

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-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



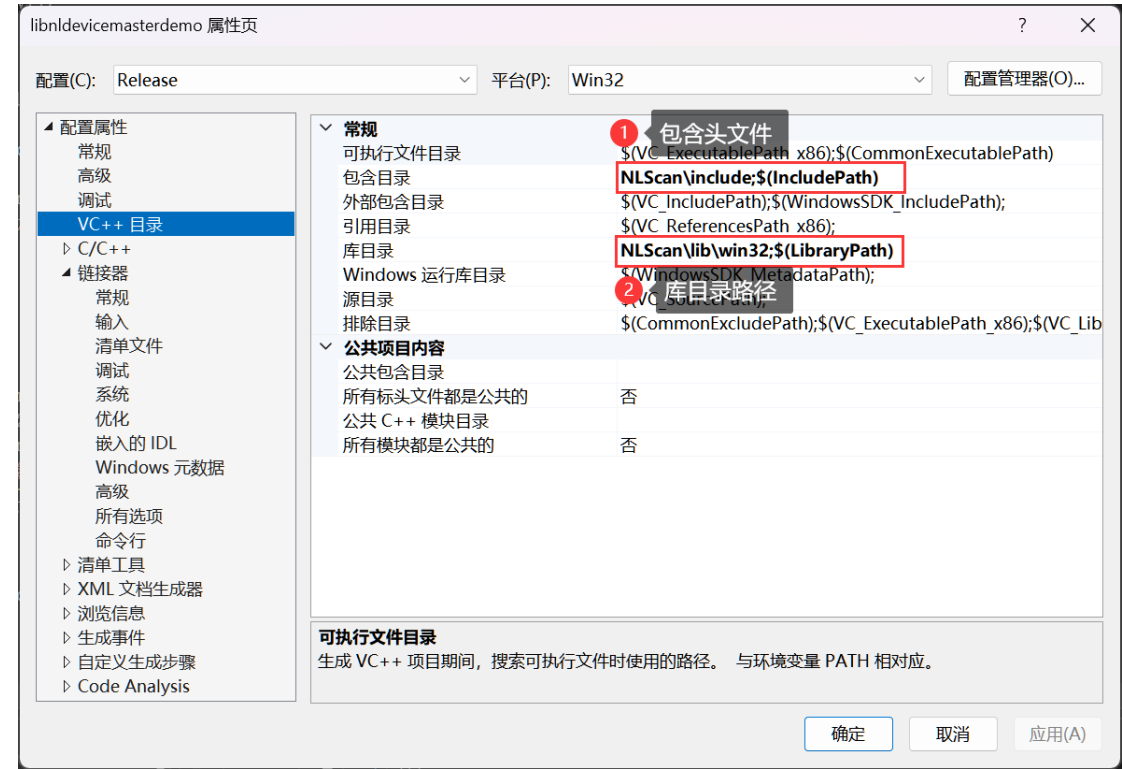
开发工具：VS2019



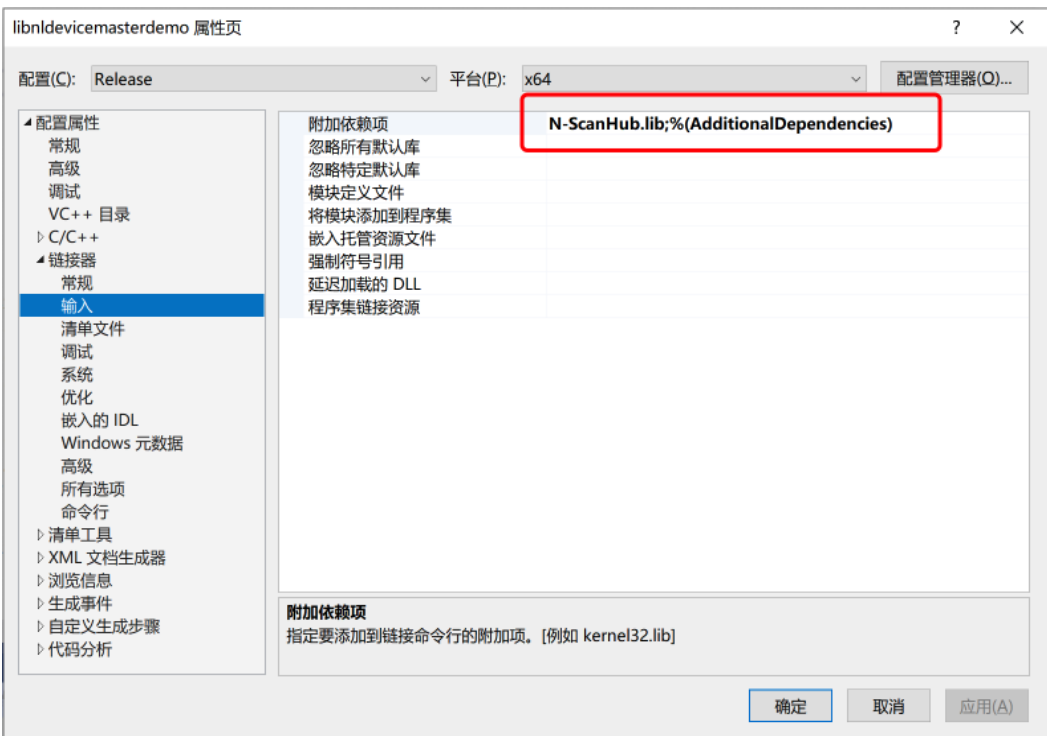
操作系统：Windows 10

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

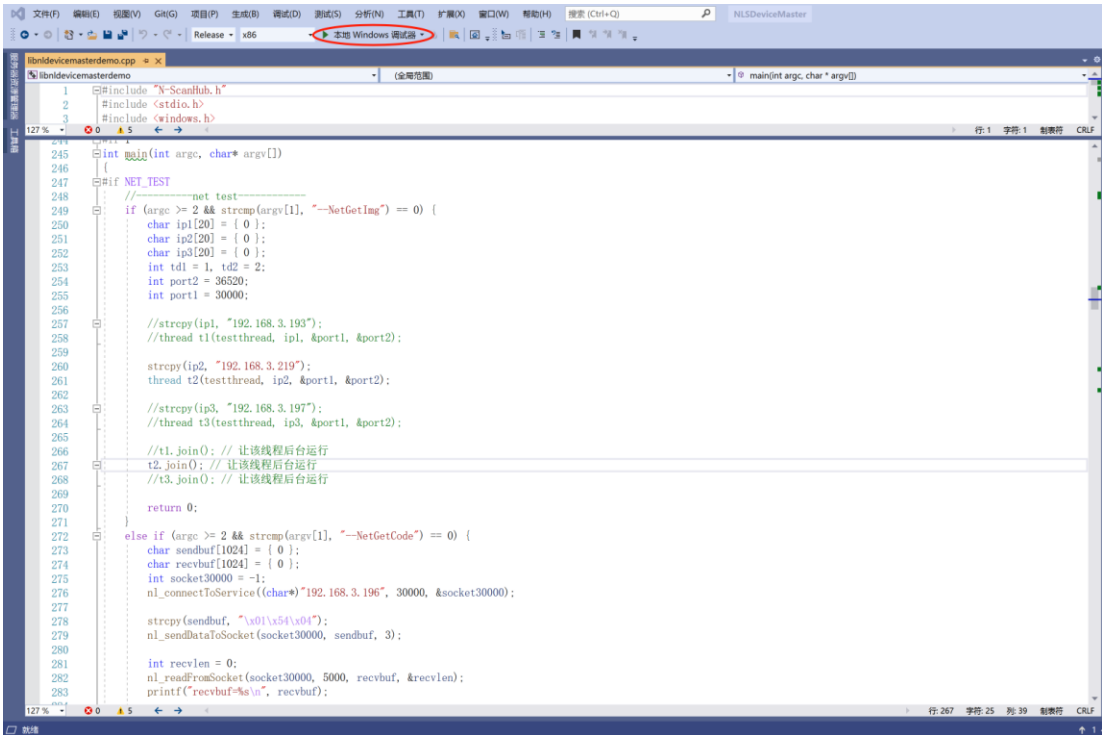


2.工程中添加库依赖 N-ScanHub.lib



3.开始调用 SDK 中的函数

4.开始运行：运行程序



效果如下：

```
C:\> Microsoft Visual Studio 调试控制台

T_DevicesInfo=1052
deviceCounts=1
succeed in opening the device
nRet=193, receivedData=@QRYSYSProduct Name: GALE
Firmware Version: UQ101.ST.G02.5
Decoder Version: 7.1.17
Hardware Version:
Serial Number: 1686722549.5771632
OEM Serial Number:
Manufacturing Date:
```

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
<pre>void nl_ReleaseDevices(HANDLEDEVLST* hDeviceList);</pre>	<p>brief 释放设备列表句柄.</p> <p>param[in] hDeviceList 设备列表句柄</p>
<pre>HANDLEDEV nl_OpenDevice(const HANDLEDEVLST hDeviceList, unsigned int index, T_Porotocol porotocol = Nlscan);</pre>	<p>brief 打开设备列表指定索引的设备.</p> <p>param[in] hDeviceList 设备列表句柄</p> <p>param[in] index 索引</p> <p>param[in] porotocol 厂家协议</p> <p>return 设备句柄，返回非 NULL，表示打开成功。 返回 NULL，表示打开失败</p>
<pre>bool nl_Write(const HANDLEDEV hDevice, const char* data, unsigned int len, bool isPacked = true);</pre>	<p>brief 向设备写数据.</p> <p>param[in] hDevice 设备句柄</p> <p>param[in] data 数据</p> <p>param[in] len 数据长度</p> <p>param[in] isPacked 数据是否打包</p> <p>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</p>
<pre>bool nl_WriteAsHex(const HANDLEDEV hDevice, const char* data, bool isPacked = false);</pre>	<p>brief 以 HEX 字符串方式向设备写数据.</p> <p>param[in] hDevice 设备句柄</p> <p>param[in] data 数据</p> <p>param[in] isPacked 数据是否打包</p> <p>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</p>
<pre>T_CommunicationResult nl_SendCommand(const HANDLEDEV hDevice, const char* command, unsigned int commandLen);</pre>	<p>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).</p> <p>param[in] hDevice 设备句柄</p> <p>param[in] command 指令</p> <p>param[in] commandLen 指令长度</p> <p>return 通讯结果</p>
<pre>T_CommunicationResult nl_SendCommandAsHex(const HANDLEDEV hDevice, const char* command, unsigned int commandLen);</pre>	<p>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).</p> <p>param[in] hDevice 设备句柄</p> <p>param[in] command 指令</p> <p>param[in] commandLen 指令长度</p> <p>return 通讯结果</p>
<pre>unsigned int nl_Read(const HANDLEDEV hDevice, char* buf, unsigned int len, unsigned int timeout);</pre>	<p>brief 读设备数据.</p> <p>param[in] hDevice 设备句柄</p> <p>param[out] buf 设备返回的数据</p> <p>param[in] len 接收的长度</p>

	<p>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</p> <p>return 设备返回的数据长度</p>
<pre>void nl_SetListenerParam(const HANDLEDEV hDevice, void *userParam);</pre>	<p>brief 设置监听用户参数</p> <p>param[in] hDevice 设备句柄</p> <p>param[in] userParam 用户参数指针</p>
<pre>void nl_SetListener(const HANDLEDEV hDevice, readCallback callback);</pre>	<p>brief 设置监听.</p> <p>param[in] hDevice 设备句柄</p> <p>param[in] callback 回调函数</p>
<pre>bool nl_StopListener(const HANDLEDEV hDevice);</pre>	<p>brief 停止监听设备数据.</p> <p>param[in] hDevice 设备句柄</p> <p>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</p>
<pre>bool nl_GetPicSize(const HANDLEDEV hDevice, unsigned int* width, unsigned int* height);</pre>	<p>brief 获取设备图像尺寸.</p> <p>param[in] hDevice 设备句柄</p> <p>param[out] width 图片的宽</p> <p>param[out] height 图片的高</p> <p>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</p>
<pre>bool nl_GetPicData(const HANDLEDEV hDevice, unsigned char* imgBuf, int imgBufLen);</pre>	<p>brief 获取设备图像.</p> <p>param[in] hDevice 设备句柄</p> <p>param[out] imgBuf 图像的数据</p> <p>param[in] imgBufLen 图像的数据长度</p> <p>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</p>
<pre>bool nl_UpdateKernelDevice(const HANDLEDEV hDevice, const char* strFileName, unsigned int reserved = 0, unsigned int* error = 0);</pre>	<p>brief 更新设备</p> <p>（固件升级成后需要重新枚举设备，才能继续使用设备句柄。</p> <p>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）.</p> <p>param[in] hDevice 设备句柄</p> <p>param[in] strFileName 固件文件路径</p> <p>param[in] reserved 保留字段</p> <p>param[out] error 更新失败后返回的失败编号</p> <p>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</p>
<pre>bool nl_CloseDevice(HANDLEDEV* hDevice);</pre>	<p>brief 关闭设备.</p> <p>param[in] hDevice 设备句柄</p> <p>return 是否关闭成功，返回 true，表示关闭成功，</p>

	返回 false ，表示关闭失败
<pre>bool nl_SavePicDataToFile(const char* bmpName, unsigned char* imgBuf, int width, int height, int flag);</pre>	<p>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)</p> <p>param[in] bmpName bmp 文件名</p> <p>param[in] imgBuf 图像缓冲数据</p> <p>param[in] width 图像宽</p> <p>param[in] height 图像高</p> <p>param[in] flag 图像参数标识</p> <p>保存文件为 bmp 位图时,表示图像位深度,取值:8 或 24</p> <p>保存文件为 jpg 时,表示图像质量高低</p> <p>黑白图片: (10-Low, 11-Middle, 12-High, 13-Highest)</p> <p>彩色图片: (20-Low, 21-Middle, 22-High, 23-Highest)</p> <p>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</p>
<pre>T_DeviceStatus nl_GetDevStatus(const HANDLEDEV hDevice);</pre>	<p>brief 获取设备当前状态.</p> <p>param[in] hDevice 设备句柄</p> <p>return 设备状态</p>
<pre>bool nl_ReadDevCfgToXml(const HANDLEDEV hDevice, const char* cfgFilePath);</pre>	<p>brief 从设备读取配置, 并保存到 xml 文件.</p> <p>param[in] hDevice 设备句柄</p> <p>param[in] cfgFilePath 配置文件路径</p> <p>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</p>
<pre>bool nl_WriteCfgToDev(const HANDLEDEV hDevice, const char* cfgFilePath);</pre>	<p>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).</p> <p>param[in] hDevice 设备句柄</p> <p>param[in] cfgFilePath 配置文件路径</p> <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, DevStatChgCallback 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,</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>

<pre>char* response, int *responseLen, unsigned int timeout, bool isPacked, bool isHex);</pre>	<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>R, 图像比率: 0R - 原始大小, 1R - 1/4 图像, 2R - 1/16 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</p> <p>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>bool nl_GetPicDataAndCodeInfo(const HANDLEDEV hDevice, STImgAndInfoParam imgParam, unsigned char* imgBuf, unsigned int* imgBufLen, STCodeInfoList* pCodeInfoList);</pre>	<p>brief 根据参数获取解码信息以及解码成功图像数据</p> <p>param[in] hDevice 设备句柄</p> <p>param[in] imgParam 图像参数结构</p> <p>R, 图像比率: 0R - 原始大小, 1R - 1/4 图像, 2R - 1/16 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,</p>

	<p>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] pCodeInfoList 解码信息结构列表,最多可存储同时解析的 64 个码信息</p> <p>return 是否接收成功, 返回 true, 表示成功 false, 表示失败</p>
<p>IMG_TYPE</p> <p>nl_GetDeviceImageColorType(const HANDLEDEV hDevice, STImgResolution* imgResOut, unsigned int * imgLen);</p>	<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>
<p>bool nl_ConvertImageColorSpace(const HANDLEDEV hDevice, unsigned char* imgBufIn, long imgBufInLen, STImgResolution imgResIn, unsigned char* imgBufOut);</p>	<p>brief 原始图像色彩空间转换 nv12->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>
<p>bool nl_GetDeviceInfo(const HANDLEDEVLST hDeviceList, unsigned int index, STDeviceInfo* stNetDevInfo);</p>	<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>
<p>bool nl_DevicelsOpenByHandle(const HANDLEDEV hDevice);</p>	<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>
<p>bool nl_DevicelsOpenByList(const HANDLEDEVLST hDeviceList, unsigned int index);</p>	<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>

char *nl_GetLastError();	brief 获取最后一次操作的错误信息 return 错误信息
bool nl_SetNetConfig(char *sn, char *mac, bool dhcp, char *ip, char *subNetMask, char *gateway, int recTimeout);	brief 设置网络设备配置信息 param[in] sn 设备序列号 param[in] mac 设备 mac 地址 param[in] dhcp 动态主机设置协议 param[in] ip ip 地址 param[in] subNetMask 子网掩码 param[in] gateway 网关 param[in] recTimeout 超时时间 return true 成功 false 失败
bool nl_GetDeviceInfoByHandle(const HANDLEDEV hDevice, STDeviceInfo* stNetDevInfo);	brief 获取设备信息 param[in] hDevice 设备句柄 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_SendBmpDecode(const HANDLEDEV hDevice, const IMG_TYPE imgType, char* bmpFileName, char* retDecode, int* retLen, int* decodeTime, int timeout);	brief 下载图片解码 param[in] hDevice 设备句柄 param[in] imgType 图片类型 param[in] bmpFileName 图片完整路径 param[out] retDecode 码池信息 param[out] retLen 码池长度 param[out] decodeTime 解码时间 param[in] timeout 超时时间 return 是否获取成功, 返回 true, 表示成功 false, 表示失败 注:必须是设备获取的图片
bool nl_GetRoiPicData(const HANDLEDEV hDevice, STRoilmgParam imgParam, STFrameData* frameData, unsigned char* imgBuf, unsigned int* imgBufLen);	brief 获取 ROI 图片 param[in] hDevice 设备句柄 param[in] imgParam 图片参数结构 p-业务方案序号 u-业务单元序号 i-ROI 序号 (99:无序号(例如上一节点裁剪的图): ROI 序号从 1 开始) t-类型: 0T - 输入图像(最后一次拍摄的图像),1T - 结果图像 param[out] frameData Roi 图片帧数据 param[out] imgBuf 图片数据 param[in/out] imgBufLen [in]调用者分配的 imgBuf 空间大小 [out]图片数据真实大小 return 是否获取成功, 返回 true, 表示成功

	false, 表示失败
void nl_LogSwitch(bool isLogging);	brief 日志开关 param[in] isLogging true-记录日志 false-关闭日志
void nl_GetVersion(char* version);	brief 获取 sdk 版本号 param[out] version 版本号
HANDLEDEVLST nl_AddNetworkDevice(char *ipList);	brief 根据 IP 地址和端口添加网络设备 param[in] ipList ip 地址列表, 例: "192.168.1.100:36520,192.168.1.101:36520,192.168.1.102:36520" 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

以下为网络独立接口,在确定设备 IP 地址时使用

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,	brief 通过网络获取图像数据

<pre>int R, int F, int Q, char *imgData, int *realLen, IMG_TYPE* imgtype, int *width, int *heigh);</pre>	<p>param[in] socket 网络套接字</p> <p>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</p> <p>param[in] R 图像比率,暂时保留,初始化为 0</p> <p>param[in] F 图像格式, 0F - 原始图像 (Raw data), 1F - BMP 格式, 2F - JPEG 格式</p> <p>param[in] Q jpg 图像质量, 0Q - Low, 1Q - Middle, 2Q - High, 3Q - Highest</p> <p>param[out] imgData 图像数据</p> <p>param[in/out] realLen</p> <p>[in] imgData 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] imgtype 图像类型</p> <p>param[out] width 分辨率宽</p> <p>param[out] heigh 分辨率高</p> <p>return 0 成功 其他 失败</p>
<pre>Int nl_GetRoiPicDataByNet(int socket, STRoilmgParam imgParam, STFrameData* frameData, unsigned char* imgBuf, unsigned int* imgBufLen);</pre>	<p>brief 获取 ROI 图片</p> <p>param[in] socket 网络套接字</p> <p>param[in] imgParam 图片参数结构</p> <p> p-SOP 序号</p> <p> u-业务单元序号</p> <p> i-ROI 序号,从 1 开始(99:无序号(例如上一节点裁剪的图);)</p> <p> t-类型: 0T - 输入图像(最后一次拍摄的图像),1T - 结果图像</p> <p>param[out] frameData Roi 图片帧数据</p> <p>param[out] imgBuf 图片数据</p> <p>param[in/out] imgBufLen</p> <p>[in]调用者分配的 imgBuf 空间大小</p> <p>[out]图片数据真实大小</p> <p>return 是否获取成功, 返回 0 表示成功,其他表示失败</p>
<pre>T_CommunicationResult nl_SendCommandFromSocket(int socket, const char* command, unsigned int commandLen);</pre>	<p>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).</p> <p>param[in] socket 网络套接字</p> <p>param[in] command 指令</p> <p>param[in] commandLen 指令长度</p> <p>return 通讯结果</p>
<pre>bool nl_GetCommandResponseFromSocket(i nt socket, const char* command, unsigned int commandLen, char*</pre>	<p>brief 发送指令并接收返回指令</p> <p>param[in] socket 网络套接字</p> <p>param[in] command 发送指令数据</p> <p>param[in] commandLen 发送指令长度</p>

<pre>response, int* responseLen, unsigned int timeout, bool isPacked = true, bool isHex = false);</pre>	<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 [in]response 分配的空间长度 [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>
---	--

以下为物理串口独立接口, 在确定设备的物理串口号时打开和关闭物理串口, 不使用 nl_EnumDeivces 枚举物理设备

<pre>HANDLEDEV nl_OpenSerialPortDevice(char* serial);</pre>	<p>brief 通过串口号打开设备</p> <p>param[in] serial 串口号,例:“COM1”</p> <p>return 返回设备句柄,为 NULL 为打开失败</p>
<pre>bool nl_CloseSerialPortDevice(char* serial, HANDLEDEV* hDevice);</pre>	<p>brief 通过串口号和对应设备句柄关闭设备</p> <p>param[in] serial 串口号,例:“COM1”</p> <p>param[in] hDevice 设备句柄</p> <p>return 是否关闭成功, 返回 true, 表示成功, 返回 false, 表示失败</p>

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
};
```

Brief 网络参数设置类型.

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