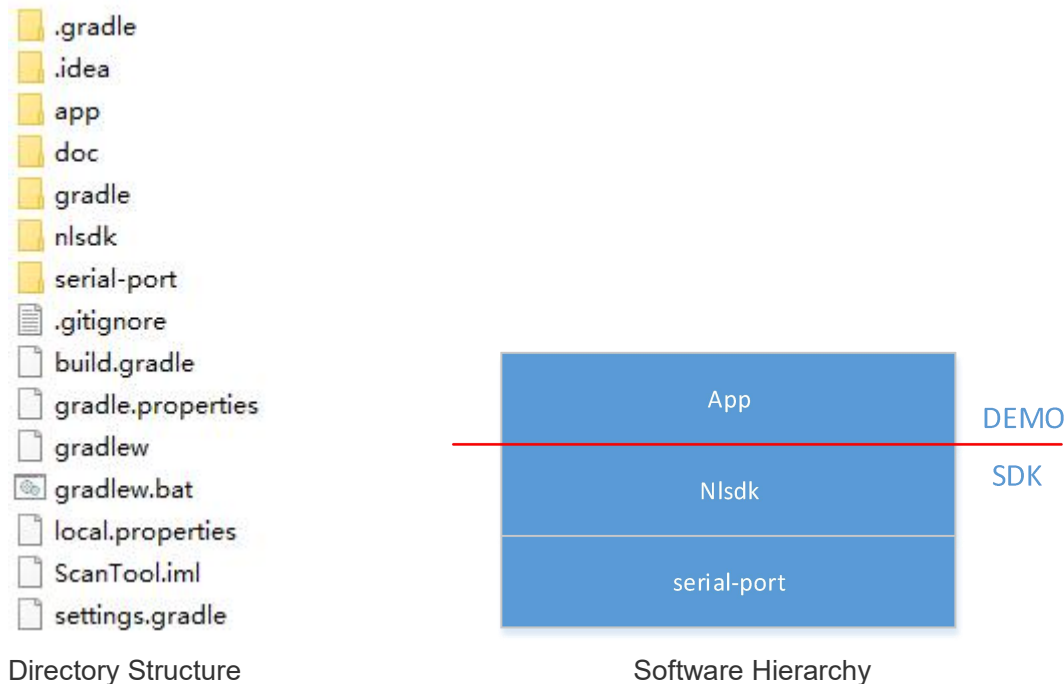# Instruction for N-ScanHub

Introduction

N-ScanHub is based on a SDK+demo package for Android system, and the project is builded by Android Studio. The directory structure and software hierarchy of the package are shown as below:

Directory Structure   Software Hierarchy

serial-port and nlsdk are SDK-related source directories, and app is demo-related source directory. N-ScanHub app adopts the following third-party development packages:

```
implementation 'com.leon:lfilepickerlibrary:1.8.0'
implementation 'io.reactivex.rxjava2:rxjava:2.0.1'
implementation 'io.reactivex.rxjava2:rxandroid:2.0.1'
```

Note:

a. When building the project for the first time, ensure that the host can connect to the Internet. Gradle will automatically update the required third-party tools, and please be patient for updating.

b. Please pay attention to the copyright statement of the third-party library.

c. Applicable products: the device that is suitable for the Easyset is also applicable to this tool.

SDK Interface

The procedure is: new device->open the device->operate the device-> close the device (optional).

1. New a stream instance of the communication interface.
   The NLDevice constructor parameter is specified as the communication type to be used.
   DEV_CDC   USB CDC

DEV_POS	USB POS
DEV_COMPOSITE	USB HID Keyboard and POS
DEV_SUREPOS	IBM SurePOS
DEV_UART	UART

```java
NLDeviceStream ds = new NLDevice(NLDeviceStream.DevClass.DEV_UART);
```

2. Open the stream instance

a. Open the serial port

Access to the serial port requires that the device has root authority to operate the device nodes in the /dev/ directory.

```java
if (!ds.nl_OpenDevice("/dev/ttyAMA0", 115200, new NLDeviceStream.NLUartListener() {
    @Override
    public void actionRecv(byte[] recvBuff, int len) {
        barcodeLen = len;
        if (usbOpenChecked) {
            System.arraycopy(recvBuff, 0, barcodeBuff, 0, len);
            observable.subscribeOn(Schedulers.newThread())
                    .observeOn(AndroidSchedulers.mainThread())
                    .subscribe(usbRecvObserver);
        }
    }

})) {
    bnOpenDevice.setText(R.string.TextOpen);
    usbOpenChecked = false;
    return;
}
```

The UART device streams have a monitoring port for receiving the data obtained via the UART interface.

```java
interface NLUartListener {
  void actionRecv(byte [] RecvBuff, int len);
}
```

b. Open the USB communication interface (including CDC/ POS/COMPOSITE)

Each time a USB device is plugged in, unplugged, or the interface type is switched, it causes the host to re-enumerate the device (similar to PNP). This action prompts the system to display an authorization window. Clients must apply a system signature to the program. Once the app has a system signature, it will not prompt the user for authorization confirmation when first opened.

```java
if (!ds.nl_OpenDevice(this, new NLDeviceStream.NLUsbListener() {
        @Override
        public void actionUsbPlug(int event) {
            if (event == 1) {
```

```java
                MainActivity.this.ShowToast(getString(R.string.TextInfoPlugin));
            } else {
                ds.close();
                MainActivity.this.ShowToast(getString(R.string.TextInfoPlugout));
                observable.subscribeOn(Schedulers.newThread())
                        .observeOn(AndroidSchedulers.mainThread())
                        .subscribe(usbPlugObserver);
            }
        }


        @Override
        public void actionUsbRecv(byte[] recvBuff, int len) {
            barcodeLen = len;
            if (usbOpenChecked) {
                System.arraycopy(recvBuff, 0, barcodeBuff, 0, len);
                String prefix = String.format("scanBarCode len:%s data: ", barcodeLen);
                String str = new String(barcodeBuff, 0, barcodeLen);
                runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        showText(prefix, str);
                    }
                });
            }
        }

    })) {
        bnOpenDevice.setText(R.string.TextOpen);
        usbOpenChecked = false;
        return;
    }
}
bnOpenDevice.setText(R.string.TextClose);
usbOpenChecked = true;
setEnable(true);
}
```

USB device streams have two monitoring ports.

```java
interface NLUsbListener {
    /**
     * Notify the application when the USB device is detected to plug in and out.
     * @param event 1:USB device plug in    0:USB device plug out
     */
    void actionUsbPlug(int event);
```

```
   void actionUsbRecv(byte [] RecvBuff, int len);
}
```

actionUsbPlug   used to listening USB plug in and out

actionUsbRecv   used to listening data receiving

3. Operate the opened device streams

A. Obtain images

  a) Obtain the image resolution (when it is called for the first time)

```
int[] wh =  ds.nl_GetPicSize();
```

  b) Obtain the image data

```
if (imgSize != 0) {
    byte[] imgBuf = new byte[imgSize];

    pbUpdate.setVisibility(View.VISIBLE);
    1 usage
    class GetImg implements Runnable {
        public void run() {
            boolean ret = ds.nl_GetPicData(imgBuf, imgSize, new NLDeviceStream.NLTransImgListner() {
                3 usages
                @Override
                public void curProgress(int percent) {
                    Log.d(TAG,  msg: "Img " + percent);
                    runOnUiThread(new Runnable() {
                        @Override
                        public void run() {
                            // update
                            pbUpdate.setProgress(percent);
                            if (percent == 100) {
                                pbUpdate.setVisibility(View.GONE);
                                showText( prefix: "GetImg:",  text: "Get image succ!");
                            }
                        }
                    });
                }
            });
            if (ret) {
                saveImage(imgBuf, w, h);        Listening the process of getting image.
            } else {
                Log.i(TAG,  msg: "Get img fail");
            }
        }
    }
    Thread t = new Thread(new GetImg());
    t.start();
```

B. Update the firmware

Updating progress is completed by setting monitoring.

```
class Update implements Runnable {
    public void run() {
        ds.nl_UpdateKernelDevice(firmware, new NLDeviceStream.NLUpdateListner() {
            10 usages
            @Override
            public void curProgress(String type, NLDeviceStream.NLUpdateState state, int percent) {
                Log.d(TAG, msg: type + ":" + state + " " + percent);
                runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        // update
                        pbUpdate.setProgress(percent);
                        if (type.equals("END update")) {
                            showText( prefix: "Firmware:", text: "Update success!");
                            pbUpdate.setVisibility(View.GONE);
                        }
                    }
                });
            }
        });
    }
}
```

Note：

1. For updating the firmware of MCU devices via USB communication, this type of devices will restart into the boot during the firmware updating process, which will cause the android device to detect the USB device plug in and out, and thus requires users to reauthorize USB access. There are two methods to deal with this situation. First, set the system signature to the application integrated with SDK, which can avoid re-requesting authorization after plug in and out. Second, the internal code of SDK adds 2s delay, which requires that the authorization must be confirmed in time when the authorization box pops up.

```
/* Since the MCU restarts will cause the USB to be unplugged,
after open, the system will pop up a permission confirmation dialog box and return failure,
Here, use the delaying 2s method to reopen again,
which requires the user to confirm that the permission is enabled within 2s!*/
if (!curCommStream.open(mContext)) {

    try {
        Thread.sleep( millis: 2000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

    if (!curCommStream.open(mContext))
        return NLError.ERROR_DEVICE_NOT_EXIST;
}
```

As the code is shown above, for applications that have obtained the system signature, the delay can be reduced to 100ms to improve the updating efficiency.

2. Assure that the device is powered on during the firmware updating process. Confirm that the device has been restarted before powering off the device.

C. Other operations

Omitted.

SDK Interface

ScanTool\nlsdk\src\main\java\com\nlscan\nlsdk\NLDeviceStream.java

Enumeration Type	
enum DevClass	USB Communication Interface
DEV_CDC,	USB CDC
DEV_POS,	USB POS
DEV_COMPOSITE,	USB HID Keyboard and POS
DEV_SUREPOS,	IBM SurePOS (not supported yet)
DEV_UART	UART
enum NLUpdateState	Firmware Updating State
STATE_ENTER_UPDATE,	
STATE_SET_PARAM,	
STATE_SEND_DATA,	
STATE_WAIT_UPDATE,	
STATE_UPDATE_COMPLETE	
Callback Interface	
NLUsbListener: Used to listen USB PNP and receiving data when open a USB device.	
void actionUsbPlug(int event);	Notify the application when the USB device is detected to be plugged in and out. 1:USB plug in 0:USB plug out
void actionUsbRecv(byte [] RecvBuff, int len);	Notify the application when the data is received via the communication interface. RecvBuff: receiving buffer Len: buffer size
NLUartListener Used to listen receiving data when open a UART device.	
void actionRecv(byte [] RecvBuff, int len);	Notify the application when the data is received via the communication interface. RecvBuff: receiving buffer Len: buffer size
transImgListner: used to listen the progress of image transmission when acquiring images.	
void curProgress(int percent);	Percent: transmission progress
updateListner: used to listen updating progress when updating firmware.	
void curProgress(String type, NLUpdateState state, int percent);	type boot: boot loader kernel: kernel code flash: other configuration files state: upgrading status indication, defined in NLUpdateState Percent: percentage of completion under every state
Interface Function	

boolean nl_OpenDevice (Context context, NLUsbListener listener);	When the USB device opens the interface, ensure that it has read and write permission to the USB device node before calling. **Parameters:** context: Android context listener: monitoring USB plug in and out **Return:** true: succeed false: failed
boolean nl_OpenDevice(String pathName, int baudrate, NLUartListener listener);	UART device opens the interface **Parameters:** pathname: UART device name, such as /dev/ttys0 baudrate: UART baud rate listener: listen data receiving **Return:** true: succeed false: failed
NLCommStream nl_GetDevObj();	The application get opened device stream types via the returned stream object. 应用通过返回流对象进行分析打开的设备流类型 **Parameters:** none **Return:** stream object
String nl_GetSdkVersion();	Obtain SDK version No. **Parameters:** none **Return:** SDK version No.
void nl_CloseDevice();	Turn off the device **Parameters:** none **Return:** none
boolean nl_DeviceIsOpen();	Judge whether the device is turned on. **Parameters:** none **Return:** true: on false: off
boolean nl_GetDevStatus();	Judge whether the device functions normally (for the specified supported device, please refer to the user guide). **Parameters:** none **Return:** true: normal false: abnormal

String nl_GetDeviceInfo();	Obtain device information **Parameters:** none **Return:** Response to the command (QRYSYS) sent
boolean nl_StartScan();	Send the trigger command (0x10 0x54 0x04) to start reading. And make sure that the serial trigger command (SCNTCE1) is enabled before scanning. It is available in the trigger mode. **Parameters:** none **Return:** true: succeed in sending the reading command false: failed to send the command
boolean nl_StopScan();	Stop reading barcodes, available in the trigger mode. Barcode scanning can be stopped if barcode data is not received within the timeout when the device receives the command. **Parameters:** none **Return:** true: succeed in sending the command false: failed to send the command
boolean nl_RestartDevice();	Restart the device **Parameters:** none **Return:** true: succeed in sending the command false: failed to send the command
boolean nl_SendCommand(String command);	Send a single programming command, like setConfig ("128ENA1 "), and it won't be saved when the device is powered off. While the command sent with @, like setConfig ("@128ENA1") can be saved after power-off. **Parameters:** Command: programming commands based on Unified Commands Set **Return:** true: succeed in sending the command false: failed to send the command
String nl_ReadDevCfg(String command);	Query the current setting, only available for the single command. For example, send SCNMOD* and respond SCNMOD0. **Parameters:**

	command: query commands **Return:** Response to query commands sent
int nl_UpdateKernelDevice (byte[] fireware, updateListner listner);	Update the firmware of the device, and the firmware upgrading package will contain different contents based on the customer's requirements. **Parameters** Fireware: firmware content buffering Listener: monitor firmware updating progress **Return:** Error types described in {class NLError}
int nl_WriteCfgToDev(File f);	Update device configuration. The configuration file contains multiple configuration contents, and it takes some time to update after the configuration settings are sent. **Parameters:** f Batch configuration file handle in xml format **Return:** >0: succeed in updating； <0: failed to update; =0: succeed in updating and switch the port
int[] nl_GetPicSize();	Obtain the size information of the image like length and width. **Parameters:** none **Return:** Width (int[0]) Height (int[1])
boolean nl_GetPicData (byte[] ImgBuff, int imgSize, transImgListner listner);	Obtain the image of the device, only available for obtaining images with original size and bmp format. **Parameters:** ImgBuff: receive image buffering imgSize: size of image buffering **Return:** true: succeed in obtaining images false: failed to obtaining images