Using Internet Resources

With Internet connectivity and WebKit browser, you might well ask if there's any reason to create native Internet-based applications when you could make a web-based version instead.

There are several benefi ts to creating thick- and thin-client native applications rather than relying on entirely web-based solutions:

❑ **Bandwidth** Static resources like images, layouts, and sounds can be expensive data consumers on devices with limited and often expensive bandwidth restraints. By creating a native application, you can limit the bandwidth requirements to only data updates.

❑ **Caching** Mobile Internet access has not yet reached the point of ubiquity. With a browserbased solution, a patchy Internet connection can result in intermittent application availability. A native application can cache data to provide as much functionality as possible without a live connection.

❑ **Native Features** Android devices are more than a simple platform for running a browser; they include location-based services, camera hardware, and accelerometers. By creating a native application, you can combine the data available online with the hardware features available on the device to provide a richer user experience.

Modern mobile devices offer various alternatives for accessing the Internet. Looked at broadly, Android provides three connection techniques for Internet connectivity. Each is offered transparently to the application layer.

❑ **GPRS, EDGE, and 3G** Mobile Internet access is available through carriers that offer mobile data plans.

❑ **Wi-Fi** Wi-Fi receivers and mobile hotspots are becoming increasingly more common.

Connecting to an Internet Resource

While the details of working with specifi c web services aren't covered within this book, it's useful to know the general principles of connecting to the Internet and getting an input stream from a remote data source.

Before you can access Internet resources, you need to add an INTERNET uses-permission node to your application manifest, as shown in the following XML snippet:

```
<uses-permission android:name=”android.permission.INTERNET”/>
```

The following skeleton code shows the basic pattern for opening an Internet data stream:

```
String myFeed = getString(R.string.my_feed);
try {
URL url = new URL(myFeed);
URLConnection connection = url.openConnection();
HttpURLConnection httpConnection = (HttpURLConnection)connection;
int responseCode = httpConnection.getResponseCode();
if (responseCode == HttpURLConnection.HTTP_OK) {
InputStream in = httpConnection.getInputStream();

[ ... Process the input stream as required ... ]
}
}
catch (MalformedURLException e) { }
catch (IOException e) { }
```

Android includes several classes to help you handle network communications. They are available in the

java.net.* and android.net.* packages.

Later in this chapter, there is a fully worked example that shows how to obtain and process an Internet feed to get a list of earthquakes felt in the last 24 hours.

Chapter 9 includes further details on Internet-based communications using the GTalk Service. Chapter 10 features more information on managing specifi c Internet connections, including monitoring connection status and confi guring Wi-Fi access point connections.

Leveraging Internet Resources

Android offers several ways to leverage Internet resources.

At one extreme, you can use the WebView widget to include a WebKit-based browser control within an Activity. At the other extreme, you can use client-side APIs such as Google's GData APIs to interact directly with server processes. Somewhere in between, you can process remote XML feeds to extract and process data using a Java-based XML parser such as SAX or javax.

Detailed instructions for parsing XML and interacting with specifi c web services are outside the scope of this book. That said, the Earthquake example, included later in this chapter, gives a fully worked example of parsing an XML feed using the javax classes.

If you're using Internet resources in your application, remember that your users' data connections depend on the communications technology available to them. EDGE and GSM connections are notoriously low bandwidth, while a Wi-Fi connection may be unreliable in a mobile setting.

Optimize the user experience by limiting the quantity of data being transmitted, and ensure that your application is robust enough to handle network outages and bandwidth limitations.