

# Instruction for N-ScanHub

## 1 Introduction

Newland's SDK supports Windows and linux platforms and offers the C/C++ interface to interact with Newland devices. With the SDK, users can carry out secondary development, obtain devices, send instructions, upgrade firmware, etc.

### Directory Structure

Items	Descriptions
Platform	Windows and Linux platforms
Programming Language	C/C++
Functions	Obtaining device, sending commands, upgrading firmware, read and write, opening and closing the device , collecting pictures, plugging and unplugging and data acquisition notification, etc.
SDK	N-ScanHubForLinux and N-ScanHubForWindows
API	N-ScanHubForLinux and N-ScanHubForWindows with the same interface name

## 2 Introduction to N-ScanHubForWindows

### 2.1 Directory Structure

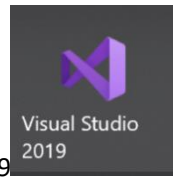
N-ScanHubForWindows offers the API under the Windows platform, and its directory is shown as below.

Contents	Descriptions
include	Header file: N-ScanHub.h (all interfaces descriptions included)
lib/x64	64-bit N-ScanHub.dll and N-ScanHub.lib
lib/x86	32-bit N-ScanHub.dll and N-ScanHub.lib
demo	Library file and demo written by Visual Studio2019
help	Help document: N-ScanHub.pdf

## 2.2 NLSInfoStreamForWindows Operating Instructions



Device: FM430

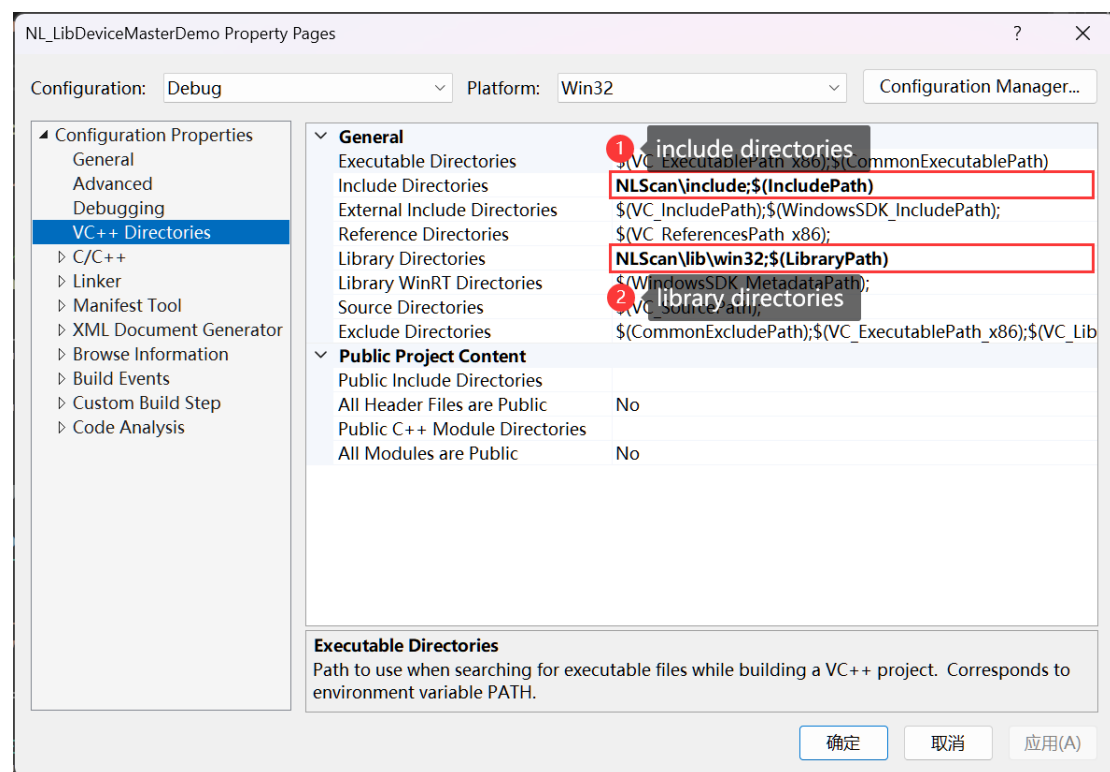


Tool: VS2019

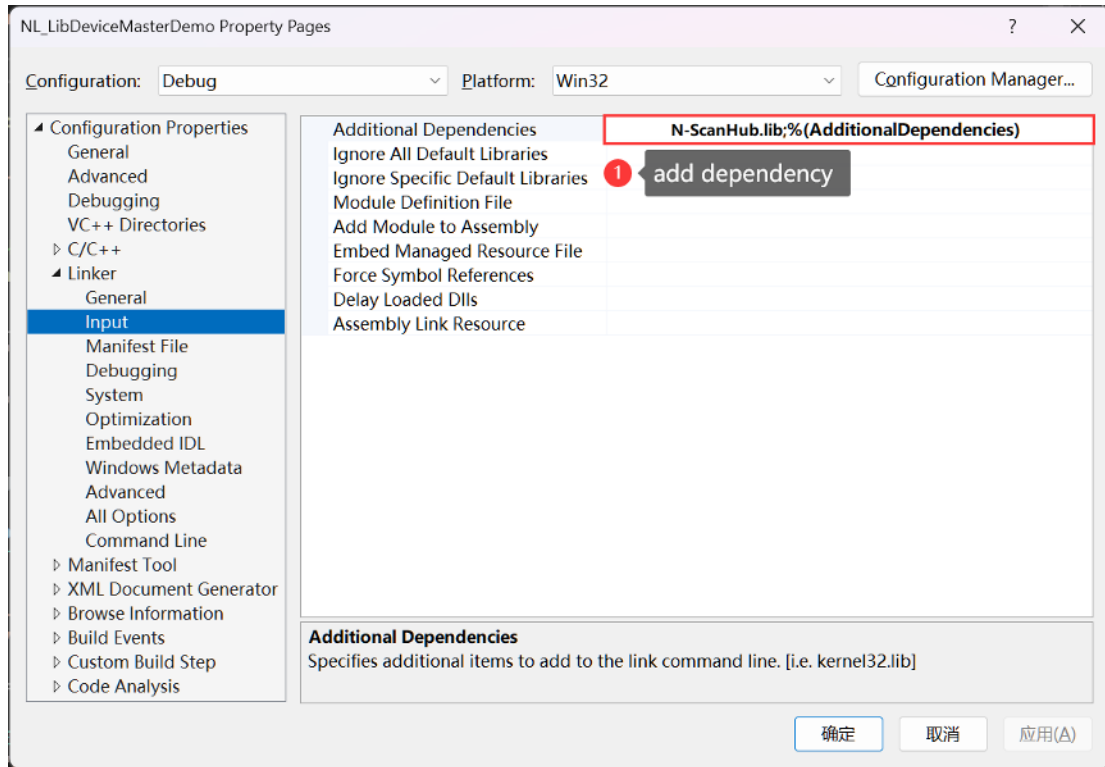
System: Windows 10

### N-ScanHubForWindows demo Operating Steps

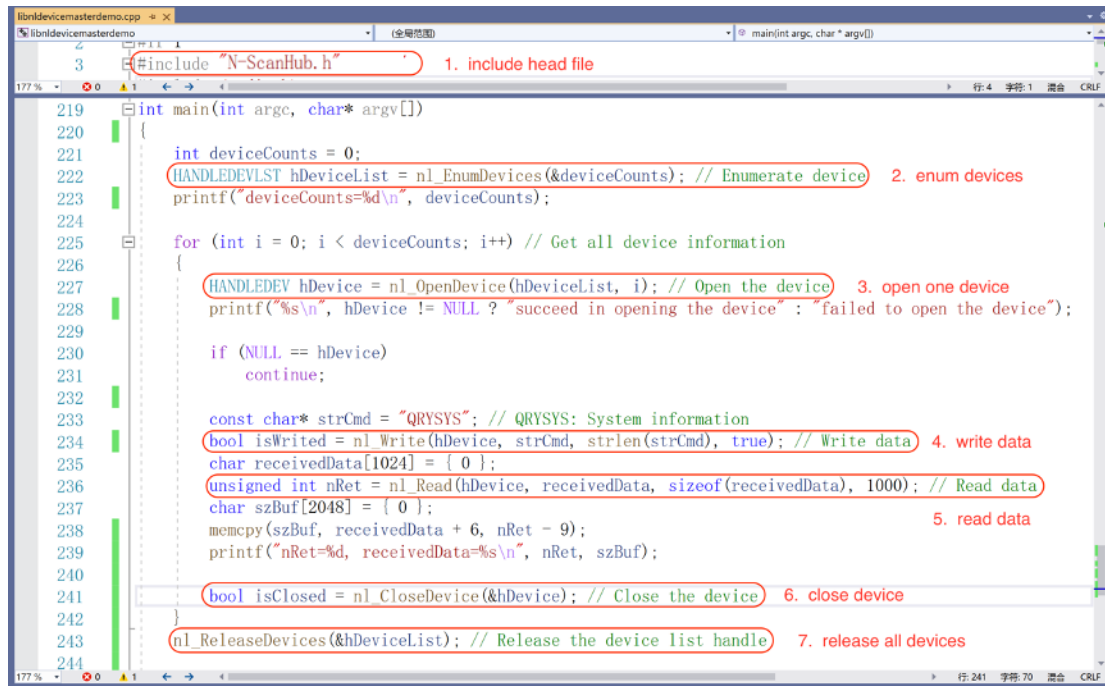
1. The project contains paths of N-ScanHub.h and N-ScanHub.lib.



2. Add the dependency N-ScanHub.lib.



3. Call the function in the SDK.



4. Start running the program.

```
219 int main(int argc, char* argv[])
220 {
221     int deviceCounts = 0;
222     HANDLEDEVLIST hDeviceList = nl_EnumDevices(&deviceCounts); // Enumerate device
223     printf("deviceCounts=%d\n", deviceCounts);
224
225     for (int i = 0; i < deviceCounts; i++) // Get all device information
226     {
227         HANDLEDEV hDevice = nl_OpenDevice(hDeviceList, i); // Open the device
228         printf("%s\n", hDevice != NULL ? "succeed in opening the device" : "failed to open the device");
229
230         if (NULL == hDevice)
231             continue;
232
233         const char* strCmd = "QRYSYS"; // QRYSYS: System information
234         bool isWrote = nl_Write(hDevice, strCmd, strlen(strCmd), true); // Write data
235         char receivedData[1024] = { 0 };
236         unsigned int nRet = nl_Read(hDevice, receivedData, sizeof(receivedData), 1000); // Read data
237         char szBuf[2048] = { 0 };
238         memcpy(szBuf, receivedData + 6, nRet - 9);
239         printf("nRet=%d, receivedData=%s\n", nRet, szBuf);
240
241         bool isClosed = nl_CloseDevice(&hDevice); // Close the device
242     }
243     nl_ReleaseDevices(&hDeviceList); // Release the device list handle
244 }
```

The result is as follows.

```
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:
```

### 3 Interface description

The SDK under Windows and Linux uses an API with the same name. The specific functions are as follows:

Function list	
Function	description
HANDLEDEVLIST nl_EnumDevices(int* deviceCount, EnumType = ENUM_ALL);	brief:enumerate device. param[in] enumType Enumerate all types of devices by default param[out] deviceCount Number of device

	<p>return:Device list handle    Non-null: device list exists.    Null: device list doesn't exist.</p>
<pre>void nl_ReleaseDevices(HANDLEDEVLST* hDeviceList);</pre>	<p>brief:Release the device list handle.</p> <p>param[in] hDeviceList Device list handle</p>
<pre>HANDLEDEV nl_OpenDevice(const HANDLEDEVLST hDeviceList, unsigned int index, T_Porotocol porotocol = Nlscan);</pre>	<p>brief:Specify the indexed device on the device list.</p> <p>param[in] hDeviceList Device list handle</p> <p>param[in] index device index</p> <p>param[in] porotocol    Protocol of the manufacturer</p> <p>return:Device handle    Non-null: succeed in opening.    Null: failed to open.</p>
<pre>bool nl_Write(const HANDLEDEV hDevice, const char* data, unsigned int len, bool isPacked = true);</pre>	<p>brief:Write data to the device.</p> <p>param[in] hDevice Device handle</p> <p>param[in] data Written data</p> <p>param[in] len Data length</p> <p>param[in] isPacked Whether data is packed</p> <p>return:Whether data is written. true: succeed in writing data. false: failed to write data.</p>
<pre>bool nl_WriteAsHex(const HANDLEDEV hDevice, const char* data, bool isPacked = false);</pre>	<p>brief:Write data to the device in the form of HEX character string.</p> <p>param[in] hDevice Device handle</p> <p>param[in] data Written data</p> <p>param[in] isPacked Whether data is packed</p> <p>return:Whether data is written. true: succeed in writing data. false: failed to write data.</p>
<pre>T_CommunicationResult nl_SendCommand(const HANDLEDEV hDevice, const char* command, unsigned int commandLen);</pre>	<p>brief:Send control commands to the device (Commands will be packed according to different protocols inside the interface).</p> <p>param[in] hDevice Device handle</p> <p>param[in] command commands sent</p> <p>param[in] commandLen Command length</p> <p>return:Communication result</p>
<pre>T_CommunicationResult nl_SendCommandAsHex(const HANDLEDEV hDevice, const char* command, unsigned int commandLen);</pre>	<p>brief:Send control commands to the device in the form of HEX character string (Commands will be packed according to different protocols inside the interface).</p> <p>param[in] hDevice Device handle</p> <p>param[in] command Commands sent</p> <p>param[in] commandLen Command length</p> <p>return:Communication result</p>

<pre>unsigned int nl_Read(const HANDLEDEV hDevice, char* buf, unsigned int len, unsigned int timeout);</pre>	<p>brief:Read device data.</p> <p>param[in] hDevice Device handle</p> <p>param[out] buf data returned from the device</p> <p>param[in] len Received data length</p> <p>param[in] timeout Data reading timeout When it is set as 0, it continues reading until there is no returned data.</p> <p>return:Data length returned from the device</p>
<pre>void nl_SetListenerParam(const HANDLEDEV hDevice, void *userParam);</pre>	<p>brief Sets the user parameters of the listening callback</p> <p>param[in] hDevice Device handle</p> <p>param[in] userParam User parameter pointer</p>
<pre>void nl_SetListener(const HANDLEDEV hDevice, readCallback callback);</pre>	<p>brief:Set monitor.</p> <p>param[in] hDevice Device handle</p> <p>param[in] callback callback function</p>
<pre>bool nl_StopListener(const HANDLEDEV hDevice);</pre>	<p>brief:Stop monitoring device data.</p> <p>param[in] hDevice Device handle</p> <p>return:Whether monitoring device data is stopped. true: succeed in stopping monitoring. false: failed to stop monitoring.</p>
<pre>bool nl_GetPicSize(const HANDLEDEV hDevice, unsigned int* width, unsigned int* height);</pre>	<p>brief:Get the size of device image.</p> <p>param[in] hDevice Device handle</p> <p>param[out] width Image width</p> <p>param[out] height Image height</p> <p>return:Whether device image size is obtained.true: succeed in getting device image. false: failed to get device image.</p>
<pre>bool nl_GetPicData(const HANDLEDEV hDevice, unsigned char* imgBuf, int imgBufLen);</pre>	<p>brief:Get device image.</p> <p>param[in] hDevice Device handle</p> <p>param[out] imgBuf Image data</p> <p>param[in] imgBufLen Image data length</p> <p>return:Whether device image is obtained.true: succeed in getting device image. false: failed to get device image.</p>
<pre>bool nl_UpdateKernelDevice(const HANDLEDEV hDevice, const char* strFileName, unsigned int reserved = 0, unsigned int* error = 0);</pre>	<p>brief:Update device.</p> <p>param[in] hDevice Device handle</p> <p>param[in] strFileName path of firmware file</p> <p>param[in] reserved Reserved field</p> <p>param[out] error Error number returned after the update failed.</p>

	<p>return:Whether updating is successful.true: succeed in updating. false: failed to update.</p>
<pre>bool nl_CloseDevice(HANDLEDEV* hDevice);</pre>	<p>brief:Close the device.</p> <p>param[in] hDevice Device handle</p> <p>return:Whether the device is closed.true: succeed in closing the device. false: failed to close the device.</p>
<pre>bool nl_SavePicDataToFile(const char* bmpName, unsigned char* imgBuf, int width, int height, int flag);</pre>	<p>brief:Encapsulate the collected image data into BMP format and save it as a file.</p> <p>param[in] bmpName bmp file name</p> <p>param[in] imgBuf Image buffer data</p> <p>param[in] width Image width</p> <p>param[in] height Image height</p> <p>param[in] flag Image bit depth or image quality level</p> <p>When saving a file as a BMP bitmap, the image bit depth is specified, with possible values of 8 or 24.</p> <p>When saving a file as a JPG, it represents the image quality level.</p> <p>gray image: (10-Low, 11-Middle, 12-High, 13-Highest)</p> <p>color image: (20-Low, 21-Middle, 22-High, 23-Highest)</p> <p>return:Whether it is saved.true: saved. false: failed to save.</p>
<pre>T_DeviceStatus nl_GetDevStatus(const HANDLEDEV hDevice);</pre>	<p>brief:Get device status.</p> <p>param[in] hDevice Device handle</p> <p>return:Device status</p>
<pre>bool nl_ReadDevCfgToXml(const HANDLEDEV hDevice, const char* cfgFilePath);</pre>	<p>brief:Read the configuration from the device and save it to the xml file.</p> <p>param[in] hDevice Device handle</p> <p>param[in] cfgFilePath Path of configuration file</p> <p>return:Whether it is saved.true: saved. false: failed to save.</p>
<pre>bool nl_WriteCfgToDev(const HANDLEDEV hDevice, const char* cfgFilePath);</pre>	<p>brief:Write the configuration file to the device.</p> <p>param[in] hDevice Device handle</p> <p>param[in] cfgFilePath Path of configuration file</p> <p>return:Whether it is written.true: written. false: failed to write.</p>
<pre>void nl_SetCbDevStatusChanged(const HANDLEDEV hDevice, DevStatChgCallback)</pre>	<p>brief:Set the callback function when device status changes.</p>

ck callback);	param[in] hDevice Device handle param[in] callback Callback function
bool nl_GetCommandResponse(const HANDLEDEV hDevice, const char* command, unsigned int commandLen, char* response, int *responseLen, unsigned int timeout, bool isPacked, bool isHex);	<p>brief Send commands and receive return commands.</p> <p>param[in] hDevice Device handle param[in] command command sent param[in] commandLen command length param[out]response command response param[in/out] responseLen [in]The length of response allocation space [out] command response length param[in]timeout time out param[in]isPacked Whether data is packed param[in]isHex Whether data is Hex return true: successful. false: failed</p>
bool nl_GetPicDataByConfig(const HANDLEDEV hDevice, STImgParam imgParam, unsigned char* imgBuf, unsigned int *imgBufLen, STImgResolution* imgR);	<p>brief Retrieve image data based on the parameters</p> <p>param[in] hDevice Device handle param[in] imgParam image param set</p> <p>T, type: 0T - Real-time image (the latest captured image), 1T - Decoded successful image. R, Image ratio: 0R - original, 1R - 1/4, 2R - 1/16 F, Image format: 0F - Raw data, 1F - BMP, 2F - JPEG Q, JPEG quality level: 0Q - Low, 1Q - Middle, 2Q - High, 3Q – Highest</p> <p>Other parameters are temporarily reserved, initialized as 0x00</p> <p>param[out]imgBuf The returned image data requires a sufficiently large space for reception param[in/out] imgBufLen [in]The length of imgBuf allocation space [out] image data length</p> <p>param[out]imgR Keep the parameters, temporarily unused. The coordinates of the four endpoints of the barcode area, if available, require applying for an STImgResolution[4] array in advance.</p> <p>return true: successful. false: failed</p>
bool nl_GetPicDataAndCodeInfo(const HANDLEDEV hDevice, STImgAndInfoParam imgParam, unsigned char* imgBuf, unsigned int* imgBufLen, STCodeInfoList* pCodeInfoList);	<p>brief According to the parameters, the decoding information and the successful decoding image data are obtained</p> <p>param[in] hDevice Device handle param[in] imgParam image param set R, Image ratio: 0R - original, 1R - 1/4, 2R - 1/16</p>



	<p>F, Image format: 0F - Raw data, 1F - BMP, 2F - JPEG  Q, JPEG quality level: 0Q - Low, 1Q - Middle, 2Q - High, 3Q – Highest</p> <p>param[out] imgBuf The returned image data requires a sufficiently large space for reception</p> <p>param[in/out] imgBufLen  [in]The length of imgBuf allocation space  [out] image data length</p> <p>param[out] pCodeInfoList list of decoding information structures that can store up to 64 code information parsed simultaneously</p> <p>return true: successful. false: failed</p>
<p>IMG_TYPE</p> <p>nl_GetDeviceImageColorType(const HANDLEDEV hDevice, STImgResolution* imgResOut, unsigned int * imgLen);</p>	<p>brief Obtaining the image type of the device's raw image</p> <p>param[in] hDevice Device handle</p> <p>param[out] imgResOut The real resolution of the raw image, If it's a color image, it's the converted resolution.</p> <p>param[out] imgLen image data real length</p> <p>return image type</p>
<p>bool</p> <p>nl_ConvertImageColorSpace(const HANDLEDEV hDevice, unsigned char* imgBufIn, long imgBufInLen, STImgResolution imgResIn, unsigned char* imgBufOut);</p>	<p>brief Color space conversion of the raw image nv12-&gt;bgr</p> <p>param[in] hDevice Device handle param[in] imgBufIn Raw image data</p> <p>param[in] imgBufInLen Raw image data length</p> <p>param[in] imgResIn The real resolution of the raw image</p> <p>param[out] imgBufOut Image data after color space conversion</p> <p>return true: successful. false: failed</p>
<p>bool nl_GetDeviceInfo(const HANDLEDEVLST hDeviceList, unsigned int index, STDeviceInfo* stNetDevInfo);</p>	<p>brief Retrieve device information</p> <p>param[in] hDeviceList Device handle list</p> <p>param[in] index device index</p> <p>param[out] stNetDevInfo device information struct</p> <p>return true: successful. false: failed</p>
<p>bool nl_DevicelsOpenByHandle(const HANDLEDEV hDevice);</p>	<p>brief Is the device open</p> <p>param[in] hDevice Device handle</p> <p>return true: open. false: close</p>
<p>bool nl_DevicelsOpenByList(const HANDLEDEVLST hDeviceList, unsigned int index);</p>	<p>brief Is the device open</p> <p>param[in] hDeviceList Device handle list</p> <p>param[in] index device index</p>

	return true: open. false: close
char *nl_GetLastError();	brief Retrieve the error message from the last operation return error message
bool nl_SetNetConfig(char *sn, char *mac, bool dhcp, char *ip, char *subNetMask, char *gateway, int recTimeout);	brief Set network device configuration information param[in] sn serial number param[in] mac mac address param[in] dhcp Dynamic host setup protocol param[in] ip ip address param[in] subNetMask Subnet mask param[in] gateway gateway param[in] recTimeout timeout return true false
bool nl_GetDeviceInfoByHandle(const HANDLEDEV hDevice, STDeviceInfo* stNetDevInfo);	brief Retrieve device information param[in] hDevice Device handle param[out] stNetDevInfo device information struct return true: successful. false: failed
bool nl_SendBmpDecode(const HANDLEDEV hDevice, const IMG_TYPE imgType, char* bmpFileName, char* retDecode, int* retLen, int* decodeTime, int timeout);	brief send image to decode param[in] hDevice Device handle param[in] imgType image type param[in] bmpFileName image path param[out] retDecode code data param[out] retLen code len param[out] decodeTime decode time param[in] timeout time out return true: successful. false: failed notice: The image must be obtained by Newland device
bool nl_GetRoiPicData(const HANDLEDEV hDevice, STRoiImgParam imgParam, STFrameData* frameData, unsigned char* imgBuf, unsigned int* imgBufLen);	brief Retrieve roi image data based on the parameters param[in] hDevice Device handle param[in] imgParam roi image param set p server number u server unit number i ROI number, Start at 1 (99: unordered number (e.g., the graph clipped at the previous node)); t type: OT - Input image (last taken image),1T - Result image param[out] frameData Roi Image frame data param[out]imgBuf The returned image data requires

	a sufficiently large space for reception param[in/out] imgBufLen [in]The length of imgBuf allocation space [out] image data length return true: successful. false: failed
void nl_LogSwitch(bool isLogging);	brief Log Switch param[in] isLogging true-on false-off
void nl_GetVersion(char* version);	brief SDK Version param[out] version SDK Version
HANDLEDEVLST nl_AddNetworkDevice(char *ipList);	brief Add Device by IP Address and Port param[in] Device Ip Address List: "192.168.1.100:36520,192.168.1.101:36520,192.168.1.102:36520" return Device list handle    Non-null: device list exists.    Null: device list doesn't exist

#### Network independent interfaces

int nl_CreateTcpService(int port, tcpServiceBack callback);	brief Create a network server. param[in] port network port param[in] callback Callback function return Less than 0 fail.
int nl_CloseClientSocket(int *socket);	brief Close the client socket param[in] socket network socket return 0 successful other fail
int nl_ExitTcpService();	brief exit tcp service return
int nl_connectToService(char* serviceIp, int port, int* socket);	brief connect to tcp service param[in] serviceIp service ip param[in] port service port param[out] socket network socket return 0 successful other fail
int nl_sendDataToSocket(int socket, char* buf, int buf_len);	brief Send data by socket param[in]socket network socket param[in]buf send data param[in]buf_len send data length return 0 successful other fail
int nl_readFromSocket(int socket, int nTimeout, char* outbuf, int *buflen);	brief Receive network data param[in] socket network socket param[in] nTimeout time out param[in] outbuf Receive data

	param[in] buflen Receive data length return 0 successful other fail
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 Image data is obtained through the network param[in] socket network socket param[in] T type: 0T - Real-time image (the latest captured image), 1T - Decoded successful image. param[in] RImage ratio, Keep the parameters, temporarily unused, initialized as 0x00 param[in] F Image format: 0F - Raw data, 1F - BMP, 2F - JPEG param[in] Q JPEG quality level: 0Q - Low, 1Q - Middle, 2Q - High, 3Q – Highest param[out] imgData image data param[in/out] realLen [in]The length of imgData allocation space [out] image data length param[out] imgtype image type param[out] width image width param[out] heigh image heigh return 0 successful other fail
int nl_GetRoiPicDataByNet(int socket, STRoImgParam imgParam, STFrameData* frameData, unsigned char* imgBuf, unsigned int* imgBufLen);	brief Getting ROI images param[in] socket network socket param[in] imgParamImage parameter struct p server number u server unit number i ROI number, Start at 1 (99: unordered number (e.g., the graph clipped at the previous node));) t type: 0T - Input image (last taken image),1T - Result image param[out] frameData Roi Image frame data param[out] imgBuf image data param[in/out] imgBufLen [in] The length of imgBuf allocation space [out] image data length return 0-successful other- failed
T_CommunicationResult nl_SendCommandFromSocket(int socket, const char* command,	Send control commands to the device (Commands will be packed according to different protocols inside the interface).

unsigned int commandLen);	param[in] socket network socket param[in] command commands sent param[in] commandLen Command length return:Communication result
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 Send commands and receive return commands. param[in] socket network socket param[in] command command sent param[in] commandLen command length param[out]response command response param[in/out] responseLen [in]The length of response allocation space [out] command response length param[in]timeout time out param[in]isPacked Whether data is packed param[in]isHex Whether data is Hex return true: successful. false: failed
Serial Port independent interfaces, No need nl_EnumDevices	
HANDLEDEV nl_OpenSerialPortDevice(char* serial);	brief Open the device through the serial port number param[in] serial serial port number like "COM1" return device handle, NULL is fail
bool nl_CloseSerialPortDevice(char* serial, HANDLEDEV* hDevice);	brief Close the device through the serial port number param[in] serial serial port number like "COM1" param[in] hDevice device handle return true-successful, false-fail

Enum Description	
brief:Abnormal type. enum T_ErrorType {	
Success	= 0, ///< Normal.
UnknownError	= 1, ///< Unknown Error.
NotExistError	= 2, ///< The device doesn't exit.
NotOpenError	= 3, ///< The device is not opened.
AlreadyOpenError	= 4, ///< The device is opened.
AccessDeniedError	= 5, ///< Access to the device is denied.
NotInitializedError	= 6, ///< The Device is not initialized.
InvalidParamsError	= 8, ///< Invalid parameters.
InvalidFileFormatError	= 9, ///< Invalid file format.
FileNameExtError	= 10, ///< File name error.
CommunicationError	= 11, ///< Communication error.
MallocError	= 12, ///< Memory allocation error.
UpdateFailedError	= 13, ///< Failed to update.

NoUpdateObjectError	= 14,/// No updating object.
FileNotExistError	= 15,/// the file doesn't exist.
BufferOverflowError	= 16,/// Buffer overflows.
FileNotSuitableError	= 17,/// The file is not suitable.
DeviceNotUniqueError	= 18,/// The device is not unique.
};	
brief:Device status.	
enum T_DeviceStatus	
{	
Opened = 0,	/// Opened.
NotOpened,	/// Not opened.
Closed,	/// Closed.
NotClosed,	/// Not closed.
Updating,	/// Updating...
Updated,	/// Updating is finished.
Writing,	/// Writing data...
Written,	/// Data writing is finished.
Reading,	/// Reading data...
ReadOK,	/// Data reading is finished.
GettingPicData,	/// Getting image data...
GetPicDataOK,	/// Image data has been obtained.
UnknownStatus	/// Unknown status.
};	
brief:Commands sending result.	
enum T_CommunicationResult	
{	
SendError = 0,	/// Sending error.
Support,	/// Commands supported.
Unsupport,	/// Commands not supported.
OutOfRange,	/// Data value is not within the range.
UnknownResult,	/// Unknown error.
};	
brief:Protocol.	
enum T_Porotocol	
{	
Nlscan = 0,	// Newland.
};	
brief:image type	
enum IMG_TYPE {	
TYPE_UNKNOW = 0,	/// unknow image type
TYPE_GRAY = 1,	/// gray image
TYPE_COLOR = 2.	/// color image
};	
brief:device type	

```
enum NL_DEVICE_TYPE {  
    DEV_TYPE_UNKNOW = 0,    ///< unknow device type  
    DEV_TYPE_COM = 1,       ///< serial device  
    DEV_TYPE_USB = 2,       ///< usb device  
    DEV_TYPE_NET = 4,       ///< net device  
};
```

brief The type of network parameter setting.

```
enum NET_SETTING_TYPE {  
    DEV_SETTING = 0,        ///< Device network parameters  
    GROUP_SETTING = 1,      ///< Network group parameters  
};
```