

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

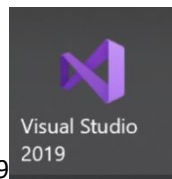
N-ScanHubForWindows offers the API under the Windows platform, and its directory is shown as below.

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

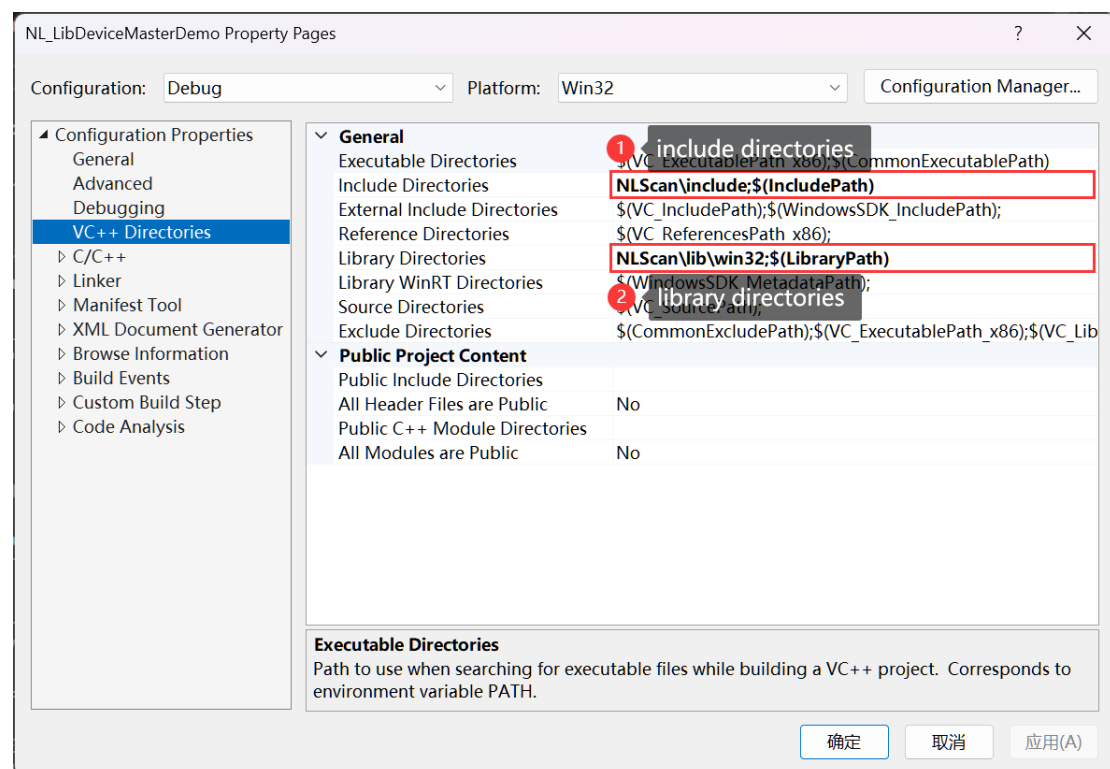


Tool: VS2019

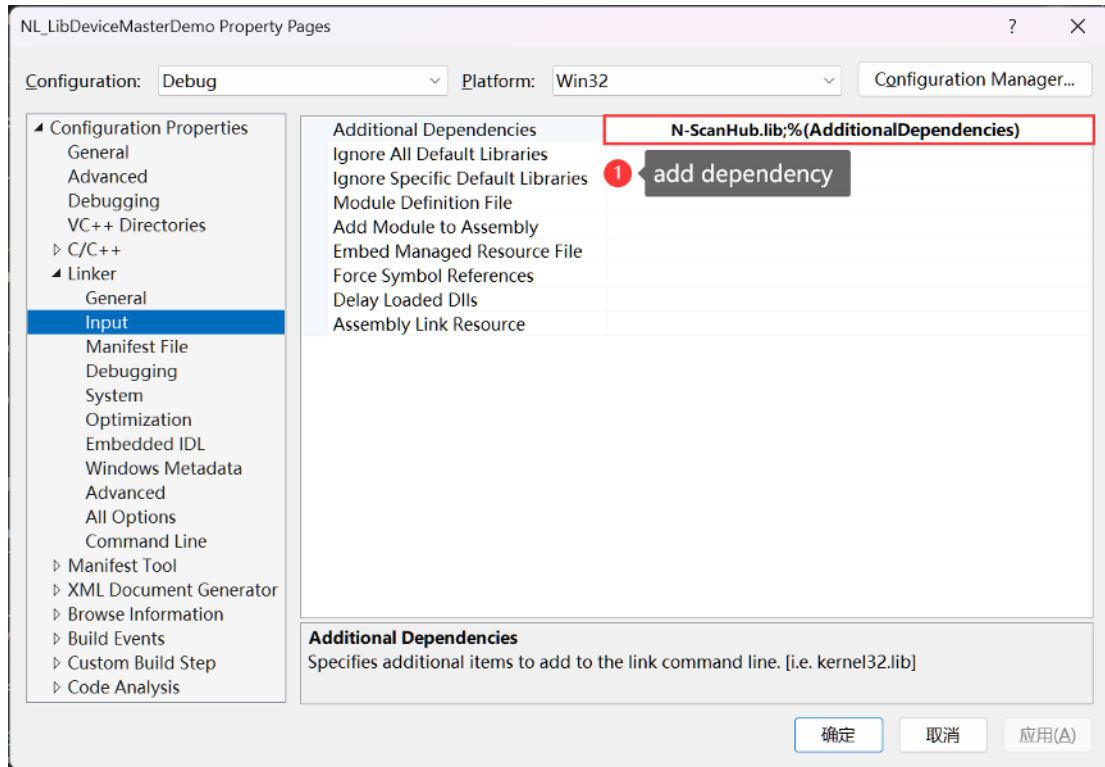
System: Windows 10

N-ScanHubForWindows demo Operating Steps

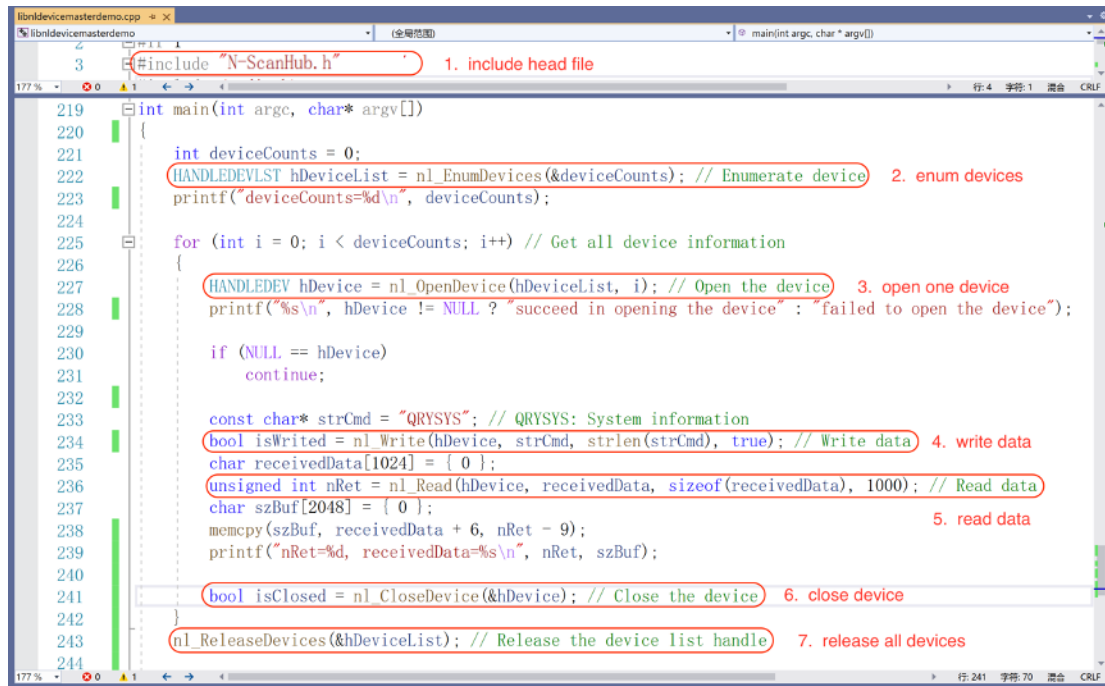
1. The project contains paths of N-ScanHub.h and N-ScanHub.lib.



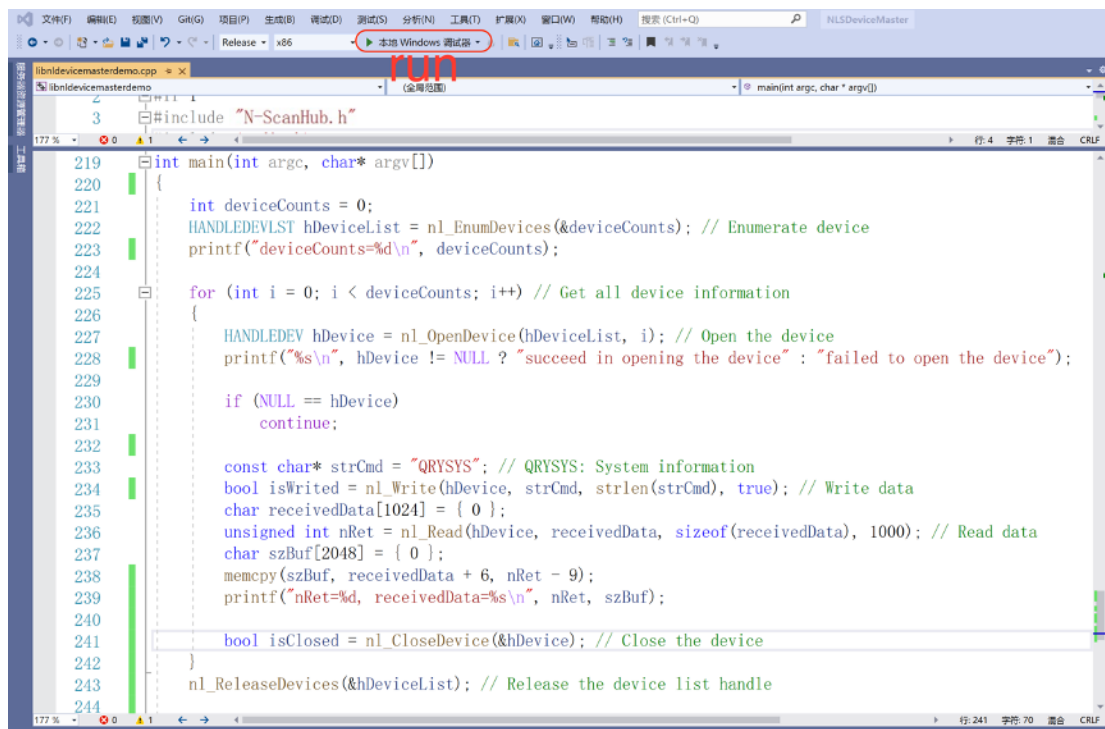
2. Add the dependency N-ScanHub.lib.



3. Call the function in the SDK.



4. Start running the program.



The result is as follows.

```

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:

```

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
HANDLEDEVLST 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

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

<pre>unsigned int nl_Read(const HANDLEDEV hDevice, char* buf, unsigned int len, unsigned int timeout);</pre>	brief:Read device data. param[in] hDevice Device handle param[out] buf data returned from the device param[in] len Received data length param[in] timeout Data reading timeout When it is set as 0, it continues reading until there is no returned data. return:Data length returned from the device
<pre>void nl_SetListenerParam(const HANDLEDEV hDevice, void *userParam);</pre>	brief Sets the user parameters of the listening callback param[in] hDevice Device handle param[in] userParam User parameter pointer
<pre>void nl_SetListener(const HANDLEDEV hDevice, readCallback callback);</pre>	brief:Set monitor. param[in] hDevice Device handle param[in] callback callback function
<pre>bool nl_StopListener(const HANDLEDEV hDevice);</pre>	brief:Stop monitoring device data. param[in] hDevice Device handle return:Whether monitoring device data is stopped. true: succeed in stopping monitoring. false: failed to stop monitoring.
<pre>bool nl_GetPicSize(const HANDLEDEV hDevice, unsigned int* width, unsigned int* height);</pre>	brief:Get the size of device image. param[in] hDevice Device handle param[out] width Image width param[out] height Image height return:Whether device image size is obtained.true: succeed in getting device image. false: failed to get device image.
<pre>bool nl_UpdateKernelDevice(const HANDLEDEV hDevice, const char* strFileName, unsigned int reserved = 0, unsigned int* error = 0);</pre>	brief:Update device. param[in] hDeviceList Device list handle param[in] index device index param[in] strFileName path of firmware file return:Whether updating is successful.true: succeed in updating. false: failed to update.
<pre>bool nl_CloseDevice(HANDLEDEV* hDe vice);</pre>	brief:Close the device. param[in] hDevice Device handle return:Whether the device is closed.true: succeed in closing the device. false: failed to close the device.
<pre>bool nl_SavePicDataToFile(const char* bmpName, unsigned char* imgBuf, int width, int height, int flag);</pre>	brief:Encapsulate the collected image data into BMP format and save it as a file. param[in] bmpName bmp file name

	param[in] imgBuf Image buffer data param[in] width Image width param[in] height Image height param[in] flag Image bit depth or image quality level When saving a file as a BMP bitmap, the image bit depth is specified, with possible values of 8 or 24. When saving a file as a JPG, it represents the image quality level. gray image: (10-Low, 11-Middle, 12-High, 13-Highest) color image: (20-Low, 21-Middle, 22-High, 23-Highest) return:Whether it is saved.true: saved. false: failed to save.
<pre>bool nl_ReadDevCfgToXml(const HANDLEDEV hDevice, const char* cfgFilePath);</pre>	brief:Read the configuration from the device and save it to the xml file. param[in] hDevice Device handle param[in] cfgFilePath Path of configuration file return:Whether it is saved.true: saved. false: failed to save.
<pre>bool nl_WriteCfgToDev(const HANDLEDEV hDevice, const char* cfgFilePath);</pre>	brief:Write the configuration file to the device. param[in] hDevice Device handle param[in] cfgFilePath Path of configuration file return:Whether it is written.true: written. false: failed to write.
<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 Send commands and receive return commands. 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
<pre>bool nl_GetPicData (const HANDLEDEV hDevice, STImgParam</pre>	brief Retrieve image data based on the parameters param[in] hDevice Device handle

imgParam, unsigned char* imgBuf, unsigned int *imgBufLen);	param[in] imgParam image param set type: 0 - Real-time image (the latest captured image), 1 - Decoded successful image. format: 0 - Raw data, 1 – BMP, 2 - JPEG quality: 0 - Low, 1 – Middle, 2 – High, 3 - Highest 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
bool nl_GetPicDataAndCodeInfo(const HANDLEDEV hDevice, STImgAndInfoParam imgParam, unsigned char* imgBuf, unsigned int* imgBufLen, STCodeInfoList* pCodeInfoList);	brief According to the parameters, the decoding information and the successful decoding image data are obtained param[in] hDevice Device handle param[in] imgParam image param set format, Image format: 0 - Raw data, 1 - BMP, 2 - JPEG quality, JPEG quality level: 0 - Low, 1 - Middle, 2 - High, 3 – Highest 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 param[out] pCodeInfoList list of decoding information structures that can store up to 64 code information parsed simultaneously return true: successful. false: failed
bool nl_GetDeviceInfo(const HANDLEDEVLST hDeviceList, unsigned int index, STDeviceInfo* stNetDevInfo);	brief Retrieve device information param[in] hDeviceList Device handle list param[in] index device index param[out] stNetDevInfo device information struct return true: successful. false: failed
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_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 sopid- server number unit- server unit number roiid- ROI number, Start at 1 (99: unordered number (e.g., the graph clipped at the previous node);) 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

Network independent interfaces

bool 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 true: successful, false: fail
void nl_CloseClientSocket(int *socket);	brief Close the client socket param[in] socket network socket
bool nl_sendDataToSocket(int socket,	brief Send data by socket

char* buf, int buf_len);	param[in]socket network socket param[in]buf send data param[in]buf_len send data length return true: successful, false: fail
bool 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 true: successful, false: fail
bool nl_getNetImgData(int socket, STImgParam imgParam, char *imgData, int *realLen);	brief Image data is obtained through the network param[in] socket network socket param[in] imgParam image param set type: 0 - Real-time image (the latest captured image), 1 - Decoded successful image. format: 0 - Raw data, 1 – BMP, 2 - JPEG quality: 0 - Low, 1 – Middle, 2 – High, 3 - Highest param[in/out] realLen [in]The length of imgData allocation space [out] image data length return true: successful, false: fail
bool 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 sopid- server number unit- server unit number roid- ROI number, Start at 1 (99: unordered number (e.g., the graph clipped at the previous node);) type- OT - 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 true: successful, false: fail
T_CommunicationResult nl_SendCommandFromSocket(int socket, const char* command, unsigned int commandLen);	Send control commands to the device (Commands will be packed according to different protocols inside the interface). 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

Enum Description
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

}