ANDROID™
A PROGRAMER’ S GUIDE
Jerome(J.F) DiMarzio

About the Author
J.F. DiMarzio is a developer with over 15 years of experience in networking and application development and is the author of seven books on computing technologies. He has become a leading resource in the fields of IT consulting and development. He lives in Central Florida.
About the Technical Editor
Gilbert L. Polo is a software developer with over 20 years of experience working in the telecommunications, financial, and, most recently, educational industries. He has programmed in various languages including C, C++, Java, and C#.

Why Eclipse?

Why is Eclipse the recommended IDE for Android applications? There are a few reasons for this particular endorsement:

- In keeping with the Open Handset Alliance’ s theme of truly opening the mobile development market, Eclipse is one of the most fully featured, free, Java IDEs available. Eclipse is also very easy to use, with a minimal learning curve. This makes Eclipse a very attractive IDE for solid, open Java development.
- The Open Handset Alliance has released an Android plugin for Eclipse that allows you to create Android-specific projects, compile them, and use the Android Emulator to run and debug them. These tools and abilities will prove invaluable when you are creating your first Android apps. You can still create Android apps in other IDEs, but the Android plugin for Eclipse creates certain setup elements—such as files and compiler settings—for you. The help provided by the Android plugin for Eclipse saves you precious development time and greatly reduces the learning curve, which means you can spend more time creating incredible applications.

NOTE

Eclipse is also available for Mac and Linux. Having greater availability, on numerous operating systems, means that almost anyone can develop Android applications on any computer. However, the examples and screenshots in this book are given from the Microsoft Windows version of Eclipse. Keep this in mind if you are using Eclipse in a non-Microsoft environment; your interface may look slightly different from the screenshots, but the overall functionality should not change. If there is a major change in operation of Eclipse under Linux, I will include an example of that change. I will provide several examples from within a Linux environment. The majority of these examples will be from the Linux/Android command-line environment.

Downloading and Installing the Android Plugin for Eclipse

The first step in setting up the Android SDK within the Eclipse development

environment is to download and install the Android plugin for Eclipse. Both tasks
of downloading and installing the plugin can be performed at the same time, and
are relatively easy to do:

1. Open the Eclipse application. You will download the Android plugin for Eclipse from within the Eclipse IDE.

2. Choose Help | Software Updates | Find and Install.

3. In the Install/Update window, which allows you to begin the process of downloading and installing any of the plugins that are available to you for Eclipse, click the Search for New Features to Install radio button and then click Next.

4. The Update Sites to Visit page of the Install window, shown next, lists all the default websites used for obtaining Eclipse plugins. However, the plugin you want, Android for Eclipse, is not available from the default sites. To download the Android plugin, you must tell Eclipse where to look for it, so click the New Remote Site button.

5. In the New Update Site dialog box, shown next, you must enter two pieces of information to continue: a name for your new site, and its associated URL. The name is only for display purposes and does not affect the downloading of the plugin. In the Name field, enter **Android Plugin**. In the URL field, enter the URL from which Eclipse will obtain information about the plugins that are available: **https://dl-ssl.google.com/android/eclipse/**. Click OK.

NOTE

The name for your site can be anything you want, as long as it will help you identify
what the link is. Feel free to use something other than Android Plugin.

6. A new site named Android Plugin should now be in your list of available sites:
At this point Eclipse has not yet looked for the plugin; this is just a list of paths that you can tell Eclipse to check when looking for new plugins to install.

7. Check the check box next to Android Plugin and then click Finish. Eclipse searches the URL associated with the Android Plugin site for any available plugins.

8. On the Search Results page of the Updates window, select the Android Plugin and then click Finish.

9. On the Feature License page of the Install window, shown next, accept the licensing agreement for the Android Development Tools and click Next.

Examining the Android-Created Files

This section discusses the new files that Android has just created for you. A fairly robust structure has been created for you, and if you do not know what you are looking at, you may end up putting some code in places that you should not. There are files provided by Android that you need to modify, and there are ones that you should not modify; knowing the difference may save you from having to re-create your project.
With your new application project open, take a look at the Package Explorer, one
of two tabs located in the pane to the left of the main development area.

NOTE

If the Package Explorer is not open, you can activate it by choosing Window | Show

View | Package Explorer.
You should see a root directory, in this case named HelloWorldText. The root directory is the home, or repository, for all of your project files. Both your user-created files and the Android auto-generated files will be placed in the directory, easily accessible from the Package Explorer. Currently there should be a few items in your root directory: an AndroidManifest.xml file, a package included in the Referenced Libraries, and three directories (res, assets, and src). These items are discussed in turn next.

AndroidManifest.xml

The AndroidManifest.xml file is where your global settings are made. If you are an ASP.NET developer, you can think of AndroidManifest.xml as Web.config and Global.asax rolled into one. (If you are not an ASP.NET developer, this means that AndroidManifest.xml is a place for storing settings.) AndroidManifest.xml will include such settings as application permissions, Activities, and intent filters.
The standard AndroidManifest.xml file should contain the following information:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android=http://schemas.android.com/apk/res/android
package="testPackage.HelloWorldText">
<application android:icon="@drawable/icon">
<activity class=".HelloWorldText" android:label="@string/app_name">
<intent-filter>
<action android:value="android.intent.action.MAIN" />
<category android:value="android.intent.category.LAUNCHER"
/>
</intent-filter>
</activity>
</application>
</manifest>
```

As you create future applications, you will be adding information to this file. Notice that the package name you supplied is listed here, as well as the action that your Activity will handle.

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.
If you have used Microsoft SQL Server or SQLite, the structure and process for using Android’ s SQLite database will not seem foreign. Whatever your experience, this section covers all the skills you need to create and use a fully functional SQLite database.
You are going to create a database on your Android Emulator. To do this, you need to access the Android SDK command-line tools and use the **shell** command to access the Android server.

TIP

Refer to Chapter 3 to refresh your memory on setting your Path statement and using the command-line tools.

Once you are shelled into the server, you need to navigate to the location where the database will reside. All SQLite databases for Android reside in the data/data/<package>/ databases directory. Use the **cd** command to change directories from your current location to the data directory, and then again to the <package> directory. Use **ls** to list the files and directories at your current location if you are unsure of the <package> directory name. Change the directory to the location where <package> is, android_programmers_ guide.FindAFriend, as shown in the following illustration.

CAUTION

If you do not have an android_programmers_guide.FindAFriend directory, create your application as explained in the previous section and quickly run the "Hello World!" default application that is created with your project. This will ensure that you have the correct directory.

Once you have navigated to the android_programmers_guide.FindAFriend directory, run the **ls** command. This command lists all the files and directories within a specific directory. This command should come up empty. As of right now, there are no files or directories inside your android_programmers_guide.FindAFriend directory.
Given that SQLite databases must be in a *databases* directory within this directory, this is a good time to create one. The tool **mkdir** creates directories for you. Therefore, run the command **mkdir databases**. This creates the directory that will hold your database.

CAUTION

Right now, you are most likely shelled into the server as root. Therefore, the directory you just created will be accessible only to root. This will prove very problematic when you attempt to run your Activity, because each Activity has a different user. To get around this, for development purposes, run **chmod 777 databases** to grant everyone access to the databases directory. However, in the future, you must be cautious about granting everyone rights to sensitive items on Android. Give to specific users only those rights that they need for specific items.

Now that you have created the directory for the database, you can create the database. Use the **cd** command to navigate into your databases directory. After you are in the databases directory, use the sqlite3 tool to create your database and name it friends.db, as follows:

```
# sqlite friends.db
```

If the command is successful, you should see a SQLite3 version message, in this case 3.5.0, and a SQLite3 prompt—sqlite>. This indicates that the database itself has been successfully created but is still empty. The database contains no tables or data. With this in mind, your next step is to create a table for your Activity’ s data.
You need to create a table called *friends*. This table will hold id, name, location,

created, and modified fields. These fields will offer more than enough information for your project.

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.