Android Simple Config Programming Guide

Version

Version	Data	Digest
0.1	2014-02-20	Introduce *Simple Config* API and working flow.
0.2	2014-03-03	Add the usage description of third-party libraries(including Wi-Fi connection, QRCode scanning, screen slipping).
0.3	2014-03-12	Add descriptions of the JNI library(.so).
0.4	2014-07-15	Add descriptions of the delay about sending packet.
0.5	2014-12-05	Add descriptions of using action bar. Add description about the new structure of library: moved configure flow from Java to C, support new mode etc.
0.6	2014-12-18	Refined.

Contents

1. The application project

The *Simple Config* application project is named *SimpleConfigApp*, it's structure is shown as *Figure 1-1*.

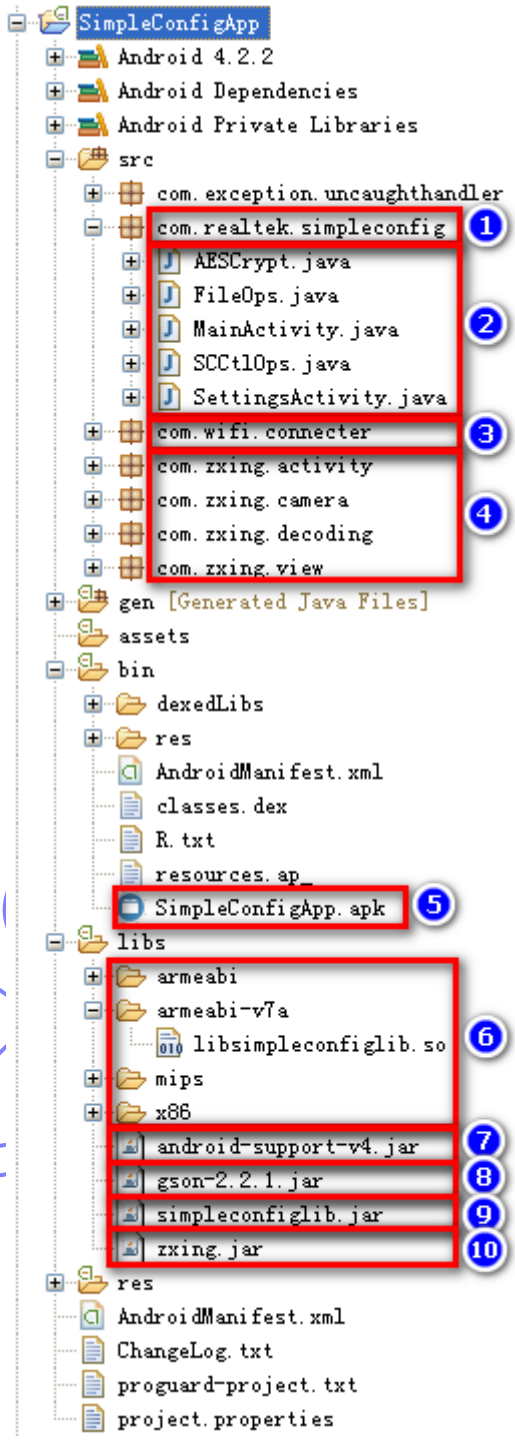

Figure 1-1 Application Structure

Description:

① The main package name;

② *Simple Config* application source code;

③ Wi-Fi connection source code;

④ QRCode scanning source code;

⑤ The compiled *apk* file;
⑥ *Simple config* JNI library;
⑦ Library for screen slipping, etc;
⑧ Library for Json data format;
⑨ *Simple config* Java library;
⑩ Library for QRCode scanning.

2. Application source code

The function of each application source code file(② in *Figure 1-1)* is shown as *Table 2-1*.

Table 2-1 Functions of application source code

File name	Description
AESCrypt.java	The class that implement MD5 and AES encryption algorithm used to store SSID/Password/PIN in file on phone.
FileOps.java	The class that implement file operations: create, delete, open, read, write, close, etc.
MainActivity.java	The main demonstration Activity that calls all the other classes and libraries(described in section 3~6) and implement a UI interface to accomplish *Simple Config* function.
SCCtlOps.java	The class that implement methods to control devices: reset, generate discovery packet, generate control packet, handle discovery ACK, get discovered devices's information, etc.
SettingsActivity.java	Activity used to set configure parameters.

Note:
The detailed *Simple Config* operations is described in section 8.

3. Simple config library

This part introduces data types and APIs of *Simple Config Library* for Android.

3.1 Description

Simple Config JNI Library is named: *libsimpleconfiglib.so*(contained in ⑥ of *Figure 1-1)*, it support ARM, MIPS, x86 platforms.

Simple Config Java Library for Android is named: *simpleconfiglib.jar*(⑨ of *Figure 1-1)*, it supplies developers with external APIs for *Simple Config* further development.

Copy *libsimpleconfiglib.so*(and its parent folder) and *simpleconfiglib.jar* to the directory *libs* of your android project, then you can call the API described in section 3.2.

3.2 External Java API

Simple Config Library supplies these Java APIs to developers:

Table 3-1 Variables of Java API

Variables		
Name	**Data type**	**Description**
TreadMsgHandler	android.os.Handler	Message Handler that ued to receive message from library. Message types: config success, config over but not success, scan devices success, rename device success, delete profile success, etc.
TotalConfigTimeMs	static int	Profile(SSID+PASSWORD, contain many packets) sending total time(ms). Default: 120000 ms (2 minutes)
OldModeConfigTimeMs	static int	Configuring by using old mode(0~ TotalConfigTimeMs) before new mode(the remaining time) Default: 30000ms(30s)
ProfileSendRounds	static int	Profile continuous sending rounds. Default: 1
ProfileSendTimeIntervalMs	static int	Time interval(ms) between sending two rounds of profiles. Default: 1000ms
PacketSendTimeIntervalMs	static int	Time interval(ms) between sending two packets. Default: 0ms
EachPacketSendCounts	static int	The count to send each packet of a profile . Default: 1, Bigger than 1 is used for transfer reliability.

Table 3-2 Functions of Java API

Functions	
Name	**Description**
void WifiInit(Context);	Encapsulated function of WifiManager to initiate Wi-Fi network.
void WifiOpen();	Open a Wi-Fi network.
int WifiStatus();	Get Wi-Fi status.
void WifiStartScan();	Start to scan the Wi-Fi network around.
List<ScanResult> WifiGetScanResults();	Get the scan results and store them in a String list.
boolean isWifiConnected(String);	Determing if a Wi-Fi network of the specified SSID(in the format of String) is connected.
String getConnectedWifiSSID();	Get the SSID of the current connected Wi-Fi network, and return a String.
int WifiGetIpInt();	Get IP address of the phone allocated from a connected Wi-Fi network in the format of integer.
String WifiGetMacStr()	Get MAC address of the phone in the format of string.
void rtk_sc_init();	Initiate the *simple config* operation.
void rtk_sc_exit();	Exit the *Simple config* progress.
void rtk_sc_reset();	Reset *simple config* status.
void rtk_sc_set_ssid(String);	Set the SSID of a Wi-Fi network to generate profile.
void rtk_sc_set_password(String);	Set the password (String) of a Wi-Fi network to generate profile.
void rtk_sc_set_default_pin(String);	Set the default PIN code(String) to generate profile. (If not using the user input PIN code)
String rtk_sc_get_default_pin();	Get the default PIN code.
void rtk_sc_set_pin(String);	User input PIN code(String) to generate profile.
void rtk_sc_set_ip(int);	Set the IP address got from a Wi-Fi network to generate profile. *Deprecated: not needed any more.*
void rtk_sc_build_profile();	Build profile for *simple config*. *Deprecated: not needed any more.*
void rtk_sc_start();	Generate profile and start the *simple config* progress. It is an interface to JNI.
void rtk_sc_stop();	Stop the *simple config* progress. It is an interface to JNI.
int rtk_sc_get_connected_sta_num();	Get connected device number in the configuration progress.
int rtk_sc_get_connected_sta_info (List<HashMap<String, Object>>);	Get connected devices's detailed information in the configuration progress.
int rtk_sc_send_discover_packet(byte[], String);	Send discover packet(byte[]) to a specified IP(String) to discover configured devices.
int rtk_sc_send_control_packet(byte[], String);	Send control packet(byte[]) to a specified IP(String). Control type: delete profile, rename device.

4. Wi-Fi connection library

The Wi-Fi connection library is provided as source code(③ in *Figure 1-1)*. It provide functions for Wi-Fi configuration and connection, and popup a dialog to user.

How to use:

```
final Intent intent = new Intent("com.wifi.connecter.CONNECT_OR_EDIT");
intent .putExtra("com.wifi.connecter.HOTSPOT", hotspot);
activity.startActivity(intent);
```

"com.wifi.connecter.CONNECT_OR_EDIT" must be declared in *AndroidManifest.xml* that in the root folder of the application project(*Figure 4-1*).

```
<activity android:name="com.wifi.connecter.MainActivity"
    android:theme="@android:style/Theme.Dialog"
    android:launchMode="singleInstance"
    android:excludeFromRecents="true"
    android:noHistory="true">
    <intent-filter>
        <category android:name="android.intent.category.INFO" />
    </intent-filter>
    <intent-filter>
        <action android:name="com.wifi.connecter.CONNECT_OR_EDIT" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

Figure 4-1 Application Structure

"com.wifi.connecter.HOTSPOT" is declared in *MainActivity.java* that in the package *com.wifi.connecter.*

Extra, permission must be added to the *AndroidManifest.xml* file(*Figure 4-2*).

```
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE"></uses-permission>
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE"></uses-permission>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"></uses-permission>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"></uses-permission>
<uses-permission android:name="android.permission.INTERNET" ></uses-permission>
<uses-permission android:name="android.permission.READ_SMS"></uses-permission>
```

Figure 4-2 Wi-Fi access permission

Note:

There are two variables *ConnectedSSID* and *ConnectedPasswd* in *SCCtlOps.java* of the main package will be set by *NewNetworkContent.java* of this library. They respectively store the SSID and password of a connected Wi-Fi network.

5. QRCode scanning library

The QRCode scanning code is both provided as source code(④ in *Figure 1-1)* and *jar* library(⑩ in *Figure 1-1)*. It provide functions to open camera and scan QRCode.

How to use:

```
Intent openCameraIntent = new Intent(MainActivity.this, CaptureActivity.class);
startActivityForResult(openCameraIntent, 0);
```

To obtain the scanning result:

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (resultCode == RESULT_OK) {
        Bundle bundle = data.getExtras();
        String QRCodeScanResult = bundle.getString("result");
    }
}
```

Then the result will be stored in a String.

Extra, camera use permission must be added to the *AndroidManifest.xml* file(*Figure 5-1*).

```
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.FLASHLIGHT" />
<uses-feature android:name="android.hardware.camera" />
<uses-feature android:name="android.hardware.camera.autofocus" />
```

Figure 5-1 Camera use permission

Also, below lines must be added to the <*application/*> section of the *AndroidManifest.xml* file:

```
<activity
    android:name="com.zxing.activity.CaptureActivity"
    android:configChanges="orientation|keyboardHidden"
    android:screenOrientation="portrait"
    android:theme="@android:style/Theme.NoTitleBar.Fullscreen"
    android:windowSoftInputMode="stateAlwaysHidden" >
</activity>
```

6. Action bar library

In the version bigger than v1.3.5, *SimpleConfigApp* use *actionbarsherlock* to create action bar for setting activity.

Firstly, import *actionbarsherlock* library project into eclipse, as shown in *Figure 6-1*. And then build it.

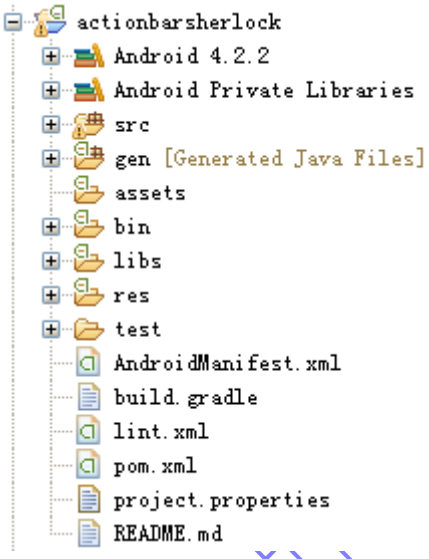

Figure 6-1 actionbarsherlock library

Secondly, in the properties of project *SimpleConfigApp*, add library *actionbarsherlock*, the step is shown in *Figure 6-2*.

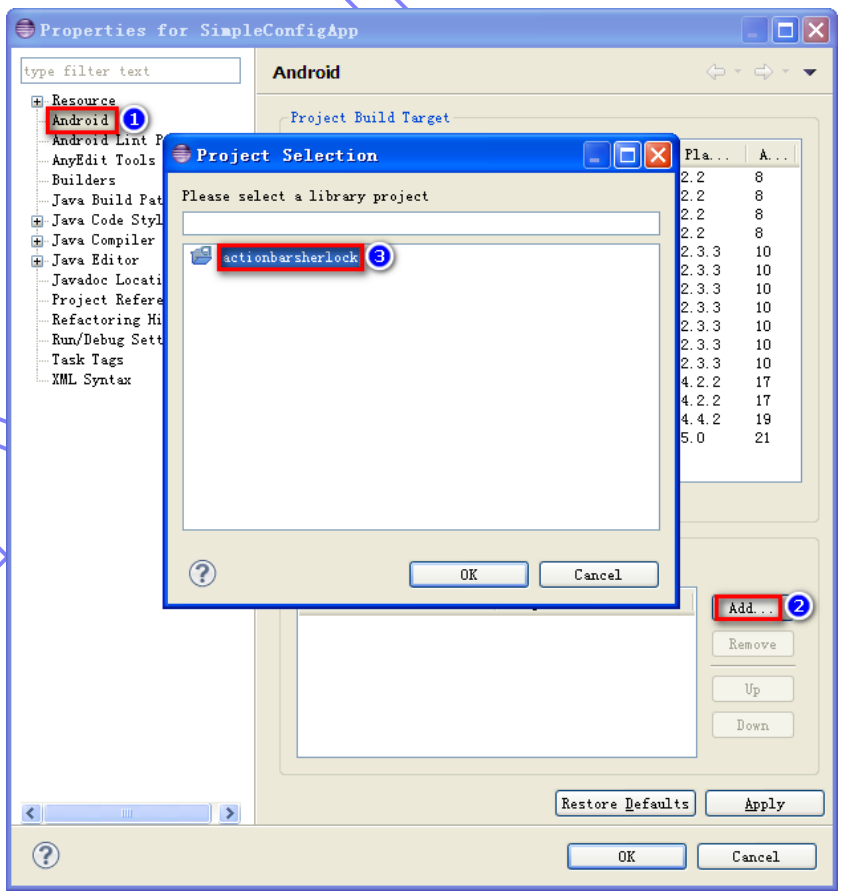

Figure 6-2 Add library

How to use:

In *com.realtek.simpleconfig.MainActivity.java*, make MainActivity extends SherlockActivity instead of Activity, then you can use the action bar provided by *actionbarsherlock*, as shown in *Figure 6-3*.

Figure 6-3 Action Bar

7. Screen slipping library

The screen slipping code is provided as *jar* library(⑦ in *Figure 1-1)*.

How to use:

```
ViewPager SCViewPager;
SCViewPager  = (ViewPager)findViewById(R.id.viewPagerLayout);
SCViewPager.setCurrentItem(0); // default select page 1
SCViewPager.setAdapter(new PageAdpt());
SCViewPager.setOnPageChangeListener(PageChangeEvent);
```

viewPagerLayout is the layout of slipping pages;
PageAdpt is a class that extends PagerAdapter;
PageChangeEvent is a Implementation of OnPageChangeListener.

Detailed usage is in the source code com.realtek.simpleconfig.*MainActivity.java*.

Note:
In newer version(≥v1.3.5), *android-support-v4.jar* is include in *actionbarsherlock*. So no need to include it in *SimpleConfigApp/libs/* any more.

8. Simple config working flow

Simple Config can be used to:

1. Configure a client device;
2. Discover devices;
3. Control devices, include deleting profile and renaming devices.

8.1 Device configuration

The working flow of device configure is shown as Figure 8-1.

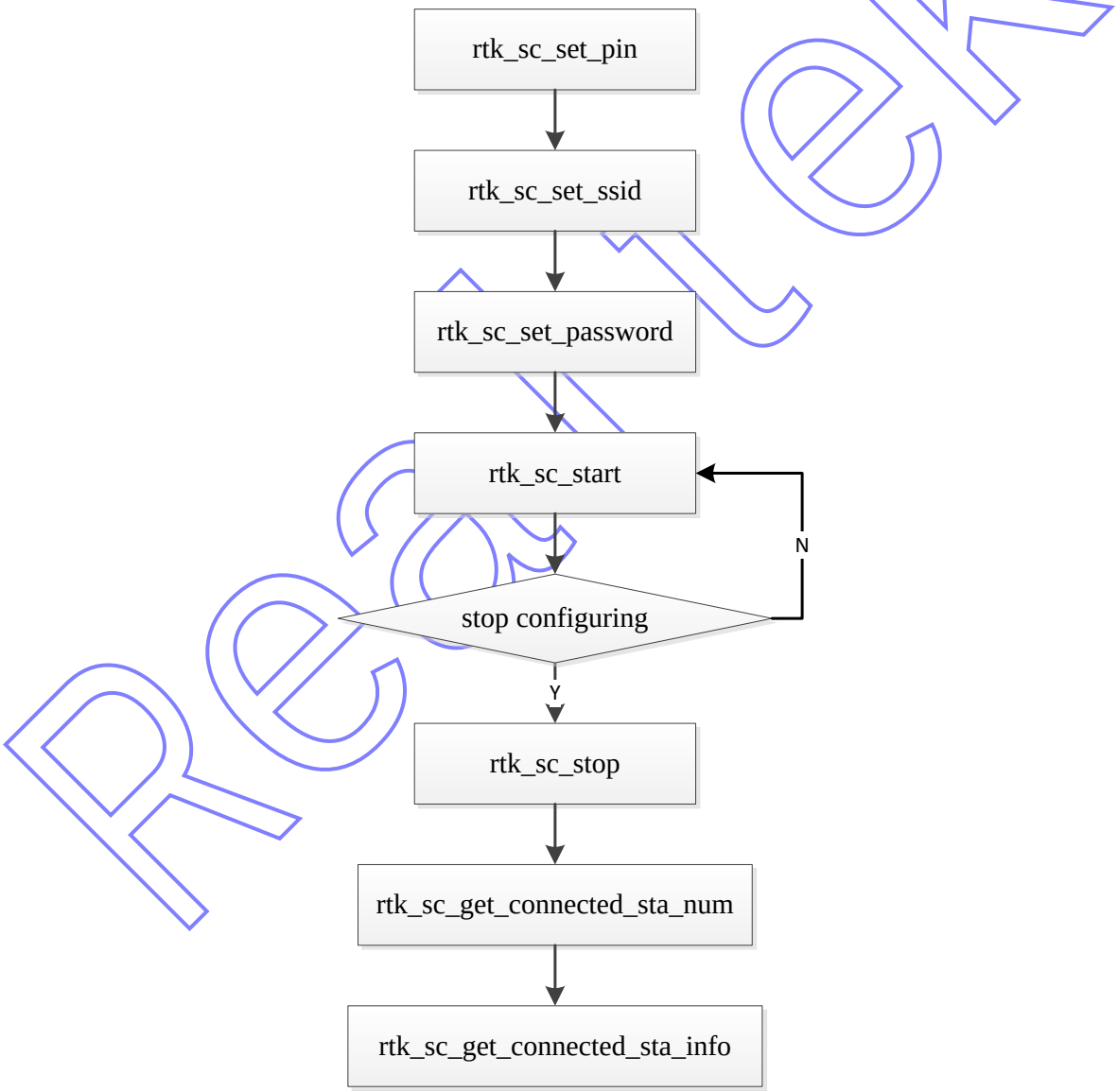

Figure 8-1 Device Configure working flow

Note that it's developer's duty to decide when to send configure packets and when to stop sending. Developers can call API rtk_sc_get_connected_sta_num() to get the connected device number, and call rtk_sc_get_connected_sta_info(List<HashMap<String, Object>>) to get the connected device's information (especially MAC address).

8.2 Device discovery

The working flow of device configure is shown as Figure 8-2.

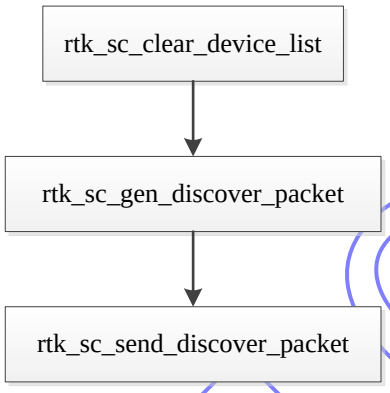

Figure 8-2 Device discovery working flow

Developers can call API rtk_sc_get_discoverd_dev_num() to get the discovered device number, and call rtk_sc_get_discoverd_dev_info(List<HashMap<String, Object>>) to get the discovered device's information.

8.3 Device control

Device control includes two parts: rename device and delete profile of a device. They all need user to input PIN first. Additionally, rename device requires user to input device's new name before renaming.

These general working flow is shown as Figure 8-3.

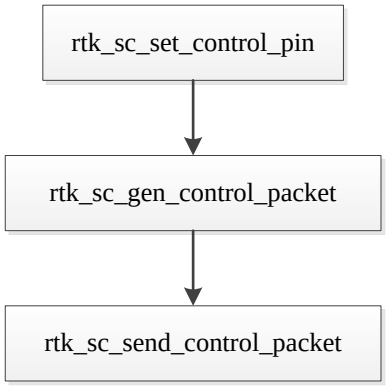

Figure 8-3 Device control working flow