Mobile Computing
And
Wireless Communication

Wifi in Android

Islamic University of Gaza

Dr.Aiman AbuSamra

Managing network connectivity

- Android broadcasts Intents that describe the changes in network connectivity
  - 3G, WiFi, etc.
- There are APIs for controlling network settings and connections
- Android networking is handled by **`ConnectivityManager`** (a network connectivity service)
  - Monitor the state of network connections
  - Configure failover settings
  - Control network radios

Managing your WiFi

- WifiManager: represents the Android WiFi connectivity service
  - Configure WiFi network connections
  - Manage current WiFi connection
  - Scan for access points
  - Monitor changes in WiFi connectivities

Monitoring WiFi connectivity

- Accessing the WiFi Manager

```
String service = Context.WIFI_SERVICE;
WifiManager wifi = (WifiManager) getSystemService(service);
```

```
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
```

- Monitoring and changing Wifi state
  - State: enabling, enabled, disabling, disabled, and

```
if (!wifi.isWifiEnabled())
          if (wifi.getWifiState() != WifiManager.WIFI_STATE_ENABLING)
                    wifi.setWifiEnabled(true);
```

Monitoring WiFi connectivity

- WifiManager broadcasts Intents whenever connectivity status changes
  - WIFI_STATE_CHANGED_ACTION
    - Wifi h/w status has changed: enabling, enabled, disabling, disabled, and unknown
    - EXTRA_WIFI_STATE, EXTRA_PREVIOUS_STATE
  - SUPPLICANT_CONNECTION_CHANGED_ACTION:
    - Whenever connection state with the active supplicant (access point) changes
    - Fired when a new conn is established, or existing conn is lost (EXTRA_NEW_STATE = true/false)
  - NEWTWORK_STATE_CHANGED_ACTION:
    - Fired whenever wifi connectivity state changes
    - EXTRA_NETWORK_INFO: NetworkInfo obj for current network status
    - EXTRA_BSSID: BSSID of the access point that you're connected to
  - RSSI_CHANGED_ACTION:
    - Monitor the signal strength of the connected WiFi network
    - EXTRA_NEW_RSSI: current signal strength

Monitoring active connection details

- Once you connected to an access point, use getConnectionInfo of WifiManager to find info of that connection
  - Returns “WifiInfo” object

```
WifiInfo info = wifi.getConnectionInfo();

if (info.getBSSID() != null) {
     int strength = WifiManager.calculateSignalLevel(info.getRssi(), 5);
     int speed = info.getLinkSpeed();
     String units = WifiInfo.LINK_SPEED_UNITS;
     String ssid = info.getSSID();
     String cSummary = String.format("Connected to %s at %s%s. Strength
     %s/5",
     ssid, speed, units, strength);
}
```

Scanning hotspots

- Use WifiManager to scan access points using startScan()
- Android will broadcast scan results with an Intent of “SCAN_RESULTS_AVAILABLE_ACTION”

```
// Register a broadcast receiver that listens for scan results.
registerReceiver(new BroadcastReceiver() {
     @Override
     public void onReceive(Context context, Intent intent) {
          List<ScanResult> results = wifi.getScanResults();
          ScanResult bestSignal = null;
          for (ScanResult result : results) {
               if (bestSignal == null ||
               WifiManager.compareSignalLevel(bestSignal.level,result.level)<0)
                    bestSignal = result;
          }
          String toastText = String.format("%s networks found. %s is
          the strongest.",
          results.size(), bestSignal.SSID);
          Toast.makeText(getApplicationContext(), toastText, Toast.LENGTH_LONG);
     }
}, new IntentFilter(WifiManager.SCAN_RESULTS_AVAILABLE_ACTION));

// Initiate a scan.
wifi.startScan();
```

Creating a WiFi configuration

- To connect to a WiFi network, a WiFi configuration must be created and registered
  - Normally a user does this, but app can do this
- Network configuration is stored as WifiConfiguration object
  - SSID (service set ID, e.g., IPv4_KAIST)
  - BSSID (MAC addr of an AP)
  - networkId (unique ID that the supplicant uses to identify this network configuration entry)
  - priority (priority of this access point)
  - status (current status: ENABLED, DISABLED, CURRENT)

Creating a WiFi configuration

```
WifiManager wifi = (WifiManager) getSystemService(Context.WIFI_SERVICE);
WifiConfiguration wc = new WifiConfiguration();

wc.SSID = "\"SSIDName\"";
wc.preSharedKey  = "\"password\""; // it should be in double quote “password”
wc.hiddenSSID = true;
wc.status = WifiConfiguration.Status.ENABLED;

// setting up WPA-PSK
wc.allowedGroupCiphers.set(WifiConfiguration.GroupCipher.TKIP);
wc.allowedGroupCiphers.set(WifiConfiguration.GroupCipher.CCMP);
wc.allowedKeyManagement.set(WifiConfiguration.KeyMgmt.WPA_PSK);
wc.allowedPairwiseCiphers.set(WifiConfiguration.PairwiseCipher.TKIP);
wc.allowedPairwiseCiphers.set(WifiConfiguration.PairwiseCipher.CCMP);
wc.allowedProtocols.set(WifiConfiguration.Protocol.RSN);

int res = wifi.addNetwork(wc); // the ID of the newly created network description
Log.d("WifiPreference", "add Network returned " + res );
boolean b = wifi.enableNetwork(res, true);
Log.d("WifiPreference", "enableNetwork returned " + b );
```

http://stackoverflow.com/questions/2140133/how-and-what-to-set-to-android-wificonfiguration-presharedkey-to-connect-to-the-w

http://developer.android.com/reference/android/net/wifi/WifiConfiguration.html#preSharedKey

Managing WiFi configurations

- Use WiFi Manager to manage the configured network settings and control which networks to connect to

```
// Get a list of available configurations
List<WifiConfiguration> configurations = wifi.getConfiguredNetworks();

// Get the network ID for the first one.
if (configurations.size() > 0) {
     int netID = configurations.get(0).networkId;
     // Enable that network.
     boolean disableAllOthers = true;
     wifi.enableNetwork(netID, disableAllOtherstrue);
}
```

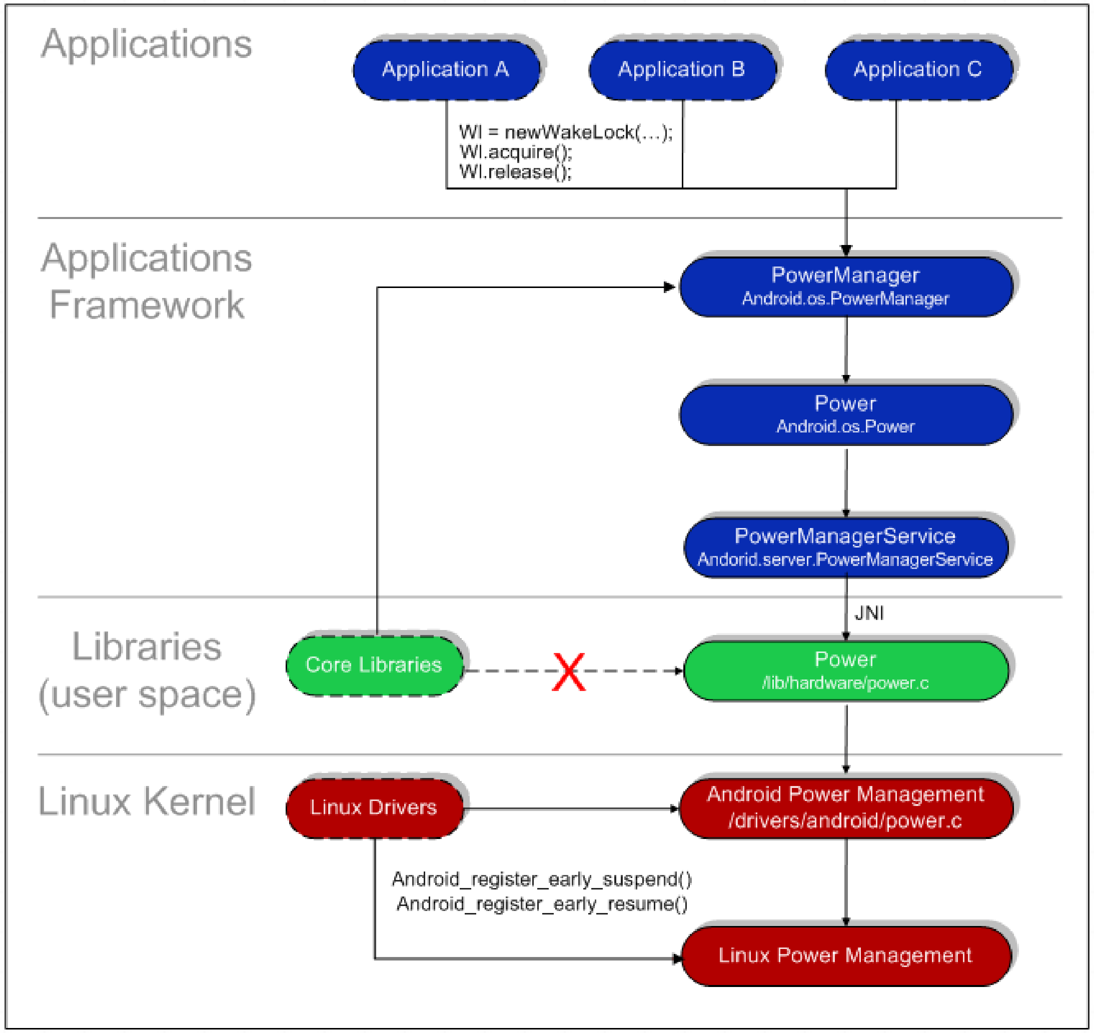
Power Management

- Android supports its own Power Management (on top of the standard Linux Power Management)
  - To make sure that CPU shouldn't consume power if no applications or services require power

- Android requires that applications and services request CPU resources with "wake locks" through the Android application framework and native Linux libraries.

- If there are no active wake locks, Android will shut down the CPU.

Power Management



WakeLock

Flag value	CPU	Screen	Keyboard
PARTIAL_WAKE_LOCK	On	Off	Off
SCREEN_DIM_WAKE_LOCK	On	Dim	Off
SCREEN_BRIGHT_WAKE_LOCK	On	BRIGHT	Off
FULL_WAKE_LOCK	On	Bright	Bright

```
// Acquire handle to the PowerManager service
PowerManager pm = (PowerManager)mContext.getSystemService(
                                    Context.POWER_SERVICE);
// Create a wake lock and specify the power management flags for
screen, timeout, etc.
PowerManager.WakeLock wl = pm.newWakeLock(
         PowerManager.SCREEN_DIM_WAKE_LOCK |
         PowerManager.ON_AFTER_RELEASE, TAG);
// Acquire wake lock
wl.acquire(); // ...
// Release wake lock
wl.release();
```

Wifi background data transfer

- Background data transfer:
  - Wifilock + Wakelock (partial)

```
// http://developer.android.com/reference/android/net/wifi/WifiManager.WifiLock.html
WifiManager.WifiLock wifiLock = null;
PowerManager.WakeLock wakeLock = null;
// acquire
if (wifiLock == null) {
        WifiManager wifiManager = (WifiManager)
        context.getSystemService(context.WIFI_SERVICE);
        wifiLock = wifiManager.createWifiLock("wifilock");
        wifiLock.setReferenceCounted(true);
        wifiLock.acquire();

        PowerManager powerManager = (PowerManager)
        context.getSystemService(context.POWER_SERVICE);
        wakeLock =
        powerManager.newWakeLock(PowerManager.PARTIAL_WAKE_LOCK, "wakelock");
        wakeLock.acquire();

}
// release
if (wifiLock != null) {
        wifiLock.release();
        wifiLock = null;
        wakeLock.release();
        wakeLock = null;
}
```

Background data transfer

- Setting > Accounts & sync settings > background data setting
- If this setting is off, an application cannot transfer data only when it is active and in the foreground
  - Services cannot transfer data (by definition)
- Use connectivity manager to check this:
  - `boolean backgroundEnabled = connectivity.getBackgroundDataSetting();`
- App can listen to changes in the background data transfer preference:

```
registerReceiver(
 new BroadcastReceiver() {
     @Override
     public void onReceive(Context context, Intent intent)
     // do something..
 },
  new IntentFilter(ConnectivityManager.
     ACTION_BACKGROUND_DATA_SERVICE_CHANGED)
 );
```

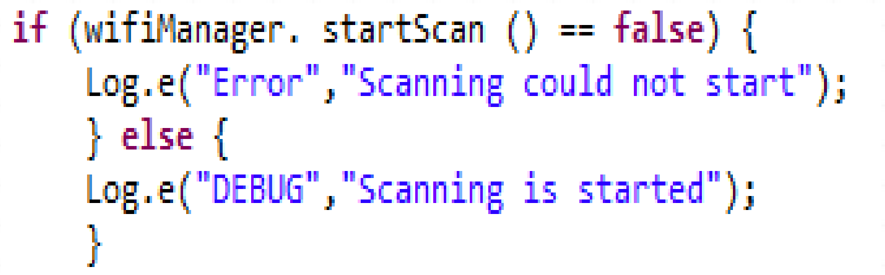
Start WiFi

- New App called WiFiFun
  - Include permissions: access_wifi_state, change_wifi_state, access_network_state, change_network_state, write_settings, write_secure_settings, change_wifi_multicast
- Include member variable
  - WifiManager wifiManager;
- In onCreate, add
  - wifiManager = (WifiManager) getSystemService(Context.*WIFI_SERVICE);*
- Start wifi, add

```
if(wifiManager.isWifiEnabled()==false){
    Log.e("DEBUG","turning on wifi");
    wifiManager.setWifiEnabled(true);
    } else {
    Log.e("DEBUG","wifi is on");
    }

    switch (wifiManager.getWifiState()) {
    case WifiManager.WIFI_STATE_DISABLED: Log.e("DEBUG","wifi state is disabled"); break;
    case WifiManager.WIFI_STATE_DISABLING: Log.e("DEBUG","wifi state is WIFI_STATE_DISABLING"); break;
    case WifiManager.WIFI_STATE_ENABLED: Log.e("DEBUG","wifi state is WIFI_STATE_ENABLED"); break;
    case WifiManager.WIFI_STATE_ENABLING: Log.e("DEBUG","wifi state is WIFI_STATE_ENABLING"); break;
    case WifiManager.WIFI_STATE_UNKNOWN: Log.e("DEBUG","wifi state is WIFI_STATE_UNKNOWN"); break;
    }
```

Scanning for access points

- Approach: start wifi scanning, get results in broadcast receiver
- At the end of onCreate, addd

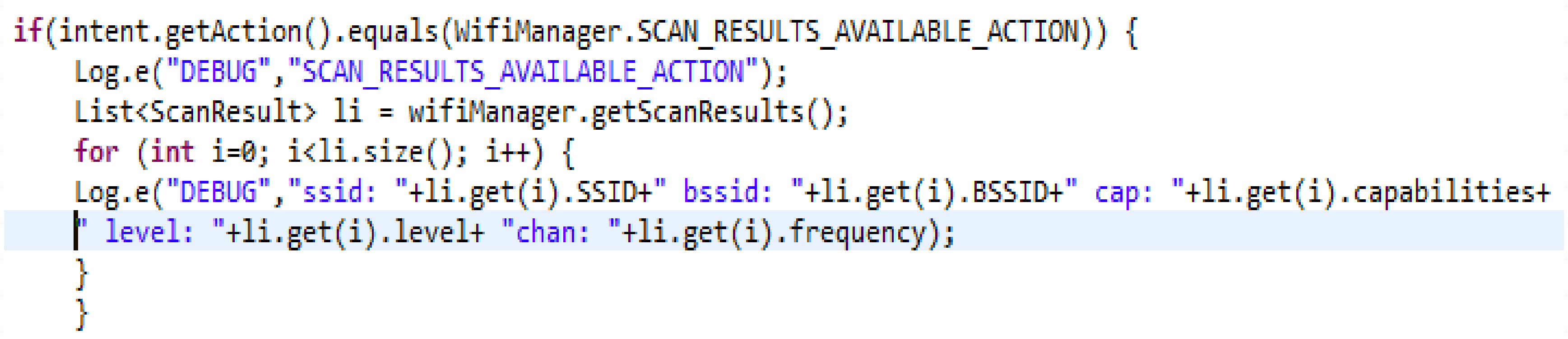


```
if (wifiManager. startScan () == false) {
    Log.e("Error","Scanning could not start");
    } else {
    Log.e("DEBUG","Scanning is started");
    }
```

- Register to receive broadcast about scanning results
  - Add member variable to WiFiFun
    - IntentFilter filter;
  - At the end of onCreate, add
    - filter = new IntentFilter();
    - filter.addAction(WifiManager.SCAN_RESULTS_AVAILABLE_ACTION);
    - registerReceiver(wifiEventReceiver, filter);

- Make broadcast receiver.

- Somewhere in WifiFun, add member variable
  - private BroadcastReceiver wifiEventReceiver = new BroadcastReceiver() {};
  - // let eclipse add unimplemented methods
  - In public void onReceive(Context arg0, Intent intent) {, add

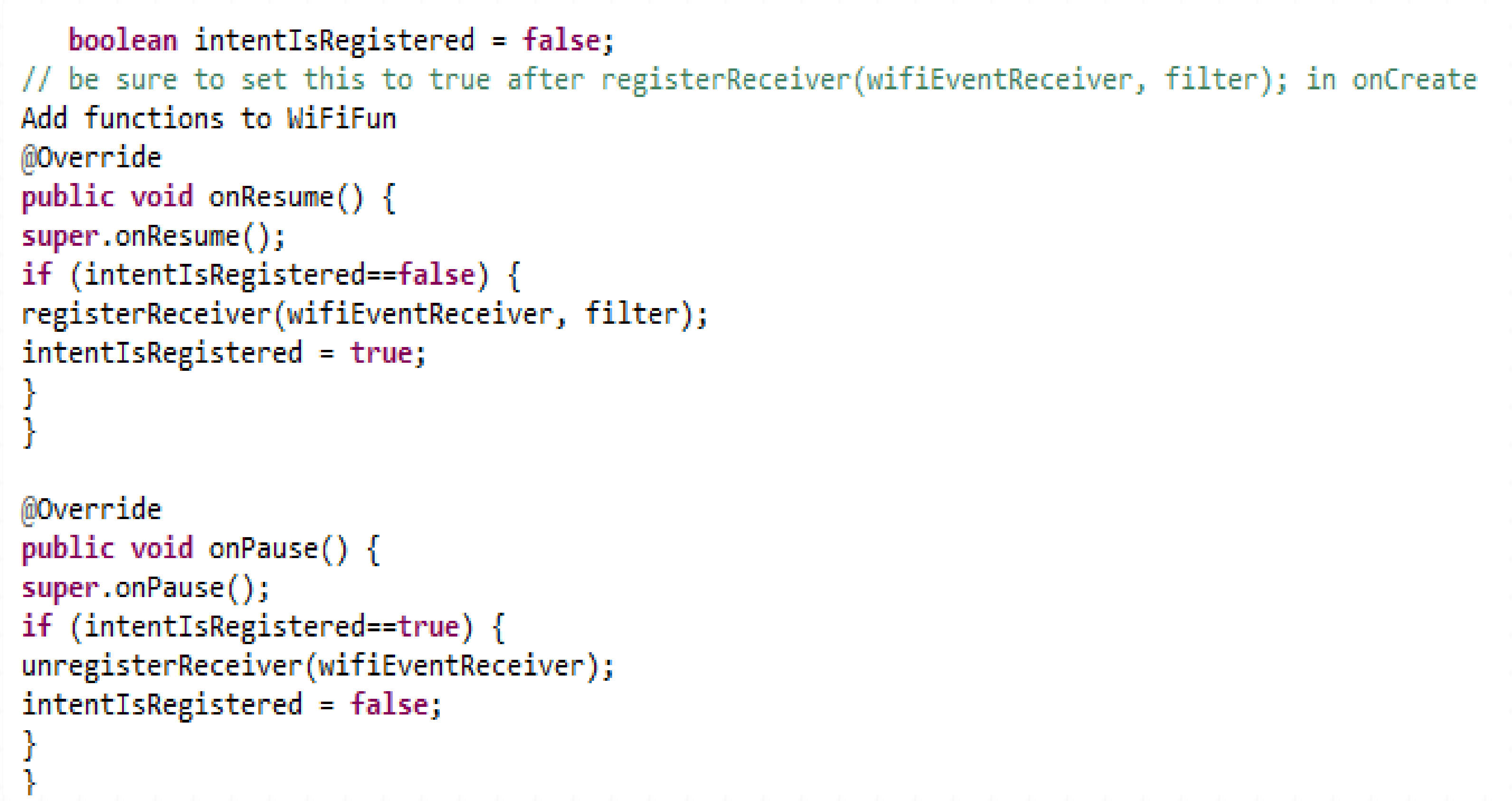


```
if(intent.getAction().equals(WifiManager.SCAN_RESULTS_AVAILABLE_ACTION)) {
    Log.e("DEBUG","SCAN_RESULTS_AVAILABLE_ACTION");
    List<ScanResult> li = wifiManager.getScanResults();
    for (int i=0; i<li.size(); i++) {
    Log.e("DEBUG","ssid: "+li.get(i).SSID+" bssid: "+li.get(i).BSSID+" cap: "+li.get(i).capabilities+
    " level: "+li.get(i).level+ "chan: "+li.get(i).frequency);
    }
    }
```

- Run
  - Try adding wifiManager. startScan () to end of BroadcastReceiver .onReceive
  - Notice that signal strength varies.

Scanning for access points

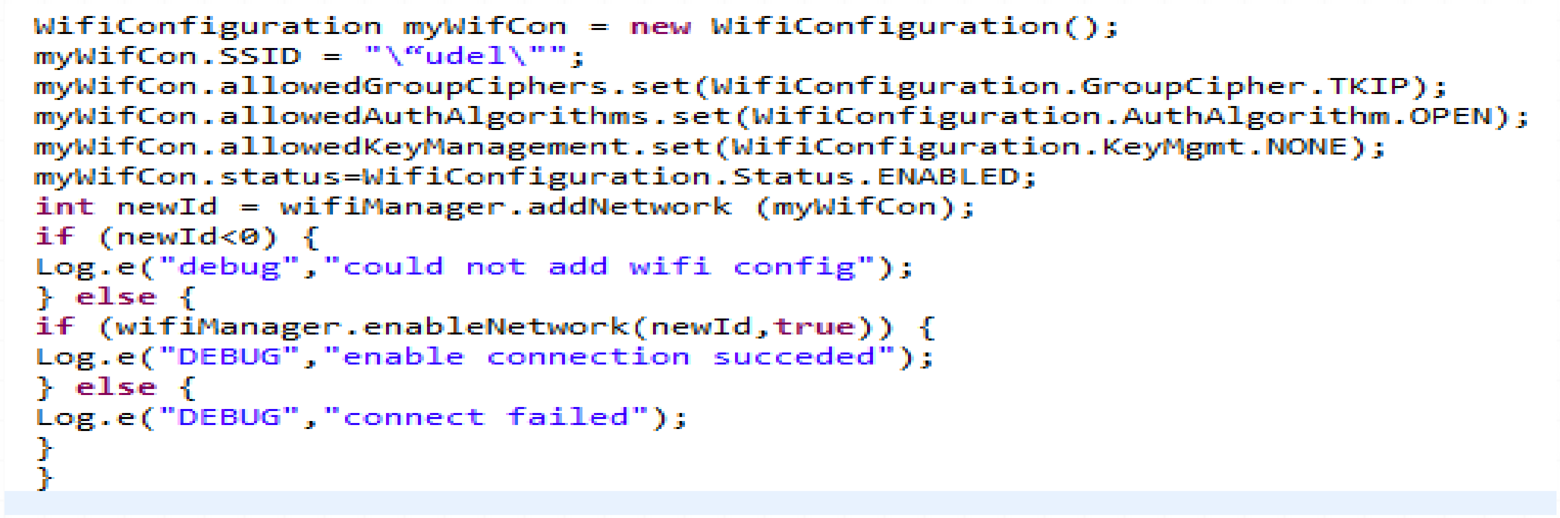
- Clean up BroadcastReceiver: unregister on pause, and regfister on resume. Be careful to not register twice
  - Add member variable to WiFiFun



```
   boolean intentIsRegistered = false;
// be sure to set this to true after registerReceiver(wifiEventReceiver, filter); in onCreate
Add functions to WiFiFun
@Override
public void onResume() {
super.onResume();
if (intentIsRegistered==false) {
registerReceiver(wifiEventReceiver, filter);
intentIsRegistered = true;
}
}

@Override
public void onPause() {
super.onPause();
if (intentIsRegistered==true) {
unregisterReceiver(wifiEventReceiver);
intentIsRegistered = false;
}
}
```

Connect to access point

- Add button
  - Connect to udel wifi
- In onCreate, add
  - Button wifiConnect = (Button)findViewById(R.id.WifiConnect);
  - wifiConnect.setOnClickListener(new View.OnClickListener() {});
  - // let eclipse add onClick
- In onClick add
  - Add new network to current list of networks



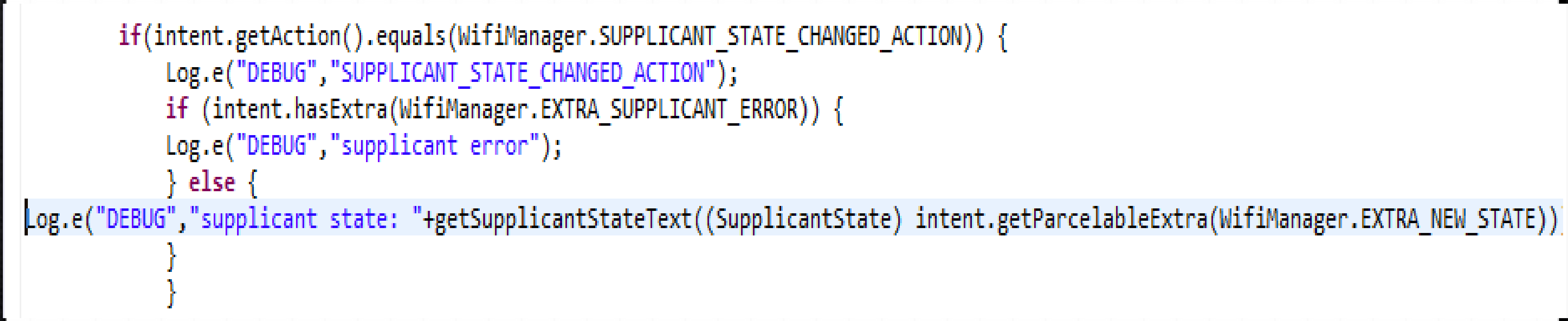
```
WifiConfiguration myWifCon = new WifiConfiguration();
myWifCon.SSID = "\"udel\"";
myWifCon.allowedGroupCiphers.set(WifiConfiguration.GroupCipher.TKIP);
myWifCon.allowedAuthAlgorithms.set(WifiConfiguration.AuthAlgorithm.OPEN);
myWifCon.allowedKeyManagement.set(WifiConfiguration.KeyMgmt.NONE);
myWifCon.status=WifiConfiguration.Status.ENABLED;
int newId = wifiManager.addNetwork (myWifCon);
if (newId<0) {
Log.e("debug","could not add wifi config");
} else {
if (wifiManager.enableNetwork(newId,true)) {
Log.e("DEBUG","enable connection succeded");
} else {
Log.e("DEBUG","connect failed");
}
}
```

- Two things
  - This might not connect, e.g., maybe udel is out of range
  - Perhaps we should not add udel to the list of networks

Add BroadcastReceiver

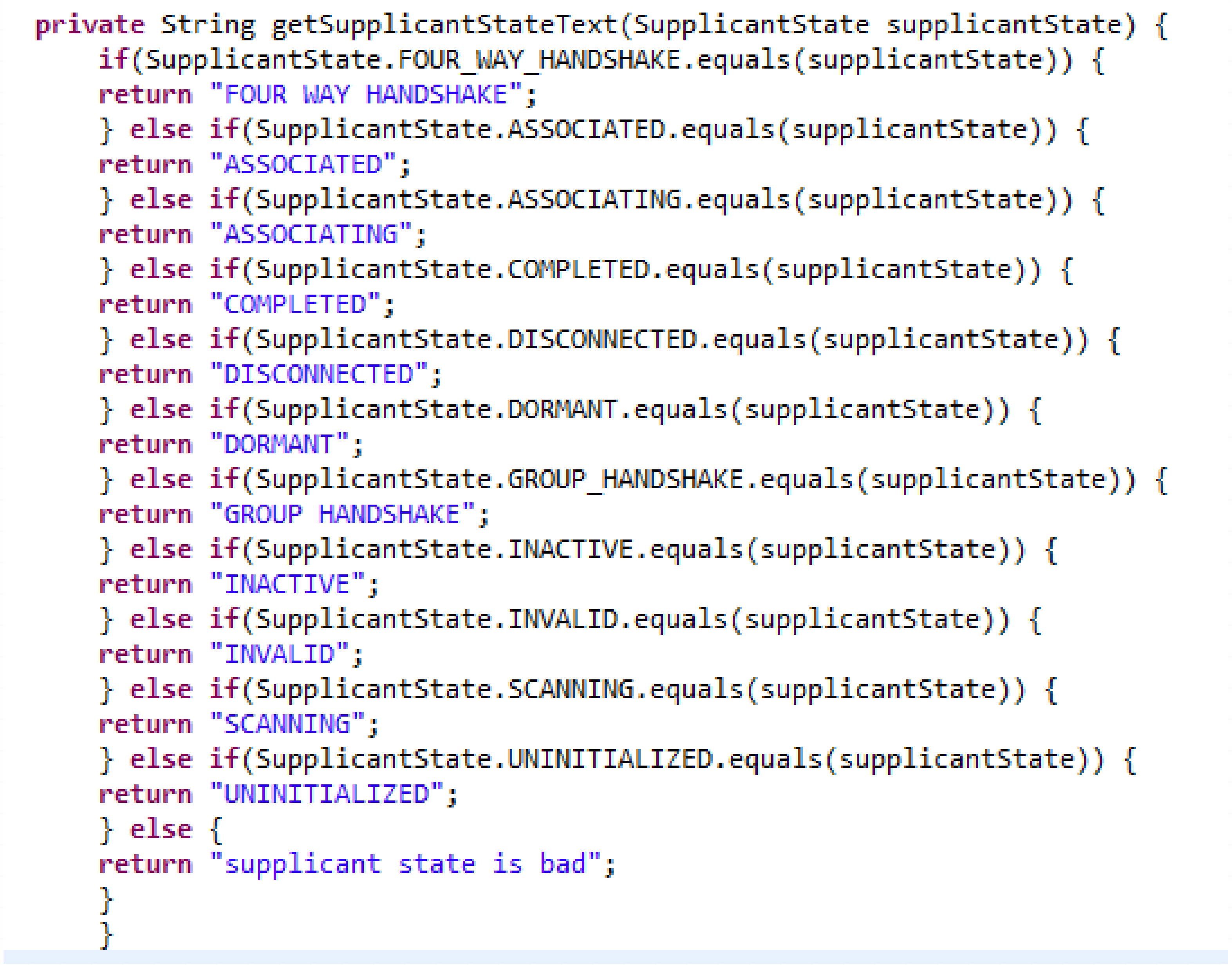
- In onCreate, when the intentFilter is being made, add
  - filter.addAction(WifiManager.SUPPLICANT_CONNECTION_CHANGE_ACTION);
  - filter.addAction(WifiManager.SUPPLICANT_STATE_CHANGED_ACTION);
  - filter.addAction(WifiManager.NETWORK_IDS_CHANGED_ACTION);
  - filter.addAction(WifiManager.NETWORK_STATE_CHANGED_ACTION);
  - filter.addAction(WifiManager.WIFI_STATE_CHANGED_ACTION);

- In the BroadcastReceiver, add

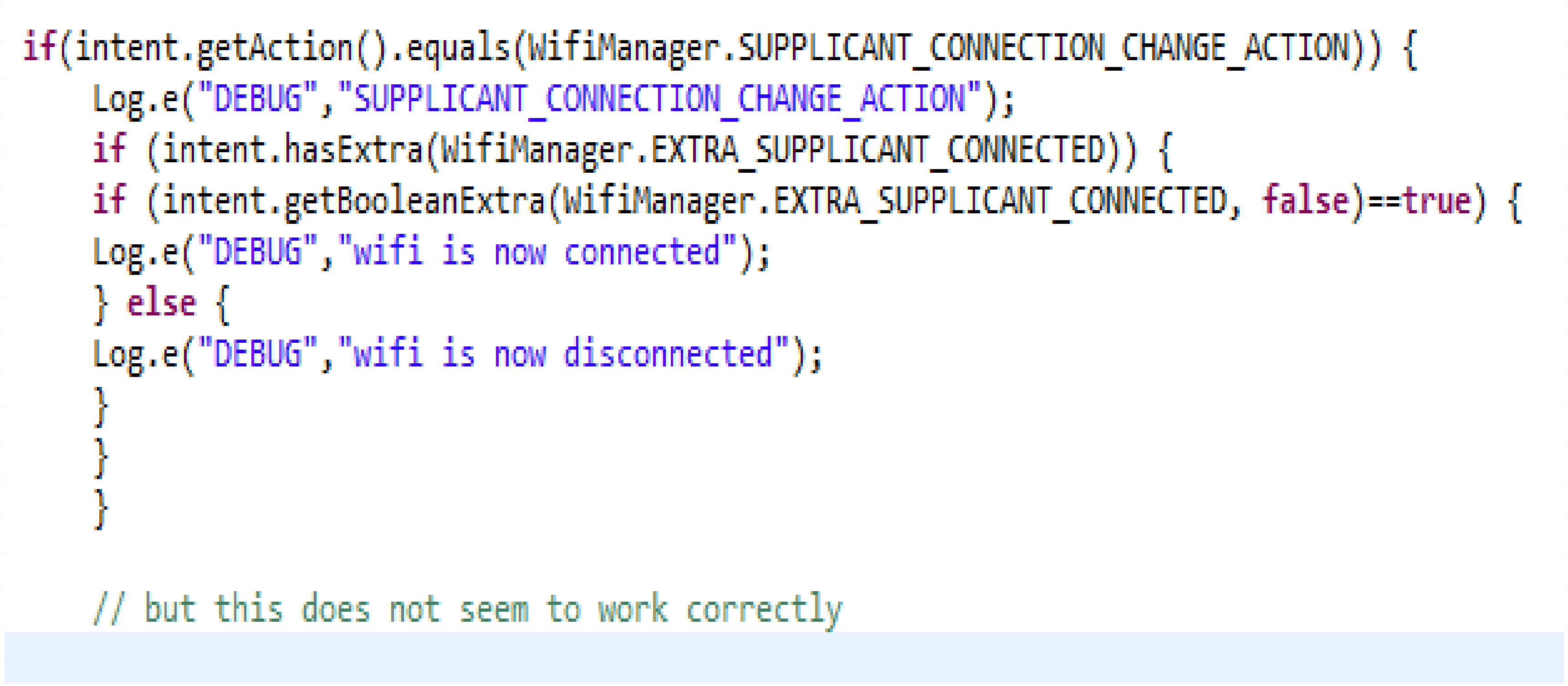


```
        if(intent.getAction().equals(WifiManager.SUPPLICANT_STATE_CHANGED_ACTION)) {
            Log.e("DEBUG","SUPPLICANT_STATE_CHANGED_ACTION");
            if (intent.hasExtra(WifiManager.EXTRA_SUPPLICANT_ERROR)) {
            Log.e("DEBUG","supplicant error");
            } else {
Log.e("DEBUG","supplicant state: "+getSupplicantStateText((SupplicantState) intent.getParcelableExtra(WifiManager.EXTRA_NEW_STATE))
            }
            }
```

❖ Add function

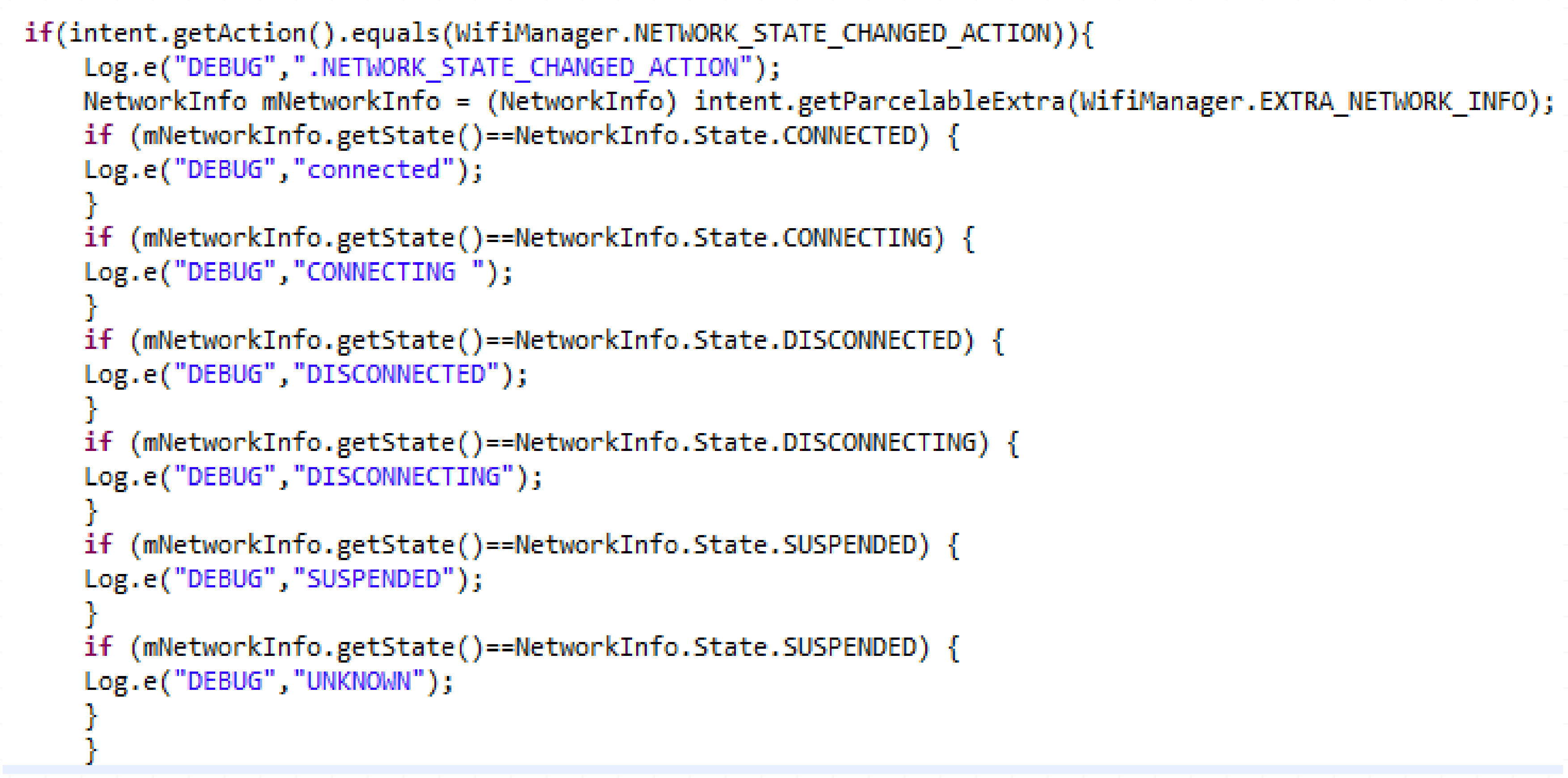


```
private String getSupplicantStateText(SupplicantState supplicantState) {
    if(SupplicantState.FOUR_WAY_HANDSHAKE.equals(supplicantState)) {
    return "FOUR WAY HANDSHAKE";
    } else if(SupplicantState.ASSOCIATED.equals(supplicantState)) {
    return "ASSOCIATED";
    } else if(SupplicantState.ASSOCIATING.equals(supplicantState)) {
    return "ASSOCIATING";
    } else if(SupplicantState.COMPLETED.equals(supplicantState)) {
    return "COMPLETED";
    } else if(SupplicantState.DISCONNECTED.equals(supplicantState)) {
    return "DISCONNECTED";
    } else if(SupplicantState.DORMANT.equals(supplicantState)) {
    return "DORMANT";
    } else if(SupplicantState.GROUP_HANDSHAKE.equals(supplicantState)) {
    return "GROUP HANDSHAKE";
    } else if(SupplicantState.INACTIVE.equals(supplicantState)) {
    return "INACTIVE";
    } else if(SupplicantState.INVALID.equals(supplicantState)) {
    return "INVALID";
    } else if(SupplicantState.SCANNING.equals(supplicantState)) {
    return "SCANNING";
    } else if(SupplicantState.UNINITIALIZED.equals(supplicantState)) {
    return "UNINITIALIZED";
    } else {
    return "supplicant state is bad";
    }
    }
```



```
if(intent.getAction().equals(WifiManager.SUPPLICANT_CONNECTION_CHANGE_ACTION)) {
    Log.e("DEBUG","SUPPLICANT_CONNECTION_CHANGE_ACTION");
    if (intent.hasExtra(WifiManager.EXTRA_SUPPLICANT_CONNECTED)) {
    if (intent.getBooleanExtra(WifiManager.EXTRA_SUPPLICANT_CONNECTED, false)==true) {
    Log.e("DEBUG","wifi is now connected");
    } else {
    Log.e("DEBUG","wifi is now disconnected");
    }
    }
    }

    // but this does not seem to work correctly
```

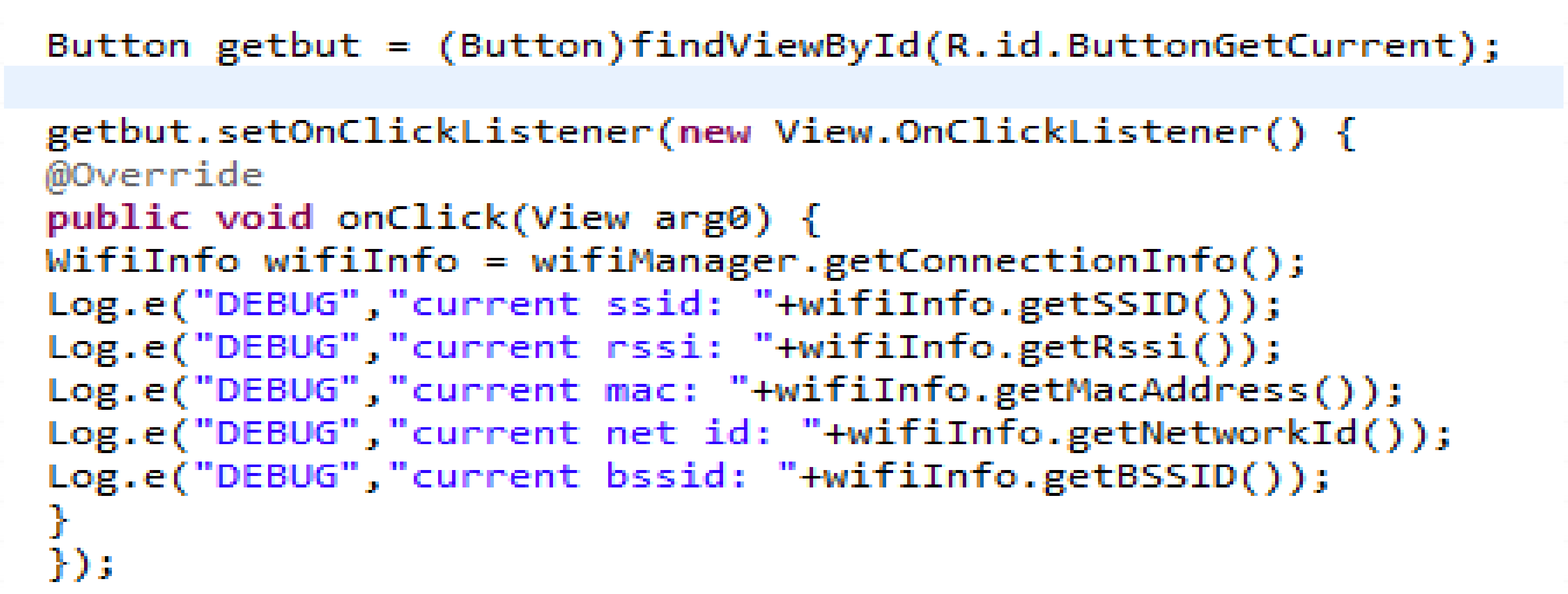
NETWORK_STATE_CHANGED_ACTION



```
if(intent.getAction().equals(WifiManager.NETWORK_STATE_CHANGED_ACTION)){
    Log.e("DEBUG",".NETWORK_STATE_CHANGED_ACTION");
    NetworkInfo mNetworkInfo = (NetworkInfo) intent.getParcelableExtra(WifiManager.EXTRA_NETWORK_INFO);
    if (mNetworkInfo.getState()==NetworkInfo.State.CONNECTED) {
    Log.e("DEBUG","connected");
    }
    if (mNetworkInfo.getState()==NetworkInfo.State.CONNECTING) {
    Log.e("DEBUG","CONNECTING ");
    }
    if (mNetworkInfo.getState()==NetworkInfo.State.DISCONNECTED) {
    Log.e("DEBUG","DISCONNECTED");
    }
    if (mNetworkInfo.getState()==NetworkInfo.State.DISCONNECTING) {
    Log.e("DEBUG","DISCONNECTING");
    }
    if (mNetworkInfo.getState()==NetworkInfo.State.SUSPENDED) {
    Log.e("DEBUG","SUSPENDED");
    }
    if (mNetworkInfo.getState()==NetworkInfo.State.SUSPENDED) {
    Log.e("DEBUG","UNKNOWN");
    }
    }
```

Get current connection info

❖Add button: Get Current
❖In onCreate, add



```
Button getbut = (Button)findViewById(R.id.ButtonGetCurrent);

getbut.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View arg0) {
WifiInfo wifiInfo = wifiManager.getConnectionInfo();
Log.e("DEBUG","current ssid: "+wifiInfo.getSSID());
Log.e("DEBUG","current rssi: "+wifiInfo.getRssi());
Log.e("DEBUG","current mac: "+wifiInfo.getMacAddress());
Log.e("DEBUG","current net id: "+wifiInfo.getNetworkId());
Log.e("DEBUG","current bssid: "+wifiInfo.getBSSID());
}
});
```

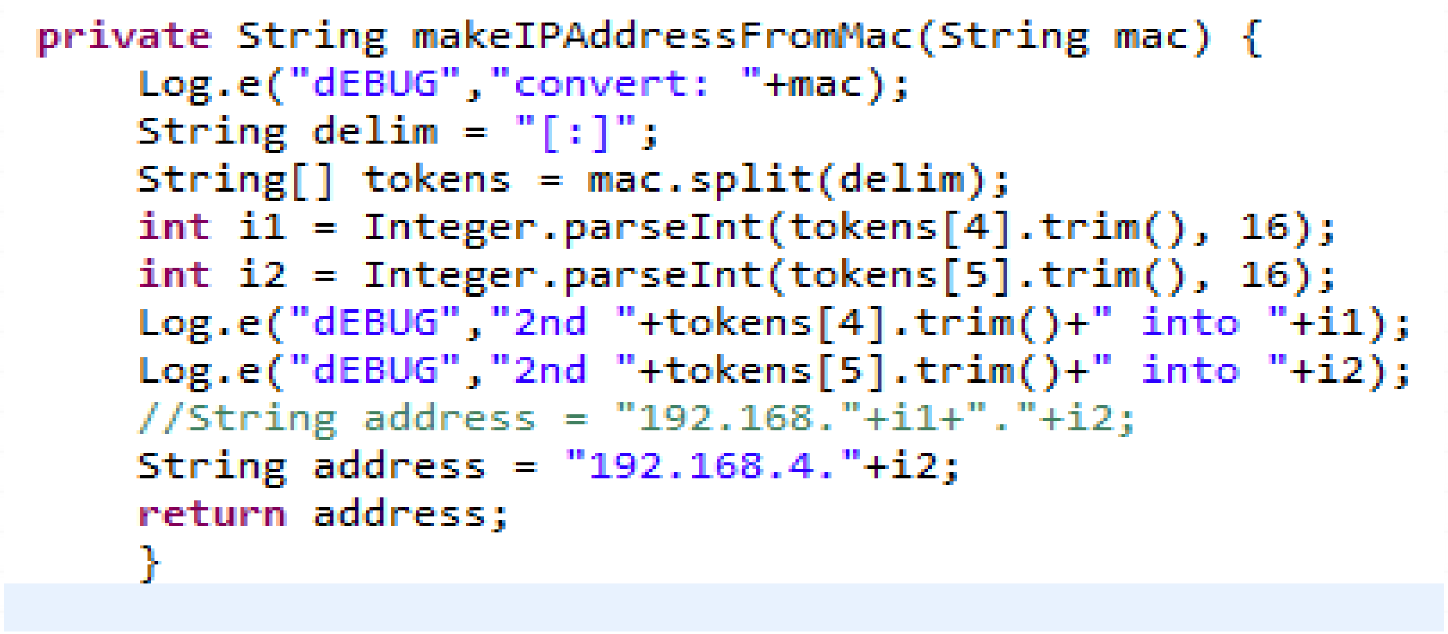
Connect to ad hoc network

- Android does not support ad hoc networking
- Only rooted phones can connect
- Add BusyBox from market place
- We need a new wpa_supplicant.conf
  - Wifi hardward <-> driver (.ko) <-> wpa_supplicant <-> (wpa_cli) <-> android system+api
- Make new app
- Add button
  - Connect to ad hoc
  - Disconnect from ad hoc

steps

- Move wpa_supplicant to correct directory
- Change permissions of wpa_supplicant
- Disable dhcp
- Set ip address etc
- Turn off wifi
  - (Perhap we could only restart wpa_supplicant)
- Set static ip
- Turn on wifi
- Check status
- Set routing table
- Try ping

Make ip from mac

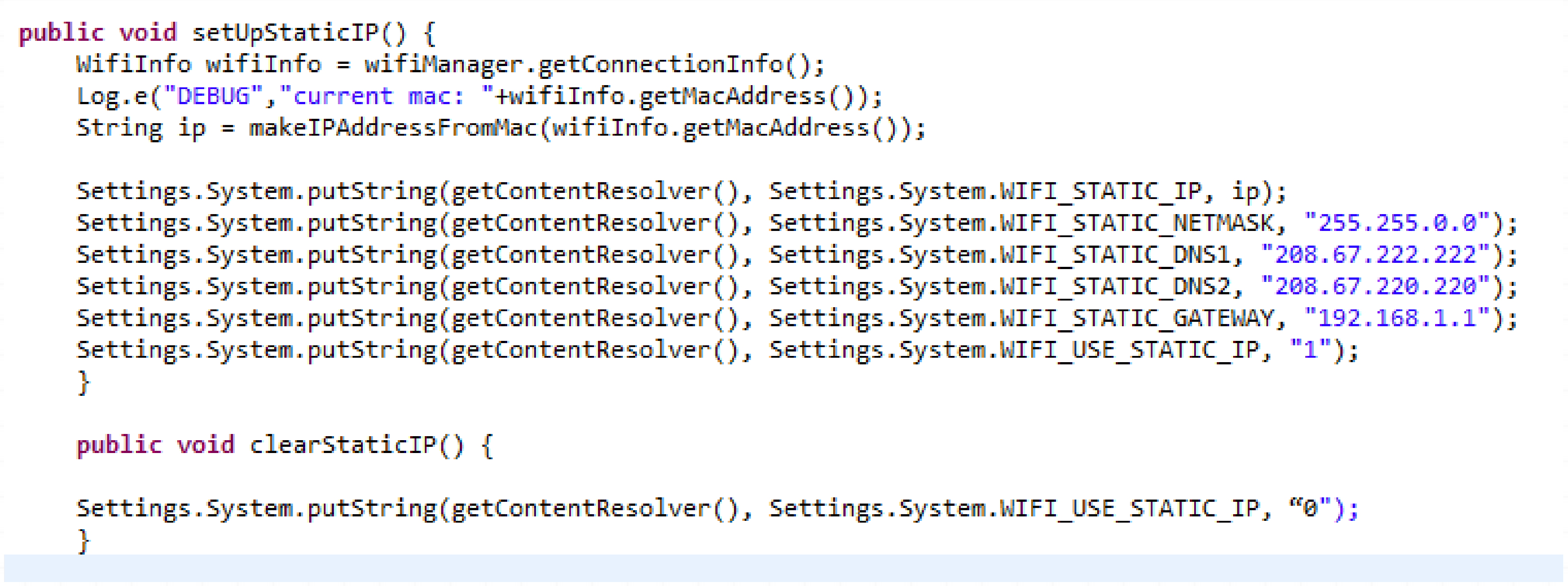


```
private String makeIPAddressFromMac(String mac) {
    Log.e("dEBUG","convert: "+mac);
    String delim = "[:]";
    String[] tokens = mac.split(delim);
    int i1 = Integer.parseInt(tokens[4].trim(), 16);
    int i2 = Integer.parseInt(tokens[5].trim(), 16);
    Log.e("dEBUG","2nd "+tokens[4].trim()+" into "+i1);
    Log.e("dEBUG","2nd "+tokens[5].trim()+" into "+i2);
    //String address = "192.168."+i1+"."+i2;
    String address = "192.168.4."+i2;
    return address;
    }
```

It would be better to embed the whole mac in an ipv6 address…

Set static ip address

- Ad hoc mode needs a static ip address, since dhcp is most likely not available
- Here we get the mac address and build an IP address in subnet 192.168/16 from the MAC
- Dns is open-dns

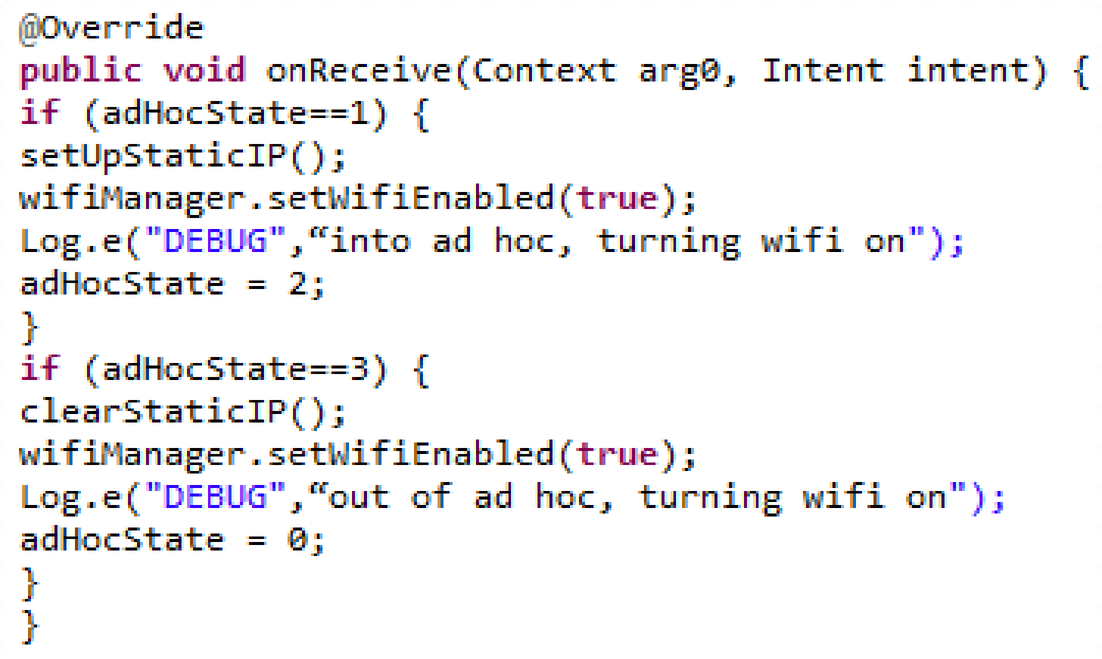


```
public void setUpStaticIP() {
    WifiInfo wifiInfo = wifiManager.getConnectionInfo();
    Log.e("DEBUG","current mac: "+wifiInfo.getMacAddress());
    String ip = makeIPAddressFromMac(wifiInfo.getMacAddress());

    Settings.System.putString(getContentResolver(), Settings.System.WIFI_STATIC_IP, ip);
    Settings.System.putString(getContentResolver(), Settings.System.WIFI_STATIC_NETMASK, "255.255.0.0");
    Settings.System.putString(getContentResolver(), Settings.System.WIFI_STATIC_DNS1, "208.67.222.222");
    Settings.System.putString(getContentResolver(), Settings.System.WIFI_STATIC_DNS2, "208.67.220.220");
    Settings.System.putString(getContentResolver(), Settings.System.WIFI_STATIC_GATEWAY, "192.168.1.1");
    Settings.System.putString(getContentResolver(), Settings.System.WIFI_USE_STATIC_IP, "1");
    }

    public void clearStaticIP() {

    Settings.System.putString(getContentResolver(), Settings.System.WIFI_USE_STATIC_IP, “0");
    }
```

steps

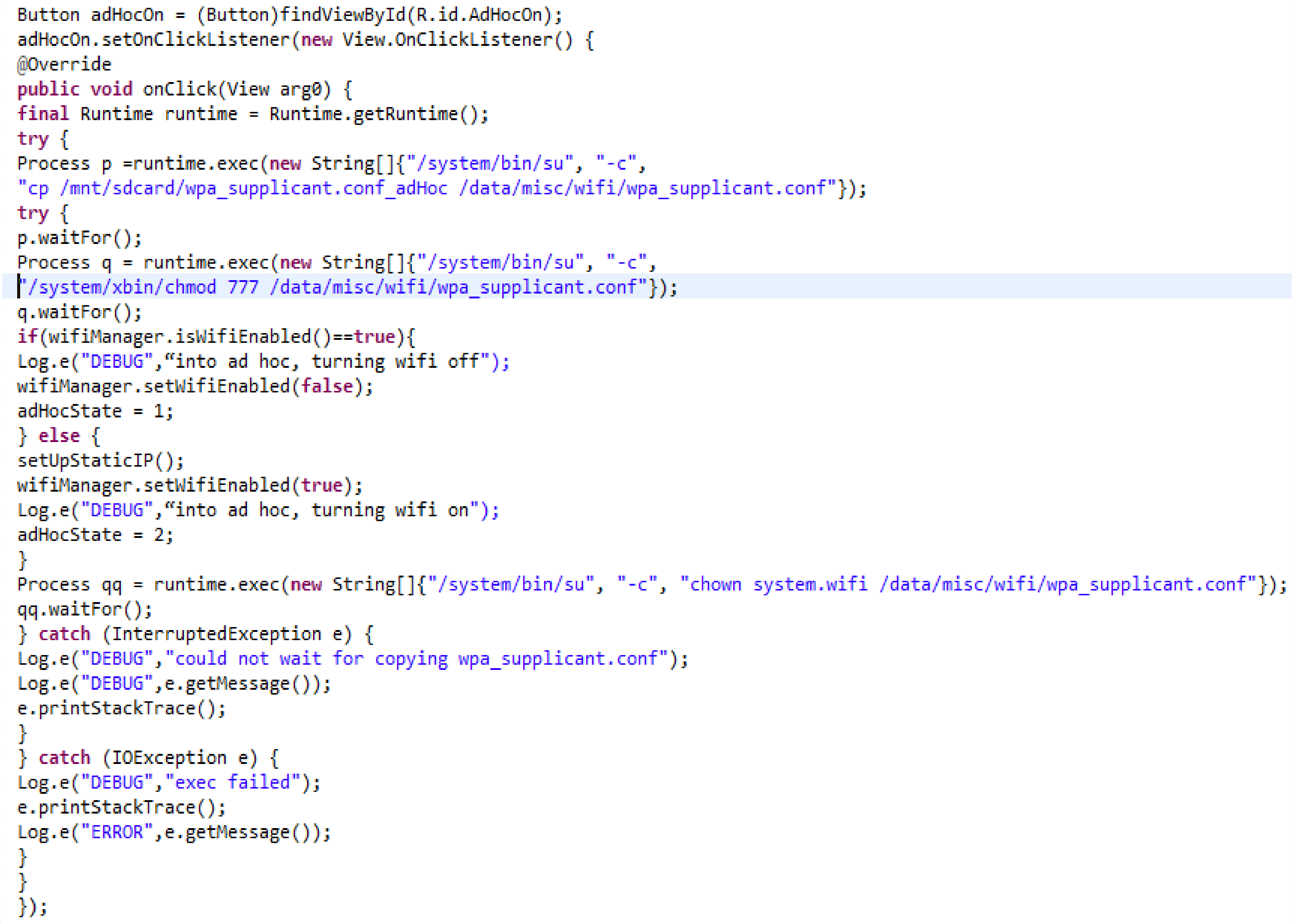
- Move wpa_supplicant to correct directory
- Change permissions of wpa_supplicant
- Disable dhcp
- Set ip address etc
- Turn off wifi
  - (Perhap we could only restart wpa_supplicant)
- Set ip
  - Trick: we want to set ip after wifi is off, which occurs well after we issue the command to turn off wifi
  - Approach:
    - Add member variable int adHocState = 0;
    - When getting into ad hoc mode
      - ✓ After turning off wifi, adHocState = 1;
      - ✓ After turning on wifi, adHocState = 2;
    - When getting out of ad hoc mode
      - ✓ After turning off wifi, adHocState = 3;
      - ✓ After turning on wifi, adHocState = 0;
    - Check for wifi state changes
      - ✓ If state change and adHocState==1, then set ip, adHocState =2, turn on wifi
      - ✓ If state change and adHocState==3, then clear static ip, adHocState =0, turn on wifi
- Turn on wifi
- Check status
- Try ping
- multi-hop Ad hoc networking - next week

- Add member variable int adHocState = 0;
- private BroadcastReceiver wifiEventReceiver = new BroadcastReceiver() {



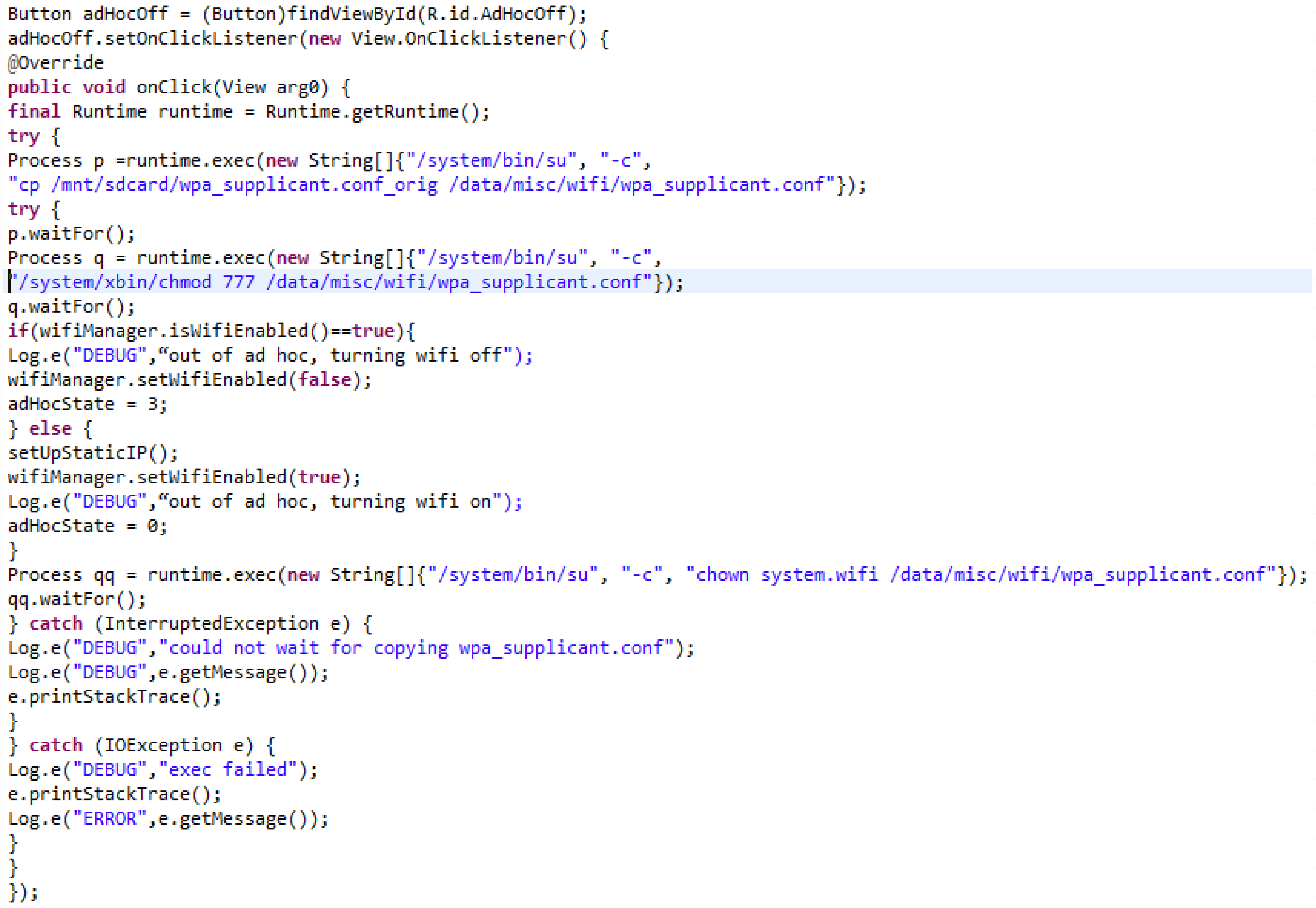
```
@Override
public void onReceive(Context arg0, Intent intent) {
if (adHocState==1) {
setUpStaticIP();
wifiManager.setWifiEnabled(true);
Log.e("DEBUG",“into ad hoc, turning wifi on");
adHocState = 2;
}
if (adHocState==3) {
clearStaticIP();
wifiManager.setWifiEnabled(true);
Log.e("DEBUG",“out of ad hoc, turning wifi on");
adHocState = 0;
}
}
```

Make button: Connect from ad hoc



```
Button adHocOn = (Button)findViewById(R.id.AdHocOn);
adHocOn.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View arg0) {
final Runtime runtime = Runtime.getRuntime();
try {
Process p =runtime.exec(new String[]{"/system/bin/su", "-c",
"cp /mnt/sdcard/wpa_supplicant.conf_adHoc /data/misc/wifi/wpa_supplicant.conf"});
try {
p.waitFor();
Process q = runtime.exec(new String[]{"/system/bin/su", "-c",
"/system/xbin/chmod 777 /data/misc/wifi/wpa_supplicant.conf"});
q.waitFor();
if(wifiManager.isWifiEnabled()==true){
Log.e("DEBUG",“into ad hoc, turning wifi off");
wifiManager.setWifiEnabled(false);
adHocState = 1;
} else {
setUpStaticIP();
wifiManager.setWifiEnabled(true);
Log.e("DEBUG",“into ad hoc, turning wifi on");
adHocState = 2;
}
Process qq = runtime.exec(new String[]{"/system/bin/su", "-c", "chown system.wifi /data/misc/wifi/wpa_supplicant.conf"});
qq.waitFor();
} catch (InterruptedException e) {
Log.e("DEBUG","could not wait for copying wpa_supplicant.conf");
Log.e("DEBUG",e.getMessage());
e.printStackTrace();
}
} catch (IOException e) {
Log.e("DEBUG","exec failed");
e.printStackTrace();
Log.e("ERROR",e.getMessage());
}
}
});
```

Make button: disconnect from ad hoc



```java
Button adHocOff = (Button)findViewById(R.id.AdHocOff);
adHocOff.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View arg0) {
final Runtime runtime = Runtime.getRuntime();
try {
Process p =runtime.exec(new String[]{"/system/bin/su", "-c",
"cp /mnt/sdcard/wpa_supplicant.conf_orig /data/misc/wifi/wpa_supplicant.conf"});
try {
p.waitFor();
Process q = runtime.exec(new String[]{"/system/bin/su", "-c",
"/system/xbin/chmod 777 /data/misc/wifi/wpa_supplicant.conf"});
q.waitFor();
if(wifiManager.isWifiEnabled()==true){
Log.e("DEBUG",“out of ad hoc, turning wifi off");
wifiManager.setWifiEnabled(false);
adHocState = 3;
} else {
setUpStaticIP();
wifiManager.setWifiEnabled(true);
Log.e("DEBUG",“out of ad hoc, turning wifi on");
adHocState = 0;
}
Process qq = runtime.exec(new String[]{"/system/bin/su", "-c", "chown system.wifi /data/misc/wifi/wpa_supplicant.conf"});
qq.waitFor();
} catch (InterruptedException e) {
Log.e("DEBUG","could not wait for copying wpa_supplicant.conf");
Log.e("DEBUG",e.getMessage());
e.printStackTrace();
}
} catch (IOException e) {
Log.e("DEBUG","exec failed");
e.printStackTrace();
Log.e("ERROR",e.getMessage());
}
}
});
```