Mobile Computing Course
5th Session

Wireless Networks

Wi-Fi scanning overview

You can use the Wi-Fi scanning capabilities provided by the WifiManager API to get a list of Wi-Fi access points that are visible from the device.

WifiManager

```
public class WifiManager
extends Object
```

java.lang.Object
↳ android.net.wifi.WifiManager

This class provides the primary API for managing all aspects of Wi-Fi connectivity.

On releases before `Build.VERSION_CODES.N`, this object should only be obtained from an application context, and not from any other derived context to avoid memory leaks within the calling process.

It deals with several categories of items:

- The list of configured networks. The list can be viewed and updated, and attributes of individual entries can be modified.
- The currently active Wi-Fi network, if any. Connectivity can be established or torn down, and dynamic information about the state of the network can be queried.
- Results of access point scans, containing enough information to make decisions about what access point to connect to.
- It defines the names of various Intent actions that are broadcast upon any sort of change in Wi-Fi state.

WifiInfo

```
public class WifiInfo
extends Object implements TransportInfo, Parcelable
```

java.lang.Object
↳ android.net.wifi.WifiInfo

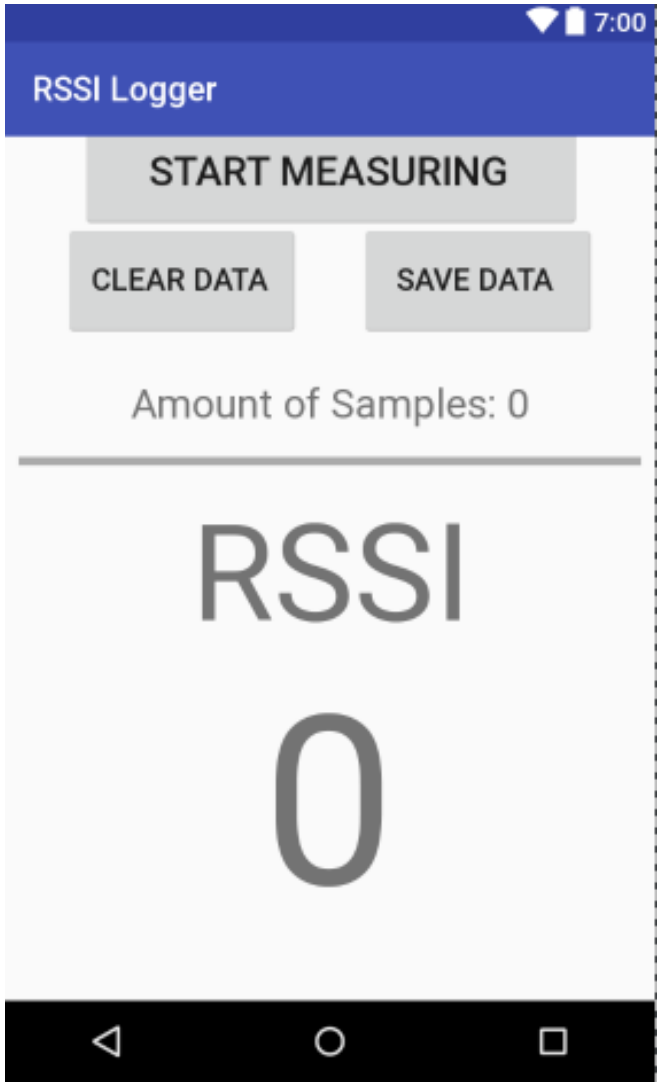
Describes the state of any Wi-Fi connection that is active or is in the process of being set up. In the connected state, access to location sensitive fields requires the same permissions as `WifiManager#getScanResults`. If such access is not allowed, `getSSID()` will return `WifiManager#UNKNOWN_SSID` and `getBSSID()` will return `"02:00:00:00:00:00"`. `getNetworkId()` will return `-1`. `getPasspointFqdn()` will return null. `getPasspointProviderFriendlyName()` will return null. `getInformationElements()` will return null. `getMacAddress()` will return `"02:00:00:00:00:00"`.

Permission

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

Example RSSI Logger

https://github.com/iamphu/RSSI_Logger/blob/master/app/src/main/java/com/adhocnetworks/rssi_logger/MainActivity.java



Determine whether your app was already granted the permission

To check if the user has already granted your app a particular permission, pass that permission into the `ContextCompat.checkSelfPermission()` method. This method returns either `PERMISSION_GRANTED` or `PERMISSION_DENIED`, depending on whether your app has the permission.

requestPermissions

```
public static void requestPermissions (Activity activity,
                String[] permissions,
                int requestCode)
```

```
// WiFi variables
    private WifiManager wifiManager;
    private WifiInfo wifiInfo;


//set wifimanger
wifiManager= (WifiManager)
getApplicationContext().getSystemService(Context
.WIFI_SERVICE);


wifiInfo = wifiManager.getConnectionInfo();

values.setText(String.valueOf(wifiInfo.getRssi()
));
```

Handler

Added in API level 1

Kotlin | **Java**

```
public class Handler
extends Object
```

java.lang.Object
↳ android.os.Handler

Known direct subclasses
AsyncQueryHandler, AsyncQueryHandler.WorkerHandler, HttpAuthHandler, SslErrorHandler

---

A Handler allows you to send and process `Message` and Runnable objects associated with a thread's `MessageQueue`. Each Handler instance is associated with a single thread and that thread's message queue. When you create a new Handler it is bound to a `Looper`. It will deliver messages and runnables to that Looper's message queue and execute them on that Looper's thread.

There are two main uses for a Handler: (1) to schedule messages and runnables to be executed at some point in the future; and (2) to enqueue an action to be performed on a different thread than your own.

Scheduling messages is accomplished with the `post(Runnable)`, `postAtTime(java.lang.Runnable, long)`, `postDelayed(Runnable, Object, long)`, `sendEmptyMessage(int)`, `sendMessage(Message)`, `sendMessageAtTime(Message, long)`, and `sendMessageDelayed(Message, long)` methods. The *post* versions allow you to enqueue Runnable objects to be called by the message queue when they are received; the *sendMessage* versions allow you to enqueue a `Message` object containing a bundle of data that will be processed by the Handler's `handleMessage(Message)` method (requiring that you implement a subclass of Handler).

Runnable

Added in API level 1

```
public interface Runnable
```

java.lang.Runnable

Known indirect subclasses
AnimationDrawable, CookieSyncManager, ForkJoinWorkerThread, FutureTask<V>, HandlerThread, RenderScript.RSErrorHandler, RenderScript.RSMessageHandler, RunnableFuture<V>, RunnableScheduledFuture<V>, Thread, TimerTask

---

The `Runnable` interface should be implemented by any class whose instances are intended to be executed by a thread. The class must define a method of no arguments called `run`.

This interface is designed to provide a common protocol for objects that wish to execute code while they are active. For example, `Runnable` is implemented by class `Thread`. Being active simply means that a thread has been started and has not yet been stopped.

Celluar Signal Logger

Report

```
370     private int getCellSignalStrength() {
371       int strength = Integer.MIN_VALUE;
372       List<CellInfo> cellInfos = mTelMgr.getAllCellInfo();   //This will give info of all sims present inside your mobile
373       if(cellInfos!=null) {
374         for (int i = 0; i < cellInfos.size(); i++) {
375           if (cellInfos.get(i).isRegistered()) {
376             if (cellInfos.get(i) instanceof CellInfoWcdma) {
377               CellInfoWcdma cellInfoWcdma = (CellInfoWcdma) mTelMgr.getAllCellInfo().get(0);
378               CellSignalStrengthWcdma cellSignalStrengthWcdma = cellInfoWcdma.getCellSignalStrength();
379               strength = cellSignalStrengthWcdma.getLevel();
380             } else if (cellInfos.get(i) instanceof CellInfoLte) {
381               CellInfoLte cellInfoLte = (CellInfoLte) mTelMgr.getAllCellInfo().get(0);
382               CellSignalStrengthLte cellSignalStrengthLte = cellInfoLte.getCellSignalStrength();
383               strength = cellSignalStrengthLte.getLevel();
384             } else if (cellInfos.get(i) instanceof CellInfoGsm) {
385               CellInfoGsm cellInfogsm = (CellInfoGsm) mTelMgr.getAllCellInfo().get(0);
386               CellSignalStrengthGsm cellSignalStrengthGsm = cellInfogsm.getCellSignalStrength();
387               strength = cellSignalStrengthGsm.getLevel();
388             }
389           }
390         }
391       }
```

See you in next session