12

Publishing Android Applications

WHAT YOU WILL LEARN IN THIS CHAPTER

- How to prepare your application for deployment
- Exporting your application as an APK file and signing it with a new certificate
- How to distribute your Android application
- Publishing your application on the Android Market

So far you have seen quite a lot of interesting things you can do with your Android device. However, in order to get your application running on users' devices, you need a way to deploy it and distribute it. In this chapter, you will learn how to prepare your Android applications for deployment and get them onto your customer's devices. In addition, you will learn how to publish your applications on the Android Market, where you can sell them and make some money!

PREPARING FOR PUBLISHING

Google has made it relatively easy to publish your Android application so that it can be quickly distributed to end users. The steps to publishing your Android application generally involve the following:

1. Export your application as an APK (Android Package) file.
2. Generate your own self-signed certificate and digitally sign your application with it.
3. Deploy the signed application.
4. Use the Android Market for hosting and selling your application.

In the following sections, you will learn how to prepare your application for signing, and then learn about the various ways to deploy your applications.

This chapter uses the `LBS` project created in Chapter 9 to demonstrate how to deploy an Android application.

Versioning Your Application

Beginning with version 1.0 of the Android SDK, the `AndroidManifest.xml` file of every Android application includes the `android:versionCode` and `android:versionName` attributes:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.learn2develop.LBS"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="14" />
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <uses-library android:name="com.google.android.maps" />
        <activity
            android:label="@string/app_name"
            android:name=".LBSActivity" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

The `android:versionCode` attribute represents the version number of your application. For every revision you make to the application, you should increment this value by 1 so that you can programmatically differentiate the newest version from the previous one. This value is never used by the Android system, but it is useful for developers as a means to obtain an application's version number. However, the `android:versionCode` attribute is used by Android Market to determine whether a newer version of your application is available.

You can programmatically retrieve the value of the `android:versionCode` attribute by using the `getPackageInfo()` method from the `PackageManager` class, like this:

```
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
```

```
import android.content.pm.PackageManager.NameNotFoundException;

    private void checkVersion() {
        PackageManager pm = getPackageManager();
        try {
            //---get the package info---
            PackageInfo pi =
                pm.getPackageInfo("net.learn2develop.LBS", 0);
            //---display the versioncode---
            Toast.makeText(getBaseContext(),
                "VersionCode: " +Integer.toString(pi.versionCode),
                Toast.LENGTH_SHORT).show();
        } catch (NameNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
```

The `android:versionName` attribute contains versioning information that is visible to users. It should contain values in the format *<major>.<minor>.<point>*. If your application undergoes a major upgrade, you should increase the *<major>* by 1. For small incremental updates, you can increase either the *<minor>* or *<point>* by 1. For example, a new application may have a version name of "1.0.0." For a small incremental update, you might change it to "1.1.0" or "1.0.1." For the next major update, you might change it to "2.0.0."

If you are planning to publish your application on the Android Market (`www.android.com/market/`), the `AndroidManifest.xml` file must have the following attributes:

- `android:versionCode` (within the `<manifest>` element)
- `android:versionName` (within the `<manifest>` element)
- `android:icon` (within the `<application>` element)
- `android:label` (within the `<application>` element)

The `android:label` attribute specifies the name of your application. This name is displayed in the Settings ➪ Apps section of your Android device. For the `LBS` project, give the application the name "Where Am I":

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.learn2develop.LBS"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="14" />
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>

    <application
        android:icon="@drawable/ic_launcher"
        android:label="Where Am I" >
```

```
        <uses-library android:name="com.google.android.maps" />
        <activity
            android:label="@string/app_name"
            android:name=".LBSActivity" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

In addition, if your application needs a minimum version of the Android OS to run, you can specify it in the `AndroidManifest.xml` file using the `<uses-sdk>` element:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.learn2develop.LBS"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="13" />
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>

    <application
        android:icon="@drawable/ic_launcher"
        android:label="Where Am I" >
        <uses-library android:name="com.google.android.maps" />
        <activity
            android:label="@string/app_name"
            android:name=".LBSActivity" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

In the preceding example, the application requires a minimum of SDK version 13, which is Android 3.2.1. In general, you should set this version number to the lowest one that your application can support. This ensures that a wider range of users will be able to run your application.

Digitally Signing Your Android Applications

All Android applications must be digitally signed before they are allowed to be deployed onto a device (or emulator). Unlike some mobile platforms, you need not purchase digital certificates from

a certificate authority (CA) to sign your applications. Instead, you can generate your own self-signed certificate and use it to sign your Android applications.

When you use Eclipse to develop your Android application and then press F11 to deploy it to an emulator, Eclipse automatically signs it for you. You can verify this by going to Windows ➪ Preferences in Eclipse, expanding the Android item, and selecting Build (see Figure 12-1). Eclipse uses a default debug keystore (appropriately named "`debug.keystore`") to sign your application. A keystore is commonly known as a *digital certificate*.

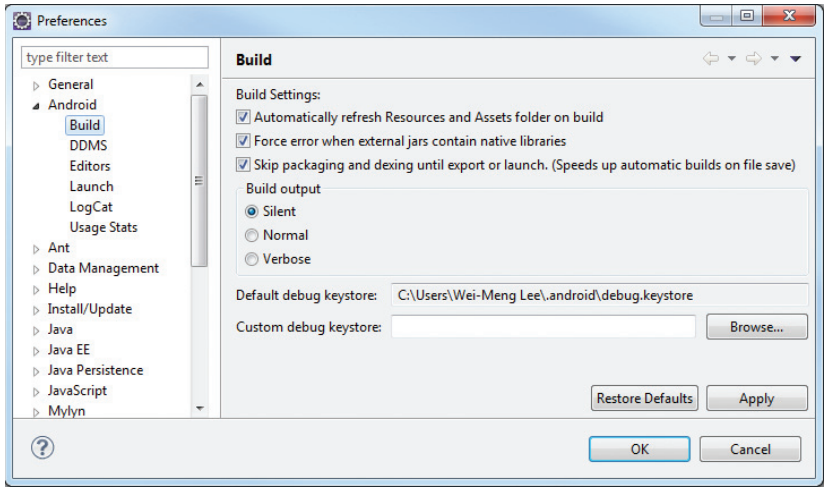

FIGURE 12-1

If you are publishing an Android application, you must sign it with your own certificate. Applications signed with the debug certificate cannot be published. Although you can manually generate your own certificates using the `keytool.exe` utility provided by the Java SDK, Eclipse makes it easy for you by including a wizard that walks you through the steps to generate a certificate. It will also sign your application with the generated certificate (which you can sign manually using the `jarsigner.exe` tool from the Java SDK).

The following Try It Out demonstrates how to use Eclipse to export an Android application and sign it with a newly generated certificate.

TRY IT OUT Exporting and Signing an Android Application

For this Try It Out, you will use the `LBS` project created in Chapter 9.

1. Select the `LBS` project in Eclipse and then select File ➪ Export. . . .
2. In the Export dialog, expand the Android item and select Export Android Application (see Figure 12-2). Click Next.
3. The `LBS` project should now be displayed (see Figure 12-3). Click Next.

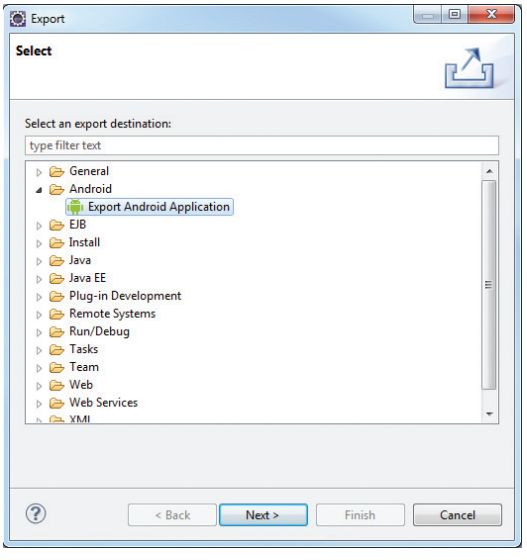

FIGURE 12-2

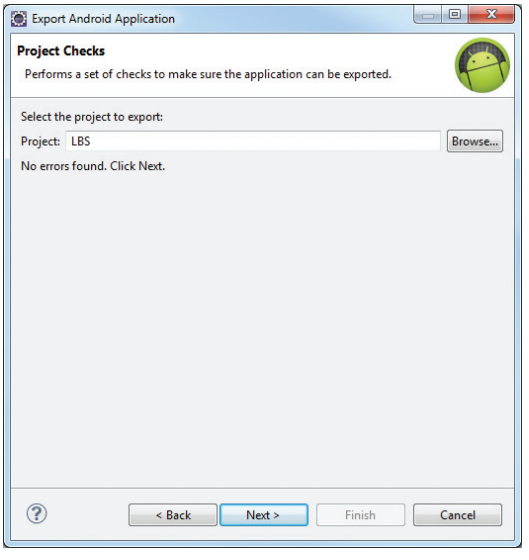

FIGURE 12-3

4. Select the "Create new keystore" option to create a new certificate (keystore) for signing your application (see Figure 12-4). Enter a path to save your new keystore and then enter a password to protect the keystore. For this example, enter **keystorepassword** as the password. Click Next.

5. Provide an alias for the private key (name it **DistributionKeyStoreAlias**; see Figure 12-5) and enter a password to protect the private key. For this example, enter **keypassword** as the password. You also need to enter a validity period for the key. According to Google, your application must be signed with a cryptographic private key whose validity period ends after 22 October 2033. Hence, enter a number that is greater than 2033 minus the current year. Finally, enter your name in the field labeled First and Last Name. Click Next.

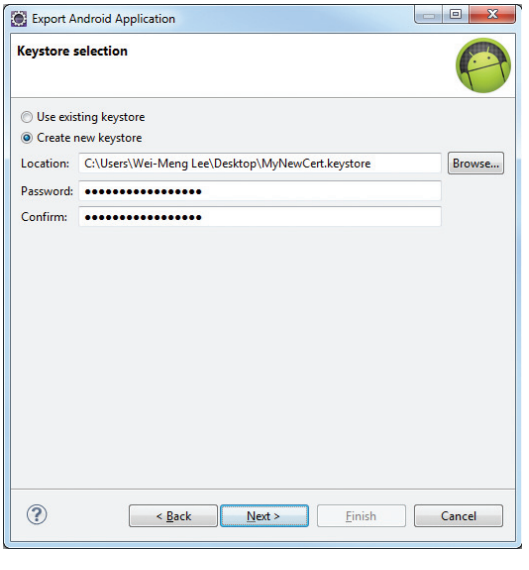

FIGURE 12-4

6. Enter a path to store the destination APK file (see Figure 12-6). Click Finish. The APK file will now be generated.

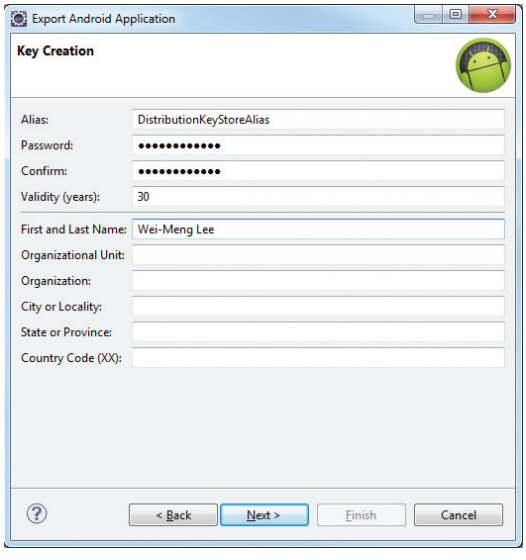

FIGURE 12-5

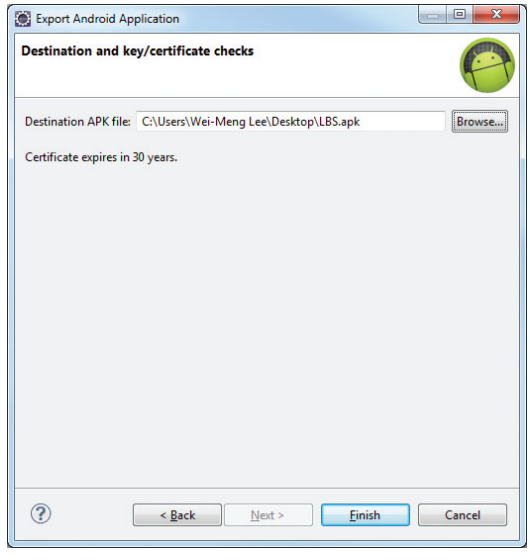

FIGURE 12-6

7. Recall from Chapter 9 that the `LBS` application requires the use of the Google Maps API key, which you applied by using your `debug.keystore`'s MD5 fingerprint. This means that the Google Maps API key is essentially tied to the `debug.keystore` used to sign your application. Because you are now generating your new keystore to sign your application for deployment, you need to apply for the Google Maps API key again, using the new keystore's MD5 fingerprint. To do so, go to the command prompt and enter the following command (the location of your `keytool.exe` utility might differ slightly; see Figure 12-7):

```
C:\Program Files\Java\jre6\bin>keytool.exe -list -v -alias DistributionKeyStoreAlias
-keystore "C:\Users\Wei-Meng Lee\Desktop\MyNewCert.keystore"
-storepass keystorepassword -keypass keypassword -v
```

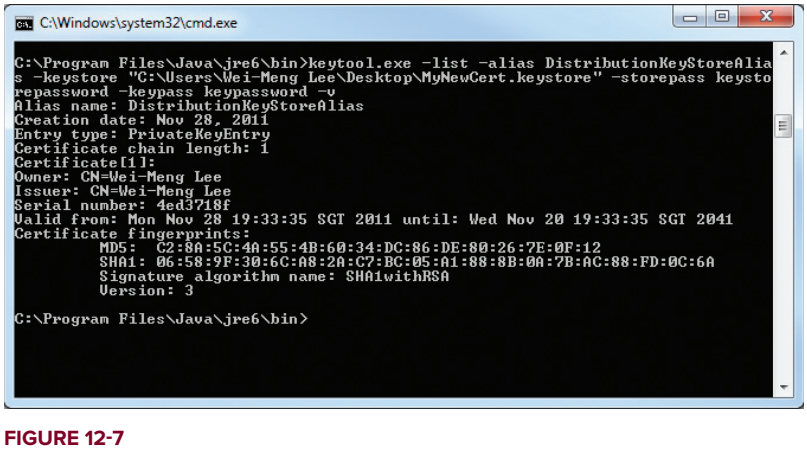

FIGURE 12-7

8. Using the MD5 fingerprint obtained from the previous step, go to `http://code.google.com/android/add-ons/google-apis/maps-api-signup.html` and sign up for a new Maps API key.

9. Enter the new Maps API key in the `main.xml` file:

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

<com.google.android.maps.MapView
    android:id="@+id/mapView"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:enabled="true"
    android:clickable="true"
    android:apiKey="your_key_here" />

</LinearLayout>
```

10. With the new Maps API key entered in the `main.xml` file, you now need to export the application once more and resign it. Repeat steps 2 through 4. When you are asked to select a keystore, select the "Use existing keystore" option (see Figure 12-8) and enter the password you used earlier to protect your keystore (in this case, **keystorepassword**). Click Next.

11. Select the "Use existing key" option (see Figure 12-9) and enter the password you set earlier to secure the private key (enter **keypassword**). Click Next.

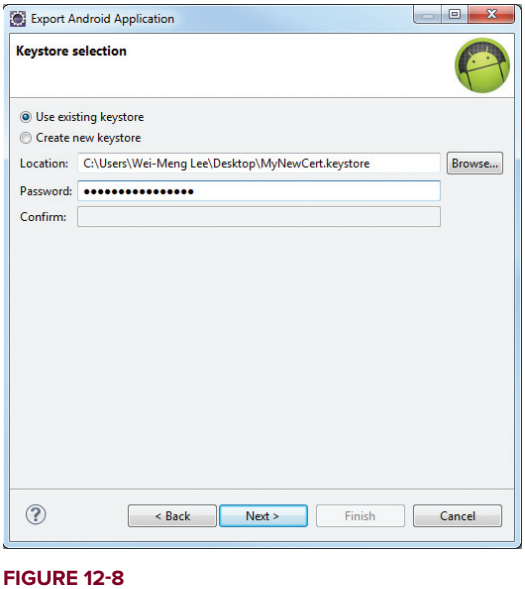

FIGURE 12-8

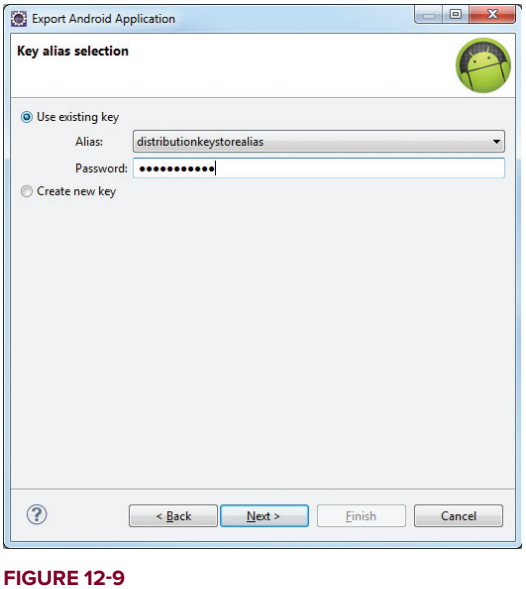

FIGURE 12-9

12. Click Finish (see Figure 12-10) to generate the APK file again.

That's it! The APK is now generated and contains the new Map API key that is tied to the new keystore.

How It Works

Eclipse provides the Export Android Application option, which helps you to both export your Android application as an APK file and generate a new keystore to sign the APK file. For applications that use the Maps API key, note that the Maps API key must be associated with the new keystore that you use to sign your APK file.

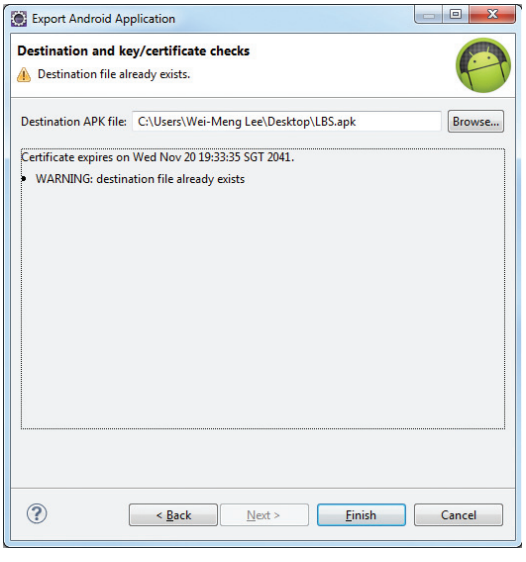

FIGURE 12-10

DEPLOYING APK FILES

After you have signed your APK files, you need a way to get them onto your users' devices. The following sections describe the various ways to deploy your APK files. Three methods are covered:

- Deploying manually using the `adb.exe` tool
- Hosting the application on a web server
- Publishing through the Android Market

Besides these methods, you can install your applications on users' devices using e-mail, an SD card, and so on. As long as you can transfer the APK file onto the user's device, the application can be installed.

Using the adb.exe Tool

Once your Android application is signed, you can deploy it to emulators and devices using the `adb.exe` (Android Debug Bridge) tool (located in the `platform-tools` folder of the Android SDK).

Using the command prompt in Windows, navigate to the `<Android_SDK>\platform-tools` folder. To install the application to an emulator/device (assuming the emulator is currently up and running or a device is currently connected), issue the following command:

```
adb install "C:\Users\Wei-Meng Lee\Desktop\LBS.apk"
```

EXPLORING THE ADB.EXE TOOL

The `adb.exe` tool is a very versatile tool that enables you to control Android devices (and emulators) connected to your computer.

By default, when you use the `adb` command, it assumes that currently there is only one connected device/emulator. If more than one device is connected, the `adb` command returns an error message:

```
error: more than one device and emulator
```

You can view the devices currently connected to your computer by using the `devices` option with `adb`, like this:

```
D:\Android 4.0\android-sdk-windows\platform-tools>adb devices
List of devices attached
HT07YPY09335    device
emulator-5554   device
emulator-5556   device
```

As the preceding example shows, this returns the list of devices currently attached. To issue a command for a particular device, you need to indicate the device using the `-s` option, like this:

```
adb -s emulator-5556 install LBS.apk
```

If you try to install an APK file onto a device that already has the APK file, it will display the following error message:

```
Failure [INSTALL_FAILED_ALREADY_EXISTS]
```

If the LBS application is still on your device or emulator from earlier, you can delete it via Settings ➪ Apps ➪ LBS ➪ Uninstall.

Sometimes the ADB will fail (when too many ADVs are opened at the same time; you will notice that you can no longer deploy applications from Eclipse onto your real devices or emulators). In this case, you need to kill the server and then restart it:

```
adb kill-server
adb start-server
```

When you inspect the launcher on the Android device/emulator, you will be able to see the `LBS` icon (on the top of Figure 12-11). If you select Settings ➪ Apps on your Android device/emulator, you will see the Where Am I application (on the bottom of Figure 12-11).

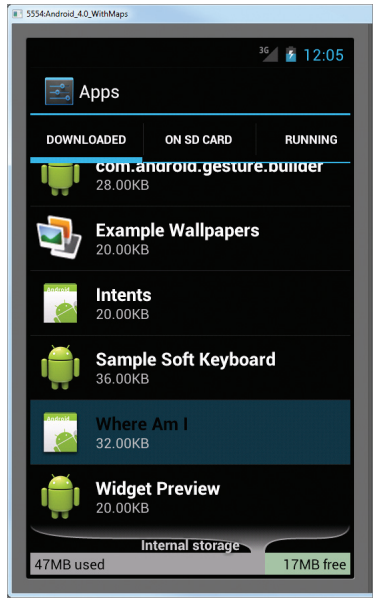

FIGURE 12-11

Besides using the `adb.exe` tool to install applications, you can also use it to remove an installed application. To do so, use the `uninstall` option to remove an application from its installed folder:

```
adb uninstall net.learn2develop.LBS
```

Another way to deploy an application is to use the DDMS tool in Eclipse (see Figure 12-12). With an emulator (or device) selected, use the File Explorer in DDMS to go to the `/data/app` folder and use the "Push a file onto the device" button to copy the APK file onto the device.

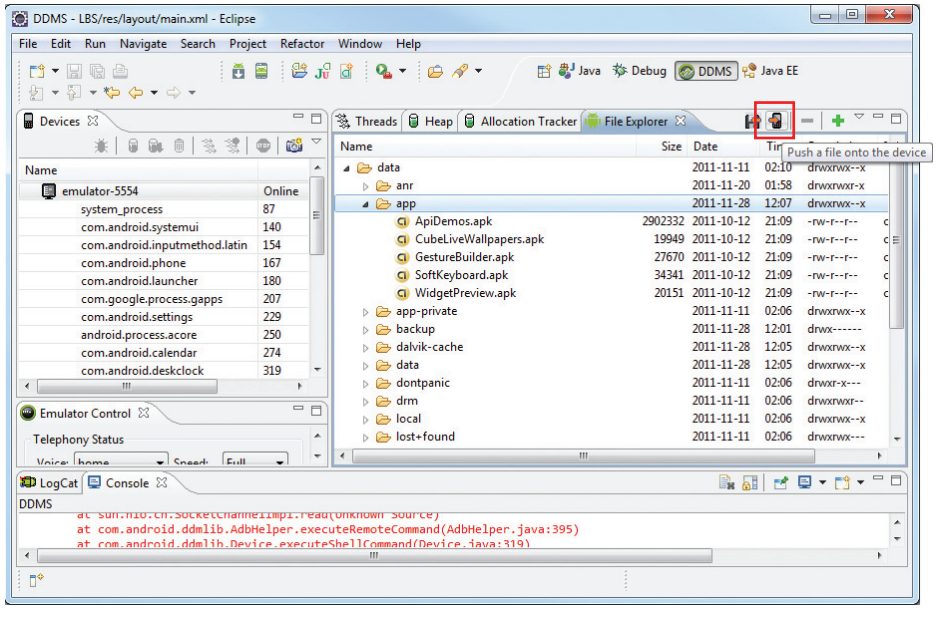

FIGURE 12-12

Using a Web Server

If you wish to host your application on your own, you can use a web server to do that. This is ideal if you have your own web hosting services and want to provide the application free of charge to your users (or you can restrict access to certain groups of people).

NOTE *Even if you restrict your application to a certain group of people, there is nothing to stop users from redistributing your application to other users after they have downloaded your APK file.*

To demonstrate this, I use the Internet Information Server (IIS) on my Windows 7 computer. Copy the signed `LBS.apk` file to `c:\inetpub\wwwroot\`. In addition, create a new HTML file named `index.html` with the following content:

```
<html>
<title>Where Am I application</title>
<body>
Download the Where Am I application <a href="LBS.apk">here</a>
</body>
</html>
```

NOTE *If you are unsure how to set up IIS on your Windows 7 computer, check out the following link:* `http://technet.microsoft.com/en-us/library/cc725762.aspx`.

On your web server, you may need to register a new MIME type for the APK file. The MIME type for the .`apk` extension is `application/vnd.android.package-archive`.

NOTE *If you are unsure how to set up the MIME type on IIS, check out the following link:*

`http://technet.microsoft.com/en-us/library/cc725608(WS.10).aspx`.

NOTE *To install APK files over the Web, you need an SD card installed on your emulator or device. This is because the downloaded APK files are saved to the* `download` *folder created on the SD card. For testing this using the emulator, ensure that your SD card has at least a size of 128MB. There are reports of developers having problems installing their apps with an SD card size smaller than 128MB.*

By default, for online installation of Android applications, the Android emulator or device only allows applications to be installed from the Android Market (`www.android.com/market`). Hence, for installation over a web server, you need to configure your Android emulator/device to accept applications from non-Market sources.

In the Settings application, click the Security item and scroll to the bottom of the screen. Check the "Unknown sources" item (see Figure 12-13). You will be prompted with a warning message. Click OK. Checking this item will allow the emulator/device to install applications from other non-Market sources (such as from a web server).

To install the `LBS.apk` application from the IIS web server running on your computer, launch the Browser application on the Android emulator/device and navigate to the URL pointing to the APK file. To refer to the computer running the emulator, you should use the computer's IP address. Figure 12-14 shows the `index.html` file loaded on the web browser. Clicking the "here" link will download the APK file onto your device. Click the status bar at the top of the screen to reveal the download's status.

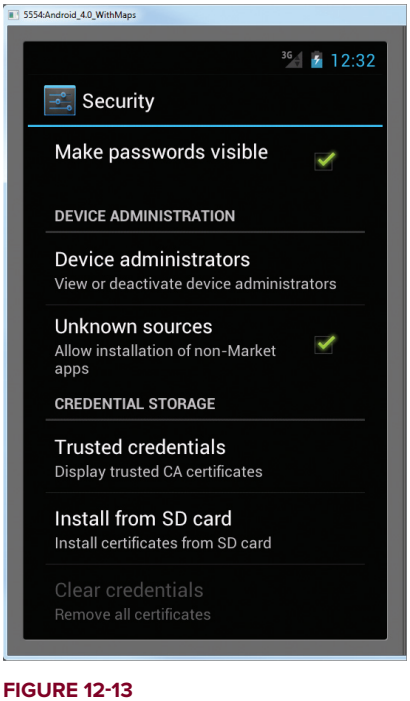

FIGURE 12-13

FIGURE 12-14

To install the downloaded application, simply tap on it. It will show the permission(s) required by the application. Click the Install button to proceed with the installation. When the application is installed, you can launch it by clicking the Open button.

Besides using a web server, you can also e-mail your application to users as an attachment; when the users receive the e-mail, they can download the attachment and install the application directly onto their device.

Publishing on the Android Market

So far, you have learned how to package your Android application and distribute it in various ways — via web server, the `adb.exe` file, e-mail, and SD card.

However, these methods do not provide a way for users to discover your applications easily. A better way is to host your application on the Android Market, a Google-hosted service that makes it very easy for users to discover and download (i.e., purchase) applications for their Android devices. Users simply need to launch the Market application on their Android device in order to discover a wide range of applications that they can install on their devices.

In this section, you will learn how to publish your Android application on the Android Market. You will walk through each of the steps involved, including the various items you need in order to prepare your application for submission to the Android Market.

Creating a Developer Profile

The first step toward publishing on the Android Market is to create a developer profile at `http://market.android.com/publish/Home`. For this, you need a Google account (such as your Gmail account). Once you have logged in to the Android Market, you first create your developer profile (see Figure 12-15). Click Continue after entering the required information.

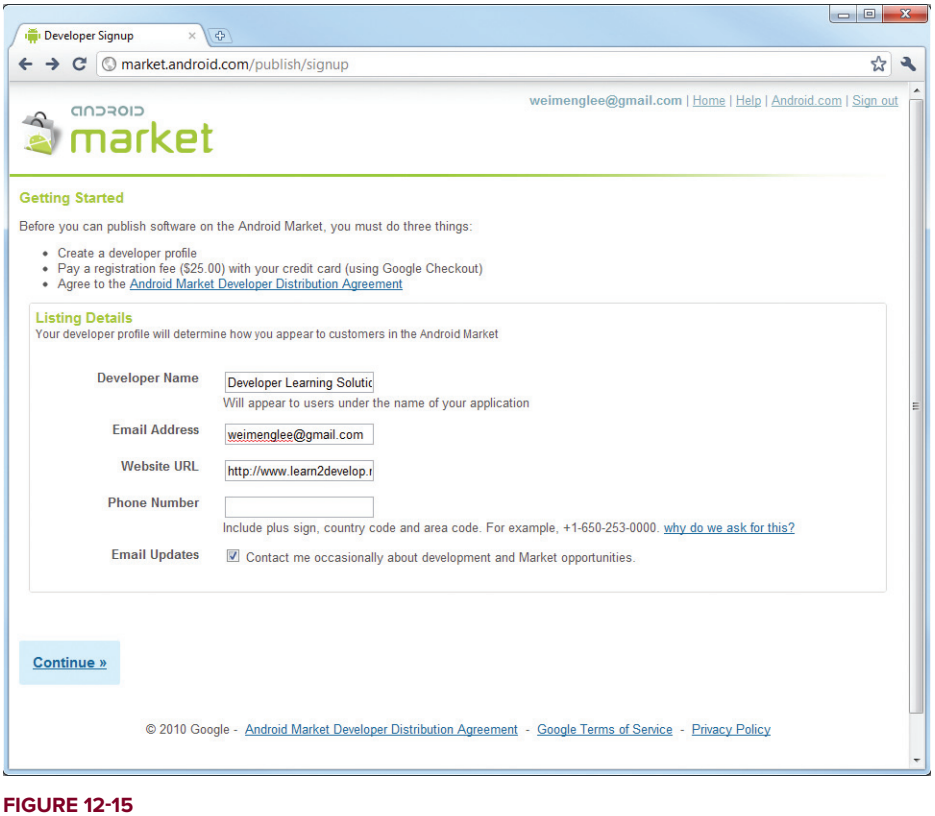

FIGURE 12-15

For publishing on the Android Market, you need to pay a one-time registration fee, currently U.S. $25. Click the Google Checkout button to be redirected to a page where you can pay the registration fee. After paying, click the Continue link.

Next, you need to agree to the Android Market Developer Distribution Agreement. Check the "I agree" checkbox and then click the "I agree. Continue" link.

Submitting Your Apps

After you have set up your profile, you are ready to submit your application to the Android Market. If you intend to charge for your application, click the Setup Merchant Account link located at the bottom of the screen. Here you enter additional information such as bank account and tax ID.

For free applications, click the Upload Application link, shown in Figure 12-16.

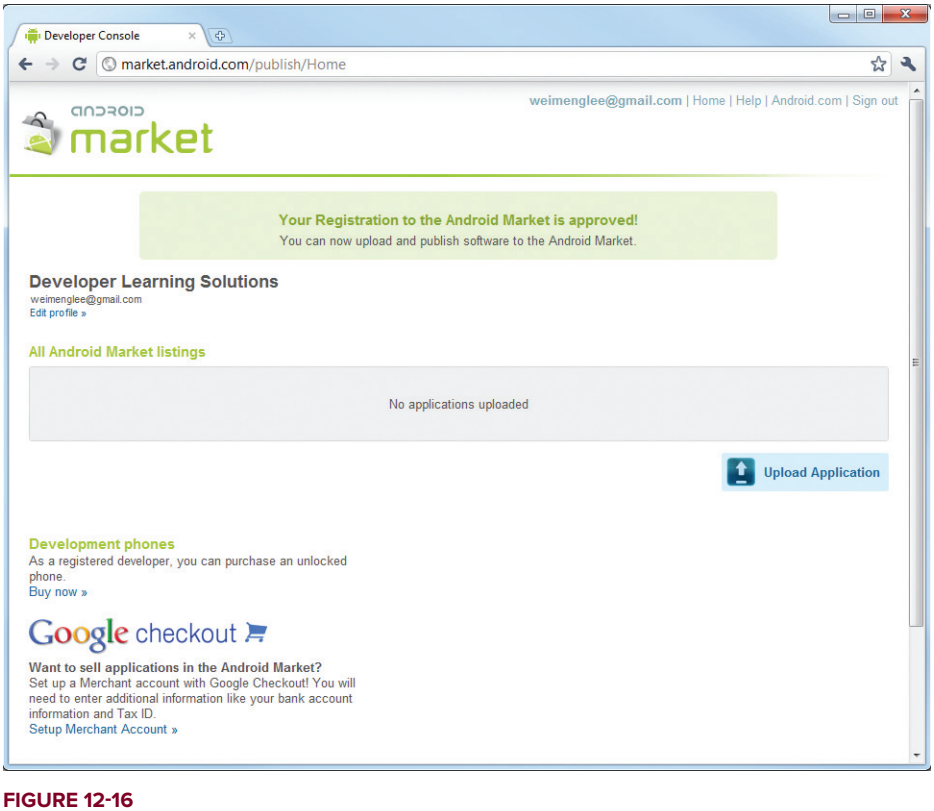

FIGURE 12-16

You will be asked to supply some information about your application. Figure 12-17 shows the first set of details you need to provide. Among the information needed, the following are compulsory:

- The application must be in APK format

- You need to provide at least two screenshots. You can use the DDMS perspective in Eclipse to capture screenshots of your application running on the emulator or real device.
- You need to provide a high-resolution application icon. This size of this image must be 512 × 512 pixels.

The other information details are optional, and you can always supply them later.

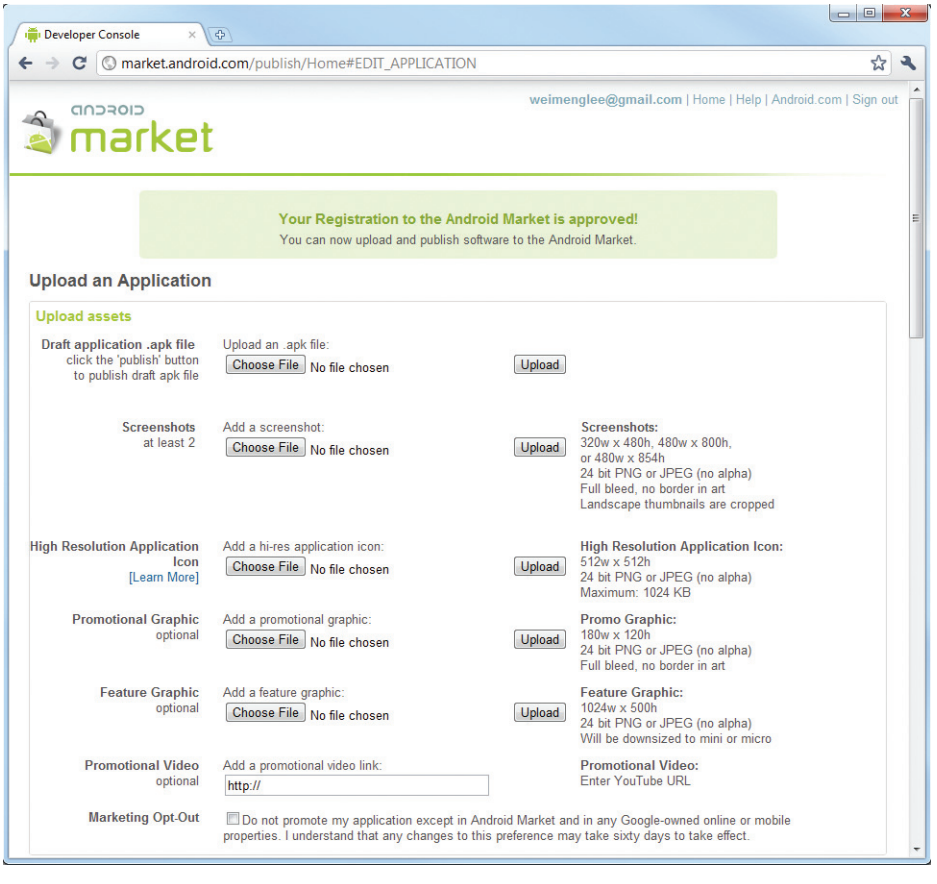

FIGURE 12-17

Figure 12-18 shows the `LBS.apk` file uploaded to the Android Market site. In particular, note that based on the APK file that you have uploaded, users are warned about any specific permissions required, and your application's features are used to filter search results. For example, because my application requires GPS access, it will not appear in the search result list if a user searches for my application on a device that does not have a GPS receiver.

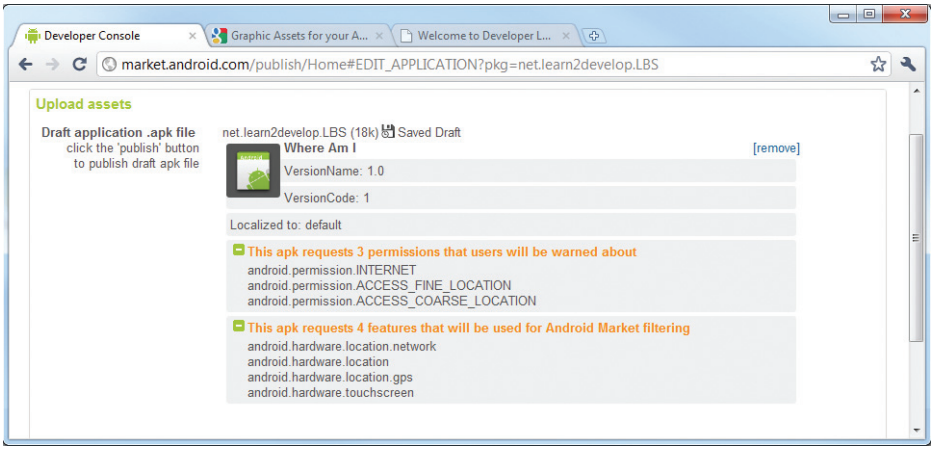

FIGURE 12-18

The next set of information you need to supply, shown in Figure 12-19, includes the title of your application, its description, as well as details about recent changes (useful for application updates). You can also select the application type and the category in which it will appear in the Android Market.

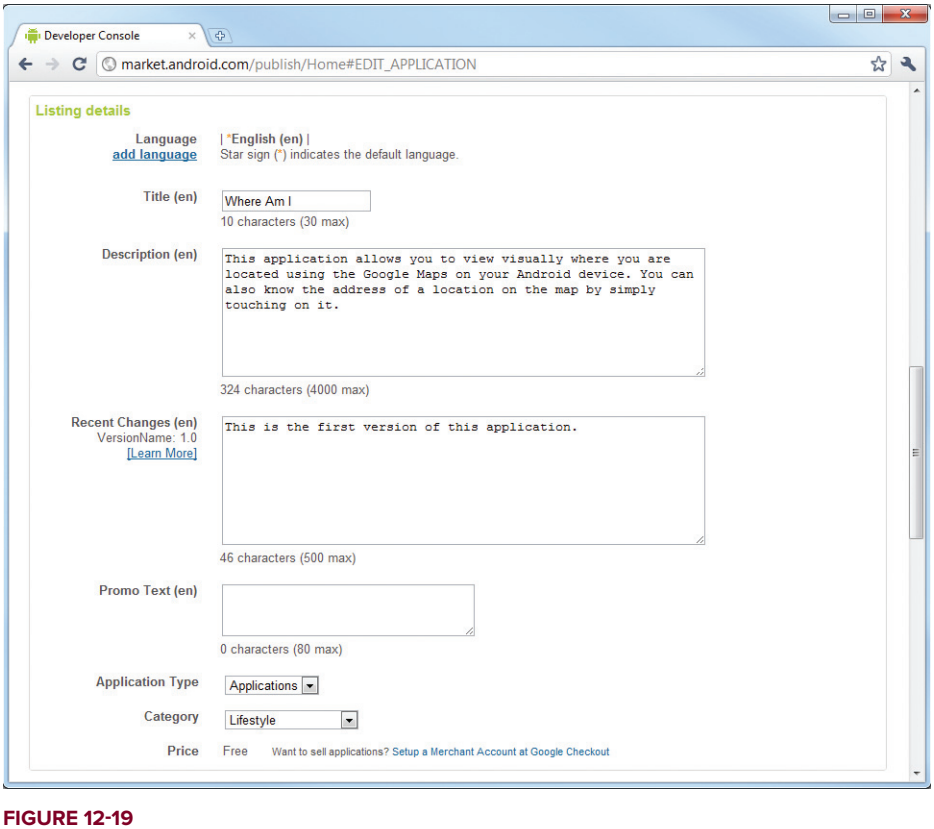

FIGURE 12-19

In the last dialog, you indicate whether your application employs copy protection, and specify a content rating. You also supply your website URL and your contact information (see Figure 12-20). When you have given your consent to the two guidelines and agreements, click Publish to publish your application on the Android Market.

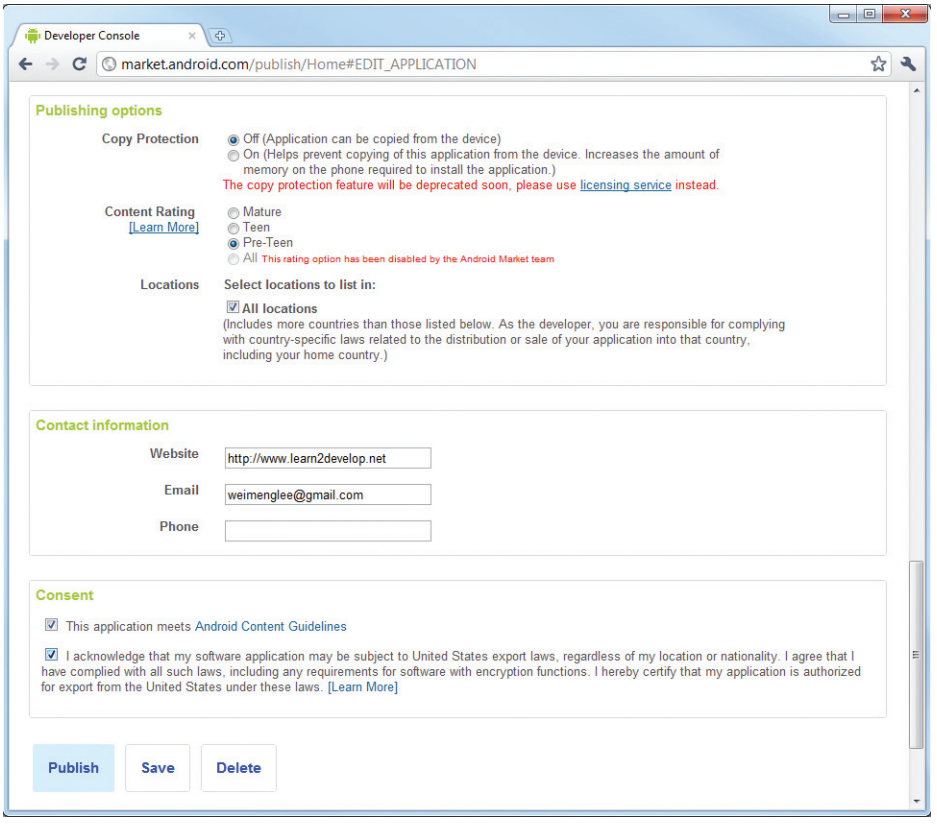

FIGURE 12-20

That's it. Your application is now available on the Android Market. You will be able to monitor any comments submitted about your application (see Figure 12-21), as well as bug reports and total number of downloads.

Good luck! All you need to do now is wait for the good news; and hopefully you can laugh your way to the bank soon!

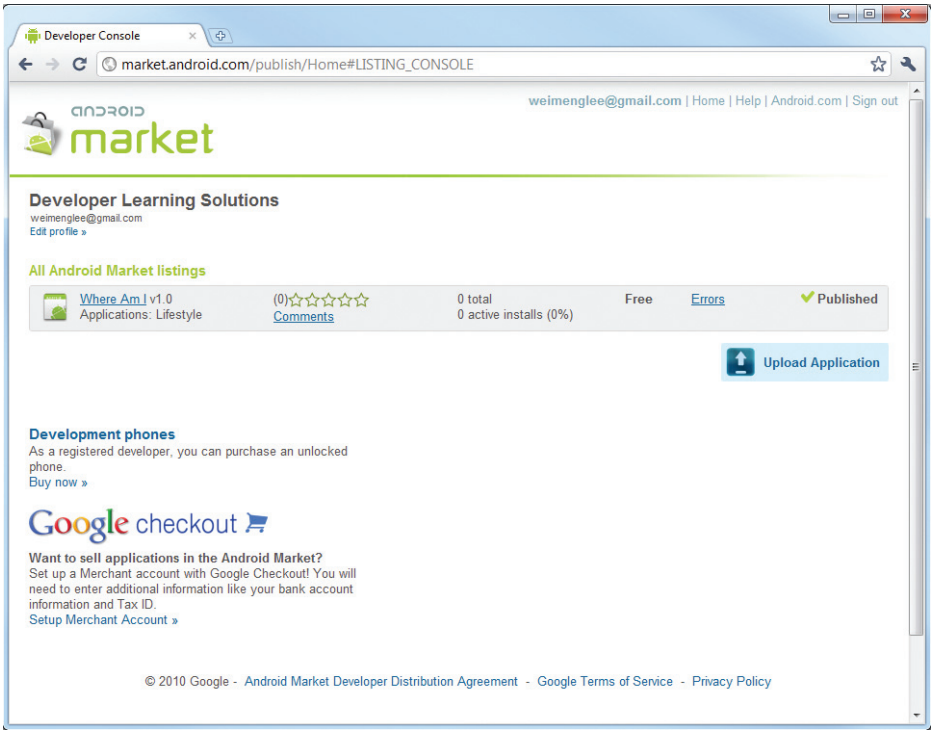

FIGURE 12-21

SUMMARY

In this chapter, you have learned how you can export your Android application as an APK file and then digitally sign it with a keystore you create yourself. You also learned about the various ways you can distribute your application, and the advantages of each method. Finally, you walked through the steps required to publish on the Android Market, which enables you to sell your application and reach out to a wider audience. It is hoped that this exposure enables you to sell a lot of copies and thereby make some decent money.

EXERCISES

1. How do you specify the minimum version of Android required by your application?
2. How do you generate a self-signed certificate for signing your Android application?
3. How do you configure your Android device to accept applications from non-Market sources?

Answers to the exercises can be found in Appendix C.

▶ WHAT YOU LEARNED IN THIS CHAPTER

TOPIC	KEY CONCEPTS

Checklist for publishing your apps	To publish an application on the Android Market, an application must have the following four attributes in the `AndroidManifest.xml` file:
	`android:versionCode`
	`android:versionName`
	`android:icon`
	`android:label`

Signing applications	All applications to be distributed must be signed with a self-signed certificate. The debug keystore is not valid for distribution.
Exporting an application and signing it	Use the Export feature of Eclipse to export the application as an APK file and then sign it with a self-signed certificate.
Deploying APK files	You can deploy using various means, including web server, e-mail, `adb.exe`, and DDMS.
Publishing your application on the Android Market	To sell and host your apps on the Android Market, you can apply with a one-time fee of U.S. $25.

A

Using Eclipse for Android Development

Although Google supports the development of Android applications using IDEs such as IntelliJ, or basic editors like Emacs, Google's recommendation is to use the Eclipse IDE together with the Android Development Tools (ADT) plug-in. Doing so makes developing Android applications much easier and more productive. This appendix describes some of the neat features available in Eclipse that can greatly improve your development work.

WARNING *If you have not downloaded Eclipse yet, please start with Chapter 1, where you will learn how to obtain Eclipse and configure it to work with the Android SDK. This appendix assumes that you have already set up your Eclipse environment for Android development.*

GETTING AROUND IN ECLIPSE

Eclipse is a highly extensible multi-language software development environment that supports application development of all sorts. Using Eclipse, you can write and test your applications using a wide variety of languages, such as Java, C, C++, PHP, Ruby, and more. Because of its extensibility, new users of Eclipse often feel overwhelmed by the IDE. Hence, the following sections aim to make you more at home with Eclipse when you develop your Android applications.

Workspaces

Eclipse adopts the concept of a *workspace*. A workspace is a folder that you have chosen to store all your projects.

When you first start Eclipse, you are prompted to select a workspace (see Figure A-1).

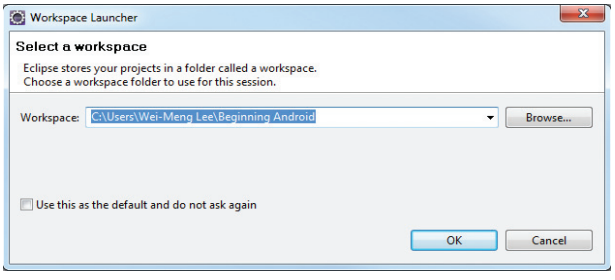

FIGURE A-1

When Eclipse has finished loading the projects located in your workspace, several panes are displayed in the IDE (see Figure A-2).

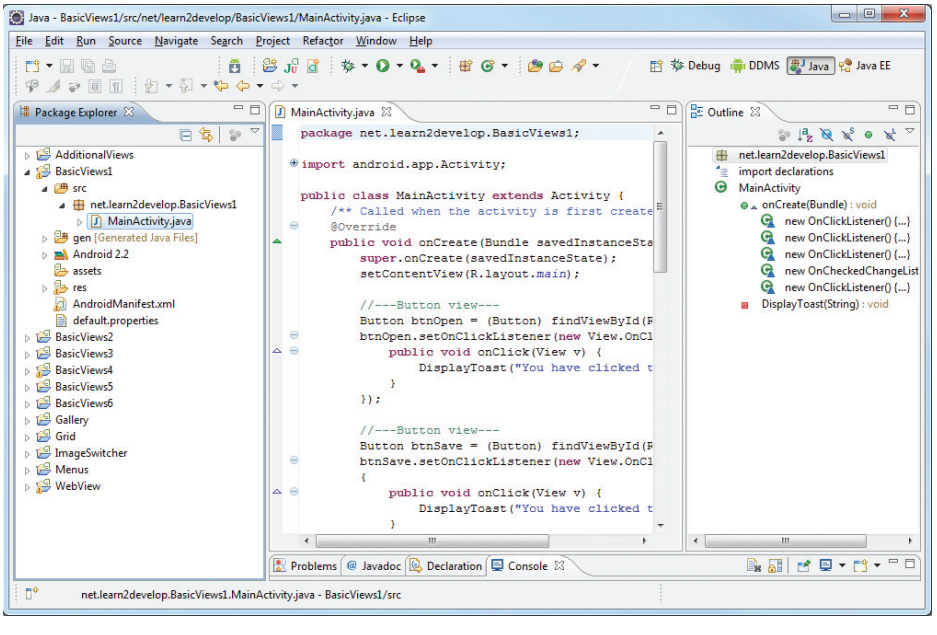

FIGURE A-2

The following sections highlight some of the more important panes that you need to know about when developing Android applications.

Package Explorer

The Package Explorer, shown in Figure A-3, lists all the projects currently in your workspace. To edit a particular item in your project, you can double-click on it and the file will be displayed in the respective editor.

You can also right-click on each item displayed in the Package Explorer to display context-sensitive menu(s) related to the selected item. For example, if you wish to add a new `.java` file to the project, you can right-click on the package name in the Package Explorer and then select New ➪ Class (see Figure A-4).

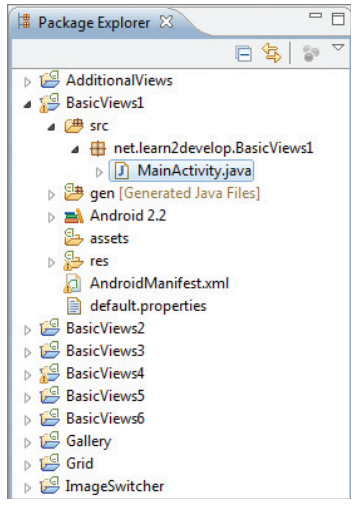

FIGURE A-3

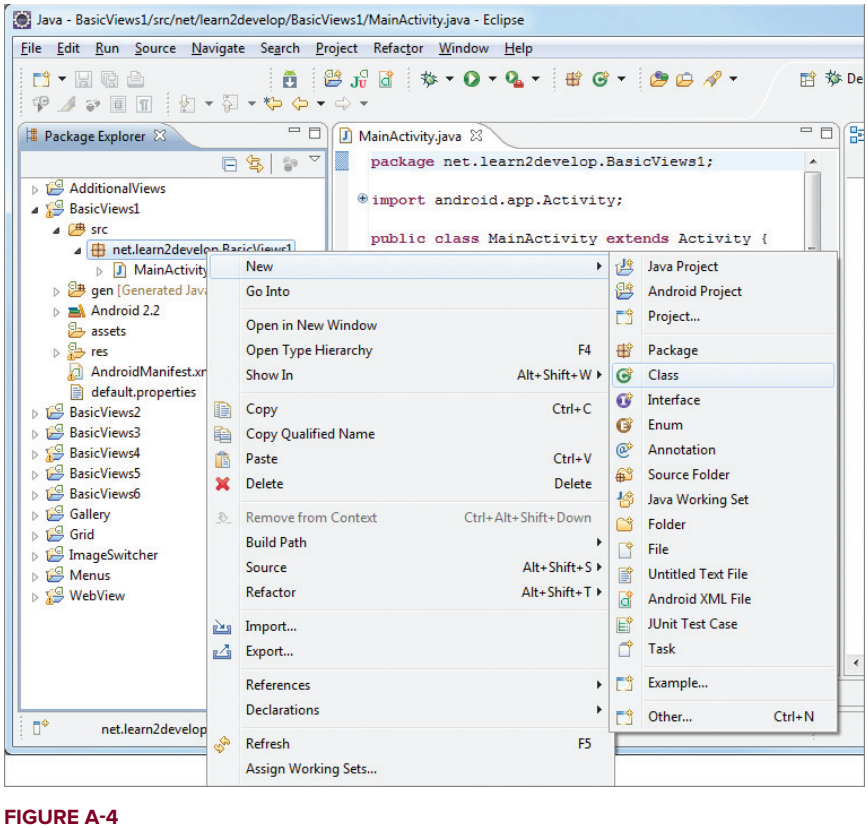

FIGURE A-4

Using Projects from Other Workspaces

There may be times when you have several workspaces created to store different projects. If you need to access the project in another workspace, there are generally two ways to go about doing so. First, you can switch to the desired workspace by selecting File ➪ Switch Workspace (see Figure A-5). Specify the new workspace to work on and then restart Eclipse.

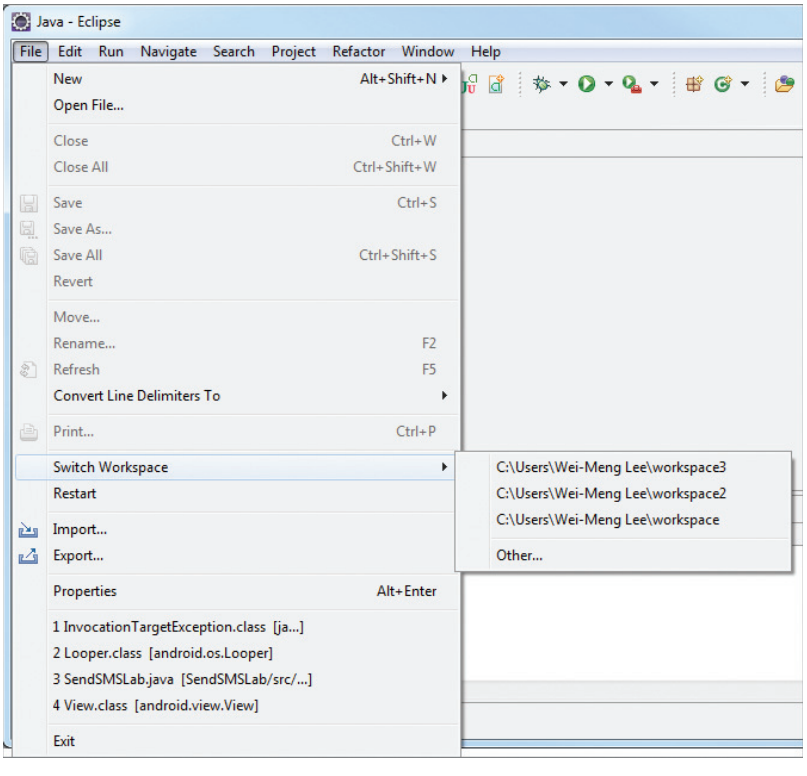

FIGURE A-5

The second method is to import the project from another workspace into the current one. To do so, select File ➪ Import… and then select General ➪ Existing Projects into Workspace (see Figure A-6). Click Next.

In the Select root directory textbox, enter the path of the workspace containing the project(s) you want to import and tick the project(s) you want to import (see Figure A-7). To import the selected project(s), click Finish.

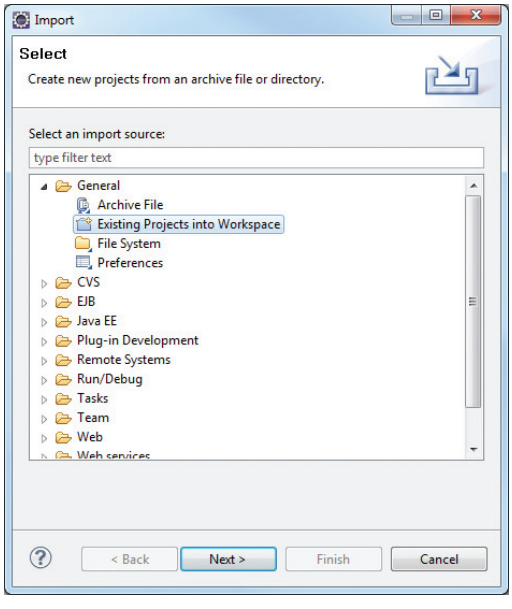

FIGURE A-6

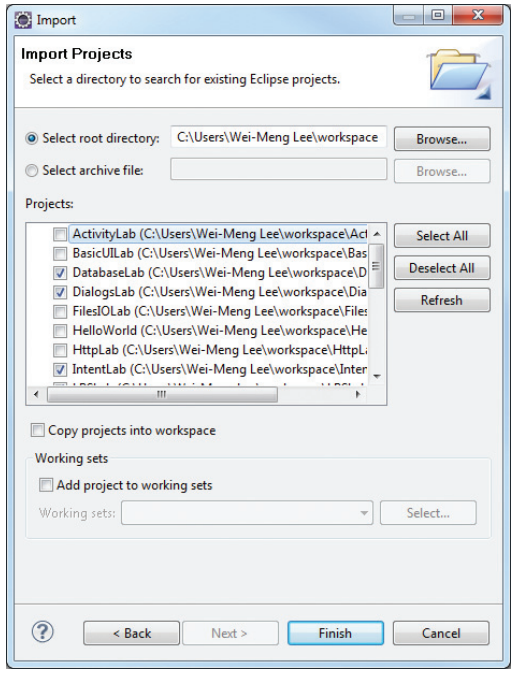

FIGURE A-7

Note that even when you import a project from another workspace into the current workspace, the physical location of the imported project remains unchanged. That is, it will still be located in its original directory. To add a copy of the project to the current workspace, check the "Copy projects into workspace" option.

Using Editors within Eclipse

Depending on the type of items you have double-clicked in the Package Explorer, Eclipse will open the appropriate editor for you to edit the file. For example, if you double-click on a `.java` file, the text editor for editing the source file will be opened (see Figure A-8).

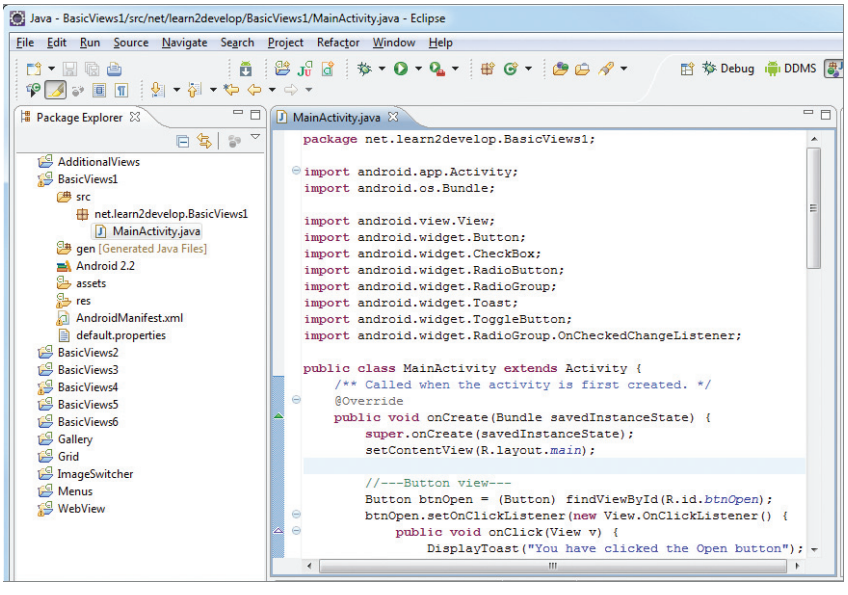

FIGURE A-8

If you double-click on the `ic_launcher.png` file in the `res/drawable-mdpi` folder, the Windows Photo Viewer application will be invoked to display the image (see Figure A-9).

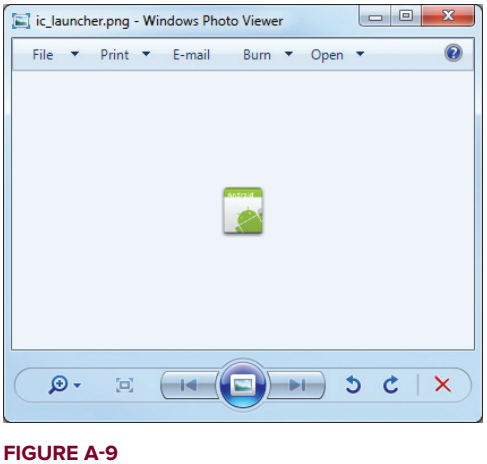

FIGURE A-9

If you double-click on the `main.xml` file in the `res/layout` folder, Eclipse will display the UI editor, where you can graphically view and build the layout of your UI (see Figure A-10).

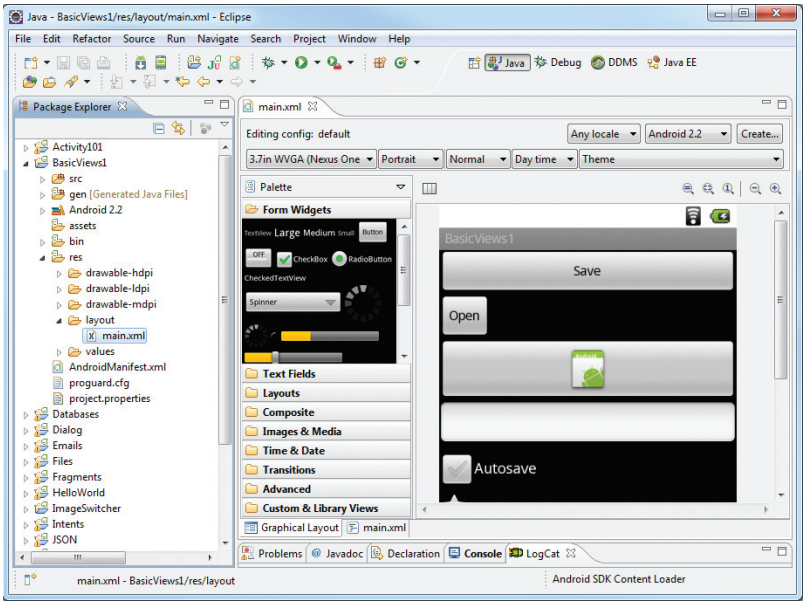

FIGURE A-10

To edit the UI manually using XML, you can switch to XML view by clicking on the main.xml tab located at the bottom of the editor (see Figure A-11).

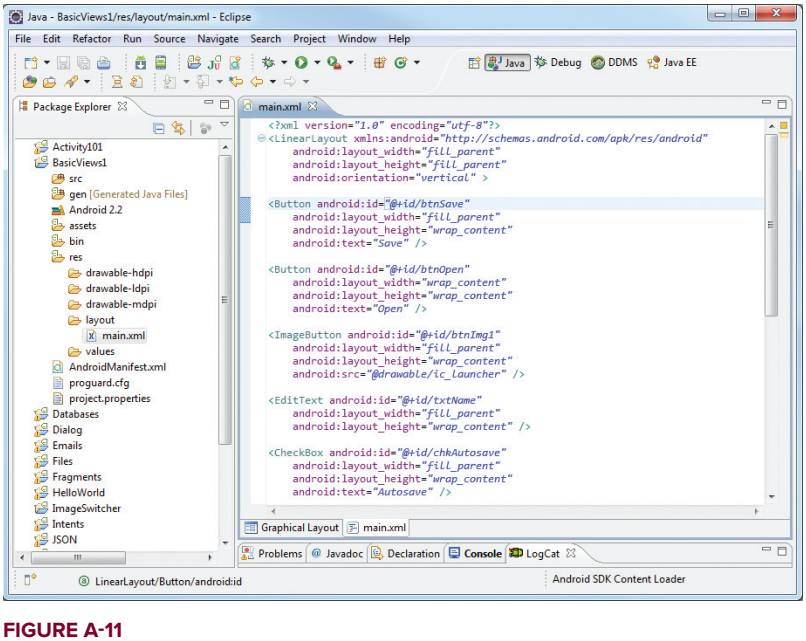

FIGURE A-11

Understanding Eclipse Perspectives

In Eclipse, a *perspective* is a visual container for a set of views and editors. When you edit your Android/Java project in Eclipse, you are in the Java perspective (see Figure A-12).

The Java EE perspective is used for developing enterprise Java applications, and it includes other modules that are relevant to it.

You can switch to other perspectives by clicking the perspective name. If the perspective name is not shown, you can click the Open Perspective button and add a new perspective (see Figure A-13).

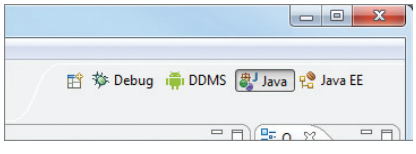

FIGURE A-12

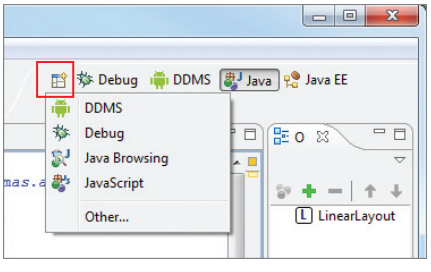

FIGURE A-13

The DDMS perspective contains tools for communicating with Android emulators and devices. This is covered in more detail in Appendix B. The Debug perspective contains panes used for debugging your Android applications. You will learn more about that later in this appendix.

Automatically Importing Packages

The various classes in the Android library are organized into packages. As such, when you use a particular class from a package, you need to import the appropriate packages, like this:

```
import android.app.Activity;
import android.os.Bundle;
```

Because the number of classes in the Android Library is very large, remembering the correct namespace for each class is not an easy task. Fortunately, Eclipse can help you find the correct namespace, which enables you to import it with just a click.

Figure A-14 shows that I have declared an object of type `Button`. Because I did not import the correct package for the `Button` class, Eclipse signals an error beneath the statement. When you move the mouse over the `Button` class, Eclipse displays a list of suggested fixes. In this case, I need to import

the `android.widget.Button` package. Clicking the "Import 'Button' (android.widget)" link will add the import statement to the top of the file.

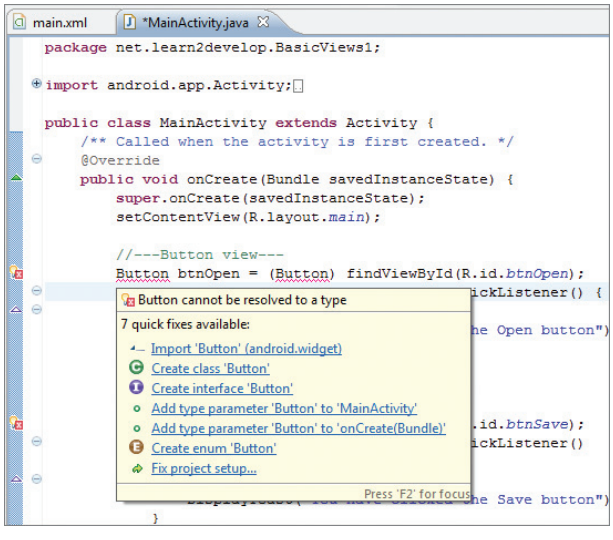

FIGURE A-14

Alternatively, you can use the following key combination: Ctrl+Shift+o. This key combination will cause Eclipse to automatically import all the namespaces required by your class.

Using the Code Completion Feature

Another very useful feature of Eclipse is its support for code completion. Code completion displays a context-sensitive list of relevant classes, objects, methods, and property names as you type in the code editor. For example, Figure A-15 shows code-completion in action. As I type the word "`fin`," I can activate the code-completion feature by pressing Ctrl+space. This brings up a list of names that begin with "`fin`."

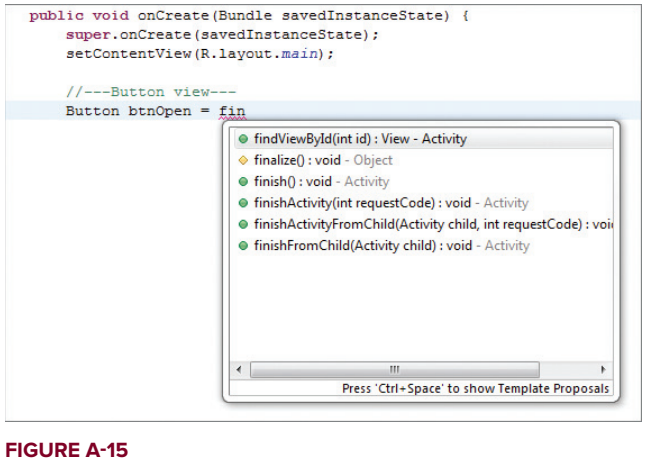

FIGURE A-15

To select the required name, simply double-click on it or use your cursor to highlight it and then press the Enter key.

Code completion also works when you type a period (.) after an object/class name. Figure A-16 shows an example.

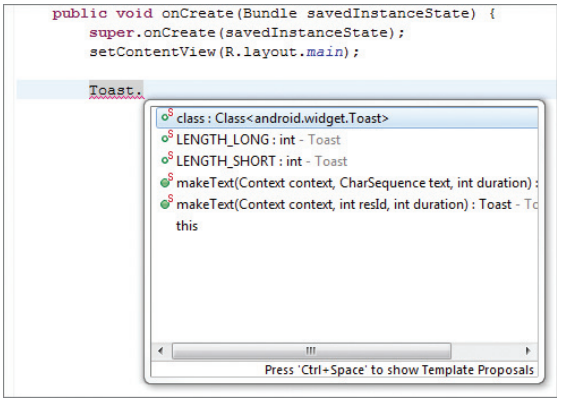

FIGURE A-16

Refactoring

Refactoring is a very useful feature that most modern IDEs support. Eclipse supports a whole slew of refactoring features that make application development efficient.

In Eclipse, when you position the cursor at a particular object/variable, the editor will highlight all occurrences of the selected object in the current source (see Figure A-17).

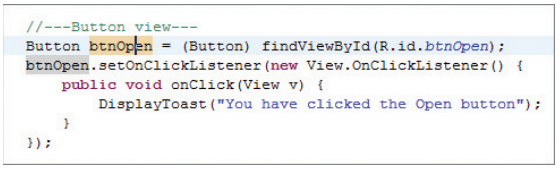

FIGURE A-17

This feature is very helpful for identifying where a particular object is used in your code. To change the name of an object, simply right-click on it and select Refactor ➪ Rename. . . (see Figure A-18).

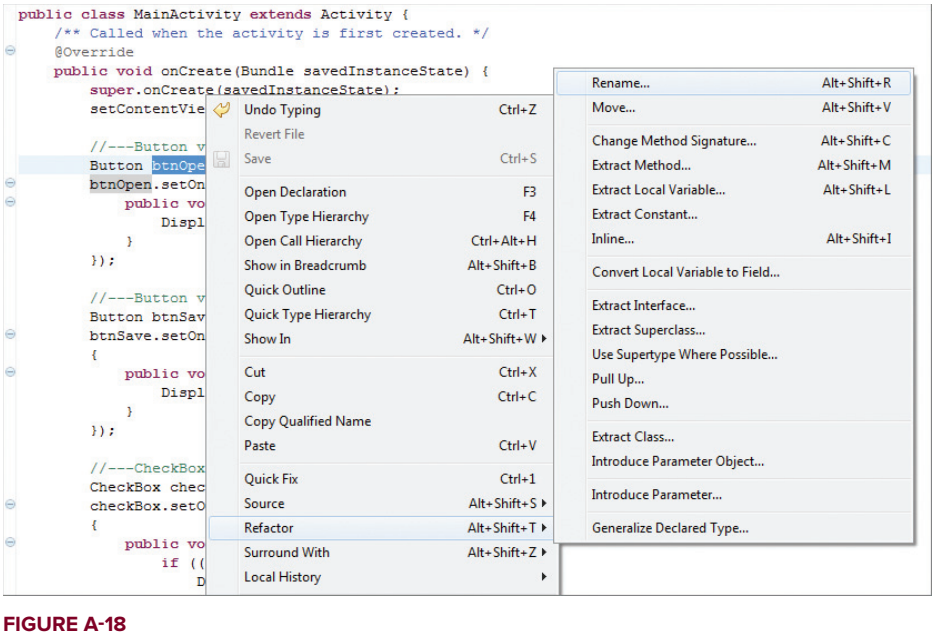

FIGURE A-18

After entering a new name for the object, all occurrences of the object will be changed automatically (see Figure A-19). Note that in order for refactoring to work correctly, your code must not have any syntax errors and must be able to be compiled correctly by the compiler.

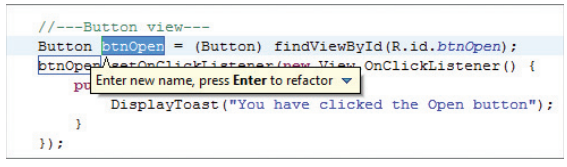

FIGURE A-19

Another area where refactoring is very useful is for extracting string constants from your UI files. As I mentioned earlier in Chapter 1, all the string constants that you use in your user interface should preferably be stored in the `strings.xml` file so that it is easy to perform localization later. However, it is very common during development to take the shortcut of entering the string constant directly. For example, you may set the `android:text` attribute of a `Button` view using a string constant:

```
<Button android:id="@+id/btnSave"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Save" />
```

Using the refactoring feature in Eclipse, you could select the string constant and then select the Refactor ➪ Android ➪ Extract Android String… (see Figure A-20).

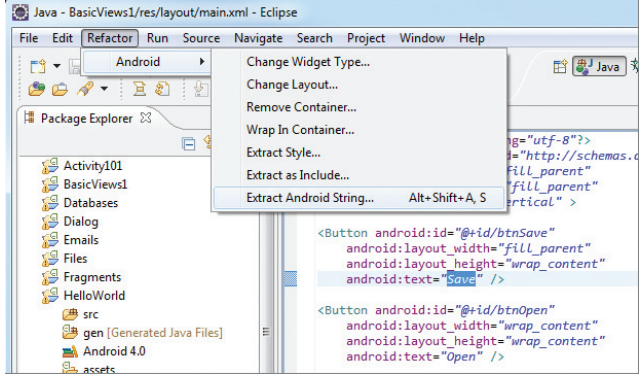

FIGURE A-20

You are then prompted to specify a name for this string constant (see Figure A-21). Click OK when you are done.

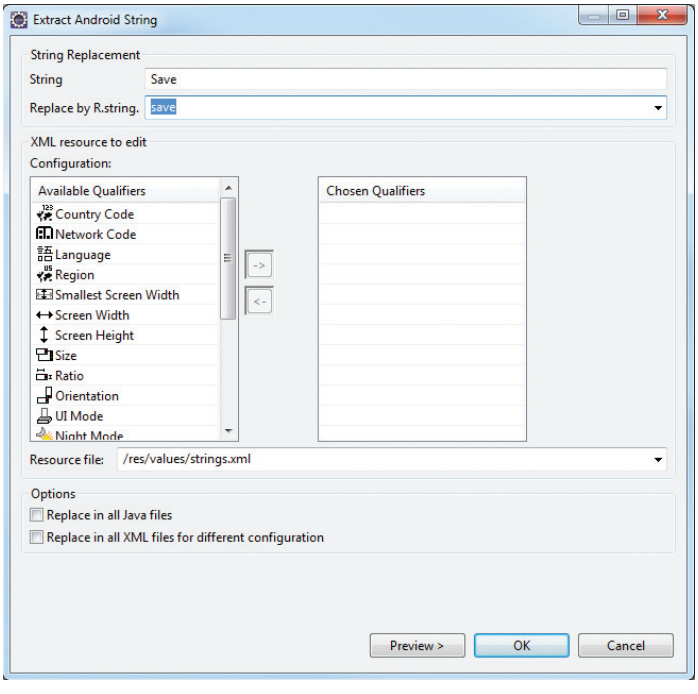

FIGURE A-21

After doing so, the value of the `android:text` attribute is now replaced with `@string/save`:

```
<Button android:id="@+id/btnSave"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/save" />
```

If you examine the `strings.xml` file, it will now contain a new entry named `save`:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, BasicViews1Activity!</string>
    <string name="app_name">BasicViews1</string>
    <string name="save">Save</string>

</resources>
```

A detailed discussion of refactoring is beyond the scope of this book. For more information on refactoring in Eclipse, refer to `www.ibm.com/developerworks/library/os-ecref/`.

DEBUGGING YOUR APPLICATION

Eclipse supports debugging your application on both Android emulators as well as real Android devices. When you press F11 in Eclipse, Eclipse first determines whether an Android emulator instance is already running or a real device is connected. If at least one emulator (or device) is

running, Eclipse will deploy the application onto the running emulator or the connected device. If no emulator is running and no device is connected, Eclipse automatically launches an instance of the Android emulator and deploys the application onto it.

If you have more than one emulator or device connected, Eclipse will prompt you to select the target emulator/device on which to deploy the application (see Figure A-22). Select the target device you want to use and click OK. Devices that do not have the minimum OS version required by your application will be marked with an X.

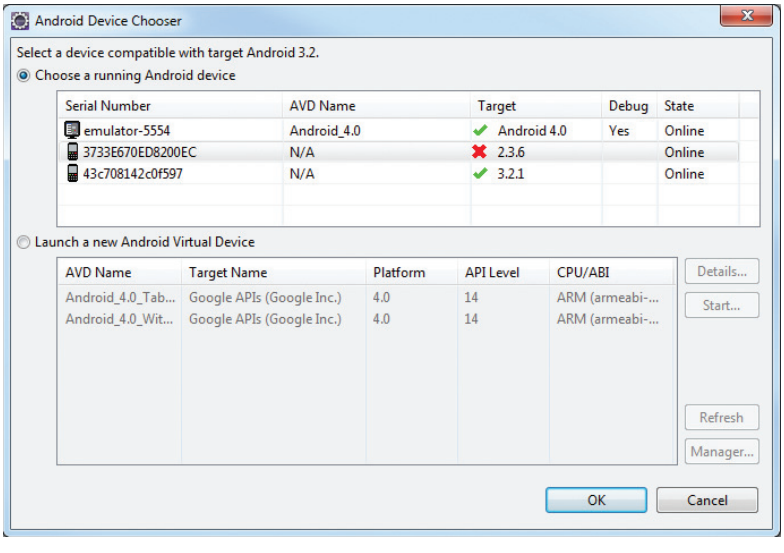

FIGURE A-22

If you want to launch a new emulator instance to test the application, select Window ➪ Android SDK and AVD Manager to launch the AVD Manager.

Setting Breakpoints

Setting breakpoints is a good way to temporarily pause the execution of the application and then examine the content of variables and objects.

To set a breakpoint, double-click on the leftmost column in the code editor. Figure A-23 shows a breakpoint set on a particular statement.

```
//---Button view---
Button btnOpen = (Button) findViewById(R.id.btnOpen);
btnOpen.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        String str = "You have clicked the Open button";
        DisplayToast(str);
    }
});
```

FIGURE A-23

When the application is running and the first breakpoint is reached, Eclipse will display a Confirm Perspective Switch dialog. Basically, it wants to switch to the Debug perspective. To prevent this window from appearing again, check the "Remember my decision" checkbox at the bottom and click Yes. Eclipse will highlight the breakpoint (see Figure A-24).

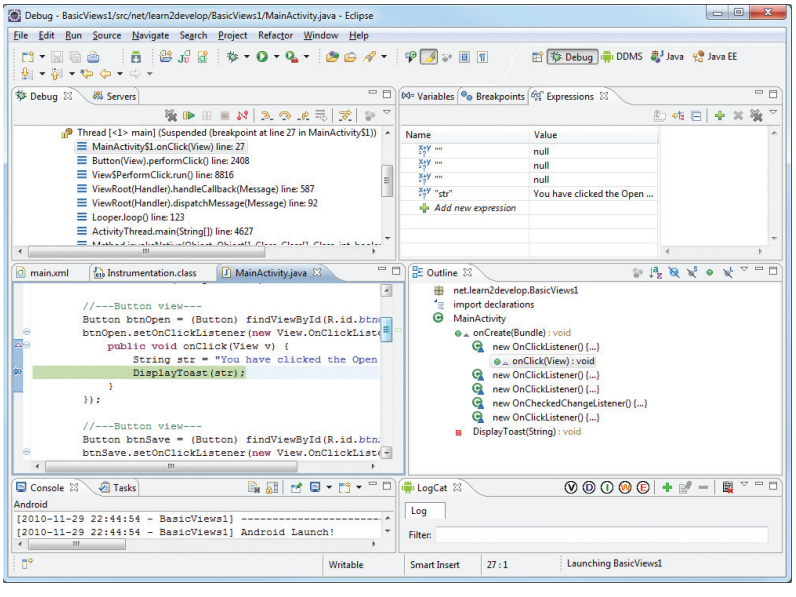

FIGURE A-24

At this point, you can right-click on any selected object/variable and view its content using the various options (e.g., Watch, Inspect, Display) shown in Figure A-25.

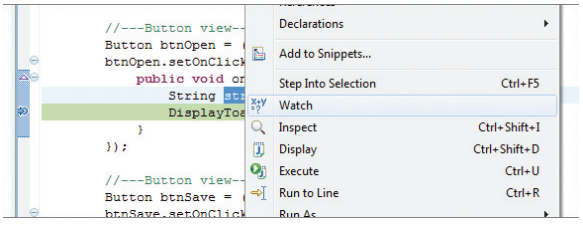

FIGURE A-25

Figure A-26 shows the Inspect option displaying the content of the `str` variable.

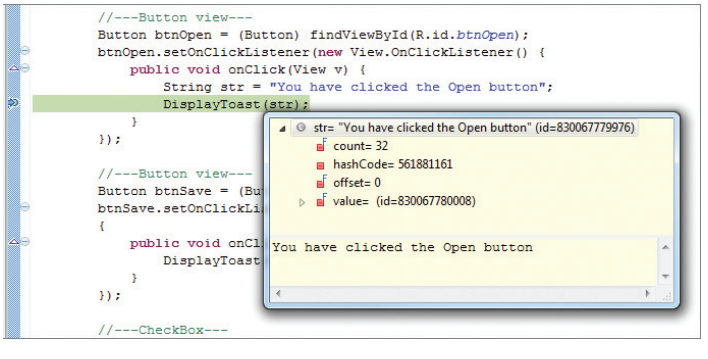

FIGURE A-26

You have several options at this point to continue the execution:

- **Step Into** — Press F5 to step into the next method call/statement.
- **Step Over** — Press F6 to step over the next method call without entering it.
- **Step Return** — Press F7 to return from a method that has been stepped into.
- **Resume Execution** — Press F8 to resume the execution.

Dealing with Exceptions

As you develop in Android, you will encounter numerous run-time exceptions that prevent your program from continuing. Examples of run-time exceptions include the following:

- Null reference exception (accessing an object that is null)
- Failure to specify the permissions required by your application
- Arithmetic operation exceptions

Figure A-27 shows the current state of an application when an exception occurred. In this example, I am trying to send an SMS message from my application and it crashes when the message is about to be sent.

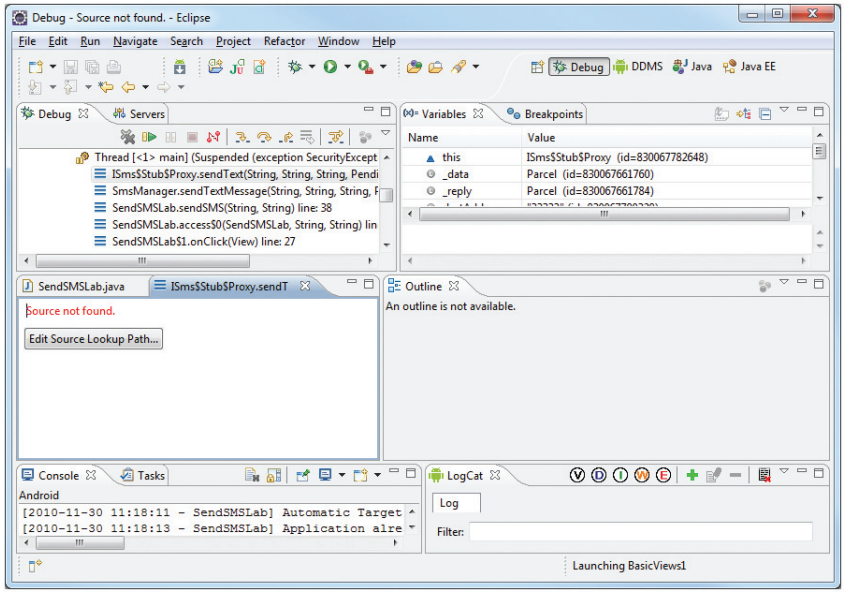

FIGURE A-27

The various windows do not really identify the cause of the exception. To find out more, press F6 in Eclipse so that it can step over the current statement. The Variables window, shown in Figure A-28, indicates the cause of the exception. In this case, the `SEND_SMS` permission is missing.

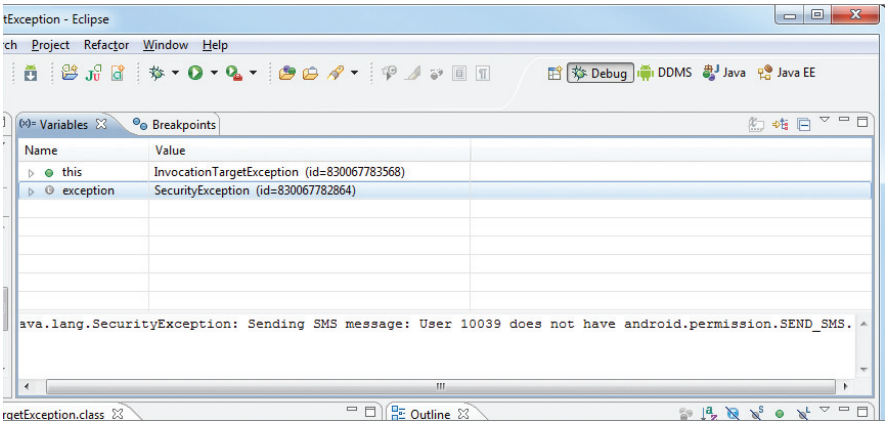

FIGURE A-28

To remedy this, all you need to do is add the following permission statement in the `AndroidManifest.xml` file:

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

B

Using the Android Emulator

The Android emulator ships with the Android SDK and is an invaluable tool to help test your application without requiring you to purchase a real device. While you should thoroughly test your applications on real devices before you deploy them, the emulator mimics most of the capabilities of real devices. It is a very handy tool that you should make use of during the development stage of your project. This appendix provides some common tips and tricks for mastering the Android emulator.

USES OF THE ANDROID EMULATOR

As discussed in Chapter 1, you can use the Android emulator to emulate the different Android configurations by creating Android Virtual Devices (AVDs).

If you want to emulate a real device, first create an AVD with the same screen resolution and abstracted LCD density as that of your real device (see the section "Emulating Devices with Different Screen Sizes" in this appendix for how to do this). You then launch the Android emulator by directly starting the AVD you have created in the AVD Manager window (see Figure B-1). Simply select the AVD and click the Start button. If you want the emulator to display using the same screen size as a real device, check the "Scale display to real size" option and set the "Screen Size (in)" option to the size of your real device. Enter the dpi of your

current monitor (if you don't know it, click the ? button and select your monitor size and resolution). The Android emulator will then display a screen size that is close to your real device. This useful option enables you to preview what your application will look like on different actual screen sizes.

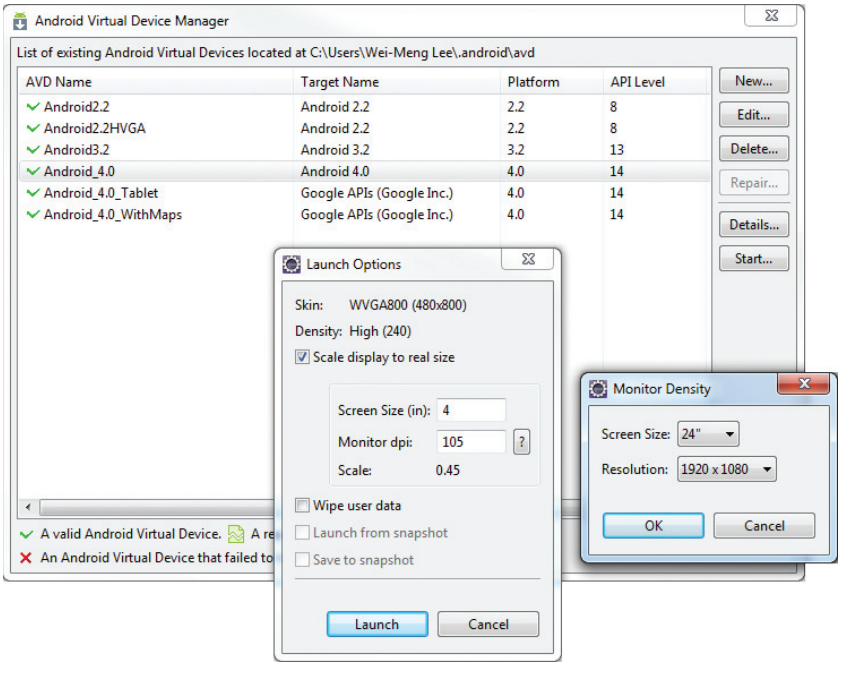

FIGURE B-1

NOTE *For best performance of the Android emulator, set the screen size to the smallest size that you can allow. Doing so will make the emulator run faster.*

Alternatively, when you run an Android project in Eclipse, the Android emulator is automatically invoked to test your application. You can customize the Android emulator for each of your Android projects in Eclipse. To do so, simply select Run ➪ Run Configurations. Select the project name listed

under Android Application on the left (see Figure B-2), and on the right you will see the Target tab. You can choose your preferred AVD to use for testing your application, as well as emulate different scenarios such as network speed and network latency.

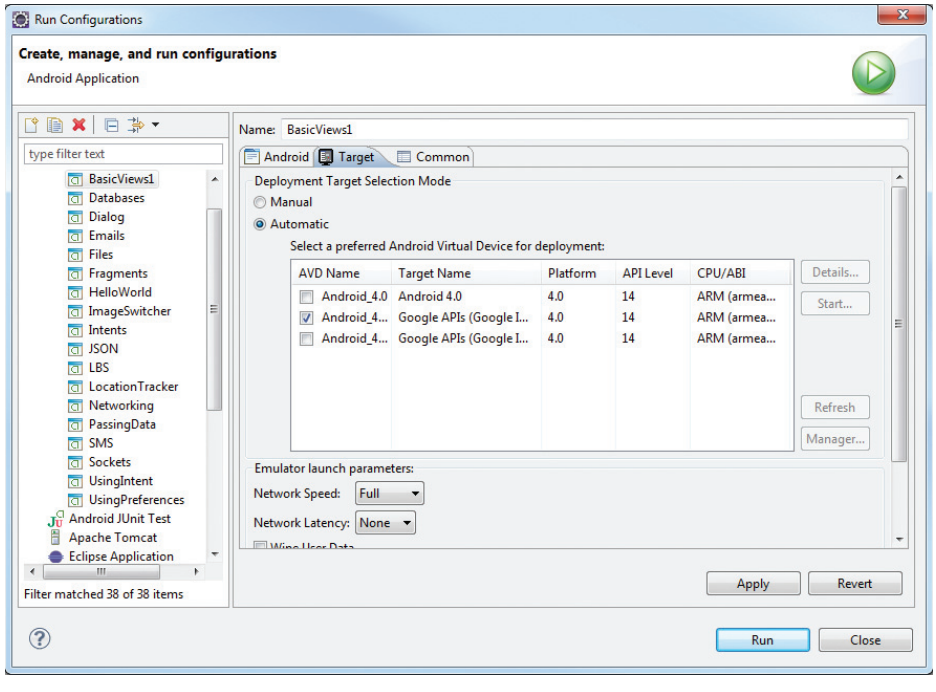

FIGURE B-2

CREATING SNAPSHOTS

In the latest version of the AVD Manager, you now have the option to save an emulator's state to a snapshot file. Saving an emulator's state to a snapshot file enables the emulator to be started quickly the next time you try to launch it, effectively bypassing the lengthy boot-up time. This is especially useful for the Android 3.0 (and later) emulator, which can take up to five minutes to boot up.

To use the snapshot feature, simply check the Snapshot Enabled checkbox when you create a new AVD (see Figure B-3).

When you launch the AVD from the Start . . . button, check the "Launch from snapshot" and "Save to snapshot" checkboxes (see Figure B-4). The first time you launch the emulator, it will boot up normally. When you close the emulator, it will then save the state to a snapshot file. The next time you launch the emulator, it will appear almost instantly, restoring its state from the snapshot file.

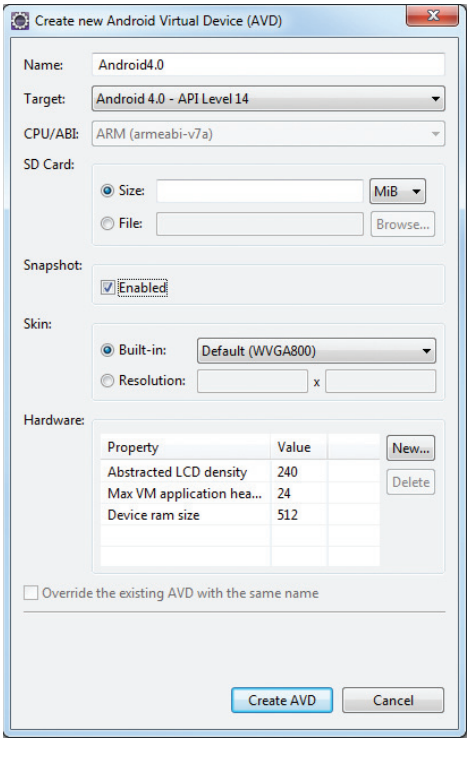

FIGURE B-3

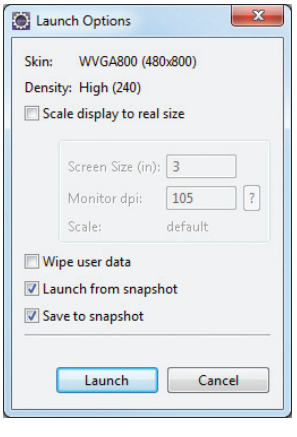

FIGURE B-4

SD CARD EMULATION

When you create a new AVD, you can emulate the existence of an SD card (see Figure B-5). Simply enter the size of the SD card that you want to emulate (in the figure, it is 200MB).

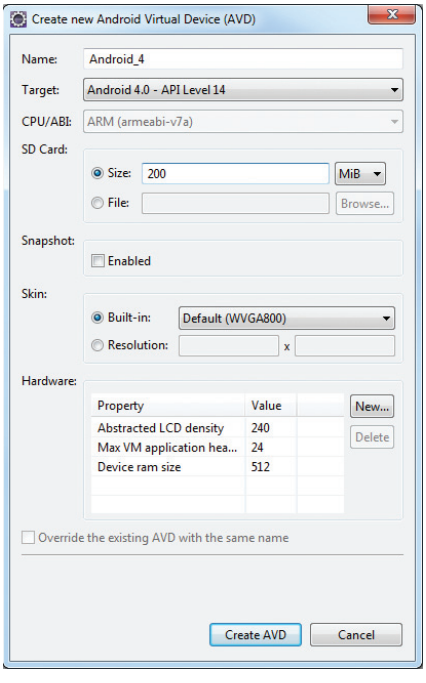

FIGURE B-5

Alternatively, you can simulate the presence of an SD card in the Android emulator by creating a disk image first and then attaching it to the AVD. The `mksdcard.exe` utility (located in the `tools` folder of the Android SDK) enables you to create an ISO disk image. The following command creates an ISO image that is 2GB in size (see also Figure B-6):

```
mksdcard 2048M sdcard.iso
```

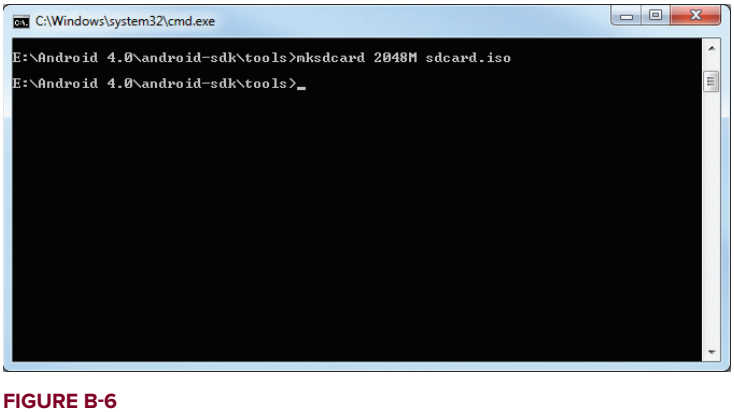

FIGURE B-6

Once the image is created, you can specify the location of the ISO file, as shown in Figure B-7.

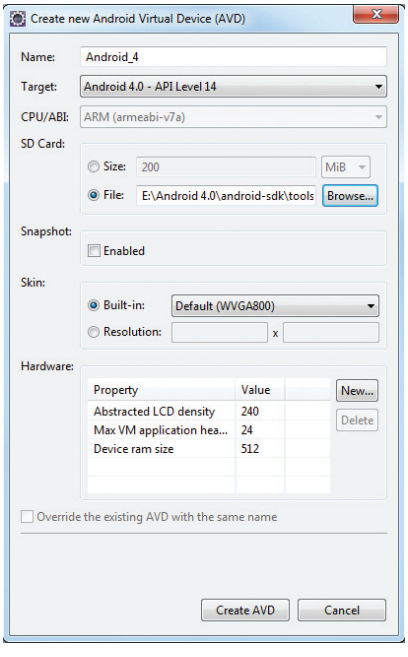

FIGURE B-7

EMULATING DEVICES WITH DIFFERENT SCREEN SIZES

Besides emulating an SD card, you can also emulate devices with different screen sizes. Figure B-8 indicates that the AVD is emulating the HVGA skin, which has a resolution of 320 × 480 pixels. Note that the Abstracted LCD density is 160, which means that this screen has a pixel density of 160 pixels per inch.

For each target that you select, a list of skins is available. The following screen resolutions are supported by Android:

- **HVGA** — 320 × 480
- **QVGA** — 240 × 320
- **WQVGA400** — 240 × 400
- **WQVGA432** — 240 × 432
- **WVGA800** — 480 × 800
- **WVGA854** — 480 × 854

In addition to using the built-in screen resolution, you can also specify your own custom resolution. For example, you can emulate the Samsung Galaxy Tab 10.1 by creating an AVD with the specifications shown in Figure B-9.

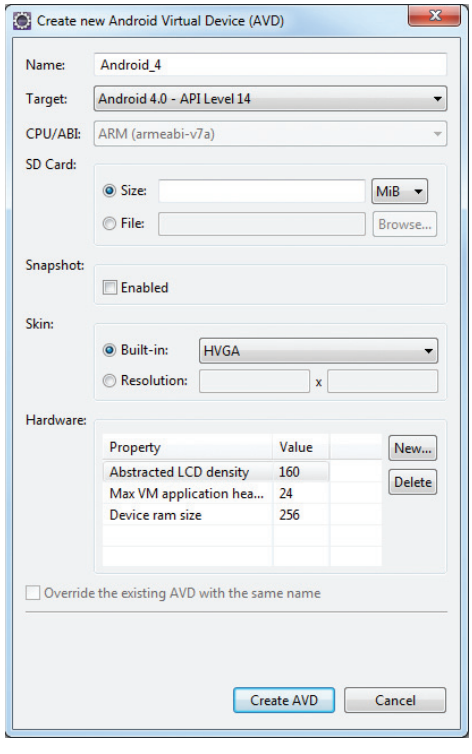

FIGURE B-8

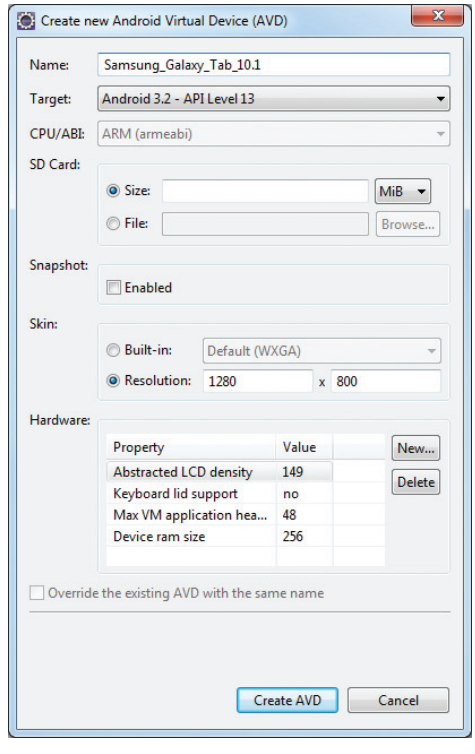

FIGURE B-9

When the AVD is started, you will see the Android emulator emulating a Honeycomb tablet (see Figure B-10).

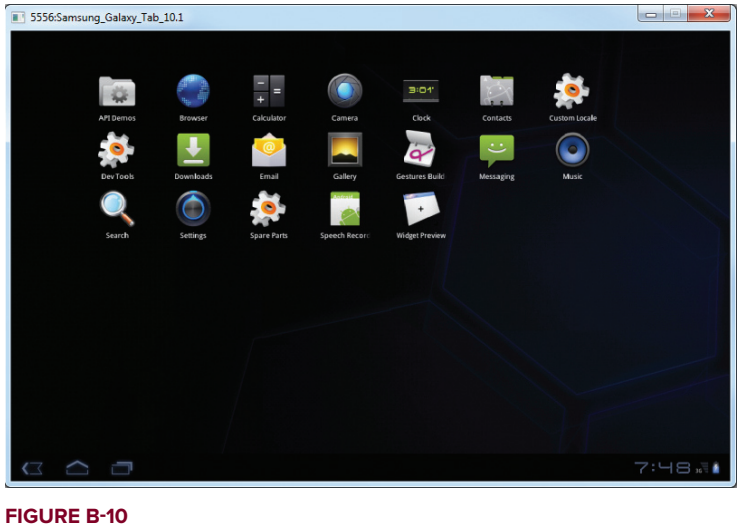

FIGURE B-10

EMULATING PHYSICAL CAPABILITIES

In addition to emulating devices of different screen sizes, you also have the option to emulate different hardware capabilities. When creating a new AVD, clicking the New . . . button will display a dialog for choosing the type of hardware you want to emulate (see Figure B-11).

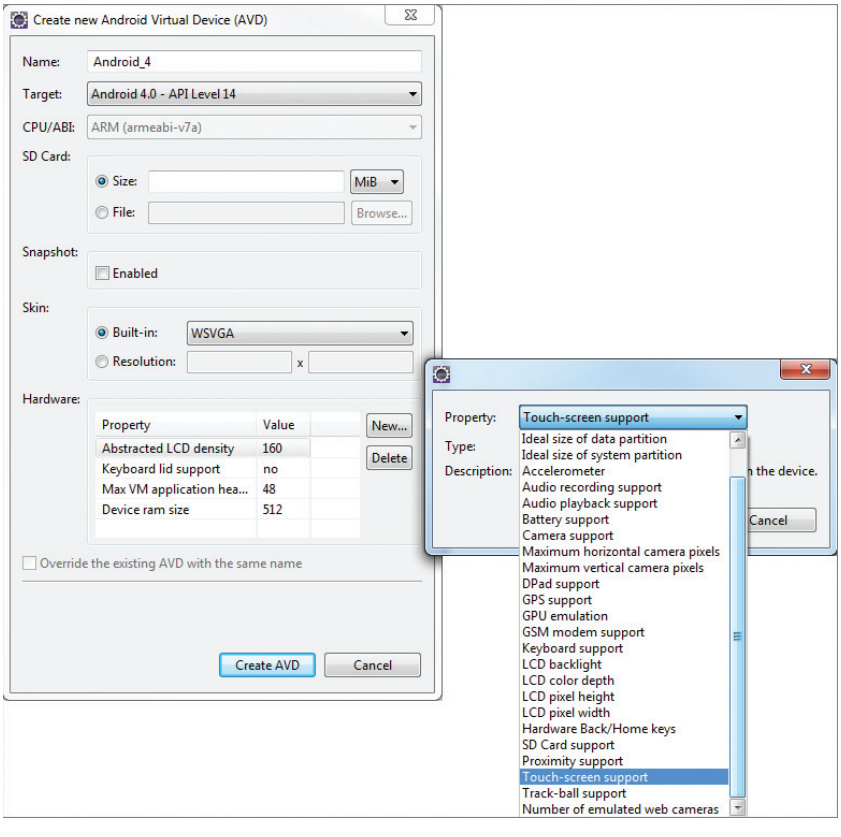

FIGURE B-11

For example, if you want to emulate an Android device with no touch screen, select the "Touch-screen support" property and click OK. Back in the AVD dialog, change the value of the property from yes to no (see Figure B-12).

This will create an AVD with no touch-screen support (i.e., users won't be able to use their mouse to click on the screen).

You can also simulate location data using the Android emulator. Chapter 9 discusses this in more detail.

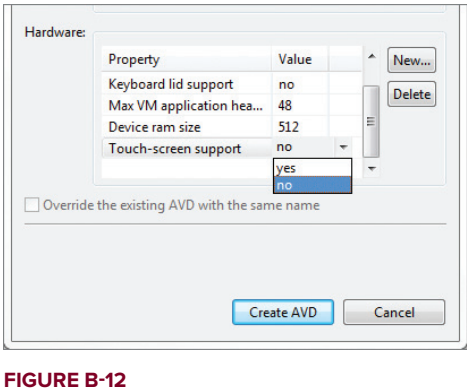

FIGURE B-12

KEYBOARD SHORTCUTS

The Android emulator supports several keyboard shortcuts that enable you to mimic the behavior of a real handset. The following list describes some of the shortcuts that you can use with the emulator:

- **Esc** — Back
- **Home** — Main screen
- **F2** — Toggles context-sensitive menu
- **F3** — Call Log
- **F4** — Hang up/end call button
- **F7** — Power button
- **F5** — Search
- **F6** — Toggle trackball mode
- **F8** — Toggles data network (3G)
- **Ctrl+F5** — Ringer volume up
- **Ctrl+F6** — Ringer volume down
- **Ctrl+F11/Ctrl+F12** — Toggle orientation

For example, by pressing Ctrl+F11, you can change the orientation of the emulator to portrait mode (see Figure B-13).

FIGURE B-13

One useful tip to make your development more productive is to keep your Android emulator running during development — avoid closing and restarting it. Because the emulator takes time to boot up, it is much better to leave it running when you are debugging your applications.

SENDING SMS MESSAGES TO THE EMULATOR

You can emulate sending SMS messages to the Android emulator using either the Dalvik Debug Monitor Service (DDMS) tool (available in Eclipse) or the Telnet client.

NOTE *The Telnet client is not installed by default in Windows 7. To install it, type the following at the Windows command prompt:* `pkgmgr /iu:"TelnetClient"`.

Take a look at how this is done in Telnet. First, ensure that the Android emulator is running. In order to telnet to the emulator, you need to know the port number of the emulator. You can obtain this by looking at the title bar of the Android emulator window. It normally starts with 5554, with each subsequent emulator having a port number incremented by two, such as 5556, 5558, and so on. Assuming that you currently have one Android emulator running, you can telnet to it using the following command (replace 5554 with the actual number of your emulator):

```
C:\telnet localhost 5554
```

To send an SMS message to the emulator, use the following command:

```
sms send +1234567 Hello my friend!
```

The syntax of the `sms send` command is as follows:

```
sms send <phone_number> <message>
```

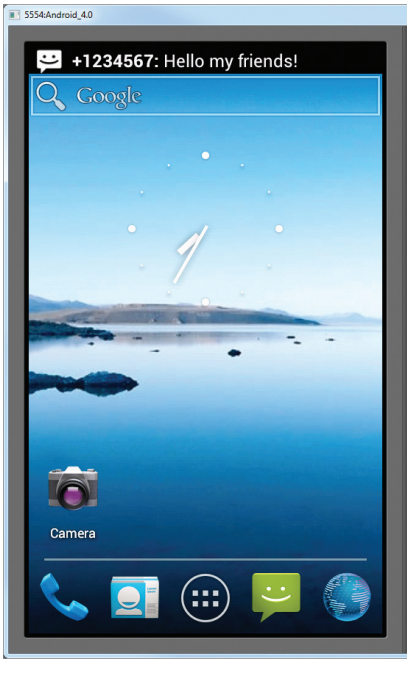

FIGURE B-14

Figure B-14 shows the emulator receiving the sent SMS message.

Besides using Telnet for sending SMS messages, you can also use the DDMS perspective in Eclipse. If the DDMS perspective is not visible within Eclipse, you can display it by clicking the Open Perspective button (highlighted in Figure B-15) and selecting Other.

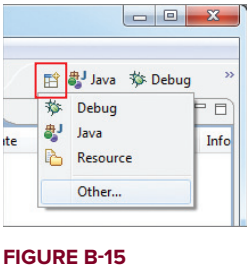

FIGURE B-15

Select the DDMS perspective (see Figure B-16) and click OK.

Once the DDMS perspective is displayed, you will see the Devices tab (see Figure B-17), which shows the list of emulators currently running. Select the emulator instance to which you want to send the SMS message, and under the Emulator Control tab you will see the Telephony Actions section. In the Incoming number field, enter an arbitrary phone number and check the SMS radio button. Enter a message and click the Send button.

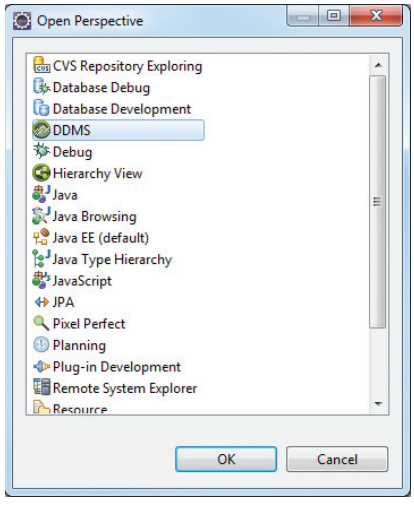

FIGURE B-16

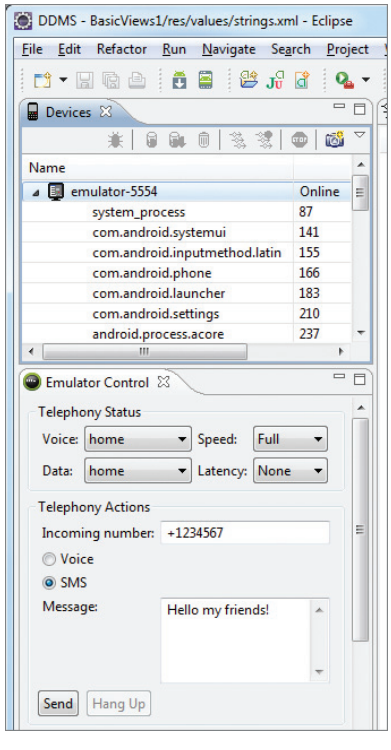

FIGURE B-17

The selected emulator will now receive the incoming SMS message.

If you have multiple AVDs running at the same time, you can send SMS messages between each AVD by using the port number of the emulator as the phone number. For example, if you have an emulator running on port number 5554 and another on 5556, their phone numbers will be 5554 and 5556, respectively.

MAKING PHONE CALLS

In addition to sending SMS messages to the emulator, you can also use the Telnet client to make a phone call to the emulator. To do so, simply use the following commands.

To telnet to the emulator, use this command (replace 5554 with the actual number of your emulator):

```
C:\telnet localhost 5554
```

To make a phone call to the emulator, use this command:

```
gsm call +1234567
```

The syntax of the `gsm send` command is as follows:

```
gsm call <phone_number>
```

Figure B-18 shows the emulator receiving an incoming call.

As with sending SMS messages, you can also use the DDMS perspective to make a phone call to the emulator. Figure B-19 shows how to make a phone call using the Telephony Actions section.

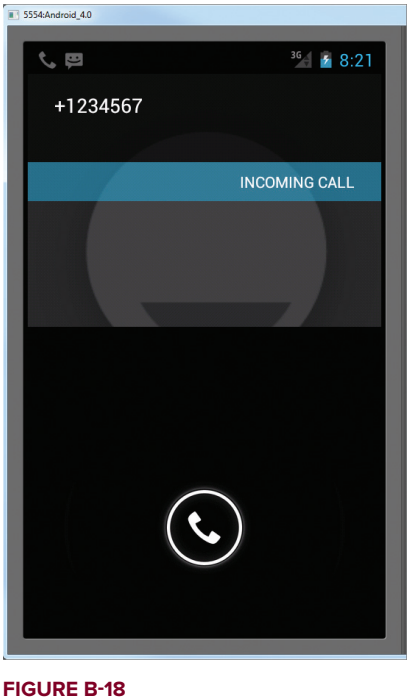

FIGURE B-18

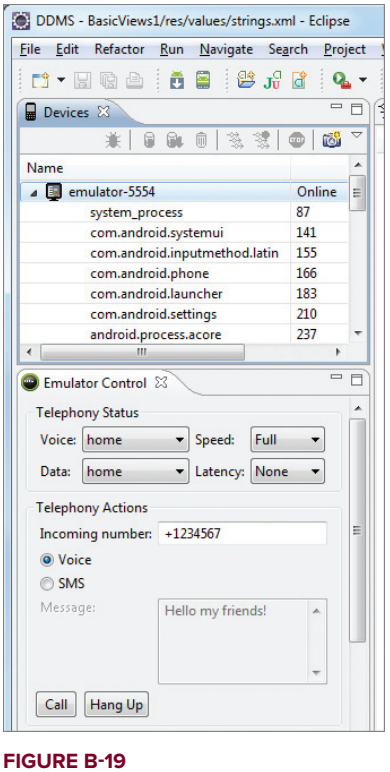

FIGURE B-19

You can also make phone calls between AVDs by using their port numbers as phone numbers.

TRANSFERRING FILES INTO AND OUT OF THE EMULATOR

Occasionally, you may need to transfer files into or out of the emulator. The easiest way is to use the DDMS perspective. From the DDMS perspective, select the emulator (or device if you have a real Android device connected to your computer) and click the File Explorer tab to examine its file systems (see Figure B-20).

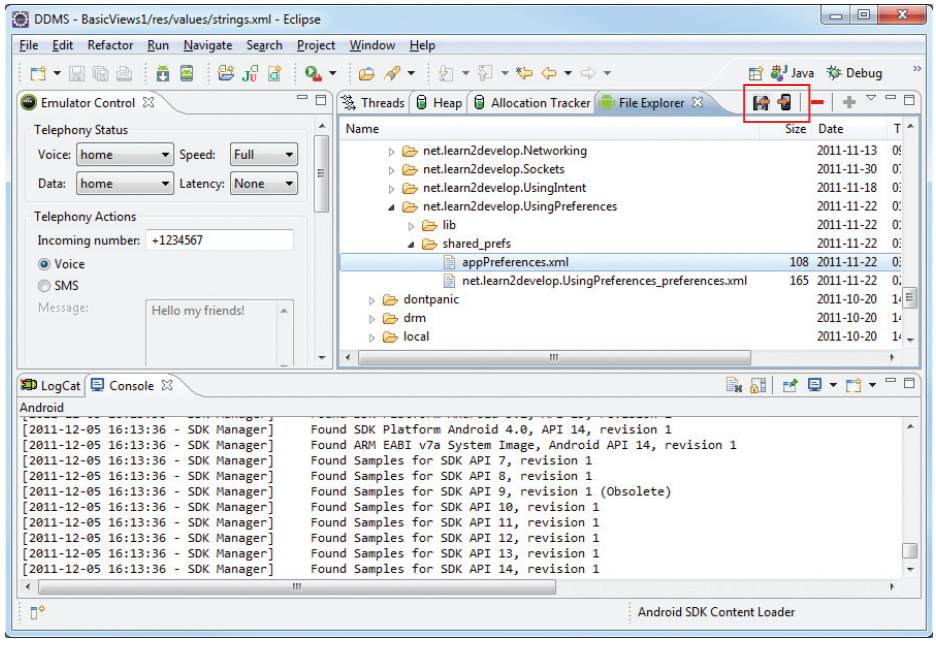

FIGURE B-20

NOTE *When using the* `adb.exe` *utility to pull or push files from or into the emulator, ensure that only one AVD is running.*

The two buttons highlighted in Figure B-20 enable you to either pull a file from the emulator or push a file into the emulator.

Alternatively, you can also use the `adb.exe` utility that is shipped with the Android SDK to push or pull files to and from the emulator. This utility is located in the `<Android_SDK_Folder>\platform-tools\` folder.

To copy a file from the connected emulator/device onto the computer, use the following command:

```
adb.exe pull <source path on emulator>
```

Figure B-21 shows how you can extract an XML file from the emulator and save it onto your computer.

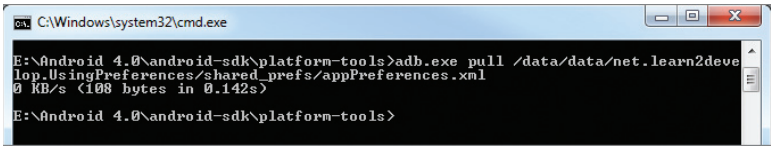

FIGURE B-21

To copy a file into the connected emulator/device, use the following command:

```
adb.exe push <filename> <destination path on emulator>
```

The command in Figure B-22 copies the `NOTICE.txt` file located in the current directory and saves it onto the emulator's `/data/data/net.learn2develop.UsingPreferences/shared_prefs` folder.

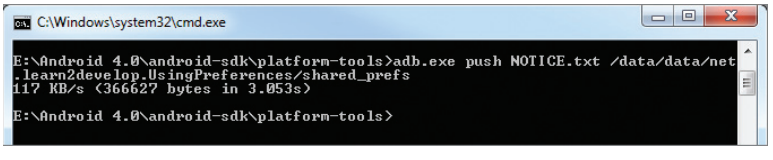

FIGURE B-22

If you need to modify the permissions of the files in the emulator, you can use the `adb.exe` utility together with the shell option, like this:

```
adb.exe shell
```

Figure B-23 shows how you can change the permissions of the `NOTICE.txt` file by using the `chmod` command.

Using the `adb.exe` utility, you can issue Unix commands against your Android emulator.

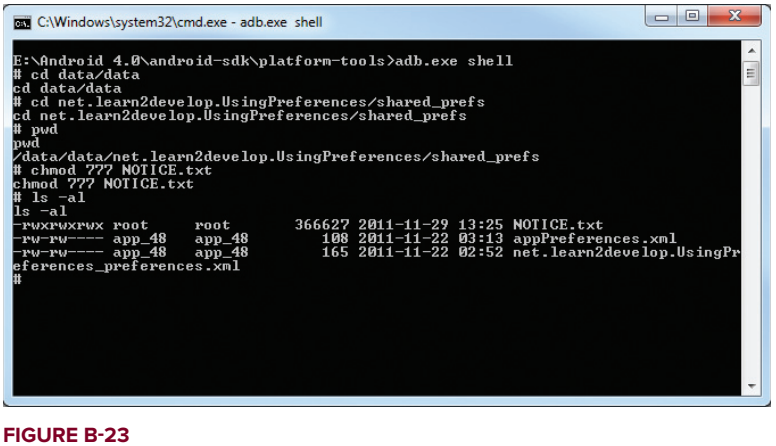

FIGURE B-23

RESETTING THE EMULATOR

There are times where you want to install your application onto a fresh AVD. For example, you may have installed other applications earlier that might interfere with your current application (a good example is an SMS-intercepting application that might intercept SMS messages meant for your application). In this case, you can either uninstall each application from the Settings application, or (a much easier way) you can wipe out the image for the emulator so as to restore it to its original state.

All applications and files that you have deployed to the Android emulator are stored in a file named `userdata-qemu.img` located in the `C:\Users\<username>\.android\avd\<avd_name>.avd` folder. For example, I have an AVD named AndroidTabletWithMaps; hence, the `userdata-qemu.img` file is located in the `C:\Users\Wei-Meng Lee\.android\avd\AndroidTabletWithMaps.avd` folder.

If you want to restore the emulator to its original state (i.e., reset it), simply delete the `userdata-qemu.img` file. All the previously installed applications on this AVD will now be gone.

C

Answers to Exercises

This appendix includes the answers to the end of chapter exercises.

CHAPTER 1 ANSWERS

1. An AVD is an Android Virtual Device. It represents an Android emulator, which emulates a particular configuration of an actual Android device.
2. The `android:versionCode` attribute is used to programmatically check whether an application can be upgraded. It should contain a running number (an updated application is set to a higher number than the older version). The `android:versionName` attribute is used mainly for displaying to the user. It is a string, such as `"1.0.1"`.
3. The `strings.xml` file is used to store all string constants in your application. This enables you to easily localize your application by simply replacing the strings and then recompiling your application.

CHAPTER 2 ANSWERS

1. The Android OS will display a dialog from which users can choose which activity they want to use.
2. Use the following code:

```
Intent i = new
    Intent(android.content.Intent.ACTION_VIEW,
      Uri.parse("http://www.amazon.com"));
startActivity(i);
```

3. In an intent filter, you can specify the following: action, data, type, and category.
4. The `Toast` class is used to display alerts to the user; it disappears after a few seconds. The `NotificationManager` class is used to display notifications on the device's status bar. The alert displayed by the `NotificationManager` class is persistent and can only be dismissed by the user when selected.
5. You can either use the `<fragment>` element in the XML file, or use the `FragmentManager` and `FragmentTransaction` classes to dynamically add/remove fragments from an activity.
6. One of the main differences between activities and fragments is that when an activity goes into the background, the activity is placed in the back stack. This allows an activity to be resumed when the user presses the Back button. Conversely, fragments are not automatically placed in the back stack when they go into the background.

CHAPTER 3 ANSWERS

1. The `dp` unit is density independent and 1dp is equivalent to one pixel on a 160 dpi screen. The `px` unit corresponds to an actual pixel on screen. You should always use the `dp` unit because it enables your activity to scale properly when run on devices of varying screen size.
2. With the advent of devices with different screen sizes, using the `AbsoluteLayout` makes it difficult for your application to have a consistent look and feel across devices.
3. The `onPause()` event is fired whenever an activity is killed or sent to the background. The `onSaveInstanceState()` event is like the `onPause()` event, except that it is not always called, such as when the user presses the back button to kill the activity.
4. The three events are `onPause()`, `onSaveInstanceState()`, and `onRetainNonConfigurationInstance()`. You generally use the `onPause()` method to preserve the activity's state because the method is always called when the activity is about to be destroyed. However, for screen orientation changes, it is easier to use the `onSaveInstanceState()` method to save the state of the activity (such as the data entered by the user) using a Bundle object. The `onRetainNonConfigurationInstance()` method is useful for momentarily saving data (such as images or files downloaded from a web service) which might be too large to fit into a Bundle object.
5. Adding action items to the Action Bar is similar to creating menu items for an options menu — simply handle the `onCreateOptionsMenu()` and `onOptionsItemSelected()` events.

CHAPTER 4 ANSWERS

1. You should check the `isChecked()` method of each `RadioButton` to determine whether it has been checked.
2. You can use the `getResources()` method.

3. The code snippet to obtain the current date is as follows:

```
//--get the current date--
Calendar today = Calendar.getInstance();
yr = today.get(Calendar.YEAR);
month = today.get(Calendar.MONTH);
day = today.get(Calendar.DAY_OF_MONTH);
showDialog(DATE_DIALOG_ID);
```

4. The three specialized fragments are `ListFragment`, `DialogFragment`, and `PreferenceFragment`. The `ListFragment` is useful for displaying a list of items, such as an RSS listing of news items. The `DialogFragment` allows you to display a dialog window modally and is useful to get a response from the user before allowing him to continue with your application. The `PreferenceFragment` displays a window containing your application's preferences and allows the user to edit them directly in your application.

CHAPTER 5 ANSWERS

1. The `ImageSwitcher` enables images to be displayed with animation. You can animate the image when it is being displayed, as well as when it is being replaced by another image.
2. The two methods are `onCreateOptionsMenu()` and on`OptionsItemSelected()`.
3. The two methods are `onCreateContextMenu()` and `onContextItemSelected()`.
4. To prevent launching the device's web browser, you need to implement the `WebViewClient` class and override the `shouldOverrideUrlLoading()` method.

CHAPTER 6 ANSWERS

1. You can do so using the `PreferenceActivity` class.
2. The method name is `getExternalStorageDirectory()`.
3. The permission is `WRITE_EXTERNAL_STORAGE`.

CHAPTER 7 ANSWERS

1. The code is as follows:

```
Cursor c;
if (android.os.Build.VERSION.SDK_INT <11) {
    //---before Honeycomb---
    c = managedQuery(allContacts, projection,
            ContactsContract.Contacts.DISPLAY_NAME + " LIKE ?",
            new String[] {"%jack"},
            ContactsContract.Contacts.DISPLAY_NAME + " ASC");
} else {
    //---Honeycomb and later---
```

```
        CursorLoader cursorLoader = new CursorLoader(
                this,
                allContacts,
                projection,
                ContactsContract.Contacts.DISPLAY_NAME + " LIKE ?",
                new String[] {"%jack"},
                ContactsContract.Contacts.DISPLAY_NAME + " ASC");
        c = cursorLoader.loadInBackground();
    }
```

2. The methods are `getType()`, `onCreate()`, `query()`, `insert()`, `delete()`, and `update()`.
3. The code is as follows:

```
<provider android:name="BooksProvider"
        android:authorities="net.learn2develop.provider.Books" />
```

CHAPTER 8 ANSWERS

1. You can either programmatically send an SMS message from within your Android application or invoke the built-in Messaging application to send it on your application's behalf.
2. The two permissions are `SEND_SMS` and `RECEIVE_SMS`.
3. The Broadcast receiver should fire a new intent to be received by the activity. The activity should implement another `BroadcastReceiver` to listen for this new intent.

CHAPTER 9 ANSWERS

1. The likely reasons are as follows:
   - No Internet connection
   - Incorrect placement of the `<uses-library>` element in the `AndroidManifest.xml` file
   - Missing `INTERNET` permission in the `AndroidManifest.xml` file
2. Geocoding is the act of converting an address into its coordinates (latitude and longitude). Reverse geocoding converts a pair of location coordinates into an address.
3. The two providers are as follows:
   - `LocationManager.GPS_PROVIDER`
   - `LocationManager.NETWORK_PROVIDER`
4. The method is `addProximityAlert()`.

CHAPTER 10 ANSWERS

1. The permission is `INTERNET`.
2. The classes are `JSONArray` and `JSONObject`.
3. The class is `AsyncTask`.

CHAPTER 11 ANSWERS

1. A separate thread should be used because a service runs on the same process as the calling activity. If a service is long-running, you need to run it on a separate thread so that it does not block the activity.
2. The `IntentService` class is similar to the `Service` class, except that it runs the tasks in a separate thread and automatically stops the service when the task has finished execution.
3. The three methods are `doInBackground()`, `onProgressUpdate()`, and `onPostExecute()`.
4. The service can broadcast an intent, and the activity can register an intent using an `IntentFilter` class.
5. The recommended method is to create a class that subclasses the `AsyncTask` class. This will ensure that the UI is updated in a thread-safe manner.

CHAPTER 12 ANSWERS

1. You specify the minimum Android version required using the `minSdkVersion` attribute in the `AndroidManifest.xml` file.
2. To generate a certificate, you can either use the `keytool.exe` utility from the Java SDK or use Eclipse's Export feature.
3. Go to the Settings application and select the Security item. Check the "Unknown sources" item.

INDEX

A

B

C

D

E

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X–Y–Z