarks

This function writes only one byte to AT24C32/64. The address must be in the range of 0 to FFFH for AT24C32 or 0 to 1FFFH for AT24C64, otherwise the data byte will be written to unpredicted address of the card.

See also

Write_AT24C64_Page

AT24C32/64 function: Write_AT24C64_Page

Prototype

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

Function

Write bytes to AT24C32/64 card.

Parameters

int Address: the start byte address of the write operation. For AT24C32, the Address must be in the range of 0 to FFFH. For AT24C64, the Address must be in the range of 0 to 1FFFH.

int NOB: the number of bytes to be written to the card. The value must be in the range of 1 to 32.

char *Data: the pointer to the buffer holding the data byte to be written to the card.

Return values

See list of return values.

Remarks

This function write multiple bytes to AT24C32/64. The address must be in the range of 0 to FFFH for AT24C32 and 0 to 1FFFH for AT24C64, and NOB must Address + NOB must not exceed the boundary of the card, otherwise unpredicted bytes will be written to the card.

The page size of AT24C32/64 is 32 bytes, and partial page write is allowed. NOB must not exceed 32.

See also

Write_AT24C64_Byte

AT24C128/256 function: Read_AT24C256

Prototype

int WINAPI Read_AT24C256(int Address, int NOB, char *Data);

Function

Read bytes from AT24C128/256.

Parameters

int Address: the start byte address of the read operation. For AT24C128, the value must be in the range of 0 to 3FFFH. For AT24C256, the value must be in he range of 0 to 7FFFH.

int NOB: number of bytes to read.

char *Data: the pointer to the buffer holding the data read back. This buffer is allocated by the caller with the size of NOB bytes. The data is put into this buffer byte by byte.

Return values

See list of return values.

Remarks

Before calling this function the caller must guarantee the validity of the paramters, that is Address and NOB must be in the valid range, the size of the Data buffer is big enough to hold the data read back. This function makes no validity check of the parameters so invalid parameter may cause unpredictable results. The data read back with the invalid parameter is meaningless.

See also

Write_AT24C256_Byte, Write_AT24C256_Page

AT24C128/256 function: Write_AT24C256_Byte

Prototype

int WINAPI Write_AT24C256_Byte(int Address, char Data);

Function

Write one byte to AT24C128/256.

Parameters

int Address: the start byte address of the read operation. For AT24C128, the value must be in the range of 0 to 3FFFH. For AT24C256, the value must be in he range of 0 to 7FFFH.

Char Data: the one byte data to be written to the card.

Return values

See list of return values.

Remarks

This function writes only one byte to AT24C128/256. The address must be in the range of 0 to 3FFFH for AT24C128 or 0 to 7FFFH for AT24C256, otherwise the data byte will be written to unpredicted address of the card.

See also

Write_AT24C256_Page

AT24C128/256 function: Write_AT24C256_Page

Prototype

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

Function

Write bytes to AT24C128/256 card.

Parameters

int Address: the start byte address of the write operation. For AT24C128, the Address must be in the range of 0 to 3FFFH. For AT24C256, the Address must be in the range of 0 to 7FFFH.

int NOB: the number of bytes to be written to the card. NOB must be in the range of 1 to 32.

char *Data: the pointer to the buffer holding the data byte to be written to the card.

Return values

See list of return values.

Remarks

This function write multiple bytes to AT24C128/256. The address must be in the range of 0 to 3FFFH for AT24C128 and 0 to 7FFFH for AT24C256, and NOB must Address + NOB must not exceed the boundary of the card, otherwise unpredicted bytes will be written to the card.

The page size of AT24C128/256 is 32 bytes, and partial page write is allowed. NOB must not exceed 32.

See also

Write_AT24C256_Byte

AT88SC1608 function: Select_1608_User_Zone

Prototype

int WINAPI Select_1608_User_Zone(int ZoneAddr);

Function

Set the User Zone Address for the following operation.

Parameters

int ZoneAddr: the User Zone Address. In the range of 0 to 7.

Return values

See list of return values.

Remarks

At power on, no access to the user zone is allowed until a Set User Zone Address command is issued. AT88SC1608 has 8 user zones, the zone address must be in the range of 0 to 7.

See also

AT88SC1608 function: Read_1608_User_Zone

Prototype

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

Function

Read data bytes of AT88SC1608 User Zone.

Parameters

int ByteAddress: the start byte address to be read.
int NOB: the number of bytes to be read from AT88SC1608.
char *Data: the pointer to the buffer holding the data bytes read back.

Return values

See list of return values.

Remarks

This function reads the data byte of AT88SC1608 card. Before this read function, the calller must issue Password Verification and/or Authentication commands to the card. Also before calling this function, the caller must make sure the parameter is valid. The char *Data buffer is allocated by the caller and the buffer size must be bigger than NOB. This function make no validity check of the parameter values, so invalid parameter may cause unpredictable results.

See also

Write_1608_User_Zone

AT88SC1608 function: Write_1608_User_Zone

Prototype

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

Function

Write data bytes to AT88SC1608 card.

Parameters

Int ByteAddr: the start byte address for the write operation.
int NOB: the number of byte to be written
char *Data: the pointer to the buffer holding the bytes to be written to the card

Return values

See list of return values.

Remarks

This function write data bytes to the card. Before apply this function, the caller must issue Password Verification and/or Authentication Commands to the card.

See also

Read_1608_User_Zone

AT88SC1608 function: Read_1608_Configuration

Prototype

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

Function

Read data bytes of AT88SC1608 Configuration Zone.

Parameters

int ByteAddress: the start byte address to be read.
int NOB: the number of bytes to be read from AT88SC1608.
char *Data: the pointer to the buffer holding the data bytes read back.

Return values

See list of return values.

Remarks

This function reads the data bytes of AT88SC1608 Configuration Zone. Before this read function, the calller must issue Password Verification and/or Authentication commands to the card. Also before calling this function, the caller must make sure the parameter is valid. The char *Data buffer is allocated by the caller and the buffer size must be bigger than NOB. This function make no validity check of the parameter values, so invalid parameter may cause unpredictable results.

See also

Write_1608_Configuration

AT88SC1608 function: Write_1608_Configuration

Prototype

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

Function

Write data bytes to AT88SC1608 Configuration Zone.

Parameters

int ByteAddr: the start byte address for the write operation.
int NOB: the number of byte to be written
char *Data: the pointer to the buffer holding the bytes to be written to the card

Return values

See list of return values.

Remarks

This function write data bytes to AT88SC1608 Configuration Zone. Before apply this function, the caller must issue Password Verification and/or Authentication Commands to the card.

See also

Read_1608_Configuration_Zone

AT88SC1608 function: Read_1608_Fuses

Prototype
  int WINAPI Read_1608_Fuses(char *Fuses);

Function
  Read the fuses of At88SC1608.

Parameters
  char *Fuses: the pointer to the byte holding the fuses values read back from the card.
  Bit0: FAB fuse, 1 = not blown, 0 = blown
  Bit1: CMA fuse, 1 = not blown, 0 = blown
  Bit2: PER fuse, 1 = not blown, 0 = blown

Return values
  See list of return values.

Remarks
  This function reads back the fuses of AT88SC1608.

See also
  Write_1608_Fuses

AT88SC1608 function: Write_1608_Fuses

Prototype

int WINAPI Write_1608_Fuses( );

Function

Write the next fuse of AT88SC1608.

Parameters

No parameters.

Return values

See list of return values.

Remarks

The fuses are blown sequentially: CMA is blown if FAB is equal to “0”, and PER is blown if CMA is equal to “0”.

See also

Read_1608_Fuses

AT88SC1608 function: Verify_1608_Password

Prototype

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

Function

Verify the read/write password of AT88SC1608 card.

Parameters

int ReadWrite: 0 = Write password, 1 = Read password
int SetNumber: Password set number, 0 to 7 for 8 password sets.
char *Password: Pointer to the 3-byte password to be Verified

Return values

See list of return values.

Remarks

This function verifies a Read or Write password. A valid verification of the password erases the attempts counter. After this function the caller must read the attempts counter to see if the password verification if successful.

See also

Read_1608_Configuration

AT88SC1608 function: Init_1608_Authentication

Prototype

int WINAPI Init_1608_Authentication(char *Q0);

Function

Initialize the authentication process of AT88SC1608 card.

Parameters

char *Q0: this is the pointer to the buffer holding the 8-byte host random number.

Return values

See list of return values.

Remarks

This function sets up the authentication process of the AT88SC1608 card. The caller must generate a random number as the input of this function.

See also

Verify_1608_Authentication

AT88SC1608 function: Verify_1608_Authentication

Prototype

int WINAPI Verify_1608_Authentication(char *Q1);

Function

Execute the 'Verify Authentication' command of the AT88SC1608 card.

Parameters

char *Q1: this is the pointer to the buffer holding the 8-byte host challenge. This challenge is computed by the reader.

Return values

See list of return values.

Remarks

After the initialize authentication command, both the reader and the card computed their result of the challenge. The reader must issue this command to let the card authenticate the reader.

See also

Initialize_1608_Authentication, Compute_Challenge

AT88SC1608 function: Compute_Challenge

Prototype

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

Function

Compute the host challenge for AT88SC1608 card.

Parameters

char *Q0: input, the pointer to the buffer holding the host random number.
char *GC: input, the pointer to the buffer holding the secret seed.
char *Ci: input, the pointer to the buffer holding the card random number.
char *Q1: output, the pointer to the buffer holding the result challenge Q1.
char *Q2: output, the pointer to the buffer holding the result challenge Q2.

Return values

See list of return values.

Remarks

This function computes the two challenges required for the 2-direction authentication. Q1 is for authentication of the reader, Q2 is for authentication of the card. Before calling this function, the caller must allocate enough buffer for Q1 and Q2, the buffer size must be not less than 8. This function does not check the validity of the pointers.

This function is the same for both AT88SC1608 and AT88SC153.

See also

Init_1608_Authentication, Verify_1608_Authentication

AT88SC153 function: Write_153

Prototype

int WINAPI Write_153(int ZoneNumber, int Address, int NOB, char *Data);

Function

Write data bytes to AT88SC153 card.

Parameters

int ZoneNumber: 0-2 = User Zone 0-2, 3 = Configuration zone
int Address: the start byte address in the zone, range 0 to 63
int NOB: the number of byte to be written
char *Data: the pointer to the buffer holding the bytes to be written to the card

Return values

See list of return values.

Remarks

This function write data bytes to the card. Before apply this function, the caller must issue Password Verification and/or Authentication Commands to the card.

See also

Read_153

AT88SC153 function: Read_153

Prototype

  int WINAPI Read_153(int ZoneNumber, int Address, int NOB, char *Data);

Function

  Read bytes of AT88SC153 card.

Parameters

  int ZoneNumber: 0-2 = User Zone 0-2, 3 = Configuration zone
  int Address: the start byte address in the zone, range 0 to 63
  int NOB: the number of byte to be read. Must not exceed 16.
  char *Data: the pointer to the buffer holding the bytes read back from the card

Return values

  See list of return values.

Remarks

  This function reads the data byte of AT88SC153 card. Before this read function, the calller must issue Password Verification and/or Authentication commands to the card. Also before calling this function, the caller must make sure the parameter is valid. The char *Data buffer is allocated by the caller and the buffer size must be bigger than NOB. This function make no validity check of the parameter values, so invalid parameter may cause unpredictable results.

See also

  Write_153

AT88SC153 function: Verify_Password_153

Prototype

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

Function

Verify the read/write password of AT88SC153 card.

Parameters

int ReadWrite: 0 = Write password, 1 = Read password
int SetNumber: Password set number, 0 or 1
char *Password: Pointer to the 3-byte password to be verified

Return values

See list of return values.

Remarks

This function verifies a Read or Write password. A valid verification of the password erases the attempts counter. After this function the caller must read the attempts counter to see if the password verification if successful.

See also

Read_153

AT88SC153 function: InitializeAuthentication_153

Prototype

int WINAPI InitializeAuthentication_153(char *Q);

Function

Initialize the authentication process of AT88SC153 card.

Parameters

char *Q: this is the pointer to the buffer holding the 8-byte host random number.

Return values

See list of return values.

Remarks

This function sets up the authentication process of the AT88SC153 card. The caller must generate a random number as the input of this function.

See also

VerifyAuthentication_153

AT88SC153 function: VerifyAuthentication_153

Prototype

int WINAPI VerifyAuthentication_153(char *CH);

Function

Execute the ‘Verify Authentication’ command of the AT88SC153 card.

Parameters

char *CH: this is the pointer to the buffer holding the 8-byte host challenge. This challenge is computed by the reader.

Return values

See list of return values.

Remarks

After the initialize authentication command, both the reader and the card computed their result of the challenge. The reader must issue this command to let the card authenticate the reader.

See also

InitializeAuthentication_153, Compute_Challenge

AT88SC153 function: Compute_Challenge

Prototype

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

Function

Compute the host challenge for AT88SC153 card.

Parameters

char *Q0: input, the pointer to the buffer holding the host random number.
char *GC: input, the pointer to the buffer holding the secret seed.
char *Ci: input, the pointer to the buffer holding the card random number.
char *Q1: output, the pointer to the buffer holding the result challenge Q1.
char *Q2: output, the pointer to the buffer holding the result challenge Q2.

Return values

See list of return values.

Remarks

This function computes the two challenges required for the 2-direction authentication. Q1 is for authentication of the reader, Q2 is for authentication of the card. Before calling this function, the caller must allocate enough buffer for Q1 and Q2, the buffer size must be not less than 8. This function does not check the validity of the pointers.

This function is the same for both AT88SC153 and AT88SC1608.

See also

InitializeAuthentication_153, VerifyAuthentication_153

AT88SC153 function: ReadFuse_153

Prototype

int WINAPI ReadFuse_153(char *Fuse);

Function

Read the fuses of the AT88SC153 card.

Parameters

char *Fuse: this is the pointer to the byte holding the values of the fuses read back:

Bit0: FAB fuse, 1 = not blown, 0 = blown
Bit1: CMA fuse, 1 = not blown, 0 = blown
Bit2: PER fuse, 1 = not blown, 0 = blown

Return values

See list of return values.

Remarks

This function reads back the fuses of the 153 card.

See also

BlowFuse_153

AT88SC153 function: BlowFuse_153

Prototype

int WINAPI BlowFuse_153(int FuseNumber);

Function

Blow the fuse of AT88SC153 card.

Parameters

int FuseNumber:

0 = FAB fuse
1 = CMA fuse
2 = PER fuse

Return values

See list of return values.

Remarks

The FAB fuse is blown by the chip manufacturer.
The CMA fuse is blown by the card manufacturer.
The PER fuse is blown by the card issuer.

See also

ReadFuse_153

The example card read/write procedures

```
//1.Initialization

    int PortHandle;
    OpenPort( "COM1", &PortHandle );
    PowerOn( );
    SelectPort( PortHandle );

//2. Check the card presence

    Wait until CardPresent == 1;

//3. Power the card and get card ATR and card type

    GetCardType( ATRBuffer );
    Compare the return value or
    Compare the ATRBuffer with the known card ATR to see if it's the desired card.

//4. Verify the password (PSC, or SC)

    Verify_102_SC( SCBuffer );
    Read back the error counter or other information to see if the verification is
successful.

//5.Read/Write card

    Read_102_Byte_Array( ..);
    Write_102_Byte_Array( ..);

//6.Notify user the operation is over.

    If another card need to be processed goto step 2.

//7.Close the reader

      PowerOff();
      ClosePort();
```