

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:

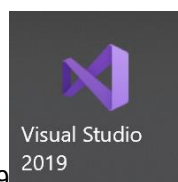
目录	说明
include	头文件: N-ScanHub.h 内含所有接口说明
lib/x64	64 位 N-ScanHub.dll、N-ScanHub.lib
lib/x86	32 位 N-ScanHub.dll、N-ScanHub.lib
demo	库文件和 Visual Studio2019 编写的 demo
help	帮助文档: N-ScanHub.pdf

2.2 使用教程

测试设备: FM430



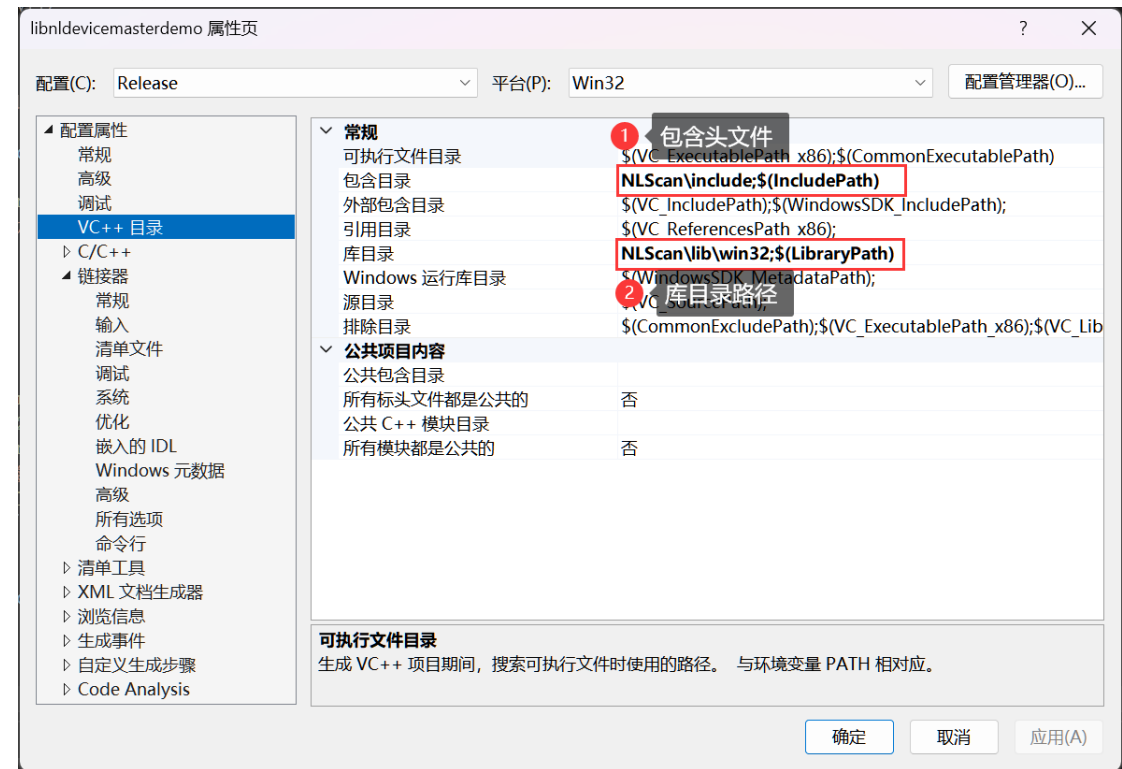
开发工具: VS2019



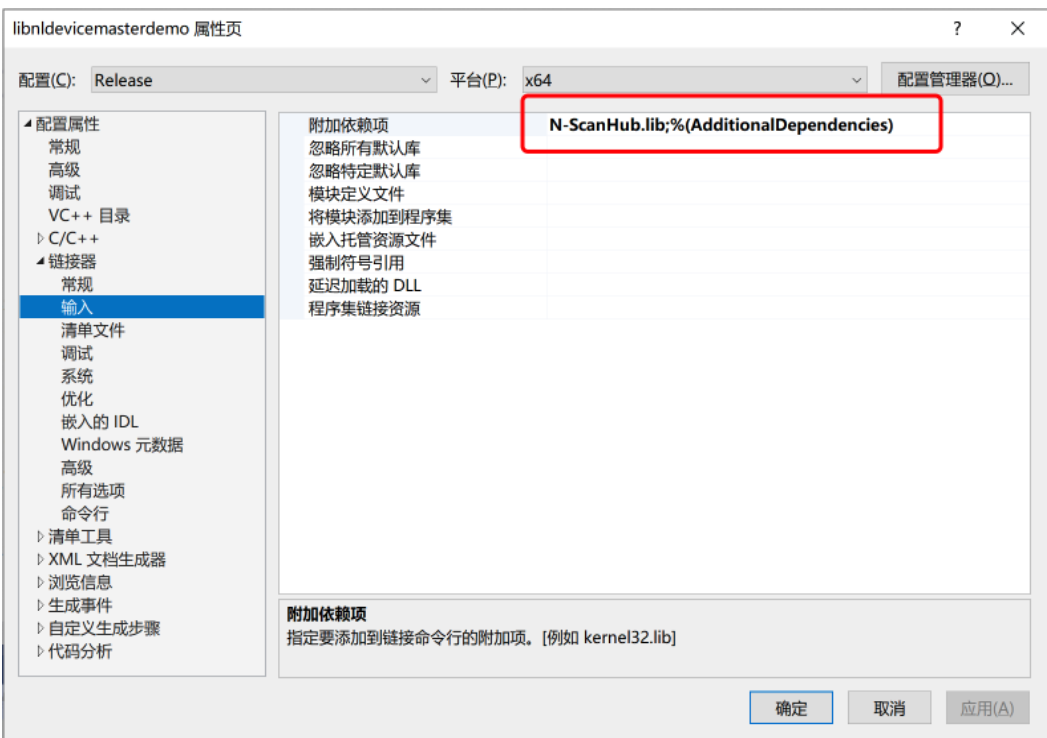
操作系统: Windows 10

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

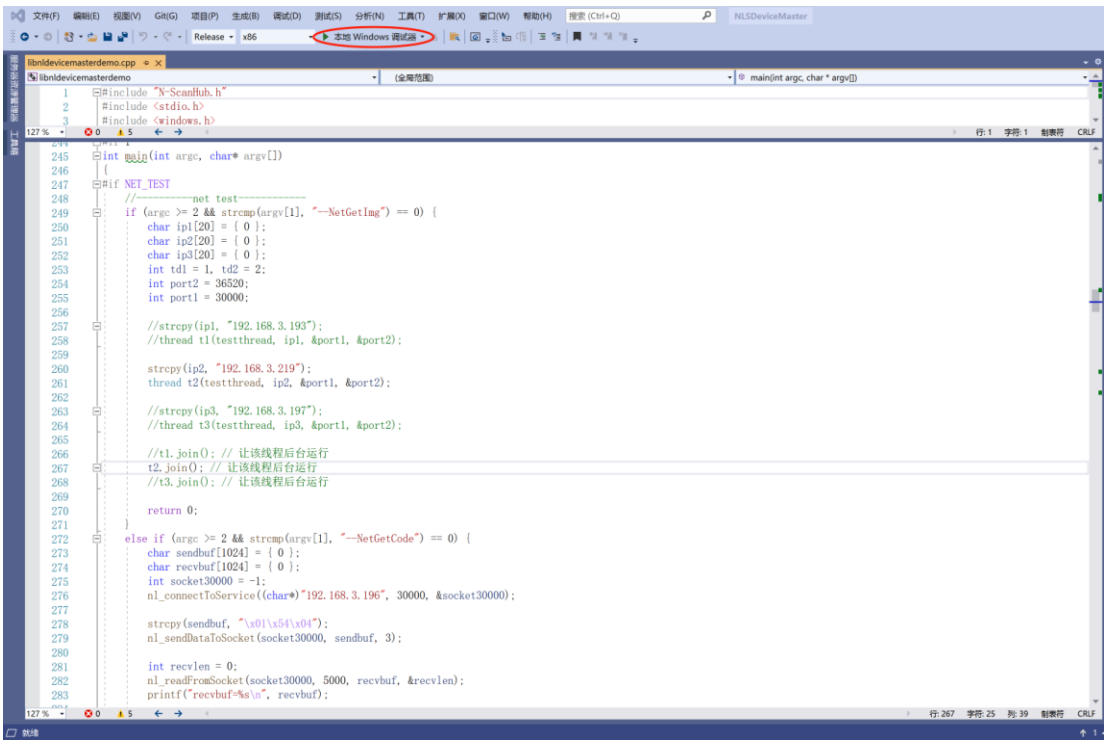


2. 工程中添加库依赖 N-ScanHub.lib



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>	brief 释放设备列表句柄. param[in] hDeviceList 设备列表句柄
<pre>HANDLEDEV nl_OpenDevice(const HANDLEDEVLST hDeviceList, unsigned int index, T_Porotocol porotocol = Nlscan);</pre>	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
<pre>bool nl_Write(const HANDLEDEV hDevice, const char* data, unsigned int len, bool isPacked = true);</pre>	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
<pre>bool nl_WriteAsHex(const HANDLEDEV hDevice, const char* data, bool isPacked = false);</pre>	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
<pre>T_CommunicationResult nl_SendCommand(const HANDLEDEV hDevice, const char* command, unsigned int commandLen);</pre>	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
<pre>T_CommunicationResult nl_SendCommandAsHex(const HANDLEDEV hDevice, const char* command, unsigned int commandLen);</pre>	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
<pre>unsigned int nl_Read(const HANDLEDEV hDevice, char* buf, unsigned int len, unsigned int timeout);</pre>	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
<pre>void nl_SetListenerParam(const HANDLEDEV hDevice, void *userParam);</pre>	brief 设置监听用户参数 param[in] hDevice 设备句柄 param[in] userParam 用户参数指针
<pre>void nl_SetListener(const HANDLEDEV hDevice, readCallback callback);</pre>	brief 设置监听. param[in] hDevice 设备句柄 param[in] callback 回调函数
<pre>bool nl_StopListener(const HANDLEDEV hDevice);</pre>	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
<pre>bool nl_GetPicSize(const HANDLEDEV hDevice, unsigned int* width, unsigned int* height);</pre>	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
<pre>bool nl_UpdateKernelDevice(const HANDLEDEVLST hDeviceList, unsigned int index, const char* strFileName);</pre>	brief 更新设备 param[in] hDeviceList 设备列表句柄 param[in] index 索引 param[in] strFileName 固件文件路径 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
<pre>bool nl_CloseDevice(HANDLEDEV* hDevice);</pre>	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
<pre>bool nl_SavePicDataToFile(const char* bmpName, unsigned char* imgBuf, int width, int height, int flag);</pre>	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 黑白图片: (10-Low, 11-Middle, 12-High, 13-Highest) 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
<pre>bool nl_ReadDevCfgToXml(const HANDLEDEV hDevice, const char* cfgFilePath);</pre>	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
<pre>bool nl_WriteCfgToDev(const HANDLEDEV hDevice, const char* cfgFilePath);</pre>	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
<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>	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[in/out]responseLen [in]response 分配的空间长度 [out]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
<pre>bool nl_GetPicData (const HANDLEDEV hDevice, STImgParam imgParam, unsigned char* imgBuf, unsigned int *imgBufLen);</pre>	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 type, 类型: 0 - 实时图像 (最后一次拍摄的图像), 1 - 解码成功的图像 format, 图像格式: 0 - 原始图像 (Raw data), 1 - BMP 格式, 2 - JPEG 格式 quality, JPEG 格式的图像质量: 0 - Low, 1 - Middle, 2 - High, 3 - Highest param[out]imgBuf 返回的图像数据,需要足够大的

	空间进行接收 param[in/out] imgBufLen [in] imgBuf 分配的空间长度 [out] 返回的图像数据真实长度 return 是否接收成功, 返回 true, 表示成功 false, 表示失败
<pre>bool nl_GetPicDataAndCodeInfo(const HANDLEDEV hDevice, STImgAndInfoParam imgParam, unsigned char* imgBuf, unsigned int* imgBufLen, STCodeInfoList* pCodeInfoList);</pre>	brief 根据参数获取解码信息以及解码成功图像数据 param[in] hDevice 设备句柄 param[in] imgParam 图像参数结构 format, 图像格式: 0 - 原始图像 (Raw data), 1 - BMP 格式, 2 - JPEG 格式 quality, JPEG 格式的图像质量: 0 - Low, 1 - Middle, 2 - High, 3 - Highest param[out]imgBuf 返回的图像数据,需要足够大的空间进行接收 param[in/out] imgBufLen [in] imgBuf 分配的空间长度 [out] 返回的图像数据真实长度 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 return 是否接收成功, 返回 true, 表示成功 false, 表示失败
<pre>bool nl_GetDeviceInfo(const HANDLEDEVLST hDeviceList, unsigned int index, STDeviceInfo* stNetDevInfo);</pre>	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
<pre>char *nl_GetLastError();</pre>	brief 获取最后一次操作的错误信息 return 错误信息
<pre>bool nl_SetNetConfig(char *sn, char *mac, bool dhcp, char *ip, char *subNetMask, char *gateway, int recTimeout);</pre>	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
<pre>bool nl_GetDeviceInfoByHandle(const</pre>	brief 获取设备信息

HANDLEDEV hDevice, STDeviceInfo* stNetDevInfo);	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_GetRoiPicData(const HANDLEDEV hDevice, STRoImgParam 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 sopid -业务方案序号 unit -业务单元序号 roid -ROI 序号 (99:无序号(例如上一节点裁剪的图): ROI 序号从 1 开始) type -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

以下为网络独立接口,在确定设备 IP 地址时使用

bool 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 true: 成功 false: 失败
void nl_CloseClientSocket(int *socket);	brief 关闭客户端 socket 套接字 param[in] socket 客户端套接字
bool 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 true: 成功 false: 失败
bool 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 true: 成功 false: 失败
<pre>bool nl_getNetImgData(int socket, STImgParam imgParam, char *imgData, int *realLen, IMG_TYPE* imgtype, int *width, int *heigh);</pre>	brief 通过网络获取图像数据 param[in] socket 网络套接字 param[in] imgParam 图像参数结构 type, 类型: 0 - 实时图像(最后一次拍摄的图像), 1 - 解码成功的图像 format, 图像格式: 0 - 原始图像(Raw data), 1 - BMP 格式, 2 - JPEG 格式 quality, JPEG 格式的图像质量: 0 - Low, 1 - Middle, 2 - High, 3 - 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 true: 成功 false: 失败
<pre>bool nl_GetRoiPicDataByNet(int socket, STRoiImgParam imgParam, STFrameData* frameData, unsigned char* imgBuf, unsigned int* imgBufLen);</pre>	brief 获取 ROI 图片 param[in] socket 网络套接字 param[in] imgParam 图片参数结构 sopid -业务方案序号 unit -业务单元序号 roiid -ROI 序号 (99:无序号(例如上一节点裁剪的图): ROI 序号从 1 开始) type -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 true: 成功 false: 失败
<pre>T_CommunicationResult nl_SendCommandFromSocket(int socket, const char* command, unsigned int commandLen);</pre>	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] socket 网络套接字 param[in] command 指令 param[in] commandLen 指令长度 return 通讯结果

```
bool
nl_GetCommandResponseFromSocket(int
socket, const char* command,
unsigned int commandLen, char*
response, int* responseLen, unsigned
int timeout, bool isPacked = true, bool
isHex = false);
```

brief 发送指令并接收返回指令

param[in] socket 网络套接字

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[in/out]responseLen

[in]response 分配的空间长度

[out]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

枚举说明

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
};
```

