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、关开、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

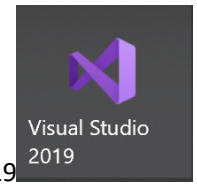
二、N-ScanHubForWindows 简介

2.1 目录结构

N-ScanHubForWindows 提供了 Windows 平台下的 API，其目录如下：

目录	说明
include	头文件: N-ScanHub.h 内含所有接口说明
lib/x64	64 位 N-ScanHub.dll、N-ScanHub.lib
lib/x86	32 位 N-ScanHub.dll、N-ScanHub.lib
demo	库文件和 Visual Studio2019 编写的 demo
help	帮助文档：N-ScanHub.pdf

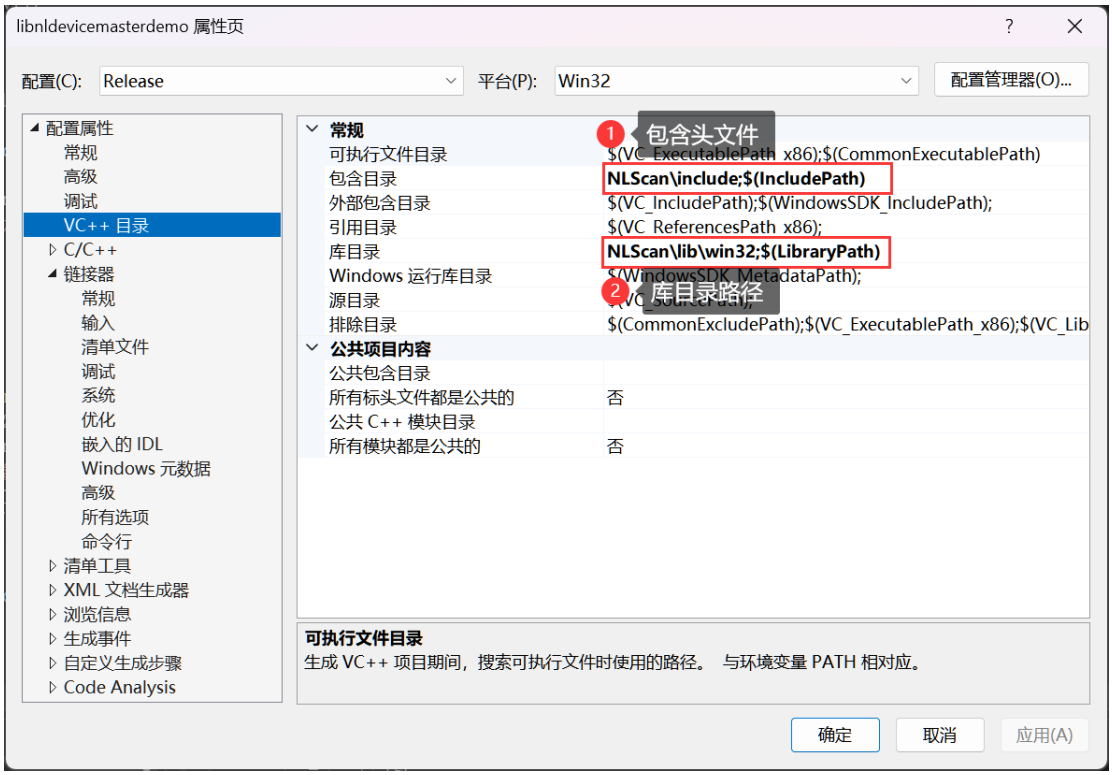
2.2 使用教程

测试设备：FM430  开发工具：VS2019 

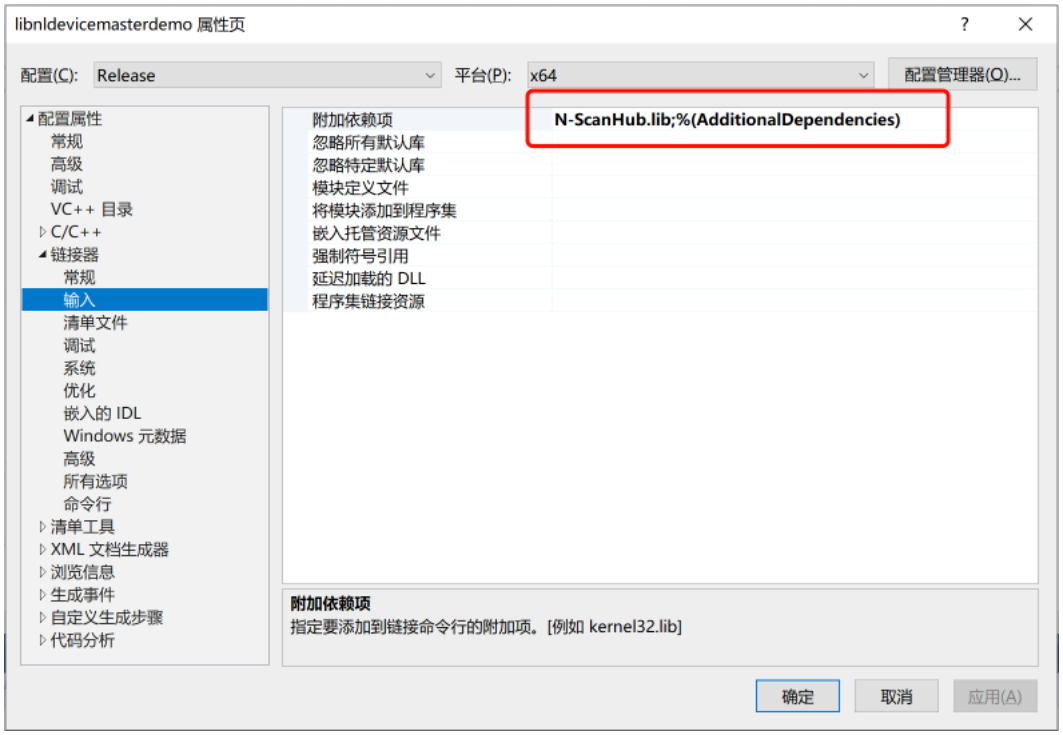
操作系统：Windows 10

N-ScanHubForWindows demo 使用步骤：

1.工程中包含头文件 N-ScanHub.h 和库 N-ScanHub.lib 路径

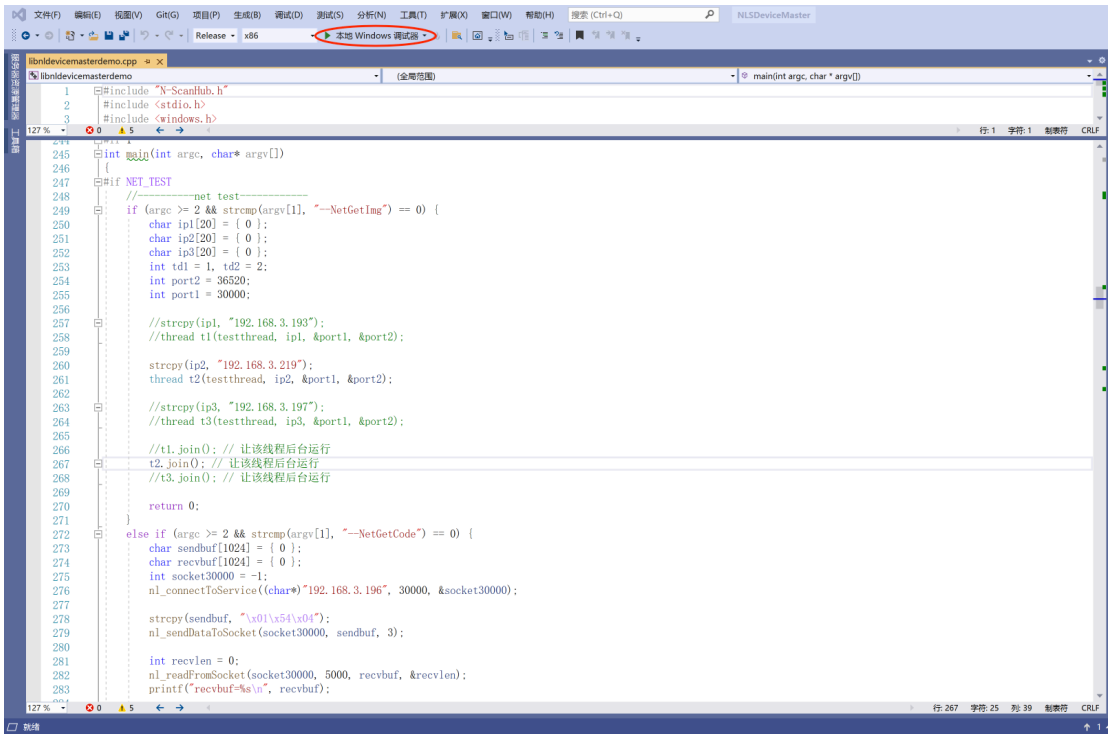


2.工程中添加库依赖 N-ScanHub.lib



3.开始调用 SDK 中的函数

4.开始运行：运行程序



效果如下：

2.3 完整的 demo 示例

```
#include "N-ScanHub.h"
#include <stdio.h>
#include <windows.h>
#include <chrono>
#include <vector>
#include <string>
```

```
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <thread>
#include "Debug.h"

using namespace std;

// Read device data
void __stdcall ReadCallback(const HANDLEDEV hDevice, const char* buf, int len)
{
	printf("-----------ReadCallback len=%d, buf=%s\n", len, buf);
	char data[2048] = { 0 };
	for (int i = 0; i < len; i++) {
		sprintf(data + i * 2, "%02X", buf[i]);
	}
	printf("-----------ReadCallback data=%s\n", data);
}

// Monitoring device status change
void __stdcall 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);
}


int main(int argc, char* argv[])
{

	int deviceCounts = 0;
	DbgPrintf("enum nl_EnumDevices begin\n");
	HANDLEDEVLST hDeviceList = nl_EnumDevices(&deviceCounts); // Enumerate device
	DbgPrintf("enum nl_EnumDevices end\n");
	printf("deviceCounts=%d\n", deviceCounts);

	for (int i = 0; i < deviceCounts; i++) // Get all device information
	{
		HANDLEDEV hDevice;
		hDevice = nl_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;

T_DeviceStatus status = nl_GetDevStatus(hDevice); // Get device status
printf("status=%d\n", status);
if (argc < 2) {
    // Write character string data
    const char* strCmd = "QRYSYS"; // QRYSYS: System information
    char receivedData[65565] = { 0 };
    int recvlen = 0;
    nl_Write(hDevice, strCmd, strlen(strCmd), true);
    int ret = nl_Read(hDevice, receivedData, 1000, 100);
    printf("system info: \n%s\n", 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
    char receivedData[1024] = { 0 };
    int nRet = 0;
    bool isWrited = nl_WriteAsHex(hDevice, strCmdhEX, true); // Write data
    nRet = nl_Read(hDevice, receivedData, sizeof(receivedData), 0); // Read data
    printf("nRet=%d, receivedData=%s\n", nRet, receivedData);
}
else if (argc >= 2 && strcmp(argv[1], "--GetDeviceInfo") == 0) // Write data to the device in HEX character string
{
    STDeviceInfo info;
    memset(&info, 0, sizeof(STDeviceInfo));
    nl_GetDeviceInfo(hDeviceList, i, &info);
    printf("GetDeviceInfo -------- info\n %s\ntype=%d\n", info.devInfo, info.devType);
}
else if (argc >= 2 && strcmp(argv[1], "--SendCommand") == 0) // Send control commands to the device and obtain the returned information
{
    char strCmd[2048] = { 0 };
    strcpy(strCmd, "QRYSYS");
    DbgPrintf("begin to nl_SendCommand QRYSYS\n");
    int result = nl_SendCommand(hDevice, strCmd, strlen(strCmd)); // Send commands
```

```
            DbgPrintf("nl_SendCommand QRYSYS end\n");
            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 = nl_SendCommandAsHex(hDevice, strCmd, strlen(strCmd)); // Send commands
            printf("result=%d\n", result);
        }
        else if (argc >= 2 && strcmp(argv[1], "--GetCommandResponse") == 0) {
            char strCmd[2048] = { 0 };
            strcpy(strCmd, "QRYSYS");
            char recvData[2048] = { 0 };
            int recvLen = 0;
           bool result = nl_GetCommandResponse(hDevice, strCmd, strlen(strCmd), recvData, &recvLen, 500, true, false);
            printf("result=%d\n", result);
            printf("recvData=%s\n", recvData);
        }
        else if (argc >= 2 && strcmp(argv[1], "--GetPicture") == 0) // Get device image
        {
            unsigned int imgWidth = 0, imgHeight = 0;
            char filename[1024] = { 0 };
            sprintf(filename, "1%d.bmp", i);
            bool isGetPicSizeOK = nl_GetPicSize(hDevice, &imgWidth, &imgHeight); // Get the image width and height
            printf("nl_GetPicSize isGetPicSizeOK=%d\n", isGetPicSizeOK);
            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);
                bool isOK = nl_GetPicData(hDevice, recvBuffer, RECV_BUFFER_SIZE); // Get the image raw data
                if(isOK)
                    nl_SavePicDataToFile(filename, recvBuffer, imgWidth, imgHeight, 8);
            }
        }
        else if (argc >= 2 && strcmp(argv[1], "--GetPictureByConfig") == 0) // Get device image
        {
            unsigned int imgWidth = 0, imgHeight = 0;
            bool isGetPicSizeOK = nl_GetPicSize(hDevice, &imgWidth, &imgHeight); // Get the
```

```
image width and height
                printf("nl_GetPicSize isGetPicSizeOK=%d\n", isGetPicSizeOK);
                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 = 2;
                    imgParam.q = 3;
                    STImgResolution imgR[4];
                    memset(imgR, 0, sizeof(STImgResolution) * 4);
                    unsigned int nRealLen = 0;
                    bool isOK = nl_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]);

                    char filename[1024] = { 0 };
                    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) {
                            sprintf(filename, "test3%d.bmp", i);
                            FILE* fp = fopen(filename, "wb");
                            fwrite(recvBuffer, 1, nRealLen, fp);
                            fclose(fp);
                        }
                        else if (imgParam.f == 2) {
                            sprintf(filename, "test4%d.jpg", i);
                            FILE* fp = fopen(filename, "wb");
                            fwrite(recvBuffer, 1, nRealLen, fp);
                            fclose(fp);
                        }
                        else if (imgParam.f == 3) {
                            sprintf(filename, "test5%d.tiff", i);
                            FILE* fp = fopen(filename, "wb");
                            fwrite(recvBuffer, 1, nRealLen, fp);
                            fclose(fp);
```

```
                        }
                        else if (imgParam.f == 4) {
                            sprintf(filename, "test6%d.bmp", i);
                            FILE* fp = fopen(filename, "wb");
                            fwrite(recvBuffer, 1, nRealLen, fp);
                            fclose(fp);
                        }
                        else if (imgParam.f == 0) {
                            long outLen = 0;
                            STImgResolution imgResIn, imgResOut;
                            imgResIn.width = imgWidth;
                            imgResIn.height = imgHeight;
                            unsigned int imgLen = 0;
                            IMG_TYPE type = nl_GetDeviceImageColorType(hDevice,
&imgResOut, &imgLen);
                            if (type == TYPE_COLOR) {
                                unsigned char* outBuf = (unsigned char*)malloc(imgLen);
                                bool res = nl_ConvertImageColorSpace(hDevice, recvBuffer,
RECV_BUFFER_SIZE, imgResIn, outBuf);
                                sprintf(filename, "test2%d.bmp", i);
                                nl_SavePicDataToFile(filename, outBuf, imgResOut.width,
imgResOut.height, 24); // Save image
                                sprintf(filename, "test2%d.jpg", i);
                                nl_SavePicDataToFile(filename, outBuf, imgResOut.width,
imgResOut.height, 23); // Save image
                            }
                            else {
                                sprintf(filename, "test1%d.bmp", i);
                                nl_SavePicDataToFile(filename, recvBuffer, imgResOut.width,
imgResOut.height, 8); // Save image
                                sprintf(filename, "test1%d.jpg", i);
                                nl_SavePicDataToFile(filename, recvBuffer, imgResOut.width,
imgResOut.height, 13); // Save image
                            }
                        }
                    }
                    free(recvBuffer);
                    recvBuffer = NULL;
                }
            }
            else if (argc >= 2 && strcmp(argv[1], "--SetListener") == 0) // Asynchronous reading of
device data
            {
                char receivedData[2048] = { 0 };
```

```
            nl_SetListener(hDevice, ReadCallback);
            Sleep(1000);
            for (int i = 0; i < 100; i++)
            {
                Sleep(7000);
            }
            nl_StopListener(hDevice);
        }
        else if (argc >= 3 && strcmp(argv[1], "--ReadDevCfgToXml") == 0) // Read the configuration from the device and save it to the xml file.
        {
            nl_ReadDevCfgToXml(hDevice, argv[2]);
        }
        else if (argc >= 3 && strcmp(argv[1], "--WriteCfgToDev") == 0) // Write the configuration file information to the device.
        {
            nl_WriteCfgToDev(hDevice, argv[2]);
        }
        else if (argc >= 2 && strcmp(argv[1], "--SetCbDevStatusChanged") == 0) // Set the callback function when the device status changes.暂时不能使用
        {
            char receivedData[2048] = { 0 };
            nl_SetCbDevStatusChanged(hDevice, DevStatChangeCallback);
            Sleep(60000);
            printf("SetCbDevStatusChanged finish\n");
        }
        else if (argc >= 2 && strcmp(argv[1], "--UpdateFirmware") == 0) // Update device
        {
            unsigned updateError = -1;
            //bool isUpdated = nl_UpdateKernelDevice(hDevice, "Y:\\Newland\\scan\\firmware\\Krnl_MONGO_V2.UJ101.ST.G05.2.bin3", 0, &updateError); // Firmware update
            bool isUpdated = nl_UpdateKernelDevice(hDevice, "Y:\\Newland\\scan\\firmware\\GALE\\GALE.UQ101.ST.H02.5.bin2", 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;
            }
        }
        else if (argc >= 2 && strcmp(argv[1], "--SetNetDeviceConfig") == 0) {
            char configData[2048] = { 0 };
            strcpy(configData, "Serial Number=N5BC00202NOM;MAC Address=E0:5A:9F:8E:D1:33;Use DHCP=0;IP Address=192.168.3.174;SubNetmask=255.255.255.0;Gateway Address=0.0.0.0;");
            char outData[2048] = { 0 };
            int nRet = nl_SetNetDeviceConfig(configData, strlen(configData), 5000, outData);
            if (nRet != 0)
            {
                printf("nl_setNetDeviceConfig error\n");
            }
            printf("\n nl_setNetDeviceConfig outData=%s\n", outData);
        }

        bool isClosed = nl_CloseDevice(&hDevice); // Close the device
        printf("hDevice=%p,%s\n", hDevice, isClosed ? "succeed in closing the device" : "failed to close the device");
    }
    nl_ReleaseDevices(&hDeviceList); // Release the device list handle
    printf("handleDeviceList=%p\n", hDeviceList);

    system("pause");
    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**
HANDLEDEVLST nl_EnumDevices(int* deviceCount, EnumType = ENUM_ALL);	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
void	brief 释放设备列表句柄.

nl_ReleaseDevices(HANDLEDEVLST* hDeviceList);	param[in] hDeviceList 设备列表句柄
HANDLEDEV nl_OpenDevice(const HANDLEDEVLST hDeviceList, unsigned int index, T_Porotocol porotocol = Nlscan);	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
bool nl_Write(const HANDLEDEV hDevice, const char* data, unsigned int len, bool isPacked = true);	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
bool nl_WriteAsHex(const HANDLEDEV hDevice, const char* data, bool isPacked = false);	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
T_CommunicationResult nl_SendCommand(const HANDLEDEV hDevice, const char* command, unsigned int commandLen);	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 true，表示更新成功，返回 false，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 true，表示关闭成功，返回 false，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 bmp 位图时,表示图像位深度,取值:8 或 24 保存文件为 jpg 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 true，表示保存成功，返回 false，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 true，表示保存成功，返回 false，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 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
void nl_SetCbDevStatusChanged(const HANDLEDEV hDevice, DevStatChgCallback callback);	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) param[in] hDevice 设备句柄 param[in] isDevExisted 设备是否存在
bool nl_GetCommandResponse(const HANDLEDEV hDevice, const char* command, unsigned int commandLen, char* response, int *responseLen, unsigned int timeout, bool isPacked, bool isHex);	brief 发送指令并接收返回指令 param[in] hDevice 设备句柄 param[in] command 发送指令数据 param[in] commandLen 发送指令长度 param[out]response 接收指令数据,需要足够大的空间进行接收 param[out]responseLen 接收指令数据的真实长度 param[in]timeout 超时时间

	param[in]isPacked 是否需要打包 param[in]isHex 是否发送 HEX 指令数据 return 是否收发成功，返回 true，表示成功，返回 false，表示失败
bool nl_GetPicDataByConfig(const HANDLEDEV hDevice, STImgParam imgParam, unsigned char* imgBuf, unsigned int *imgBufLen, STImgResolution* imgR);	brief 根据参数获取图像数据 param[in] hDevice 设备句柄 param[in] imgParam 图像参数结构 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 F，图像格式：0F - 原始图像（Raw data), 1F - BMP 格式， 2F - JPEG 格式 Q，JPEG 格式的图像质量：0Q - Low， 1Q - Middle，2Q - High， 3Q – Highest 其他参数暂时保留,初始化为 0 param[out]imgBuf 返回的图像数据,需要足够大的空间进行接收 param[out]imgBufLen 返回的图像数据真实长度 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 return 是否接收成功，返回 true，表示成功 false，表示失败
IMG_TYPE nl_GetDeviceImageColorType(const HANDLEDEV hDevice, STImgResolution* imgResOut, unsigned int * imgLen);	brief 获取设备原始图像的图片类型 param[in] hDevice 设备句柄 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 param[out] imgLen 图像数据的真实大小 return 原始图像类型
bool nl_ConvertImageColorSpace(const HANDLEDEV hDevice, unsigned char* imgBufIn, long imgBufInLen, STImgResolution imgResIn, unsigned char* imgBufOut);	brief 原始图像色彩空间转换 nv12->bgr param[in] hDevice 设备句柄 param[in] imgBufIn 原始图像信息 param[in] imgBufInLen 原始图像数据长度 param[in] imgResIn 原始图像的分辨率 param[out] imgBufOut 转换后的图像数据 return 是否接收成功，返回 true，表示成功 false，表示失败
bool nl_GetDeviceInfo(const HANDLEDEVLST hDeviceList, unsigned int index, STDeviceInfo* stNetDevInfo);	brief 获取设备信息 param[in] hDeviceList 设备句柄列表 param[in] index 索引 param[out] stNetDevInfo 设备信息结构

	return 是否获取成功，返回 true，表示成功 false，表示失败
bool nl_DeviceIsOpenByHandle(const HANDLEDEV hDevice);	brief 设备是否打开 param[in] hDevice 设备句柄 return 是否打开，返回 true，表示打开 false，表示关闭
bool nl_DeviceIsOpenByList(const HANDLEDEVLST hDeviceList, unsigned int index);	brief 设备是否打开 param[in] hDeviceList 设备句柄列表 param[in] index 索引 return 是否打开，返回 true，表示打开 false，表示关闭
char *nl_GetLastError();	brief 获取最后一次操作的错误信息 return 错误信息
void nl_BeginEnumNetDevice();	brief 异步搜索网络设备 return
void nl_StopEnumNetDevice();	brief 停止异步搜索网络设备 return
int nl_SetNetDeviceConfig(char* inData,int inDataLen,int recTimeout,char* outdata);	brief 设置网络设备配置信息 param[in] inData 配置信息 param[in] inDataLen 配置信息的长度 param[in] recTimeout 超时时间 param[in] outdata 返回信息 return 0 成功 其他 失败
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* serviceIp, int port, int* socket);	brief 连接网络服务端 param[in] serviceIp 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[out] realLen 图像数据长度 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.
    NotClosed,              ///< 未关闭.
    Updating,               ///< 升级中.
    Updated,                ///< 升级结束.
    Writing,                ///< 写数据中.
    Written,                ///< 写数据结束.
    Reading,                ///< 读数据中.
    ReadOK,                 ///< 读数据结束.
    GettingPicData,         ///< 读图像数据中.
    GetPicDataOK,           ///< 读图像数据结束.
    GettingPicColorType,    ///< 获取图像色彩类型.
    GetPicColorTypeOk,      ///< 获取图像色彩类型结束.
    ConveringColorSpace,    ///< 图像色彩转换.
    ConvertColorSpaceOK,    ///< 图像色彩转换结束.
    UnknownStatus           ///< 未知状态.
};
```

```
brief 指令发送结果.
enum T_CommunicationResult
{
    SendError = 0,      ///< 发送错误.
```

```
    Support,            ///< 支持指令.
    Unsupport,          ///< 不支持指令.
    OutOfRange,         ///< 数据的值不在支持范围.
    UnknownResult,      ///< 未知错误.
};
```

```
brief 厂家协议.
enum T_Porotocol
{
    Nlscan = 0, // 新大陆.
};
```

```
Brief 图像色彩类型
enum IMG_TYPE {
    TYPE_UNKNOW = 0,//未知类型
    TYPE_GRAY = 1,   //黑白
    TYPE_COLOR = 2   //彩色
};
```

```
Brief 设备类型.
enum NL_DEVICE_TYPE {
    DEV_TYPE_UNKNOW = 0,//未知
    DEV_TYPE_USB = 1,//usb 设备
    DEV_TYPE_COM = 2,//串口设备
    DEV_TYPE_NET = 3,//网络设备
};
```