Hopeland RFID Reader

Development Guide

Android

(Android version)

V1.11

Contents

1.Summary

1.1 Summary of content

To facilitate the secondary development of users, we provide a library of functions that can be run on the Android platform. The library is written in the JAVA language and packaged into a standard JAR package with a development environment of JDK1.8.

For secondary development of users, please refer to the development guide of the “Hopeland RFID Device Development - PC - JAVA_2.8”. The application development guide will comprehensively introduce the corresponding technical indicators, application development instructions and notes, application interface function instructions and so on.

This document only adds to the differences in the development guide.

1.2 Copyright notice

2.API Function call (supplementary note)

2.1 Create a serial connection

Package	com.rfidread.RFIDReader
Function	**static boolean CreateSerialConn(string seriaParam, IAsynchronousMessage log)**
Parameter	seriaParam: serial port connection Parameter,eg:” /dev/ttySAC1:115200” log: Data callback interface, all tags data will be called back from this interface.
Return	True: successful; false: failed
Remark	1.The connection established by this method, ”seriaParam” The ID that is the connection channel is distinguished from the other link channel, In the Android system, the serial port is described as a node similar to "/dev/ttysac1". 2. **log** Data callback interface, for specific please refer to Callback interface description

2.2 Create a USB connection

USB to serial cable only
Connection diagram is Android device (supports OTG) → USB to serial cable → Reader serial port

2.2.1 Enumerate USB

Package	com.rfidread.RFIDReader
Function	**static List<UsbSerialPort> GetUSBList(UsbManager mUsbManager)**
Parameter	1. mUsbManager: Manager, allows you to enumerate USB. 2.Access to system services through Context, such as:UsbManager mUsbManager=(UsbManager)getSystemService(Context.USB_SERVICE);
Return	1. Returns a list of USB devices. 2. UsbSerialPort: Interface for a single serial port.
Remark	1. This method will close all the created connections. 2. **import** com.clou.usbserial.driver.UsbSerialPort; **import** android.hardware.usb.UsbManager; 3. See the sample code UHFBaseActivity. Java invocation method.

2.2.2 Get the USB description list (connection parameter)

Package	com.rfidread.RFIDReader
Function	**static List<String> GetUsbDeviceStrList(List<UsbSerialPort> portList)**
Parameter	portList: USB device list
Return	Usb device description list, connection parameter. Such as: {/dev/bus/usb/001/028#VID_0403 PID_6001}
Remark	

2.2.3 Create USB connection

Package	com.rfidread.RFIDReader
Function	**static boolean CreateUsbConn(string usbParam, IAsynchronousMessage log)**
Parameter	usbParam: usb connection parameter. Such as:"{/dev/bus/usb/001/028#VID_0403 PID_6001}" Log: Data callback interface, all tag data will be callback from the interface object.
Return	True: successful; false: failed
Remark	1. Through the connection established by this method, "usbParam" must be enumerated by calling the GetUsbDeviceStrList () and the GetUSBList () function. 2. **log** Data callback interface, for specific please refer to Callback interface description

2.2.4 Apply for USB permission

Package	com.rfidread.RFIDReader
Function	**static boolean GetUsbPermission(Context context, String usbParam)**
Parameter	usbParam: usb connection parameter ,e.g."{/dev/bus/usb/001/028#VID_0403 PID_6001}" context: interface context
Return	True: successful; false: failed
Remark	1. "usbParam" must be enumerated by calling the GetUsbDeviceStrList () and the GetUSBList () function. 2. The application for USB permission needs to be called in the non-main thread, otherwise it will cause the program to crash.

2.2.5 USB permission

Note: Add USB permission to the androidmanife.xml, and the specified type of device automatically gets permission.

Otherwise, requestPermission is required to obtain permission before operation.

```
<uses-feature android:name="android.hardware.usb.host" />

 <intent-filter>
             <action
android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED" />
          </intent-filter>

          <meta-data

android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED"
             android:resource="@xml/device_filter" />
```

2.3 Create a Bluetooth connection

2.3.1 Open Bluetooth

Package	com.rfidread.RFIDReader
Function	**static void OpenBluetooth()**
Parameter	None
Return	None
Remark	Open Bluetooth

2.3.2 Get the Bluetooth description list (connection parameter)

Package	com.rfidread.RFIDReader
Function	**static List<String> GetBT4DeviceStrList()**
Parameter	None
Return	Bluetooth description list , connection parameter, e.g.:"7202K8_080003"
Remark	

2.3.3 Create a Bluetooth connection

Package	com.rfidread.RFIDReader
Function	**static boolean CreateBT4Conn(String bT4Param,IAsynchronousMessage log)**
Parameter	bT4Param: Bluetooth connection parameter, e.g." 7202K8_080003" log: Data callback interface, all tag data will be callback from the interface object.
Return	True: successful; false: failed
Remark	1. The connection established by this method, " **bT4Param** " must be enumerated by calling the **GetBT4DeviceStrList()** function. 2. log Data callback interface, for specific please refer to Callback interface description

2.4 Special interfaces

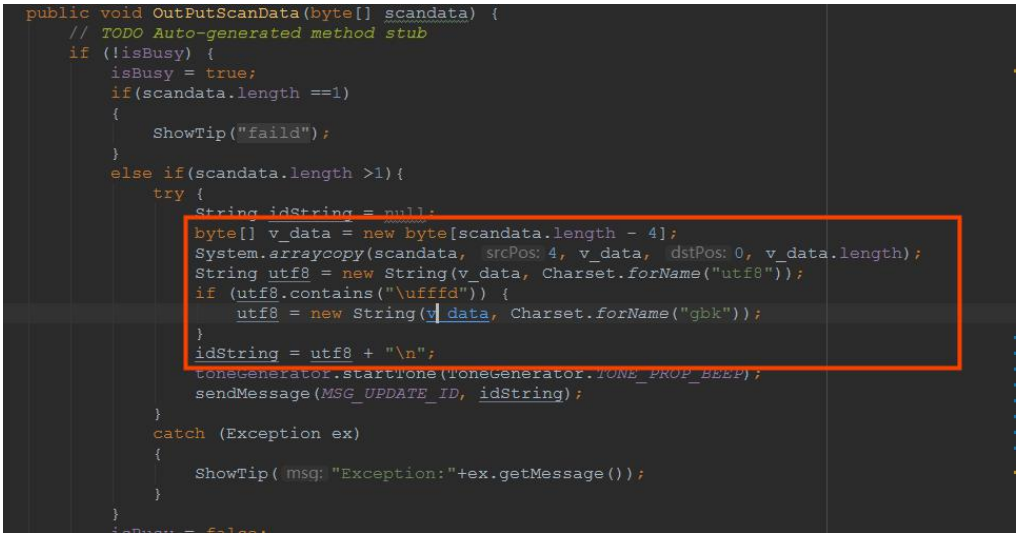
Currently only used for HL7202K8 series

2.4.1 Query battery power

Package	com.rfidread.RFIDReader
Function	**static String GetBluetoothDeviceSOC(String connID)**
Parameter	connID, connection identification
Return	return battery power
Remark	0-100

2.4.2 Start Bluetooth Device Scan

Package	com.rfidread.RFIDReader
Function	**static String StartBluetoothDeviceScan(String connID)**
Parameter	connID, connection identification
Return	return the barcode information **the first 4 bytes is not barcode data, the first byte is scan result(1 success, 0**

	failed), the second byte is fixed as 01, the third and fourth bytes is the data length
Remark	

```
public void OutPutScanData(byte[] scandata) {
    // TODO Auto-generated method stub
    if (!isBusy) {
        isBusy = true;
        if(scandata.length ==1)
        {
            ShowTip("faild");
        }
        else if(scandata.length >1){
            try {
                String idString = null;
                byte[] v_data = new byte[scandata.length - 4];
                System.arraycopy(scandata, srcPos: 4, v_data, dstPos: 0, v_data.length);
                String utf8 = new String(v_data, Charset.forName("utf8"));
                if (utf8.contains("\ufffd")) {
                    utf8 = new String(v_data, Charset.forName("gbk"));
                }
                idString = utf8 + "\n";
                toneGenerator.startTone(ToneGenerator.TONE_PROP_BEEP);
                sendMessage(MSG_UPDATE_ID, idString);
            }
            catch (Exception ex)
            {
                ShowTip( msg: "Exception:"+ex.getMessage());
            }
        }
```

2.4.3 Set beep sound switch of Bluetooth device

Package	com.rfidread.RFIDReader
Function	**static String SetBeep(String connID,int onORoff)**
Parameter	connID, connection identification onORoff, 1 means on, 0 means off
Return	0 success, others failed
Remark	

3.Notice

3.1 TCP communication mode

When the APP using TCP connection, if throw android.os.NetworkOnMainThreadException abnormalities, mean to say: network anomalies in the main thread. An APP that requests a network operation in the main thread will throw this exception. The Android design is designed to prevent web requests from being too long and causing the interface to die.

The solution is to use StrictMode

Add the following code to the MainActivity file's setContentView(R.layout. Activity_main).

```
    if (android.os.Build.VERSION.SDK_INT > 9) {
            StrictMode.ThreadPolicy policy = new
StrictMode.ThreadPolicy.Builder().permitAll().build();
            StrictMode.setThreadPolicy(policy);
        }
```