Managing Network and Wi -Fi Connections

The incredible growth of Internet services and the ubiquity of mobile devices has made mobile Internet access an increasingly prevalent feature on mobile phones.

With the speed, reliability, and cost of Internet connectivity dependent on the network technology being used (Wi-Fi, GPRS, 3G), letting your applications know and manage these connections can help to ensure that they run effi ciently and responsively.

Android provides access to the underlying network state, broadcasting Intents to notify application components of changes in network connectivity and offering control over network settings and connections.

Android networking is principally handled using the ConnectivityManager, a Service that lets you monitor the connectivity state, set your preferred network connection, and manage connectivity failover.

Later you'll learn how to use the WifiManager to monitor and control the device's Wi-Fi connectivity specifically. The Wi-Fi Manager lets you see and modify the confi gured Wi-Fi networks, manage the active connection, and perform access point scans.

Monitoring and Managing Your Internet Connectivity

The ConnectivityManager represents the Network Connectivity Service. It's used to monitor the state of network connections, confi gure failover settings, and control the network radios. To access the Connectivity Manager, call getSystemService, passing in Context.CONNECTIVITY_SERVICE as the service name, as shown in the code snippet below:

```
String service = Context.CONNECTIVITY_SERVICE;
ConnectivityManager connectivity = (ConnectivityManager)getSystemService(service);
```

Before it can use the Connectivity Manager, your application will need Read and Write network state access permissions to be added to the manifest, as shown below:

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

Managing Active Connections

The Connectivity Manager provides a high-level view of the available network connections. Using the getActiveNetworkInfo or getNetworkInfo methods, you query NetworkInfo objects that include details on the currently active network or on an inactive network of the type specified.

In both cases, the NetworkInfo returned includes methods that indicate the connection status and network type of the specifi ed network.

Configuring Network Preferences and Controlling Hardware

The Connectivity Manager can also be used to control network hardware and confi gure failover preferences. Android will attempt to connect to the preferred network whenever an authorized application requests an Internet connection. You can set the preferred network using the setNetworkPreference method and specifying the network you would prefer to connect to, as shown in the code snippet below:

```
connectivity.setNetworkPreference(NetworkPreference.PREFER_WIFI);
```

If the preferred connection is unavailable, or connectivity on this network is lost, Android will automatically attempt to connect to the secondary network.

You can control the availability of the network types, using the setRadio method. This method lets you set the state of the radio associated with a particular network (Wi-Fi, mobile, etc.). For example, in the following code snippet, the Wi-Fi radio is turned off and the mobile radio is turned on:

```
connectivity.setRadio(NetworkType.WIFI, false);
connectivity.setRadio(NetworkType.MOBILE, true);
```

Monitoring Network Connectivity

One of the most useful functions of the Connectivity Manager is to notify applications of changes in network connectivity.

To monitor network connectivity, create your own Broadcast Receiver implementation that listens for ConnectivityManager.CONNECTIVITY_ACTION Intents. Such Intents include several extras that provide additional details on the change to the connectivity state:

- ConnectivityManager.EXTRA_IS_FAILOVER Is a Boolean that returns true if the current connection is the result of a failover from a preferred network.
- ConnectivityManager.EXTRA_NO_CONNECTIVITY Is a Boolean that returns true if the device is not connected to any network.
- ConnectivityManager.EXTRA_REASON If this broadcast represents a connection failure, this string value includes a description of why the connection attempt failed.
- ConnectivityManager.EXTRA_NETWORK_INFO This returns a NetworkInfo object containing more fi ne-grained details on the network associated with the current connectivity event.
- ConnectivityManager.EXTRA_OTHER_NETWORK_INFO After a network disconnection, this value will return a NetworkInfo object populated with the details for the possible failover network connection.
- ConnectivityManager.EXTRA_EXTRA_INFO Contains additional network-specific extra connection details.

Android SDK beta 0.9 included a NetworkConnectivityListener that encapsulated this functionality. This class has been removed for version 1.0.

Managing Your Wi-Fi

The WifiManager represents the Android Wi-Fi Connectivity Service. It can be used to confi gure Wi-Fi network connections, manage the current Wi-Fi connection, scan for access points, and monitor changes in Wi-Fi connectivity.

As with the Connectivity Manager, access the Wi-Fi Manager using the getSystemService method, passing in the Context.WIFI_SERVICE constant, as shown in the following code snippet:

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

To use the Wi-Fi Manager, your application must have uses-permissions for Read/Write Wi-Fi state access included in its manifest.

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

You can use the Wi-Fi Manager to enable or disable your Wi-Fi hardware using the setWifiEnabled method, or request the current Wi-Fi state using the getWifiState or isWifiEnabled methods as shown in the code snippet below:

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

The following sections begin with tracking the current Wi-Fi connection status and monitoring changes in signal strength. Later you'll also learn how to scan for and connect to specifi c access points.

While these functions may be suffi cient for most developers, the WifiManager also provides low-level access to the Wi-Fi network confi gurations, giving you full control over each confi guration setting and allowing you to completely replace the native Wi-Fi management application. Later in the section, you'll be introduced to the API used to create, delete, and modify network confi gurations.

Monitoring Wi-Fi Connectivity

The Wi-Fi Manager broadcasts Intents whenever the connectivity status of the Wi-Fi network changes, using the following actions:

❑ WifiManager.WIFI_STATE_CHANGED_ACTION Indicates that the Wi-Fi hardware status has changed, moving between enabling, enabled, disabling, disabled, and unknown. It includes two extra values keyed on EXTRA_WIFI_STATE and EXTRA_PREVIOUS_STATE that provide the previous and new Wi-Fi states.

❑ WifiManager.SUPPLICANT_CONNECTION_CHANGE_ACTION This Intent is broadcast whenever the connection state with the active supplicant (access point) changes. It is fi red when a new connection is established or an existing connection is lost using the EXTRA_NEW_STATE Boolean extra that returns true in the former case.

❑ WifiManager.NETWORK_STATE_CHANGED_ACTION The network state change broadcast is fi red whenever the Wi-Fi connectivity state changes. This Intent includes two extras — the fi rst EXTRA_NETWORK_INFO includes a NetworkInfo object that details the current network state, while the second EXTRA_BSSID includes the BSSID of the access point you're connected to.

❑ WifiManager.RSSI_CHANGED_ACTION You can monitor the current signal strength of the connected Wi-Fi network by listening for the RSSI_CHANGED_ACTION Intent. This Broadcast Intent includes an integer extra, EXTRA_NEW_RSSI, that holds the current signal strength. To use this signal strength, you should use the calculateSignalLevel static method on the

Wi-Fi Manager to convert it to an integer value on a scale you specify.

Creating and Managing Wi-Fi Connections and Configurations

You can use the Wi-Fi Manager to manage the confi gured network settings and control which networks to connect to. Once connected, you can interrogate the active network connection to get additional details of its configuration and settings.

Get a list of the current network confi gurations using getConfiguredNetworks. The list of Wifi Configuration objects returned includes the network ID, SSID, and other details for each configuration.

To use a particular network confi guration, use the enableNetwork method, passing in the network ID to use and specifying true for the disableAllOthers parameter as shown below:

```
// 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);
}
```

Once an active network connection has been established, use the getConnectionInfo method to return information on the active connection's status. The returned WifiInfo object includes the BSSID, Mac address, and IP address of the current access point, as well as the current link speed and signal strength.

The following code snippet queries the currently active Wi-Fi connection and displays a Toast showing the connection speed and signal strength:

```
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 toastText = String.format("Connected to {0} at {1}{2}. Strength {3}/5",
ssid, speed, units, strength);
Toast.makeText(this, toastText, Toast.LENGTH_LONG);
}
```

Scanning for Hotspots

You can use the Wi-Fi Manager to conduct access point scans using the startScan method. An Intent with the SCAN_RESULTS_AVAILABLE_ACTION action will be broadcast to asynchronously announce that the scan is complete and results are available.

Call getScanResults to get those results as a list of ScanResult objects. Each ScanResult includes the details retrieved for each access point detected, including link speed, signal strength, SSID, and the authentication techniques supported.

The following skeleton code shows how to initiate a scan for access points that displays a Toast indicating the total number of access points found and the name of the Access Point with the strongest signal:

```
// 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("{0} networks found. {1} 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();
```

Managing Wi-Fi Network Confi gurations

To connect to a Wi-Fi network, you need to create and register a confi guration. Normally your users would do this using the native Wi-Fi confi guration settings, but there's no reason you can't expose the same functionality within your own applications, or for that matter replace the native Wi-Fi confi guration Activity entirely.

Network confi gurations are stored as WifiConfiguration objects. The following is a non-exhaustive list of some of the public fi elds available for each Wi-Fi Confi guration:

- BSSID Specifi es the BSSID for an access point.
- SSID The SSID for a particular network
- networkId A unique identifi er used to identify this network confi guration on the current device
- priority The priority of each network confi guration when choosing which of several access points to connect to
- status The current status of this network connection, will be one of WifiConfiguration.Status.ENABLED, WifiConfiguration.Status.DISABLED, or WifiConfiguration.Status.CURRENT

The confi guration object also contains the supported authentication technique, as well as the keys used previously to authenticate with this access point.

The addNetwork method lets you specify a new confi guration to add to the current list; similarly, updateNetwork lets you update a network confi guration by passing in a WifiConfiguration that's sparsely populated with a network ID and the values you want to change.

You can also use removeNetwork, passing in a network ID, to remove a configuration.

To persist any changes made to the network configurations, you must call saveConfiguration.