Professional of Smart Card Reader/Writer

URF-35-P

Reference Manual

Document Version: 1.0.1
Last Modified: Aug.2008

INDEX

1 BRIEF

The Smart Card Reader/Writer URF-35-P is interface for the comunication between a computer(for example, a PC) and a smart card. Different types of smart cards have different commands and different communication protocols. This prevents in most cases the direct communication between a smart card and a computer. The URF-35-P establishes a uniform interface from the computer to the smart card for a wide variety of cards.

By taking care of the card specific particulars, it releases the computer software programmer of getting involved with the technical details of the smart card operation, which are in many cases not relevant for the implementation of a smart card system.

The URF-35-P is connected to the computer through a USB interface. The reader accepts commands from the computer, carries out the specified function at the smart card and returns the requested data or status information.

2 FEATURES

- Supports ISO14443A/MIFARE® and ISO14443A-4 cards
- USB Interface between PC and URF-35-P
- Operate distance: 0~30mm
- Lighter
- Buzzer

3 SUPPORTED CARD TYPES

The URF-35-P can operate ISO14443A /MIFARE® and ISO14443A-4 cards.

3.1 ISO14443A/MIFARE® CARDS

- MIFARE® Std. 1K
- MIFARE® Std. 4K
- UltraLight
- MIFARE® Mini

3.2 ISO14443A-4 CARDS

- MIFARE® proX
- MIFARE® DESFire
- Cards compliliant with ISO14443A-4

3.3 UNLISTED ISO14443A CARDS

The reader is designed to read/write the cards compliliant with ISO14443A. Any card compliliant with ISO14443A and unlisted above can also be supported by the reader. Please refer to the card commands set from chip datasheet.

4 API INTRODUCTION

The URF-35-P Application Programming Interface (API) defines a common way of accessing the URF-35-P reader/writer. Application programs invoke URF-35-P through the interface functions and perform operations on the card in the RF field around the antenna through the using of RF communication commands.

The header file mwrf_p.H is available for the program developer which contains all the function prototypes described below.

4.1 Interface Function Prototypes

Interface function set include ①Connection Function, ②Device Operation Function, ③Card Function for MIFARE®, ④Card Command Exchange for ISO14443A, ⑤Protocol Command Exchange, and ⑥Auxiliary Function.

Generally, a program is required to call Open_Device(①Connection Function) at first to obtain a handle which is required for subsequent calls to many other functions, and to call Close_Device(①Connection Function) to release the handle before exiting the program.

The ②Device Operation Function can access the reader for indicator such as buzzer and lighter.

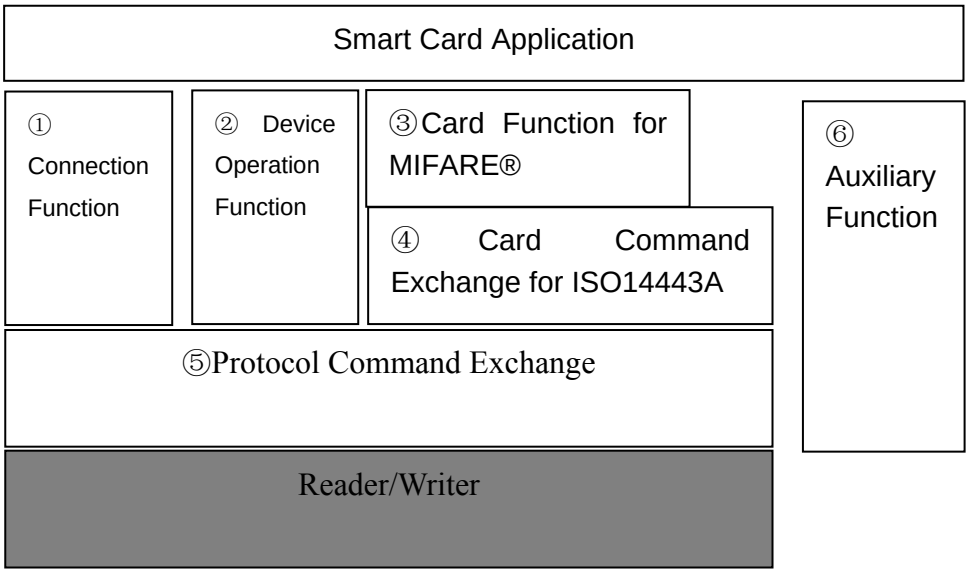
The ③Card Function for MIFARE® is the program interface to MIFARE® standard card.

The ④Card Command Exchange for ISO14443A can support all cards not only MIFARE®.

The ⑤Protocol Command Exchange can finish all the functions ①②③④ covered.

The ⑥Auxiliary Function include a few tools not related to smart card or reader/writer but helpful for program.

The diagram below displays an smart card application and API based on the reader/writer.



4.1.1 Connection Function

4.1.1.1 Open_Device

This function opens a port and returns a valid reader handle for the application program.

Format:

HANDLE Open_Device(char *PortName);

Input Parameters:

The table below lists the parameters for this function.

Parameters	Definition/Values
PortName	The port that conneted with the reader. If using the USB communication port, the input value is “USB”.

Returns:

If the function succeeds, the return value is an open handle to the specified port. Otherwise, it returns NULL or negative value containing the error code.

4.1.1.2 Close_Device

This function closes a previously opened reader port.

Format:

__int16 Close_Device(HANDLE icdev);

Input Parameters:

The table below lists the parameters for this function.

Parameters	Definition/Values
icdev	A valid reader handle previously returned by Open_Device.

Returns:

The return value is zero if the function is successful. Otherwise, it returns a negative value containing the error code.

4.1.2 Device Operation Function

4.1.2.1 Device_Beep

This function controls the buzzer of the reader to beep specified period of time.

Format:

__int16 Device_Beep(HANDLE icdev, unsigned char _Msec);

Input Parameters:

The table below lists the parameters for this function.

Parameters	Definition/Values
icdev	A valid reader handle previously returned by Open_Device.
_Msec	Lasting time of beeping

Returns:

The return value is zero if the function is successful. Otherwise, it returns a nonzero value containing the error code.

4.1.2.2 Device_Light

This function controls the Bi-Color LED of the reader. The Bi-Color LED can be red , yellow or green. And the Bi-Color LED can be turn off.

Format:

__int16 Device_Light(HANDLE icdev, unsigned char _Mode);

Input Parameters:

The table below lists the parameters for this function.

Parameters	Definition/Values
icdev	A valid reader handle previously returned by Open_Device.
_Mode	LED display mode. 0 the LED is turn off. 1 the LED is red. 2 the LED is green.

	3 the LED is yellow.

Returns:

The return value is zero if the function is successful. Otherwise, it returns a nonzero value containing the error code.

4.1.2.3 GetReaderInfo

Calling this function to read the information(version no.) of the reader.

Format:

__int16 GetReaderInfo(HANDLE icdev, char *Info);

Input Parameters:

The table below lists the parameters for this function.

Parameters	Definition/Values
icdev	A valid reader handle previously returned by Open_Device.

Output Parameters:

The table below listed the parameters returned by this function.

Parameters	Definition/Values
Info	The information of the reader.

Returns:

The return value is zero if the function is successful. Otherwise, it returns a nonzero value containing the error code.

4.1.2.4 RF_SwitchOn

This function will generate RF field whatever external is present or not..

Format:

__int16 RF_SwitchOn(HANDLE icdev);

Input Parameters:

The table below lists the parameters for this function.

Parameters	Definition/Values
icdev	A valid reader handle previously returned by Open_Device.

Returns:

The return value is zero if the function is successful. Otherwise, it returns a nonzero value containing the error code.

4.1.2.5 RF_SwitchOff

This function will switch off the RF field immediately.

Format:

__int16 RF_SwitchOff(HANDLE icdev);

Input Parameters:

The table below lists the parameters for this function.

Parameters	Definition/Values
icdev	A valid reader handle previously returned by Open_Device.

Returns:

The return value is zero if the function is successful. Otherwise, it returns a nonzero value containing the error code.

4.1.3 Card Function for MIFARE®

4.1.3.1 RF_Enable_AutomaticRATS

This function can enable/disable automatic ISO14443-4 activation.

Format:

__int16 RF_Enable_AutomaticRATS(HANDLE icdev, BOOL bEnable);

Input Parameters:

The table below lists the parameters for this function.

Parameters	Definition/Values
icdev	A valid reader handle previously returned by Open_Device.
bEnable	If the value is TRUE, the function will enable automatic ISO 14443-4 activation. If the value is FALSE, the function will disable automatic ISO 14443-4 activation.

Returns:

The return value is zero if the function is successful. Otherwise, it returns a nonzero value containing the error code.

4.1.3.2 ChangeCardBitRate

This function can change the defined bit rates with a ISO14443-4 card.

Format:

__int16 ChangeCardBitRate(HANDLE icdev, unsigned char BRit, unsigned char BRti);

Input Parameters:

The table below lists the parameters for this function.

Parameters	Definition/Values
icdev	A valid reader handle previously returned by Open_Device.
BRit	**BRit** is the baud rate to be negotiated for communication from the reader to the Card. – 0x00 : 106 kbps – 0x01 : 212 kbps – 0x02 : 424 kbps
BRti	**BRti** is the baud rate to be negotiated for communication from the Card to the reader. – 0x00 : 106 kbps – 0x01 : 212 kbps – 0x02 : 424 kbps

Returns:

The return value is zero if the function is successful. Otherwise, it returns a nonzero value containing the error code.

4.1.3.3 MF_OpenCard

This function can activate one MIFARE® card.

Format:

__int16 MF_OpenCard(HANDLE icdev, unsigned char SnrLen, unsigned char *Snr);

Input Parameters:

The table below lists the parameters for this function.

Parameters	Definition/Values
icdev	A valid reader handle previously returned by Open_Device.
Snrlen	If the value is 0, the reader will detect one MIFARE® card, and activate the card. If the value is nonzero, the reader will only initizlize the a card with a known ID.
Snr	If Snrlen is 0, this parameter only be used to return the ID of the activated card. If Snrlen is nonzero, this parameter will specify the ID of the card that the reader will initialize.

Output Parameters:

The table below listed the parameters returned by this function.

Parameters	Definition/Values
Snr	The unique ID of the activated card.

Returns:

The return value is zero if the function is successful. Otherwise, it returns a nonzero value

containing the error code.

4.1.3.4 MF_Authentication

This function will perform a MIFARE® card specific command of Authencation A or Authentication B.

Format:

__int16 MF_Authentication(HANDLE icdev,unsigned char Mode, unsigned char BlockNr, unsigned char *Key);

Input Parameters:

The table below lists the parameters for this function.

Parameters	Definition/Values
icdev	A valid reader handle previously returned by Open_Device.
Mode	The authentication mode. If the value is 0, authentication using KEY A If the value is 1, authentication using KEY B
BlockNr	The address to be authenticated..
Key	6 bytes of kkey which be used to authenticate.

Returns:

The return value is zero if the function is successful. Otherwise, it returns a nonzero value containing the error code.

4.1.3.5 MF_Read

This function can read 16 bytes data from a MIFARE® card.

Format:

__int16 MF_Read(HANDLE icdev,unsigned char BlockNr, unsigned char *Data);

Input Parameters:

The table below lists the parameters for this function.

Parameters	Definition/Values
icdev	A valid reader handle previously returned by Open_Device.
BlockNr	the address to be read.

Output Parameters:

The table below listed the parameters returned by this function.

Parameters	Definition/Values

Data	16 bytes of data read from the MIFARE® card.

Returns:

The return value is zero if the function is successful. Otherwise, it returns a nonzero value containing the error code.

4.1.3.6 MF_Read_Hex

This function sends a Read command to a MIFARE® card, and receives 16 bytes hexdecimal data. The returned data are 32 ASCII charaters converted from the 16 bytes hexdecimal data.

Format:

__int16 MF_Read_Hex(HANDLE icdev,unsigned char BlockNr, char *Data);

Input Parameters:

The table below lists the parameters for this function.

Parameters	Definition/Values
icdev	A valid reader handle previously returned by Open_Device.
BlockNr	The address to be read.

Output Parameters:

The table below listed the parameters returned by this function.

Parameters	Definition/Values
Data	32 ASCII characters read from the MIFARE® card

Returns:

The return value is zero if the function is successful. Otherwise, it returns a nonzero value containing the error code.

4.1.3.7 MF_Write

This function can write 16 bytes data to a MIFARE® card.

Format:

__int16 MF_Write(HANDLE icdev,unsigned char BlockNr, unsigned char *Data);

Input Parameters:

The table below lists the parameters for this function.

Parameters	Definition/Values
icdev	A valid reader handle previously returned by Open_Device.

BlockNr	The address to be written.
Data	16 bytes data to be written into the MIFARE® card.

Returns:

The return value is zero if the function is successful. Otherwise, it returns a nonzero value containing the error code.

4.1.3.8 MF_Write_Hex

This function can convert 32 ASCII characters to 16 bytes hexdecimal data, and write the 16 bytes data to the MIFARE® card.

Format:

__int16 MF_Write_Hex(HANDLE icdev,unsigned char BlockNr, char *Data);

Input Parameters:

The table below lists the parameters for this function.

Parameters	Definition/Values
icdev	A valid reader handle previously returned by Open_Device.
BlockNr	The address to be written.
Data	32 ASCII characters to be written.

Returns:

The return value is zero if the function is successful. Otherwise, it returns a nonzero value containing the error code.

4.1.3.9 MF_Write_4_Bytes

This function can write 4 bytes data to the MIFARE® card.

Format:

__int16 MF_Write_4_Bytes(HANDLE icdev,unsigned char BlockNr,unsigned char *Data);

Input Parameters:

The table below lists the parameters for this function.

Parameters	Definition/Values
icdev	A valid reader handle previously returned by Open_Device.
BlockNr	The address to be written.
Data	4 btyes of data to be written.

Returns:

The return value is zero if the function is successful. Otherwise, it returns a nonzero value containing the error code.

4.1.3.10 MF_Write_4_Bytes_Hex

This function can convert 8 ASCII characters to 4 bytes hexdecimal data, and write the 4 bytes hexdecimal data to the MIFARE® card.

Format:

__int16 MF_Write_4_Bytes_Hex(HANDLE icdev,unsigned char BlockNr, char *Data);

Input Parameters:

The table below lists the parameters for this function.

Parameters	Definition/Values
icdev	A valid reader handle previously returned by Open_Device.
BlockNr	The Address to be written.
Data	8 ASCII characters to be written.

Returns:

The return value is zero if the function is successful. Otherwise, it returns a nonzero value containing the error code.

4.1.3.11 MF_Increment

Before calling this function the data block must be intitialized to a value block by calling MF_InitValue first. After calling this function the value block will be incremented a special value.

Format:

__int16 MF_Increment(HANDLE icdev,unsigned char BlockNr, unsigned long Value);

Input Parameters:

The table below lists the parameters for this function.

Parameters	Definition/Values
icdev	A valid reader handle previously returned by Open_Device.
BlockNr	The address to be increment..
Value	Value to be increment.

Returns:

The return value is zero if the function is successful. Otherwise, it returns a nonzero value containing the error code.

4.1.3.12 MF_Decrement

Before calling this function the data block must be intitialized to a value block by calling MF_InitValue first. After calling this function the value block will be decremented a special value.

Format:

__int16 MF_Decrement(HANDLE icdev,unsigned char BlockNr,unsigned long Value);

Input Parameters:

The table below lists the parameters for this function.

Parameters	Definition/Values
icdev	A valid reader handle previously returned by Open_Device.
BlockNr	The address to be decremented.
Value	Value to be decremented.

Returns:

The return value is zero if the function is successful. Otherwise, it returns a nonzero value containing the error code.

4.1.3.13 MF_Transfer

This function can transmit the data of register to a value block.

Format:

__int16 MF_Transfer(HANDLE icdev,unsigned char BlockNr);

Input Parameters:

The table below lists the parameters for this function.

Parameters	Definition/Values
icdev	A valid reader handle previously returned by Open_Device.
BlockNr	The address to receive data from register.

Returns:

The return value is zero if the function is successful. Otherwise, it returns a nonzero value containing the error code.

4.1.3.14 MF_Restore

Before calling this function the data block must be intitialized to a value block by calling

MF_InitValue first. This function can transmit the value block data to register.

Format:

__int16 MF_Restore(HANDLE icdev,unsigned char BlockNr);

Input Parameters:

The table below lists the parameters for this function.

Parameters	Definition/Values
icdev	A valid reader handle previously returned by Open_Device.
BlockNr	The address which data be transmited to register.

Returns:

The return value is zero if the function is successful. Otherwise, it returns a nonzero value containing the error code.

4.1.3.15 MF_InitValue

This function can initialize a data block to a value block of a MIFARE® card, and write a value to the value block.

Format:

__int16 MF_InitValue(HANDLE icdev,unsigned char BlockNr,unsigned long Value);

Input Parameters:

The table below lists the parameters for this function.

Parameters	Definition/Values
icdev	A valid reader handle previously returned by Open_Device.
BlockNr	The address to be initialized.
Value	Value to be written.

Returns:

The return value is zero if the function is successful. Otherwise, it returns a nonzero value containing the error code.

4.1.3.16 MF_ReadValue

This function can read the current value from a value block of a MIFARE® card.

Format:

__int16 MF_ReadValue(HANDLE icdev,unsigned char BlockNr,unsigned long *Value);

Input Parameters:

The table below lists the parameters for this function.

Parameters	Definition/Values
icdev	A valid reader handle previously returned by Open_Device.
BlockNr	The address to be read.

Output Parameters:

The table below listed the parameters returned by this function.

Parameters	Definition/Values
Value	The current value read from the value block of the MIFARE® card

Returns:

The return value is zero if the function is successful. Otherwise, it returns a nonzero value containing the error code.

4.1.3.17 MF_Halt

This function can halt a MIFARE® card in activated state. The card in halt state can be wakeup by calling MF_WakeupCard.

Format:

__int16 MF_Halt(HANDLE icdev);

Input Parameters:

The table below lists the parameters for this function.

Parameters	Definition/Values
Icdev	A valid reader handle previously returned by Open_Device.

Returns:

The return value is zero if the function is successful. Otherwise, it returns a nonzero value containing the error code.

4.1.3.18 MF_WakeupCard

This function can wakeup the card in halt state, and enter to activated state.

Format:

__int16 MF_WakeupCard(HANDLE icdev);

Input Parameters:

The table below lists the parameters for this function.

Parameters	Definition/Values

icdev	A valid reader handle previously returned by Open_Device.

Returns:

The return value is zero if the function is successful. Otherwise, it returns a nonzero value containing the error code.

4.1.3.19 MF_CloseCard

This function can release a card in any state. If the card is released, no further data exchanges will be possible with the card.

Format:

__int16 MF_CloseCard(HANDLE icdev);

Input Parameters:

The table below lists the parameters for this function.

Parameters	Definition/Values
icdev	A valid reader handle previously returned by Open_Device.

Returns:

The return value is zero if the function is successful. Otherwise, it returns a nonzero value containing the error code.

4.1.4 Card Command Exchange for ISO14443A

4.1.4.1 Card_Command_Exchange

This function is used to support protocol data exchanges between the reader and a card whatever type it is.

Format:

__int16 Card_Command_Exchange(HANDLE icdev, unsigned short sLen, unsigned char *SendBlock, unsigned short *rLen, unsigned char *ReceiveBlock);

Input Parameters:

The table below lists the parameters for this function.

Parameters	Definition/Values
icdev	A valid reader handle previously returned by Open_Device.
sLen	The number of bytes to be sent to the card.
SendBlock	sLen bytes of command data to be sent to the card.

Output Parameters:

The table below listed the parameters returned by this function.

Parameters	Definition/Values
rLen	The number of bytes that received from the card, the status byte is not included.
ReceiveBlock	rLen bytes of response data received from the card

Returns:

The return value is zero if the function is successful. Otherwise, it returns a nonzero value containing the error code.

4.1.4.2 Card_Command_Exchange_Hex

This function is used to support protocol data exchanges between the reader and a card whatever type it is.

Format:

__int16 Card_Command_Exchange_Hex(HANDLE icdev, char *SendBlock, char *ReceiveBlock);

Input Parameters:

The table below lists the parameters for this function.

Parameters	Definition/Values
icdev	A valid reader handle previously returned by Open_Device.
SendBlock	ASCII characters which be converted to hexdecimal data, and to be sent to the card

Output Parameters:

The table below listed the parameters returned by this function.

Parameters	Definition/Values
ReceiveBlock	ASCII characters which be converted from the hexdecimal ronsponse data received from the card

Returns:

The return value is zero if the function is successful. Otherwise, it returns a nonzero value containing the error code.

4.1.5 Protocol Command Exchang

4.1.5.1Command_Protocol

This function can support protocol command exchanges between the reader and PC.

Format:

__int16 Command_Protocol(HANDLE icdev, unsigned short sLen, unsigned char *SendBlock, unsigned short *rLen, unsigned char *ReceiveBlock);

Input Parameters:

The table below lists the parameters for this function.

Parameters	Definition/Values
icdev	A valid reader handle previously returned by Open_Device.
sLen	The number of bytes to be sent to the reader.
SendBlock	sLen bytes of command package to be sent to the reader.

Output Parameters:

The table below listed the parameters returned by this function.

Parameters	Definition/Values
rLen	The number of bytes that received from the reader.
ReceiveBlock	rLen bytes of response Package received from the reader

Returns:

The return value is zero if the function is successful. Otherwise, it returns a nonzero value containing the error code.

4.1.5.2 Command_Protocol_Hex

This function can support protocol command exchanges between the reader and PC.
The input command package and the output response package are described in ASCII characters.
The length of input command package is double of the length of bytes to be sent to the reader.
The length of output response package is double of the length of bytes received from the reader.

Format:

__int16 Command_Protocol_Hex(HANDLE icdev, char *SendBlock, char *ReceiveBlock);

Input Parameters:

The table below lists the parameters for this function.

Parameters	Definition/Values

icdev	A valid reader handle previously returned by Open_Device.
SendBlock	ASCII characters which be converted to hexdecimal data, and to be sent to the reader.

Output Parameters:

The table below listed the parameters returned by this function.

Parameters	Definition/Values
ReceiveBlock	ASCII characters which be converted from the hexdecimal ronsponse data received from the reader.

Returns:

The return value is zero if the function is successful. Otherwise, it returns a nonzero value containing the error code.

4.1.6 Auxiliary Function

4.1.6.1 hex_a

This function can convert an array of hexdecimal data to an array of ASCII characters.

Format:

__int16 hex_a(unsigned char *hex, char *a,unsigned char length);

Input Parameters:

The table below lists the parameters for this function.

Parameters	Definition/Values
icdev	A valid reader handle previously returned by Open_Device.
hex	An array of hexdecimal data to be converted.
length	The length of bytes to be converted.

Output Parameters:

The table below listed the parameters returned by this function.

Parameters	Definition/Values
a	An array of ASCII characters after conversion.

Returns:

The return value is zero if the function is successful. Otherwise, it returns a nonzero value containing the error code.

4.1.6.2 a_hex

This function can convert an array of ASCII characters to an array of hexdecimal data.

Format:

__int16 a_hex(char *a,unsigned char *hex,unsigned char len);

Input Parameters:

The table below lists the parameters for this function.

Parameters	Definition/Values
icdev	A valid reader handle previously returned by Open_Device.
a	An array of ASCII characters to be converted.
len	The length of characters to be converted.

Output Parameters:

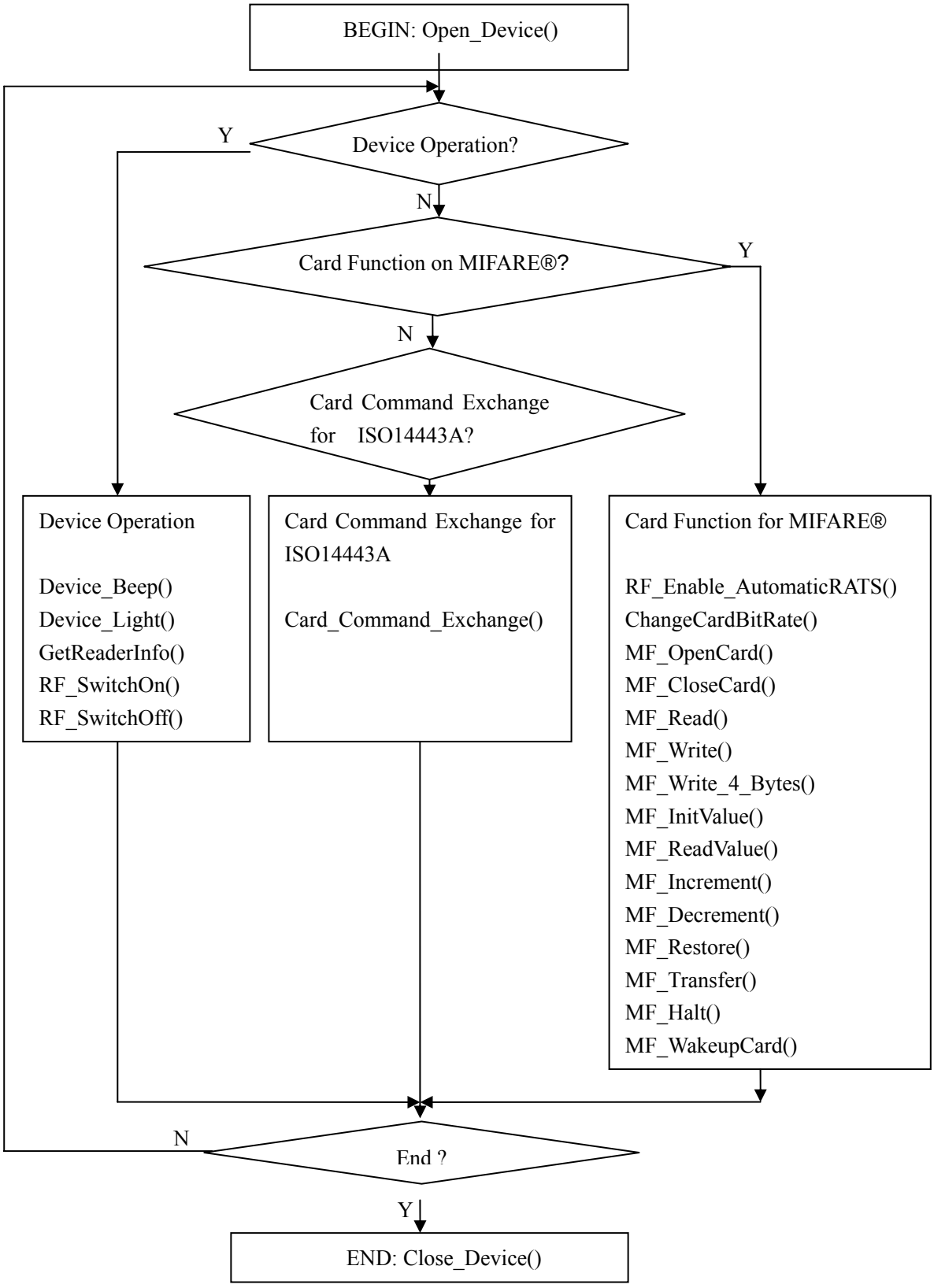
The table below listed the parameters returned by this function.

Parameters	Definition/Values
hex	An array of hexdecimal data after conversion.

Returns:

The return value is zero if the function is successful. Otherwise, it returns a nonzero value containing the error code.

4.2 Basic Smart card Program Flow



4.3 Program with Card Function for MIFARE®

Step0: Activate one MIFARE® Card in passive mode by MF_OpenCard().

If one card is polled, the card will enter to active state. And the UID of the card will be returned.

If no card be polled, let the card out of the RF field, then let the card in the RF field again. Repeat Step0.

Setp1:The card is in active state, then it is available to send operation command to the card. To different type of card the operation command list is different.

◆ Operation Command List for MIFARE® Std. 1K / MIFARE® Std. 4K / MIFARE® Mini

Authentication A/Authentication B by MF_Authentication();
Read by MF_Read()
Write by MF_Write()
Initialize one value block by MF_InitValue()
Read current balue by MF_ReadValue()
Increment by MF_Increment()
Decrement by MF_Decrement()
Restore by MF_Restore()
Transfer by MF_Transfer()

◆ Operation Command List for UltraLight

Read by MF_Read()
Write by MF_Write_4_Bytes()

◆ Operation Command List for MIFARE® ProX

APDU by Card_Command_Exchange()

Step2:If have no further more operation on the card, close the card by MF_CloseCard. All information of the card will be released including card's state and logical number.

Remarks:

1) By default, the reader will perform automatic ISO14443-4 activation if the tag supports ISO14443-4. RF_Enable_AutomaticRATS function can enable/disable automatic ISO14443-4 activation.
2) The reader can change the defined bit rates with a ISO14443-4 target by calling the ChangeCardBitRate function.
3) The antenna can be switched off in order to save the power.
   Calling the function RF_SwitchOn can turn on the antenna power.
   Calling the function RF_SwitchOff can turn off the antenna power.

4.4 Program with Card Command Exchange for ISO14443A

Step0: Activate one ISO14443A Card in passive mode by MF_OpenCard().

If one card is polled, the card will enter to active state. And the UID of the card will be returned.

If no card be polled, let the card out of the RF field, then let the card in the RF field again. Repeat Step0.

Step1:Card Command Exchange by Card_Command_Exchange(). The source data of the command and parameters will be sent to the card, and the original response data from the card will be returned.

◆ **MIFARE® Card**

The source data to be sent must be formatted in the following way:

Cmd	Addr	[Data1..16]

- **Cmd** is the Mifare® specific command byte
- **Addr** is the address associated with the Mifare® command
- **Data1..16** is an array of maximum 16 bytes containing either

– the data to be sent to the card during a writing operation,

– or the data to be used during an authentication operation :

o Data1..6 contain the 6 bytes key,

o Data7..10 contain the 4 bytes serial number of the card.

The data returned in the **DataIn[]** buffer are formatted in the same way:

[Data1..16]

- **Data1..16** is an array of maximum 16 bytes containing data read from the card in case of a reading command.

The Mifare® specific command byte **Cmd** may take one of the possible values:

0x60 / 0x61 Authentication A / Authentication B

0x30 16 bytes reading

0xA0 16 bytes writing

0xA2 4 bytes writing

0xC1 Incrementation

0xC0 Decrementation

0xB0 Transfert

0xC2 Restore

Refer to Mifare® card documentation to have a more detailed description of the Mifare® command set.

Example:

//Authentication A of block 1 whith the key “FFFFFFFFFFFF”, the UID of card is 12345678.

60 01 FFFFFFFFFFFF 12345678

//Read block 1

30 01

◆ **ISO14443-4 card (MIFARE® ProX)**

Card's APDU.
//Select MF file
00 A4 00 00 02 3F 00

Step2:If have no further more operation on the card, close the card by MF_CloseCard. All information of the card will be released including card's state and logical number.

Remarks:

4) By default, the reader will perform automatic ISO14443-4 activation if the tag supports ISO14443-4. RF_Enable_AutomaticRATS function can enable/disable automatic ISO14443-4 activation.
5) The reader can change the defined bit rates with a ISO14443-4 target by calling the ChangeCardBitRate function.
6) The antenna can be switched off in order to save the power.
   Calling the function RF_SwitchOn can turn on the antenna power.
   Calling the function RF_SwitchOff can turn off the antenna power.

4.5 Program With Protocol Command Exchange

All operation can be finished by Command_Protocol(). The input data will be sent as original data, and the original response data will be returned from reader or card.

◆ Start buzzer

<< 0E 9F 87

>> 0F

◆ Stop buzzer

<< 0E BF 87

>> 0F

◆ LED light

LED turn off	0E B0 87
LED red	0E B1 87
LED green	0E B2 87
LED yellow	0E B3 87

◆ RF SwitchOn

<< 32 01 01

>> 33

◆ RF SwitchOff

<< 32 01 00

>> 33

◆ Enable automatic ISO14443-4 activation

<< 12 34

>> 13

◆ Disable automatic ISO14443-4 activation

<< 12 24

>> 13

◆ How to access MIFARE Classic Cards?

Typical sequence may be:
- Scanning the tags in the field (Polling)
- Authentication
- Read / Write the memory of the tag
- Halt the tag (optional)

Step 1) **Polling** for the MIFARE 1K/4K Tags, 106 kbps

<< 4A 01 00

>> 4B 01 01 04 00 08 04 72 CB C6 A9

In which, Number of Card found = [01]; Card number = 01

SENS_RES = 04 00; SEL_RES = 08,

Length of the UID = 4; UID = 72 CB C6 A9

Tip: The card type can be determined by recognizing the SEL_RES.

SEL_RES of some common tag types.

00 = MIFARE Ultralight

08 = MIFARE 1K

09 = MIFARE MINI

18 = MIFARE 4K

20 = MIFARE DESFIRE

Step 2) **KEY A Authentication,** Block **04**, KEY = FF FF FF FF FF FF, UID = 72 CB C6 A9

<< 40 01 **60 04** FF FF FF FF FF FF 72 CB C6 A9

>> 41 [00]

Tip: If the authentication failed, the error code [XX] will be returned.

[00] = Valid, other = Error. Please refer to Error Codes Table for more details.

*Tip: For **KEY B Authentication***

<< 40 01 **61 04** FF FF FF FF FF FF 72 CB C6 A9

Step 3) **Read** the content of Block **04**

<< 40 01 **30 04**

>> 41 [00] 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16

In which, Block Data = 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16

Step 4) **Update** the content of Block **04**

<< 40 01 **A0 04** 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10

>> 41 [00]

Step 5) **Halt the card** (optional)

<< 44 01

>> 45 [00]

MIFARE 1K Memory Map. (1K Bytes)

Sectors (Total 16 sectors. Each sector consists of 4 consecutive blocks)	**Data Blocks** (3 blocks, 16 bytes per block)	**Trailer Block** (1 block, 16 bytes)

Sector 0	0x00 ~ 0x02	0x03
Sector 1	0x04 ~ 0x06	0x07
..		
Sector 14	0x38 ~ 0x0A	0x3B
Sector 15	0x3C ~ 0x3E	0x3F

MIFARE 4K Memory Map.(table 1-2K Bytes)

Sectors (Total 32 sectors. Each sector consists of 4 consecutive blocks)	**Data Blocks** (3 blocks, 16 bytes per block) **Trailer Block** (1 block, 16 bytes)	**Trailer Block** (1 block, 16 bytes)
Sector 0	0x00 ~ 0x02	0x03
Sector 1	0x04 ~ 0x06	0x07
..		
Sector 30	0x78 ~ 0x7A	0x7B
Sector 31	0x7C ~ 0x7E	0x7F

MIFARE 4K Memory Map.(table 2-2K Bytes)

Sectors (Total 8 sectors. Each sector consists of 16 consecutive blocks)	**Data Blocks** (15 blocks, 16 bytes per block)	**Trailer Block** (1 block, 16 bytes)
Sector 32	0x80 ~ 0x8E	0x8F
Sector 33	0x90 ~ 0x9E	0x9F
..		
Sector 38	0xE0 ~ 0xEE	0xEF
Sector 39	0xF0 ~ 0xFE	0xFF

Tip: Once the authentication is done, all the data blocks of the same sector are free to access. For
example, once the data block 0x04 is successfully authenticated (Sector 1), the data blocks 0x04 ~0x07 are free to access.

◆ How to handle Value Blocks of MIFARE 1K/4K Card?

The value blocks are used for performing electronic purse functions. E.g. Increment, Decrement,
Restore and Transfer .. etc. The value blocks have a fixed data format which permits error detection
and correction and a backup management.

Byte Number 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Description Value Value Value Adr Adr Adr Adr
Value: A signed 4-Byte value. The lowest significant byte off a value is stored in the lowest address
byte. Negative values are stored in standard 2's complement format.
Adr: 1-Byte address, which can be used to save the storage address of a block. (optional)
e.g. Value 100 (decimal) = 64 (Hex), assume Block = 0x05
The formatted value block = 64 00 00 00 9B FF FF FF 64 00 00 00 05 FA 05 FA

Step 1) **Update** the content of Block **05 with a value 100 (dec)**

<< 40 01 **A0 05** 64 00 00 00 9B FF FF FF 64 00 00 00 05 FA 05 FA

>> 41 [00]

Step 2) **Increment** the value of Block **05 by 1 (dec)**

<< 40 01 **C1 05** 01 00 00 00

>> 41 [00]

*Tip: **Decrement** the value of Block **05 by 1 (dec)***

<< 40 01 **C0 05** 01 00 00 00

Step 3) **Transfer** the prior calculated value of Block **05 (dec)**

<< 40 01 **B0 05**

>> 41 [00]

*Tip: **Restore** the value of Block **05** (cancel the prior Increment or Decrement operation)*

<< 40 01 **C2 05**

Step 4) **Read** the content of Block **05**

<< 40 01 **30 05**

>> 41 [00] 65 00 00 00 9A FF FF FF 65 00 00 00 05 FA 05 FA

In which, the value = 101 (dec)

Step 5) **Copy** the value of Block **05** to Block **06 (dec)**

<< 40 01 **C2 05**

>> 41 [00]

<< 40 01 **B0 06**

>> 41 [00]

Step 6) **Read** the content of Block **06**

<< 40 01 **30 06**

>> 41 [00] 65 00 00 00 9A FF FF FF 65 00 00 00 05 FA 05 FA

In which, the value = 101 (dec). The Adr "05 FA 05 FA" tells us the value is
copied from Block 05.

#please refer to the MIFARE specification for more detailed information.

◆ How to access MIFARE Ultralight Tags?

Typical sequence may be:

- Scanning the tags in the field (Polling)
- Read / Write the memory of the tag
- Halt the tag (optional)

Step 1) **Polling** for the MIFARE Ultralight Tags, 106 kbps

<< 4A 01 00

>> 4B 01 01 00 44 00 07 04 6E 0C A1 BF 02 84

In which, Number of Tag found = [01]; Target number = 01

SENS_RES = 00 44; SEL_RES = 00,

Length of the UID = 7; UID = 04 6E 0C A1 BF 02 84

Tip: If no tag is found, the following response will be returned.

>> 4B 00 90 00

Step 2) **Read** the content of Block **04**

<< 40 01 **30 04**

>> 41 [00] 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16

In which, Block Data = 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16

Tip: 4 consecutive blocks will be retrieved. Blocks 4, 5, 6 and 7 will be retrieved. Each data block consists of 4 bytes.

Step 3) **Update** the content of Block **04 with** the data "AA BB CC DD"

<< 40 01 **A0 04** AA BB CC DD 00 00 00 00 00 00 00 00 00 00 00 00

>> 41 [00]

Tip: we have to assemble the data into a 16 bytes frame. The first 4 bytes are for data, the rest of the bytes (12 ZEROS) are for padding. Only the block 4 (4 bytes) is updated even through 16 byte are sent to the reader.

Step 4) **Read** the content of Block **04 again**

<< 40 01 **30 04**

>> 41 [00] AA BB CC DD 05 06 07 08 09 10 11 12 13 14 15 16

In which, Block Data = AA BB CC DD 05 06 07 08 09 10 11 12 13 14 15 16

Tip: Only the block 4 is updated. Blocks 5, 6 and 7 remain the same.

Step 5) **Halt the tag** (optional)

<< 44 01

>> 45 [00]

#please refer to the MIFARE Ultralight specification for more detailed information.

◆ How to access ISO14443-4 Type A Cards?

Typical sequence may be:

- Scanning the tags in the field (Polling) with the correct parameter (Type A or B)
- Change the Baud Rate (optional for Type A tags only)
- Perform any T=CL command
- Deselect the tag

Step 1) **Polling** for the ISO14443-4 Type A Tag, 106 kbps

<< 4A 01 00

>> 4B 01 01 00 08 28 04 85 82 2F A0 07 77 F7 80 02 47 65

In which, Number of Tag found = [01]; Target number = 01

SENS_RES = 00 08; SEL_RES = 28,

Length of the UID = 4; UID = 85 82 2F A0

ATS = 07 77 F7 80 02 47 65

Operation Finished = 90 00

Step 2) **Change the default Baud Rate to other Baud Rate (optional)**

<< 4E 01 02 02 // Change to Baud Rate 424 kbps

Or

<< 4E 01 01 01 // Change to Baud Rate 212 kbps

>> 4F [00]

Please check the maximum baud rate supported by the tags. Only Type A tags are

supported.

Step 3) **Perform T=CL command, Get Challenge APDU = 00 84 00 00 08**

<< 40 01 00 84 00 00 08

>> 41 [00] 62 89 99 ED C0 57 69 2B 90 00

In which, Response Data = 62 89 99 ED C0 57 69 2B 90 00

Step 4) **Deselect the Tag**

<< 44 01

>> 41 [00]

Step 5) **Turn off the Antenna Power (optional)**

<< 32 01 00

>> 33

#please refer to the Tag specification for more detailed information.

5 APPENDIX 1—Diagram of MIFARE® Card States