

<b>1. OPERATION SYSTEM REQUIREMENT:</b> .....	<b>1</b>
<b>2. FUNCTION LIST:</b> .....	<b>1</b>
<b>2.1) GENERAL FUNCTION:</b> .....	<b>1</b>
<b>2.2) EPCC1-G2 FUNCTION:</b> .....	<b>2</b>
<b>2.3) 18000-6B FUNCTION :</b> .....	<b>3</b>
<b>3. FUNCTION EXPLANATION:</b> .....	<b>4</b>
<b>3.1) GENERAL FUNCTION:</b> .....	<b>4</b>
3.1.1) AutoOpenComPort(): .....	4
3.1.2) OpenComPort(): .....	5
3.1.3) CloseComPort(): .....	5
3.1.4) CloseSpecComPort(): .....	6
3.1.5)GetReaderInformation(): .....	6
3.1.6) WriteComAdr (): .....	7
3.1.7) WriteScanTime ():.....	7
3.1.8) SetPowerDbm (): .....	7
3.1.9) Writedfre (): .....	8
3.1.10) Writebaud (): .....	8
3.1.11) SetWGParameter (): .....	8
3.1.12) SetWorkMode ():.....	9
3.1.13) GetWorkModeParameter (): .....	10
3.1.14) ReadActiveModeData (): .....	10
<b>3.2) EPCC1-G2 FUNCTION:</b> .....	<b>10</b>
3.2.1) Inventory_G2 (): .....	10
3.2.2) ReadCard_G2 (): .....	11
3.2.3) WriteCard_G2 (): .....	12
3.2.4) EraseCard_G2 ():.....	13
3.2.5) SetCardProtect_G2 (): .....	14
3.2.6) DestroyCard_G2 (): .....	15
3.2.7) WriteEPC_G2 (): .....	15
3.2.8) SetReadProtect_G2 (): .....	16
3.2.9) SetMultiReadProtect_G2 (): .....	16
3.2.10) RemoveReadProtect_G2 (): .....	17
3.2.11) CheckReadProtected_G2 ():.....	17
3.2.12) SetEASAlarm_G2 (): .....	17
3.2.13) CheckEASAlarm_G2 (): .....	18
3.2.14) LockUserBlock_G2 (): .....	18
<b>3.3) 18000-6B FUNCTION:</b> .....	<b>19</b>
3.3.1) Inventory_6B (): .....	19
3.3.2) Inventory2_6B (): .....	20
3.3.3) ReadCard_6B ():.....	20
3.3.4) WriteCard_6B (): .....	21
3.3.5) CheckLock_6B (): .....	21
3.3.6) LockByte_6B (): .....	22
<b>4. RETURN VALUE DEFINITION</b> .....	<b>23</b>
<b>5.ERRORCODE DEFINITION</b> .....	<b>24</b>

RRU1861.DLL is a dynamic link library designed to facilitate EPCC1-G2 and 18000-6B protocol UHF tag application software development.

1. **Operation System Requirement:**

WINDOWS 2000/XP

2. **Function List:**

**2.1) General Function:**

1)long WINAPI AutoOpenComPort(long\* Port, unsigned char \*ComAdr, unsigned char \* Baud ,long \*FrmHandle);

2)long WINAPI OpenComPort(long Port, unsigned char \*ComAdr, unsigned char \* Baud, long \*FrmHandle);

3)long WINAPI CloseComPort(void);

4)long WINAPI CloseSpecComPort(long FrmHandle);

5)long WINAPI GetReaderInformation(unsigned char \*ComAdr, unsigned char \*VersionInfo, unsigned char \*ReaderType, unsigned char \*TrType, unsigned char \* dmaxfre, unsigned char \*dminfre, unsigned char \*powerDbm, unsigned char \*ScanTime, long FrmHandle);

6)long WINAPI WriteComAdr(unsigned char \*ComAdr, unsigned char \*ComAdrData, long FrmHandle);

7)long WINAPI WriteScanTime(unsigned char \*ComAdr, unsigned char \*ScanTime, long FrmHandle);

8) long WINAPI SetPowerDbm (unsigned char \*ComAdr, unsigned char \* powerDbm, long FrmHandle);

9)long WINAPI Writedfre (unsigned char \*ComAdr, unsigned char \* dmaxfre, unsigned char \* dminfre, long FrmHandle);

10)long WINAPI Writebaud (unsigned char \*ComAdr, unsigned char \* baud, long FrmHandle);

11)long WINAPI SetWGParameter(unsigned char \*ComAdr, unsigned char Wg\_mode, unsigned char Wg\_Data\_Inteval, unsigned char Wg\_Pulse\_Width, unsigned char Wg\_Pulse\_Inteval, long FrmHandle);

12)long WINAPI SetWorkMode(unsigned char \*ComAdr, unsigned char \* Parameter, long FrmHandle);

13)long WINAPI GetWorkModeParameter (unsigned char \*ComAdr, unsigned char \* Parameter, long FrmHandle);

14) long WINAPI ReadActiveModeData (unsigned char \*ActiveModeData, unsigned char \* Datalength, long FrmHandle);

## **2.2) EPCC1-G2 Function:**

1) Long WINAPI Inventory\_G2 (unsigned char \*ComAdr, unsigned char \* EPCLenandEPC, long \* Totallen, long \*CardNum, long FrmHandle);

2) Long WINAPI ReadCard\_G2 (unsigned char \*ComAdr, unsigned char \* EPC, unsigned char Mem, unsigned char WordPtr, unsigned char Num, unsigned char \* Password , unsigned char maskadr, unsigned char maskLen, unsigned char maskFlag, unsigned char \* Data , unsigned char EPCLength, unsigned char \* errorcode, long FrmHandle);

3) Long WINAPI WriteCard\_G2 (unsigned char \*ComAdr, unsigned char \* EPC, unsigned char Mem, unsigned char WordPtr, unsigned char Writedatalen, unsigned char \*Writedata, unsigned char \* Password, unsigned char maskadr, unsigned char maskLen, unsigned char maskFlag, long WrittenDataNum, unsigned char EPCLength, unsigned char \* errorcode, long FrmHandle);

4) Long WINAPI EraseCard\_G2 (unsigned char \*ComAdr, unsigned char \* EPC, unsigned char Mem, unsigned char WordPtr, unsigned char Num, unsigned char \* Password, unsigned char maskadr, unsigned char maskLen, unsigned char maskFlag, unsigned char EPCLength, unsigned char \* errorcode, long FrmHandle);

5) Long WINAPI SetCardProtect\_G2 (unsigned char \*ComAdr, unsigned char \* EPC, unsigned char select, unsigned char setprotect, unsigned char \* Password, unsigned char maskadr, unsigned char maskLen, unsigned char maskFlag, unsigned char EPCLength, unsigned char \*errorcode, long FrmHandle);

6) Long WINAPI DestroyCard\_G2 (unsigned char \*ComAdr, unsigned char \* EPC, unsigned char \* Password, unsigned char maskadr, unsigned char maskLen, unsigned char maskFlag, unsigned char EPCLength, unsigned char \* errorcode, long FrmHandle);

7) Long WINAPI WriteEPC\_G2 (unsigned char \*ComAdr, unsigned char \* Password, unsigned char \* WriteEPC, unsigned char WriteEPClen, unsigned char \* errorcode, long FrmHandle);

8) Long WINAPI SetReadProtect\_G2 (unsigned char \*ComAdr, unsigned char \* EPC, unsigned char \* Password, unsigned char maskadr, unsigned char maskLen, unsigned char maskFlag, unsigned char EPCLength, unsigned char \* errorcode, long FrmHandle);

9) Long WINAPI SetMultiReadProtect\_G2 (unsigned char \*ComAdr, unsigned char \* Password, unsigned char \* errorcode, long FrmHandle);

10) Long WINAPI RemoveReadProtect\_G2 (unsigned char \*ComAdr, unsigned char \* Password, unsigned char \* errorcode, long FrmHandle);

11) Long WINAPI CheckReadProtected\_G2 (unsigned char \*ComAdr, unsigned char \*readpro, unsigned char \* errorcode, long FrmHandle);

12) Long WINAPI SetEASAlarm\_G2 (unsigned char \*ComAdr, unsigned char \* EPC, unsigned char \* Password, unsigned char maskadr, unsigned char maskLen, unsigned char maskFlag, unsigned char EAS, unsigned char EPCLength, unsigned char \* errorcode, long FrmHandle);

13) Long WINAPI CheckEASAlarm\_G2 (unsigned char \*ComAdr, unsigned char \* errorcode, long FrmHandle);

14) Long WINAPI LockUserBlock\_G2 (unsigned char \*ComAdr, unsigned char \* EPC, unsigned char \* Password, unsigned char maskadr, unsigned char maskLen, unsigned char maskFlag, unsigned char BlockNum, unsigned char EPCLength, unsigned char \* errorcode, long FrmHandle);

### **2.3) 18000-6B Function :**

1) long WINAPI Inventory\_6B (unsigned char \*ComAdr, unsigned char \* ID\_6B , long FrmHandle);

2) long WINAPI Inventory2\_6B (unsigned char \*ComAdr, unsigned char Condition , unsigned char StartAddress, unsigned char mask , unsigned char \* ConditionContent, unsigned char \* ID\_6B , long \* Cardnum, long FrmHandle);

3) long WINAPI ReadCard\_6B (unsigned char \*ComAdr, unsigned char \* ID\_6B , unsigned char StartAddress, unsigned char Num, unsigned char \* Data, unsigned char \* errorcode, long FrmHandle);

4) long WINAPI WriteCard\_6B (unsigned char \*ComAdr, unsigned char \* ID\_6B , unsigned char StartAddress, unsigned char \* Writedata, unsigned char Writedatalen, unsigned char \* writtenbyte, unsigned char \* errorcode, long FrmHandle);

5) long WINAPI LockByte \_6B (unsigned char \*ComAdr, unsigned char \* ID\_6B , unsigned char Address, unsigned char \* errorcode, long FrmHandle);

6) long WINAPI CheckLock\_6B (unsigned char \*ComAdr, unsigned char \* ID\_6B , unsigned char Address, unsigned char \* ReLockState, unsigned char \* errorcode, long FrmHandle);

### 3. Function Explanation:

#### 3.1) General Function:

##### 3.1.1) AutoOpenComPort():

###### Function description:

This function is used to automatically detect the communication port unoccupied by other application and attached with a reader. The function try to establish the connection between them. The protocol parameters are 57600bps, 8 data bits, 1 start bit, 1 stop bit, no parity bit.

If the connection is established successfully, the function will open the communication port and return a valid handle, otherwise the function will return an error code with a invalid handle(value as -1).

###### Usage:

```
Long WINAPI AutoOpenComPort(long* Port, unsigned char *ComAdr, unsigned char * Baud,long* FrmHandle);
```

###### Parameter:

**Port:** Pointed to the communication port number (COM1~COM9) that the reader is detected and connected.

**ComAdr:** Pointed to the address of the reader.

When using broadcasting address 0xFF as ComAdr to call the function, the port number to which the reader is detected and the address of the reader will be writed back to parameter Port and ComAdr;

When using a designated address 0x00~0xFE as ComAdr to call the function, the port number to which the reader with the specified address is detected will be writed back to parameter Port.

Constants COM1~COM9 are defined as follows:

```
#define COM1 1
#define COM2 2
#define COM3 3
#define COM4 4
#define COM5 5
#define COM6 6
#define COM7 7
#define COM8 8
#define COM9 9
```

**Baud :** This value set the baud rate of the serial communication control.

baudrate	Actual baud rate
0	9600bps
1	19200 bps
2	38400 bps
4	56000 bps
5	57600 bps
6	115200 bps

**FrmHandle:** Pointed to the communication handle which is binding with the communication port opened successfully. The application software should use this handle to manipulate the reader connected to the port.

**Returns:**

Zero value when successfully, non-zero value when error occurred.

**3.1.2) OpenComPort():****Function description:**

This function is used to establish the connection between the reader and a specified communication port. The protocol parameters are 19200bps, 8 data bits, 1 start bit, 1 stop bit, no parity bit.

**Usage:**

Long WINAPI OpenComPort(long Port, unsigned char \*ComAdr, unsigned char \* Baud, long\* FrmHandle);

**Parameter:**

**Port:** Communication port number which is a constant from COM1 to COM9 defined as following:

```
#define COM1  1
#define COM2  2
#define COM3  3
#define COM4  4
#define COM5  5
#define COM6  6
#define COM7  7
#define COM8  8
#define COM9  9
```

**ComAdr:** Pointed to the address of the reader.

When using broadcasting address 0xFF as ComAdr to call the function, the address of the reader will be written back to parameter ComAdr;

When using a designated address 0x00~0xFE as ComAdr to call the function, the function will detect whether a specified address reader is connected to the designated communication port.

**Baud :** This value set the baud rate of the serial communication control.

baudrate	Actual baud rate
0	9600bps
1	19200 bps
2	38400 bps
4	56000 bps
5	57600 bps
6	115200 bps

**FrmHandle:** Pointed to the communication handle which is binding with the communication port opened successfully. The application software should use this handle to manipulate the reader connected to the port.

**Returns:**

Zero value when successfully, non-zero value when error occurred.

**3.1.3) CloseComPort():****Function description:**

This function is used to disconnect the reader and release the corresponding

communication port resources. In some development environment, the communication port resources must be released before exiting. Otherwise the operation system will become unstable.

**Usage:**

Long WINAPI CloseComPort(void);

**Parameter:** None.

**Returns:**

Zero value when successfully, non-zero value when error occurred.

### 3.1.4) CloseSpecComPort():

**Function description:**

This function is used to disconnect the reader with the designated communication port and release the corresponding resources.

**Usage:**

Long WINAPI CloseSpecComPort(long FrmHandle);

**Parameter:**

**FrmHandle:** Handle of the corresponding communication port the reader is connected. The handle value is got when calling function AutoOpenComPort or OpenComPort.

**Returns:**

Zero value when successfully, non-zero value when error occurred.

### 3.1.5) GetReaderInformation():

**Function description:**

This function is used to get reader-related information such as reader address(ComAdr), firmware version, supported protocol type and InventoryScanTime.

**Usage:**

Long WINAPI GetReaderInformation(unsigned char \*ComAdr, unsigned char \*VersionInfo, unsigned char \*ReaderType, unsigned char \*TrType, unsigned char \*dmaxfre, unsigned char \*dminfre, unsigned char \*powerdBm, unsigned char \*ScanTime, long FrmHandle);

**Parameter:**

**ComAdr:** Pointed to the address of the reader.

**VersionInfo:** Pointed to 2 bytes firmware version information. The first byte is version number, the second byte is sub-version number.

**ReaderType:** Pointed to the reader type byte. 0x09 lines on RRU1861

**TrType:** Pointed to One bytes supported protocol information.

**dmaxfre:** Output variable, Bit7-Bit6 band set for use; Bit5-Bit0 that the current maximum frequency reader to work, the specific definitions, see the user manual.

**dminfre:** Output variable, Bit7-Bit6 band set for use; Bit5-Bit0 reader work that the current minimum frequency, the specific definitions, see the user manual.

**PowerdBm:** The output power of reader. Range is 0 to 18, when PowerdBm is 0x00, it means the output power of reader unknown.

**ScanTime:** Point to the value of time limit for inventory command. Please refer to RRU1861 User's manual for details.

**FrmHandle:** Handle of the corresponding communication port the reader is connected. The handle value is got when calling function AutoOpenComPort or OpenComPort.

**Returns:**

Zero value when successfully, non-zero value when error occurred.

### 3.1.6) WriteComAdr ():

#### Function description:

This function is used to set a new address of the reader. The address value will store in reader's inner nonvolatile memory. Default address value is 0x00. The value range is 0x00~0xFE. The address 0xFF is reserved as the broadcasting address. When user try to write a 0xFF to ComAdr, the reader will set the value to 0x00 automatically.

#### Usage:

Long WINAPI WriteComAdr(unsigned char \*ComAdr, unsigned char \*ComAdrData, long FrmHandle);

#### Parameter:

**ComAdr:** Pointed to the original address of the reader.

**ComAdrData:** Pointed to the new address of the reader.

**FrmHandle:** Handle of the corresponding communication port the reader is connected. The handle value is got when calling function AutoOpenComPort or OpenComPort.

#### Returns:

Zero value when successfully, non-zero value when error occurred.

### 3.1.7) WriteScanTime ():

#### Function description:

This function is used to set a new value to InventoryScanTime of an appointed reader. The range is 3~255 corresponding to 3\*100ms~255\*100ms InventoryScanTime. The default value of InventoryScanTime is 30\*100ms.

#### Usage:

Long WINAPI WriteScanTime(unsigned char \*ComAdr, unsigned char \*ScanTime, long FrmHandle);

#### Parameter:

**ComAdr:** Pointed to the address of the reader.

**InventoryScanTime:** Pointed to the value of InventoryScanTime.

**FrmHandle:** Handle of the corresponding communication port the reader is connected. The handle value is got when calling function AutoOpenComPort or OpenComPort.

#### Returns:

Zero value when successfully, non-zero value when error occurred.

### 3.1.8) SetPowerDbm ():

#### Function description:

The function is used to set the power of reader.

#### Usage:

Long WINAPI SetPowerDbm (unsigned char \*ComAdr, unsigned char \* powerDbm, long FrmHandle);

#### Parameter:

**ComAdr:** Pointed to the address of the reader.

**Powerdbm:** The output power of reader. range is 0~18

**FrmHandle:** Handle of the corresponding communication port the reader is connected. The handle value is got when calling function AutoOpenComPort or OpenComPort.

**Returns:**

Zero value when successfully, non-zero value when error occurred.

**3.1.9) Writedfre ():****Function description:**

The function is used to set the reader working of the lower limit and the upper limit of frequency.

**Usage:**

Long WINAPI Writedfre (unsigned char \*ComAdr, unsigned char \* dmaxfre, unsigned char \* dminfre, long FrmHandle);

**Parameter:**

**ComAdr:** Pointed to the address of the reader.

**dmaxfre:** Input variable, Bit7-Bit6 band set for use; Bit5-Bit0 that the current maximum frequency reader to work, the specific definitions, see the user manual.

**dminfre:** Input variable, Bit7-Bit6 band set for use; Bit5-Bit0 reader work that the current minimum frequency, the specific definitions, see the user manual.

**FrmHandle:** Handle of the corresponding communication port the reader is connected. The handle value is got when calling function AutoOpenComPort or OpenComPort.

**Returns:**

Zero value when successfully, non-zero value when error occurred.

**3.1.10) Writebaud ():****Function description:**

The function is used to change the serial port baud rate.

**Usage:**

Long WINAPI Writebaud (unsigned char \*ComAdr, unsigned char \* baud, long FrmHandle);

**Parameter:**

**ComAdr:** Pointed to the address of the reader.

**Baud:** After reader power on, the baud rate of reader is 57600bps.

Range is 0~6.

baudrate	Actual baud rate
0	9600bps
1	19200 bps
2	38400 bps
4	56000 bps
5	57600 bps
6	115200 bps

Reader support 43000bps baud rate, but ApdComPort control in DLL is not support 43000bps.

**FrmHandle:** Handle of the corresponding communication port the reader is connected. The handle value is got when calling function AutoOpenComPort or OpenComPort.

**Returns:**

Zero value when successfully, non-zero value when error occurred.

**3.1.11) SetWGParameter ():**

**Function description:**

The function is used to set wiegand parameter.

**Usage:**

```
Long WINAPI SetWGParameter (unsigned char *ComAdr, unsigned char Wg_mode, unsigned char Wg_Data_Inteval, unsigned char Wg_Pulse_Width, unsigned char Wg_Pulse_Inteval, long FrmHandle);
```

**Parameter:**

**ComAdr:** Pointed to the address of the reader.

**Wg\_mode:** Pointed to the mode of wiegand.

**Bit0:** Wigan 26,34 select bit. Bit0 = 0, choose Wiegand 26, Bit0 = 1 the selection of Wigan 34.

**Bit1:** Bit1 = 0 Wiegand output MSB first, Bit1 = 1, Wiegand output LSB first. Other bits to retain, the default is 0.

**Wg\_Data\_Inteval:** Output data interval of time (0 ~ 255) \* 100ms, the default is 30.

**Wg\_Pulse\_Width:** Data pulse width (1 ~ 255) \* 100us, the default value is 10.

**Wg\_Pulse\_Inteval:** Data pulse interval (1 ~ 255) \* 100us, the default value is 15.

**FrmHandle:** Handle of the corresponding communication port the reader is connected. The handle value is got when calling function AutoOpenComPort or OpenComPort.

**Returns:**

Zero value when successfully, non-zero value when error occurred.

**3.1.12) SetWorkMode ():****Function description:**

The function is used to set work mode parameter.

**Usage:**

```
Long WINAPI SetWorkMode(unsigned char *ComAdr, unsigned char * Parameter, long FrmHandle);
```

**Parameter:**

**ComAdr:** Pointed to the address of the reader.

**Parameter:** Pointed to 6 byte with work mode parameter.

From the first byte to the sixth were:

**Read\_Mode:** Working mode selection, Bit1Bit0 = 0: Responses; Bit1Bit0 = 1: active mode;

Bit1Bit0 = 2: Trigger mode (active low); Bit1Bit0 = 3: Trigger mode (active HIGH).

Other places to retain, the default is 0. Only when the Bit1Bit0 not equal to 0, the following parameters to be effective. Other place to retain, the default is 0. Only Bit0 = 1 Shi following parameters to be effective.

**Mode\_State:** Bit0: protocol select bits. Bit0 = 0 , reader support 18000-6C protocol.

Bit1: Output mode select bits. Bit1 = 1 = RS232 output.

Bit2: buzzer prompts to select the bit. Bit2 = 0 opens when the buzzer prompt, Bit2 = 1 buzzer closed when prompted, the default value is 0.

Other places to retain, the default are 0.

**Mem\_Inven:** When the reader work in the 18000-6C protocol when effective, choose to read the storage area or inquiry conducted labels. 0x00: reserved area; 0x01: EPC store; 0x02: TID store; 0x03: user storage area; 0x04: inquiry conducted labels. Other

values reserved.

**First\_Adr:** Specifies the starting address of the child to read. 0x00 said that starting from the first sub-read, 0x01, said beginning from the first two words, and so on.

**Word\_Num:** the number of words to read, RS232 output mode to be valid, the range of 0x01 ~ 0x32 rooms.

**Reserved:** reservation, default is 0x00.

**FrmHandle:** Handle of the corresponding communication port the reader is connected. The handle value is got when calling function AutoOpenComPort or OpenComPort.

**Returns:**

Zero value when successfully, non-zero value when error occurred.

### 3.1.13) GetWorkModeParameter ():

**Function description:**

The function is used to get work mode parameter.

**Usage:**

```
Long WINAPI GetWorkModeParameter(unsigned char *ComAdr, unsigned char *Parameter, long FrmHandle);
```

**Parameter:**

**ComAdr:** Pointed to the address of the reader.

**Parameter:** Pointed to 10 byte with work mode parameter.

From the first byte to the tenth, respectively: 0, 0, 0, 0, Read\_Mode, Mode\_State, Mem\_Inven, First\_Adr, Word\_Num, Reserved.

**FrmHandle:** Handle of the corresponding communication port the reader is connected. The handle value is got when calling function AutoOpenComPort or OpenComPort.

**Returns:**

Zero value when successfully, non-zero value when error occurred.

### 3.1.14) ReadActiveModeData ():

**Function description:**

The function is used to read data with Active mode.

**Usage:**

```
Long WINAPI ReadActiveModeData(unsigned char *ActiveModeData, unsigned char *Datalength, long FrmHandle);
```

**Parameter:**

**ActiveModeData:** Point to the output array variable, read the active mode, the reader sends the data, the size of Datalength bytes.

**Datalength:** Output ,the Length of ActiveModeData.

**FrmHandle:** Handle of the corresponding communication port the reader is connected. The handle value is got when calling function AutoOpenComPort or OpenComPort.

**Returns:**

Zero value when successfully, non-zero value when error occurred.

## 3.2) EPCC1-G2 Function:

### 3.2.1) Inventory\_G2 ():

**Function description:**

The function is used to detect tags in the inductive area and get their EPC values.

**Usage:**

Long WINAPI Inventory\_G2 (unsigned char \*ComAdr, unsigned char \*EPCLenandEPC, long \* TotalLen, long \*CardNum, long FrmHandle);

**Parameter:**

**ComAdr:** Input, pointed to the address of the reader.

**EPCLenandEPC:** Output, Pointed to the array storing the inventory result. It is the EPC data of tag Reader read. The unit of the array includes 1 byte EPCLen and N (the value of EPCLen) bytes EPC. It is the former high-word, low word in the EPC of each tag. It is the former high-byte, low byte in the each word.

**TotalLen:** Output. Pointed to the byte count of the `EPCLenandEPC`.

**CardNum:** Output. Pointed to the number of tag detected.

**FrmHandle:** Handle of the corresponding communication port the reader is connected. The handle value is got when calling function AutoOpenComPort or OpenComPort.

**Returns:**

Zero value when successfully, value:

- 0x01 Return before Inventory finished
  - 0x02 the Inventory-scan-time overflow
  - 0x03 More Data
  - 0x04 Reader module MCU is Full
- others when error occurred.

**3.2.2) ReadCard\_G2 ():****Function description:**

The function is used to read part or all of a Tag's Password, EPC, TID, or User memory. To the word as a unit, start to read data from the designated address.

**Usage:**

Long WINAPI ReadCard\_G2 (unsigned char \*ComAdr, unsigned char \* EPC, unsigned char Mem, unsigned char WordPtr, unsigned char Num, unsigned char \* Password , unsigned char maskadr, unsigned char maskLen, unsigned char maskFlag, unsigned char \* Data , unsigned char EPCLength, unsigned char \* errorcode, long FrmHandle);

**Parameter:**

**ComAdr:** Input. Pointed to the address of the reader.

**EPC:** Input, Pointed to the array of tag's EPC value. It is the EPC number of tag.

**Mem:** Input, Pointed to select the memory area to read.

0x00: Password area;

0x01: EPC memory area;

0x02: TID memory area;

0x03: User's memory area;

Other value when error occurred.

**WordPtr:** Input, Pointed to the address of tag data to read (Word/Hex). Such as, 0x00 stand in start to read data from first word, 0x01 stand in start to read data from second word, and so on.

**Num:** Input, Pointed to the number of word to read. Can not set 0 or 120, otherwise, return the parameter error information. Num <= 120

**Password:** Input, Pointed to the 8 bytes of tag's accesspassword value. From left to right it is the former high-word, low word in the accesspassword.

**maskadr:** Input, EPC masking starting address of byte.

**maskLen:** Input, Masking bytes.

**maskFlag:** Input,EPC masking Flag.

0x00: disabled;

0x01: enabled;

**Data:** Output. Pointed to the array of the data read from tag.

**EPClength:** Input, Pointed to the byte length of EPC.

**Errorcode:** Output, Pointed to an explanation byte when the function return value equals 0xFC.

**FrmHandle:** Handle of the corresponding communication port the reader is connected. The handle value is got when calling function AutoOpenComPort or OpenComPort.

**Returns:**

Zero value when successfully, the Read data is in **Data**, non-zero value when error occurred.

### 3.2.3) WriteCard\_G2 ():

**Function description:**

The function is used to write several words in a Tag's Reserved, EPC, TID, or User memory.

**Usage:**

Long WINAPI WriteCard\_G2 (unsigned char \*ComAdr, unsigned char \* EPC, unsigned char Mem, unsigned char WordPtr, unsigned char Writedatalen, unsigned char \* Writedata, unsigned char \* Password, unsigned char maskadr, unsigned char maskLen, unsigned char maskFlag, long WrittenDataNum, unsigned char EPClength, unsigned char \* errorcode, long FrmHandle);

**Parameter:**

**ComAdr:** Input. Pointed to the address of the reader.

**EPC:** Input, Pointed to the array of tag's EPC value. It is the EPC number of tag.

**Mem:** Input, Pointed to select the memory area to read.

0x00: Password area

0x01: EPC memory area

0x02: TID memory area

0x03: User's memory area

Other value when error occurred.

**WordPtr:** Input, Pointed to the starting address of tag data to write (Word/Hex). If write in the EPC area, it will ignore the start address, and start to write at the address 0x02.

**Writedatalen:** Input, Pointed to the number of bytes to be written. It must be even and greater than 0. The number of bytes is equal to the actual number of data to be written. Otherwise, return the parameter error information.

**Writedata:** Input, Pointed to the array of the word to be written. For example, WordPtr equal 0x02, then the first word in Data write in the address 0x02 of designated Mem, the second word write in 0x03, and so on.

**Password:** Input, Pointed to the 8 bytes of tag's accesspassword value. From left to right it is the former high-word, low word in the accesspassword.

**maskadr:** Input, EPC masking starting address of byte.

**maskLen:** Input, Masking bytes.

**maskFlag:** Input,EPC masking Flag.

0x00: disabled;

0x01: enabled;

**WrittenDataNum:** Output, the number of the word has been written.in word units.

**EPCLength:** Input, Pointed to the byte length of EPC.

**Errorcode:** Output, Pointed to an explanation byte when the function return value equals 0xFC.

**FrmHandle:** Handle of the corresponding communication port the reader is connected. The handle value is got when calling function AutoOpenComPort or OpenComPort.

**Returns:**

Zero value when successfully, non-zero value when error occurred.

### 3.2.4) EraseCard\_G2 ():

**Function description:**

The function is used to erase multiple words in a Tag's Password, EPC, TID, or User memory.

**Usage:**

Long WINAPI EraseCard\_G2 (unsigned char \*ComAdr, unsigned char \* EPC, unsigned char Mem, unsigned char WordPtr, unsigned char Num, unsigned char \* Password, unsigned char maskadr, unsigned char maskLen, unsigned char maskFlag,unsigned char EPCLength, unsigned char \* errorcode, long FrmHandle);

**Parameter:**

**ComAdr:** Input. Pointed to the address of the reader.

**EPC:** Input, Pointed to the array of tag's EPC value. It is the EPC number of tag.

**Mem:** Input, Pointed to select the memory area to read.

0x00: Password area

0x01: EPC memory area

0x02: TID memory area

0x03: User's memory area

Other value when error occurred.

**WordPtr:** Input, Pointed to the address of tag data to erase (Word/Hex). Such as, 0x00 stand in start to erase data from first word, 0x01 stand in start to erase data from second word, and so on. When erase EPC area, **WordPtr** must be greater than or equal to 0x02. Otherwise, return the parameter error information.

**Num:** Input, Pointed to the number of word to erase. Can not set 0, otherwise, return the parameter error information.

**Password:** Input, Pointed to the 8 bytes of tag's accesspassword value. From left to right it is the former high-word, low word in the accesspassword.

**maskadr:** Input, EPC masking starting address of byte.

**maskLen:** Input, Masking bytes.

**maskFlag:** Input,EPC masking Flag.

0x00: disabled;

0x01: enabled;

**EPCLength:** Input, Pointed to the byte length of EPC.

**Errorcode:** Output, Pointed to an explanation byte when the function return value equals 0xFC.

**FrmHandle:** Handle of the corresponding communication port the reader is connected. The handle value is got when calling function AutoOpenComPort or OpenComPort.

**Returns:**

Zero value when successfully, non-zero value when error occurred.

### 3.2.5) SetCardProtect\_G2 ():

#### Function description:

The function is used to set Password area as readable and writeable from any state, readable and writeable from the secured state, permanently readable and writeable, never readable and writeable. It is used to set EPC, TID or User as writeable from any state, writeable from the secured state, permanently writeable, never writeable.

#### Usage:

Long WINAPI SetCardProtect\_G2 (unsigned char \*ComAdr, unsigned char \* EPC, unsigned char select, unsigned char setprotect, unsigned char \* Password, unsigned char maskadr, unsigned char maskLen, unsigned char maskFlag, unsigned char EPCLength, unsigned char \* errorcode, long FrmHandle);

#### Parameter:

**ComAdr:** Input. Pointed to the address of the reader.

**EPC:** Input, Pointed to the array of tag's EPC value. It is the EPC number of tag.

**Select:** Input.

0x00, Control kill password protection setting.

0x01, Control access password protection setting.

0x02, Control EPC memory protection setting.

0x03, Control TID memory protection setting.

0x04, Control User memory protection setting.

Other value when error occurred.

**Setprotect:** Input.

When **Select** is 0x00 or 0x01, **SetProtect** means as follows:

0x00: readable and writeable from any state.

0x01: permanently readable and writeable.

0x02: readable and writeable from the secured state.

0x03: never readable and writeable

When **Select** is 0x02, 0x03 or 0x04, **SetProtect** means as follows:

0x00: writeable from any state.

0x01: permanently writeable.

0x02: writeable from the secured state.

0x03: never writeable.

**Password:** Input, Pointed to the 8 bytes of tag's accesspassword value. From left to right it is the former high-word, low word in the accesspassword.

**maskadr:** Input, EPC masking starting address of byte.

**maskLen:** Input, Masking bytes.

**maskFlag:** Input, EPC masking Flag.

0x00: disabled;

0x01: enabled;

**EPCLength:** Input, Pointed to the byte length of EPC.

**Errorcode:** Output, Pointed to an explanation byte when the function return value equals 0xFC.

**FrmHandle:** Handle of the corresponding communication port the reader is connected. The handle value is got when calling function AutoOpenComPort or OpenComPort.

#### Returns:

Zero value when successfully, non-zero value when error occurred.

### 3.2.6) DestroyCard\_G2 ():

#### Function description:

The function is used to destroy tag. After the tag destroyed, it never process command.

#### Usage:

Long WINAPI DestroyCard\_G2 (unsigned char \*ComAdr, unsigned char \* EPC, unsigned char \* Password, unsigned char maskadr, unsigned char maskLen, unsigned char maskFlag, unsigned char EPClength, unsigned char \* errorcode, long FrmHandle);

#### Parameter:

**ComAdr:** Input. Pointed to the address of the reader.

**EPC:** Input, Pointed to the array of tag's EPC value. It is the EPC number of tag.

**Password:** Input, Pointed to the 8 bytes of tag's accesspassword value. From left to right it is the former high-word, low word in the accesspassword.

**maskadr:** Input, EPC masking starting address of byte.

**maskLen:** Input, Masking bytes.

**maskFlag:** Input, EPC masking Flag.

0x00: disabled;

0x01: enabled;

**EPClength:** Input, Pointed to the byte length of EPC.

**Errorcode:** Output, Pointed to an explanation byte when the function return value equals 0xFC.

**FrmHandle:** Handle of the corresponding communication port the reader is connected. The handle value is got when calling function AutoOpenComPort or OpenComPort.

#### Returns:

Zero value when successfully, non-zero value when error occurred.

### 3.2.7) WriteEPC\_G2 ():

#### Function description:

The function is used to write EPC value in a Tag's EPC memory. Random write one tag in the antenna.

#### Usage:

Long WINAPI WriteEPC\_G2 (unsigned char \*ComAdr, unsigned char \* Password, unsigned char \* WriteEPC, unsigned char WriteEPClen, unsigned char \* errorcode, long FrmHandle);

#### Parameter:

**ComAdr:** Input. Pointed to the address of the reader.

**Password:** Input, Pointed to the 8 bytes of tag's accesspassword value. From left to right it is the former high-word, low word in the accesspassword.

**WriteEPC:** Input, Pointed to the array of the new tag's EPC value to overwrite the old tag's EPC value.

**WriteEPClen:** Input, Pointed to the number of bytes of new EPC value. range is 2~30. It must be even number, such as 2, 4, and so on.

**Errorcode:** Output, Pointed to an explanation byte when the function return value equals 0xFC.

**FrmHandle:** Handle of the corresponding communication port the reader is connected. The handle value is got when calling function AutoOpenComPort or OpenComPort.

**Returns:**

Zero value when successfully, non-zero value when error occurred.

### 3.2.8) SetReadProtect\_G2 ():

**Function description:**

The function is used to set designated tag read protection. After the tag destroyed, it never process command. Even if inventory tag, reader can not get the EPC number. Only NXP's UCODE EPC G2X tags valid.

**Usage:**

Long WINAPI SetReadProtect\_G2 (unsigned char \*ComAdr, unsigned char \* EPC, unsigned char \* Password, unsigned char maskadr, unsigned char maskLen, unsigned char maskFlag, unsigned char EPCLength, unsigned char \* errorcode, long FrmHandle);

**Parameter:**

**ComAdr:** Input. Pointed to the address of the reader.

**EPC:** Input, Pointed to the array of tag's EPC value. It is the EPC number of tag.

**Password:** Input, Pointed to the 8 bytes of tag's accesspassword value. From left to right it is the former high-word, low word in the accesspassword.

**maskadr:** Input, EPC masking starting address of byte.

**maskLen:** Input, Masking bytes.

**maskFlag:** Input, EPC masking Flag.

0x00: disabled;

0x01: enabled;

**EPCLength:** Input, Pointed to the byte length of EPC.

**Errorcode:** Output, Pointed to an explanation byte when the function return value equals 0xFC.

**FrmHandle:** Handle of the corresponding communication port the reader is connected. The handle value is got when calling function AutoOpenComPort or OpenComPort.

**Returns:**

Zero value when successfully, non-zero value when error occurred.

### 3.2.9) SetMultiReadProtect\_G2 ():

**Function description:**

The function is used to random set one tag read protection in the antenna. The tag must be have the same access password. Only NXP's UCODE EPC G2X tags valid.

**Usage:**

Long WINAPI SetMultiReadProtect\_G2 (unsigned char \*ComAdr, unsigned char \* Password, unsigned char \* errorcode, long FrmHandle);

**Parameter:**

**ComAdr:** Input. Pointed to the address of the reader.

**Password:** Input, Pointed to the 8 bytes of tag's accesspassword value. From left to right it is the former high-word, low word in the accesspassword.

**Errorcode:** Output, Pointed to an explanation byte when the function return value equals 0xFC.

**FrmHandle:** Handle of the corresponding communication port the reader is connected.

The handle value is got when calling function AutoOpenComPort or OpenComPort.

**Returns:**

Zero value when successfully, non-zero value when error occurred.

**3.2.10) RemoveReadProtect\_G2 ():**

**Function description:**

The function is used to reset only one tag read protection in the antenna. The tag must have the same access password. Only NXP's UCODE EPC G2X tags valid.

**Usage:**

Long WINAPI RemoveReadProtect\_G2 (unsigned char \*ComAdr, unsigned char \* Password, unsigned char \* errorcode, long FrmHandle);

**Parameter:**

**ComAdr:** Input. Pointed to the address of the reader.

**Password:** Input, Pointed to the 8 bytes of tag's accesspassword value. From left to right it is the former high-word, low word in the accesspassword.

**Errorcode:** Output, Pointed to an explanation byte when the function return value equals 0xFC.

**FrmHandle:** Handle of the corresponding communication port the reader is connected. The handle value is got when calling function AutoOpenComPort or OpenComPort.

**Returns:**

Zero value when successfully, non-zero value when error occurred.

Note: If tag does not support the read protection setting, it must be unprotected.

**3.2.11) CheckReadProtected\_G2 ():**

**Function description:**

The function is used to check only one tag in the antenna whether the tag is protected. It can not check the tag whether the tag support protection setting. Only NXP's UCODE EPC G2X tags valid.

**Usage:**

Long WINAPI CheckReadProtected\_G2 (unsigned char \*ComAdr, unsigned char \*Readpro, unsigned char \* errorcode, long FrmHandle);

**Parameter:**

**ComAdr:** Input. Pointed to the address of the reader.

**Readpro:** Output.

0x00 Tag is unprotected

0x01 Tag is protected

Note: If tag does not support the read protection setting, it must be unprotected.

**Errorcode:** Output, Pointed to an explanation byte when the function return value equals 0xFC.

**FrmHandle:** Handle of the corresponding communication port the reader is connected. The handle value is got when calling function AutoOpenComPort or OpenComPort.

**Returns:**

Zero value when successfully, non-zero value when error occurred.

Note: If tag does not support the read protection setting, it must be unprotected.

**3.2.12) SetEASAlarm\_G2 ():**

**Function description:**

The function is used to set or reset the EAS status bit of designated tag. Only NXP's UCODE EPC G2X tags valid.

**Usage:**

Long WINAPI SetEASAlarm\_G2 (unsigned char \*ComAdr, unsigned char \* EPC, unsigned char \* Password, unsigned char maskadr, unsigned char maskLen, unsigned char maskFlag, unsigned char EAS, unsigned char EPCLength, unsigned char \* errorcode, long FrmHandle);

**Parameter:**

**ComAdr:** Input. Pointed to the address of the reader.

**EPC:** Input, Pointed to the array of tag's EPC value. It is the EPC number of tag.

**Password:** Input, Pointed to the 8 bytes of tag's accesspassword value. From left to right it is the former high-word, low word in the accesspassword.

**maskadr:** Input, EPC masking starting address of byte.

**maskLen:** Input, Masking bytes.

**maskFlag:** Input, EPC masking Flag.

0x00: disabled;

0x01: enabled;

**EPCLength:** Input, Pointed to the byte length of EPC.

**EAS:** Input. The bit0 of **EAS** is 0, it means setting EAS closed.

The bit0 of **EAS** is 1, it means setting EAS opened.

bit1 ~ bit7 default is 0.

**Errorcode:** Output, Pointed to an explanation byte when the function return value equals 0xFC.

**FrmHandle:** Handle of the corresponding communication port the reader is connected. The handle value is got when calling function AutoOpenComPort or OpenComPort.

**Returns:**

Zero value when successfully, non-zero value when error occurred.

### 3.2.13) CheckEASAlarm\_G2 ():

**Function description:**

The function is used to check EAS status bit of any tag in the antenna. Only NXP's UCODE EPC G2X tags valid.

**Usage:**

Long WINAPI CheckEASAlarm\_G2 (unsigned char \*ComAdr, unsigned char \* errorcode, long FrmHandle);

**Parameter:**

**ComAdr:** Input. Pointed to the address of the reader.

**Errorcode:** Output, Pointed to an explanation byte when the function return value equals 0xFC.

**FrmHandle:** Handle of the corresponding communication port the reader is connected. The handle value is got when calling function AutoOpenComPort or OpenComPort.

**Returns:**

Zero value when EAS alarm,

0xFB value when no EAS alarm,

Otherwise, return other value when error occurred.

### 3.2.14) LockUserBlock\_G2 ():

**Function description:**

The function is used to permanently lock the designated data in designated tag's user memory. The locked data can be read only, but not written and not erased. Only NXP's UCODE EPC G2X tags valid.

**Usage:**

Long WINAPI LockUserBlock\_G2 (unsigned char \*ComAdr, unsigned char \* EPC, unsigned char \* Password, unsigned char maskadr, unsigned char maskLen, unsigned char maskFlag, unsigned char BlockNum,unsigned char EPCLength, unsigned char \* errorcode, long FrmHandle);

**Parameter:**

**ComAdr:** Input. Pointed to the address of the reader.

**EPC:** Input, Pointed to the array of tag's EPC value. It is the EPC number of tag.

**Password:** Input, Pointed to the 8 bytes of tag's accesspassword value. From left to right it is the former high-word, low word in the accesspassword.

**maskadr:** Input, EPC masking starting address of byte.

**maskLen:** Input, Masking bytes.

**maskFlag:** Input,EPC masking Flag.

0x00: disabled;

0x01: enabled;

**BlockNum:** Input. Pointed to the address to lock:

- 0 or 1 permanently lock block data 0 and 1
- 2 or 3 permanently lock block data 2 and 3
- 4 or 5 permanently lock block data 4 and 5
- 6 or 7 permanently lock block data 6 and 7
- 8 or 9 permanently lock block data 8 and 9
- 10 or 11 permanently lock block data 10 and 11
- 12 or 13 permanently lock block data 12 and 13

**EPCLength:** Input, Pointed to the byte length of EPC.

**Errorcode:** Output, Pointed to an explanation byte when the function return value equals 0xFC.

**FrmHandle:** Handle of the corresponding communication port the reader is connected. The handle value is got when calling function AutoOpenComPort or OpenComPort.

**Returns:**

Zero value when successfully, non-zero value when error occurred.

**3.3) 18000-6B Function:**

**3.3.1) Inventory\_6B ():**

**Function description:**

The function is used to detect only one tag in the inductive area and get their ID values.If more than one tag in the inductive area at the same time, reader may be detect nothing.

**Usage:**

Long WINAPI Inventory\_6B (unsigned char \*ComAdr, unsigned char \* ID\_6B , int FrmHandle);

**Parameter:**

**ComAdr:** Input. Pointed to the address of the reader.

**ID\_6B:** Output, Pointed to the array storing the inventory result. The array is 8 bytes ID.

**FrmHandle:** Handle of the corresponding communication port the reader is connected. The handle value is got when calling function AutoOpenComPort or OpenComPort.

**Returns:**

Zero value when successfully, non-zero value when error occurred.

### 3.3.2) Inventory2\_6B ():

**Function description:**

The function is used to according to the given conditions detect tags in the inductive area and get their ID values.

**Usage:**

Long WINAPI Inventory2\_6B (unsigned char \*ComAdr, unsigned char \* Condition , unsigned char StartAddress, unsigned char mask , unsigned char \* ConditionContent , unsigned char \* ID\_6B , long \* Cardnum, long FrmHandle);

**Parameter:**

**ComAdr:** Input. Pointed to the address of the reader.

**Condition:** Input, the condition of detecting tags.

0x00: equal to condition.

0x01: unequal to condition.

0x02: greater than to condition.

0x03: less than to condition.

**StartAddress:** Input, the tag's start address to compare.

**Mask:** Input, mask. Pointed to the data is used to compare. Highest bit in the mask correspond with the far-left byte in the ConditionContent. The corresponding bit in the mask is 1 to compare the bit in the ConditionContent with the corresponding byte in the tag. The corresponding bit in the mask is 0, not compare.

**ConditionContent:** Input, Pointed to the array is used to compare. The array is 8 bytes

**ID\_6B:** Output, Pointed to the array storing the inventory result. The unit of the array is 8 bytes ID.

**Cardnum:** Output, the count of tag

**FrmHandle:** Handle of the corresponding communication port the reader is connected.

The handle value is got when calling function AutoOpenComPort or OpenComPort.

**Returns:**

Zero value when successfully, value:

0x15 Return before Inventory finished

0x16 the Inventory-scan-time overflow

0x17 More Data

0x18 Reader module MCU is Full

others when error occurred.

### 3.3.3) ReadCard\_6B ():

**Function description:**

The function is used to start to read several bytes from the designated address.

**Usage:**

Long WINAPI ReadCard\_6B (unsigned char \*ComAdr, unsigned char \* ID\_6B , unsigned char StartAddress, unsigned char Num, unsigned char \* Data, unsigned char \* errorcode, long FrmHandle);

**Parameter:**

**ComAdr:** Input. Pointed to the address of the reader.

**ID\_6B:** Input, Pointed to the array storing the inventory result. The unit of the array is 8 bytes ID.

**StartAddress:** Input, the start address to read data. Range is 8~223. If the address more than range, reader will return parameter error information.

**Num:** Input, pointed to the number of bytes to read, range is 1~32. If **StartAddress** + **Num** greater than 224, or **Num** greater than 32 or is zero, reader will return parameter error information.

**Data:** Output, Pointed to the array storing the read result.

**Errorcode:** Output, reservation.

**FrmHandle:** Handle of the corresponding communication port the reader is connected. The handle value is got when calling function AutoOpenComPort or OpenComPort.

**Returns:**

Zero value when successfully, non-zero value when error occurred.

### 3.3.4) WriteCard\_6B ():

**Function description:**

The function is used to start to write several bytes from the designated address.

**Usage:**

Long WINAPI WriteCard\_6B (unsigned char \*ComAdr, unsigned char \* ID\_6B , unsigned char StartAddress, unsigned char \* Writedata, unsigned char WritedataLen, unsigned char \* writtenbyte, unsigned char \* errorcode, long FrmHandle);

**Parameter:**

**ComAdr:** Input. Pointed to the address of the reader.

**ID\_6B:** Input, Pointed to the array storing the inventory result. The unit of the array is 8 bytes ID.

**StartAddress:** Input, pointed to the start address to read data. Range is 8~223. If the address more than range, reader will return parameter error information.

**Writedata:** Input, pointed to the array to write, range is 1~32. If **Address** + **WritedataLen** greater than 224, or **Writedata** greater than 32 or is zero, reader will return parameter error information. The high bytes of **Writedata** write in the low address in tag.

**WritedataLen:** Input, pointed to the number of bytes to write.

**Writtenbyte:** Output. Pointed to the number of bytes written successfully start from the high byte in the **Writedata**.

**Errorcode:** Output, reservation.

**FrmHandle:** Handle of the corresponding communication port the reader is connected. The handle value is got when calling function AutoOpenComPort or OpenComPort.

**Returns:**

Zero value when successfully, non-zero value when error occurred.

Note: If tag does not support the read protection setting, it must be unprotection.

### 3.3.5) CheckLock\_6B ():

**Function description:**

The function is used to check whether the designated byte is locked.

**Usage:**

Long WINAPI CheckLock\_6B (unsigned char \*ComAdr, unsigned char \* ID\_6B ,

unsigned char Address, unsigned char \* ReLockState, unsigned char \* errorcode, long FrmHandle);

**Parameter:**

**ComAdr:** Input, pointed to the address of the reader.

**ID\_6B:** Input, Pointed to the array storing the inventory result. The unit of the array is 8 bytes ID.

**Address:** Input. Pointed to the address is used to check whether is locked. Range is 0~223. Beyond this range, reader will return parameter error.

**ReLockState:** Output.

0x00, the byte is unlocked.

0x01, the byte is locked, cannot lock it again.

**Errorcode:** Output, reservation.

**FrmHandle:** Handle of the corresponding communication port the reader is connected. The handle value is got when calling function AutoOpenComPort or OpenComPort.

**Returns:**

Zero value when successfully, non-zero value when error occurred.

**3.3.6) LockByte\_6B ():**

**Function description:**

The function is used to lock the designated byte.

**Usage:**

Long WINAPI LockByte\_6B (unsigned char \*ComAdr, unsigned char \* ID\_6B , unsigned char Address, unsigned char \* errorcode, long FrmHandle);

**Parameter:**

**ComAdr:** Input, pointed to the address of the reader.

**ID\_6B:** Input, Pointed to the array storing the inventory result. The unit of the array is 8 bytes ID.

**Address:** Input, pointed to the address to lock. Range is 8~223. Beyond this range, reader will return parameter error.

**Errorcode:** Output, reservation.

**FrmHandle:** Handle of the corresponding communication port the reader is connected. The handle value is got when calling function AutoOpenComPort or OpenComPort.

**Returns:**

Zero value when successfully, non-zero value when error occurred.

#### 4. Return Value Definition

#define InventoryReturnEarly_G2	0x01 Return before Inventory finished
#define InventoryTimeOut_G2	0x02 the Inventory-scan-time overflow
#define InventoryMoreData_G2	0x03 More Data
#define ReadermoduleMCUFull_G2	0x04 Reader module MCU is Full
#define AccessPasswordError	0x05 Access password error
#define DestroyPasswordError	0x09 Destroy password error
#define DestroyPasswordCannotZero	0x0a Destroy password error can't be Zero
#define TagNotSupportCMD	0x0b Tag Not Support the command
#define AccessPasswordCannotZero	0x0c Use the commmand,Access Password Can't be Zero
#define TagProtectedCannotSetAgain	0x0d Tag is protected,cannot set it again
#define TagNoProtectedDonotNeedUnlock	0x0e Tag is unprotected,no need to reset it
#define ByteLockedWriteFail	0x10 There is some locked bytes,write fail
#define CannotLock	0x11 can not lock it
#define LockedCannotLockAgain	0x12 is locked,cannot lock it again
#define ParameterSaveFailCanUseBeforeNoPower	0x13 Save Fail,Can Use Before Power
#define CannotAdjust	0x14 Cannot adjust
#define InventoryReturnEarly_6B	0x15 Return before Inventory finished
#define InventoryTimeOut_6B	0x16 Inventory-Scan-Time overflow
#define InventoryMoreData_6B	0x17 More Data
#define ReadermoduleMCUFull_6B	0x18 Reader module MCU is full
#define NotSupportCMDOrAccessPasswordCannotZero	0x19 Not Support Command Or AccessPassword Cannot be Zero
#define CMDExecuteErr	0xF9 Command execute error
#define GetTagPoorCommunicationCannotOperation	0xFA Get Tag,Poor Communication,Inoperable
#define NoTagOperation	0xFB No Tag Operable
#define TagReturnErrorCode	0xFC Tag Return ErrorCode
#define CMDLengthWrong	0xFD Command length wrong
#define IllegalCMD	0xFE Illegal command
#define ParameterError	0xFF Parameter Error
#define CommunicationErr	0x30 Communication error
#define RetCRCErr	0x31 CRC checksumat error
#define RetDataErr	0x32 Return data length error
#define CommunicationBusy	0x33 Communication busy
#define ExecuteCmdBusy	0x34 Busy,command is being executed
#define ComPortOpened	0x35 ComPort Opened
#define ComPortClose	0x36 ComPort Closed
#define InvalidHandle	0x37 Invalid Handle
#define InvalidPort	0x38 Invalid Port
#define RecmdErr	0XEE Return command error

## 5. ErrorCode Definition

#define OtherError	0x00 Other error
#define MemoryOutPcNotSupport	0x03 Memory out or pc not support
#define MemoryLocked	0x04 Memory Locked and unwritable
#define NoPower	0x0b No Power,memory write operation cannot be executed
#define NotSpecialError	0x0f Not Special Error,tag not support special errorcode