

## UHFGATE.DLL User Guide V1.1

1. Operation System Requirement .....	1
2. Function List .....	1
3. Function Explanation .....	5
3.1) AutoOpenComPort .....	5
3.2) OpenComPort().....	6
3.3) CloseComPort.....	6
3.4) CloseSpecComPort .....	7
3.5) OpenNetPort .....	7
3.6) CloseNetPort.....	7
3.7) GetChannelMessage .....	8
3.8) Acknowledge .....	8
3.9) SetClock.....	9
3.10) GetClock.....	9
3.11) ClearControllerBuffer .....	10
3.12) ConfigureController.....	10
3.13) GetControllerConfig .....	11
3.14) GetControllerInfo .....	12
3.15) GetControllerReaderConnectionStatus.....	12
3.16) SetControllerAddr.....	13
3.17) ModeSwitch.....	13
3.18) IRDirectionSetting.....	14
3.19) GetEASMessage .....	14
3.20) Set EAS work style .....	15
3.21) Get EAS work style .....	16
3.22) set read parameter .....	17
3.23) Get read parameter.....	18
3.24) Get statistical message .....	19
3.25) Ser reader work paramater.....	19
3.26) Ger reader work paramater .....	20
3.27) SetRelay .....	21
3.28) BuzzerAndLEDControl .....	21
4. Appendix 1 .....	22

# 1. Operation System Requirement

WINDOWS 2000/XP/7/8

## 2. Function List

UHFGATE.DLL includes the following functions:

- 1) int AutoOpenComPort(int\* Port, unsigned char \*ConAddr, int\* PortHandle);
- 2) int OpenComPort(int Port, unsigned char \*ConAddr, int\* PortHandle);
- 3) int CloseComPort( void);
- 4) int CloseSpecComPort(int PortHandle);
- 5) int OpenNetPort(int Port, LPSTR IPAddr, unsigned char \*ConAddr, int\* PortHandle);
- 6) int CloseNetPort(int PortHandle);
- 7) int GetChannelMessage(unsigned char \*ConAddr, unsigned char\* Msg, unsigned char\* MsgLength , unsigned char \*MsgType, unsigned char\* IRStatue, int PortHandle);
- 8) int Acknowledge(unsigned char \*ConAddr, int PortHandle);
- 9) int SetClock(unsigned char \*ConAddr, unsigned char\* SetTime, unsigned char\* IRStatue, int PortHandle);
- 10) int GetClock(unsigned char \*ConAddr, unsigned char\* CurrentTime, unsigned char\* IRStatue, int PortHandle );
- 11) int ClearControllerBuffer(unsigned char \*ConAddr, unsigned char\* IRStatue, int PortHandle);
- 12) int ConfigureController(unsigned char \*ConAddr, unsigned char IREnable, unsigned char IRTime, unsigned char TagExistTime, unsigned char AlarmEn, unsigned char DelayTime, unsigned char Pepolemsg, unsigned char AEn, unsigned char\* IRStatue, int PortHandle);

13) int GetControllerConfig(unsigned char \*ConAddr, unsigned char \*IReEnable, unsigned char \*IRTime, unsigned char \*TagExistTime, unsigned char \*AlarmEn, unsigned char \*DelayTime, unsigned char \*Pepolemsg, unsigned char \*AEn, unsigned char\* IRStatue, int PortHandle);

14) int GetControllerInfo(unsigned char \*ConAddr, unsigned char\* ProductCode, unsigned char\* MainVer, unsigned char\* SubVer, unsigned char\* IRStatue, int PortHandle);

15) int GetControllerReaderConnectionStatus(unsigned char \*ConAddr, unsigned char \*ConnectionStatus, unsigned char\* IRStatue, int PortHandle) ;

16) int SetControllerAddr(unsigned char \*ConAddr, unsigned char Flag, unsigned char NewAddr, unsigned char \*IRStatue, int PortHandle);

17) int ModeSwitch(unsigned char \*ConAddr, unsigned char \*Mode, unsigned char \*IRStatue, int PortHandle);

18) int IRDirectionSetting(unsigned char \*ConAddr, unsigned char \*Flag, unsigned char \*IRStatue, int PortHandle);

19)int GetEASMessage (unsigned char \*ConAddr, unsigned char\* Msg, unsigned char\* MsgLength , unsigned char \*MsgType, unsigned char\* IRStatue, int PortHandle);

20)int SetEASWorkStyle (unsigned char \*ConAddr, unsigned char \*EASMode, unsigned char \*IRStatue, int PortHandle);

21)int GetEASWorkStyle (unsigned char \*ConAddr, unsigned char \*EASMode, unsigned char \*IRStatue, int PortHandle);

22) int SetReadParameter(unsigned char \*ComAdr, unsigned char Qvalue, unsigned char Session, unsigned char AdrTID, unsigned char LenTID, unsigned char MaskMem, unsigned char \*MaskAdr, unsigned char MaskLen, unsigned char \*MaskData, unsigned char \*IRStatue, int FrmHandle);

23) int GetReadParameter(unsigned char \*ComAdr, unsigned char \*Qvalue, unsigned char \*Session, unsigned char \*AdrTID, unsigned char \*LenTID, unsigned char \*MaskMem, unsigned char \*MaskAdr, unsigned char \*MaskLen, unsigned char \*MaskData, unsigned char \*IRStatue, int FrmHandle);

24)int StatisticalMsg(unsigned char \*ComAdr, unsigned char \*positive, unsigned char \*reverse, unsigned char \*AlarmNum, unsigned char \*IRStatue, int FrmHandle);

25)int SetWorkParameter(unsigned char \*ComAdr, unsigned char Power, unsigned char MaxFre, unsigned char MinFre, unsigned char BeepEn, unsigned char \*IRStatue, int FrmHandle);

26)int GetWorkParameter(unsigned char \*ComAdr, unsigned char \*Power, unsigned char \*MaxFre, unsigned char \*MinFre, unsigned char \*BeepEn, unsigned char \*IRStatue, int FrmHandle);

27) int SetRelay( unsigned char \* ComAddr, unsigned char RelayTime, unsigned char \*IRStatue, int FrmHandle);

28)int BuzzerAndLEDControl( unsigned char \* ComAddr,unsigned char BuzzerOnTime,unsigned char BuzzerOffTime,unsigned char BuzzerActTimes, unsigned char LEDOnTime,unsigned char LEDOffTime,unsigned char LEDFlashTimes,unsigned char \*IRStatue,int FrmHandle);

## 3. Function Explanation

### 3.1) AutoOpenComPort

#### Function description:

This function is used to automatically detect the communication port unoccupied by other application and attached with a controller. The function tries to establish the connection between them. The protocol parameters are 38400bps, 8 data bits, 1 start bit, 1 stop bit, even 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:

```
int AutoOpenComPort(int * Port, unsigned char *ConAddr, int *PortHandle);
```

#### Parameter:

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

ConAddr: Pointed to the address of the controller.

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

When using a designated address 0x00~0xFE as ConAddr to call the function, the port number to which the controller with the specified address is detected will be written 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
```

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

#### Returns:

Zero value when successful, non-zero value(ErrorCode) when error occurred.

### 3.2) OpenComPort()

**Function description:**

This function is used to establish the connection between the controller and a specified communication port. The protocol parameters are 38400bps, 8 data bits, 1 start bit, 1 stop bit, even parity bit.

**Usage:**

```
int OpenComPort(int Port, unsigned char *ConAddr, int *PortHandle);
```

**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
```

ConAddr: Pointed to the address of the controller.

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

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

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

**Returns:**

Zero value when successful, non-zero value(ErrorCode) when error occurred.

### 3.3) CloseComPort

**Function description:**

This function is used to disconnect the controller 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:**

```
int CloseComPort(void);
```

**Parameter:** None.

**Returns:**

Zero value when successful, non-zero value(ErrorCode) when error occurred.

### 3.4) CloseSpecComPort

**Function description:**

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

**Usage:**

```
int CloseSpecComPort (int PortHandle);
```

**Parameter:**

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

**Returns:**

Zero value when successful, non-zero value(ErrorCode) when error occurred.

### 3.5) OpenNetPort

**Function description:**

The function is used to open net port.

**Usage:**

```
int OpenNetPort(int Port,LPSTR IPaddr, unsigned char *ComAdr, int FrmHandle);
```

**Parameter:**

Port: Pointed to the net port of the reader.

IPaddr: Pointed to string of reader IP.

ComAdr: Pointed to the address of the reader.

FrmHandle: Handle of the corresponding communication port the reader is connected. The handle value is got when calling function OpenNetPort..

**Returns:**

None

### 3.6) CloseNetPort

**Function description:**

The function is used disconnected net port.

**Usage:**

```
int CloseNetPort (int FrmHandle);
```

**Parameter:**

FrmHandle: Handle of the corresponding communication net port the device is connected. The handle value is got when calling function OpenNetPort..

**Returns:**

None

### 3.7) GetChannelMessage

**Function description:**

This function is used to get message information from controller in Channel Mode. The messages include routine message and other alarming or statistic message.

**Usage:**

```
Int GetChannelMessage (unsigned char *ConAddr, unsigned char* Msg, unsigned char* MsgLength, unsigned char *MsgType, unsigned char* IRStatue, int PortHandle);
```

**Parameter:**

ConAddr: Address of the controller.

Msg: Pointed to the message data from controller.

MsgLength: Pointed to the length of Msg.

MsgType: Pointed to the type of Msg information.

(1) MsgType = 0, Routine Response:

The 1<sup>st</sup> to 6<sup>th</sup> bytes of the Msg are the time stamp as year/month/day/hour/minute/second

The 7<sup>th</sup> byte is the number of tag had read

Else (MsgLength-7) bytes are tags information. Length+epc/id for each tag .

(2) MsgType = 1, Auxiliary Response:

The 1st of Msg is flag of direction.

The 2st to 4th bytes of the Msg are number of people forward passed (LSB).

The 5<sup>rd</sup> to 7<sup>th</sup> bytes of the Msg are number of people reversely passed (LSB).

The 8<sup>th</sup> to 11<sup>th</sup> bytes of the Msg are number of alarm(LSB).

The 12<sup>th</sup> to 17<sup>th</sup> bytes of the Msg are the time stamp as year/month/day/hour/minute/second .

IRStatue: Pointed to IR blocking state. bit3-bit0 is effective. bit=1, IR is not normal ,bit=0 IR is normal.

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

**Returns:**

Zero value when successful, non-zero value(ErrorCode) when error occurred.

### 3.8) Acknowledge

**Function description:**

When the host has correctly received the feedback of command 'C', 'L' and 'E', it should issue this command as an acknowledgement.



**Usage:**

Int Acknowledge (unsigned char \*ConAddr, int PortHandle);

**Parameter:**

ConAddr: Address of the controller.

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

**Returns:** None.

### 3.9) SetClock

**Function description:**

This function is used to set the device's built-in real time clock.

**Usage:**

int SetClock(unsigned char \*ConAddr, unsigned char\* SetTime, unsigned char\* IRStatue, int PortHandle);

**Parameter:**

ConAddr: Address of the controller.

SetTime: 6 bytes time stamp for year-month-day-hour-minute-second in 24hour format.

IRStatue: Pointed to IR blocking state. bit3-bit0 is effective. bit=1, IR is not normal ,bit=0 IR is normal.

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

**Returns:**

Zero value when successful, non-zero value(ErrorCode) when error occurred.

### 3.10) GetClock

**Function description:**

This function is used to query controller's clock information

**Usage:**

Int GetClock(unsigned char \*ConAddr, unsigned char\* CurrentTime, unsigned char\* IRStatue, int PortHandle );

**Parameter:**

ConAddr: Address of the controller.

CurrentTime: Pointed to 6 bytes time stamp for year-month-day-hour-minute-second in 24hour format.

IRStatue: Pointed to IR blocking state. bit3-bit0 is effective. bit=1, IR is not normal ,bit=0 IR is normal.

PortHandle: Handle of the corresponding communication port the controller is connected. The handle value is got when calling function AutoOpenComPort or

OpenComPort.

**Returns:**

Zero value when successful, non-zero value(ErrorCode) when error occurred.

### 3.11) ClearControllerBuffer

**Function description:**

This function is used to clear the controller's all buffered tag UID information, messages, personnel passing counter and detected tag counter.

**Usage:**

```
Int ClearControllerBuffer(unsigned char *ConAddr, unsigned char* IRStatue,  
int PortHandle);
```

**Parameter:**

ConAddr: Address of the controller.

IRStatue: Pointed to IR blocking state. bit3-bit0 is effective. bit=1, IR is not normal ,bit=0 IR is normal.

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

**Returns:**

Zero value when successful, non-zero value(ErrorCode) when error occurred.

### 3.12) ConfigureController

**Function description:**

This function is used to configure the controllers' parameters.

**Usage:**

```
int ConfigureController(unsigned char *ConAddr, unsigned char IREnable,  
unsigned char IRTime, unsigned char TagExistTime, unsigned char AlarmEn,  
unsigned char DelayTime, unsigned char Pepolemsg, unsigned char AEn,  
unsigned char* IRStatue, int PortHandle);
```

**Parameter:**

ConAddr: Address of the controller.

IREnable: IR trigger flag , 0-Disable;1-Enable.

IRTime:IR trigger delay time T1. ( $T1 * 1s$ ,  $0 \leq T1 \leq 255$ )

TagExistTime: If the filtering time ( $T2 * 1s$  and  $0 \leq T2 \leq 255$ ) detected many times within the same tag, just upload once EPC/TID.

AlarmEn: Alarm flag , 0-Disable;1-Enable;

DelayTime: EAS mode, the controller of tag eligible for alarm, if can sound and light alarm, according to the set off time ( $T3 * 100ms$ ,  $0 \leq T3 \leq 255$ ) and relay. The default value is 0.

Pepolemsg:Used to set the controller whether produce personnel in and out of the news , 0-Disable;1-Enable;

AEn: 'A' command flag. 0-Disable;1-Enable;

IRStatue: Pointed to IR blocking state. bit3-bit0 is effective. bit=1, IR is not normal ,bit=0 IR is normal.

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

**Returns:**

Zero value when successful, non-zero value(ErrorCode) when error occurred.

### 3.13) GetControllerConfig

**Function description:**

This function is used to get configure parameters of controllers.

**Usage:**

```
int GetControllerConfig(unsigned char *ConAddr, unsigned char *IReEnable,
unsigned char *IRTime, unsigned char *TagExistTime, unsigned char *AlarmEn,
unsigned char *DelayTime, unsigned char *Pepolemsg, unsigned char *AEn,
unsigned char* IRStatue, int PortHandle);
```

**Parameter:**

ConAddr: Address of the controller.

IReEnable: IR trigger flag , 0-Disable;1-Enable.

IRTime:IR trigger delay time T1. ( $T1 * 1s$ ,  $0 \leq T1 \leq 255$ )

TagExistTime: If the filtering time ( $T2 * 1s$  and  $0 \leq T2 \leq 255$ ) detected many times within the same tag, just upload once EPC/TID.

AlarmEn: Alarm flag , 0-Disable;1-Enable;

DelayTime: EAS mode, the controller of tag eligible for alarm, if can sound and light alarm, according to the set off time ( $T3 * 100ms$ ,  $0 \leq T3 \leq 255$ ) and relay. The default value is 0.

Pepolemsg:Used to set the controller whether produce personnel in and out of the news , 0-Disable;1-Enable;

AEn: 'A' command flag. 0-Disable;1-Enable;

IRStatue: Pointed to IR blocking state. bit3-bit0 is effective. bit=1, IR is not normal ,bit=0 IR is normal.

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

**Returns:**

Zero value when successful, non-zero value(ErrorCode) when error occurred.

### 3.14) GetControllerInfo

**Function description:**

This function is used to get controller's information.

**Usage:**

```
int GetControllerInfo(unsigned char *ConAddr, unsigned char* ProductCode,
unsigned char* MainVer, unsigned char* SubVer, unsigned char* IRStatue, int
PortHandle);
```

Function Description: Infrared controller to obtain detailed information

**Parameter:**

ConAddr: Address of the controller.

ProductCode: Pointed to controller's product code.

0x90: RRU-CH-WL;

0x91: RRU-CH-C16058。

MainVer: Pointed to the main version number of the controller

SubVer: Pointed to the sub-version number of the controller

IRStatue: Pointed to IR blocking state. bit3-bit0 is effective. bit=1, IR is not normal ,bit=0 IR is normal.

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

**Returns:**

Zero value when successful, non-zero value(ErrorCode) when error occurred.

### 3.15) GetControllerReaderConnectionStatus

**Function description:**

This function is used to check whether the reader is correctly connected with the controller.

**Usage:**

```
int GetControllerReaderConnectionStatus(unsigned char *ConAddr, unsigned
char *ConnectionStatus, unsigned char* IRStatue, int PortHandle) ;
```

**Parameter:**

ConAddr: Address of the controller.

ConnectionStatus: Pointed to connection status of the controller and the reader

0x00: the connection is broken and the controller is trying to reconnect with the reader.

0x01: the connection is normal.

IRStatue: Pointed to IR blocking state. bit3-bit0 is effective. bit=1, IR is not normal ,bit=0 IR is normal.

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

**Returns:**

Zero value when successful, non-zero value(ErrorCode) when error occurred.

**3.16) SetControllerAddr****Function description:**

This function is used to set or get the controller's address.

**Usage:**

```
int SetControllerAddr(unsigned char *ConAddr, unsigned char Flag, unsigned char NewAddr, unsigned char * IRStatue, long PortHandle);
```

**Parameter:**

ConAddr: Address of the controller.

Flag: Operation flag

Flag = 0: Get the controller address;

Flag = 1: Set the controller address.

NewAddr: New controller's address. Range is 0~254.

IRStatue: Pointed to IR blocking state. bit3-bit0 is effective. bit=1, IR is not normal ,bit=0 IR is normal.

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

**Returns:**

Zero value when successful, non-zero value(ErrorCode) when error occurred.

**3.17) ModeSwitch****Function description:**

This function is used to view or change the work mode of the controller. The controller supports two work modes: Inventory Mode and EAS Mode.

**Usage:**

```
int ModeSwitch(unsigned char *ConAddr, unsigned char *Mode, unsigned char * IRStatue, long PortHandle);
```

**Parameter:**

ConAddr: Address of the controller.

Mode: Work mode:

Bit1	Bit0	Work mode
0	0	Inventory Mode
0	1	EAS Mode

bit7 = 0, read current work mode.,

bit7 = 1, Set current work mode according to bit0&bit1.

IRStatue: Pointed to IR blocking state. bit3-bit0 is effective. bit=1, IR is not normal ,bit=0 IR is normal.

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

**Returns:**

Zero value when successful, non-zero value(ErrorCode) when error occurred.

### 3.18) IRDirectionSetting

**Function description:**

This function is used to query or set the infrared sensors' sensing sequences.

**Usage:**

Int IRDirectionSetting(unsigned char \*ConAddr, unsigned char \*Flag, unsigned char \*FreeSpaceRate, long PortHandle);

**Parameter:**

ConAddr: Address of the controller.

Flag: Input/output variable.

When as input:

bit0 = 0: forward infrared sensor sequence.

bit0 = 1: reversed infrared sensor sequence.

bit7 = 0: query operation. Bit0 value should be neglected.

bit7 = 1: set operation. The infrared sensor sequence will be set as bit0 defined.

Other bits are reserved.

When as output:

bit0=0: forward infrared sensor sequence.

bit0=1: reversed infrared sensor sequence.

Other bits are reserved.

IRStatue: Pointed to IR blocking state. bit3-bit0 is effective. bit=1, IR is not normal ,bit=0 IR is normal.

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

**Returns:**

Zero value when successful, non-zero value(ErrorCode) when error occurred.

### 3.19) GetEASMessage

**Function description:**

This function is used to get person-passing message, EAS alarm message and other statistic or alarm messages of controller in EAS mode.

**Usage:**

Int GetEASMessage (unsigned char \*ConAddr, unsigned char\* Msg, unsigned char\* MsgLength , unsigned char \*MsgType, unsigned char\* IRStatue, int PortHandle);

**Parameter:**

ConAddr: Address of the controller.

Msg: Pointed to the message information data access from controller.

MsgLength: Pointed to the length of Msg.

MsgType: Pointed to the type of information.

(1) MsgType = 0, routine response message:

The 1<sup>st</sup> bytes of the Msg is the direction data.

The 2<sup>nd</sup> to 7<sup>th</sup> bytes are the time stamp as year/month/day/hour/minute/second.

(2) MsgType = 1, statistic Message:

The 1<sup>st</sup> bytes of the Msg is the direction data.

The 2<sup>st</sup> to 4<sup>th</sup> bytes of the Msg are number of people forward passed (LSB).

The 5<sup>th</sup> to 6<sup>th</sup> bytes of the Msg are number of people reversely passed (LSB).

The 8<sup>th</sup> to 11<sup>th</sup> bytes of the Msg are number of people passed in uncertain direction (LSB).

The 12<sup>th</sup> to 17<sup>th</sup> bytes are the time stamp as year/month/day/hour/minute/second.

(3) MsgType = 2, EPC response Message:

The 1<sup>th</sup> byte is 1.

The 2<sup>th</sup> to 7<sup>th</sup> bytes are the time stamp as year/month/day/hour/minute/

Second , else bytes are epc, length is MsgLength-7

IRStatue: Pointed to IR blocking state. bit3-bit0 is effective. bit=1, IR is not normal ,bit=0 IR is normal.

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

**Returns:**

Zero value when successful, non-zero value(ErrorCode) when error occurred.

**3.20) Set EAS work style****Function description:**

This function is used to set EAS detect mode .

**Usage:**

int SetEASWorkStyle (unsigned char \*ConAddr, unsigned char \*EASMode,

unsigned char \*IRStatue, int PortHandle);

**Parameter:**

ConAddr: Address of the controller.

EASMode: 2 byte.

① Configure byte. Only bit0 and bit4 are valid. Other bits are reserved and should be 0.

bit0=0, standard EAS detection enabled and is the default setting.

bit0=1, emulate EAS alarm enabled.

bit4=0, when emulate EAS alarm enabled and EAS alarm detected, not return EPC

bit4=1, when emulate EAS alarm enabled and EAS alarm detected, return EPC.

②emulate Type:

Valid at the time of simulation EAS alarm Settings.

0: when the detection to the label of the EPC number 92-93 bit values of 01 alarm, when other values do not call the police.

1: when the detection to the label of EPC area the first word of highest alarm when equal to zero, equal to 1 does not report to the police.

2: alarming detected the label with any EPC number, otherwise don't call the police.

Other values, the default is 0

IRStatue: Pointed to IR blocking state. bit3-bit0 is effective. bit=1, IR is not normal, bit=0 IR is normal.

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

**Returns:**

Zero value when successful, non-zero value(ErrorCode) when error occurred.

### 3.21) Get EAS work style

**Function description:**

This function is used to get EAS detect mode .

**Usage:**

int GetEASWorkStyle (unsigned char \*ConAddr, unsigned char \*EASMode, unsigned char \*IRStatue, int PortHandle);

**Parameter:**

ConAddr: Address of the controller.

EASMode: 2 byte

① Configure byte. Only bit0 and bit4 are valid. Other bits are reserved and should be 0.

bit0=0, standard EAS detection enabled and is the default setting.

bit0=1, emulate EAS alarm enabled.

bit4=0, when emulate EAS alarm enabled and EAS alarm detected, not return EPC

bit4=1, when emulate EAS alarm enabled and EAS alarm detected, return EPC.

②emulate Type:

Valid at the time of simulation EAS alarm Settings.



0: when the detection to the label of the EPC number 92-93 bit values of 01 alarm, when other values do not call the police.

1: when the detection to the label of EPC area the first word of highest alarm when equal to zero, equal to 1 does not report to the police.

2: alarming detected the label with any EPC number, otherwise don't call the police.

Other values, the default is 0

IRStatue: Pointed to IR blocking state. bit3-bit0 is effective. bit=1, IR is not normal, bit=0 IR is normal.

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

**Returns:**

Zero value when successful, non-zero value(ErrorCode) when error occurred.

### 3.22) set read parameter

**Function description:**

This function is used to set parameter when inventory tag.

**Usage:**

```
int SetReadParameter(unsigned char *ComAdr, unsigned char Qvalue, unsigned char Session, unsigned char AdrTID, unsigned char LenTID, unsigned char MaskMem, unsigned char *MaskAdr, unsigned char MaskLen, unsigned char *MaskData, unsigned char *IRStatue, int FrmHandle);
```

**Parameter:**

ConAddr: Address of the controller.

Qvalue: input, 2<sup>Q</sup> closed to tag number is best.

Session: input,

0x00: S0

0x01: S1;

0x02: S2;

0x03: S3。

0xff: AUTO(only EPC query is effective).

AdrTID: the start address of tid query.

LenTID: the length of tid query. if LenTID=0, query EPC.

MaskMem: mask memory. 0x01: EPC; 0x02: TID; 0x03: User.

MaskAdr: 2bytes, mask start bit address.

MaskLen: the bit length of mask. if MaskLen=0, not mask.

MaskData: the data of mask, length is MaskData/8.

IRStatue: Pointed to IR blocking state. bit3-bit0 is effective. bit=1, IR is not normal, bit=0 IR is normal.

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

**Returns:**

Zero value when successful, non-zero value(ErrorCode) when error occurred.

### 3.23) Get read parameter

**Function description:**

This function is used to get parameter when inventory tag.

**Usage:**

```
int GetReadParameter(unsigned char *ComAdr, unsigned char *Qvalue,
unsigned char *Session, unsigned char *AdrTID, unsigned char *LenTID,
unsigned char *MaskMem, unsigned char *MaskAdr, unsigned char *MaskLen,
unsigned char *MaskData, unsigned char *IRStatue, int FrmHandle);
```

**Parameter:**

ConAddr: Address of the controller.

Qvalue: input, 2<sup>Q</sup> closed to tag number is best.

Session: input,

0x00: S0

0x01: S1;

0x02: S2;

0x03: S3。

0xff: AUTO(only EPC query is effective).

AdrTID:the start address of tid query.

LenTID:the length of tid query.if LenTID=0,query EPC.

MaskMem: mask memery.0x01:EPC; 0x02: TID; 0x03:User.

MaskAdr: 2bytes, mask start bit address.

MaskLen: the bit length of mask. if MaskLen=0,not mask.

MaskData: the data of mask ,length is MaskData/8.

IRStatue: Pointed to IR blocking state. bit3-bit0 is effective. bit=1, IR is not normal ,bit=0 IR is normal.

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

**Returns:**

Zero value when successful, non-zero value(ErrorCode) when error occurred.

### 3.24) Get statistical message

**Function description:**

This function is used to get statistical message.

**Usage:**

```
int StatisticalMsg(unsigned char *ComAdr, unsigned char *positive, unsigned char *reverse, unsigned char *AlarmNum, unsigned char *IRStatue, int FrmHandle);
```

**Parameter:**

ConAddr: Address of the controller.

positive: 3 bytes, number of people forward passed.

reverse: 3 bytes, number of people reversely passed.

AlarmNum: 4 bytes, number of alarm.

IRStatue: Pointed to IR blocking state. bit3-bit0 is effective. bit=1, IR is not normal, bit=0 IR is normal.

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

**Returns:**

Zero value when successful, non-zero value(ErrorCode) when error occurred.

### 3.25) Ser reader work paramater

**Function description:**

This function is used to set reader work parameters.

**Usage:**

```
int SetWorkParameter(unsigned char *ComAdr, unsigned char Power, unsigned char MaxFre, unsigned char MinFre, unsigned char BeepEn, unsigned char *IRStatue, int FrmHandle);
```

**Parameter:**

ConAddr: Address of the controller.

Power: reader power, range is 0-30, 30dBm is 1W.

MaxFre: bit7-bit6 is set Spectrum, bit5-bit0 is set maximum frequency.

MinFre: bit7-bit6 is set Spectrum, bit5-bit0 is set minimum frequency.

MaxFre(Bit7)	MaxFre(Bit6)	MinFre(Bit7)	MinFre(Bit6)	FreqBand
0	0	0	0	Reserve
0	0	0	1	Chinese band2
0	0	1	0	US band
0	0	1	1	Korean band

0	1	0	0	EU band
0	1	0	1	Reserve
...	...	...	...	...
1	1	1	1	Reserve

Chinese band2:  $F_s = 920.125 + N * 0.25$  (MHz)  $N \in [0, 19]$ 。

US band:  $F_s = 902.75 + N * 0.5$  (MHz)  $N \in [0, 49]$ 。

Korean band:  $F_s = 917.1 + N * 0.2$  (MHz)  $N \in [0, 31]$ 。

EU band:  $F_s = 865.1 + N * 0.2$  (MHz)  $N \in [0, 14]$ 。

BeepEn: Reader whether beep when the tag is prompt, bit0 = 0: Disabled. (the default), bit0 = 1: enabled.

IRStatue: Pointed to IR blocking state. bit3-bit0 is effective. bit=1, IR is not normal, bit=0 IR is normal.

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

#### Returns:

Zero value when successful, non-zero value(ErrorCode) when error occurred.

### 3.26) Ger reader work paramater

#### Function description:

This function is used to Get reader work parameters.

#### Usage:

```
int GetWorkParameter(unsigned char *ComAdr, unsigned char *Power,
unsigned char *MaxFre, unsigned char *MinFre, unsigned char *BeepEn,
unsigned char *IRStatue, int FrmHandle);
```

#### Parameter:

ConAddr: Address of the controller.

Power: reader power, range is 0-30,30dBm is 1W.

MaxFre: bit7-bit6 is set Spectrum, bit5-bit0 is set maximum frequency.

MinFre: bit7-bit6 is set Spectrum, bit5-bit0 is set minimum frequency.

MaxFre(Bit7)	MaxFre(Bit6)	MinFre(Bit7)	MinFre(Bit6)	FreqBand
0	0	0	0	Reserve
0	0	0	1	Chinese band2
0	0	1	0	US band
0	0	1	1	Korean band
0	1	0	0	EU band
0	1	0	1	Reserve
...	...	...	...	...
1	1	1	1	Reserve

Chinese band2:  $F_s = 920.125 + N * 0.25$  (MHz)  $N \in [0, 19]$ 。

US band:  $F_s = 902.75 + N * 0.5$  (MHz)  $N \in [0, 49]$ 。

Korean band:  $F_s = 917.1 + N * 0.2$  (MHz)  $N \in [0, 31]$ 。

EU band:  $F_s = 865.1 + N * 0.2$  (MHz)  $N \in [0, 14]$ 。

BeepEn: Reader whether beep when the tag is prompt, bit0 = 0: Disabled. (the default), bit0 = 1: enabled.

IRStatue: Pointed to IR blocking state. bit3-bit0 is effective. bit=1, IR is not normal ,bit=0 IR is normal.

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

**Returns:**

Zero value when successful, non-zero value(ErrorCode) when error occurred.

### 3.27) SetRelay

**Function description:**

This function is used to control the device's built-in relay to pick-up, last for a requested time and drop out.

**Usage:**

int SetRelay( unsigned char \* ComAddr, unsigned char RelayTime, unsigned char \*IRStatue, int FrmHandle);

**Parameter:**

ConAddr: Address of the controller.

RelayTime: the relay's pick-up duration is Time\*100ms,  $0 \leq \text{Time} \leq 255$ .

IRStatue: Pointed to IR blocking state. bit3-bit0 is effective. bit=1, IR is not normal ,bit=0 IR is normal.

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

**Returns:**

Zero value when successful, non-zero value(ErrorCode) when error occurred.

### 3.28) BuzzerAndLEDControl

**Function description:**

This function is used to control the device's built-in LED & Buzzer to flash & beep.

**Usage:**

int BuzzerAndLEDControl( unsigned char \* ComAddr, unsigned char

BuzzerOnTime,unsigned char BuzzerOffTime,unsigned char BuzzerActTimes,  
unsigned char LEDOnTime,unsigned char LEDOffTime,unsigned char  
LEDFlashTimes,unsigned char \*IRStatue,int FrmHandle);

**Parameter:**

ConAddr: Address of the controller.

BuzzerOnTime: Buzzer beep duration ( $T1 \times 100\text{ms}$ ),  $0 \leq T1 \leq 255$ .

BuzzerOffTime: Buzzer mute duration ( $T2 \times 100\text{ms}$ ),  $0 \leq T2 \leq 255$ .

BuzzerActTimes: Buzzer action times ( $0 \leq T3 \leq 255$ ).

LEDOnTime: LED light on duration ( $T4 \times 100\text{ms}$ ),  $0 \leq T4 \leq 255$ .

LEDOffTime: LED light off duration ( $T5 \times 100\text{ms}$ ),  $0 \leq T5 \leq 255$

LEDFlashTimes: LED flash times ( $0 \leq T6 \leq 255$ );

IRStatue: Pointed to IR blocking state. bit3-bit0 is effective. bit=1, IR is not normal ,bit=0 IR is normal.

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

**Returns:**

Zero value when successful, non-zero value(ErrorCode) when error occurred.

## 4. Appendix 1

**List of Command Returned value:**

Value	COMMENT
0x00	Operation succeed
0x08	The command is invalid.
0x09	The current mode of the controller is incorrect.
0x0E	EEPROM operation error
0x30	Communication error.
0x31	CRC Check error.
0x32	Length of returned data error.
0x33	Communication Busy.
0x0F	General error.