SDK Development Guidance

The SDK package is a company developing applications for the user-developed software package. SDK provides the users with the form of the dynamic link library files.

When the users use our company' series of readers to carry on application software development, they can efficiently and accurately complete the DRF series readers application software developmental in accordance with the Company's SDK. SDK supports VC, VB, C #, VB.NET, C + + Builder and Delphi development.

Use a function note:

1. To label operation (including to read the tag, write tag), had better be in master-slave mode to use, because the UART communication is reading and writing half-duplex, timing mode, this time the card reader timing to read the label and upload data, then easy to keep up with a machine to send commands conflict and cause read down the success rate, if want to use words in timing mode, had better send stop to read the label orders, again to operate. At this time to return to regular mode if, to send the reset to again into reading head command to timing mode.

2. If you want to read sheet labels, use our DEMO software, to the parameters set the first reading and writing is there set to single tag mode.

3. If you want to read the label, use our DEMO software, to the parameters set the first reading and writing is there more set to tag mode.

1、 OpenComm

Function prototype: HANDLE OpenComm (int portNo);

Function description: open serial port.

Return value: Successful return to a serial port HANDLE, failed to return to HANDLE INVALID VALUE (1)

Parameters: ● - portNo: serial number

Calling routine:

```
HANDLE hCom = OpenComm (1); / / open serial port 1
```

```
if (hCom! = INVALID_HANDLE_VALUE)

AfxMessageBox ("Open serial port successful!");

else

AfxMessageBox ("Failed to open serial port!");
```

2、CloseComm

Function prototype: void CloseComm (HANDLE hCom, BYTE ReaderAddr);

Function description: Close the serial port.

Return Value: None

Parameters:

- - hCom: serial port handle
- - ReaderAddr: reader address, default is 0;

Calling routine:

```
CloseComm (hCom,0);
```

3、 ReadFirmwareVersion

Function prototype: BOOL ReadFirmwareVersion (HANDLE hCom, int * main, int * sub, BYTE ReaderAddr);

Function description: Read Firmware version (read the first version).

Return value: Successful return to TRUE (1), failed to return to FALSE (0)

Parameters: ●-hCom: serial port handle

●-main: variable address of the major version number (output parameters)

●-sub: variable address of the sub-version number (output parameter)

● - ReaderAddr: reader address, default is 0;

Calling routine:

```
int main = 0;
int sub = 0;
if (ReadFirmwareVersion (hCom, & main, & sub,0))
{
CString str;
str.Format ("Firmware Version:% d.% d", main, sub);
AfxMessageBox (str);
}
else
AfxMessageBox ("Read failed!");
```

4、 StopReading

Function prototype: BOOL StopReading (HANDLE hCom, BYTE ReaderAddr);

Function description: To make readers stop reading the tag. Usually in timing worke mode, to temporary stop to read the tag, use this function, if want to return to timing worke mode , right now to send the ResumeReading, namely the reset reading head command.

Return value: Successful return to TRUE (1), failed to return to FALSE (0)

Parameters: ● - hCom: serial port handle

● - ReaderAddr: reader address, default is 0;

Calling routine:

```
if (StopReading (hCom,0))
AfxMessageBox ("Stop reading tags successful)
else
```

AfxMessageBox ("Stop reading tags failed!");

5、 ResumeReading

Function prototype: BOOL ResumeReading (HANDLE hCom, BYTE ReaderAddr);

Function description: to restore the card reader reads the tag (reset read head). Reading is equivalent to the guy on the new electricity, the used for timing worke mode , a temporary halt to read the label later, back to the time to read the tag.

Return value: Successful return to TRUE (1), failed to return to FALSE (0)

Parameters: ● - hCom: serial port handle

● - ReaderAddr: reader address, default is 0;

Calling routine:

```
if (ResumeReading (hCom,0))
AfxMessageBox ("Reset Reader successful!");
else
AfxMessageBox ("Read the first reset failed!");
```

6、 IdentifySingleTag

Function prototype: BOOL IdentifySingleTag (HANDLE hCom, BYTE * tagID, BYTE * antennaNo = NULL, BYTE ReaderAddr);

Function description: Identify a single tag.

Return value: Successful return to TRUE (1), failed to return to FALSE (0)

Parameters: ● - hCom: Serial port handle

● - tagID: Array address of the receiving tag ID (output parameter), a length of 12

● - antennaNo: Variable address of the receiving antenna number (output parameter). Set to NULL when is not required.

● - ReaderAddr: reader address, default is 0;

Calling routine:

```
BYTE tagID [12];
BYTE antennaNo;
if (IdentifySingleTag (hCom, tagID, & antennaNo,0))
/ / Just TagID time: if (IdentifySingleTag (hCom, tagID))
{
/ / Transfer tag ID to Hex Format
CString strID;
CString strTmp;
for (int i = 0; i <12; i + +)
{
strTmp.Format ("% 02X", id [i]);
strID + = strTmp;
}
/ / Antenna No
CString strAntennaNo;
strAntennaNo.Format ("% d", antennaNo);
/ / Display
AfxMessageBox (strID + "Device No:" + strAntennaNo);
}
```

7、 IdentifyUploadedSingleTag

Function prototype: BOOL IdentifyUploadedSingleTag (HANDLE hCom, BYTE * tagID, BYTE * devNo = NULL, BYTE * antennaNo = NULL);

Function description: Identify a single tag which is uploaded by the reader. In the "timing worke mode" and "single card mode" to get recognition of the label use this function.

Return value: Successful return to TRUE (1), failed to return to FALSE (0)

Parameters: ● - hCom: Serial port handle

● - tagID: Array address of the receiving tag ID (output parameter), the length is 12

● - devNo: Variable address of the receiving device number (output parameters). Set to NULL when is not required.

● - antennaNo: Variable address of the receiving antenna number (output parameter). Set to NULL when is not required.

● - ReaderAddr: reader address, default is 0;

Calling routine:

```
BYTE tagID [12];

BYTE devNo;

BYTE antennaNo;

if (IdentifyUploadedSingleTag (hCom, tagID, & devNo, & antennaNo))

/ / Just TagID time: if (IdentifyUploadedSingleTag (hCom, tagID))

{

/ / Transfer tag ID to Hex Format

CString strID;

CString strTmp;

for (int i = 0; i <12; i + +)

{
```

```
strTmp.Format ("% 02X", id [i]);

strID + = strTmp;

}

/ / Device No

CString strDevNo;

strDevNo.Format ("% d", devNo);

/ / Antenna No

CString strAntennaNo;

strAntennaNo.Format ("% d", antennaNo);

/ / Display

AfxMessageBox (strID + "Device No:" + strDevNo

+ "Antenna No:" + strAntennaNo);

}
```

Notes: It's usually used with Timer and you may refer to the demonstration procedure for detailed information.

8、 IdentifyUploadedMultiTags

Function prototype: BOOL IdentifyUploadedMultiTags (HANDLE hCom, BYTE * tagNum, BYTE * tagIDs,

BYTE* devNos=NULL, BYTE* antennaNos=NULL); BYTE * devNos = NULL, BYTE * antennaNos = NULL);

Function description: Identify multi-tags which are uploaded by the reader. In the "timing worke mode" and "multiple CARDS mode", access to read more of the reading and writing data using this function label.

Return value: Successful return to TRUE (1), failed to return to FALSE (0)

Parameters: ● - hCom: serial port handle

● - tagNum: the number of variables to receive the a ddress tags (output parameters). the maximum number of the tags which can be read is 200 each time

● - tagIDs: Array address of the receiving tag ID (output parameter), the length is 12 * tagNum

● - devNos: Array address of the receiving device number (output parameters). Length 1 * tagNum, Set to NULL when is not required. When not set to NULL.

● - antennaNos: Array address of the receiving antenna number (output parameter). Length 1 * tagNum, Set to NULL when is not required.

Assuming it reads 10 tags:

* TagNum value is 10;

tagIDs [0 ~ 11], tagIDs [12 ~ 23] ... ..., tagIDs [108 ~ 119], respectively for the 1st, 2nd ... ... 10th tag ID;

devNos [0], devNos [1] ... ... devNos [9] respectively for the 1st, 2nd ... ... 10th tag' device number;

antennaNos [0], antennaNos [1] ... ... antennaNos [9] for the 1st, 2nd ... ... 10th tag' antenna number;

Calling routine:

```
BYTE tagNum;

BYTE tagIDs [12 * 200];

BYTE devNos [200];

BYTE antennaNos [200];

if (IdentifyUploadedMultiTags (hCom, & tagNum, tagIDs, devNos, antennaNos))
```

/ / Just TagID time: if (IdentifyUploadedMultiTags (hCom, & tagNum, tagIDs))

```
{
```

```
CString str;
CString strTmp;
for (int j = 0; j <tagNum; j + +)
{
/ / Line No.
strTmp.Format ("% d", j);
str + = ("No." + strTmp + ":");
/ / Transfer tag ID to Hex Format
      for (int i = 0; i <12; i + +)
{
strTmp.Format ("% 02X", ids [12 * j + i]);
str + = strTmp;
}
/ / Device No
CString strDevNo;
strDevNo.Format ("% d", devNos [j]);
str + = ("Device No:" + strDevNo);
/ / Antenna No
CString strAntennaNo;
strAntennaNo.Format ("% d", antennaNos [j]);
str + = ("Antenna No:" + strAntennaNo);
/ / Change Line
str + = "\ r \ n";
}
```

```
// Display
AfxMessageBox (str);
}
```

Notes: It's usually used with Timer and you may refer to the demonstration procedure for detailed information.

9、 ReadTag

Function prototype: BOOL ReadTag (HANDLE hCom, BYTE memBank, BYTE address, BYTE length, BYTE * data, BYTE ReaderAddr);

Function description: Read tags. The best in "master-slave mode" use. Can read the reservations, EPC area, TID area, and the user area data, this function IdentifySingleTag function with the difference is that

1. IdentifySingleTag function can only read the label EPC ID number. But this function can be read four area, a wider range.

2. IdentifySingleTag label identify distance farther, effect is better, and the function of the distance to read the label nearer, and IdentifySingleTag effect is good.

Return value: Successful return to TRUE (1), failed to return to FALSE (0)

Parameters: ● - hCom: serial port handle

● - memBank: The area to be read. The significance of the values are as follows:

0x00 - Reserved area

0x01 - EPC Area

0x02 - TID District

0x03 - User Area

● - address: The address area to be read, the value range is 0-7.

● - length: The length to be read , the value range is 1 to 8 (Unit is the Word,1 Word = 2Byte).

● - data: The address to be written in the contents (output parameter)

● - ReaderAddr: reader address, default is 0;

Calling routine:

```
BYTE data [2];

if (:: ReadTag (hCom, 0x01, 0x03, 1, data,0)) / / Read a word (2 bytes) content which starts with the address 02 of EPC area / /

{

CString strData;

strData.Format ("% 02X% 02X", data [0], data [1]);

AfxMessageBox ("Read a word (2 bytes) content which starts with the address 02 of EPC area successful!" + StrData);

}

else

AfxMessageBox ("read tags failed!");
```

10、 WriteTagSingleWord

Function prototype: BOOL WriteTagSingleWord (HANDLE hCom, BYTE memBank, BYTE address,BYTE data1, BYTE data2, BYTE ReaderAddr);

Function description: write a word (2 bytes) content to the tag. (Note: EPC area can not be written to address 0, 1) . Best used in master-slave mode.

Return value: Successful return to TRUE (1), failed to return to FALSE (0)

Parameters: ● - hCom: serial port handle

● - memBank: The area to be written. The significance of the values are as follows:

0x00 - reserved area

0x01 - EPC Area

0x02 - TID District

0x03 - User Area

● - address: write the address area, the value for the range is 0-7 (When memBank is the EPC zone, 0, 1 desirable).

● - data1: The first byte of the content to be written

● - data2: The second byte of the content to be written

● - ReaderAddr: reader address, default is 0;

Calling routine:

```
// Add a word (2 bytes) content to address 02 of the EPC area, 2-byte values are 0xFF

if (:: WriteTagSingleWord (hCom, 0x01, 0x02, 0xFF, 0xFF,0))

AfxMessageBox ("Add a word (2 bytes) content to address 02 of the EPC area successful!");

else

AfxMessageBox ("Failed to write tag!");
```

11、FastWriteTagID

Function prototype: BOOL FastWriteTagID (HANDLE hCom, int bytesNum, const BYTE * bytes, BYTE ReaderAddr);

Function description: Fast write tag ID. Only write tag of EPC area, what say normally namely tag number, had better use in master-slave mode.

Return value: Successful return to TRUE (1), failed to return to FALSE (0)

Parameters: ● - hCom: serial port handle

● - bytesNum: the bytes to be written must be 2,4,6,8,10, or 12.

● - bytes: the content of the address to be written

● - ReaderAddr: reader address, default is 0;

Calling routine:

```
BYTE id [4] = {0x00, 0x11, 0x22, 0x33};
if (:: FastWriteTagID (hCom, 4, id,0)) / / write 4 bytes
    AfxMessageBox ("write card successfully!");
else
    AfxMessageBox ("write card failed!");
```

Note: The more bytes you write, the success rate is lower, so please write only the necessary bytes.

12. FastWriteTag

Function prototype: BOOL FastWriteTag(HANDLE hCom, BYTE memBank, BYTE address, BYTE WordCount,const BYTE* bytes,BYTE ReaderAddr)

Function description: Fast write tag, including EPC area, data area and reservations, had better use in master-slave mode.

Parameters:
- ●——hCom: Serial port handle
- ●——memBank: Write to the area. The value of meaning are as follows:
  - 0x00——Reserved area
  - 0x01——EPC area
  - 0x02——TID area
  - 0x03——User area
- ●——address: To write the address in the area, memBank For EPC area, address range of 2-7, the biggest WordCount for 6; For reserve area, address the range of 0-3, the biggest WordCount for 4; For data area, address addresses the range of 0-31, the biggest WordCount for 8;
- ●——WordCount: Write to the length of the content, in words as a unit, a word for 2 bytes
- ●——bytes: To write the address of the content
- ●——ReaderAddr: Reading head address, a host is read by many sets head use, then read single machine head set to 0;

Calling routine:

```
BYTE id[8] = {0x00,0x11,0x22,0x33,0x44,0x55,0x66,0x77};
if(::FastWriteTag(hCom, 3, 0, 4, id,0)) // Writing 4 words, that is,
```

8 bytes to the user area AfxMessageBox (" write CARDS success! ");

```
else
    AfxMessageBox("Write CARDS failed!");
```

Note: once the number of bytes written, the more success rate is lower, so please only write necessary number of bytes.

13、 InitializeTag

Function prototype: BOOL InitializeTag (HANDLE hCom, BYTE ReaderAddr);

Function description: initialize tag. Initialize Tag control word agreement, usually don't need to use this function.

Return value: Successful return to TRUE (1), failed to return to FALSE (0)

Parameters:

- - hCom: serial port handle
- - ReaderAddr: reader address, default is 0;

Calling routine:

```
if (InitializeTag (hCom,0))
  AfxMessageBox ("initialize tag successful!");
else
  AfxMessageBox ("initialize the tag failed!");
```

14、 LockPassWordTag

Function prototype LockPassWordTag (HANDLE hCom, BYTE passwd1, BYTE passwd2, BYTE passwd3, BYTE passwd4, BYTE lockType, BYTE ReaderAddr);

Function description: Through the password lock Tag, usually in master-slave mode using the function.

Parameters:

- - hCom: serial port handle

- passwd1: access-password 1
- passwd2: access-password 2
- passwd3: access-password 3
- passwd4: access-password 4
- lockType: lock type.
- ReaderAddr: reader address, default is 0;

The significance of the values are as follows:

Value	Lock type
00	LOCK USER
01	LOCK TID
02	LOCK EPC
03	LOCK ACCESS
04	LOCK KILL
05	LOCK ALL
Other values	DO NOT LOCK

Calling routine:

```
if (LockPassWordTag (hCom, 0X12, 0X34, 0X56, 0X78, 02,0)) / / LOCK EPC
   AfxMessageBox ("Lock tag successful!");
else
   AfxMessageBox ("Lock tag failed!");
```

15、 UnlockPassWordTag

Function prototype UnlockPassWordTag (HANDLE hCom, BYTE passwd1, BYTE passwd2, BYTE passwd3, BYTE passwd4, BYTE lockType BYTE ReaderAddr);

Function description: Through the password to unlock the Tag, usually in master-slave mode using the function.

Parameters:

- - hCom: serial port handle
- - passwd1: access-password 1
- - passwd2: access-password 2
- - passwd3: access-password 3
- - passwd4: access-password 4
- - lockType: lock type.
- - ReaderAddr: reader address, default is 0;

Locktype , The significance of the values are as follows:

Value	Unlock Type
00	UNLOCK USER
01	UNLOCK TID
02	UNLOCK EPC
03	UNLOCK ACCESS
04	UNLOCK KILL
05	UNLOCK ALL
Other values	DO NOT UNLOCK

Calling routine:

```
if (UnlockPassWordTag (hCom, 0X12, 0X34, 0X56, 0X78, 02,0)) / / UNLOCK EPC
   AfxMessageBox ("unlock tag successful!");
else
   AfxMessageBox ("unlock the tag failed!");
```

16、 KillTag

Function prototype: BOOL KillTag (HANDLE hCom, BYTE passwd1, BYTE passwd2, BYTE passwd3, BYTE passwd4,BYTE ReaderAddr);

Function description: the destruction of tags. Usually in master-slave mode using the function.

Return value: Successful return to TRUE (1), failed to return to FALSE (0)

Parameters:

- - hCom: serial port handle
- - passwd1: Password 1
- - passwd2: Password 2
- - passwd3: Password 3
- - passwd4: Password 4
- - ReaderAddr: reader address, default is 0;

Calling routine:

```
if (KillTag (hCom, passwd1, passwd2, passwd3, passwd4,0))
    AfxMessageBox ("destroy tag successful!");
else
    AfxMessageBox ("destroy tag failed!");
```

17、 GetReaderParameters

Function prototype: BOOL GetReaderParameters (HANDLE hCom, int addr, int paramNum, BYTE * params,BYTE ReaderAddr);

Function description: read-write access to multiple parameters.

Return value: Successful return to TRUE (1), failed to return to FALSE (0)

Parameters:

- -hCom: serial port handle
- -addr: to query the beginning address of the reader parameters(Please refer to the attached list 1-11 for parameters' corresponding address)

●-paramNum: the number of reader parameters to be queried

●-params: the array address of the receiving reader parameters (output parameter)

●——ReaderAddr: reader address, default is 0;

Calling routine:

```
BYTE params [2];

/ / Get the user' flag code and transmit power (corresponding address are 0x64, 0x65)

if (GetReaderParameters (hCom, 0x64, 2, params,0))

{

 CString str;

 str.Format ("User flag code:% d. Power:% d", params [0], params [1]);

AfxMessageBox (str);

}

else

AfxMessageBox ("Failed to get reader parameter!");
```

18、 SetReaderParameters

Function prototype: BOOL SetReaderParameters (HANDLE hCom, int addr, int paramNum, const BYTE * params,BYTE ReaderAddr);

Function description: Set multiple reader parameters. Usually in master-slave mode using the function.

Return value: Successful return to TRUE (1), failed to return to FALSE (0)

Parameters: ● - hCom: serial port handle

● - addr: to query the beginning address of the reader parameters(Please refer to the attached list 1-11 for parameters' corresponding address)

● - paramNum: the number of reader parameters to be queried

● - params: the array address of the reader parameters

● - ReaderAddr: reader address, default is 0;

Calling routine:

```
BYTE params [2] = {0, 140};

/ / User flag code = 0, transmitter power = 140
(corresponding to the address are 0x64, 0x65)

if (SetReaderParameters (hCom, 0x64, 2, params,0))

    AfxMessageBox ("reader parameters for success!");

else

    AfxMessageBox ("Failed to get reader parameter!");
```

19、 ReadTIDByEpcID

Function prototype: BOOL ReadTIDByEpcID (HANDLE hCom, const BYTE * bytes, BYTE * data, BYTE ReaderAddr);

Function description: Assigns tags' EPC area number (12 bytes) to read corresponding tag's TID area (8 bytes). Usually in master-slave mode using the function.

Return value: Successful return to TRUE (1), failed to return to FALSE (0)

Parameters: ● - hCom: serial port handle

● - bytes: EPC number pointer, contains 12 bytes of EPC data

● - data returned by the TID data pointer, the TID contains 8 bytes of data

●——ReaderAddr: reader address, default is 0;

Calling routine:

BYTE params [2] = {0, 140};

//User flag code = 0, transmitter power = 140 (corresponding to the address are 0x64, 0x65)

if ReadTIDByEpcID (hCom, bytes, data,0);

AfxMessageBox ("EPC designated area, read the TID success!");

else

AfxMessageBox ("EPC designated area, read the TID failed!");

20、ReadByEpcID

Function prototype: BOOL ReadByEpcID(HANDLE hCom, BYTE memBank, BYTE address, BYTE WordCount,const BYTE* EpcID, BYTE* data,BYTE ReaderAddr)

Function description: Read tag by epc numbers;

Return value: Successful return to TRUE (1), failed to return to false(0). Usually in master-slave mode using the function.

Parameters:

●——hCom: serial port handle

●——memBank: The area to be written.

The significance of the values are as follows:

0x00 - reserved area

0x01 - EPC Area

0x02 - TID District

0x03 - User Area

●——address: the address of Reading,

When the memBank is EPC area, the address 2-7, max WordCount is 6;

When the memBank is reserved area , the address 0-3, max WordCount is 4;

When the memBank is user area, the address

is 0-31, max WordCount is 8;

●——WordCount: The length of reading, the unit is word, 1 word Is two bytes;

●——EpcID: the pointer of epc data, include 12 byte EPC data

●——data: Read out data

●——ReaderAddr: reader address, default is 0;

Example:

```
BYTE data[8];
BYTE EPCID[12]={11,22,33,44,55,66,77,88,99,00,11,22};
 if ReadByEpcID (hCom, 03,00,4, EPCID, data,0);//指定 EPC 读用户区
     AfxMessageBox(“Read success!”);
 else
     AfxMessageBox(“Read failed!”);
```

21、WriteByEpcID

Function prototype: BOOL WriteByEpcID(HANDLE hCom, BYTE memBank, BYTE address, BYTE WordCount,const BYTE* EpcID,BYTE* data,BYTE ReaderAddr)

Function description: Write tag by epc numbers;

Function description: Read tag by epc numbers;

Return value: Successful return to TRUE (1), failed to return to false(0)

Parameters:

●——hCom: serial port handle

●——memBank: The area to be written.

The significance of the values are as follows:

0x00 - reserved area

0x01 - EPC Area

0x02 - TID District

0x03 - User Area

●——address: the address of Reading,

When the memBank is EPC area, the address 2-7, max WordCount is 6;

When the memBank is reserved area , the address 0-3, max WordCount is 4;

When the memBank is user area, the address is 0-31, max WordCount is 8;

●——WordCount: The length of reading, the unit is word, 1 word Is two bytes;

●——EpcID: The pointer of epc data, include 12 byte EPC data

●——data: Read out data

●——ReaderAddr: reader address, default is 0;

Example:

```
BYTE data[8];
BYTE EPCID[12]={11,22,33,44,55,66,77,88,99,00,11,22};
if WriteByEpcID (hCom, 03,00,4, EPCID, data,0);//write user area
      AfxMessageBox(“Write user area success by epc id!”);
else
      AfxMessageBox(“Write user area failed by epc id!”);
```

22、BeepCtrl

Function prototype: BOOL BeepCtrl(HANDLE hCom,BYTE BeepStatus,BYTE ReaderAddr)

Function description: Control the buzzer beep;

Return value: Success TRUE (1), fail return FALSE (0)

Parameters:

●——hCom: serial port handle

●——BeepStatus: Control the BUZZER

BeepStatus=0: play the beep when read the tag;

BeepStatus=1:close the beep when read the tag;

BeepStatus>=2: play one time the beep;

●——ReaderAddr: Reader address , default is 0;

```
If (BeepCtrl(hCom,0,0))
   AfxMessageBox(“close the beep success!”);
```

23、RelayCtrl

Function description: BOOL RelayCtrl(HANDLE hCom,BYTE RelayOnOff,BYTE ReaderAddr)

Function description: Control relay;

Return value: Success TRUE (1), fail return FALSE (0)

Parameters: ●——hCom: serial port handle

●——RelayOnOff: Control relay parameters;

RelayOnOff =0: Closed relay;

RelayOnOff =1: Open relay;

●——ReaderAddr: Open relay reading head address, a host is read by many sets head use, then read single machine head set to 0;

```
If (RelayCtrl(hCom,0,0))
   AfxMessageBox(“Close relay success!”);
```

Table 1:

Parameters in the **EEPROM** address (Hex)	Project Meaning	Set operation effective value (decimal)	Explain the meaning of values	Other
0x64	Equipment No.	0 - 255	The default is 0	When send or receive commands, some commands have device number, At this time, the device number of the command must be the same with the read head device number.
0x65	Transmit power	0 - 150	Power Analog	
0x70	Card reader mode operation occurs	1, 2, 3	1:master-slave mode 2: Timer mode 3: Trigger mode	**Note: Working in mode 2,3, the master-slave mode is still valid**

Table **2:**

Parameters in the **EEPROM** address (Hex)	Project Meaning	Set operation effective value (decimal)	Explain the meaning of values	Other
0x71	Reader interval	N	Value units: (N * 10) ms N is 10 - 100;	The reader-writer reads the card effectively for the mode 2,3
0x72	When the reader reads the data ,it sends the data link options initiatively	1, 2, 3	1: RS485 link 2: wiegand link 3: RS232 link	

Table **3:**

Parameters in the **EEPROM** address (Hex)	Project Meaning	Set operation effective value (decimal)	Explain the meaning of values	Other
0x73	Wiegand protocol selection	1, 2, 3	1: wiegand26 2: wiegand34 3: wiegand32	Wiegand is effective
0x74	Wiegand output data pulse width	1 - 255	Internal card reader converts into time, time = this value * 10 (microseconds).	
0x75	Wiegand output data pulse period	1 - 255	Internal card reader converts into time, time = this value * 100 (microseconds).	
0x76	Wiegand output repeats	1, 2, 3	Does not support temporarily	
0x77	Wiegand output repetition interval	1, 2, 3, 4, 5, 6, 7, 8, 9, 10	The reader converts internally into time, time = this value * 10 (ms). Does not support temporarily	

Table **4:**

Parameters in the **EEPROM** address (Hex)	Project Meaning	Set operation effective value (decimal)	Explain the meaning of values	Other
0x80	Trigger pin is enabled to set the trigger mode	The low four bits set to 0 or 1, means work does not trigger or trigger	Bit0: corresponding to the trigger pin 0 Bit1: corresponding to the trigger pin (not supported) Bit2: corresponds to the trigger pin 2 (not supported) Bit3: corresponds to the trigger pin 3	The reader-writer reads the card effectively for the mode 2,3

			(not support)	
0x81	Trigger pin's trigger mode	The low four bits set to 0 or 1, indicating low level triggering or high level triggering	Bit0: corresponding to the trigger pin 0 (supports high level trigger)	
			Bit1: corresponding to the trigger pin 1 (not supported)	
			Bit2: corresponding to the trigger pin 2 (not supported)	
			Bit3: corresponding to the trigger pin 3 (not support)	
0x84	OFF-delay time	0 - 240	Internal card reader converts into time, time = this value * 100 ((ms).	

Table 5:

Parameters in the **EEPROM** address (Hex)	Project Meaning	Set operation effective value (decimal)	Explain the meaning of values	Other
0x90	Hopping Setting	0 - 50	0: frequency-hopping mode.	
			1 - 50: Fixed-frequency operation, this value determines the frequency	
0x92 ~ 0x98	FH - Frequency parameters	Bit set to 0 or 1, means they do not select the frequency or not	From 0x92 BIT0 (1st frequency) - BIT7 (7th frequency), the rest may be deduced by analogy, may set 50 frequency periodic duty	

Table 6:

Parameters in the EEPROM address (Hex)	Project Meaning	Set operation effective value (decimal)	Explain the meaning of values	Other
0x87	Single tag and multi-tag identification	0, 1, 2, 3	0: EPC single tag identification	
			1: EPC multi-tag identification	
			2: 18000_6B single tag identification	
			3: 18000_6B multi-tag identification (Not support)	

Table 7:

Parameters in the EEPROM address (Hex)	Project Meaning	Set operation effective value (decimal)	Explain the meaning of values	Other
0x89	Antenna work mode	1, 4	1: Single antenna work mode 4: Multi-loop antenna work mode	
0x8A	Choose antenna work mode	The low four bits set to 0 or 1, means they do not choose or select the appropriate antenna work mode.	0: do not choose any antenna work 1: antenna 1 work 2: antenna 2 work	

			4: Antenna 3 work 8: Antenna 4 work 15: All antennas working	

Table **8:**

Parameters in the **EEPROM** address (Hex)	Project Meaning	Set operation effective value (decimal)	Explain the meaning of values	Other
0x7B	ID neighboring identification	1, 2	1: ID neighboring identification starts 2: Don’t start (send real-time data available)	
0x7A	ID neighboring identification the time	1 - 255	Internal card reader converts into time, time = this value *1(sec)	Note: When ID neighboring identification starts, time value can not be 0, otherwise turns to not start automatically.