

# N-ScanHub 使用手册

## 一、简介

新大陆 SDK 软件开发包支持 Windows 平台和 linux 平台，提供 C/C++接口与新大陆设备交互，用户可通过该 SDK 开发包进行二次开发。通过 SDK，用户可以获取设备、发送指令、固件升级等常用功能。目录结构如下：

业务	说明
支持平台	Windows 平台和 Linux 平台
支持编程语言	C/C++
功能	获取设备、发送指令、固件升级、设备读写、开关、采集图片、拔插通知、获取数据通知等
SDK 组成	N-ScanHubForLinux 和 N-ScanHubForWindows
API 说明	N-ScanHubForLinux 和 N-ScanHubForWindows 使用同名的接口名称

## 二、N-ScanHubForLinux 简介

### 2.1 目录结构

N-ScanHubForLinux 提供了 linux 平台下的 API，其目录如下：

目录	说明
libusb	动态库所依赖的第三方 usb 库源码
include	头文件: N-ScanHub.h 内含所有接口说明
demo	demo 代码和可执行文件
help	帮助文档: N-ScanHub.pdf
Lib	编译好的 x64 平台动态库

### 2.2 编译过程

#### 2.2.1 编译第三方库 libusb-master

./configure

```

root@ubuntu:/media/psf/Home/Newland/scan/code/NLSDeviceMasterForLinux/NLSDeviceMaster/libusb-master# ./configure
checking for gcc... gcc
checking whether the C compiler works... yes
checking for C compiler default output file name... a.out
checking for suffix of executables...
checking whether we are cross compiling... no
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ISO C89... none needed
checking whether gcc understands -c and -o together... yes
checking for g++... g++
checking whether we are using the GNU C++ compiler... yes
checking whether g++ accepts -g... yes
checking for inline... inline
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a thread-safe mkdir -p... /usr/bin/mkdir -p
checking for gawk... no
checking for mawk... mawk
checking whether make sets $(MAKE)... yes
checking whether make supports the include directive... yes (GNU style)
checking whether make supports nested variables... yes
checking dependency style of gcc... gcc3
checking dependency style of g++... gcc3
checking build system type... x86_64-pc-linux-gnu

```

## make && make install

```

[root@centos-linux libusb-1.0.9]# make && make install
make all-recursive
make[1]: Entering directory '/libusb-1.0.9'
Making all in libusb
make[2]: Entering directory '/libusb-1.0.9/libusb'
CC libusb_1_0_la-core.lo
CC libusb_1_0_la-descriptor.lo
CC libusb_1_0_la-io.lo
CC libusb_1_0_la-sync.lo
CC libusb_1_0_la-linux_usbfs.lo
CC libusb_1_0_la-threads_posix.lo
CCLD libusb-1.0.la
make[2]: Leaving directory '/libusb-1.0.9/libusb'
Making all in doc
make[2]: Entering directory '/libusb-1.0.9/doc'
make[2]: Nothing to be done for 'all'.
make[2]: Leaving directory '/libusb-1.0.9/doc'
make[2]: Entering directory '/libusb-1.0.9'
make[2]: Leaving directory '/libusb-1.0.9'
make[1]: Leaving directory '/libusb-1.0.9'
Making install in libusb
make[1]: Entering directory '/libusb-1.0.9/libusb'
make[2]: Entering directory '/libusb-1.0.9/libusb'
test -z "/usr/local/lib" || /bin/mkdir -p "/usr/local/lib"
/bin/sh ./libtool --mode=install /bin/install -c libusb-1.0.la '/usr/local/lib'
libtool: install: /bin/install -c .libs/libusb-1.0.so.0.1.0 /usr/local/lib/libusb-1.0.so.0.1.0
libtool: install: (cd /usr/local/lib && { ln -s -f libusb-1.0.so.0.1.0 libusb-1.0.so.0 || { rm -f libusb-1.0.so.0 && ln -s libusb-1.0.so.0.1.0 libusb-1.0.so.0; }; })
libtool: install: (cd /usr/local/lib && { ln -s -f libusb-1.0.so.0.1.0 libusb-1.0.so || { rm -f libusb-1.0.so && ln -s libusb-1.0.so.0.1.0 libusb-1.0.so; }; })
libtool: install: /bin/install -c .libs/libusb-1.0.lai /usr/local/lib/libusb-1.0.la
libtool: install: /bin/install -c .libs/libusb-1.0.a /usr/local/lib/libusb-1.0.a
libtool: install: chmod 644 /usr/local/lib/libusb-1.0.a
libtool: install: ranlib /usr/local/lib/libusb-1.0.a
libtool: finish: PATH="/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin:/sbin" ldconfig -n /usr/local/lib

Libraries have been installed in:
  /usr/local/lib

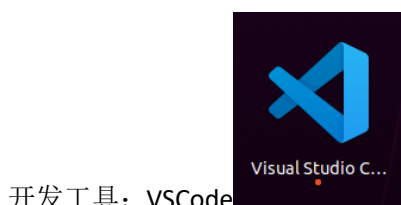
If you ever happen to want to link against installed libraries
in a given directory, LIBDIR, you must either use libtool, and
specify the full pathname of the library, or use the '-LLIBDIR'
flag during linking and do at least one of the following:
- add LIBDIR to the 'LD_LIBRARY_PATH' environment variable
  during execution
- add LIBDIR to the 'LD_RUN_PATH' environment variable

```

## 2.3 使用教程



测试设备：FM430



开发工具：VSCode



操作系统：Ubuntu



20.04.3 LTS

N-ScanHubForLinux demo 使用步骤:

1. 将动态库 N-ScanHub.so 复制到 demo 目录下以备调用
2. 开始调用 SDK 中的函数

```
1 #include "N-ScanHub.h" // Include header file 1. include head file

90 bool OpenDll()
91 {
92     g_handle = dlopen("./N-ScanHub.so", RTLD_NOW); 2. lib path and name

816 int main(int argc, char *argv[])
817 {
818     if (!OpenDll()) // Open dynamic library
819         return 0;
820
821     unsigned int deviceCounts = 0; 3. enum devices
822     HANDLEDEVLST hDeviceList = EnumDevices(&deviceCounts, ENUM_USB | ENUM_COM); // Enumerate device
823     printf("deviceCounts=%d,hDeviceList=%p\n", deviceCounts, hDeviceList);
824
825     for (unsigned int i = 0; i < deviceCounts; i++) // Get all device information
826     {
827         HANDLEDEV hDevice = OpenDevice(hDeviceList, i); // Open the device 4. open one device
828         printf("hDevice=%p, %s\n", hDevice, hDevice != NULL ? "succeed in opening the device" : "failed to open t
829
830         if (NULL == hDevice)
831             continue;
832         if (argc < 2)
833         {
834             //Write character string data
835             const char* strCmd = "QRYSYS"; // QRYSYS: System information
836             bool isWrited = Write(hDevice, strCmd, strlen(strCmd), true); // Write data
837             if(isWrited){
838                 char receivedData[1024] = { 0 };
```

3. 输入 `sudo ./N-ScanHubDemo` 即可运行

```
root@ubuntu:/media/psf/Home/Newland/scan/code/NLSDeviceMasterForLinux/NLSDeviceMaster/demo2# ls
N-ScanHubDemo N-ScanHub.so
root@ubuntu:/media/psf/Home/Newland/scan/code/NLSDeviceMasterForLinux/NLSDeviceMaster/demo2# ./N-ScanHubDemo
deviceCounts=1,hDeviceList=0x56128855b2a0
[Open] lpInfo->pOpenStream is not null
hDevice=0x56128855c4f0, succeed in opening the device
hDevice=0x56128855c4f0
res=1
system info:
0000@QRYSYSProduct Name: GALE
Firmware Version: UQ101.ST.H02.5
Decoder Version: 7.1.17
Hardware Version:
Serial Number:
OEM Serial Number:
Manufacturing Date:
;
CloseDevice hDevice=0x7ffc1a24d1f0,*hDevice=0x56128855c4f0
hDevice=nil,succeed in closing the device
handleDeviceList=(nil)
```

## 2.4 完整的 demo 示例

```
#include <stdio.h>
#include <dlfcn.h>
#include <cstring>
#include <stdlib.h>
#include "N-ScanHub.h" // Include header file
```

```

#include <fstream>
#include <unistd.h>
using namespace std;
static void *g_handle = NULL;          // Dynamic library handle

// Read device data
void ReadCallback(const HANDLEDEV hDevice, const char* buf, int len)
{
    printf("Callback hDevice=%p,receivedDataLen=%d\nreceivedData=%s\n", hDevice, len, buf);
}

// Monitoring device state change
void DevStatChangeCallback(const HANDLEDEV hDevice, bool isDevExisted)
{
    if (isDevExisted)
        printf("hDevice=%p, device is pushed in\n", hDevice);
    else
        printf("hDevice=%p, device is pushed out\n", hDevice);
}

/**
 * @brief Open the dynamic library.
 * @return Return the result of opening the dynamic library. true: succeed in opening the
dynamic library; false: failed to open the dynamic library.
 */
bool Opendl()
{
    g_handle = dlopen("./N-ScanHub.so", RTLD_NOW);
    if (!g_handle)
    {
        fprintf(stderr, "%s\n", dlerror());
        return false;
    }

    return true;
}

/**
 * @brief Close dynamic library
 */
void Closedl()
{
    if (g_handle != NULL)
        dlclose(g_handle); // Close dynamic library calling handle
}

```

```
}
```

```
bool GetPicData(const HANDLEDEV hDevice, unsigned char* imgBuf, int imgBufLen)
{
    if (g_handle == NULL)
        return false;

    char *error = NULL;
    typedef bool (*pf_t)(const HANDLEDEV, unsigned char*, int); // Declare function pointer
type
    pf_t pf = (pf_t)dlsym(g_handle, "nl_GetPicData");

    if ((error = dlerror()) != NULL)
    {
        fprintf(stderr, "%s\n", error);
        return false;
    }

    bool ret = pf(hDevice, imgBuf, imgBufLen);
    return ret;
}
```

```
bool GetPicDataByConfig(const HANDLEDEV hDevice, STImgParam imgParam, unsigned char*
imgBuf, unsigned int *imgBufLen, STImgResolution *imgR)
{
    if (g_handle == NULL)
        return false;

    char *error = NULL;
    typedef bool (*pf_t)(const HANDLEDEV, STImgParam, unsigned char*, unsigned int *,
STImgResolution *); // Declare function pointer type
    pf_t pf = (pf_t)dlsym(g_handle, "nl_GetPicDataByConfig");

    if ((error = dlerror()) != NULL)
    {
        fprintf(stderr, "%s\n", error);
        return false;
    }

    bool ret = pf(hDevice, imgParam, imgBuf, imgBufLen, imgR);
    return ret;
}
```

```
IMG_TYPE GetDeviceImageColorType(const HANDLEDEV hDevice, STImgResolution* imgResOut,
```

```

unsigned int *imgLen)
{
    if (g_handle == NULL)
        return TYPE_UNKNOWN;

    char *error = NULL;
    typedef IMG_TYPE (*pf_t)(const HANDLEDEV, STImgResolution*, unsigned int *); // Declare
function pointer type
    pf_t pf = (pf_t)dlsym(g_handle, "nl_GetDeviceImageColorType");

    if ((error = dlerror()) != NULL)
    {
        fprintf(stderr, "%s\n", error);
        return TYPE_UNKNOWN;
    }

    IMG_TYPE ret = pf(hDevice, imgResOut, imgLen);
    return ret;
}

```

```

bool ConvertImageColorSpace(const HANDLEDEV hDevice, unsigned char* imgBufIn, long
imgBufInLen, STImgResolution imgResIn, unsigned char* imgBufOut)
{
    if (g_handle == NULL)
        return false;

    char *error = NULL;
    typedef bool (*pf_t)(const HANDLEDEV, unsigned char*, long, STImgResolution, unsigned
char*); // Declare function pointer type
    pf_t pf = (pf_t)dlsym(g_handle, "nl_ConvertImageColorSpace");

    if ((error = dlerror()) != NULL)
    {
        fprintf(stderr, "%s\n", error);
        return false;
    }

    bool ret = pf(hDevice, imgBufIn, imgBufInLen, imgResIn, imgBufOut);
    return ret;
}

```

```

bool GetPicSize(const HANDLEDEV hDevice, unsigned int* width, unsigned int* height)
{
    if (g_handle == NULL)

```

```

        return false;

    char *error = NULL;
    typedef unsigned (*pf_t)(const HANDLEDEV, unsigned int*, unsigned int*); // Declare
function pointer type
    pf_t pf = (pf_t)dlsym(g_handle, "nl_GetPicSize");

    if ((error = dlerror()) != NULL)
    {
        fprintf(stderr, "%s\n", error);
        return false;
    }

    bool ret = pf(hDevice, width, height);
    return ret;
}

```

```

unsigned int Read(const HANDLEDEV hDevice, char* buf, unsigned int len, unsigned int timeout)
{
    if (g_handle == NULL)
        return false;

    char *error = NULL;
    typedef unsigned int (*pf_t)(const HANDLEDEV, char*, unsigned int, unsigned int); //
Declare function pointer type
    pf_t pf = (pf_t)dlsym(g_handle, "nl_Read");

    if ((error = dlerror()) != NULL)
    {
        fprintf(stderr, "%s\n", error);
        return false;
    }

    unsigned int ret = pf(hDevice, buf, len, timeout);
    return ret;
}

```

```

bool GetCommandResponse(const HANDLEDEV hDevice, const char* command, unsigned int
commandLen, char* response, int *responseLen, unsigned int timeout, bool isPacked, bool isHex)
{
    if (g_handle == NULL)
        return false;

    char *error = NULL;

```

```

typedef bool (*pf_t)(const HANDLEDEV, const char*, unsigned int, char*, int*, unsigned int,
bool, bool); // Declare function pointer type
pf_t pf = (pf_t)dlsym(g_handle, "nl_GetCommandResponse");

if ((error = dlerror()) != NULL)
{
    fprintf(stderr, "%s\n", error);
    return false;
}
printf("hDevice=%p\n", hDevice);

bool isSended = pf(hDevice, command, commandLen, response, responseLen, timeout,
isPacked, isHex);
return isSended;
}

bool Write(const HANDLEDEV hDevice, const char* data, unsigned int len, bool isPacked = true)
{
    if (g_handle == NULL)
        return false;

    char *error = NULL;
    typedef bool (*pf_t)(const HANDLEDEV, const char*, unsigned int, bool); // Declare function
pointer type
    pf_t pf = (pf_t)dlsym(g_handle, "nl_Write");

    if ((error = dlerror()) != NULL)
    {
        fprintf(stderr, "%s\n", error);
        return false;
    }
    printf("hDevice=%p\n", hDevice);

    bool isSended = pf(hDevice, data, len, isPacked);
    return isSended;
}

bool WriteAsHex(const HANDLEDEV hDevice, const char* data, bool isPacked = false)
{
    if (g_handle == NULL)
        return false;

    char *error = NULL;
    typedef bool (*pf_t)(const HANDLEDEV, const char*, bool); // Declare function pointer type

```



```

    pf_t pf = (pf_t)dlsym(g_handle, "nl_WriteAsHex");

    if ((error = dlerror()) != NULL)
    {
        fprintf(stderr, "%s\n", error);
        return false;
    }
    printf("hDevice=%p\n", hDevice);

    bool isSended = pf(hDevice, data, isPacked);
    return isSended;
}

T_CommunicationResult SendCommand(const HANDLEDEV hDevice, const char* command,
unsigned int commandLen)
{
    if (g_handle == NULL)
        return T_CommunicationResult::SendError;

    char *error = NULL;
    typedef T_CommunicationResult (*pf_t)(const HANDLEDEV, const char*, unsigned int); //
Declare function pointer type
    pf_t pf = (pf_t)dlsym(g_handle, "nl_SendCommand");

    if ((error = dlerror()) != NULL)
    {
        fprintf(stderr, "%s\n", error);
        return T_CommunicationResult::SendError;
    }
    printf("hDevice=%p\n", hDevice);

    T_CommunicationResult result = pf(hDevice, command, commandLen);
    return result;
}

T_CommunicationResult SendCommandAsHex(const HANDLEDEV hDevice, const char* command,
unsigned int commandLen)
{
    if (g_handle == NULL)
        return T_CommunicationResult::SendError;

    char *error = NULL;
    typedef T_CommunicationResult (*pf_t)(const HANDLEDEV, const char*, unsigned int); //
Declare function pointer type

```

```

    pf_t pf = (pf_t)dlsym(g_handle, "nl_SendCommandAsHex");

    if ((error = dlerror()) != NULL)
    {
        fprintf(stderr, "%s\n", error);
        return T_CommunicationResult::SendError;
    }
    printf("hDevice=%p\n", hDevice);

    T_CommunicationResult result = pf(hDevice, command, commandLen);
    return result;
}

void SetListener(const HANDLEDEV hDevice, readCallback callback)
{
    if (g_handle == NULL)
        return ;

    char *error = NULL;
    typedef bool (*pf_t)(const HANDLEDEV, readCallback); // Declare function pointer type
    pf_t pf = (pf_t)dlsym(g_handle, "nl_SetListener");

    if ((error = dlerror()) != NULL)
    {
        fprintf(stderr, "%s\n", error);
        return ;
    }
    printf("hDevice=%p\n", hDevice);

    pf(hDevice, callback);
}

void StopListener(const HANDLEDEV hDevice)
{
    if (g_handle == NULL)
        return;

    char *error = NULL;
    typedef bool (*pf_t)(const HANDLEDEV); // Declare function pointer type
    pf_t pf = (pf_t)dlsym(g_handle, "nl_StopListener");

    if ((error = dlerror()) != NULL)
    {
        fprintf(stderr, "%s\n", error);
    }
}

```

```

        return;
    }
    printf("hDevice=%p\n", hDevice);

    pf(hDevice);
}

bool ReadDevCfgToXml(const HANDLEDEV hDevice, const char* cfgFilePath)
{
    if (g_handle == NULL)
        return false;

    char *error = NULL;
    typedef bool (*pf_t)(const HANDLEDEV, const char*); // Declare function pointer type
    pf_t pf = (pf_t)dlsym(g_handle, "nl_ReadDevCfgToXml");

    if ((error = dlerror()) != NULL)
    {
        fprintf(stderr, "%s\n", error);
        return false;
    }
    printf("hDevice=%p\n", hDevice);

    bool isok = pf(hDevice, cfgFilePath);
    return isok;
}

bool WriteCfgToDev(const HANDLEDEV hDevice, const char* cfgFilePath)
{
    if (g_handle == NULL)
        return false;

    char *error = NULL;
    typedef bool (*pf_t)(const HANDLEDEV, const char*); // Declare function pointer type
    pf_t pf = (pf_t)dlsym(g_handle, "nl_WriteCfgToDev");

    if ((error = dlerror()) != NULL)
    {
        fprintf(stderr, "%s\n", error);
        return false;
    }
    printf("hDevice=%p\n", hDevice);

    bool isok = pf(hDevice, cfgFilePath);

```

```

        return isok;
    }

void SetCbDevStatusChanged(const HANDLEDEV hDevice, DevStatChgCallback callback)
{
    if (g_handle == NULL)
        return;

    char *error = NULL;
    typedef bool (*pf_t)(const HANDLEDEV, DevStatChgCallback); // Declare function pointer
type
    pf_t pf = (pf_t)dlsym(g_handle, "nl_SetCbDevStatusChanged");

    if ((error = dlerror()) != NULL)
    {
        fprintf(stderr, "%s\n", error);
        return;
    }
    printf("hDevice=%p\n", hDevice);

    pf(hDevice, callback);
}

bool UpdateKernelDevice(const HANDLEDEV hDevice, const char* strFileName, unsigned int
reserved = 0, unsigned int* errorUpdate = 0)
{
    if (g_handle == NULL)
        return false;

    char *error = NULL;
    typedef unsigned (*pf_t)(const HANDLEDEV, const char *, unsigned int, unsigned int*); //
Declare function pointer type
    pf_t pf = (pf_t)dlsym(g_handle, "nl_UpdateKernelDevice");

    if ((error = dlerror()) != NULL)
    {
        fprintf(stderr, "%s\n", error);
        return false;
    }

    bool isUpdated = pf(hDevice, strFileName, 0, errorUpdate);
    return isUpdated;
}

```

```

bool CloseDevice(HANDLEDEV* hDevice)
{
    if (g_handle == NULL)
        return false;

    char *error = NULL;
    typedef unsigned (*pf_t)(HANDLEDEV*);          // Declare function pointer type
    pf_t pf = (pf_t)dlsym(g_handle, "nl_CloseDevice");
    if ((error = dlerror()) != NULL)
    {
        fprintf(stderr, "%s\n", error);
        return false;
    }

    printf("CloseDevice hDevice=%p,*hDevice=%p\n",hDevice,*hDevice);
    bool isClosed = pf(hDevice);
    return isClosed;
}

HANDLEDEV OpenDevice(const HANDLEDEVLST hDeviceList, unsigned int index, T_Protocol
porotocol = Nlscan)
{
    if (g_handle == NULL)
        return NULL;

    char *error = NULL;
    typedef HANDLEDEV (*pf_t)(const HANDLEDEVLST, unsigned, T_Protocol); // Declare
function pointer type
    pf_t pf = (pf_t)dlsym(g_handle, "nl_OpenDevice");
    if ((error = dlerror()) != NULL)
    {
        fprintf(stderr, "%s\n", error);
        return NULL;
    }

    HANDLEDEV isOpened = pf(hDeviceList, index, porotocol);
    return isOpened;
}

void ReleaseDevices(HANDLEDEVLST* deviceList)
{
    if (g_handle == NULL)
        return ;
}

```

```

char *error = NULL;
typedef void (*pf_t)(HANDLEDEVLIST*);    // Declare function pointer type
pf_t pf = (pf_t)dlsym(g_handle, "nl_ReleaseDevices");

if ((error = dlerror()) != NULL)
{
    fprintf(stderr, "%s\n", error);
    return ;
}

return pf(deviceList);
}

HANDLEDEVLIST EnumDevices(unsigned int* deviceCounts)
{
    if (g_handle == NULL)
        return 0;

    char *error = NULL;
    typedef HANDLEDEVLIST (*pf_t)(unsigned int*);    // Declare function pointer type
    pf_t pf = (pf_t)dlsym(g_handle, "nl_EnumDevices");

    if ((error = dlerror()) != NULL)
    {
        fprintf(stderr, "%s\n", error);
        return 0;
    }

    return pf(deviceCounts);
}

bool SavePicDataToFile(const char* bmpName, unsigned char* imgBuf, int width, int height, int
biBitCount = 8)
{
    if (g_handle == NULL)
        return false;

    char *error = NULL;
    typedef bool (*pf_t)(const char*, unsigned char*, int, int, int);    // Declare function
pointer type
    pf_t pf = (pf_t)dlsym(g_handle, "nl_SavePicDataToFile");

    if ((error = dlerror()) != NULL)
    {

```

```

        fprintf(stderr, "%s\n", error);
        return false;
    }

    bool isSaved = pf(bmpName, imgBuf, width, height, biBitCount);
    return isSaved;
}

int main(int argc, char *argv[])
{
    if (!Opendl()) // Open dynamic library
        return 0;

#ifdef 1
    unsigned int deviceCounts = 0;
    HANDLEDEVLST hDeviceList = EnumDevices(&deviceCounts); // Enumerate device
    printf("deviceCounts=%d,hDeviceList=%p\n", deviceCounts, hDeviceList);

    for (unsigned int i = 0; i < deviceCounts; i++) // Get all device information
    {
        HANDLEDEV hDevice = OpenDevice(hDeviceList, i); // Open the device
        printf("hDevice=%p, %s\n", hDevice, hDevice != NULL ? "succeed in opening the
device" : "failed to open the device");

        if (NULL == hDevice)
            continue;
        if (argc < 2) {
            // Write character string data
            //const char* strCmd = "IMGFMT"; // QRYSYS: System information
            const char* strCmd = "QRYSYS"; // QRYSYS: System information
            char receivedData[1024] = { 0 };
            int recvlen = 0;
            bool res = GetCommandResponse(hDevice, strCmd, strlen(strCmd), receivedData,
&recvlen, 0, true, false);
            printf("res=%d\n", res);
            printf("system info: \n%s\n", receivedData);
        }
        // Write character string data
        // const char* strCmd = "QRYSYS"; // QRYSYS: System information
        // bool isWrited = Write(hDevice, strCmd, strlen(strCmd), true); // Write data
        // if(isWrited){
        //     char receivedData[1024] = { 0 };
        //     unsigned int nRet = Read(hDevice, receivedData, sizeof(receivedData), 0); //
Read data

```

```

//          printf("nRet=%d, receivedData=%s\n", nRet, receivedData);
//      }
      if (argc >= 2 && strcmp(argv[1], "--WriteAsHex") == 0) // Write data to the device in
      HEX character string
      {
          // Write hex character string data
          const char* strCmdhEX = "7e 01 30 30 30 30 40 51 52 59 53 59 53 3b 03"; //
      System information
          bool isWrited = WriteAsHex(hDevice, strCmdhEX, false); // Write data
          if(isWrited){
              char receivedData[1024] = { 0 };
              unsigned int nRet = Read(hDevice, receivedData, sizeof(receivedData), 0); //
      Read data
              printf("nRet=%d, receivedData=%s\n", nRet, receivedData);
          }
      }
      else if (argc >= 2 && strcmp(argv[1], "--SendCommand") == 0) // Send control
      commands to the device and obtain the returned information
      {
          const char* strCmd = "QRYSYS"; // QRYSYS: System information
          T_CommunicationResult result = SendCommand(hDevice, strCmd, strlen(strCmd));
      // Send commands
          printf("result=%d\n", result);
      }
      else if (argc >= 2 && strcmp(argv[1], "--SendCommandAsHex") == 0) // Send control
      commands to the device in the form of HEX character string and get the returned information.

      {
          const char* strCmd = "51 52 59 53 59 53 "; // QRYSYS: System information
          T_CommunicationResult result = SendCommandAsHex(hDevice, strCmd,
      strlen(strCmd)); // Send commands
          printf("result=%d\n", result);
      }
      else if (argc >= 2 && strcmp(argv[1], "--GetPicture") == 0) // Get the device image
      {
          unsigned int imgWidth = 0, imgHeight = 0;
          bool isGetPicSizeOK = GetPicSize(hDevice, &imgWidth, &imgHeight); // Get the
      image width and height
          if (isGetPicSizeOK && imgWidth > 0 && imgHeight > 0)
          {
              printf("imgWidth=%d,imgHeight=%d\n", imgWidth, imgHeight);
              const int RECV_BUFFER_SIZE = imgWidth * imgHeight * 4;
              unsigned char* recvBuffer = (unsigned char*)malloc(RECV_BUFFER_SIZE);

```



```

        STImgParam imgParam;
        memset(&imgParam, 0, sizeof(STImgParam));
        imgParam.f = 3;
//        imgParam.q = 3;
        //strcpy(imgParam.b, "0001000100100010");
        STImgResolution imgR[4];
        memset(imgR, 0, sizeof(STImgResolution) * 4);
        unsigned int nRealLen = 0;
        bool isOK = GetPicDataByConfig(hDevice, imgParam, recvBuffer, &nRealLen,
imgR); // Get the image data
        printf("isOK=%d,      recvBuffer1=%02x      recvBuffer1=%02x\n",      isOK,
recvBuffer[RECV_BUFFER_SIZE - 2], recvBuffer[RECV_BUFFER_SIZE - 1]);

```

```

        if (isOK) {
            if (imgParam.t == 2) {
                for (int i = 0; i < 4; i++) {
                    printf("imgR[%d] width=%d height=%d\n", i, imgR->width,
imgR->height);
                }
            }
            if (imgParam.f == 1) {
                FILE* fp = fopen("test3.bmp", "wb");
                fwrite(recvBuffer, 1, nRealLen, fp);
                fclose(fp);
            }
            else if (imgParam.f == 2) {
                FILE* fp = fopen("test4.jpg", "wb");
                fwrite(recvBuffer, 1, nRealLen, fp);
                fclose(fp);
            }
            else if (imgParam.f == 3) {
                FILE* fp = fopen("test5.tiff", "wb");
                fwrite(recvBuffer, 1, nRealLen, fp);
                fclose(fp);
            }
            else if (imgParam.f == 4) {
                FILE* fp = fopen("test6.bmp", "wb");
                fwrite(recvBuffer, 1, nRealLen, fp);
                fclose(fp);
            }
            else if (imgParam.f == 0) {
                STImgResolution imgResIn, imgResOut;

```

```

        imgResIn.width = imgWidth;
        imgResIn.height = imgHeight;
        unsigned int imgLen = 0;
        IMG_TYPE type = GetDeviceImageColorType(hDevice, &imgResOut,
&imgLen);

        printf("ConvertImageColorSpace IMG_TYPE=%d\n", type);
        if (type == TYPE_COLOR) {
            unsigned char* outBuf = (unsigned char*)malloc(imgLen);
            bool res = ConvertImageColorSpace(hDevice, recvBuffer,
RECV_BUFFER_SIZE, imgResIn, outBuf);
            printf("ConvertImageColorSpace res=%d\n", res);
            if(res == false)
                return 0;
            SavePicDataToFile("test2.bmp", outBuf, imgResOut.width,
imgResOut.height, 24); // Save image
            SavePicDataToFile("test2.jpg", outBuf, imgResOut.width,
imgResOut.height, 23); // Save image
        }
        else {
            SavePicDataToFile("test1.bmp", recvBuffer, imgResOut.width,
imgResOut.height, 8); // Save image
            SavePicDataToFile("test1.jpg", recvBuffer, imgResOut.width,
imgResOut.height, 13); // Save image
        }
    }

        //nl_SavePicDataToFile("test1.bmp", recvBuffer, imgWidth, imgHeight, 8);
// Save image
    }
    free(recvBuffer);
    recvBuffer = NULL;
}
}
else if (argc >= 2 && strcmp(argv[1], "--SetListener") == 0) // Asynchronous reading of
device data
{
    SetListener(hDevice, ReadCallback);
    sleep(5);
    StopListener(hDevice);
}
else if (argc >= 3 && strcmp(argv[1], "--ReadDevCfgToXml") == 0) // Read the
configuration from the device and save it to the xml file.
{

```

```

        bool isok = ReadDevCfgToXml(hDevice, argv[2]);
        printf(isok ? "ReadDevCfgToXml succeeded\n" : "ReadDevCfgToXml failed\n");
    }
    else if (argc >= 3 && strcmp(argv[1], "--WriteCfgToDev") == 0) // Read the configuration
from the device and save it to the xml file.
    {
        bool isok = WriteCfgToDev(hDevice, argv[2]);
        printf(isok ? "WriteCfgToDev succeeded\n" : "WriteCfgToDev failed\n");
    }
    else if (argc >= 2 && strcmp(argv[1], "--SetCbDevStatusChanged") == 0) // Set the
callback function when the device status changes.
    {
        SetCbDevStatusChanged(hDevice, DevStatChangeCallback);
    }
    else if (argc >= 3 && strcmp(argv[1], "--UpdateFirmware") == 0) // Update device
    {
        unsigned updateError = -1;
        bool isUpdated = UpdateKernelDevice(hDevice, argv[2], 0, &updateError); //
Firmware update
        printf("updateError=%d,%s\n", updateError, isUpdated ? "succeed in updating the
firmware " : "failed to update the firmware");

        switch (updateError)
        {
            case Success:
                printf("The firmware update is normal.\n");
                break;
            case FileNameExtError:
                printf("file name error\n");
                break;
        }
    }

    bool isClosed = CloseDevice(&hDevice); // Close the device
    printf("hDevice=%p,%s\n", hDevice, isClosed ? "succeed in closing the device" : "failed
to close the device");
}

ReleaseDevices(&hDeviceList); // Release the device list handle
printf("handleDeviceList=%p\n", hDeviceList);
#endif
Closedl();
return 0;
}

```

### 三、接口说明


Windows 和 linux 下的 SDK 使用同名的 API，具体功能如下：

功能列表	
函数	说明
<code>HANDLEDEVLST nl_EnumDevices(int* deviceCount, EnumType = ENUM_ALL);</code>	brief 枚举设备. Param[in] enumType 枚举类型,默认枚举所有类型设备 param[out] deviceCount 设备数 return 设备列表句柄，返回非 NULL，表示设备列表存在。返回 NULL，表示设备列表不存在
<code>void nl_ReleaseDevices(HANDLEDEVLST* hDeviceList);</code>	brief 释放设备列表句柄. param[in] hDeviceList 设备列表句柄
<code>HANDLEDEV nl_OpenDevice(const HANDLEDEVLST hDeviceList, unsigned int index, T_Porotocol porotocol = Nlscan);</code>	brief 打开设备列表指定索引的设备. param[in] hDeviceList 设备列表句柄 param[in] index 索引 param[in] porotocol 厂家协议 return 设备句柄，返回非 NULL，表示打开成功。返回 NULL，表示打开失败
<code>bool nl_Write(const HANDLEDEV hDevice, const char* data, unsigned int len, bool isPacked = true);</code>	brief 向设备写数据. param[in] hDevice 设备句柄 param[in] data 数据 param[in] len 数据长度 param[in] isPacked 数据是否打包 return 是否写数据成功，返回 true，表示写数据成功，返回 false，表示写数据失败
<code>bool nl_WriteAsHex(const HANDLEDEV hDevice, const char* data, bool isPacked = false);</code>	brief 以 HEX 字符串方式向设备写数据. param[in] hDevice 设备句柄 param[in] data 数据 param[in] isPacked 数据是否打包 return 是否写数据成功，返回 true，表示写数据成功，返回 false，表示写数据失败
<code>T_CommunicationResult nl_SendCommand(const HANDLEDEV hDevice, const char* command, unsigned int commandLen);</code>	brief 向设备发送控制指令(接口内部会根据不同协议打包指令). param[in] hDevice 设备句柄 param[in] command 指令

	param[in] commandLen 指令长度 return 通讯结果
T_CommunicationResult nl_SendCommandAsHex(const HANDLEDEV hDevice, const char* command, unsigned int commandLen);	brief 以 HEX 字符串方式向设备发送控制指令(接口内部会根据不同协议打包指令). param[in] hDevice 设备句柄 param[in] command 指令 param[in] commandLen 指令长度 return 通讯结果
unsigned int nl_Read(const HANDLEDEV hDevice, char* buf, unsigned int len, unsigned int timeout);	brief 读设备数据. param[in] hDevice 设备句柄 param[out] buf 设备返回的数据 param[in] len 接收的长度 param[in] timeout 超时时间, 该值为 0 时, 表示一直读到设备无数据返回 return 设备返回的数据长度
void nl_SetListener(const HANDLEDEV hDevice, readCallback callback);	brief 设置监听. param[in] hDevice 设备句柄 param[in] callback 回调函数
bool nl_StopListener(const HANDLEDEV hDevice);	brief 停止监听设备数据. param[in] hDevice 设备句柄 return 是否停止成功, 返回 true, 表示停止成功, 返回 false, 表示停止失败
bool nl_GetPicSize(const HANDLEDEV hDevice, unsigned int* width, unsigned int* height);	brief 获取设备图像尺寸. param[in] hDevice 设备句柄 param[out] width 图片的宽 param[out] height 图片的高 return 获取设备图像尺寸是否成功, 返回 true, 表示成功, 返回 false, 表示失败
bool nl_GetPicData(const HANDLEDEV hDevice, unsigned char* imgBuf, int imgBufLen);	brief 获取设备图像. param[in] hDevice 设备句柄 param[out] imgBuf 图像的数据 param[in] imgBufLen 图像的数据长度 return 获取设备图像是否成功, 返回 true, 表示成功, 返回 false, 表示失败
bool nl_UpdateKernelDevice(const HANDLEDEV hDevice, const char* strFileName, unsigned int reserved = 0, unsigned int* error = 0);	brief 更新设备 (固件升级成后需要重新枚举设备, 才能继续使用设备句柄。 固件升级过程中会重启设备, 所以在升级之前会关闭设备状态监测, 如有需要, 请在升级完成后重新调用 nl_SetCbDevStatusChanged) .

	param[in] hDevice 设备句柄 param[in] strFileName 固件文件路径 param[in] reserved 保留字段 param[out] error 更新失败后返回的失败编号 return 是否更新成功，返回 <b>true</b> ，表示更新成功，返回 <b>false</b> ，表示更新失败
bool nl_CloseDevice(HANDLEDEV* hDevice);	brief 关闭设备. param[in] hDevice 设备句柄 return 是否关闭成功，返回 <b>true</b> ，表示关闭成功，返回 <b>false</b> ，表示关闭失败
bool nl_SavePicDataToFile(const char* bmpName, unsigned char* imgBuf, int width, int height, int flag);	brief 将采集到的图像数据封装成 BMP 格式并保存为文件.(图像数据为 raw 原始数据时,才需要调用此接口,如果获取的图像数据已经是 bmp 或 jpg 等成熟数据,直接以二进制方式保存文件即可,无需调用此接口) param[in] bmpName bmp 文件名 param[in] imgBuf 图像缓冲数据 param[in] width 图像宽 param[in] height 图像高 param[in] flag 图像参数标识 保存文件为 <b>bmp</b> 位图时,表示图像位深度,取值:8 或 24 保存文件为 <b>jpg</b> 时,表示图像质量高低 1. 黑白图片: (10-Low, 11-Middle, 12-High, 13-Highest) 2. 彩色图片: (20-Low, 21-Middle, 22-High, 23-Highest) return 是否保存成功，返回 <b>true</b> ，表示保存成功，返回 <b>false</b> ，表示保存失败
T_DeviceStatus nl_GetDevStatus(const HANDLEDEV hDevice);	brief 获取设备当前状态. param[in] hDevice 设备句柄 return 设备状态
bool nl_ReadDevCfgToXml(const HANDLEDEV hDevice, const char* cfgFilePath);	brief 从设备读取配置，并保存到 xml 文件. param[in] hDevice 设备句柄 param[in] cfgFilePath 配置文件路径 return 是否保存成功，返回 <b>true</b> ，表示保存成功，返回 <b>false</b> ，表示保存失败
bool nl_WriteCfgToDev(const HANDLEDEV hDevice, const char* cfgFilePath);	brief 将配置文件信息写入设备（写入后，需要延时 3 秒左右，才能执行后续操作）。 param[in] hDevice 设备句柄 param[in] cfgFilePath 配置文件路径

	<p>return 是否写入成功, 返回 true, 表示写入成功, 返回 false, 表示写入失败</p>
<pre>void nl_SetCbDevStatusChanged(const HANDLEDEV hDevice, DevStatChgCallba ck callback);</pre>	<p>brief 设备状态发生变化时的回调函数.(只针对串口和 usb 物理连接有效,网络设备请自行通过 ip 和端口进行检测)</p> <p>param[in] hDevice 设备句柄</p> <p>param[in] isDevExisted 设备是否存在</p>
<pre>bool nl_GetCommandResponse(const HANDLEDEV hDevice, const char* command, unsigned int commandLen, char* response, int *responseLen, unsigned int timeout, bool isPacked, bool isHex);</pre>	<p>brief 发送指令并接收返回指令</p> <p>param[in] hDevice 设备句柄</p> <p>param[in] command 发送指令数据</p> <p>param[in] commandLen 发送指令长度</p> <p>param[out]response 接收指令数据,需要足够大的空间进行接收</p> <p>param[in/out]responseLen</p> <p>[in]response 分配的空间长度</p> <p>[out]接收指令数据的真实长度</p> <p>param[in]timeout 超时时间</p> <p>param[in]isPacked 是否需要打包</p> <p>param[in]isHex 是否发送 HEX 指令数据</p> <p>return 是否收发成功, 返回 true, 表示成功, 返回 false, 表示失败</p>
<pre>bool nl_GetPicDataByConfig(const HANDLEDEV hDevice, STImgParam imgParam, unsigned char* imgBuf, unsigned int *imgBufLen, STImgResolution* imgR);</pre>	<p>brief 根据参数获取图像数据</p> <p>param[in] hDevice 设备句柄</p> <p>param[in] imgParam 图像参数结构</p> <p>T, 类型: 0T - 实时图像 (最后一次拍摄的图像), 1T - 解码成功的图像</p> <p>F, 图像格式: 0F - 原始图像 (Raw data), 1F - BMP 格式, 2F - JPEG 格式</p> <p>Q, JPEG 格式的图像质量: 0Q - Low, 1Q - Middle, 2Q - High, 3Q - Highest</p> <p>其他参数暂时保留,初始化为 0</p> <p>param[out]imgBuf 返回的图像数据,需要足够大的空间进行接收</p> <p>param[in/out] imgBufLen</p> <p>[in] imgBuf 分配的空间长度</p> <p>[out] 返回的图像数据真实长度</p> <p>param[out]imgR 保留参数,暂时无用.条码区域的四个端点坐标(如果有),需要提前申请 STImgResolution[4]数组</p> <p>return 是否接收成功, 返回 true, 表示成功 false, 表示失败</p>

 <pre> nl_GetDeviceImageColorType(const HANDLEDEV hDevice, STImgResolution* imgResOut, unsigned int * imgLen); </pre>	<p>brief 获取设备原始图像的图片类型</p> <p>param[in] hDevice 设备句柄</p> <p>param[out] imgResOut 原始图像的真实分辨率结构, 如果是彩色图像,是转换后的分辨率</p> <p>param[out] imgLen 图像数据的真实大小</p> <p>return 原始图像类型</p>
<pre> bool nl_ConvertImageColorSpace(const HANDLEDEV hDevice, unsigned char* imgBufIn, long imgBufInLen, STImgResolution imgResIn, unsigned char* imgBufOut); </pre>	<p>brief 原始图像色彩空间转换 nv12-&gt;bgr</p> <p>param[in] hDevice 设备句柄</p> <p>param[in] imgBufIn 原始图像信息</p> <p>param[in] imgBufInLen 原始图像数据长度</p> <p>param[in] imgResIn 原始图像的分辨率</p> <p>param[out] imgBufOut 转换后的图像数据</p> <p>return 是否接收成功, 返回 true, 表示成功 false, 表示失败</p>
<pre> bool nl_GetDeviceInfo(const HANDLEDEVLST hDeviceList, unsigned int index, STDeviceInfo* stNetDevInfo); </pre>	<p>brief 获取设备信息</p> <p>param[in] hDeviceList 设备句柄列表</p> <p>param[in] index 索引</p> <p>param[out] stNetDevInfo 设备信息结构</p> <p>return 是否获取成功, 返回 true, 表示成功 false, 表示失败</p>
<pre> bool nl_DevicelsOpenByHandle(const HANDLEDEV hDevice); </pre>	<p>brief 设备是否打开</p> <p>param[in] hDevice 设备句柄</p> <p>return 是否打开, 返回 true, 表示打开 false, 表示关闭</p>
<pre> bool nl_DevicelsOpenByList(const HANDLEDEVLST hDeviceList, unsigned int index); </pre>	<p>brief 设备是否打开</p> <p>param[in] hDeviceList 设备句柄列表</p> <p>param[in] index 索引</p> <p>return 是否打开, 返回 true, 表示打开 false, 表示关闭</p>
<pre> char *nl_GetLastError(); </pre>	<p>brief 获取最后一次操作的错误信息</p> <p>return 错误信息</p>
<pre> int nl_SetNetDeviceConfig(NET_SETTING_ TYPE type, char* inData,int inDataLen,int recTimeout,char* outdata); </pre>	<p>brief 设置网络设备配置信息</p> <p>param[in] type 配置类型</p> <p>param[in] inData 配置信息</p> <p>param[in] inDataLen 配置信息的长度</p> <p>param[in] recTimeout 超时时间</p> <p>param[in] outdata 返回信息</p> <p>return 0 成功 其他 失败</p>

以下为网络独立接口



int nl_CreateTcpService(int port, tcpServiceBack callback);	brief 创建网络服务端 param[in] port 端口 param[in] callback 回调函数 return 小于 0 失败
int nl_CloseClientSocket(int *socket);	brief 关闭客户端 socket 套接字 param[in] socket 客户端套接字 return 0 成功 其他 失败
int nl_ExitTcpService();	brief 退出网络服务端 return
int nl_connectToService(char* servicelp, int port, int* socket);	brief 连接网络服务端 param[in] servicelp 服务端 IP 地址 param[in] port 服务端端口 param[out] socket 网络套接字 return 0 成功 其他 失败
int nl_sendDataToSocket(int socket, char* buf, int buf_len);	brief 通过 socket 发送网络数据 param[in] socket 网络套接字 param[in] buf 发送数据 param[in] buf_len 发送数据长度 return 0 成功 其他 失败
int nl_readFromSocket(int socket, int nTimeout, char* outbuf, int *buflen);	brief 接收网络数据 param[in] socket 网络套接字 param[in] nTimeout 超时时间 param[in] outbuf 接收数据 param[in] buflen 接收数据长度 return 0 成功 其他 失败
int nl_getNetImgData(int socket, int T, int R, int F, int Q, char *imgData, int *realLen, IMG_TYPE* imgtype, int *width, int *heigh);	brief 通过网络获取图像数据 param[in] socket 网络套接字 param[in] T 图像类型, 0T - 实时图像 (最后一次拍摄的图像), 1T - 解码成功的图像 param[in] R 图像比率, 暂时保留, 初始化为 0 param[in] F 图像格式, 0F - 原始图像 (Raw data), 1F - BMP 格式, 2F - JPEG 格式 param[in] Q jpg 图像质量, 0Q - Low, 1Q - Middle, 2Q - High, 3Q - Highest param[out] imgData 图像数据 param[in/out] realLen [in] imgData 分配的空间长度 [out] 返回的图像数据真实长度 param[out] imgtype 图像类型

	param[out] width 分辨率宽 param[out] heigh 分辨率高 return 0 成功 其他 失败
--	---

枚举说明	
brief 异常类型.	
enum T_ErrorType	
{	
Success	= 0, ///< 无异常.
UnknownError	= 1, ///< 未知异常.
NotExistError	= 2, ///< 设备不存在.
NotOpenError	= 3, ///< 设备未打开.
AlreadyOpenError	= 4, ///< 设备已打开.
AccessDeniedError	= 5, ///< 设备拒绝访问.
NotInitializedError	= 6, ///< 设备未初始化.
InvalidParamsError	= 8, ///< 无效参数.
InvalidFileFormatError	= 9, ///< 无效文件格式.
FileNameExtError	= 10, ///< 文件名错误.
CommunicationError	= 11, ///< 通讯异常.
MallocError	= 12, ///< 内存分配错误.
UpdateFailedError	= 13, ///< 更新失败.
NoUpdateObjectError	= 14, ///< 无更新对象.
FileNotExistError	= 15, ///< 文件不存在.
BufferOverflowError	= 16, ///< 缓冲区溢出.
FileNotSuitableError	= 17, ///< 文件不适用.
DeviceNotUniqueError	= 18, ///< 设备不唯一.
NoConversionNeeded	= 19, ///< 图片不需要色彩转换
ConvertColorSpaceError	= 20, ///< 色彩转换错误
ParamError	= 21, ///< 参数错误
};	
brief 设备状态.	
enum T_DeviceStatus	
{	
Opened = 0,	///< 已打开.
NotOpened,	///< 未打开.
Closed,	///< 已关闭.

<pre> NotClosed,          ///&lt; 未关闭. Updating,           ///&lt; 升级中. Updated,            ///&lt; 升级结束. Writing,            ///&lt; 写数据中. Written,            ///&lt; 写数据结束. Reading,            ///&lt; 读数据中. ReadOK,             ///&lt; 读数据结束. GettingPicData,     ///&lt; 读图像数据中. GetPicDataOK,       ///&lt; 读图像数据结束. GettingPicColorType, ///&lt; 获取图像色彩类型. GetPicColorTypeOk,  ///&lt; 获取图像色彩类型结束. ConvergingColorSpace, ///&lt; 图像色彩转换. ConvertColorSpaceOK, ///&lt; 图像色彩转换结束. UnknownStatus       ///&lt; 未知状态. }; </pre>	
<pre> brief 指令发送结果. enum T_CommunicationResult {     SendError = 0,    ///&lt; 发送错误.     Support,          ///&lt; 支持指令.     Unsupport,        ///&lt; 不支持指令.     OutOfRange,       ///&lt; 数据的值不在支持范围.     UnknownResult,    ///&lt; 未知错误. }; </pre>	
<pre> brief 厂家协议. enum T_Porotocol {     Nlscan = 0, // 新大陆. }; </pre>	
<pre> Brief 图像色彩类型 enum IMG_TYPE {     TYPE_UNKNOW = 0, ///&lt; 未知类型     TYPE_GRAY = 1,   ///&lt; 黑白     TYPE_COLOR = 2   ///&lt; 彩色 }; </pre>	
<pre> Brief 设备类型. enum NL_DEVICE_TYPE {     DEV_TYPE_UNKNOW = 0, ///&lt; 未知     DEV_TYPE_USB = 1,    ///&lt; usb 设备     DEV_TYPE_COM = 2,    ///&lt; 串口设备     DEV_TYPE_NET = 3,    ///&lt; 网络设备 }; </pre>	
<pre> Brief 网络参数设置类型. enum NET_SETTING_TYPE { </pre>	

```
DEV_SETTING = 0,    ///< 设备网络参数
GROUP_SETTING = 1,  ///< 网络分组参数
};
```