

PullSDK 接口使用指南

版本: 2.0

日期: 2015 年 07 月

内容介绍

本文档主要介绍了 PullSDK 接口的概述和技术说明。

目录

1. PULLSDK 接口概述	1
2. PULLSDK 接口技术说明	1
3. PULLSDK 接口安装	1
4. PULLSDK 接口函数详述	2
4.1 CONNECT.....	2
4.2 CONNECTEXT	2
4.3 DISCONNECT.....	3
4.4 SETDEVICEPARAM.....	3
4.5 GETDEVICEPARAM.....	4
4.6 CONTROLDEVICE	4
4.7 SETDEVIDATA	5
4.8 GETDEVIDATA	6
4.9 GETDEVICEDATACOUNT	7
4.10 DELETEDEVIDATA.....	7
4.11 GETRTLOG.....	9
4.12 GETRTLOGEXT	9
4.13 SEARCHDEVICE	10
4.14 MODIFYIPADDRESS.....	11
4.15 PULLLASTERROR	11
4.16 SETDEVICEFILEDATA	12
4.17 GETDEVICEFILEDATA	13
4.18 PROCESSBACKUPDATA.....	13
4.19 BUFFERTOPROTOCOLDATA	14
4.20 PROTOCOLDATATOBUFFER	14
5.1 附表一：接口文件的详细说明	15
5.2 附表二：控制器参数的说明	15
5.3 附表三：CONTROLDEVICE 参数说明.....	18
5.4 附表四：功能函数的表结构说明	18
5.5 附表五：返回值错误码说明	21
5.6 附表六：事件类型及代码表	23
5.7 附表七：GETRTLOG 函数中参数 BUFFER 返回数据格式说明.....	27
5.7 附表八：GETRTLOGEXT 函数中参数 BUFFER 返回数据格式说明.....	28

1. PullSDK 接口概述

PullSDK 接口是一组功能函数，对 C3、inBIO 系列门禁控制器进行数据访问的通讯接口。PullSDK 使最终应用程序开发用户在访问门禁控制器时，更直观、更方便、更简洁，其接口完成的功能有：

- 读取、设置控制器参数；
- 读取、设置、删除控制器的时间段、用户信息、假日信息、指纹等数据；
- 搜索在线设备、修改设备 IPAddress；
- 解析固件备份在 SD 卡里的文件。

2. PullSDK 接口技术说明

PullSDK 接口使最终应用程序开发用户看起来是一组对门禁控制器内数据 Set/Get 的抽象接口，在具体访问设备用户数据时，类似于在使用最通用的 SQL 语句。PullSDK 接口在应用程序开发用户眼中看起来更像一个数据库服务器。

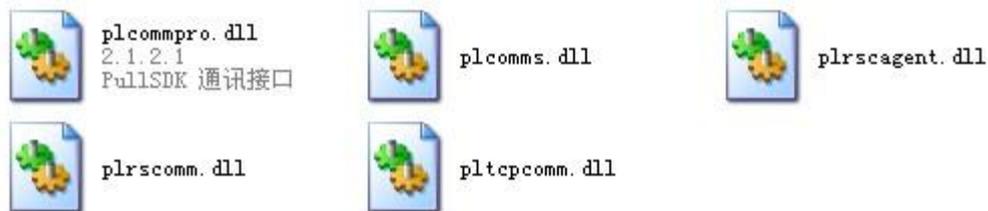
PullSDK 接口支持 TCP/IP 和 RS485 通信协议，并支持 UDP 广播方式搜索门禁控制器及修改门禁控制器的 IP 地址、子网掩码、网关。

PullSDK 接口使用 C 语言开发，对数据通讯做了高度优化，使其成为一个简洁高效的访问接口。

PullSDK 接口在设计开始参考了 SQL，但更考虑到业务中最常用的模型，是精心提炼出来的一组接口，做到了设计、实现、使用三方的平衡。

3. PullSDK 接口安装

PullSDK 接口函数存在于文件 plcommpro.dll 中，该库依赖于其他几个文件，需要将下列五个 DLL 文件一同拷贝到 windows 下的系统目录（32 位操作系统下为 windows/system32，64 位操作系统下为 windows/syswow64）。文件拷贝完毕后，**无需进行注册**，直接按照下述接口函数使用即可。



（注：每个文件功能描述见附表一）

4. PullSDK 接口函数详述

4.1 Connect

[函数]

int Connect(const char *Parameters)

[功能]

连接设备，连接成功后返回连接句柄。

[参数说明]

Parameters:

[in] 通过 Parameter 参数指定连接选项，如下列例子：

"protocol=RS485,port=COM2,baudrate=38400bps,deviceid=1,timeout=50000,passwd=";

"protocol=TCP,ipaddress=192.168.12.154,port=4370,timeout=4000,passwd=";

需要向该函数传递与设备相关的连接参数方可实现连接功能。

protocol 是通讯使用的协议，目前有 RS485 和 TCP 两种方式；

port: 设备通讯端口。例如，以 RS485 方式连接，可设置 port 为 COM1；以 TCP 通讯的端口，如果不特别强调，port 默认为 4370；

deviceid: 串口使用的设备 RS485 通讯地址；

baudrate: 串口通讯使用的波特率；

ipaddress: TCP/IP 通讯相关设备的 IP 地址；

timeout: 连接超时时间，单位为毫秒。遇到网络连接质量不好时，应加大 timeout 的值。一般的，“timeout=5000”（5 秒）可以满足基本网络使用；当查询数据经常出现-2 错误码时，应加大 timeout 的值，可以设置：“timeout=20000”（20 秒）。

passwd: 设置通讯的连接密码，可以为空表示不使用密码。

（注：Parameters 连接字符串大小写敏感）

[返回值]

与设备连接成功后返回该连接句柄，否则连接失败返回 0。

[示例]

Python:

```
params = "protocol=TCP,ipaddress=192.168.12.154,port=4370,timeout=4000,passwd="
```

```
self.commpro = windll.LoadLibrary("plcommpro.dll")
```

```
con_str = create_string_buffer(params)
```

```
self.hcommpro = self.commpro.Connect(con_str)
```

c#:

```
params = "protocol=TCP,ipaddress=192.168.12.154,port=4370,timeout=2000,passwd=";
```

```
IntPtr h = Connect(params);
```

4.2 ConnectExt

[函数]

int ConnectExt(const char *Parameters, int *pErrorCode)

[功能]

连接设备，连接成功后返回连接句柄，连接失败则通过 *pErrorCode 来返回错误码。

[参数说明]

Parameters:

[in] 通过 Parameter 参数指定连接选项，如下列例子：

"protocol=RS485,port=COM2,baudrate=38400bps,deviceid=1,timeout=50000,passwd=";

"protocol=TCP,ipaddress=192.168.12.154,port=4370,timeout=4000,passwd=";

需要向该函数传递与设备相关的连接参数方可实现连接功能。

protocol 是通讯使用的协议，目前有 RS485 和 TCP 两种方式；

port: 设备通讯端口。例如，以 RS485 方式连接，可设置 port 为 COM1；以 TCP 通讯的端口，如

果不特别强调，port 默认为 4370;

deviceid: 串口使用的设备 RS485 通讯地址;

baudrate: 串口通讯使用的波特率;

ipaddress: TCP/IP 通讯相关设备的 IP 地址;

timeout: 连接超时时间，单位为毫秒。遇到网络连接质量不好时，应加大 timeout 的值。一般的，“timeout=5000”（5 秒）可以满足基本网络使用；当查询数据经常出现-2 错误码时，应加大 timeout 的值，可以设置：“timeout=20000”（20 秒）。

passwd: 设置通讯的连接密码，可以为空表示不使用密码。

（注：Parameters 连接字符串大小写敏感）

pErrorCode:

[in] 通过 pErrorCode 参数来得到连接失败后的错误码，错误码说明见附表五。使用出参返回改善了 LastError 机制的线程安全性。

[返回值]

与设备连接成功后返回该连接句柄，否则连接失败返回 0。

[示例]

4.3 Disconnect

[函数]

Void Disconnect(HANDLE handle)

[功能]

断开与设备的连接。

[参数说明]

handle

[in] 由 Connect()成功返回的句柄。

[返回值]

无

[示例]

Python:

```
self.commpro.Disconnect(self.hcommpro)
self.hcommpro = 0
```

c#:

```
Disconnect(h);
```

```
h = IntPtr.Zero;
```

4.4 SetDeviceParam

[函数]

int SetDeviceParam(HANDLE handle, const char *ItemValues)

[功能]

设置控制器参数，例如设备号、门磁类型、锁驱动时间、读卡间隔等。

[参数说明]

handle

[in] 由 Connect()成功返回的句柄。

ItemValues

[in] 将要设置的设备参数值，多个参数值之间可以用逗号分开，一次至多可同时设置 30 个参数（可设置的参数值属性请参见表二）。

[返回值]

返回为 0 时，表示成功；返回负数时为错误，错误代码的信息请参见附表五。

[示例]

Python:

```
items = ("DeviceID=1,Door1SensorType=2,Door1Drivertime=6,Door1Intertime=3")
```

```

p_items = create_string_buffer(items)
ret = self.commpro.SetDeviceParam(self.hcommpro, p_items)
c#:
int ret = 0;
items = ("DeviceID=1,Door1SensorType=2,Door1Drivertime=6,Door1Intertime=3")
ret = SetDeviceParam(h, items);

```

4.5 GetDeviceParam

[函数]

```
int GetDeviceParam(HANDLE handle, char *Buffer, int BufferSize, const char *Items)
```

[功能]

读取控制器参数，例如设备号、门磁类型、锁驱动时间、读卡间隔等。

[参数说明]

handle

[in] 由 Connect()成功返回的句柄。

Buffer

[in] 用于接收返回数据的缓冲区，返回的数据是文本格式的，可能是多个参数值，各参数之间用逗号分隔。

BufferSize

[in] 用于接收返回数据的缓冲区大小。

Items

[in] 将要读取的设备参数名称表，多个名称之间可以用逗号分开，一次至多可同时读取 30 个参数（可读取的参数值属性请参见表一）。

[返回值]

返回为 0 时，表示操作成功；返回负数时为失败，错误代码的信息请参见附表五。

[示例]

```

Python:
buffer = create_string_buffer(2048)
items = ("DeviceID,Door1SensorType,Door1Drivertime,Door1Intertime")
p_items = create_string_buffer(items)
ret = self.commpro.GetDeviceParam(self.hcommpro, buffer, 256, p_items)
c#:
int ret = 0;
int BUFFERSIZE = 10 * 1024 * 1024;
byte[] buffer = new byte[BUFFERSIZE];
items = ("DeviceID,Door1SensorType,Door1Drivertime,Door1Intertime");
ret = GetDeviceParam(h, ref buffer [0], BUFFERSIZE, items);

```

4.6 ControlDevice

[函数]

```
int ControlDevice(HANDLE handle, LONG OperationID, LONG Param1, LONG Param2, LONG Param3, LONG Param4, const char *Options)
```

[功能]

控制控制器动作。

[参数说明]

handle

[in] 由 Connect()成功返回的句柄；

OperationID

[in] 操作内容：1 表示锁输出或者辅助输出，2 表示取消报警，3 重启设备,4 启用和禁用常开（这里的禁用常开包括首开常开、连续五次卡启用的常开、远程启用的常开）。

Param1

[in] 当 OperationID 为输出操作时，若 Param2 为门输出，此参数表示设备中门的编号，若 Param2 为辅助输出，此参数表示设备中辅助输出口的编号，详细请参见附表三；当 OperationID 为取消报警时，默认值为 0；

Param2

[in] 当 OperationID 为输出操作时，此参数表示设备输出点地址类型（1：锁输出，2：辅助输出），详细请参见附表三；当 OperationID 为取消报警时，默认值为 0；当 OperationID 为 4，即常开时，该参数表示是常开启用还是禁用常开（0：禁用；1：启用）；

Param3

[in] 当 OperationID 为输出操作时，此参数表示开门时间（0 表示关，255 表示常开，取值范围为 1~60（秒）），默认值为 0；

Param4

[in] 预留之用，默认为 0；

Option

[in] 默认为空，扩展之用；

[返回值]

返回为 0 或者正数时，表示成功；返回负数时表示失败，错误代码的信息请参见附表五。

[示例]

Python: #开编号为1 的门6 秒

```
operation_id = 1
door_id = 1
address_type = 2
door_action = 6
ret = self.commpro.ControlDevice(self.hcommpro, operation_id, door_id, address_type,
door_action, 0, '')
```

c#: 使编号为 2 的门常开

```
int ret = 0;
int operid = 1;
int doorid = 2;
int addresstype = 0;
int dooraction = 255;
ret = ControlDevice(h, operid, doorid, addresstype, dooraction, 0, "");
```

4.7 SetDeviceData

[函数]

```
int SetDeviceData(HANDLE handle,const char *TableName,const char *Data, const char *Options)
```

[功能]

设置数据到设备,用于设置时间段、用户信息、假日设置等数据，数据可以是一条记录，也可以是多条记录，如果插入的记录的主键已在设备中则覆盖原记录。

[参数说明]

handle

[in] 由 Connect()成功返回的句柄；

TableName

[in] 数据表名，目前可使用表请参见附表四

Data

[in] 数据记录表示，数据是文本格式的，多条记录之间用\r\n 分隔，各个“字段=值”对之间用\t 分隔；

Options

[in] 默认为空，扩展之用；

[返回值]

返回为 0 时，表示操作成功；返回负数时表示失败，错误代码的信息请参见附表五。

[示例]

Python:

```
table = "user"          # 用户信息表
data = "Pin=19999\tCardNo=13375401\tPassword=1\r\nPin=2\tCardNo=14128058\tPassword=1"
p_table = create_string_buffer(table)
str_buf = create_string_buffer(data)
ret = self.commpro.SetDeviceData(self.hcommpro, p_table, str_buf, '') # 向用户信息表上传 str_buff 数据
```

c#:

```
int ret = 0;
string devtablename = "user";
string data = "Pin=19999\tCardNo=13375401\tPassword=1\r\nPin=2\tCardNo=14128058\tPassword=1";
string options = "";
ret = SetDeviceData(h, devtablename, data, options);
```

4.8 GetDeviceData

[函数]

```
int GetDeviceData(HANDLE handle, char *Buffer, int BufferSize, const char *TableName, const char *FieldNames, const char *Filter, const char *Options)
```

[功能]

从设备读取数据,用于读取刷卡记录、时间段、用户信息、假日设置等数据，数据可以是一条记录，也可以是多条记录。

[参数说明]

handle

[in] 由 Connect()成功返回的句柄；

Buffer

[in] 用于接收返回数据的缓冲区，返回的数据是文本格式的，可能是多条记录，各条记录之间用\r\n 分隔。

BufferSize

[in] 用于接收返回数据的缓冲区大小。

TableName

[in] 数据表名，目前可用的表名参见附表四；

FieldNames

[in] 字段名列表，多个字段之间用\t 分开，“*”表示全部字段，此时返回数据字段的第一行是字段名。

Filter

[in] 读取数据的条件，当单独一个“字段名 操作符 值”构成的字符串时，可以支持多个条件，使用逗号分隔，按如下方式：

<字段名>=<值> (“=”符号两边不可以有空格)

Options

[in] 当前仅在下载门禁事件记录表的数据时有效，值为“New Record”时下载新记录，当为空时下载全部记录。下载其他表数据时，该字段置为空字符串即可。

[返回值]

返回为 0 或者正数时，表示操作成功，其值为记录条数；返回负数时为失败，错误代码的信息请参见附表五。

[示例]

Python:

```
table = "user"          # 从 user 表下载用户数据
fieldname = "*"        # 下载表中的所用字段信息
```

```

pfilter = ""          # 没有过滤条件, 全部下载
options = ""
query_buf = create_string_buffer(4*1024*1024)
query_table = create_string_buffer(table)
query_fieldname = create_string_buffer(fieldname)
query_filter = create_string_buffer(filter)
query_options = create_string_buffer(options)
ret = self.commpo.GetDeviceData(self.hcommpo, query_buf, 4*1024*1024, query_table,
query_fieldname, query_filter, query_options)
c#:
int ret = 0;
int BUFFERSIZE = 10 * 1024 * 1024;
byte[] buffer = new byte[BUFFERSIZE];
string devtablename = "user";
string str = "*";
string devdatfilter = "";
string options = "";
ret = GetDeviceData(h, ref buffer[0], BUFFERSIZE, devtablename, str, devdatfilter, options);

```

4.9 GetDeviceDataCount

[函数]

```

int GetDeviceDataCount(void *Handle, const char *TableName, const char
*Filter, const char *Options)

```

[功能]

读取设备中的记录总数信息, 返回指定数据的记录条数。

[参数说明]

Handle

[in] 由 Connect()成功返回的句柄;

TableName

[in] 数据表名, 目前可用的表名参见附表四

Filter

[in] 默认为空, 扩展之用;

Options

[in] 默认为空, 扩展之用;

[返回值]

返回为 0 或者正数时, 表示操作成功, 其值为记录条数; 返回负数时为失败, 错误代码的信息请参见附表五。

[示例]

Python:

```

Table = 'user'
Filter = ""
p_table = create_string_buffer(table)
p_filter = create_string_buffer(filter)
ret = self.commpo.GetDeviceDataCount(self.hcommpo, p_table, p_filter, '')

```

c#:

```

int ret = 0;
string devtablename = "user";
string devdatfilter = "";
string options = "";
ret = GetDeviceDataCount(h, devtablename, devdatfilter, options);

```

4.10 DeleteDeviceData

[函数]

```
int DeleteDeviceData(HANDLE handle, const char *TableName, const char *Data, const char *Options)
```

[功能]

删除设备中的数据，例如用户信息、时间段等数据。

[参数说明]

handle

[in] 由 Connect()成功返回的句柄；

TableName

[in] 数据表名，目前可用的表名参见附表四。

Data

[in] 删除的条件，数据记录表示，数据是文本格式的，各个“条件字段=值”对之间用\t分隔；

Options

[in] 默认为空，扩展之用；

[返回值]

返回为 0 或者正数时，表示成功；返回负数时为失败，错误代码的信息请参见附表五。

[示例]

Python:

```
table = "user"
data = "Pin=2"          # 删除数据的条件
p_table = create_string_buffer(table)
p_data = create_string_buffer(data)
ret = self.commpo.DeleteDeviceData(self.hcommpo, p_table, p_data, "")
```

c#:

```
int ret = 0;
string devtablename = "user";
string data = "Pin=2";
string options = "";
ret = DeleteDeviceData(h, devtablename, data, options);
```

4.11 GetRTLog

[函数]

int GetRTLog(HANDLE handle, char *Buffer, int BufferSize)

[功能]

获取设备产生的实时事件记录以及设备的门状态、报警状态等。

[参数说明]

handle

[in] 由 Connect()成功返回的句柄。

Buffer

[in] 用于接收返回数据的缓冲区，返回的数据是文本格式的。

该缓冲区中保存的数据有两类，一类是实时事件记录，另一类是门状态/报警状态。使用该函数返回的数据，每次只能是其中的一类。如果为实时事件记录时，可同时返回多条事件记录（取决于当前时刻设备中实时监控缓冲区中的事件记录条数）。

缓冲区中的详细数据格式请参见附表七。

The buffer saved data has two types, one is the real-time event record, another kind is the door state/alarm condition. Use this function returns data, every time can only is one of the category. If for real-time event record, but also to return to more event record

BufferSize

[in] 用于接收返回数据的缓冲区大小。

[返回值]

返回为 0 或者正数时，为接收数据的记录数；返回负数时为失败，错误代表的信息请参见附表五。

[示例]

Python:

```
rt_log = create_string_buffer(256)
```

```
ret = self.commpo.GetRTLog(self.hcommpo, rt_log, 256)
```

c#:

```
int ret = 0;
```

```
int buffersize = 256;
```

```
byte[] buffer = new byte[256];
```

```
ret = GetRTLog(h, ref buffer[0], buffersize);
```

4.12 GetRTLogExt

[函数]

int GetRTLogExt(HANDLE handle, char *Buffer, int BufferSize)

[功能]

获取设备产生的实时事件记录以及设备的门状态、报警状态等。不同于 GetRTLog，该函数获取 PUSH 格式的记录。

[参数说明]

handle

[in] 由 Connect()成功返回的句柄。

Buffer

[in] 用于接收返回数据的缓冲区，返回的数据是文本格式的。

该缓冲区中保存的数据有两类，一类是实时事件记录，另一类是门状态/报警状态。使用该函数返回的数据，每次只能是其中的一类。如果为实时事件记录时，可同时返回多条事件记录（取决于当前时刻设备中实时监控缓冲区中的事件记录条数）。

缓冲区中的详细数据格式请参见附表八。

The buffer saved data has two types, one is the real-time event record, another kind is the door state/alarm condition. Use this function returns data, every time can only is one of the category. If for real-time event record, but also to return to more event record

BufferSize

[in] 用于接收返回数据的缓冲区大小。

[返回值]

返回为 0 或者正数时, 为接收数据的记录数; 返回负数时为失败, 错误代表的信息请参见附表五。

[示例]

Python :

```
rt_log = create_string_buffer(256)
ret = self.commpro.GetRTLogExt(self.hcommpro, rt_log, 256)
c#:
int ret = 0;
int buffersize = 256;
byte[] buffer = new byte[256];
ret = GetRTLogExt(h, ref buffer[0], buffersize);
```

4.13 SearchDevice

[函数]

```
int SearchDevice(char *CommType,char *Address, char *Buffer)
```

[功能]

搜索局域网内的门禁控制器。

[参数说明]

CommType

[in] 通讯类型为 “UDP” (或者为 Ethernet), 将会搜索指定通讯类型的设备;

Address

[in] 广播地址, 将会搜索指定 IP 地址范围内局域网的设备, 默认为 255.255.255.255, 即全网广播;

Buffer

[in] 用于存储搜索到的设备的缓冲区, 用户应根据网内设备数量来确定申请内存值。例如 50 台设备以内建议申请 32K 的内存, 100 台设备以内建议申请 64K 的内存大小。

[返回值]

返回为 0 或者正数时, 为搜索到的门禁控制器数量; 返回负数时为失败, 错误代码的信息请参见附表五。

[说明]

由于该方法为 UDP 广播方式搜索局域网内的门禁控制器, UDP 广播包不能穿过路由器, 所以不能将控制器与服务器用路由器隔开。

另外, 对于设备和服务器不在同一个网段的情况, 如果出现能通过该方法搜索到, 但 Ping 不通控制器 IP 地址的情况, 请尝试调整控制器 IP 地址和服务器地址到同一个子网中 (未必是同一个网段), 具体网络中的网络设置, 可咨询相应的网管, 以获取正确的 IP 地址, 子网掩码以及网关。

[示例]

Python :

```
dev_buf = create_string_buffer("", 64*1024)
ret=self.commpro.SearchDevice("UDP", "255.255.255.255", dev_buf)
c#:
int ret = 0;
string udp = "UDP";
string address = "255.255.255.255";
byte[] buffer = new byte[64 * 1024];
ret = SearchDevice(udp, address, ref buffer[0]);
```

4.14 ModifyIPAddress

[函数]

```
int ModifyIPAddress(char *CommType, char *Address, char *Buffer)
```

[功能]

UDP 广播方式修改控制器 IP 地址（考虑到设备的安全性，只能修改没有设密码的控制器的 IP 地址，子网掩码和网关）。

[参数说明]

CommType

[in] 搜索门禁控制器时采用的通讯方式。此处为“UDP”（或者为 Ethernet）；

Address

[in] 广播地址，默认为 255.255.255.255；

Buffer

[in] 用于存储目标设备的 MAC 地址、新 IP 地址；

除 IP 地址外，子网掩码和网关请根据当前网络进行设置。

[返回值]

返回为 0 或者正数时，为接收数据的记录数；返回负数时为失败，错误代码的信息请参见附表五。

[说明]

如果当前设备已经设置了设备通讯密码，为保证设备安全，将无法通过该方法来修改设备的 IP 地址。此时如需修改 IP 地址，请使用 SetDeviceParam 来设置 IP 地址、网关、子网掩码。如果忘记通讯密码，可使用拨码开关来进行通讯参数的初始化，通讯密码也将被取消。拨码开关的第 7 位，默认为 OFF；当在 10 秒内将其往上再往下拨动重复 3 次，最后拨回 OFF 位，重启设备就可恢复 IP 地址、网关、子网掩码、通讯密码。

[示例]

Python :

```
mac = '00:17:61:01:88:27'      # 目标设备的 MAC 地址
new_ip = '192.168.12.156'     # 设备新的 IP 地址
comm_pwd = ""
str = "MAC=%s,IPAddress=%s " % (mac,new_ip)
p_buf = create_string_buffer(str)
modify_ip = self.commpro.ModifyIPAddress("UDP", "255.255.255.255", p_buf)
c#:
int ret = 0;
string udp = "UDP";
string address = "255.255.255.255";
string buffer = "MAC=00:17:61:01:88:27, IPAddress=192.168.12.156";
ret = ModifyIPAddress(udp,address,buffer);
```

4.15 PullLastError

[函数]

```
int PullLastError()
```

[功能]

获取返回值错误码，用其他返回返回失败时，可通过该函数获取错误码。比如在使用函数 Connect()连接设备失败返回 0 时，可以使用该函数获取当前的错误码。

[参数说明]

无

[返回值]

返回错误 ID 号。

[示例]

Python :

见下面的新增部分

删除上面一行，添加下面的代码：

```
params= u"protocol=TCP,ipaddress=192.168.1.201,port=4370,timeout=3000,passwd=123abc"
constr = create_string_buffer(params)
self.hcommpro = self.commpro.Connect(constr)
if self.hcommpro > 0:
    self.connected = True
else:
    error = self.commpro.PullLastError()
c#:
int ret = 0;          // Error code
string str = " protocol=TCP,ipaddress=192.168.1.201,port=4370,timeout=3000,passwd=123abc ";
h = Connect(str);
if (h != IntPtr.Zero)
{
    MessageBox.Show("Connect device succeed!");
}
else
{
    ret = PullLastError();
    MessageBox.Show("Connect device Failed! The error code is: " + ret);
}
```

4.16 SetDeviceFileData

[函数]

```
int SetDeviceFileData(void *Handle, const char *FileName, char *Buffer,int BufferSize,const
char *Options)
```

[功能]

将文件从 PC 传送到设备。

[参数说明]

Handle

[in] 由 Connect()成功返回的句柄；

FileName

[in] 传送到设备的文件名,例如 emfw.cfg 文件；

Buffer

[in] 待传送文件的数据缓冲区；

BufferSize

[in] 传送数据的长度；

Options

[in] 默认为空，扩展之用

[返回值]

返回为 0 或者正数时，表示成功；返回负数时为失败，错误代码的信息请参见附表五。

[示例]

Python:

```
file_name = "emfw.cfg"
pfile_name = create_string_buffer(file_name)
buff_len = len(pfile_name)
pbuffer = create_string_buffer(buff_len)
ret = self.commpro.SetDeviceFileData(self.hcommpro, pfile_name, pbuffer, buff_len, "")
c#: (升级固件时，就是用该接口把 emfw.cfg 传到固件中)
int ret = 0;
string filename = "emfw.cfg";
FileStream fsFile = File.OpenRead(this.openFileDialog1.FileName);
string buffersize = (int)fsFile.Length;
byte[] buffer = new byte[buffersize];
```

```
string options = "";
ret = SetDeviceFileData(h, filename, ref buffer[0], buffersize, options);
```

4.17 GetDeviceFileData

[函数]

```
int GetDeviceFileData(void *Handle,char *Buffer,int *BufferSize,const char *FileName,const char *Options)
```

[功能]

从设备获取文件到 PC。

[参数说明]

Handle

[in] 由 Connect()成功返回的句柄;

FileName

[in] 从设备获取的文件名,例如 main 文件;

Buffer

[in] 接收数据的缓冲区;

BufferSize

[in] 接收数据的长度;

Options

[in] 默认为空, 扩展之用

[返回值]

返回为 0 或者正数时, 表示成功; 返回负数时为失败, 错误代码的信息请参见附表五。

[示例]

Python:

```
file_name = "main"
```

```
pfile_name = create_string_buffer(file_name)
```

```
pbuffer = create_string_buffer(4*1024*1024)
```

```
ret = self.commpo.GetDeviceFileData(self.hcommpo, pbuffer, buff_len, pfile_name, "")
```

c#:

```
int ret = 0;
```

```
int buffersize = 4 * 1024 * 1024;
```

```
byte[] buffer = new byte[buffersize];
```

```
string filename = "user.dat";
```

```
string options = "";
```

```
ret = GetDeviceFileData(h, ref buffer[0], ref buffersize, filename, options);
```

4.18 ProcessBackupData

[函数]

```
int ProcessBackupData(const unsigned char *revBuf, int fileLen, char *outBuf, int outSize)
```

[功能]

用来处理设备备份的文件, 如 SD 卡中的备份文件等。

[参数说明]

revBuf

[in] 是传进来的文件内容;

fileLen

[in] 是文件长度;

outBuf

[in] 是接收返回的数据;

outsize

[in] 是接受数据的最大长度。

[返回值]

返回为 0 或者正数时，表示成功；返回负数时为失败，错误代码的信息请参见附表五。

[示例]

Python :

```
filename = "sddata.dat"
buff_len = len(filename)
buf = create_string_buffer(filename)
buffer = create_string_buffer(16*1024*1024)
ret = self.commpro. ProcessBackupData(buf, buff_len, ref buffer[0], 16 * 1024 * 1024)
```

c#:

```
byte[] buffer = new byte[16 * 1024 * 1024];
byte[] buf = new byte[16 * 1024 * 1024];
int BufferSize = 0;
int ret = -1;
string filename = "user.dat";
StreamReader proFile = new StreamReader(filename);
BufferSize = proFile.BaseStream.Read(buf, 0, 16 * 1024 * 1024);
ret = ProcessBackupData(buf, BufferSize, ref buffer[0], 16 * 1024 * 1024);
```

4.19 BufferToProtocolData

[函数]

```
int BufferToProtocolData(char *ProStruct, char *Buffer, int Datalen);
```

[功能]

将字节流转换为格式化字符串。

[参数说明]

ProStruct

[in] 表结构信息；

Datalen

[in] 字节流的数据长度；

Buffer

[in,out] 调用函数时，需要转换的字节流的 **buffer**，函数调用结束以后的格式化字符串。；

[返回值]

返回为 0 或者正数时，表示成功，返回的是格式化字符串的记录数；返回负数时为失败，错误代码的信息请参见附表五。

[示例]

4.20 ProtocolDataToBuffer

[函数]

```
int ProtocolDataToBuffer(char *Buffer, char *Datas, char *ProStruct, char *TableName);
```

[功能]

将格式化字符串转换为字节流。

[参数说明]

ProStruct

[in] 表结构信息；

TableName

[in] 表名；

Buffer

[out]转换后的字节流的 **buffer**；

Datas

[in]需要转换的格式化字符串的内容

[返回值]

返回为 0 或者正数时，表示成功，返回的字节流的长度；返回负数时为失败，错误代码的信息请参见附表五。

[示例]

5. 附录

5.1 附表一：接口文件的详细说明

文件名	说明
plcommpro.dll	PullSDK 函数的动态连接库接口
plcomms.dll	PullSDK 接口的依赖库
plrscomm.dll	PullSDK 接口的依赖库
pltcpcomm.dll	PullSDK 接口的依赖库
plrscagent.dll	PullSDK 接口的依赖库

5.2 附表二：控制器参数的说明

属性名称	参数	读写类型	备注
序列号	~SerialNumber	只读	
Mac 地址	MAC	只读	
门数量	LockCount	只读	即控制板上 Lock 的数量
读头数量	ReaderCount	只读	仅指韦根读头的数量
辅助输入数量	AuxInCount	只读	
辅助输出数量	AuxOutCount	只读	
通信密码	ComPwd	读写	默认为空字符串。最大 15 位字符（含数字和字母）。
IP 地址	IPAddress	读写	默认 192.168.1.201
网关	GATEIPAddress	读写	没有设置时为 IP 地址的值
波特率	RS232BaudRate	读写	默认为 38400
子网掩码	NetMask	读写	默认 255.255.255.0
反潜回规则	AntiPassback	读写	单门双向控制器 1 代表'1 号门读头间反潜' 双门单向控制器 1 代表'1,2 号门间反潜' 双门双向控制器 1 代表'1 号门读头间反潜' 2 代表'2 号门读头间反潜' 3 代表'1,2 号门各自读头间反潜' 4 代表'1,2 号门间反潜' 四门单向控制器（此处读头间反潜仅指 inBIO 读头间的反潜） 1 代表'1-2 门反潜' 2 代表'3-4 门反潜' 3 代表'1-2 门反潜'和'3-4 门反潜' 4 代表'1/2-3/4 门反潜'

			<p>5 代表'1-2/3 门反潜' 6 代表'1-2/3/4 门反潜' 16 代表仅 1 号门读头间反潜 32 代表仅 2 号门读头间反潜 64 代表仅 3 号门读头间反潜 128 代表仅 4 号门读头间反潜 其他备选项: 48 代表 1,2 号门的各自读头间同时反潜 80 代表 1,3 号门的各自读头间同时反潜 144 代表 1,4 号门的各自读头间同时反潜 96 代表 2,3 号门的各自读头间同时反潜 160 代表 2,4 号门的各自读头间同时反潜 196 代表 3,4 号门的各自读头间同时反潜 112 代表 1,2,3 号门的各自读头间同时反潜 176 代表 1,2,4 号门的各自读头间同时反潜 208 代表 1,3,4 号门的各自读头间同时反潜 224 代表 2,3,4 号门的各自读头间同时反潜 240 代表 1,2,3,4 号门的各自读头间同时反潜 (以上请根据实际需要进行选择和配置)</p>
互锁	InterLock	读写	<p>两门控制器 1 代表'1-2 两门互锁' 四门控制器 1 代表'1-2 两门互锁' 2 代表'3-4 两门互锁' 3 代表'1-2-3 三门互锁' 4 代表'1-2 两门互锁'和'3-4 两门互锁' 5 代表'1-2-3-4 四门互锁'</p>
胁迫密码	Door1ForcePass Word Door2ForcePass Word Door3ForcePass Word Door4ForcePass Word	读写	最大 8 位
紧急密码	Door1SupperPass Word Door2SupperPass Word Door3SupperPass Word Door4SupperPass Word	读写	最大 8 位
闭门回锁	Door1CloseAndLock Door2CloseAndLock Door3CloseAndLock Door4CloseAndLock	读写	<p>1 启用 0 不启用</p>
门磁类型	Door1SensorType Door2SensorType Door3SensorType Door4SensorType	读写	<p>0 代表无 1 代表常开 2 代表常闭</p>
锁驱动时长	Door1Drivertime Door2Drivertime Door3Drivertime Door4Drivertime	读写	<p>设置范围 (0~255) 0 代表常闭 255 代表常开 1~254 代表开门时长</p>
门磁超时报警时长	Door1Detectortime Door2Detectortime Door3Detectortime Door4Detectortime	读写	<p>设置范围 (0~255) 单位为秒 (s)</p>
开门方式	Door1VerifyType Door2VerifyType Door3VerifyType Door4VerifyType	读写	<p>1 为“仅指纹” 4 为“仅卡” 6 为“卡或指纹” 10 为“卡加指纹” 11 为“卡+密码”</p>
多卡开门启用	Door1MultiCardOpenDoor Door2MultiCardOpenDoor	读写	<p>0 不启用 1 启用</p>

4. PullSDK 接口函数详述

	Door3MultiCardOpenDoor Door4MultiCardOpenDoor		
首卡开门启用	Door1FirstCardOpenDoor Door2FirstCardOpenDoor Door3FirstCardOpenDoor Door4FirstCardOpenDoor	读写	0 不启用 1 首卡常开
门有效时间段(接收有效刷卡时间段)	Door1ValidTZ Door2ValidTZ Door3ValidTZ Door4ValidTZ	读写	默认 0 表示锁未激活
门常开时间段	Door1KeepOpenTimeZone Door2KeepOpenTimeZone Door3KeepOpenTimeZone Door4KeepOpenTimeZone	读写	默认 0, 没有设置
刷卡间隔	Door1Intertime Door2Intertime Door3Intertime Door4Intertime	读写	0 为无间隔 以 S (秒) 为单位
MCU 看门狗	WatchDog	读写	0 不启用 1 启用
同步控制器的时间	DateTime	只写	DateTime= ((Year-2000)*12*31 + (Month -1)*31 + (Day-1))*(24*60*60) + Hour* 60 *60 + Minute*60 + Second; 例如, 要设置的时间 2010-10-26 20:54:55,转换后 DateTime= 347748895; 解析方法: 如果获取到 “DateTime = 347748895”; 那么: Second = DateTime % 60; Minute = (DateTime / 60) % 60; Hour = (DateTime / 3600) % 24; Day = (DateTime / 86400) % 31 + 1; Month= (DateTime / 2678400) % 12 + 1; Year = (DateTime / 32140800) + 2000;
四门转两门	Door4ToDoor2	读写	0 不启用 1 启用
取消门常开日期	Door1CancelKeepOpenDay Door2CancelKeepOpenDay Door3CancelKeepOpenDay Door4CancelKeepOpenDay	只读	取消常开时保存的日期
SD 卡备份时间	BackupTime	读写	1~24, 设置为整点
夏令时显示参数	~DSTF	读写	为 0 不显示, 为 1 时显示, 默认为 0
夏令时启动参数	DaylightSavingTimeOn	读写	为 0 不启动, 为 1 时启动, 默认为 0
夏令时模式	DLSTMode	读写	0 为模式一, 1 为模式二
夏令时标记	CurTimeMode	读写	1 当前为夏令时, 2 当前不是夏令时, 固件内部使用
夏令时模式一开始时间	DaylightSavingTime	读写	值为 4 个字节分别表示 “月-日-时-分”
夏令时模式一结束时间	StandardTime	读写	值为 4 个字节分别表示 “月-日-时-分”
夏令时模式二开始时间: 月	WeekOfMonth1	读写	值为 1~12
夏令时模式二开	WeekOfMonth2	读写	值为 1~6

始：第几个周			
夏令时模式二开始：周几	WeekOfMonth3	读写	值为 1~7
夏令时模式二开始：时	WeekOfMonth4	读写	值为 0~23
夏令时模式二开始：分	WeekOfMonth5	读写	值为 0~59
夏令时模式二结束时间：月	WeekOfMonth6	读写	值为 1~12
夏令时模式二结束：第几周	WeekOfMonth7	读写	值为 1~6
夏令时模式二结束：周几	WeekOfMonth8	读写	值为 1~7
夏令时模式二结束：时	WeekOfMonth9	读写	值为 0~23
夏令时模式二结束：分	WeekOfMonth10	读写	值为 0~59
指纹比对阈值	MThreshold	读写	值为 0~100

5.3 附表三：ControlDevice 参数说明

Operation ID	说明	Param1	Param2	Param3	Param4	Options
1	输出操作	门编号或者辅助输出编号	1: 锁输出 2: 辅助输出（输出操作的地址类型）	0: 关 255: 常开 1~60: 开以及开的持续时间 （当 Param2=1 时，Param3 的值才有意义；）	保留	扩展参数为空
2	取消报警	0（无意义）	0（无意义）	0（无意义）	保留	扩展参数为空
3	重启设备	0（无意义）	0（无意义）	0（无意义）	保留	扩展参数为空
4	开启/禁用常开	门编号	0: 禁用 1: 开启	0（无意义）	保留	扩展参数为空

注：当 OperationID=1 时，Param2 决定了 Param1 是表示门编号还是辅助输出编号。Param1 参数取值若为门编号，最大值是该设备门总个数，Param1 参数取值若为辅助输出编号，最大辅助输出编号是该设备辅助输出总个数。

5.4 附表四：功能函数的表结构说明

表名称	TableName	字段（黄色背景的字是主键）	备注
人员信息表	user	CardNo、Pin、Password、Group、StartTime、EndTime	CardNo 最大为 2147483647。 Pin 人员编号，为九位码，只能为数字。 Password 只能为数字，最大长度为 6 位。 StartTime 和 EndTime 格式填写规范：

4. PullSDK 接口函数详述

			YYYYMMDD, 例如: 20100823; Group 是多卡开门人员组;
人员门禁权限表	userauthorize	Pin、AuthorizeTimezoneId、 AuthorizeDoorId	AuthorizeDoorId 表明这个权限包含该设备哪些门, 取值是通过二进制编码而得, 每个门用一个二进制位表示, 如果该位为 1, 则该门属于该权限, 具体可参照如下: 1 代表 LOCK1; 2 代表 LOCK2; 3 代表 LOCK1 和 LOCK2; 4 代表 LOCK3; 5 代表 LOCK1 和 LOCK3; 6 代表 LOCK2 和 LOCK3; 7 代表 LOCK1、LOCK2 和 LOCK3; 8 代表 LOCK4; 9 代表 LOCK1 和 LOCK4; 10 代表 LOCK2 和 LOCK4; 11 代表 LOCK1、LOCK2 和 LOCK4; 12 代表 LOCK3 和 LOCK4; 13 代表 LOCK1、LOCK3 和 LOCK4; 14 代表 LOCK2、LOCK3 和 LOCK4; 15 代表 LOCK1、LOCK2、LOCK3 和 LOCK4 15 可以这样计算, 四个门的编号分别为 1, 2, 3, 4, 则: $1 \ll (1-1) + 1 \ll (2-1) + 1 \ll (3-1) + 1 \ll (4-1) = 15$ 或者 $(1111)_2 = (15)_{10}$
节假日表	holiday	Holiday、HolidayType、Loop	假日类型 " HolidayType " 的值: 1、2、3 Loop 的值: 1-按年循环; 2-不按年循环
时间段表	timezone	TimezoneId、 SunTime1、SunTime2、 SunTime3、MonTime1、 MonTime2、MonTime3、 TueTime1、TueTime2、 TueTime3、WedTime1、 WedTime2、WedTime3、 ThuTime1、ThuTime2、 ThuTime3、FriTime1、 FriTime2、FriTime3、 SatTime1、SatTime2、 SatTime3、Hol1Time1、 Hol1Time2、Hol1Time3、 Hol2Time1、Hol2Time2、 Hol2Time3、Hol3Time1、 Hol3Time2、Hol3Time3	Time 格式 (hour*100+minute) <<16+(hour*100 + minute) 例如: 设置周一时间段 1 为 8:30 ~12:30 , 那么 MonTime1=54396110 ; 8:30 → 8*100+30 → 33E (十六进制) 12:30 → 12*100+30 → 4CE (十六进制) 033E04CE → 54396110 (十进制)
门禁事件记录表	transaction	Cardno、Pin、Verified、 DoorID、EventType、 InOutState、Time_second	验证方式 " Verified ": 3 代表 "仅密码" 验证; 4 代表 "仅卡" 验证; 11 代表 "卡加密码" 验证; 200 代表 "其他" Time_second 书写规范: YYYY-MM-DD hh:mm:ss (写入后, 数据格式会转换, 如果要取出来解析, 解析公式如下: second = t % 60; t /= 60; minute = t % 60;

			<p>t /= 60; hour = t % 24; t /= 24; day = t % 31 + 1; t /= 31; month = t % 12 + 1; t /= 12; year = t + 2000;)</p> <p>事件选项 EventType 详见附表六</p>
首卡常开表	firstcard	Pin、DoorID、TimezoneID	
多卡开门组合表	multimcard	Index、DoorId、Group1、Group2、Group3、Group4、Group5	Group1~Group5 是多卡开门的组号
联动信息表	inoutfun	Index、EventType、InAddr、OutType、OutAddr、OutTime、Reserved	<p>全部事件类型 EventType 请参见门禁记录表类型； 当事件类型为 220(辅助输入点断开)和 221(辅助输入点短路)时，输入点为辅助输入； 当事件类型除了上述两种类型以外的其他类型，输入点是门： 输入点 InAddr 是门： 0 任意； 1 门 1； 2 门 2； 3 门 3； 4 门 4； 输入点 InAddr 是辅助输入： 0 任意； 1 辅助输入 1； 2 辅助输入 2； 3 辅助输入 3； 4 辅助输入 4； 输出类型 OutType 为 0 时，输出点 OutAddr 表示锁： 0 所有门锁； 1 门锁 1； 2 门锁 2； 3 门锁 3； 4 门锁 4； 当输出类型 OutType 为 1 时，输出点 OutAddr 表示辅助输出： 0 所有辅助输出； 1 辅助输出 1； 2 辅助输出 2； 3 辅助输出 3； 4 辅助输出 4； 5 辅助输出 5； 6 辅助输出 6；</p>

指纹模板表	templatev10	Size、UID、Pin、FingerID、Valid、Template、Resverd、EndTag	<p>Size: 指纹模板长度;</p> <p>UID: 固件内部使用;</p> <p>Pin: 人员 Pin;</p> <p>FingerID: 手指编号, 若指纹是普通指纹, 取值 0~9, 若为胁迫指纹, 取值 16~25 (胁迫指纹的手指编号是正常指纹的手指编号+16);</p> <p>Valid 是否是胁迫指纹: 1 表示普通指纹; 3 表示胁迫指纹;</p> <p>Template: 指纹模板;</p> <p>Resverd: 保留字段;</p> <p>EndTag: 固件内部使用;</p>
指静脉表	fvtemplate	Size、Pin2、FingerID、Duress、Fv_ID_Index	<p>Size: 指纹模板长度;</p> <p>Pin2: 人员 Pin</p> <p>FingerID: 手指编号, 若指纹是普通指纹, 取值 0~9</p> <p>Duress: 胁迫指静脉</p> <p>Fv_ID_Index: 具体个体模板编号, 从 0 开始计算。(如一个手指登记 3 枚模板)</p>

注: 请注意表格字段的大小写。

5.5 附表五: 返回值错误码说明

(1) PullSDK 及固件提供的错误码。

错误码	说明
-1	命令发送失败
-2	命令没有回应
-3	需要的缓存不足
-4	解压失败
-5	读取数据长度不对
-6	解压的长度和期望的长度不一致
-7	命令重复
-8	连接尚未授权
-9	数据错误, CRC 校验失败
-10	数据错误, PullSDK 无法解析
-11	数据参数错误
-12	命令执行错误

-13	命令错误，没有此命令
-14	通讯密码错误
-15	写文件失败
-16	读文件失败
-17	文件不存在
-18	设备空间不足
-19	校验和出错
-20	接受到的数据长度与给出的数据长度不一致
-21	设备中，没有设置平台参数
-22	固件升级，传来的固件的平台与本地的平台不一致
-23	升级的固件版本比设备中的固件版本老
-24	升级的文件标识出错
-25	固件升级，传来的文件名不对，即不是 emfw.cfg
-26	传来的指纹模板长度为 0
-27	传来的指纹 pin 号错误，找不到此用户
-28	常开时间段执行开门命令
-99	未知错误
-100	表结构不存在
-101	表结构中，条件字段不存在
-102	字段总数不一致
-103	字段排序不一致
-104	实时事件数据错误
-105	解析数据时，数据错误
-106	数据溢出，下发数据超出 4M
-107	获取表结构失败
-108	无效 OPTIONS 选项
-112	PC 传入的数据接收缓冲不足
-201	LoadLibrary 失败
-202	调用接口失败
-203	通讯初始化失败
-206	串口代理程序启动失败，原因一般是串口不存在或串口被占用
-301	获取 TCP/IP 版本失败
-302	错误版本号
-303	获取协议类型失败
-304	无效 SOCKET
-305	SOCKET 错误
-306	HOST 错误
-307	连接超时

(2) 部分常用 WinSocket 错误码

10035	<p>资源暂时不可用。</p> <p>此返回错误无法完成的立即，是例如 nonblocking 套接字操作 recv (Wsapiref_2i9e.asp)，无数据排队从套接字读取时。一个非致命的错误和操作可以稍后重试。WSAEWOULDBLOCK 通常报告为已调用的结果 nonblocking SOCK_STREAM 套接字连接 (Wsapiref_8m7m.asp)，因为一些时间必须经过建立连接。</p>
-------	--

10038	一个操作已尝试在不套接字的。套接字句柄参数引用一个有效的套接字，或者选择 (Wsapieref_lab6.asp)，一个 fd_set 的成员是无效。
10054	通过对等方重置的连接。 一个现有的连接被远程主机强制关闭。如果远程主机上的对等程序突然停止、主机重新启动或远程主机使用硬盘的关闭则，通常会发生此错误。有关详细信息，有关远程套接字上 SO_LINGER 选项，请参阅 Setsockopt (Wsapieref_94aa.asp)。如果由于的正在进行一个或多个操作时检测到错误的保持连接活动的连接已断开，也可能导致此错误。在进行的操作失败，并 WSAENETRESET。随后的操作失败，出现 WSAECONNRESET。
10060	连接超时。 由于连接的方没有不正确响应时间，一段时间后或建立的连接失败，因为响应失败的连接的主机 A 的连接尝试失败。
10061	连接被拒绝。 无连接可建立，因为目标计算机积极地拒绝。此错误通常导致尝试连接到处于非活动状态这就是指没有服务器程序运行在外部主机上的服务。
10065	无路由主机。 A 套接字操作已尝试到无法访问主机。请参阅 WSAENETUNREACH。

5.6 附表六：事件类型及代码表

代码	事件类型	说明
0	正常刷卡开门	指具有开门权限的人员刷卡且验证通过后所触发的正常事件
1	常开时间段内刷卡	指具有开门权限的人员在常开时间段内(含对单个门设置的门常开时段或者首卡常开设置的开门时段)，门已处于打开状态时刷有效卡所触发的正常事件
2	首卡开门(刷卡)	指具有首卡开门权限的人员在设定的首卡常开时间段内，门尚未打开时刷卡且验证通过后所触发的正常事件
3	多卡开门(刷卡)	指多卡组合开门时，最后一张卡验证通过时所触发的正常事件
4	紧急状态密码开门	指使用在当前门上设置的紧急状态密码(也称为超级密码)开门且验证通过后所触发的正常事件
5	常开时间段开门	指对当前门设定了门常开时间段后，到达设定的开始时间时，门自动打开所触发的事件
6	触发联动事件	当设定的联动生效时，触发该正常事件
7	取消报警	当用户对存在报警的门进行了取消报警操作，且取消报警成功时，触发该正常事件

8	远程开门	当用户对某个门进行远程开门操作且开门成功时，触发该正常事件
9	远程关门	当用户对某个门进行远程关门操作且关门成功时，触发该正常事件
10	禁用当天常开时间段	当门处于常开状态时，在读头上刷五下有效卡（必须为同一个用户）或者使用 ControlDevice 来禁用当天常开时间段时，触发该正常事件。
11	启用当天常开时间段	当门的当天常开时间段已被禁用时，在读头上刷五下有效卡（必须为同一个用户）或者使用 ControlDevice 来启用该门当天常开时间段时，触发该正常事件。
12	开启辅助输出	用户在联动设置中设定的联动动作，如果输出点地址选择了辅助输出点，动作类型选择了打开，则当该联动设置生效时，将会触发该正常事件。
13	关闭辅助输出	用户在通过联动关闭了辅助输出或者直接调用 ControlDevice 关闭辅助输出时触发的事件。
14	正常按指纹开门	指在“仅指纹”或者“卡或指纹”的验证方式下，具有开门权限的人员按指纹且验证通过后所触发的正常事件。
15	多卡开门(按指纹)	多卡开门（按指纹）：指多卡组合开门时，在“仅指纹”的验证方式下，最后一个指纹验证通过时所触发的正常事件。
16	常开时间段内按指纹	指具有开门权限的人员在常开时间段内（含对单个门设置的门常开时段或者首卡开门（常开）设置的开门时段），以及通过远程开门常开的操作，门已处于打开状态时按有效指纹所触发的正常事件。
17	卡加指纹开门	指在“卡加指纹”的验证方式下，具有开门权限的人员刷卡并按指纹且验证通过后所触发的正常事件。
18	首卡开门(按指纹)	指在仅指纹或者卡或指纹的验证方式下，具有首卡开门权限的人员在设定的首卡常开时间段内，门尚未打开时按指纹且验证通过后所触发的正常事件。
19	首卡开门(卡加指纹)	指在卡加指纹的验证方式下，具有首卡开门权限的人员在设定的首卡常开时间段内，门尚未打开时刷卡并按指纹且验证通过后所触发的正常事件
20	刷卡间隔太短	当两次刷卡之间的间隔小于该门设定的刷卡间隔时间时，触发该异常事件
21	门非有效时间段(刷卡)	指具有当前门开门权限的卡，在设定的门有效时间段之外的时间刷卡所触发的异常事件
22	非法时间段	指具有当前门开门权限的卡，在门禁权限设定的时间段之外的时间刷卡所触发的异常事件
23	非法访问	已注册的卡，尚未在当前门设置门禁权限时刷卡所触发的异常事件
24	反潜	当系统内设定的反潜规则生效时，触发该异常事件

4. PullSDK 接口函数详述

25	互锁	当系统内设定的互锁规则生效时，触发该异常事件
26	多卡验证(刷卡)	指多卡组合开门时，最后一张卡之前的刷卡验证（无论通过与否），触发该事件
27	卡未注册	指当前卡号未注册时，所触发的事件
28	门开超时	门被打开后超过延时时间后检测门磁，如果门没有关上，触发该事件
29	卡已过有效期	当已设置门禁有效时间的人员，在结束门禁日期后刷卡无法验证通过，触发该事件
30	密码错误	使用卡加密码验证方式或者胁迫密码、紧急密码开门时，密码验证错误触发该事件
31	按指纹间隔太短	当两次按指纹之间的间隔小于该门设定的刷卡间隔时间时，触发该异常事件
32	多卡验证(按指纹)	指在仅指纹或者卡或指纹验证方式下，多卡组合开门时，最后一次验证之前的指纹验证（无论通过与否），触发该事件
33	指纹已过有效期	当已设置门禁有效时间的人员，在结束门禁日期后按指纹无法验证通过，触发该事件。
34	指纹未注册	指当前指纹未在系统内登记或者虽已登记但未同步到设备中，所触发的事件。
35	门非有效时间段(按指纹)	指具有当前门开门权限的人员，在设定的门有效时间段之外的时间按指纹所触发的异常事件
36	门非有效时间段(按出门按钮)	在设定的门有效时间段外按出门按钮，无法开门时产生此事件
37	常开时间段无法关门	指当前门正处在常开状态，无法通过 ControlDevice 关门时所触发的异常事件
38	卡已挂失	当卡号被挂失处理时，刷对应的卡触发的事件
39	黑名单	当用户号被划为黑名单时，该用户进行任何比对都会产生此事件。
40	多指纹验证失败	组合验证时，多用户进行指纹比对，其中有一用户指纹验证失败时产生该事件
41	验证方式错误	当用户所使用的验证方式与设定不一致时产生该事件
42	韦根格式错误	所刷的卡的位数与配置不一致时，刷卡触发事件
43	后台验证	
44	后台验证失败	
45	后台验证超时	

46	后台验证事件	
47	发送命令失败	梯控中定义，用于主控发命令给分控时通讯失败提示的事件
48	多卡开门失败	组合验证失败时提示的事件
100	防拆报警	机器被拆除时上传的事件
101	胁迫密码开门	指使用在当前门上设置的胁迫密码开门且验证通过后所触发的报警事件
102	门被意外打开	指在正常事件（如门禁权限的人员刷卡开门、密码开门、门常开时段开门、远程开门、联动设置的开门）以外的情况下，使得门磁检测到门被打开的情况，均为门被意外打开
103	胁迫指纹开门	指人员使用已经登记的胁迫指纹开门时所触发的报警事
200	门已打开	当门磁检测到门已被正常打开时，触发该正常事件
201	门已关闭	当门磁检测到门已被正常关闭时，触发该正常事件
202	出门按钮开门	当使用出门按钮开门时，触发该正常事件
203	多卡开门(卡加指纹)	指多卡组合开门时，在“卡加指纹”的验证方式下，最后一次卡加指纹验证通过时所触发的正常事件。
204	常开时间段结束	设置的常开时间到，门将自动关闭，门常开包含了设置门的门常开时间段以及设置首卡开门时选择的时间段
205	远程开门常开	指使用远程开门时选择常开让门处于常开状态时，触发的正常事件
206	设备启动	设备启动时触发该正常事件
207	密码开门	用户使用密码产生开门事件
208	超级用户开门	用户为超级用户比对时产生的事件
209	门被锁定	主要针对出门开关按键功能，将门配置为锁定状态时，使用出门开关按键不能开门，并触发此事件。
210	消防开启	消防功能，使用所有门全部常开
211	超级用户关门	梯控中使用事件，超级用户启动常开后，再刷卡一次即可关闭常开状态。
220	辅助输入点断开	当辅助输入点断开时，触发该正常事件
221	辅助输入点短路	当辅助输入点短路时，触发该正常事件
222	后台验证成功	
223	后台验证网络不稳开门	
224	后台重新启用反潜	
255	实际为获取门状态和报警状态， 详见附表七	详见附表七

5.7 附表七：GetRTLog 函数中参数 Buffer 返回数据格式说明

在解析 Buffer 中的数据时，如果为实时事件记录，且为多条时，首先根据多条记录间通过'\r\n'来分离成单条记录。如果获取到的数据类型为门状态和报警状态，那么已经为单条记录。单条记录的数据间通过逗号隔开，故我们使用逗号将单条记录分隔开。

在解析单条记录时，首先需要根据分隔开之后的数据的第四位的值来判断，如果该值为 255，那么该记录中仅包含了门状态和报警状态，反之，该记录代表实时监控事件记录。

这两类记录的数据结构对比如下：

	第 0 位	第 1 位	第 2 位	第 3 位	第 4 位	第 5 位	第 6 位
门状态/ 报警状态	时间	门磁状态 (0 无门磁, 1 门关, 2 门开)	报警状态 (1 报警, 2 门开超时, 3 报警+门开 超时)	暂时不使用	255	暂时不使 用	200 (表示验 证方式为 “其他”), 可不使用
实时事件 记录	时间	Pin (人员 编号)	卡号	事件点编号 (包含门编 号 (即锁编 号), 辅助输 入编号, 辅 助输出编 号)	事件类 型代码, 详见附 表六	出入状态 (0 为入, 1 为出, 2 为无)	验证方式, 同附表二 中的控制 器参数的 说明中的 开门方式

注：

- (1) 设备中的实时事件记录缓存最多可以存放 30 条事件记录，当调用 GetRTLog 函数时，如果缓存中有事件记录，将会返回当前缓存中所有的记录（最多 30 条）。如果缓存中没有事件记录，设备将会返回如上所述的门状态和报警状态事件。
- (2) 除了上述监控时门状态记录中会包含当前门的开关状态外(前提是已接门磁), 还可以通过事件“门已打开”（事件代码 200）和“门已关闭”（事件代码 201）来判断当前门的开关状态。
- (3) 当记录为“门状态/报警状态”时，每条记录中包含的门磁状态（第 1 位）实际为当前设备所有门的门状态（最多四个门），四个字节分别表示四个门的状态，从低位到高位分别代表 1 号门到 4 号门。例：如果该值为 0x01020001，则表示该设备的 1 号门为关闭，2 号门为没有

设置门磁，3号门为门打开，4号门为门关闭。包含的报警状态（及门开超时）（第2位），同样是四个字节分别表示四个门的状态，每个字节中的后面两位分别表示是否有报警或者门开超时，从低位到高位分别表示报警和门开超时。如 0x01020102 表示门1和门3门开超时，门2和门4报警。

- (4) 当记录为“实时事件”且事件类型为触发联动事件（事件类型代码：6）时，第六位保存的是触发联动的事件类型代码，而第二位则被复用为联动的ID，该联动ID有软件给设备同步联动设置时设定（通常为该联动在软件端数据库中的ID值）。
- (5) 事件点编号中的三种类型的值，可以根据门禁控制板上的丝印来对应，比如门编号为1对应控制板上的Lock1，辅助输入1对应控制板上的Aux In1，辅助输出2对应控制板上的Aux Out2（注：设备型号不同，丝印内容可能不同，但数字是对应的）。

5.7 附表八：GetRTLogExt 函数中参数 Buffer 返回数据格式说明

Buffer 中的数据为 PUSH 格式的字符串记录。如果记录为实时事件记录，则包含多条以 '\r\n' 分割的单条记录。如果获取到的数据类型为门状态和报警状态，那么所包含的已经为单条记录。

单条记录分为实时事件记录与门状态报警状态的两种字符串格式，每种格式分为以 '\t' 分割的多个字段。单条记录的两种字符串格式如下：

	字符串格式
门状态/ 报警状态	<pre>type=rtstate\ttime=%s\tsensor=%02X\trelay=%02X\talarm=%02X%02X%02X%02X %\r\n</pre> <p>说明：</p> <pre>type=rtstate\ttime=xx\tsensor=AA\trelay=CC\talarm=DDEEFFGGHHIIJJKK\r\n</pre> <p>time 表示当前时间，格式为%04d-%02d-%02d %02d:%02d:%02d；</p> <p>sensor 表示门磁状态，每个门占用两个二进制位，AA 表示表示 1-4 门，1 门占用第一个字节的第 1、2 位，依此类推，0b00 表示当前门磁类型被设为无门磁，0b01 表示当前门是关着的(有门磁)，0b10 门是开着的(无门磁)；</p> <p>relay 表示继电器状态，每个门占用一个二进制位，0b0 表示继电器吸合，0b1 表示继电器断开，第一位为 1 门继电器状态，依此类推；（目前都是 0）</p> <p>alarm 表示报警状态，每个门占用一个字节，最多可表达 8 种报警，DD 为 1 门报警状态，依此类推，目前报警定义如下：</p> <p>第一位： 意外开门事件</p> <p>第二位： 防拆报警</p> <p>第三位： 胁迫密码报警</p>

	<p>第四位： 胁迫指纹报警</p> <p>第五位： 门磁超时报警</p> <p>第六位-第八位： 保留</p>
实时事件记录	<pre>type=rtlog\ttime=%s\tpin=%u\tcardno=%u\teventaddr=%d\tevent=%d\tinoutstatus=%d\tverifytype=%d\r\n</pre> <p>说明：</p> <p>time 表示当前时间，格式为%04d-%02d-%02d %02d:%02d:%02d;</p> <p>pin 表示人员编号;</p> <p>cardno 表示卡号;</p> <p>eventaddr 表示事件点编号，包含门编号（即锁编号），辅助输入编号，辅助输出编号;</p> <p>event 事件类型代码，详见附表六;</p> <p>inoutstatus 表示出入状态（0 为入，1 为出，2 为无）;</p> <p>verifytype 表示验证方式，同附表二中的控制器参数的说明中的开门方式。</p>

注：

- (1) 设备中的实时事件记录缓存最多可以存放 30 条事件记录，当调用 GetRTLogExt 函数时，如果缓存中有事件记录，将会返回当前缓存中所有的记录（最多 30 条）。如果缓存中没有事件记录，设备将会返回如上所述的门状态和报警状态事件。
- (2) 除了上述监控时门状态记录中会包含当前门的开关状态外(前提是已接门磁),还可以通过事件“门已打开”（事件代码 200）和“门已关闭”（事件代码 201）来判断当前门的开关状态。
- (3) 当记录为“实时事件”且事件类型为触发联动事件（事件类型代码：6）时，第六位保存的是触发联动的事件类型代码，而第二位则被复用为联动的 ID，该联动 ID 有软件给设备同步联动设置时设定（通常为该联动在软件端数据库中的 ID 值）。
- (4) 事件点编号中的三种类型的值，可以根据门禁控制板上的丝印来对应，比如门编号为 1 对应控制板上的 Lock1,辅助输入 1 对应控制板上的 Aux In1,辅助输出 2 对应控制板上的 Aux Out2（注：设备型号不同，丝印内容可能不同，但数字是对应的）。
- (5) 在解析单条记录时，需要对 Buffer 中的字符串格式记录进行解析。