ClouReaderAPI_CPP interface documentation

Contents

1 Range

This article describes the ClouReaderAPI interface of C language version .

2 Call specification

```
#define CLAPI __stdcall
```

3 Data structure

3.1 Antenna

Type: enumeration

Source code definition:

```
enum CL_ANT
{//Antenna
	ANT_1 = 0x01,
	ANT_2 = 0x02,
	ANT_3 = 0x04,
	ANT_4 = 0x08
};
```

Description: The function of API needs to get the parameter of the antenna to use this type, you can use a single antenna, you can also use the combination of methods, such as ANT_1|ANT_2 or ANT_1|ANT_3|ANT_4 any combination, if not supported by bit (or | Operation), you can also use + instead.

3.2 Reading type

Type: enumeration

Source code definition:

```
enum CL_READ_TYPE
{
	SINGLE_READ = 0x00,	//single read
	CONSTANT_READ = 0x01//read continuously
};
```

Explanation: Assign reader to read the tag mode.

3.3 Data area

Type: enumeration

Source code definition:

```
enum CL_DATA_FIELD
{
        EPC_FIELD = 0x01,    //EPC data area
        TID_FIELD = 0x02,    //TID data area
        USER_FIELD = 0x03, //User data area
        HOLD_FIELD = 0x04,        //reserved data area
        TID_USER = 0x05,     //TID and user data area
        TID_HOLD = 0x06,   //TID and reserved data area
        USER_HOLD = 0x07,//User and reserved data area
        TID_USER_HOLD = 0x08  //TID, User and reserved data area
};
```

Explanation: Assign the data area to be manipulated or matched.

3.4 Tag data

Type: Structure

Source code definition:

```
//The tag data that has been read
struct CL_TAG
{
        int TagType;      //0:6C, 1:6B, -1:End of tag reading notification
        char ReaderName[1024];  //Reader name
        int ReaderName_len;                 //the length of reader name
        char EPC[1024];//EPC data
        int EPC_len;       //the length of EPC data
        char TID[1024]; //TID data
        int TID_len;       //the length of TID data
        char UserData[1024];//user data
        int UserData_len;               //the length of user data
        char PC[10];       //PC value
        int PC_len;                 //the length of PC value
        unsignedchar ANT_NUM; //antenna number which detected tag
        unsignedchar RSSI;   //tag RSSI value
        unsignedchar Result;//the value means the read result if tag read command includes read TID,
user area or reserved area data parameter
        char TagetData[1024];       //reserved area data
        int TagetData_len;     //the length of reserved area data
};
```

Note: Including the return tag data

3.5 Callback Function

Type: Function Pointer

Source code definition: `typedef void (CLAPI *CL_CALLBACK)(CL_TAG*);`

Note: The application (API user) to implement a function of the same type, then the function pointer passed to API by open the connection interface, the reading tag data will be returned by the callback function to the application.

3.6 Returned code

Type: Integer

Source code definition:

```
#define    CL_SUCCESS    0      //success
#define    CL_OTHER      -1     //other error
#define CL_TIMEOUT       -2     //operation timeout
#define CL_PARAM  -3     //parameter error
//return value>0, internal error, please check the communication protocol for more detailed error information
```

4 Interface

4.1 Test Call

Definition: `char* CLAPI CLTest();`

Parameter: None

Return Value:String

Instruction: Simply return a string to facilitate testing whether API can pass

Eg:

```
int main()
```

```
{
    printf("%s\n", CLTest());
    return 0;
}
```

4.2 Test Callback

Definition：void CLAPI CLTestCallBack(CL_CALLBACK clCall);

Parameter: Callback function pointer,details refer to 3.5

Return Value:None

Instruction: Through tag's data structure ReaderName to return a test string, convenient test API callback function can be used normally

Eg:

```
void CLAPI TestCallBack(CL_TAG* tm)
{
        if (tm->ReaderName != NULL)
        {
                printf("%s\n", tm->ReaderName);
        }
}

int main()
{
    CLTestCallBack(&testCallBack);
    return 0;
}
```

4.3 Open the reader connection（TCP）

Definition：int CLAPI CLOpenTcpConnect(char* connParam, CL_CALLBACK clCall);

Parameter: connParam:Connection parameter,is the reader's identity

Parameter: clCall:Callback function pointer, details refer to 3.5

Return Value:details refer to 3.6

Eg:

```
void CLAPI CClouReaderDemoDlg::TagCallBack(CL_TAG* tag)
{}
```

```
int main()
{
        int ret = 0;
        char* connParam = "192.168.1.116:9090";
        ret = CLOpenTcpConnect(connParam, clCallBack);
        if (ret == CL_SUCCESS)
        {
                printf("connect reader %s success.\n", connParam);
        }
        return 0;
}
```

4.4 Open the reader connection(Serial Port)

Definition： int CLAPI CLOpenSerialConnect(char* connParam, CL_CALLBACK clCall);

Parameter: Ditto(4.3)

Return Value: Ditto(4.3)

Eg:har* connParam = "COM1:115200"; other ditto（4.3）

4.5 Read Tags

Definition： int CLAPI CLReadEPC(char* connParam, CL_ANT ant, CL_READ_TYPE readType, CL_DATA_FIELD field=EPC_FIELD, char* pwd=NULL, char* match=NULL, int* startPos=NULL, size_t start_len=0,int* readLen=NULL, size_t read_Len=0);

- Parameter: connParam , reader identity
- Parameter: ant, Antenna Number,details refer to 3.1
- Parameter: readtype, details refer to 3.2
- Parameter： field – selected reading data area ,details refer to 3.3, Note: no matter which area to choose,EPC will be read and return .
- Parameter： pwd – access password of tag， must be 8 Hexadecimal numbers .
- Parameter： match—matching parameter ,only matched tag can be read ,format "matched data area, matched starting position, matched length, matched string";

  matched data area: 1:EPC,2:TID,3:UserData;

  matched starting position : decimal numbers;

  matched length: the length of matched string*4;

  matched string;

e.g.： “1,0,16,ABCD”

- Parameter： startPos, int array with the length of 4, respectively correspond to reading start position of EPC,TID,UserData, reserved area data
- Parameter： start_len, length of array startPos, should be 4, or greater than 4
- Parameter： readLen, int array with the length of 4, respectively correspond to reading length of EPC,TID,UserData, reserved data area
- Parameter： read_Len, length of array readLen, should be 4, or greater than 4

Return Value： Details refer to 3.6

Eg：

```
void CLAPI CClouReaderDemoDlg::TagCallBack(CL_TAG* tag)
{
    if (tag->TagType == -1)
      printf("Received the notice of stop reading tag\n");

if (tag->TagType == 0)
{
    if (tag->EPC_len >0)
        printf("EPC readed: %s\n", tag->EPC);
if (tag->TID_len > 0)
    printf("TID readed: %s\n", tag->TID);
if (tag->UserData_len > 0)
            printf("UserData readed: %s\n", tag->UserData);
if (tag->Target_len > 0)
            printf("reserved area data readed: %s\n", tag->Target);
    }
}
int main()
{
    int ret = 0;
    char* connParam = "192.168.1.116:9090";
    ret = CLOpenTcpConnect(connParam, clCallBack);
    if (ret == CL_SUCCESS)
    {
        printf("reader connection %s succeeded.\n", connParam);
        ret = CLReadEPC(connParam, ANT_1, SINGLE_READ);
        //ret = CLReadEPC(connParam, ANT_1, SINGLE_READ,TID_FIELD); // read TID
//ret = CLReadEPC(connParam, ANT_1, SINGLE_READ,TID_USER_HOLD); //read all data area
        if (ret == CL_SUCCESS)
            printf("EPC single reading succeeded by antenna1.\n");
        ret = CLReadEPC(connParam, ANT_1, CONSTANT_READ);
        if (ret == CL_SUCCESS)
            printf("EPC continuously reading succeeded by antenna 1.\n");
        Sleep(3000); // read 3 seconds
        ret = CLStopRead(connParam);
        if (ret == CL_SUCCESS)
            printf("stop reading succeeded.\n");
        ret = CLCloseConnect(connParam);
        if (ret == CL_SUCCESS)
            printf("reader disconnection succeeded.\n");
}
return 0;
}
```

4.6 Stop reading tags

Definition: int CLAPI CLStopRead(char* connParam);

Parameter: connParam – reader identity

Return Value: details refer to 3.6

Note： when reader continually read tags, use this interface to stop reading, and after opening the connection , you can ascertain if the reader response the instruction normally by this interface.

Eg: details refer to 4.5

4.7 Write Tag

Definition： int CLAPI CLWriteEPC((char* connParam, CL_ANT ant, CL_DATA_FIELD writeField,
char* writeData, CL_DATA_FIELD matchField, char* matchData,char* pwd=NULL,
unsignedshort writeStart=0, unsignedshort matchStart=0, unsignedchar block=0);

- Parameter： connParam – Reader identity
- Parameter： ant –Antenna NO., details refer to 3.1
- Parameter： writeField – The data area will be written,details refer to 3.3
- Parameter：writeData –The data will be written, must be hexadecimal string, if the length is not the multiple of 4, add "0" in the end automatically.
- Parameter： matchField – Select matched area then assign which tag to write ,datails refer to 3.3
- Parameter： matchData- Matched data
- Parameter： pwd - access password of tag, must be 8 Hexadecimal numbers.

- Parameter：writeStart – starting position of writing, decimal number.
- Parameter：matchStart – matched starting position , decimal number.
- Parameter：block – block write，length of single block write, 0 means not using block write

Return value：refer to 3.6

Eg：

```
//write  “BBBB”to EPC area of the tag which EPC is  “AAAA”
ret = CLWriteEPC(connParam, ANT_1, EPC_FIELD,"BBBB", EPC_FIELD,"AAAA");
//Write matched on TID
ret = CLWriteEPC(connParam, ANT_1, EPC_FIELD,"BBBB",TID_FIELD,”E2006003175E9B42");
//Write user area data
ret = CLWriteEPC(connParam, ANT_1, USER_FIELD,"12345678",TID_FIELD,”E2006003175E9B42");
//with password
ret = CLWriteEPC(connParam, ANT_1, EPC_FIELD,"BBBB", EPC_FIELD,"AAAA",“00000000”);
```

4.8 Query antenna RF output power

Definition：`int CLAPI CLGetReaderPower(char* connParam, int* powerBuf, size_t bufLen);`

Parameter：connParam - Reader identity

Parameter： powerBuf – Power value will be configured, should be int array of length 4; if interface returns correct, will be corresponding to the power of antenna 1- antenna4; if -1 ,means no corresponding antenna .

Parameter： bufLen - Length of array powerBuf, this parameter is passed in to prevent memory misoperation.

Return value：refer to 3.6

Eg：

```
int powerBuf[4];
ret = CLGetReaderPower(connParam, powerBuf, 4);
for (int i=1; i<=4; ++i)
{
	if (powerBuf[i] != -1)
		printf("the power of antenna %d is %d\n", i, powerBuf[i]);
	else
		printf("no antenna %d\n", i);
```

4.9 Configure antenna RF output power

Definition： int CLAPI CLSetReaderPower(char* connParam, int* powerBuf, size_t bufLen);

Parameter： connParam- Reader identity

Parameter： powerBuf – Power value to be configured, should be int array of length 4， corresponding to the power of antenna 1- antenna4, if don’t want to configure some antenna,only need to set it to “-1” in corresponding position.

Parameter： bufLen - length of array powerBuf, this parameter is passed in to prevent memory misoperation.

Eg：

```
//configure the power of antenna 1 and 2 to 23
int powerBuf[4];
powerBuf[0] = 23;
powerBuf[1] = 23;
powerBuf[2] = -1;
powerBuf[3] = -1;
ret = CLSetReaderPower(connParam, powerBuf, 4);
```

4.10 Close Reader Connection

Definition： int CLAPI CLAPI CLCloseConnect(char* connParam);

Parameters： connParamreader identity

Return value: see 3.6

Example:

```
ret = CLCloseConnect(connParam);
```

4.11 Close All Readers Connection

Definition： int CLAPI CLCloseAllConnect();

Parameters: None

Return value: see 3.6

Description: API supports max. 10 readers simultaneous connection, could close all the reader connection via this interface

Example:

```
ret = CLCloseAllConnect();
```