Realtek Android Simple Configure Wizard Guide

Version: 3

1. Introduction

This is the document describes how to use Realtek Android Simple Config Wizard APP to configure WiFi DUT and introduce simple config API.

2. Source Code Location and Description

2.1 Simple Config main application

I. src\com\rtk\simpleconfig_wizard*: main package

src\com\wifi\connection*: Wi-Fi connection

src\com\zxing*: QRCode scanner

2.2 Providing Resources

res\layout: XML files that define a user interface layout

res\drawable: Bitmap files

res\values: XML files that contain simple values, such as strings, integers, and colors.

2.3 Simple Config library

libs\armeabi\libsimpleconfiglib.so: Simple config Java library for armeabi

libs\armeabi-v7a\libsimpleconfiglib.so: Simple config Java library for armeabi-v7a

libs\mips\libsimpleconfiglib.so: Simple config Java library for mips

libs\x86\libsimpleconfiglib.so: Simple config Java library for x86

libs\android-support-v4.jar: Library for screen slipping

libs\gson-2.2.1.jar: Library for JSON

libs\simpleconfiglib.jar: Library for Simple config Java layer

libs\zxing.jar: Library for QR Code scanner

3 Simple Config Library

3.1 Description

Simple Config JNI Library is named: libsimpleconfiglib.so, it supports ARM, MIPS, x86 platforms.

Simple Config Java Library for Android is named: simpleconfiglib.jar, 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 and section 3.3.

3.2 External Java API

3.2.1 Variables of 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 used to receive message from library. Message types: config success, config over but not success, scan devices success, rename device success, and delete profile success.
TotalConfigTimeMs	static int	he time to send a profile each round. Default: 2min (120000ms).
OldModeConfigTimeMs	static int	Configuring by using old mode Default: 0ms
ProfileSendRounds	static int	Profile continuous sending rounds. Default: 1
ProfileSendTimeIntervalMs	static int	Time interval (ms) between sending two rounds of profiles. -The value must be set before rtk_sc_start().
PacketSendTimeIntervalMs	static int	Time interval (ms) between sending two packets. -The value must be set before rtk_sc_start().
EachPacketSendCounts	static int	The count to send each packet of a profile. Default: 1. Bigger than 1 is used for transfer reliability.

3.2.2 Functions of Java API

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();	Gets the Wi-Fi enabled state.
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);	Determine 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 the IP address of a Wi-Fi network in integer format.
java.lang.String WifiGetMacStr();	Get the MAC address of a Wi-Fi network in the format of String.
void rtk_sc_init();	Initiate the simple config operation.
void rtk_sc_exit();	Simple config progress exit.
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_bssid (String);	Set the BSSID 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.
void rtk_sc_build_profile();	Build profile for simple config.
void rtk_sc_start();	Start the simple config progress.
void rtk_sc_stop();	Stop the simple config progress.
int rtk_sc_get_connected_sta_num();	Get connected device number in the configuration progress.
int rtk_sc_get_connected_sta_info	Get connected devices’s detailed information in the

(List<HashMap<String, Object>>);	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.
int SoftAPInit(int,int);	Initiate the simple config operation with soft AP mode.
void SoftAP_ss_close();	Close the simple config operation with soft AP mode.
ScanResult getDUT_by_scanResults();	Get target information by site survey.
String softAP_cmd_send(String SSID, String pwd, String bssid, String user_pinCode, int DUTserverAddress, String softAp_bssid)	Send to target wifi profile with SSID, wifi password, BSSID, PIN code, server ip address of target and BSSID of target.
void rtk_sc_set_deviceNum(int);	Set target number. Default: 1

3.2.3 Flag of UDP Payload Format

Table 3-3 The Flag of UDP Payload Format for Request

Flag Name	Flag Value	Usage
Discover	0x00	To discovery device within request message
SaveProf	0x01	non use
DelProf	0x02	Send to device request message to disconnect AP
RenameDev	0x03	Send to device request message to rename device
ReturnACK	0x04	To confirm remove/rename action for DelProfACK and RenameDevACK message

Table 3-4 The Flag of UDP Payload Format for Response

Flag Name	Flag Value	Usage

CfgSuccessACK	0x00	To check config success from device
DiscoverACK	0x01	To receive discover packet from device
SaveProfACK	0x02	non use
DelProfACK	0x03	To receive remove ACK after phone sent DelProf request message to device
RenameDevACK	0x04	To receive rename ACK after phone sent RenameDev request message to device
CfgSuccessACKSendBack	0x05	To send CfgSuccessACKSendBack message after phone received CfgSuccessACK message.
CfgSuccessACKFinish	0x06	To check config success when configuration finish

3.3 JNI interface

Simpleconfiglib.jar needs libsimpleconfiglib.so to work.

User should add below static code to your main Activity class:

```
static {
        System.loadLibrary("simpleconfiglib");
    }
```

After upper code is executed, the correct libsimpleconfiglib.so of your device's platform is loaded. Then the JNI functions would work properly.

3.4 Wi-Fi connection library

The Wi-Fi connection library is provided as source code. It provides functions for Wi-Fi configuration and connection.

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 3-1*).

```
<activity android:name="com.wifi.connection.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.connection.CONNECT_OR_EDIT" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

Figure 3-1 Application Structure

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

Extra, permission must be added to the *AndroidManifest.xml* file (*Figure 3-2*).

```
<!-- wifi usage -->
<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>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"></uses-permission>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"></uses-permission>
```

Figure 3-2 Wi-Fi access permission

In order to connect hidden SSID, you should call wifi.connectToNewNetwork. to connect the AP.

Note:

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

3.5 QRCode scanning library

The QRCode scanning code is both provided as source code and jar library. It provides functions to open camera and scan QRCode.

How to use:

```
Intent openCameraIntent = new Intent (SCTest.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 3-3*).

```
<!-- Camera usage -->
<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 3-3 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>
```

4 Simple Config Working Flow

Simple Config can be used to:

1. Configure a target device;
2. Discover devices. (option)
3. Control devices, and include deleting profile and renaming devices. (option)

4.1 Device configuration

There are two methods to configure target: soft AP method and normal method.
The soft AP method working flow of device configure is shown as Fig 4-1.

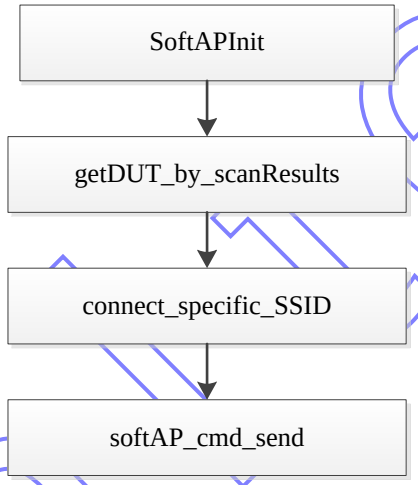

Figure 4-1 Device Configure working flow by soft AP method

The R1 method working flow of device configure is shown as Fig 4-2.

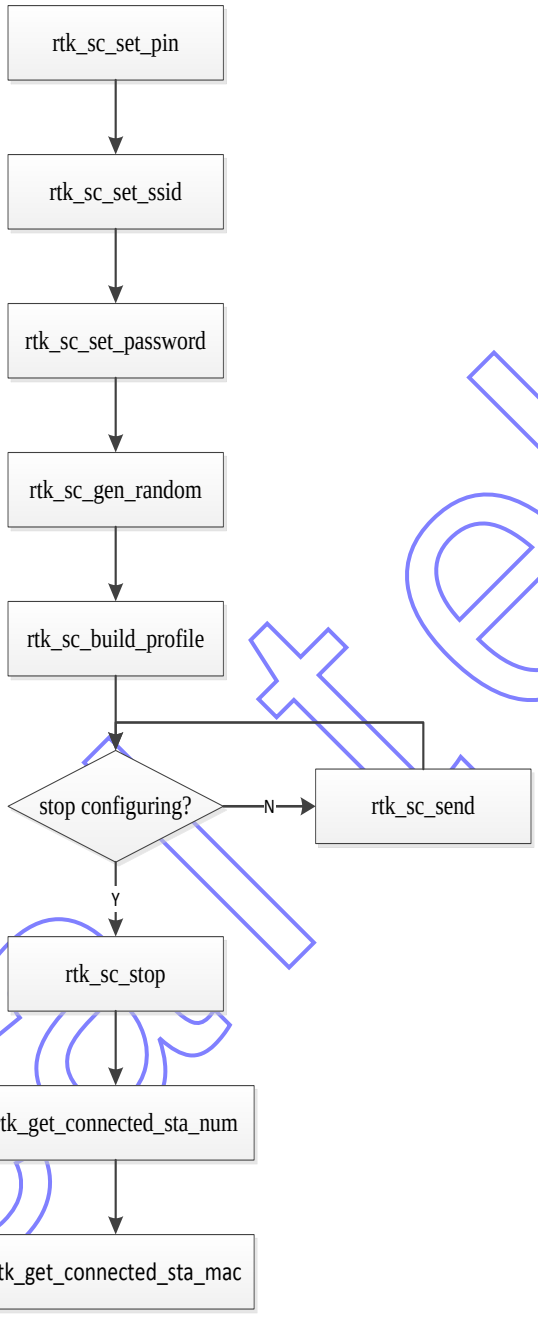

Figure 4-2 Device Configure working flow by normal method

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).

4.2 Device discovery

The working flow of device discovery is shown as Fig 4-3.

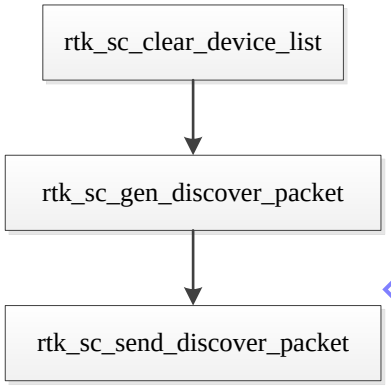

Figure 4-3 Device discovery working flow

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.

4.3 Device control

Device control includes two parts: rename device and delete profile of a device. Rename device requires user to input device's new name before renaming. Delete target profile in order to target disconnect home AP.

This general working flow is shown as Fig 4-4.

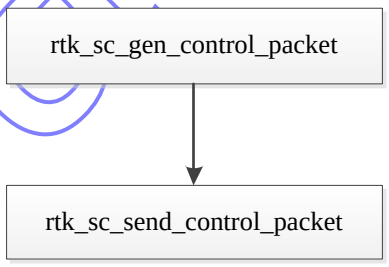

Figure 4-4 Device control working flow

4.4 Device connects to other AP

If the device support new AP profile controller, the operation is enabled. When device is at configured state and device can connect to other AP.

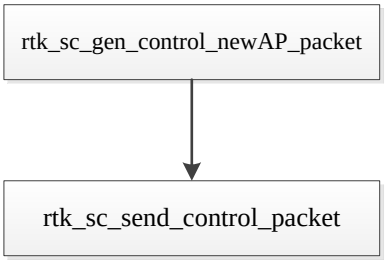

Figure 4-5 Device connects to other AP working flow

5 Reference

N/A