EWTJ-680
API Specification

East Wind Technologies, Inc.
7F-3, No. 390, Sec. 1, Fu-Hsing S. Rd.,
Taipei, Taiwan, R.O.C.
TEL : 886-2-77128686
FAX : 886-2-77128688

1. MIFARE Card Concept

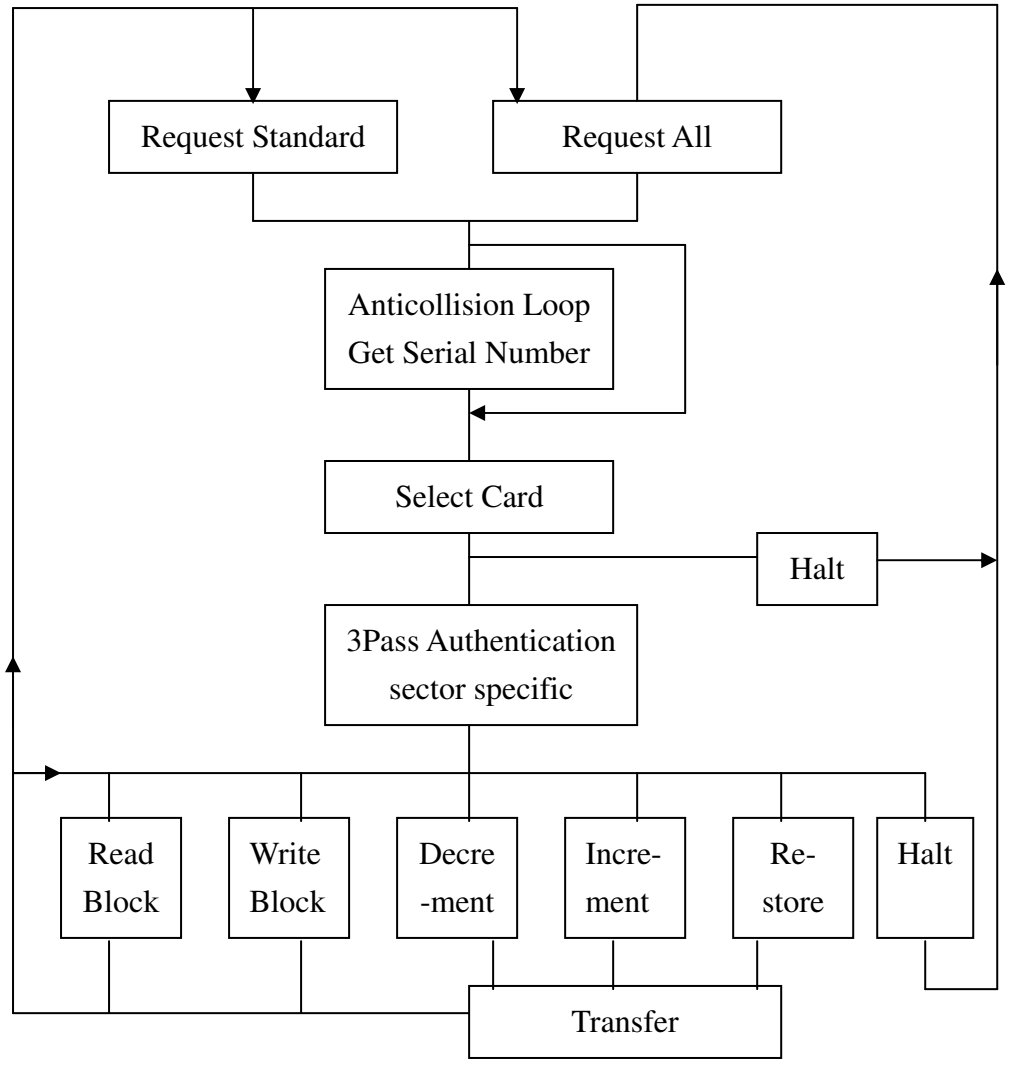
- **General Description**

  The MIFARE RWD Core Module is the kernel of a MIFARE read write unit. It concerns all the necessary front-end functions to access MIFARE cards. Its versatility allows a flexible and efficient application in different system devices such as bus terminals, metro gate controllers, handheld devices, booking office computers or even PCs.

  In this document the user will find a description of how to use the API which has been implemented to work on a PC. With these functions the card can be easily accessed.

- **Communication scheme RWD⟷Card**

Transaction Sequence



- **Security**

  To provide a very high security level three pass authentication (according to ISO 9798-2) and encryption based on a stream cipher algorithm with random generator, serial number and 48 Bit keys are integrated in the RWD's Interface ASIC and the cards. Keys in the cards are read protected but can be altered, provided one knows the actual key. This gives the possibility to any system integrator who knows the transport key of any card to program his own secret keys. Splitting the card's memory into several sections with separate access keys makes the system open for multi-functionality (same cards for different applications).

  Since there are two different keys (Key A and Key B) per sector available with corresponding access conditions available MIFARE 1 S50 card IC provides the possibility of a system with key hierarchy. Key A for example can be used for protecting the decrement function and key B the usually more sensible increment function.

- **Memory Organization and Access Conditions**

  The MIFARE 1 IC has integrated a 8192 Bit EEPROM which is split into 16 sectors with 4 blocks. One block consists of 16 bytes (1 Byte = 8 Bit).

  Memory Organization:

Sector	Block 0	Block 1	Block 2	Block 3
0	Manufacturer Code	Data Block	Data Block	Sector Trailer
1	Data Block	Data Block	Data Block	Sector Trailer
2	Data Block	Data Block	Data Block	Sector Trailer
3	Data Block	Data Block	Data Block	Sector Trailer
4	Data Block	Data Block	Data Block	Sector Trailer
5	Data Block	Data Block	Data Block	Sector Trailer
6	Data Block	Data Block	Data Block	Sector Trailer
7	Data Block	Data Block	Data Block	Sector Trailer
8	Data Block	Data Block	Data Block	Sector Trailer
9	Data Block	Data Block	Data Block	Sector Trailer
10	Data Block	Data Block	Data Block	Sector Trailer
11	Data Block	Data Block	Data Block	Sector Trailer
12	Data Block	Data Block	Data Block	Sector Trailer
13	Data Block	Data Block	Data Block	Sector Trailer
14	Data Block	Data Block	Data Block	Sector Trailer
15	Data Block	Data Block	Data Block	Sector Trailer

  - ✓ **Sector Trailer (Block 3)**

    The fourth block of any sector is the Sector Trailer. The Sector Trailer contains access Key A,

an optional Key B and the access conditions for the four blocks of that sector. If Key B is not needed, the last 6 Bytes of block 3 can be used as data bytes.

Note: Regarding to the detailed information of access conditions, please refer to mifare standard card IC functional specification provided by Philips Semiconductors.

✓ **Manufacturer Code (Block 0 of Sector 0)**

The first block of the memory is reserved for manufacturer data like 32 bit serial number. This is a read only block. In many documents it is named "Block 0".

✓ **Data Block (Block 0 to 3 except "Block 0")**

Access conditions for the Data Blocks are defined in the Sector Trailers. According to these conditions data can be read, written, incremented, decremented, transferred or restored either with Key A, Key B or never.

- **Read/write blocks**

  Are used to read and write general 16 bytes of data.

- **Value blocks**

  Are used for electronic purse functions (read, increment, decrement, transfer, restore). The maximum size of a value is 4 byte including sign bit, even when a complete 16 byte block has to be reserved. To provide error detection and correction capability, any value is stored 3 times into one value block. The remaining 4 bytes are reserved to some extent for check bits.

✓ **Key management and Multi-functionality**

The described memory organization makes it possible to appoint different sectors to different applications and to prevent data corruption by using application specific secret keys. Keys can only be altered by a RWD which has stored the actual Key A or Key B if this is allowed according to access conditions. Otherwise the actual key cannot be changed anymore.

Note:

Before the execution of a command the correct format of the Access Conditions is checked by the Card-IC. Thus, when programming the Sector Trailer the card needs to be fixed within the operating range of a RWD's antenna to prevent interruption of the write operation because any unsuccessful write operation may lead to blocking the whole sector.

2. Windows API

Subject

EWTJ-680 Windows API functions definition and description.

Contact point

Please contact us if you have any question regarding to this API.

email：info@ewt.com.tw

TEL：886-2-7712-8686

FAX：886-2-7712-8688

Related file names and description:

-- 680API.DLL API DLL file for Windows program

-- 680API.LIB API libraries file for C/C++ program

-- EWTJ-680 API.PDF API User Manual

-- 680TEST.EXE API Test Program

-- EWTJ-680 SPEC.PDF Specification / Communication Protocol Manual

EWTJ-680 API functions

1. **int E680_Open_ComPort(int port);**
2. **int E680_Close_ComPort(void);**
3. **int E680_ledset(char state);**
4. **int E680_buzzerset(char time);**
5. **int E680_eeprom_read(int startadd, char length, char* data);**
6. **int E680_eeprom_read_by_string(int startadd, char length, char* data);**
7. **int E680_eeprom_write(int startadd, char length, char* data);**
8. **int E680_eeprom_write_by_string(int startadd, char length, char* data);**
9. **int E680_reset_SAM(int* infolen, char* info);**
10. **int E680_reset_SAM_by_string(int* infolen, char* info);**
11. **int E680_send_APDU_to_SAM(int apdulen, char* apdu, int* outlen, char* response);**
12. **int E680_send_APDU_to_SAM_by_string(int apdulen, char* apdu, int* outlen, char* response);**
13. **int E680_Set_Working_Mode(char antenna, char autoq);**
14. **int E680_Set_Readcard_Mode(char cardtype);**

ISO14443A API functions

15. **int E680_Request_CardSN(char* serial);**
16. **int E680_loadkey(char keyid, char* keyvalue);**
17. **int E680_block_read(char block, char keytype, char keyid, char* data);**
18. **int E680_block_write(char block, char keytype, char keyid, char* data);**
19. **int E680_sector_read(char sector, char keytype, char keyid, char* data);**
20. **int E680_initvalue(char block, char keytype, char keyid, long value);**
21. **int E680_readvalue(char block, char keytype, char keyid, long* value);**
22. **int E680_increment(char block, char keytype, char keyid, unsigned long value);**
23. **int E680_decrement(char block, char keytype, char keyid, unsigned long value);**
24. **int E680_UL_read(char page, char* data);**
25. **int E680_UL_write(char page, char* data);**
26. **int E680_loadkey_by_string(char keyid, char* keyvalue);**
27. **int E680_block_read_by_string(char block, char keytype, char keyid, char* data);**
28. **int E680_block_write_by_string(char block, char keytype, char keyid, char* data);**
29. **int E680_sector_read_by_string(char sector, char keytype, char keyid, char* data);**
30. **int E680_UL_read_by_string(char page, char* data);**
31. **int E680_UL_write_by_string(char page, char* data);**

ISO14443B API functions

32. **int E680_14443B_card_reset(char* serial);**
33. **int E680_14443B_card_halt(char* pupi);**
34. **int E680_SR_initiate(char* chipid);**
35. **int E680_SR_select(char chipid);**
36. **int E680_SR_completion();**
37. **int E680_SR_pcall16(char* chipid);**
38. **int E680_SR_slot_marker(char slot, char* chipid);**
39. **int E680_SR_reset_to_inventory();**
40. **int E680_SR_authenticate(char* rnd, char* data);**
41. **int E680_SR_get_uid(char* uid);**
42. **int E680_SR_read_block(char addr, char* data);**
43. **int E680_SR_write_block(char addr, char* data);**

ISO15693 API functions

44. **int E680_15693_Inventory(char* serial);**
45. **int E680_15693_stay_quiet();**
46. **int E680_15693_get_tag_information(char* data);**
47. **int E680_15693_reset_quiet(char* uid);**
48. **int E680_15693_read_blocks(char start, char blocks, char* data);**
49. **int E680_15693_write_blocks(char start, char blocks, char* data);**
50. **int E680_15693_lock_block(char block);**
51. **int E680_15693_write_AFI(char afi);**
52. **int E680_15693_lock_AFI();**
53. **int E680_15693_write_DSFID(char dsfid);**
54. **int E680_15693_lock_DSFID();**
55. **int E680_15693_get_blocks_security(char start, char blocks, char* data);**
56. **int E680_15693_read_blocks_by_string(char start, char blocks, char* data);**
57. **int E680_15693_write_blocks_by_string(char start, char blocks, char* data);**

1. Opening of the Serial Port

Function name **E680_Open_ComPort(int port)**

Description

This function should be called at first when you power on the EWTJ-680. Then you can control EWTJ-680 by the other function call. The function of E680_Open_ComPort include:

(1) To specify comm. port parameters

(2) To initialize the EWTJ-680

Parameters

port: 1 ==> COM1 port is used

2 ==> COM2 port is used

3….

Return code

0 ==> FAIL

1 ==> SUCCESS

Example

C

```
int     port;
port = 1; // COM1
if   (E680_Open_ComPort(port) == 0)
     printf("E680_Open_ComPort failed");
```

Visual Basic

```
Declare Function E680_Open_ComPort Lib "680api" (ByVal
port As Byte) As Long
```

2. Closing of the Serial Port

Function name **E680_Close_ComPort(void)**

Description To close the comm. port. It should be called before you close the application.

Parameters None

Return code 0 ==> FAIL

1 ==> SUCCESS

Example

C

```
E680_Close_ComPort();
```

Visual Basic

```
Declare Function E680_Close_ComPort Lib "680api" () As
Long
```

3. LED set

Function name E680_ledset(char state)

Description This function sets LED on or off.

Parameters state: 0 ==> off

1 ==> on

Return code 0 ==> FAIL

1 ==> SUCCESS

Example

C

```
char    state;
state = 1; // ON
if   (E680_ledset(state) == 0)
     printf("E680_ledset failed");
```

Visual Basic

```
Declare Function E680_ledset Lib "680api" (ByVal state As
Byte) As Long
```

4. Buzzer set

Function name E680_buzzerset(char time)

Description This function sets buzzer beeping.

Parameters time: unit is 10mS. If the time is 10, then the beep time is 100mS.

Return code 0 ==> FAIL

1 ==> SUCCESS

Example

C

```
char    time;
time = 10; // ON
if   (E680_buzzerset(time) == 0)
     printf("E680_buzzerset failed");
```

Visual Basic

```
Declare Function E680_buzzerset Lib "680api" (ByVal time
As Byte) As Long
```

5. EEPROM Read

Function name **E680_eeprom_read(int startadd, char length, char* data)**

Description This function reads EEPROM.

Parameters

startadd: start address to be read.

length: number of bytes to be read, max. 16 bytes.

data: Pointer to the data read from the EEPROM.

Return code

0 ==> FAIL

1 ==> SUCCESS

Example

C

```
Int startadd;
char length;
char data[17];
startdd = 15;
length = 16;
if   (E680_eeprom_read(startadd, length, data) == 0)
     printf("E680_eeprom_read failed");
```

Visual Basic

```
Declare Function E680_eeprom_read Lib "680api" (ByVal
startadd As Byte, ByVal length As Byte, ByVal data As
String) As Long
```

6. EEPROM Read by string

Function name **E680_eeprom_read_by_string(int startadd, char length, char* data)**

Description This function reads EEPROM. The data to be read is double length long in hex format.

Parameters

startadd: start address to be read.

length: number of bytes to be read, max. 16 bytes.

data: Pointer to the data read from the EEPROM. This is a double length bytes hex value character string.

Return code

0 ==> FAIL

1 ==> SUCCESS

Example

C

```
Int startadd;
char length;
char data[33];
startdd = 15;
length = 16;
if (E680_eeprom_read_by_string(startadd, length, data) == 0)
    printf("E680_eeprom_read failed");
```

Visual Basic

```
Declare Function E680_eeprom_read_by_string Lib "680api" (ByVal startadd As Byte, ByVal length As Byte, ByVal data As String) As Long
```

7. EEPROM Write

Function name **E680_eeprom_write(int startadd, char length, char* data)**

Description This function writes data to EEPROM.

Parameters

startadd: start address to be written.

length: number of bytes to be written, max. 16 bytes.

data: Pointer to the data to be written to the EEPROM.

Return code

0 ==> FAIL

1 ==> SUCCESS

Example

C

```
Int startadd;
char length;
char data[17];
startdd = 15;
length = 16;
memcpy(data, “0123456789ABCDEF”, 16);
if   (E680_eeprom_write(startadd, length, data) == 0)
     printf(“E680_eeprom_write failed”);
```

Visual Basic

```
Declare Function E680_eeprom_write Lib "680api" (ByVal
startadd As Byte, ByVal length As Byte, ByVal data As
String) As Long
```

8. EEPROM Write by string

Function name **E680_eeprom_write_by_string(int startadd, char length, char* data)**

Description This function writes data to EEPROM. The data to be written is double length long in hex format.

Parameters

startadd: start address to be written.

length: number of bytes to be written, max. 16 bytes.

data: Pointer to the data to be written to the EEPROM. This is a double length bytes hex value character string.

Return code

0 ==> FAIL

1 ==> SUCCESS

Example

C

```
Int startadd;
char length;
char data[33];
startdd = 15;
length = 16;
memcpy(data,
“0123456789ABCDEF0123456789ABCDEF”, 32);
if  (E680_eeprom_write_by_string(startadd, length, data)
== 0)
    printf(“E680_eeprom_write failed”);
```

Visual Basic

```
Declare Function E680_eeprom_write_by_string Lib
"680api" (ByVal startadd As Byte, ByVal length As Byte,
ByVal data As String) As Long
```

9. Reset SAM

Function name **E680_reset_SAM(int* infolen, char* info)**

Description This function resets the SAM.

Parameters infolen: the length of card reset information.

info: card reset information.

Return code 0 ==> FAIL

1 ==> SUCCESS

Example

C

```
int infolen;
char info[300];
if  (E680_reset_SAM(&infolen, info) == 0)
    printf("E680_reset_SAM failed");
```

Visual Basic

```
Declare Function E680_reset_SAM Lib "680api" (ByRef
infolen as Long, ByVal info As String) As Long
```

10. Reset SAM by string

Function name **E680_reset_SAM_by_string(int* infolen, char* info)**

Description This function resets the SAM.

Parameters infolen: the length of card reset information.

info: card reset information. This is a double infolen bytes hex value character string.

Return code 0 ==> FAIL

1 ==> SUCCESS

Example

C

```
int infolen;
char info[600];
if  (E680_reset_SAM_by_string(&infolen, info) == 0)
    printf("E680_reset_SAM failed");
```

Visual Basic

```
Declare Function E680_reset_SAM_by_string Lib "680api"
(ByRef infolen as Long, ByVal info As String) As Long
```

11. Send APDU to SAM

Function name

E680_send_APDU_to_SAM(int apdulen, char* apdu, int* outlen, char* response)

Description

This function send APDU to SAM and then get back the return response if any.

Parameters

apdulen: the length of APDU to be sent to SAM

apdu: the APDU data to be sent to SAM.

outlen: the length of response data

response: the response data returned from SAM

Return code

0 ==> FAIL

1 ==> SUCCESS

Example

C

```
int apdulen, outlen;
char apdu[300], response[300];
apdulen = 10;
apdu[0] = 0x00;
apdu[1] = 0x20;
apdu[2] = 0x00;
apdu[3] = 0x01;
apdu[4] = 0x08;
apdu[5] = 0x00;
apdu[6] = 0x00;
apdu[7] = 0x00;
apdu[8] = 0x00;
apdu[9] = 0x00;
```

```
if    (E680_send_APDU_to_SAM(apdulen, apdu, &outlen,
response) == 0)
      printf("E680_send_APDU_to_SAM failed");
```

Visual Basic

```
Declare Function E680_send_APDU_to_SAM Lib "680api"
(ByVal apdulen As Long, ByVal apdu As String, ByRef
outlen as Long, ByVal response as String) As Long
```

12. Send APDU to SAM by string

Function name **E680_send_APDU_to_SAM_by_string(int apdulen, char* apdu, int* outlen, char* response)**

Description This function sends APDU to SAM and then getS back the return response if any.

Parameters

apdulen: the length of APDU to be sent to SAM

apdu: the APDU data to be sent to SAM. This is a hex value character string.

outlen: the length of response data

response: the response data returned from SAM. This is a double outlen bytes hex value character string.

Return code

0 ==> FAIL

1 ==> SUCCESS

Example

C

```
int apdulen, outlen;
char apdu[600], response[600];
apdulen = 10;
memcpy(apdu, “00200001080000000000”, 20);
if (E680_send_APDU_to_SAM_by_string(apdulen, apdu, &outlen, response) == 0)
    printf(“E680_send_APDU_to_SAM_by_string failed”);
```

Visual Basic

```
Declare Function E680_send_APDU_to_SAM_by_string Lib "680api" (ByVal apdulen As Long, ByVal apdu As String, ByRef outlen as Long, ByVal response as String) As Long
```

13. Module working mode set

Function name **E680_Set_Working_Mode(char antenna, char autoq)**

Description This function sets working mode of the module.

Parameters antenna: antenna status, 0: OFF, 1: ON

autoq: Auto requet, 0: OFF, 1: ON

Return code 0 ==> FAIL

1 ==> SUCCESS

Example

C

```
char     antenna, autoq;
antenna = 1; // ON
autoq = 1; // ON
if    (E680_Set_Working_Mode(antenna, qutoq) == 0)
      printf("E680_Set_Working_Mode failed");
```

Visual Basic

```
Declare Function E680_Set_Working_Mode Lib "680api"
(ByVal antenna As Byte, ByVal autoq As Byte) As Long
```

14. Set card operate model

Function name **E680_Set_Readcard_Mode(char cardtype)**

Description This function sets card operate model.

Parameters cardtype: 0: ISO14443 type A

1: ISO14443 type B

2: ISO15693

Return code 0 ==> FAIL

1 ==> SUCCESS

Example

C

```
char     cardtype;
cardtype = 2; // ISO15693
if   (E680_Set_Readcard_Mode(cardtype) == 0)
     printf("E680_Set_Readcard_Mode failed");
```

Visual Basic

```
Declare Function E680_Set_Readcard_Mode Lib "680api"
(ByVal cardtype As Byte) As Long
```

15. Request ISO14443 type A Card

Function name **E680_Request_CardSN(char* serial)**

Description This high level function is equivalent to Request, Anticoll and Select command to access card.

Parameters serial: The serial number of the card. It is interpreted as a 8 bytes character string for Mifare 1K (S50) and 14 bytes for UltraLight.

Return code 0 ==> FAIL

1 ==> SUCCESS

Example

C

```
char serial[20];
if   (E680_Request_CardSN(serial) == 0)
     printf("E680_Request_CardSN failed");
```

Visual Basic

```
Declare Function E680_Request_CardSN Lib "680api"
(ByVal serial As String) As Long
```

16. Load Key

Function name **E680_loadkey(char keyid, char* keyvalue)**

Description This function loads a new key into the key EEPROM of the MIFARE Reader.

Parameters

keyid: This is the keyid (0… 31) in which the key is loaded.

keyvalue: This 6 byte value data (Key A or Key B) contain the new key written into the EEPROM.

Return code

0 ==> FAIL

1 ==> SUCCESS

Example

C

```
char keyid;
char keyvalue[10];
keyid = 15;
keyvalue[0] = 0xA0;
keyvalue[1] = 0xA1;
keyvalue[2] = 0xA2;
keyvalue[3] = 0xA3;
keyvalue[4] = 0xA4;
keyvalue[5] = 0xA5;

if   (E680_loadkey(keyid, keyvalue) == 0)
     printf("E680_loadkey failed");
```

Visual Basic

```
Declare Function E680_loadkey Lib "680api" (ByVal keyid
As Byte, ByVal KeyValue As String) As Long
```

17. Block Read

Function name **E680_block_read(char block, char keytype, char keyid, char* data)**

Description This function reads one block of 16 bytes from a selected and authenticated card sector.

Parameters

block: This is the block (0… 63 for Mifare 1K or 0…255 for Mifare 4K) from which the data is read.

keytype: 0 ==> Authenticate with key A

1 ==> Authenticate with key B

keyid: This is the keyid (0… 31) in which the key will be verified.

data: Pointer to the data read from the card.

Return code

0 ==> FAIL

1 ==> SUCCESS

Example

C

```
char block;
char keytype;
char keyid;
char data[17];
block = 1;
keytype = 0;
keyid = 15;
if   (E680_block_read(block, keytype, keyid, data) == 0)
     printf("E680_block_read failed");
```

Visual Basic

```
Declare Function E680_block_read Lib "680api" (ByVal
block As Byte, ByVal keytype As Byte, ByVal keyid As Byte,
ByVal data As String) As Long
```

18. Write Block

Function name **E680_block_write(char block, char keytype, char keyid, char* data)**

Description This function writes one block of 16 bytes to a selected and authenticated card sector.

Parameters

block: This is the block (0… 63 for Mifare 1K or 0…255 for Mifare 4K) to which the data is written on the card.

keytype: 0 ==> Authenticate with key A

3 ==> Authenticate with key B

keyid: This is the keyid (0… 31) in which the key will be verified.

data: Pointer to the data written on the card.

Return code

0 ==> FAIL

1 ==> SUCCESS

Example

C

```
char block;
char keytype;
char keyid;
char data[17];
block = 1;
keytype = 0;
keyid = 15;
memcpy(data, “0123456789ABCDEF”, 16);
if   (E680_block_write(block, keytype, keyid, data) == 0)
     printf(“E680_block_write failed”);
```

Visual Basic

```
Declare Function E680_block_write Lib "680api" (ByVal
block As byte, ByVal keytype As byte, ByVal keyid As byte,
ByVal data As String) As Long
```

19. Sector Read

Function name **E680_sector_read(char sector, char keytype, char keyid, char* data)**

Description This function reads one sector (64 bytes, four blocks) from a selected and authenticated card sector.

Parameters

sector: This is the sector (0… 15 for Mifare 1K or 0…63 for Mifare 4K) from which the data is read.

keytype: 0 ==> Authenticate with key A

1 ==> Authenticate with key B

keyid: This is the keyid (0… 31) in which the key will be verified.

data: Pointer to the data read from the card.

Return code

0 ==> FAIL

1 ==> SUCCESS

Example

C

```
char sector;
char keytype;
char keyid;
char data[65];
sector = 1;
keytype = 0;
keyid = 15;
if   (E680_sector_read(sector, keytype, keyid, data) == 0)
     printf("E680_sector_read failed");
```

Visual Basic

```
Declare Function E680_sector_read Lib "680api" (ByVal
sector As Byte, ByVal keytype As Byte, ByVal keyid As
Byte, ByVal data As String) As Long
```

20. Initial Value Block

Function name **E680_initvalue(char block, char keytype, char keyid, long value)**

Description This function initials a value block with a specified value.

Parameters

block: This is the block (0… 63 for Mifare 1K or 0…255 for Mifare 4K) to which the value will be initialized.

keytype: 0 ==> Authenticate with key A

1 ==> Authenticate with key B

keyid: This is the keyid (0… 31) in which the key will be verified.

value: The initial value.

Return code

0 ==> FAIL

1 ==> SUCCESS

Example

C

```
char block;
char keytype;
char keyid;
long value;
block = 1;
keytype = 0;
keyid = 15;
value = 0;
if   (E680_initvalue(block, keytype, keyid, value) == 0)
     printf("E680_initvalue failed");
```

Visual Basic

```
Declare Function E680_initvalue Lib "680api" (ByVal block
As Byte, ByVal keytype As Byte, ByVal keyid As Byte,
ByVal value As Long) As Long
```

21. Read Value Block

Function name **E680_readvalue(char block, char keytype, char keyid, long *value)**

Description This function reads the value from a value block.

Parameters

block: This is the block (0… 63 for Mifare 1K or 0…255 for Mifare 4K) from which the value will be read.

keytype: 0 ==> Authenticate with key A

1 ==> Authenticate with key B

keyid: This is the keyid (0… 31) in which the key will be verified.

value: The value read from the value block.

Return code

0 ==> FAIL

1 ==> SUCCESS

Example

C

```
char block;
char keytype;
char keyid;
long value;
block = 1;
keytype = 0;
keyid = 15;
if   (E680_readvalue(block, keytype, keyid, &value) == 0)
     printf("E680_readvalue failed");
```

Visual Basic

```
Declare Function E680_readvalue Lib "680api" (ByVal block
As Byte, ByVal keytype As Byte, ByVal keyid As Byte,
ByRef value As Long) As Long
```

22. Increment Value

Function name **E680_increment(char block, char keytype, char keyid, unsigned long value)**

Description This function reads the accessed value block, checks the data structure, increases the value to the block.

Parameters

block: This is the block (0… 63 for Mifare 1K or 0…255 for Mifare 4K) to which the value will be added.

keytype: 0 ==> Authenticate with key A

1 ==> Authenticate with key B

keyid: This is the keyid (0… 31) in which the key will be verified.

value: The value added to the value block.

Return code

0 ==> FAIL

1 ==> SUCCESS

Example

C

```
char block, keytype, keyid;
unsigned long value;
block = 61;
keybypte = 0;
keyid = 15;
value = 100;
if   (E680_increment(block, keytype, keyid, value) == 0)
     printf("E680_increment failed.");
```

Visual Basic

```
Declare Function E680_increment Lib "680api" (ByVal
block As Byte, ByVal keytype As Byte, ByVal keyid As Byte,
ByVal value As Long) As Long
```

23. Decrement Block

Function name **E680_decrement(char block, char keytype, char keyid, unsigned long value)**

Description This function reads the accessed value block, checks the data structure, decreases the contents of the value block by the value and stores the result into the value block.

Parameters

block: This is the block (0… 63 for Mifare 1K or 0…255 for Mifare 4K) from which the value will be decreased.

keytype: 0 ==> Authenticate with key A

1 ==> Authenticate with key B

keyid: This is the keyid (0… 31) in which the key will be verified.

value: The decreasing value.

Return code

0 ==> FAIL

1 ==> SUCCESS

Example

C

```
char block, keytype, keyid;
unsigned long value;
block = 61;
keytype = 0;
keyid = 15;
value = 100;
if   (E680_decrement(block, keytype, keyid, value) == 0)
     printf("E680_decrement failed.");
```

Visual Basic

```
Declare Function E680_decrement Lib "680api" (ByVal
block As Byte, ByVal keytype As Byte, ByVal keyid As Byte,
ByVal value As Long) As Long
```

24. UltraLight card read

Function name **E680_UL_read(char page, char* data)**

Description This function reads 16 bytes (4 pages) from a selected page.

Parameters

page: This is the start page from which the 16-bytes data is read.

data: Pointer to the data read from the UltraLight card.

Return code

0 ==> FAIL

1 ==> SUCCESS

Example

C

```
char page;
char data[17];
page = 1;
if   (E680_UL_read(page, data) == 0)
     printf("E680_UL_read failed");
```

Visual Basic

```
Declare Function E680_UL_read Lib "680api" (ByVal page
As Byte, ByVal data As String) As Long
```

25. UltraLight card write

Function name **E680_UL_write(char page, char* data)**

Description This function writes 4 bytes (1 pages) to a selected page.

Parameters

Page: This is the page to which the 4-bytes data is written.

data: Pointer to the data to be written to the UltraLight card.

Return code

0 ==> FAIL

1 ==> SUCCESS

Example

C

```
char page;
char data[5];
page = 12;
memcpy(data, “0123”, 4);
if   (E680_UL_write(page, data) == 0)
     printf(“E680_UL_write failed”);
```

Visual Basic

```
Declare Function E680_UL_write Lib "680api" (ByVal page
As Byte, ByVal data As String) As Long
```

26. Load Key by string

Function name **E680_loadkey_by_string(char keyid, char* keyvalue)**

Description This function loads a new key into the key EEPROM of the MIFARE Reader. The keyvalue is 12-byte long in hex format.

Parameters

keyid: This is the keyid (0… 31) in which the key is loaded.

keyvalue: This is a 12 bytes hex value character string.

Return code

0 ==> FAIL

1 ==> SUCCESS

Example

C

```
char keyid;
char keyvalue[20];
keyid = 15;
memcpy(keyvalue, “A0A1A2A3A4A5”, 12);
if   (E680_loadkey_by_string(keyid, keyvalue) == 0)
     printf(“E680_loadkey_by_string failed”);
```

Visual Basic

```
Declare Function E680_loadkey_by_string Lib "680api"
(ByVal keyid As Byte, ByVal KeyValue As String) As Long
```

27. Block Read by string

Function name **E680_block_read_by_string(char block, char keytype, char keyid, char* data)**

Description This function reads one block of 16 bytes from a selected and authenticated card sector. The data to be read is 32-byte long in hex format.

Parameters

block: This is the block (0… 63 for Mifare 1K or 0…255 for Mifare 4K) from which the data is read.

keytype: 0 ==> Authenticate with key A

1 ==> Authenticate with key B

keyid: This is the keyid (0… 31) in which the key will be verified.

data: Pointer to the data read from the card. This is a 32 bytes hex value character string.

Return code

0 ==> FAIL

1 ==> SUCCESS

Example

C

```
char block;
char keytype;
char keyid;
char data[33];
block = 1;
keytype = 0;
keyid = 15;
if  (E680_block_read_by_string(block, keytype, keyid, data)
```

```
== 0)
    printf("E680_block_read_by_string failed");
```

Visual Basic

```
Declare Function E680_block_read_by_string Lib "680api"
(ByVal block As Byte, ByVal keytype As Byte, ByVal keyid
As Byte, ByVal data As String) As Long
```

28. Write Block by string

Function name **E680_block_write_by_string(char block, char keytype, char keyid, char* data)**

Description This function writes one block of 32 bytes hexi data to a selected and authenticated card sector.

Parameters

block: This is the block (0… 63 for Mifare 1K or 0…255 for Mifare 4K) to which the data is written on the card.

keytype: 0 ==> Authenticate with key A

3 ==> Authenticate with key B

keyid: This is the keyid (0… 31) in which the key will be verified.

data: Pointer to the data written on the card. The data to be written is 32-byte long in hex format.

Return code

0 ==> FAIL

1 ==> SUCCESS

Example

C

```
char block;
char keytype;
char keyid;
char data[33];
block = 1;
keytype = 0;
keyid = 15;
memcpy(data,
"0123456789ABCDEF0123456789ABCDEF", 32);
if  (E680_block_write_by_string(block, keytype, keyid,
```

```
data) == 0)
    printf("E680_block_write_by_string failed");
```

Visual Basic

```
Declare Function E680_block_write_by_string Lib "680api"
(ByVal block As byte, ByVal keytype As byte, ByVal keyid
As byte, ByVal data As String) As Long
```

29. Sector Read by string

Function name **E680_sector_read_by_string(char sector, char keytype, char keyid, char* data)**

Description This function reads one sector (64 bytes, four blocks) from a selected and authenticated card sector. The data to be read is 128-byte long in hex format.

Parameters

sector: This is the sector (0… 15 for Mifare 1K or 0…63 for Mifare 4K) from which the data is read.

keytype: 0 ==> Authenticate with key A

1 ==> Authenticate with key B

keyid: This is the keyid (0… 31) in which the key will be verified.

data: Pointer to the data read from the card. This is a 128 bytes hex value character string.

Return code

0 ==> FAIL

1 ==> SUCCESS

Example

C

```
char sector;
char keytype;
char keyid;
char data[129];
sector = 1;
keytype = 0;
keyid = 15;
if  (E680_sector_read_by_string(sector, keytype, keyid,
```

```
data) == 0)
    printf("E680_sector_read_by_string failed");
```

Visual Basic

```
Declare Function E680_sector_read_by_string Lib "680api"
(ByVal sector As Byte, ByVal keytype As Byte, ByVal keyid
As Byte, ByVal data As String) As Long
```

30. UltraLight card read by string

Function name **E680_UL_read_by_string(char page, char* data)**

Description This function reads 16 bytes (4 pages) from a selected page. The data to be read is 32-byte long in hex format.

Parameters

page: This is the start page from which the 16-bytes data is read.

data: Pointer to the data read from the UltraLight card. This is a 32 bytes hex value character string.

Return code

0 ==> FAIL

1 ==> SUCCESS

Example

C

```
char page;
char data[33];
page = 1;
if   (E680_UL_read_by_string(page, data) == 0)
     printf("E680_UL_read failed");
```

Visual Basic

```
Declare Function E680_UL_read_by_string Lib "680api"
(ByVal page As Byte, ByVal data As String) As Long
```

31. UltraLight card write by string

Function name **E680_UL_write_by_string(char page, char* data)**

Description This function writes 4 bytes (1 pages) to a selected page. The data to be written is 8-byte long in hex format.

Parameters

Page: This is the page to which the 4-bytes data is written.

data: Pointer to the data to be written to the UltraLight card. This is a 8 bytes hex value character string.

Return code

0 ==> FAIL

1 ==> SUCCESS

Example

C

```
char page;
char data[9];
page = 12;
memcpy(data, “01234567”, 8);
if   (E680_UL_write_by_string(page, data) == 0)
     printf(“E680_UL_write_by_string failed”);
```

Visual Basic

```
Declare Function E680_UL_write_by_string Lib "680api"
(ByVal page As Byte, ByVal data As String) As Long
```

32. ISO14443 TYPE B card reset

Function name **E680_14443B_card_reset(char* serial)**

Description This function requests all type B card and get back the card reset information.

Parameters serial: 12 bytes card reset information. It is interpreted as a 24 bytes hexi-value string.

Return code 0 ==> FAIL

1 ==> SUCCESS

Example

C

```
char serial[25];
if   (E680_14443B_card_reset(serial) == 0)
     printf("E680_14443B_card_reset failed");
```

Visual Basic

```
Declare Function E680_14443B_card_reset Lib "680api"
(ByVal serial As String) As Long
```

33. ISO14443 type B card halt

Function name **E680_14443B_card_halt(char* pupi)**

Description This function halts the current card.

Parameters pupi: 4 bytes, PUPI of the card to halt.

Return code 0 ==> FAIL

1 ==> SUCCESS

Example

C

```
char pupi[10];
pupi[0] = 0x01;
pupi[1] = 0x23;
pupi[2] = 0x45;
pupi[3] = 0x67;
pupi[4] = 0x00;

if  (E680_14443B_card_halt(pupi) == 0)
    printf("E680_14443B_card_halt failed");
```

Visual Basic

```
Declare Function E680_14443B_card_halt Lib "680api"
(ByVal pupi As String) As Long
```

34. SR Initiate

Function name **E680_SR_initiate(char* chipid)**

Description This function initiates SR176 & SRIX4K cards.

Parameters chipid: 1 byte, Chip ID returned from chip.

Return code 0 ==> FAIL

1 ==> SUCCESS

Example

C

```
char chipid;
if   (E680_SR_initiate(&chipid) == 0)
     printf("E680_SR_initiate failed");
```

Visual Basic

```
Declare Function E680_SR_initiate Lib "680api" (ByVal
chipid As String) As Long
```

35. SR Select

Function name **E680_SR_select(char chipid)**

Description This function calls SR176 & SRIX4K select.

Parameters chipid: 1 byte, send to card

Return code 0 ==> FAIL

1 ==> SUCCESS

Example

C

```
char chipid;
if   (E680_SR_initiate(&chipid) == 0)
     printf("E680_SR_initiate failed");
if   (E680_SR_select(chipid) == 0)
     printf("E680_SR_select failed");
```

Visual Basic

```
Declare Function E680_SR_select Lib "680api" (ByVal
chipid As Byte) As Long
```

36. SR Completion

Function name **E680_SR_completion (void)**

Description SR176 & SRIX4K Completion

Parameters None

Return code 0 ==> FAIL

1 ==> SUCCESS

Example

C

```
if   (E680_SR_completion() == 0)
     printf("E680_SR_completion failed");
```

Visual Basic

```
Declare Function E680_SR_completion Lib "680api" () As
Long
```

37. SRIX4K PCall16

Function name **E680_SR_pcall16(char* chipid)**

Description This function calls SRIX4K PCall16.

Parameters chipid: 1 byte, Chip ID returned from card.

Return code 0 ==> FAIL

1 ==> SUCCESS

Example

C

```
char chipid;
if   (E680_SR_pcall16(&chipid) == 0)
     printf("E680_SR_pcall16 failed");
```

Visual Basic

```
Declare Function E680_SR_pcall16 Lib "680api" (ByVal
chipid As String) As Long
```

38. SRIX4K Slot Marker

Function name **E680_SR_slot_marker(char slot, char* chipid)**

Description This function calls SRIX4K Slot Marker.

Parameters slot: slot number, 1 byte, 0 to 15.

chipid: 1 byte, Chip ID returned from card.

Return code 0 ==> FAIL

1 ==> SUCCESS

Example

C

```
char slot;
char chipid;
slot = 0;
if   (E680_SR_slot_marker(slot, &chipid) == 0)
     printf("E680_SR_slot_marker failed");
```

Visual Basic

```
Declare Function E680_SR_slot_marker Lib "680api"
(ByVal slot As Byte, ByVal chipid As String) As Long
```

39. SRIX4K Reset To Inventory

Function name **E680_SR_reset_to_inventory (void)**

Description SRIX4K reset to inventory

Parameters None

Return code 0 ==> FAIL

1 ==> SUCCESS

Example

C

```
if   (E680_SR_reset_to_inventory() == 0)
     printf("E680_SR_reset_to_inventory failed");
```

Visual Basic

```
Declare Function E680_SR_reset_to_inventory Lib "680api"
() As Long
```

40. SRIX4K Authenticate

Function name **E680_SR_auhenticate(char* rnd, char* data)**

Description This function calls SRIX4K Authenticate.

Parameters rnd: 6 bytes, input random number data

data: 3 bytes, signature data output from the card.

Return code 0 ==> FAIL

1 ==> SUCCESS

Example

C

```
char rnd[10];
char data[10];
rnd[0] = 0x01;
rnd[1] = 0x23;
rnd[2] = 0x45;
rnd[3] = 0x67;
rnd[4] = 0x89;
rnd[5] = 0xAB;

if   (E680_SR_authenticate(rnd, data) == 0)
     printf("E680_SR_authenticate failed");
```

Visual Basic

```
Declare Function E680_SR_authenticate Lib "680api"
(ByVal rnd As String, ByVal data As String) As Long
```

41. SRIX4K Get UID

Function name **E680_SR_get_uid(char* uid)**

Description This function gets UID from card.

Parameters uid: 8 bytes UID. It is interpreted as a 16 bytes hexi-value string.

Return code 0 ==> FAIL

1 ==> SUCCESS

Example

C

```
char serial[25];
if   (E680_SR_get_uid(serial) == 0)
     printf("E680_SR_get_uid failed");
```

Visual Basic

```
Declare Function E680_SR_get_uid Lib "680api" (ByVal
serial As String) As Long
```

42. SRIX4K Read Block

Function name **E680_SR_read_block(char addr, char* data)**

Description This function reads block from SRIX4K card.

Parameters addr: 1 byte, the block to be read

data: 4 bytes, data read from the card

Return code 0 ==> FAIL

1 ==> SUCCESS

Example

C

```
char addr;
char data[20];
addr = 17;
if   (E680_SR_read_block(addr, data) == 0)
     printf("E680_SR_read_block failed");
```

Visual Basic

```
Declare Function E680_SR_read_block Lib "680api" (ByVal
addr As Byte, ByVal data As String) As Long
```

43. SRIX4K Write Block

Function name **E680_SR_write_block(char addr, char* data)**

Description This function writes data to the block of SRIX4K card.

Parameters addr: 1 byte, the block to be written

data: 4 bytes, data to be written

Return code 0 ==> FAIL

1 ==> SUCCESS

Example

C

```
char addr;
char data[20];
addr = 17;
memcpy(data, “0123”, 4);
if   (E680_SR_write_block(addr, data) == 0)
     printf(“E680_SR_write_block failed”);
```

Visual Basic

```
Declare Function E680_SR_write_block Lib "680api"
(ByVal addr As Byte, ByVal data As String) As Long
```

44. Find an ISO15693 tag in antenna field

Function name **E680_15693_Inventory(char* serial)**

Description This function finds an ISO15693 tag. If success, set the tag as Current tag.

Parameters serial: 9 bytes tag information. DSFID: 1 byte, UID: 8 bytes. It is interpreted as a 18 bytes hexi-value string.

Return code 0 ==> FAIL

1 ==> SUCCESS

Example

C

```
char serial[25];
if   (E680_15693_Inventory (serial) == 0)
     printf("E680_15693_Inventory failed");
```

Visual Basic

```
Declare Function E680_15693_Inventory Lib "680api"
(ByVal serial As String) As Long
```

45. Stay quiet

Function name **E680_15693_stay_quiet(void)**

Description Set the current tag stay quiet.

Parameters None

Return code 0 ==> FAIL

1 ==> SUCCESS

Example

C

```
E680_15693_stay_quiet();
```

Visual Basic

```
Declare Function E680_15693_stay_quiet Lib "680api" () As
Long
```

46. Get tag information

Function name **E680_15693_get_tag_information(char* data)**

Description This function gets the current tag information.

Parameters data: tag information, the length is variable, depends on the manufacturer of the tag. It is interpreted as the hexi-value string.

Return code 0 ==> FAIL

1 ==> SUCCESS

Example

C

```
char data[50];
if   (E680_15693_get_tag_information(data) == 0)
     printf("E680_15693_get_tag_information failed");
```

Visual Basic

```
Declare Function E680_15693_get_tag_information Lib
"680api" (ByVal data As String) As Long
```

47. Reset quiet tag to ready

Function name **E680_15693_reset_quiet(char* uid)**

Description This function reset a stay quiet tag to ready.

Parameters uid: 8 bytes, UID of the tag to reset to ready.

Return code 0 ==> FAIL

1 ==> SUCCESS

Example

C

```
char uid[20];
memcpy(uid, “12345678”, 8);
if   (E680_15693_reset_quiet(uid) == 0)
     printf(“E680_15693_reset_quiet failed”);
```

Visual Basic

```
Declare Function E680_15693_reset_quiet Lib "680api"
(ByVal uid As String) As Long
```

48. Read blocks

Function name **E680_15693_read_blocks(char start, char blocks, char* data)**

Description This function reads data blocks from current tag.

Parameters

start: 1 byte, start block to be read

blocks: 1 byte, number of blocks to be read, max 32 blocks in one command

data: Pointer to the data read from the card (blocks x 4 bytes)

Return code

0 ==> FAIL

1 ==> SUCCESS

Example

C

```
char start;
char blocks;
char data[200];
start = 1;
blocks = 10;
if   (E680_15693_read_blocks(start, blocks, data) == 0)
     printf("E680_15693_read_blocks failed");
```

Visual Basic

```
Declare Function E680_15693_read_blocks Lib "680api"
(ByVal start As Byte, ByVal blocks As Byte, ByVal data As
String) As Long
```

49. Write blocks

Function name **E680_15693_write_blocks(char start, char blocks, char* data)**

Description This function writes data blocks to the current tag.

Parameters

start: 1 byte, start block to be written

blocks: 1 byte, number of blocks to be written, max 32 blocks in one command

data: Pointer to the data written to the card (blocks x 4 bytes)

Return code

0 ==> FAIL

1 ==> SUCCESS

Example

C

```
char start;
char blocks;
char data[200];
start = 1;
blocks = 3;
memcpy(data, “0123456789AB”, 12);
if   (E680_15693_write_blocks(start, blocks, data) == 0)
     printf(“E680_15693_write_blocks failed”);
```

Visual Basic

```
Declare Function E680_15693_write_blocks Lib "680api"
(ByVal start As Byte, ByVal blocks As Byte, ByVal data As
String) As Long
```

50. Lock block

Function name **E680_15693_lock_block(char block)**

Description This function locks a block of the current tag.

Parameters block: 1 byte, block number to lock

Return code 0 ==> FAIL

1 ==> SUCCESS

Example

C

```
char block;
block = 1;
if   (E680_15693_lock_block(block) == 0)
     printf("E680_15693_ lock_block failed");
```

Visual Basic

```
Declare Function E680_15693_lock_block Lib "680api"
(ByVal block As Byte) As Long
```

51. Write AFI

Function name **E680_15693_write_AFI(char afi)**

Description This function writes AFI to the current tag.

Parameters afi: 1 byte, AFI to be written to the tag

Return code 0 ==> FAIL

1 ==> SUCCESS

Example

C

```
char afi;
afi = 0x01;
if   (E680_15693_write_AFI(afi) == 0)
     printf("E680_15693_write_AFI failed");
```

Visual Basic

```
Declare Function E680_15693_write_AFI Lib "680api"
(ByVal afi As Byte) As Long
```

52. Lock AFI

Function name **E680_15693_lock_AFI(void)**

Description This function locks AFI of the current tag.

Parameters None

Return code 0 ==> FAIL

1 ==> SUCCESS

Example

C

```
if   (E680_15693_lock_AFI() == 0)
     printf("E680_15693_ lock_AFI failed");
```

Visual Basic

```
Declare Function E680_15693_lock_AFI Lib "680api" () As
Long
```

53. Write DSFID

Function name **E680_15693_write_DSFID(char dsfid)**

Description This function writes DSFID to the current tag.

Parameters dsfid: 1 byte, DSFID to be written to the tag

Return code 0 ==> FAIL

1 ==> SUCCESS

Example

C

```
char dsfid;
dsfid = 0x03;
if   (E680_15693_write_DSFID(dsfid) == 0)
     printf("E680_15693_write_DSFID failed");
```

Visual Basic

```
Declare Function E680_15693_write_DSFID Lib "680api"
(ByVal dsfid As Byte) As Long
```

54. Lock DSFID

Function name **E680_15693_lock_DSFID(void)**

Description This function locks DSFID of the current tag.

Parameters None

Return code 0 ==> FAIL

1 ==> SUCCESS

Example

C

```
if   (E680_15693_lock_DSFID() == 0)
     printf("E680_15693_lock_DSFID failed");
```

Visual Basic

```
Declare Function E680_15693_lock_DSFID Lib "680api" ()
As Long
```

55. Get blocks security

Function name **E680_15693_get_blocks_security(char start, char blocks, char* data)**

Description This function gets blocks security from current tag.

Parameters

start: 1 byte, start block to be gotten

blocks: 1 byte, number of blocks to be gotten

data: Pointer to the data gotten from the card (blocks x 1 bytes)

Return code

0 ==> FAIL

1 ==> SUCCESS

Example

C

```
char start;
char blocks;
char data[50];
start = 1;
blocks = 10;
if (E680_15693_get_blocks_security(start, blocks, data) == 0)
    printf("E680_15693_get_blocks_security failed");
```

Visual Basic

```
Declare Function E680_15693_get_blocks_security Lib "680api" (ByVal start As Byte, ByVal blocks As Byte, ByVal data As String) As Long
```

56. Read blocks by string

Function name **E680_15693_read_blocks_by_string(char start, char blocks, char* data)**

Description This function reads data blocks from current tag. The data to be read is (blocks x 4 x 2) bytes long in hex format.

Parameters

start: 1 byte, start block to be read

blocks: 1 byte, number of blocks to be read, max 32 blocks in one command

data: Pointer to the data read from the card (blocks x 4 x 2 bytes)

Return code

0 ==> FAIL

1 ==> SUCCESS

Example

C

```
char start;
char blocks;
char data[300];
start = 1;
blocks = 10;
if (E680_15693_read_blocks_by_string(start, blocks, data) == 0)
    printf("E680_15693_read_blocks_by_string failed");
```

Visual Basic

```
Declare Function E680_15693_read_blocks_by_string Lib "680api" (ByVal start As Byte, ByVal blocks As Byte, ByVal data As String) As Long
```

57. Write blocks by string

Function name **E680_15693_write_blocks_by_string(char start, char blocks, char* data)**

Description This function writes hexi format data blocks to the current tag.

Parameters

start: 1 byte, start block to be written

blocks: 1 byte, number of blocks to be written, max 32 blocks in one command

data: Pointer to the data written to the card (blocks x 4 x 2 bytes) hexi format

Return code

0 ==> FAIL

1 ==> SUCCESS

Example

C

```
char start;
char blocks;
char data[300];
start = 1;
blocks = 3;
memcpy(data, “303132333435363738394041”, 24);
if (E680_15693_write_blocks_by_string(start, blocks, data) == 0)
    printf(“E680_15693_write_blocks_by_string failed”);
```

Visual Basic

```
Declare Function E680_15693_write_blocks_by_string Lib "680api" (ByVal start As Byte, ByVal blocks As Byte, ByVal data As String) As Long
```