developed in Java and run on the Linux 2.6 kernel. If you are a quick learner, you may be able to understand what is going on with just some basic object-oriented programming (OOP) experience. Chapter 2 explains how to download and install the preferred integrated development environment, Eclipse. All the code samples and screenshots in this book are provided using Eclipse (Europa release) and the Android plugin for Eclipse.

Any comments, questions, or suggestions about any of the material in this book can be forwarded directly to the author at **jfdimarzio@jfdimarzio.com**.

Device manufacturers still closely guard the operating systems that run on their devices. While a few have opened up to the point where they will allow some Java-based applications to run within a small environment on the phone, many do not allow this. Even the systems that do allow some Java apps to run do not allow the kind of access to the "core" system that standard desktop developers are accustomed to having.

Open Handset Alliance and Android

This barrier to application development began to crumble in November of 2007 when Google, under the Open Handset Alliance, released Android. The Open Handset Alliance is a group of hardware and software developers, including Google, NTT DoCoMo, Sprint Nextel, and HTC, whose goal is to create a more open cell phone environment. The first product to be released under the alliance is the mobile device operating system, Android. (For more information about the Open Handset Alliance, see www.openhandsetalliance.com.)

With the release of Android, Google made available a host of development tools and tutorials to aid would-be developers onto the new system. Help files, the platform software development kit (SDK), and even a developers' community can be found at Google's Android website, http://code.google.com/android. This site should be your starting point, and I highly encourage you to visit the site.

NOTE

Google, in promoting the new Android operating system, even went as far as to create a $10 million contest looking for new and exciting Android applications.

While cell phones running Linux, Windows, and even PalmOS are easy to find, as of this writing, no hardware platforms have been announced for Android to run on. HTC, LG Electronics, Motorola, and Samsung are members of the Open Handset Alliance, under which Android has been released, so we can only hope that they have plans for a few Android-based devices in the near future. With its release in November 2007, the system itself is still in a software-only beta. This is good news for developers because it gives us a rare advance look at a future system and a chance to begin developing applications that will run as soon as the hardware is released.

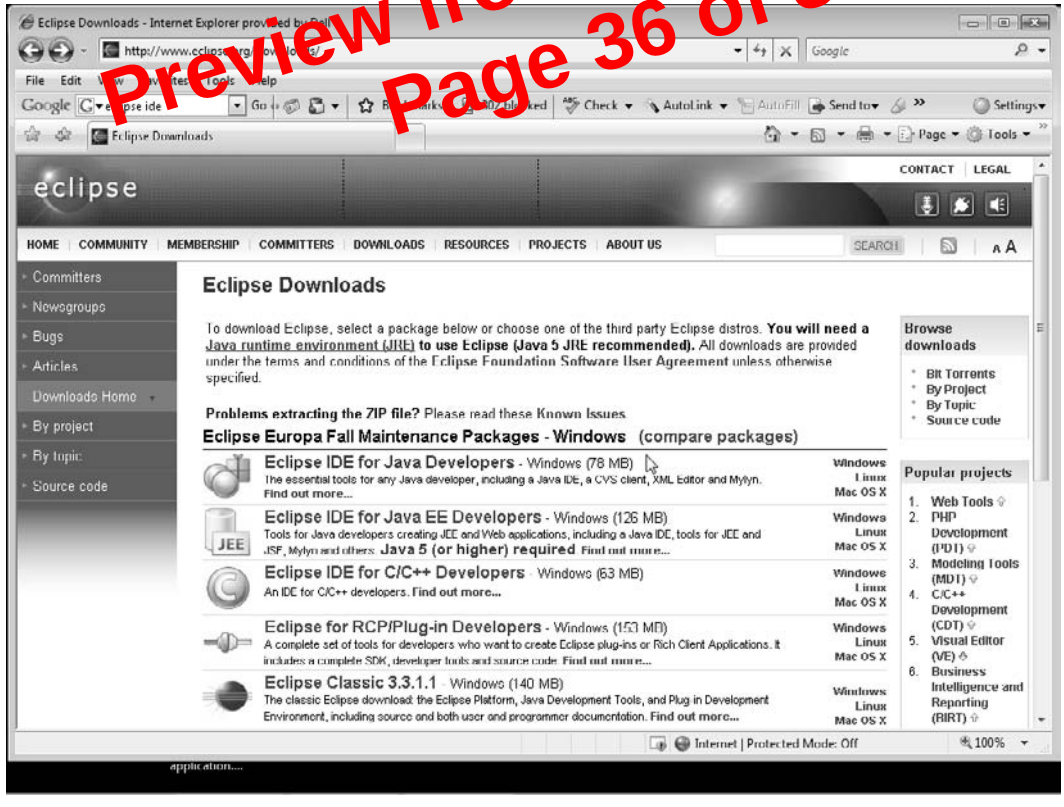
Chapter 2

Downloading and Installing Eclipse

Once you complete the Java JDK installation—and by default the JRE installation—you can begin to install Eclipse.

Downloading and Installing Eclipse

Navigate to the Eclipse Downloads page at www.eclipse.org/downloads, shown in the following illustration. As the opening paragraph states, the JRE is required (Java 5 JRE recommended) to develop in Eclipse, which you took care of in the previous section. Download the Eclipse IDE for Java Developers from this site. The package is relatively small (79MB) and should download fairly quickly. Be sure not to download the Eclipse IDE for Java EE Developers, as this is a slightly different product and I will not be covering its usage.



Key Skills & Concepts

- Using the Android SDK documentation
- Using the Android SDK tools
- Using the sample applications
- Learning the life cycle of an Android application

Now that you have your development environment established, you are ready to explore the Android SDK, which contains multiple files and tools specifically intended to help you design and develop applications that run on the Android platform. These tools are very well designed and can help you make some incredible applications. You really need to be familiar with the Android SDK and its tools before you begin programming.

The Android SDK also contains libraries for tying your applications into core Android features such as those associated with cell phone functions (making and receiving calls), GPS functionality, and text messaging. These libraries make up the core of the SDK and will be the ones that you use most often, so take the time to learn all about these core libraries.

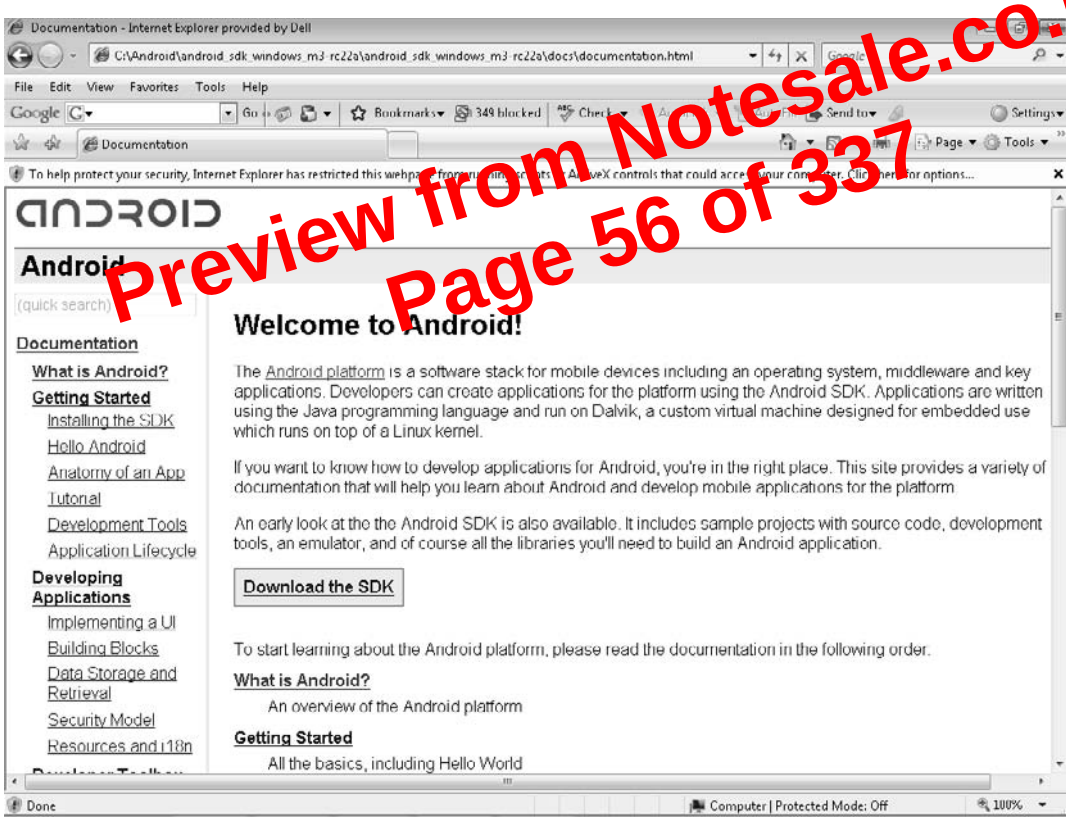
This chapter covers all of the important items contained within the Android SDK. By the end of the chapter, after familiarizing yourself with the contents of the Android SDK, you will be comfortable enough to begin writing applications. However, as with any subject, before you can dive into the practice of the discipline, you must familiarize yourself with its contents and instructions.

CAUTION

I am not going to go over every minute detail of the Android SDK; Google does a very good job of that within its documentation. To avoid the risk of spending too much time *discussing* how things work instead of *showing* you how they work, I have tried to keep this discussion as brief as possible. I cover only the most important topics and items, leaving you free to explore the rest in more depth yourself, at your own pace.

Android Documentation

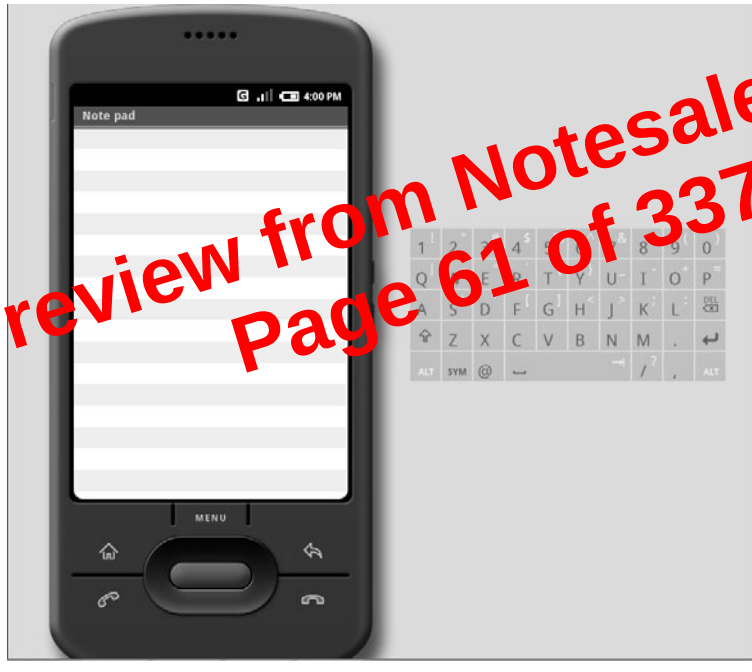
The Android documentation is located in the Docs folder within the Android SDK at ../%sdk folder%/DOCS. The documentation that is supplied with the SDK includes steps on downloading and installing the SDK, "Getting Started" quick steps for developing applications, and package definitions. The documentation is in HTML format and can be accessed though the documentation.html file in the root of the SDK folder. The following illustration depicts the main page of the Android SDK documentation.



You can navigate to all of the documentation that is included in the Android SDK by using the links within documentation.html.

Note Pad

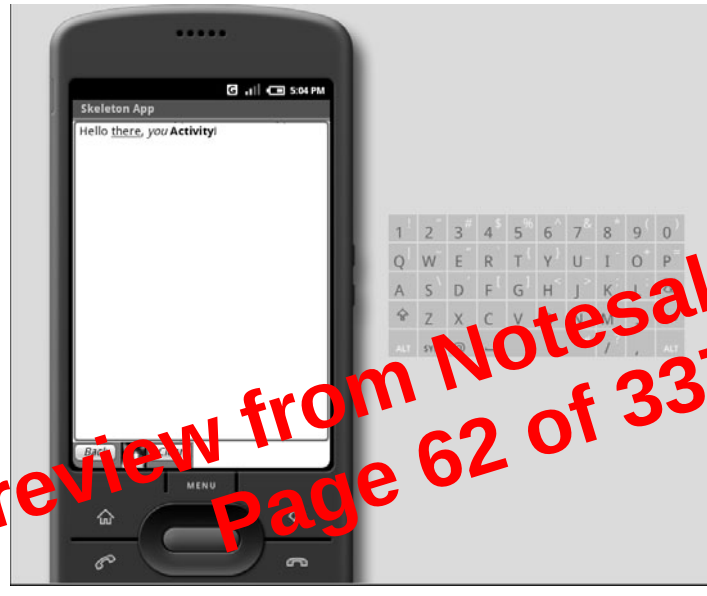
Note Pad, as shown in the illustration that follows, allows you to open, create, and edit small notes. Note Pad is not a full-featured word editor, so do not expect it to be something to rival Word for Windows Mobile. However, it does a good job as a demonstration tool to show what is possible with a relatively small amount of code.



Skeleton App

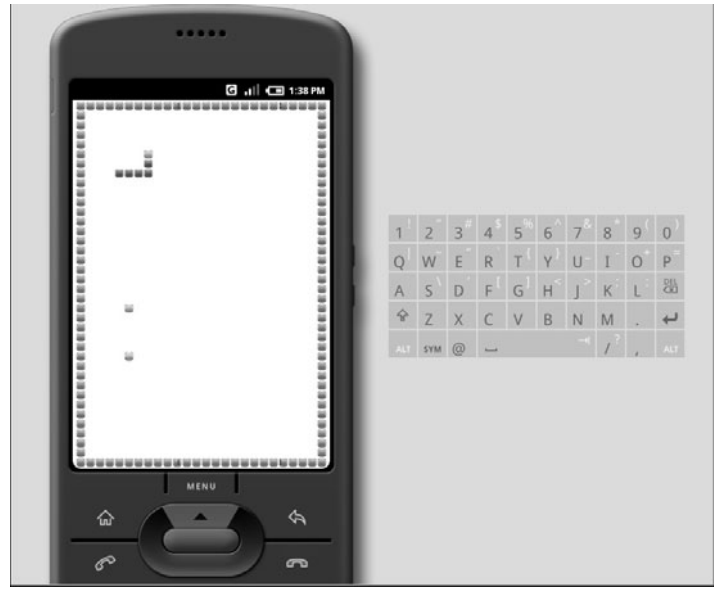
Skeleton App, shown next, is an application shell. This is more of a base application that demonstrates a couple of different application features, such as fonts, buttons, images, and forms. If you are going to run Skeleton App by itself, you really are not going to get much

out of it. You will be better served by referring to Skeleton App as a resource for how to implement specific items.



Snake

The final demo that is included with the Android SDK is Snake. This is a small, SNAFU-style game that is far more simplistic than Lunar Lander. This illustration shows what Snake looks like when run.



NOTE

If you navigate to the base folder of each of the sample applications, you will see a folder named src. This is the source code folder for the given sample application. You can use this to view, edit, and recompile the code for any of the applications. Take advantage of this source code to learn some tricks and tips about the Android platform.

Android Tools

The Android SDK supplies developers with a number of powerful and useful tools. Throughout this book, you will use only a handful of them directly. This section takes a quick look at just a few of these tools, which will be covered in much more depth in the following chapters, as you dive into command-line development.

NOTE

For more detailed information about the other tools included in the Android SDK, consult the Android doc files.

emulator.exe

Arguably one of the most important tools included in the Android SDK is emulator.exe. emulator.exe launches the Android Emulator. The Android Emulator is used to run your applications in a pseudo-Android environment. Given that, as of the writing of this book, there were no hardware devices yet released for the Android platform, emulator.exe is going to be your only means to test applications on a "native" platform.

You can run emulator.exe from the command line or execute it from within Eclipse. In this book, you'll usually let Eclipse launch the Android Emulator environment for you. However, in the interest of giving you all the information you need to program with the Android SDK outside of Eclipse, Chapter 6 covers command-line usage of emulator.exe when you create your Hello World! applications.

When using the Android Emulator to test your applications, you have two choices for navigating the user interface. First, the emulator comes with usable buttons, as shown in Figure 4-1. You can use these buttons to navigate Android and any applications that you develop for the platform.

TIP

The Power On/Off, Volume Up, and Volume Down buttons are slightly hidden to the sides of the virtual device. They identify themselves when you hover the mouse pointer over them.

APIs

The API, or application programming interface, is the core of the Android SDK. An API is a collection of functions, methods, properties, classes, and libraries that is used by application developers to create programs that work on specific platforms. The Android API contains all the specific information that you need to create applications that can work on and interact with an Android-based application.

The Android SDK also contains two supplementary sets of APIs—the Google APIs and the Optional APIs. Subsequent chapters will focus much more on these APIs as you begin writing applications that utilize them. For now, take a quick look at what they include so that you are familiar with their uses.

Google APIs

The Google APIs are included in the Android SDK and contain the programming references that allow you to tie your applications into existing Google services. If you are writing an Android application and want to allow your user to access Google services through your application, you need to include the Google API.

Located in the android.jar file, the Google API is contained within the com.google.* package. There are quite a few packages that are included with the Google API. Some of the packages that are shipped in the Google API include those for graphics, portability, contacts, and calendar utilities. However, the packages devoted to Google Maps will be the primary focus in this book.

Using the com.google.android.maps package, which contains information for Google Maps, you can create applications that interact seamlessly with the already familiar interface of Google Maps. This one set of packages opens a whole world of useful applications just waiting to be created.

The Google API also contains a useful set of packages that allows you to take advantage of the newer Extensible Messaging and Presence Protocol (XMPP) developed by the Jabber open source community. Using XMPP, applications can quickly become aware of other clients' presence and availability for the purpose of messaging and communications. The API packages dealing with XMPP are very useful if you want to create a "chat"-style program that utilizes the phone messaging capabilities.

Optional APIs

The Android SDK includes a number of Optional APIs that cover functionality not covered by the standard Android APIs. These Optional APIs are considered optional because they deal with functionality that may or may not be present on a given handset

To make sure that you get a good overall look at programming in Android, in Chapter 6 you will create both of these applications in the Android SDK command-line environment for Microsoft Windows and Linux. In other words, this chapter covers the creation process in Eclipse, and Chapter 6 covers the creation process using the command-line tools. Therefore, before continuing, you should check that your Eclipse environment is correctly configured. Review the steps in Chapter 3 for setting the PATH statement for the Android SDK. You should also ensure that the JRE is correctly in your PATH statement.

TIP

If you have configuration-related issues while attempting to work with any of the command-line examples, try referring to the configuration steps in Chapters 2 and 3 and look at the Android SDK documentation.

Creating Your First Android Project in Eclipse

To start your first Android project, open Eclipse. When you open Eclipse for the first time, it opens to an empty development environment (see Figure 5-1), which is where you want to begin. Your first task is to set up and name the workspace for your application. Choose File | New | Android Project, which will launch the New Android Project wizard.

CAUTION

Do not select Java Project from the New menu. While Android applications are written in Java, and you are doing all of your development in Java projects, this option will create a standard Java application. Selecting Android Project enables you to create Android-specific applications.

If you do not see the option for Android Project, this indicates that the Android plugin for Eclipse was not fully or correctly installed. Review the procedure in Chapter 3 for installing the Android plugin for Eclipse to correct this.

The New Android Project wizard creates two things for you:

- A shell application that ties into the Android SDK, using the android.jar file, and ties the project into the Android Emulator. This allows you to code using all of the Android libraries and packages, and also lets you debug your applications in the proper environment.

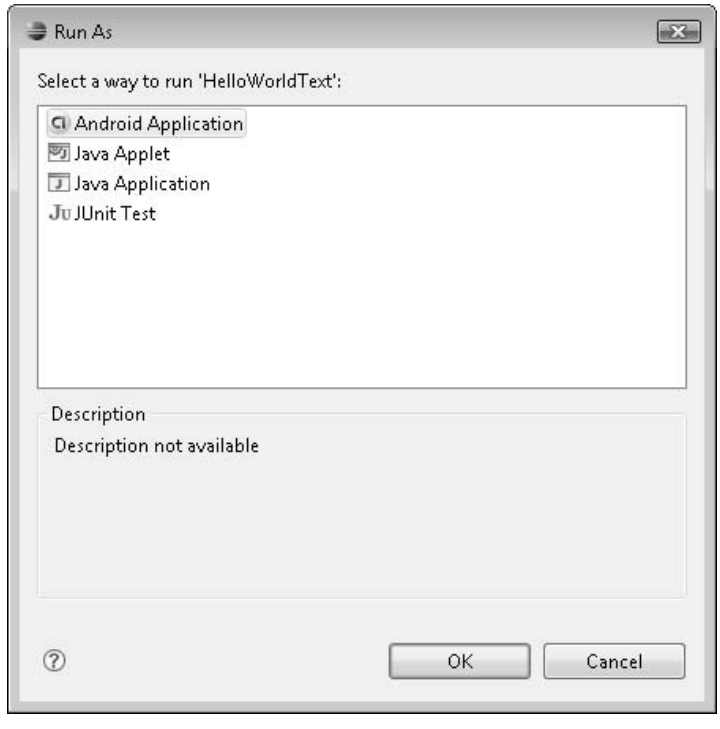
The next line in the file is the one that really does some perceptible action:

```
setContentView(R.layout.main);
```

The method setContentView( ) sets the Activity's content to the specified resource. In this case, we are using the main.xml file from the layout directory via the pointer in the R.java file. The main.xml file, right now, contains nothing more than the size of the HelloWorldText screen and a TextView. The TextView is derived from View and is used to display text in an Android environment. Reviewing the contents of main.xml, you can see that it contains the following line:

```
android:text="Hello World, HelloWorldText"
```

Considering that the setContentView( ) method is being told to set main.xml as the current View, and main.xml contains a TextView that says "Hello World, HelloWorldText," it may be safe to assume that compiling and running HelloWorldText now will give you your Hello World! application. To do this, run your unaltered HelloWorldText application. Choose Run | Run to open the Run As dialog box, select Android Application, and click OK.



The new project you just established contains the code to create a Hello World! application on its own. However, that is not very engaging, nor does it teach you very much about programming an Android application. You need to dissect the project and see exactly how the project displayed the "Hello World!" message.

What happened when you created the new Android project is that the Android plugin modified main.xml. This is a perfect example of one way to modify the UI in Android. The following lines of code are added to main.xml by the Android SDK when the project is created:

```
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Hello World, HelloWorldText"
/>
```


While I have discussed the existence of this TextView in the xml, I have not yet discussed why it works without any corresponding code. I mentioned earlier in this book that there are two ways to design a UI for Android: through the code, and through the main.xml file. The preceding code sample creates a TextView in xml and sets the text to "Hello World, HelloWorldText." Edit this line of the main.xml file to read as follows:

```
android:text="This is the text of an Android TextView!"
```

Rerun the project, and your results should appear as they do in this illustration.

Take some time and experiment with the xml TextView. Then you can move on to another way of creating a Hello World! application.

Hello World! Again

In this section, you will create another Hello World! application for Android. However, this time you will program the UI in code rather than by using the xml file—and you will actually do most of the work. The first step here is to remove the TextView code that is in main.xml. The following section of code represents the TextView. Removing it essentially makes your application an empty shell.

```
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Hello World, HelloWorldText"
/>
```


After you have removed the TextView code, your main.xml file should look like this:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >

</LinearLayout>
```

Now that you have a clean main.xml file, and thus a clean application shell, you can begin to add the code that will display “Hello World!” on the screen. Start by opening the HelloWorldText.java file and removing the following line:

```
setContentView(R.layout.main);
```

NOTE

You still need to set a ContentView for your new application; however, you are going to implement it slightly differently from how it is implemented here, so it is best to just remove the entire statement for now.

This line uses setContentView( ) to draw the main.xml file to the screen. Since you will not be using main.xml to define your TextView, you will not be setting it to your view. Instead, you will be building the TextView in code.

Key Skills & Concepts

- Using the Android SDK command-line tools
- Creating a command environment
- Navigating the Android server with a shell
- Using the Android SDK in Linux

So far this book has covered some very broad subjects to get you up and running on the Android platform. At this point, you should be fairly comfortable using Eclipse to create and run a small Android application. You created a new project, edited the main.xml and <activity>.java files, and recompiled the R.java file. These are the basic skills that you need to create Android applications.

In this chapter, you are going to expand and round out those skills by experimenting with command-line application development. Android development does not have to be limited to the confines of the Eclipse IDE. The Android SDK offers a host of command-line tools that can help you develop full applications without the need of a graphical IDE. You will use these command-line tools to create, compile, and run a Hello World! application, first in Windows and then in Linux.

Creating a Shell Activity Using the Windows CLI

The Android SDK comes with multiple tools to help you create and compile Android applications. These tools are in place to help users who do not wish to, or do not have a system capable of supporting, work within a GUI IDE. However, if you are doing all of your Android development work within Eclipse, you still should be aware of the Android SDK command-line tools and their functionality.

When you run Android-related functions, such as creating an Android project or running an application in the Android Emulator, you are actually calling connections to the Android command-line tools. These Android command-line tools, whether run from a command-line interface or from a GUI IDE, are the real core to the functionality of the Android SDK.

```
<property name="zip"         value="zip" />

<!-- Rules -->

<!-- Create the output directories if they don't exist yet. -->
<target name="dirs">
    <mkdir dir="${outdir}" />
    <mkdir dir="${outdir-classes}" />
</target>

<!-- Generate the R.java file for this project's resources. -->
<target name="resource-src" depends="dirs">
    <echo>Generating R.java...</echo>
    <exec executable="${aapt}" failonerror="true">
        <arg value="compile" />
        <arg value="-m" />
        <arg value="-J" />
        <arg value="${outdir-r}" />
        <arg value="-M" />
        <arg value="AndroidManifest.xml" />
        <arg value="-S" />
        <arg value="${resource-dir}" />
        <arg value="-I" />
        <arg value="${android-jar}" />
    </exec>
</target>

<!-- Generate java classes from .aidl files. -->
<target name="aidl" depends="dirs">
    <apply executable="${aidl}" failonerror="true">
       <arg value="-p${android-framework}" />
        <arg value="-I${srcdir}" />
        <fileset dir="${srcdir}">
            <include name="**/*.aidl"/>
        </fileset>
    </apply>
</target>

<!-- Compile this project's .java files into .class files. -->
<target name="compile" depends="dirs, resource-src, aidl">
    <javac encoding="ascii" target="1.5" debug="true" extdirs=""
            srcdir="."
            destdir="${outdir-classes}"
            bootclasspath="${android-jar}" />
</target>
<!-- Convert this project's .class files into .dex files. -->
```

```
    <!-- Invoke the proper target depending on whether or not
         an assets directory is present. -->
    <!-- TODO: find a nicer way to include the "-A ${asset-dir}" argument
         only when the assets dir exists. -->
    <target name="package-res">
        <available file="${asset-dir}" type="dir"
                property="res-target" value="and-assets" />
        <property name="res-target" value="no-assets" />
        <antcall target="package-res-${res-target}" />
    </target>

    <!-- Put the project's .class files into the output package file. -->
    <target name="package-java" depends="compile, package-res">
        <echo>Packaging java...</echo>
        <jar destfile="${out-package}"
                basedir="${outdir-classes}"
                update="true" />
    </target>

    <!-- Put the project's .dex files into the output package file.
         Use the zip command, available on most Unix/Linux/MacOS systems,
         for preference the new package (Ant 1.7) has an internal zip command,
         however Ant 1.6.5 lacks it, and is still widely installed.)
    -->
    <target name="package-dex" depends="dex, package-res">
        <echo>Packaging dex...</echo>
        <exec executable="${zip}" failonerror="true">
            <arg value="-qj" />
            <arg value="${out-package}" />
            <arg value="${intermediate-dex}" />
        </exec>
    </target>

    <!-- Create the package file for this project from the sources. -->
    <target name="package" depends="package-dex" />

    <!-- Create the package and install package on the default emulator -->
    <target name="install" depends="package">
        <echo>Sending package to default emulator...</echo>
        <exec executable="${adb}" failonerror="true">
            <arg value="install" />
            <arg value="${out-package}" />
        </exec>
    </target>

</project>
```


Now that you have a good understanding of how build.xml is used in the manual, command-line creation of Android projects, you can begin to edit your project files and

Key Skills & Concepts

- Using Intents
- Creating code that interacts with the phone hardware
- Learning the difference between dialing and calling

The chapters up to this point have introduced you to the basics of Android programming. You have examined the outline of an Android application and installed your first applications to the Android server. You have learned how to use Views and setContentView( ), as well as how to create a UI in XML. These skills have helped you to create a static application. What you have not done yet is use the application interface to interact with the hardware that the platform was created for—the cell phone.

You should not lose sight of the fact that the platform for which Android was created is, in essence, still a cell phone. The underlying hardware for the devices that Android will run on is designed for the purpose of person-to-person communication. If you strip away all the bells and whistles that the Android SDK is capable of adding to the cell phone, it must still be able to send and receive phone calls. For this reason, this chapter focuses on the code that enables you to interact with the phone hardware.

By the end of this chapter, you should have the skills needed to interact with some of the basic functions of the phone. You will be able to work with the dialer to send and receive calls. These tools and skills will be your keys to creating useful applications on this flexible platform.

You are reading this book because you intend to design applications that run on a cell phone, so it stands to reason that you should learn how Android allows for interaction with the phone hardware—in particular, the process that enables the phone to send and receive calls.

When we think of a cell phone, a few basic functions come to mind. The first, and most obvious, of which is the ability to send and receive phone calls. This is inarguably the quintessential function of a cell phone. There are a few peripheral features that make the cell phone easier to use, such as the ability to keep and manage contacts and the ability to store and review missed calls. As you'll read in this chapter, you can access and manipulate the code for all of these functions.

Broadcast Intent	Message
SIM_STATE_CHANGED_ACTION	The state of the SIM card has changed
TIME_CHANGED_ACTION	The device's time was set
TIME_TICK_ACTION	The current time has changed
TIMEZONE_CHANGED_ACTION	The device's timezone has changed
UMS_CONNECTED_ACTION	The device has connected via USB
UMS_DISCONNECTED_ACTION	The device has been disconnected from its USB host
WALLPAPER_CHANGED_ACTION	The device's wallpaper has been changed

Table 7-2 Broadcast Intents *(continued)*

NOTE

Some of these Broadcast Intents are sent quite often, such as TIME_TICK_ACTION and SIGNAL_STRENGTH_CHANGED_ACTION. Be careful how you use them. You should try not to receive such broadcasts if at all possible.

The Intent is only one-third of the picture. An Intent is really just that, an intent to do something; an Intent cannot actually do anything by itself. You need Intent Filters and Intent Receivers to listen for, and interpret, the Intents.

An Intent Receiver is like the mailbox of an Activity. The Intent Receiver is used to allow an Activity to receive the specified Intent. Using the previous web browser example, the Web Browser Activity is set up to receive web browser Intents. A system like this allows unrelated Activities to ignore Intents that they would not be able to process. It also allows Activities that need the assistance of another Activity to utilize that Activity without needing to know how to call it.

With Intents and Intent Receivers, one Activity can send out an Intent and another can receive it. However, there needs to be something that governs the type of information that can be sent between the two Activities. This is where Intent Filters come in.

Intent Filters are used by Activities to describe the types of Intents they want to receive. More importantly, they outline the type of data that should be passed with the Intent. Therefore, in our example scenario, we want the web browser to open a web page. The Intent Filter would state that the data passed with the WEB_SEARCH_ACTION Intent should be in the form of a URL.

In the next section, you will begin to use Intents to open and utilize the phone's dialer.

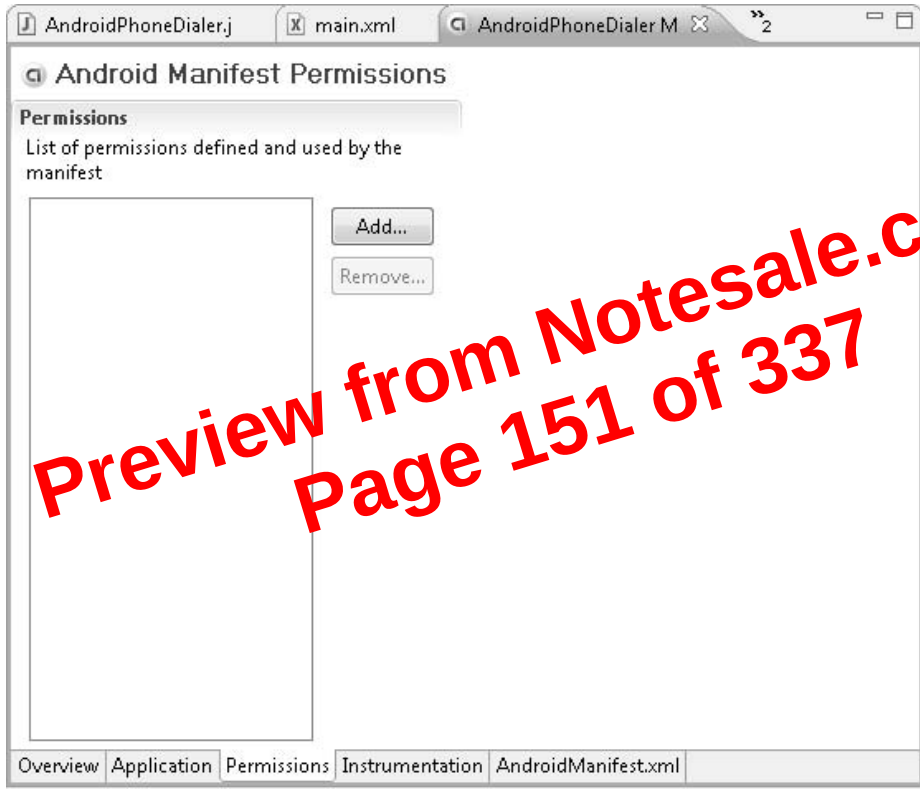
```
package android_programmers_guide.AndroidPhoneDialer;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.net.Uri;

public class AndroidPhoneDialer extends Activity {
    /** Called when the Activity is first created. */
    @Override
    public void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        setContentView(R.layout.main);
       /** Create our Intent to call the Dialer */
       /** Pass the Dialer the number 5551212 */
        Intent DialIntent = new
Intent(Intent.DIAL_ACTION,Uri.parse("tel:5551212"));
       /** Use NEW_TASK_LAUNCH to launch the Dialer Activity */
        DialIntent.setFlags(Intent.NEW_TASK_LAUNCH );
       /** Finally start the Activity                    */
        startActivity(DialIntent);
    }
}
```


You should now compile AndroidPhoneDialer and run it on your Emulator. The process for compiling and running applications was covered in previous chapters, so you should be familiar with that process. Once you run your application, the Emulator should launch. After the lengthy boot process, your Activity will launch.

TIP

It is a good idea to keep the Emulator running, even after you are finished with your Activity and have returned to the code window. It is most people's instinct to close the Emulator window when they have finished testing their Activity. However, I have found that leaving the Emulator open helps with two major issues. The first is the amount of time it takes for the Emulator to start. By leaving the Emulator open, you avoid the lengthy load time. Second, I have noticed that there are times when I make minor changes to an Activity and they are not copied to the Emulator. Leaving the Emulator open seems to alleviate this issue as well. If you continue to have issues in the Emulator, remove the userdata-qemu.img file from your computer. This allows the Emulator to start up with a clean image.

To edit the Activity's permissions, click the Permission link. This should take you to the Android Manifest Permissions window, shown in the following illustration.



This window lists the permissions that are currently assigned to your Activity. Given that you are working in a new project, you do not have any assigned permissions. Therefore, click the Add button to begin the process. In the dialog box that opens, select Uses Permission and click OK.

First, lay out the Views in your main.xml. You will actually add two Views here: a TextView to act as a label and give some direction to the user, and an EditText to accept the user's input. Together these two Views will add the needed depth and practicality to your Activity.

As you form the look of your Activity, keep in mind that the .xml file is formed visually. This means that if you want the TextView to appear above the EditText on the finished Activity, you should place it before the EditText in main.xml.

Because you have used TextViews a few times now, creation of this View will not get too involved. Simply take a look at the attributes that you set in your TextView:

```
<TextView android:id="@+id/textLabel"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Enter Number to Dial:"
/>
```


There's nothing out of the ordinary here. This is just a simple TextView with the text Enter Number to Dial:. This TextView will serve as a label for your EditView. Here's how you set the attributes for the EditView.

```
<EditText android:id="@+id/phoneNumber"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
/>
```

NOTE

You do not have to set the android:text attribute because you do not need any default text.

The id is set to phoneNumber, which is the name you will use to refer to the EditText View in the code. Again, there should be no surprises when setting up main.xml. Your final file should look like this:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android=http://schemas.android.com/apk/res/android
    android:orientation="vertical"
    android:layout_width="fill_parent"
```

Once your phoneNumber EditText is created, you can use it to reference the text that is input on the device. All you have to do now is call phoneNumber.getText( ) to retrieve the user's input. Replace the hard-coded value "tel:5551212" in the following line,

```
Intent(Intent.CALL_ACTION,Uri.parse("tel:5551212"));
```

with the value of getText( ):

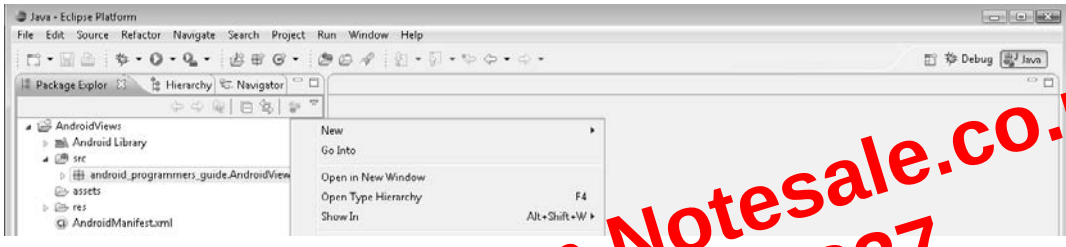
```
Intent(Intent.CALL_ACTION,Uri.parse("tel:" + phoneNumber.getText()));
```

That is all the new code you need to update your project. With these simple two additions, you can give the user an object with which to input a phone number, and send that number to the phone's Call Activity. The full code in the .java file should look like this:


```
package android_programmers_guide.AndroidPhoneDialer;

import android.app.Activity;
import android.os.Bundle;
import android.widget.Button;
import android.view.View;
import android.content.Intent;
import android.net.Uri;
import android.widget.EditText;

public class AndroidPhoneDialer extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        setContentView(R.layout.main );
      final EditText phoneNumber = (EditText) findViewById(R.id.phoneNumber
);
      final Button callButton = (Button) findViewById(R.id.callButton);
      callButton.setOnClickListener(new Button.OnClickListener() {
        public void onClick(View v){
              Intent CallIntent = new
Intent(Intent.CALL_ACTION,Uri.parse("tel:" + phoneNumber.getText()));
            CallIntent.setLaunchFlags(Intent.NEW_TASK_LAUNCH );
            startActivity(CallIntent);

        }

      });
   }
}
```

Intent Code for the .java File

Using the Package Explorer again, navigate to the src directory, open it, and right-click the android_programmers_guide.AndroidViews package, as shown in the following illustration.



Once again, you are going to add a new file to the folder. After you right-click the AndroidViews package, select New | File from the context menu. This file will hold all the code for the second Activity in this project. Name the file **test.java**. You should now have a nice, new (but empty) .java file. You just need to add a few lines of code to the file to make it usable:

```
package testPackage.test;
import android.app.Activity;
import android.os.Bundle;
public class test extends Activity {
    /** Called when the Activity is first created. */
    @Override
    public void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        setContentView(R.layout.test);
        /** This is our Test Activity
              All code goes below     */
    }
}
```

Notice that you call test.xml in the setContentView method, using the context R.layout.*test*. This line tells the new Activity to use the .xml file you created as the layout file for this "page."

Modifying the AndroidManifest.xml

Open your AndroidManifest.xml file in Eclipse. AndroidManifest.xml has not been discussed in great detail in this book. AndroidManifest.xml contains the global settings for your project. More importantly, AndroidManifest.xml also contains the Intent Filters for your project.

Chapter 7 discussed how Android uses the Intent Filters to marshal what Intents can be accepted by what Activities. The information that facilitates this process is kept in AndroidManifest.xml.

NOTE

There is only one AndroidManifest.xml file per project.

If your AndroidManifest.xml file is currently open, it should appear as follows:

```
<activity android:name=".AndroidViews" android:label="@string/app_name">
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
   </intent-filter>
</activity>
```

What you are looking at here is the Intent Filter for the AndroidViews Activity, the main Activity that was created with the project. To this file you can add any other Intent Filters that you want your project to handle. In this case, you want to add an Intent Filter that will handle the new Test Activity that you created.

The following is the code for the Intent Filter that you need to add to the AndroidManifest.xml file:

```
<activity android:name=".Test" android:label="Test Activity">
<intent-filter>
   <action android:name="android.intent.action.MAIN" />
   <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
```

Adding this code to AndroidManifest.xml enables Android to pass Intents for the Test Activity to the correct place. The full AndroidManifest.xml file should look like this:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android=http://schemas.android.com/apk/res/android
```

```
        setContentView(R.layout.main);
    }
}
```

As with everything you add to your Activities, you need to import a new package to create your menu. Import the android.view.Menu into your AndroidViews Activity:

```
Import android.view.Menu;
```

To create the Menu, you need to override the onCreateOptionsMenu( ) method of the Activity. The method onCreateOptionsMenu( ) is a Boolean method that is called when the user first selects the Menu Button. You will use this method to build your Menu and add your selection items to it. Add the following code to AndroidViews.java:

```
@Override
  public boolean onCreateOptionsMenu(Menu menu) {
 super.onCreateOptionsMenu(menu);
}
```


You will add the code to create the Menu inside the onCreateOptionsMenu( ) method. The items that you need to add to the Menu are the Views that you are going to create in this project. The following is the list of View names that you will need to add to the Menu:

- AutoComplete
- Button
- CheckBox
- EditText
- RadioGroup
- Spinner

In the preceding code that you created to override the onCreateOptionsMenu( ) method, you passed in a Menu variable called *menu*. This variable represents the actual menu item that is created on the Android interface.

To add your list of items to the Menu, you will use the *menu*.add( ) method. The syntax for this call is as follows:

```
menu.add(<group>,<id>,<title>)
```

CheckBox

In this section you will be creating an Activity for the CheckBox View. The steps for creating the Activities are identical to those in the preceding sections. Therefore, you will be provided with the full code of the three main Activity files—AndroidManifest.xml, checkbox.xml, and testCheckBox.java. These files are provided for you in the following sections.

AndroidManifest.xml

This section contains the full code of the current AndroidViews' AndroidManifest.xml. If you are following along in Eclipse, modify your Activity's AndroidManifest.xml to look as follows:


```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android=http://schemas.android.com/apk/res/android
    package="android_programmers_guide.AndroidViews">
    <application android:icon="@drawable/icon"
                 android:name=".AndroidViews"
android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".AutoComplete" android:label="AutoComplete">
            <intent-filter>
                  <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
         <activity android:name=".testButton" android:label="TestButton">
            <intent-filter>
                  <action android:name="android.intent.action.MAIN" />
                  <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <activity android:name=".testCheckBox" android:label="TestCheckBox">
            <intent-filter>
                  <action android:name="android.intent.action.MAIN" />
                  <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>
</manifest>
```

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
 super.onCreateOptionsMenu(menu);

 menu.add(0, 0, "AutoComplete");
 menu.add(0, 1, "Button");
 menu.add(0, 2, "CheckBox");
 menu.add(0, 3, "EditText");
 menu.add(0, 4, "RadioGroup");
 menu.add(0, 5, "Spinner");
 return true;
}
@Override
public boolean onOptionsItemSelected(Menu.Item item){
 switch (item.getId()) {
 case 0:
     showAutoComplete();
     return true;
 case 1:
     showButton();
     return true;
 case 2:
     showCheckBox();
     return true;
 case 3:
     showEditText();
     return true;
 case 4:
     showRadioGroup();
     return true;
 case 5:
     showSpinner();
     return true;
 }
 return true;
}
public void showButton() {
      Intent showButton = new Intent(this, testButton.class);
      startActivity(showButton);
}
public void showAutoComplete(){
      Intent autocomplete = new Intent(this, AutoComplete.class);
```

```
        startActivity(autocomplete);
}
public void showCheckBox(){
        Intent checkbox = new Intent(this, testCheckBox.class);
        startActivity(checkbox);
 }
public void showEditText() {
        Intent edittext = new Intent(this, testEditText.class);
        startActivity(edittext);
 }
}
```

Launch your application and select the EditText option from the Main (shown earlier in Figure 8-1).

The following illustration shows what the EditText Activity looks like.

Click the Change Layout and Change Test Color Buttons. The results are depicted in the following illustrations.

```
    package="android_programmers_guide.AndroidViews">
    <application android:icon="@drawable/icon">
        <activity android:name=".AndroidViews"
android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".AutoComplete" android:label="AutoComplete">
            <intent-filter>
                  <action android:name="android.intent.action.MAIN" />
                  <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
        </activity>
         <activity android:name=".testButton" android:label="TestButton">
            <intent-filter>
                  <action android:name="android.intent.action.MAIN" />
                  <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <activity android:name=".testCheckBox" android:label="TestCheckBox">
            <intent-filter>
                  <action android:name="android.intent.action.MAIN" />
                  <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <activity android:name=".testEditText" android:label="TestEditText">
            <intent-filter>
                  <action android:name="android.intent.action.MAIN" />
                  <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
                <activity android:name=".testRadioGroup" android:label="Test
RadioGroup">
            <intent-filter>
                  <action android:name="android.intent.action.MAIN" />
                  <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    <activity android:name=".testSpinner" android:label="Test Spinner">
             <intent-filter>
                    <action android:name="android.intent.action.MAIN" />
                    <category android:name="android.intent.category.LAUNCHER" />
             </intent-filter>
        </activity>
    </application>
</manifest>
```

Key Skills & Concepts

- Using the Android location-based service APIs
- Obtaining coordinate data from the GPS hardware
- Changing your Activity's look and feel with a RelativeLayout
- Using a MapView to plot your current location
- Using Google Maps to find your current location

In this chapter, you are going to learn about the Android Location-Based API. This chapter is invaluable if you want to leverage the ability of Android to work with the Global Positioning System (GPS) hardware of a device. You will use the Android Location-Based API to collect your current position and display that location to the screen. Toward the end of this chapter, you will use Google Maps to display your current location on your cell phone.

You will also learn some new techniques that will add some depth and creativity to your Activities. Resources such as RelativeLayouts and small buttons will let you create more user-friendly and visually appealing Activities for Android.

In the first section of this chapter, you will learn about using your device's GPS hardware to obtain your current location. However, before you jump into that section, you need to create your project for this chapter. Create a new Project in Eclipse and name it **AndroidLBS**.

Using the Android Location-Based API

The Android SDK contains an API that is specifically geared to help you interface your Activity with any GPS hardware that may be in your device. This chapter assumes that your device will include GPS hardware.

CAUTION

Just as Android-based cell phones are not required to include a camera, they are not required to include GPS hardware either, although many models likely will include both a camera and GPS hardware. Android included the Android Location-Based API in anticipation that GPS hardware will be included in many cell phones.

Because you are working on a software-based emulator, and not on a real device, the presence of GPS hardware has to be simulated. In this case, Android provides a file in the adb server that simulates having GPS hardware. The file is located at

```
data/misc/location/<provider>
```

where <provider> represents the location information provider. The provider that Android supplies to you is

```
data/misc/location/gps
```

TIP

You can have multiple providers to simulate different scenarios. Therefore, you can create a provider named *test* or *gps1*, whichever you prefer.

Within the specific provider's folder could be any number of files that will hold the sample coordinates that you want Android to use. When you are using the Android Emulator, you can use the following types of files to store/retrieve GPS style coordinates. Each of these file types has a different format for providing information to the Android Location-Based API.

- kml
- nmea
- track

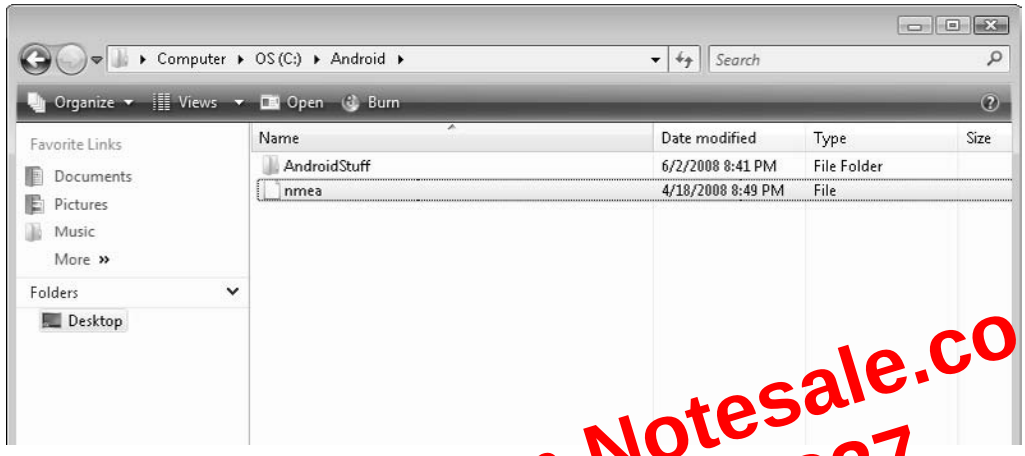
Let's take a look at what each of these files does and how they differ from each other.

Creating a kml File

A .kml file is a Keyhole Markup Language file. These files were originally developed for, and can be created by, Google Earth. The Android Location-Based API can parse a .kml file for coordinates to simulate a GPS.

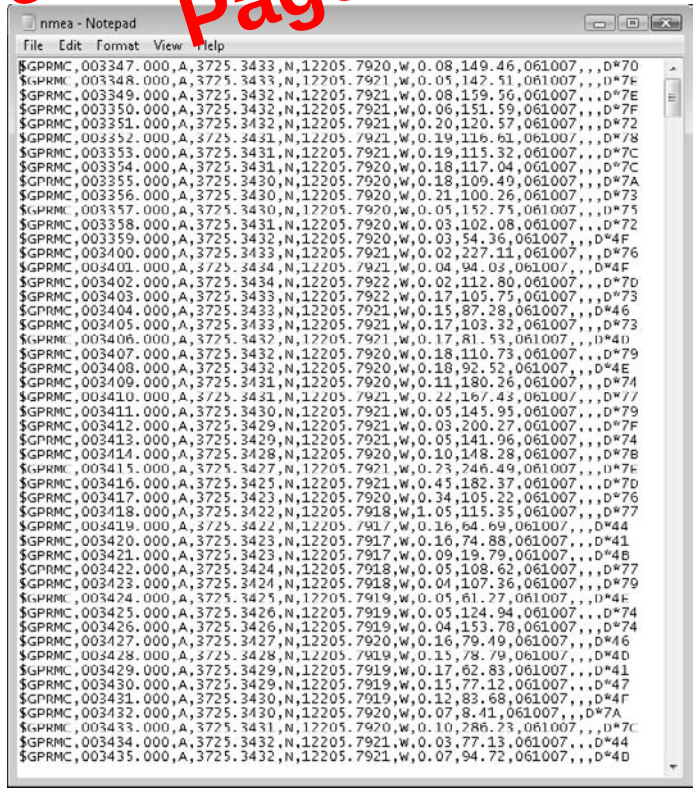
NOTE

If you do not have Google Earth, it is a free download from Google. Installing it may be worth your time if you want to develop more Android Location-Based API Activities.



Now that you have the file placed on your desktop, associate it with Notepad.

Finally, open the file to examine its contents. You should see many rows of coordinate data, as shown here.



```
import android.location.LocationManager;
```

Next, create the code for the Button. The goal is to retrieve the current coordinate information from the GPS. You have created a few Buttons already in this book, and the format for this one is no different. You need to set up your Button and load its layout from main.xml. Then you can set up the onClick event to call a function, LoadCoords( ).

```
final Button gpsButton = (Button) findViewById(R.id.gpsButton);
gpsButton.setOnClickListener(new Button.OnClickListener() {
public void onClick(View v){
                    LoadCoords();
              }});
```

The final step to create this Activity is to fill out the code of the LoadCoords( ) function. Create the TextViews that you will post your coordinates to:

```
TextView latText = (TextView) findViewById(R.id.latText);
TextView lngText = (TextView) findViewById(R.id.lngText);
```


NOTE

You do not have to create the two TextViews that you will use as labels because you will not be posting anything to them.

Now create a LocationManager from which you can pull the coordinate values. The important part of this instantiation is that you must pass the LocationManager a context; use the LOCATION_SERVICE:

```
LocationManager myManager =
(LocationManager)getSystemService(Context.LOCATION_SERVICE);
```

To pull the coordinates from myManager, use the getCurrentLocation( ) method. This method needs one parameter, a *provider*, which represents the location that the API will pull the coordinates from. In this case, Android has provided a mock location *gps* that contains the nmea file discussed earlier in this chapter:

```
Double latPoint = myManager.getCurrentLocation("gps").getLatitude();
Double lngPoint = myManager.getCurrentLocation("gps").getLongitude();
```

```
    android:layout_height="fill_parent"
    >
  <view class="com.google.android.maps.MapView"
        android:id="@+id/myMap"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
</RelativeLayout>
```

Now you can add your two additional buttons. Place the buttons so that they appear in the upper-left and lower-left corners of the MapView. You need to make one change to the standard Button layout. By default, the RelativeLayout adds the Button to align with the top edge of the anchor view, in this case, the MapView. Therefore, in the layout, use the android:layout_alignBottom attribute and assign it the id of the MapView. This will align the button to the bottom of the map.

```
<Button android:id="@+id/buttonZoomIn"
     style="?android:attr/buttonStyleSmall"
            android:text="+"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
            <Button android:id="@+id/buttonZoomOut"
            style="?android:attr/buttonStyleSmall"
            android:text="-"
            android:layout_alignBottom="@+id/myMap"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
```


TIP

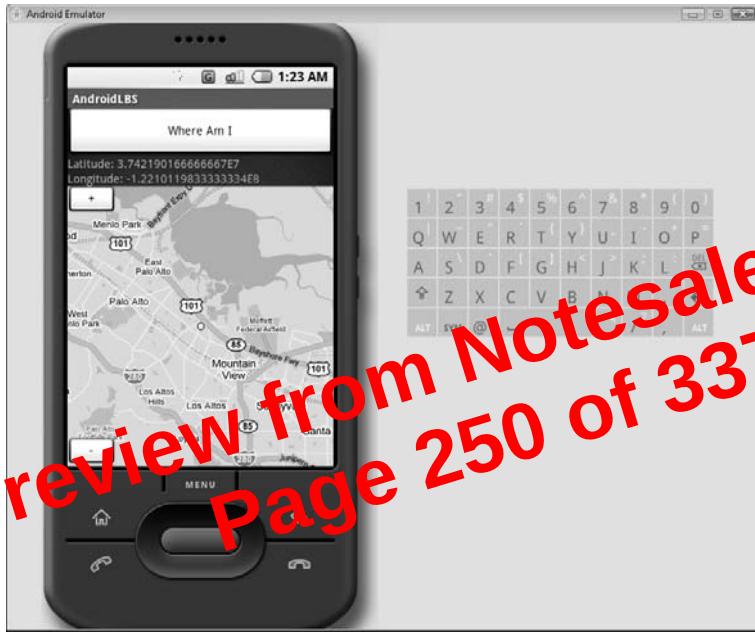
Take a close look at the layout attributes for the Button layout. I use a new attribute, *style*, to make this Button a small button.

Your full main.xml file should look like this:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<Button
      android:id="@+id/gpsButton"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Where Am I"
    />
```

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    >
<TextView
      android:id="@+id/latLabel"
      android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Latitude: "
    />
<TextView
      android:id="@+id/latText"
      android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    />
 </LinearLayout>
 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    >
 <TextView
      android:id="@+id/lngLabel"
      android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Longitude: "
    />
 <TextView
      android:id="@+id/lngText"
      android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    />
</LinearLayout>

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
  <view class="com.google.android.maps.MapView"
        android:id="@+id/myMap"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
          <Button android:id="@+id/buttonZoomIn"
            style="?android:attr/buttonStyleSmall"
            android:text="+"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
            <Button android:id="@+id/buttonZoomOut"
            style="?android:attr/buttonStyleSmall"
            android:text="-"
```

Test the zoom in and zoom out buttons. When you zoom in, you should see something that looks similar to the following illustration.



Try This Toggling Between MapView's Standard and Satellite Views

Edit the AndroidLBS Activity one more time. You should add two more buttons to the RelativeLayout. These buttons should toggle the MapView between standard view and satellite view. Here are some points to consider:

- Add the toggle buttons to the opposite corners of the MapView using the align layout attributes.
- Research the MapView to find the toggling method.
- Create a function that you can pass the MapView to and toggle it.

The complete text of solution main.xml and AndroidLBS.java are as follows.

```
public void ZoomIn(MapView mv, MapController mc){
  if(mv.getZoomLevel()!=21){
  mc.zoomTo(mv.getZoomLevel()+ 1);
  }
}
public void ZoomOut(MapView mv, MapController mc){
  if(mv.getZoomLevel()!=1){
        mc.zoomTo(mv.getZoomLevel()- 1);
        }
}
public void ShowMap(MapView mv){
        if (mv.isSatellite()){
              mv.toggleSatellite();
        }
}
public void ShowSat(MapView mv){
        if (!mv.isSatellite()){
              mv.toggleSatellite();
        }
}
}
```


When you run your Activity, you should be able to toggle the satellite view on and off, as shown in the following illustrations.

Key Skills & Concepts

- Implementing a Google API package
- Configuring the XMPP development settings for Google access
- Implementing the View.OnClickListener( ) method

Chapter 9 introduced you to the Google API. You created an Activity that leveraged the Google API and Google Maps. Because of the ease and flexibility of the API, you were able to quickly display a Google Map of a user's current location. You also learned how to manipulate that map with relatively few lines of code.

The Google API contains more than just hooks into Google Maps. You used a small part of a much larger API in the last chapter. The base package for the Google API is com.google. From this base, the Google API contains packages that allow you to create Activities that leverage the power of GTalk (Google's chat service), Google Calendar, Google Docs, Google Spreadsheet, and Google Services.

When I started writing this book, the version of the Android SDK was m3-rc22. By the time I completed writing, Google had released m5-15. In the time between these two releases, Google had deprecated a few of these packages—while leaving them in the SDK.

Google Calendar, Google Spreadsheet, and Google Services appear to be undergoing an upgrade that, unfortunately, leaves them in a state of incompletion for the m5-rc15 release of the SDK. Google also removed any associated help files from the SDK for these packages, to avoid any confusion. Therefore, the focus of this chapter is a package that works quite well with the latest release of the Android SDK—GTalk.

In this chapter, you will build a small Activity that utilizes the GTalk package of the Android SDK. When the Activity is complete, you will be able to send GTalk messages from your phone to other GTalk users and receive messages from them.

NOTE

In the first iteration of the Google API for Android, the package dealing with GTalk was a much broader XMPP package. (XMPP is the protocol on which many chat platforms are based, including GTalk and Jabber.) With the latest release of the SDK, the original XMPP package was tightened up and renamed to reflect the specificity of GTalk.

To get started, create a new Project in Eclipse and name **GoogleAPI**.

```
<EditText
      android:id="@+id/messageText"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:textSize="16sp"
      android:minWidth="250dp"
      android:scrollHorizontally="true" />
<Button
      android:id="@+id/btnSend"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Send Msg">
</Button>
</LinearLayout>
```

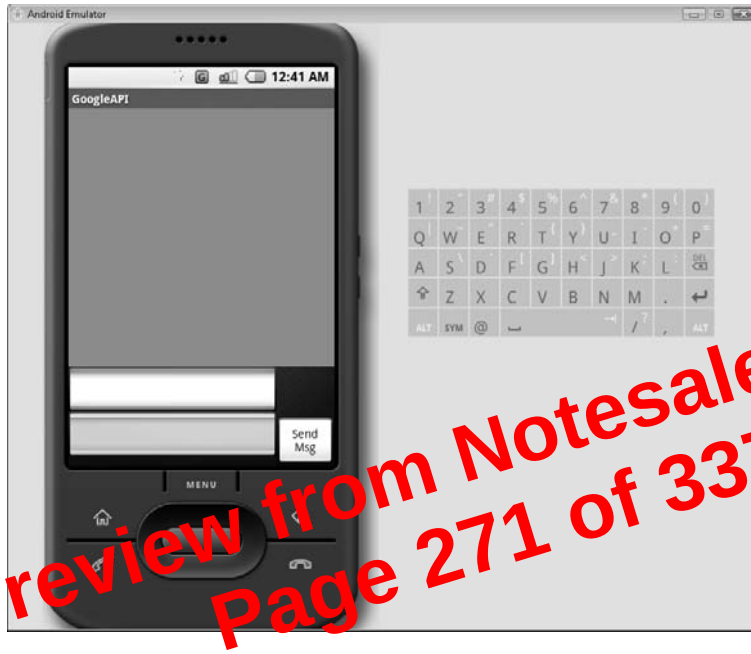
This layout will line up your Views so that they fall inline with each other. Place this LinearLayout in your main LinearLayout. Your final GoogleAPI.xml file should look like this:


```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent">
<ListView
      android:id="@+id/messageList"
      android:layout_width="fill_parent"
      android:layout_height="0dip"
      android:scrollbars="vertical"
      android:layout_weight="1"
      android:drawSelectorOnTop="false" />
<EditText
      android:id="@+id/messageTo"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:textSize="16sp"
      android:minWidth="250dp"
      android:scrollHorizontally="true" />
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="horizontal"
      android:layout_width="fill_parent"
```

```
        this.bindService(new
Intent().setComponent(GTalkServiceConstants.GTALK_SERVICE_COMPONENT),
connection, 0);

    }
    private ServiceConnection connection = new ServiceConnection() {
      public void onServiceConnected(ComponentName name, IBinder service) {
            try {
            myIGTalkSession =
IGTalkService.Stub.asInterface(service).getDefaultSession();
            } catch (DeadObjectException e) {
                  myIGTalkSession = null;
            }
            }

            public void onServiceDisconnected(ComponentName name) {
                  myIGTalkSession = null;
            }

    };
    public void onClick(View view) {

            Cursor cursor = managedQuery(Im.Messages.CONTENT_URI, null,
                  "contact='" + messageTo.getText().toString() + "'",
null, null);
            ListAdapter adapter = new SimpleCursorAdapter(this,
                  android.R.layout.simple_list_item_1, cursor,
                  new String[]{Im.MessagesColumns.BODY},
                  new int[]{android.R.id.text1});
            this.messageList.setAdapter(adapter);

            try {
                  IChatSession chatSession;
                  chatSession =
myIGTalkSession.createChatSession(messageTo.getText().toString());
             chatSession.sendTextMessage(messageText.getText().toString());
            } catch (DeadObjectException ex) {
                  myIGTalkSession = null;
            }
        }

}
```


Compiling and Running GoogleAPI

Now, compile and run your GoogleAPI Activity in the Emulator. If your connection is successful, you should see a screen that looks like the following.



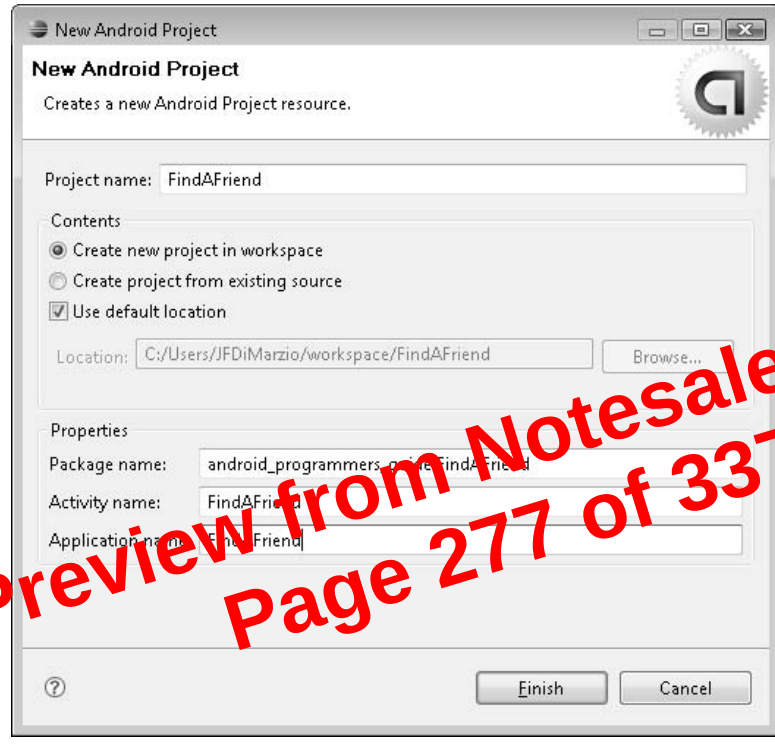
To test the Activity, I sent the message "Hello" to androidprogrammersguide@gmail.com, as shown here:

- Change the font color in the message list for messages you send as opposed to messages you receive
- Change the background color of the message list

Ask the Expert

Q: **Can the GTalk API be used to communicate with other XMPP-based chat clients?**

A: The answer to this is still unclear. The m3-rc22 version of the SDK included an XMPP API rather than the more specific GTalk API included in the m5-1.5 SDK. It is possible that these two will be combined in a future release of the Android SDK, in which case the GTalk API can be used to communicate with other XMPP-based chat clients.



While you should be fairly comfortable creating Android applications by now, you will have a little bit of help creating this project. Google includes in the Android SDK an application called NotePad, a simple interface that lets you store, modify, and delete "notes" in a database. You are going to modify some of this sample code to create the interface for your Friends database.

If you want to see how Google NotePad works, load the project into Eclipse and run it in your Android Emulator before you move on. You will begin to modify this code shortly, but first, in the following section, you will create your first SQLite database.

Creating a SQLite Database

Android devices will ship with an internal SQLite database. The purpose of this database is to give users and developers a location in which to store information that can be used in Activities.

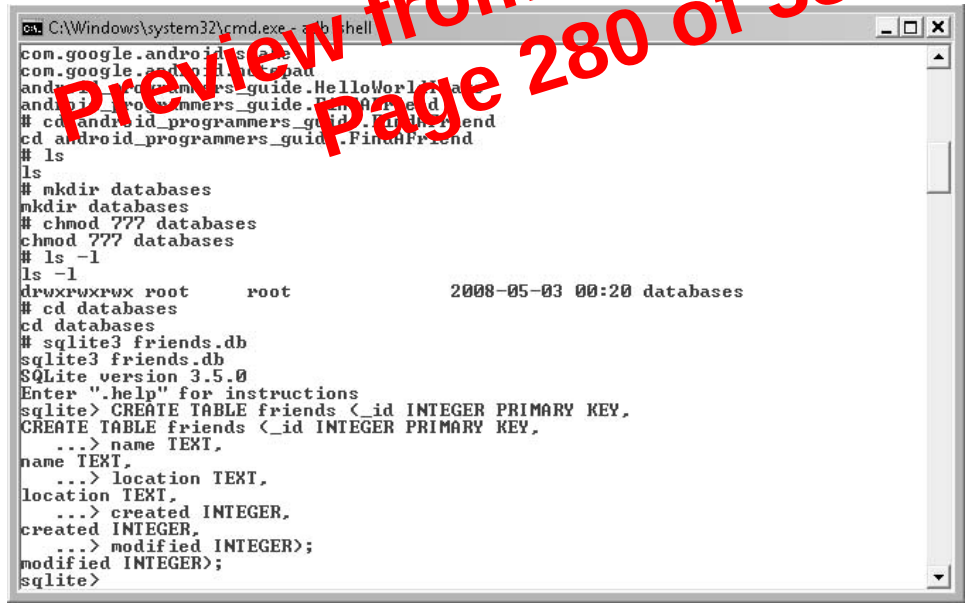
TIP

If you are not familiar with SQLite, a SQLite command must terminate with a semicolon. This is helpful if you want to span commands across prompts. Pressing the ENTER key without terminating a SQLite command will give you a continuation prompt, ...>. You can continue to enter your command at this prompt until you use the semicolon. SQLite will treat such continued commands as one full command once the semicolon is used.

To create your friends table within your database, enter the following command at the sqlite> prompt:

```
CREATE TABLE friends (_id INTEGER PRIMARY KEY, name TEXT, location TEXT,
created INTEGER, modified INTEGER);
```

If your command executes successfully, you will be returned to the sqlite> prompt, as shown in the following illustration.



Your database is now ready to be used, and you can exit SQLite. Use the command **.exit** to exit. You can then quit your shell session and return to Eclipse.

Creating the database was the first step in setting up your application. Now that the database and corresponding table are created, you need a method to access the data. The data access method employed by Android is a Content Provider. The following section walks you through creating a custom Content Provider for your new database and accessing your data.

```
    private static final int NAME_INDEX = 1;

    private static final String[] PROJECTION = new String[] {
            Friends.Friend._ID,
            Friends.Friend.NAME,
    };

    Cursor mCursor;
    EditText mText;
}
```

Next, you need to override some methods, starting with onCreate( ). You have seen this method overridden in other chapters. Typically, it holds all the code that should be executed when the Activity is created.

```
    public void onCreate(Bundle icicle) {
        super.onCreate(icicle);

        setContentView(R.layout.name_editor);

        Uri uri = getIntent().getData();

        mCursor = managedQuery(uri, PROJECTION, null, null);

        mText = (EditText) this.findViewById(R.id.name);
        mText.setOnClickListener(this);

        Button b = (Button) findViewById(R.id.ok);
        b.setOnClickListener(this);
    }
```


Notice that, in the previous code sample, you assign layouts to their respective Views and initiate some of your variables. However, you may be wondering where the data is for the name field. That is, you have created a cursor, but you have not retrieved anything from it. You will use the onResume( ) method for that.

The two methods that you will override next, onResume( ) and onPause( ), will do the work of reading from and writing to the database, respectively. Within the Android life cycle, onResume( ) is called when an Activity is open and on the top of the focus. onPause( ) is called when an Activity is being closed but before focus is handed to another Activity.

Override your onResume( ) method to read the database and retrieve the name field:

```
    protected void onResume() {
        super.onResume();
```

```
            Friends.Friend.NAME,
    };

    Cursor mCursor;
    EditText mText;

    @Override
    public void onCreate(Bundle icicle) {
        super.onCreate(icicle);

        setContentView(R.layout.name_editor);

        Uri uri = getIntent().getData();

        mCursor = managedQuery(uri, PROJECTION, null, null);

        mText = (EditText) this.findViewById(R.id.name);
        mText.setOnClickListener(this);

        Button b = (Button) findViewById(R.id.ok);
        b.setOnClickListener(this);
    }

    @Override
    protected void onResume() {
        super.onResume();

        if (mCursor != null) {
            mCursor.first();
            String title = mCursor.getString(NAME_INDEX);
            mText.setText(title);
        }
}

    @Override
    protected void onPause() {
        super.onPause();

        if (mCursor != null) {
            String title = mText.getText().toString();
            mCursor.updateString(NAME_INDEX, title);
            mCursor.commitUpdates();
        }
    }

    public void onClick(View v) {
         finish();
    }
}
```

At this point, you can edit name values in the Friends database. However, there are two fields of importance in the database, name and location. In the next section, you will create an editor for the location field.

Creating the LocationEditor Activity

In this section, you will create an editor for the location field of the Friends database. You are going to make this Activity slightly different from the NameEditor Activity. Therefore, the code will be different and follow a slightly unfamiliar process.

If you explored the Google demo NotePad, you should have noticed that the "notes" editor is a white screen with a dynamically drawn line on it that repeats itself as needed. This effect is performed using a custom View. You are going to use this same custom View for the LocationEditor.

location_editor.xml

The first step is to create location_editor.xml and LocationEditor.java files for the layout and code, respectively. The layout file should contain a call to the custom View layout. The full layout is as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<view xmlns:android="http://schemas.android.com/apk/res/android"
class="android_programmers_guide.FindAFriend.LocationEditor$MyEditText"
android:id="@+id/location"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#ffffff"
    android:padding="10dip"
    android:scrollbars="vertical"
    android:fadingEdge="vertical" />
```

The LocationEditor will also contain a menu system that will allow the user to discard, delete, or revert any changes they make. This will be a pretty complex Activity. Therefore, it is best to start at the beginning, the imports section of the LocationEditor.java.

LocationEditor.java

Take a look at the following imports for this Activity, many of which deal with drawing the custom View on the screen:

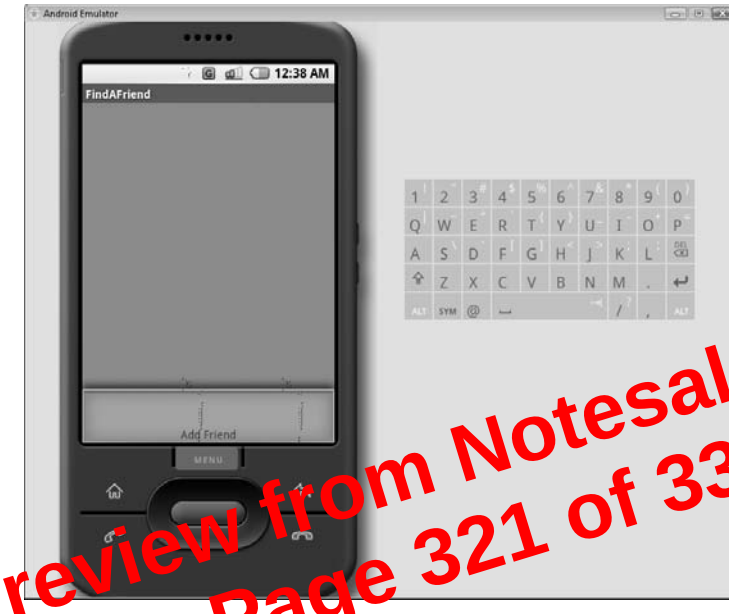
```
import android.app.Activity;
import android.content.ComponentName;
```

```
    DrawFriendsOverlay drawFriendsOverlay = new DrawFriendsOverlay();

    @Override
    public void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        setContentView(R.layout.friendsmap);

        Intent intent = getIntent();
        if (intent.getData() == null) {
            intent.setData(Friends.Friend.CONTENT_URI);
        }
        mCursor = managedQuery(getIntent().getData(), PROJECTION, null,null);

        final MapView myMap = (MapView) findViewById(R.id.myMap);
        final MapController myMapController = myMap.getController();
        LoadFriends(myMap, myMapController, mCursor);
        OverlayController myOverlayController =
myMap.createOverlayController();
        myOverlayController.add(drawFriendsOverlay, true);
        final Button zoomIn = (Button) findViewById(R.id.buttonZoomIn);
        zoomIn.setOnClickListener(new Button.OnClickListener() {
              public void onClick(View v){
                    ZoomIn(myMap,myMapController);
              }});
        final Button zoomOut = (Button) findViewById(R.id.buttonZoomOut);
        zoomOut.setOnClickListener(new Button.OnClickListener() {
              public void onClick(View v){
                    ZoomOut(myMap,myMapController);
              }});
       final Button viewMap = (Button) findViewById(R.id.buttonMapView);
       viewMap.setOnClickListener(new Button.OnClickListener() {
              public void onClick(View v){
                    ShowMap(myMap,myMapController);
              }});
       final Button viewSat = (Button) findViewById(R.id.buttonSatView);
       viewSat.setOnClickListener(new Button.OnClickListener() {
              public void onClick(View v){
                    ShowSat(myMap,myMapController);
              }});

    }

    public void LoadFriends(MapView mv, MapController mc, Cursor c){
      Point myLocation = null;
      Double latPoint = null;
      Double lngPoint = null;
      c.first();
      do{
            if (c.getString(c.getColumnIndex("location")) != null) {
           final String geoPattern = "(geo:[\\-]?[0-9]{1,3}\\.[0
9]{1,6}\\,[\\-]?[0-9]{1,3}\\.[0-9]{1,6}\\#)";
                Pattern pattern = Pattern.compile(geoPattern);
```

```
                mv.toggleSatellite();
            }
        }
        protected class DrawFriendsOverlay extends Overlay{
          public String[] friendName = new String[0];
          public Point[] friendPoint = new Point[0];
          final Paint paint = new Paint();

          @Override
          public void draw(Canvas canvas, PixelCalculator calculator, Boolean
shadow){
                for(int x=0;x<friendPoint.length; x++){
                      int[] coords = new int[2];
                      calculator.getPointXY(friendPoint[x], coords);
            RectF oval = new RectF(coords[0] - 7, coords[1] - 7,
                            coords[0] + 7, coords[1] + 7);
            paint.setTextSize(14);
              canvas.drawText(friendName[x],
                      coords[0]+7, coords[1], paint);
              canvas.drawOval(oval, paint);
                }
          }
          public void addNewFriend(String name,Point point ){
          int x = friendPoint.length;

                String[] friendNameB = new String[x + 1];
                Point[] friendPointB = new Point[x + 1];

                System.arraycopy(friendName, 0, friendNameB, 0, x );
                System.arraycopy(friendPoint, 0, friendPointB, 0, x);

                friendNameB[x] = name;
                friendPointB[x]= point;

                friendName = new String[x + 1];
                friendPoint = new Point[x + 1];
                System.arraycopy(friendNameB, 0, friendName, 0, x + 1 );
                System.arraycopy(friendPointB, 0, friendPoint, 0, x + 1 );


          }

        }
}
```


The last task to finish this project is to create the main Activity, FindAFriend, which will be a shell that calls the other Activities you created in this chapter.

```
        menu.add(0, INSERT_ID, R.string.menu_insert).setShortcut('3', 'a');

        Intent intent = new Intent(null, getIntent().getData());
        intent.addCategory(Intent.ALTERNATIVE_CATEGORY);
        menu.addIntentOptions(
            Menu.ALTERNATIVE, 0, new ComponentName(this, FindAFriend.class),
            null, intent, 0, null);

        return true;
    }

    @Override
    public boolean onPrepareOptionsMenu(Menu menu) {
        super.onPrepareOptionsMenu(menu);
        final boolean haveItems = mCursor.count() > 0;

        if (haveItems) {
            Uri uri = ContentUris.withAppendedId(getIntent().getData(),
getSelectedItemId());

            Intent[] specifics = new Intent[1];
            specifics[0] = new Intent(Intent.EDIT_ACTION, uri);
            Menu.Item[] items = new Menu.Item[1];

            Intent intent = new Intent(null, uri);
            intent.addCategory(Intent.SELECTED_ALTERNATIVE_CATEGORY);
            menu.addIntentOptions(Menu.SELECTED_ALTERNATIVE, 0, null,
specifics, intent, 0, items);
            menu.add(Menu.SELECTED_ALTERNATIVE, DELETE_ID,
R.string.menu_delete)
                    .setShortcut('2', 'd');
            menu.add(Menu.SELECTED_ALTERNATIVE, FIND_FRIENDS,
R.string.find_friends).setShortcut('4', 'f');
            if (items[0] != null) {
                items[0].setShortcut('1', 'e');
            }
        } else {
            menu.removeGroup(Menu.SELECTED_ALTERNATIVE);
        }

        menu.setItemShown(DELETE_ID, haveItems);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(Menu.Item item) {
        switch (item.getId()) {
        case DELETE_ID:
            deleteItem();
```



This option launches the custom View you created. Enter a friend's name on the line provided, as shown here, and return to the main Activity by clicking the back arrow on the Emulator.

This page intentionally left blank

E

F

G

H

I

J

K