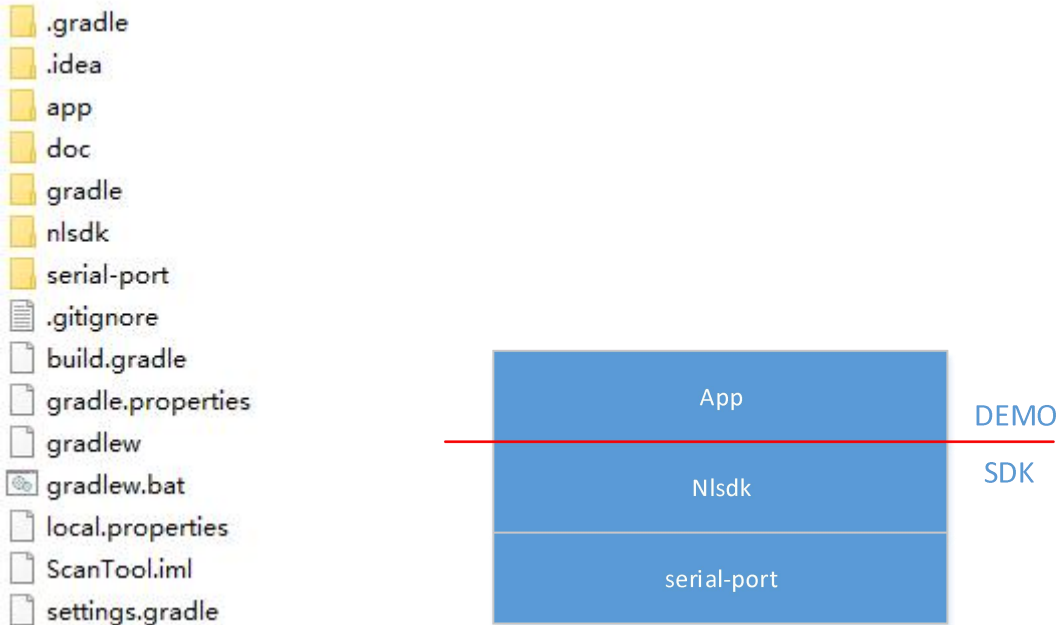


N-ScanHub 使用手册

一. 简介

N-ScanHub 是基于一个用于 Android 系统的 SDK+demo 程序包，工程采用 [Android Studio](#) 编译。
开发包的目录结构和软件层次如下图所示：



开发包目录结构

软件层次

serial-port、nl sdk 是 SDK 相关源码目录，app 是 demo 相关源码目录。

N-ScanHub app 采用了以下第三方开发包：

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

注意：

- 目前仅支持在 Android 主机上打开一台我司设备。
- 在首次编译工程时需确保主机能够连接外网，gradle 会自动更新所需的第三方工具，时间较长请保持耐心。
- 请注意第三方库的版权声明。
- 适用产品：可以使用 [Newland Easyset](#) 软件的设备亦可使用本工具

二. SDK 接口简要使用说明

基本使用流程：创建设备—打开设备—使用设备—关闭设备（可选）

1. 创建通讯接口的流实例

NLDevice 构造函数参数指定为需要使用的通讯类型

DEV_CDC usb 虚拟串口

DEV_POS	USB pos 接口
DEV_COMPOSITE	USB 键盘和 POS 的复合设备
DEV_SUREPOS	IBM SurePos 接口
DEV_UART	物理串口

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

2. 打开流实例

a. 打开物理串口实例

这里对物理串口的访问要求设备具备 root 权限来操作/dev/目录下设备节点。

```
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;
}
```

UART 设备流有 1 个监听接口用于接收 UART 收到的数据:

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

b. 打开 USB 通讯口实例（包含 CDC/ POS/COMPOSITE）

注意：USB 设备每次拔插动和接口类型的切换都会导致主机重新枚举设备（相当于设备拔插），这都会系统的弹出授权的窗口，客户必须给程序打上设备的系统签名，app 有了系统签名就不会在首次打开时提示需求授权的确认框。

```
if (!ds.nl_OpenDevice(this, new NLDeviceStream.NLUsbListener() {
    @Override
    public void actionUsbPlug(int event) {
        if (event == 1) {
            MainActivity.this.ShowToast(getString(R.string.TextInfoPlugin));
        } else {
            ds.close();
            MainActivity.this.ShowToast(getString(R.string.TextInfoPlugout));
        }
    }
})) {
    bnOpenDevice.setText(R.string.TextOpen);
    usbOpenChecked = false;
    return;
}
```

```

        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;
setEnabled(true);
}
}

```

USB 设备流有 2 个监听接口:

```

interface NLUbListener {
    /**
     * 当检测到 USB 设备拔插动作是通知应用
     * @param event 1:USB 设备插入,    0:USB 设备拔出
     */
    void actionUsbPlug(int event);
    void actionUsbRecv(byte [] RecvBuff, int len);
}

```

actionUsbPlug 用来监听 USB 的拔插动作

actionUsbRecv 用来监听接收数据

3. 对打开的设备流进行操作

A. 获取图片：

a) 获取图片分辨率（第一次调用时）

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

b) 获取图像数据

```
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.NLTransImgListener() {  
                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!");  
                            }  
                        }  
                    });  
                }  
            });  
            监听取图进度  
            if (ret) {  
                saveImage(imgBuf, w, h);  
            } else {  
                Log.i(TAG, msg: "Get img fail");  
            }  
        }  
    }  
    Thread t = new Thread(new GetImg());  
    t.start();  
}
```

B. 更新固件：

更新进度通过设置监听来完成

```

class Update implements Runnable {
    public void run() {
        ds.nl_UpdateKernelDevice(firmware, new NLDeviceStream.NLUpdateListener() {
            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);
                        }
                    }
                });
            }
        });
    }
}

Thread t = new Thread(new Update());
t.start();

```

监听下载进度

注:

1. 对于采用 USB 通讯方式更新 mcu 类产品的固件，由于该类产品在固件更新的过程中会重启进入 boot，导致 android 设备会检测到 USB 设备的拔插动作，从而导致要求用户重新进行 USB 访问授权。对此有两种处理方式：一对于集成了 SDK 的应用程序给与系统签名，这么做可以避免拔插重新要求授权；二是 SDK 内部代码增加了 2s 延时，这就要求在设备弹出授权框的时候必须及时确认授权，否则授权失败。

```

/* 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;
}

curCommStream.setUsbListener(mListener);

```

如上述代码所示，对于已经获取系统签名的程序，该处延时可以缩减到 100ms 以提高更新效率。

2. 固件在更新过程中必须保持设备不能断电，务必在确认设备已经重启后再执行断电操作！

C. 其他操作：

略。

- 三. SDK 接口说明

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

内部枚举类型	
enum DevClass USB 通信接口类型,	
DEV_CDC,	usb 虚拟串口
DEV_POS,	USB pos 接口
DEV_COMPOSITE,	USB 键盘和 POS 的复合设备
DEV_SUREPOS,	BM SurePos 接口（目前还未支持）
DEV_UART	物理串口
enum NLUpdateState 固件更新状态	
STATE_ENTER_UPDATE,	
STATE_SET_PARAM,	
STATE_SEND_DATA,	
STATE_WAIT_UPDATE,	
STATE_UPDATE_COMPLETE	
内部回调接口	
NLUsbListener 调用 open 方式时该接口作为系统 USB 事件的监听器接口，用于监听 USB 拔插事件	
void actionUsbPlug(int event);	当检测到 USB 设备拔插动作时通知应用 1:USB 设备插入， 0:USB 设备拔出
void actionUsbRecv(byte [] RecvBuff, int len);	当通讯口接收到数据通知应用 RecvBuff 接收缓冲 len 缓冲大小
NLUartListener 调用 open 方式时，该接口作为串口的接收数据的接口，当串口接收到数据的时候会回调该接口的方法	
void actionRecv(byte [] RecvBuff, int len);	当通讯口接收到数据通知应用 RecvBuff 接收缓冲 len 缓冲大小
transImgListner 调用获取图像时监听传输图像进度	
void curProgress(int percent);	percent 传输进度
updateListner 调用更新固件时监听更新进度信息	
void curProgress(String type, NLUpdateState state, int percent);	type boot: 引导程序 kernel: 内核代码 flash: 其他配置文件 state 升级状态指示，由 NLUpdateState 中定义 percent 指示每个 state 状态下的完成百分比
接口方法	
boolean nl_OpenDevice (Context context, NLUsbListener listener);	USB 设备打开接口，调用前必须保证拥有对 USB 设备节点的读写权限。当需要打开的设备是 DEV_COMPOSITE 时,会通知设备进行通讯口的切换。 参数说明： context: 安卓 context 用于枚举设备 listener: 系统 USB 事件监听器 返回值： true: 成功 false: 失败

boolean nl_OpenDevice(String pathName, int baudrate, NLUartListener listener);	<p>串口设备打开接口</p> <p>参数说明：</p> <p>pathName 串口设备名 如： /dev/ttyS0</p> <p>baudrate 串口波特率</p> <p>listener 串口接收事件监听</p> <p>返回值：</p> <p>true：成功 false：失败</p>
NLCommStream nl_GetDevObj();	<p>应用通过返回流对象进行分析打开的设备流类型</p> <p>参数说明：</p> <p>无</p> <p>返回值：</p> <p>返回流创建的流对象</p>
String nl_GetSdkVersion();	<p>获取 SDK 版本号</p> <p>参数说明：</p> <p>无</p> <p>返回值：</p> <p>SDK 版本号</p>
void nl_CloseDevice();	<p>关闭设备</p> <p>参数说明： 无</p> <p>返回值： 无</p>
boolean nl_DeviceIsOpen();	<p>判断设备是否打开</p> <p>参数说明： 无</p> <p>返回值：</p> <p>true：打开 false：关闭</p>
boolean nl_GetDevStatus();	<p>判断设备是否正常(特定设备支持,详情参见用户手册)</p> <p>参数说明： 无</p> <p>返回值：</p> <p>true：工作正常 false：工作异常</p>
String nl_GetDeviceInfo();	<p>获取设备信息</p> <p>参数说明： 无</p> <p>返回值：</p> <p>统一指令集 QRYSYS 指令的返回结果</p>
boolean nl_StartScan();	<p>发送默认自定义触发指令（0x10 0x54 0x04）开始读码，由于该指令须启用配置，在使用前请确认串行触发指令已打开（SCNTCE1）。触发模式下有效。</p> <p>参数说明： 无</p> <p>返回值：</p> <p>true：读码指令发送成功 false：指令发送失败</p>
boolean nl_StopScan();	<p>停止读码，触发模式下有效。当设备接收到触发读码指令在超时时间内没有达到码词，可以通过该接口停止设备解码。</p> <p>参数说明：</p> <p>返回值：</p>

	true: 指令发送成功 false: 指令发送失败
<code>boolean nl_RestartDevice();</code>	重启设备 参数说明: 无 返回值: true: 指令发送成功 false: 指令发送失败
<code>boolean nl_SendCommand(String command);</code>	发送单条设置指令 例子: <code>setConfig("128ENA1")</code> 该设置码掉电不保存; 如果需要设置码掉电后依然生效, 可以在命令前增加字符"@ ", 比如 <code>setConfig("@128ENA1")</code> 参数说明: command 统一指令集的设置指令 返回值: true: 指令发送成功 false: 指令发送失败
<code>String nl_ReadDevCfg(String command);</code>	查询统一指令的当前设置, 仅支持单条指令的查询。比如 <code>SCNMOD*</code> 查询返回结果为 <code>SCNMOD0</code> 参数说明: command 统一指令集的获取指令 返回值: 返回当前查询指令的应答
<code>int nl_UpdateKernelDevice (byte[] fireware, updateListner listner);</code>	更新模组头固件, 固件升级包会根据客户的要求包含不同的内容。 参数说明: fireware 固件内容缓冲 listner 监听更新固件的进度监听器 返回值: {class NLError} 中描述的错误类型
<code>int nl_WriteCfgToDev(File f);</code>	更新模组头配置, 设备的配置文件通常包含多条配置信息, 配置在发送到设备后需要较长的执行时间 参数说明: f xml 格式的批量配置文件句柄 返回值: >0:更新成功; <0:更新失败 ; =0 更新成功, 且进行了端口切换
<code>int[] nl_GetPicSize();</code>	获取设备当前图像的长度信息 (长、宽) 参数说明: 无 返回值: 设备当前图像的宽 (int[0]) 高 (int[1])
<code>boolean nl_GetPicData (byte[] ImgBuff, int imgSize, transImgListner listner);</code>	遵循统一指令集获取图像的方法, 获取设备当前图像, 目前该接口仅支持获取原始大小、bmp 格式的图片 参数说明: ImgBuff 接收获取到的图像缓存 imgSize 图像缓存的大小 返回值: true: 图像获取成功 false: 图像获取失败