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*.

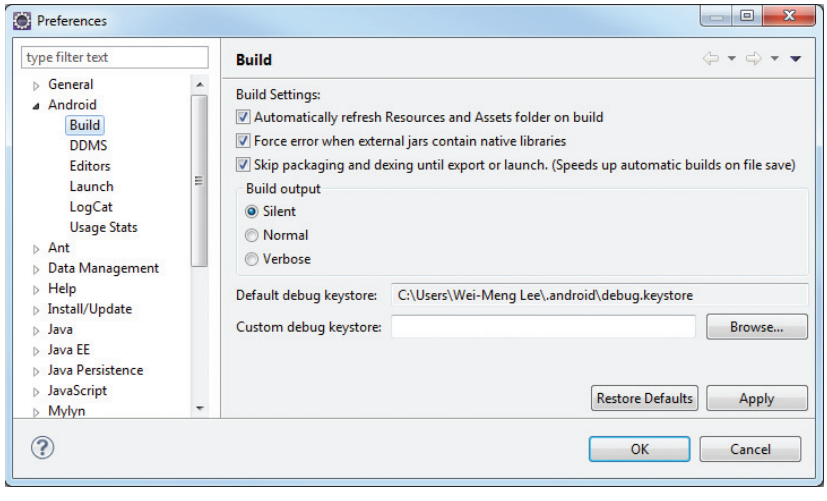

FIGURE 12-1

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.

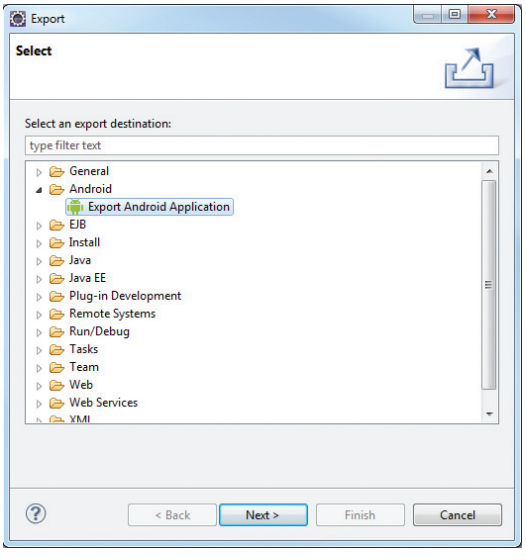

FIGURE 12-2

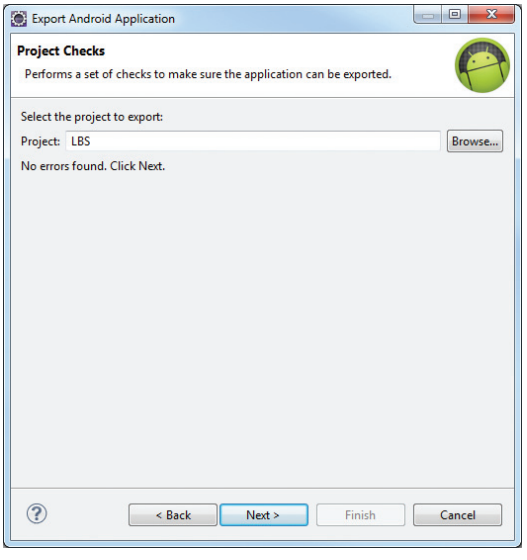

FIGURE 12-3

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.

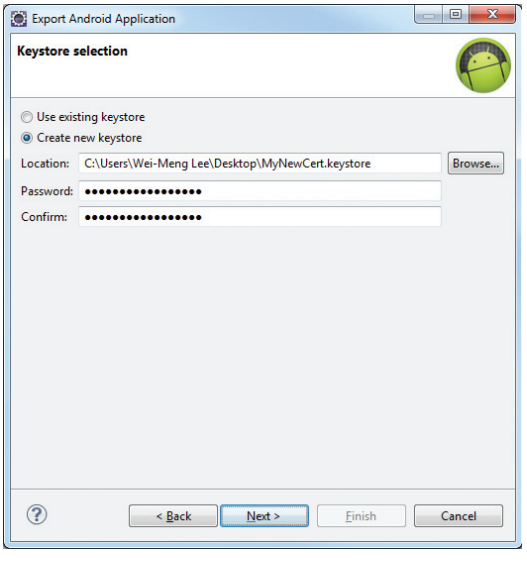

FIGURE 12-4

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

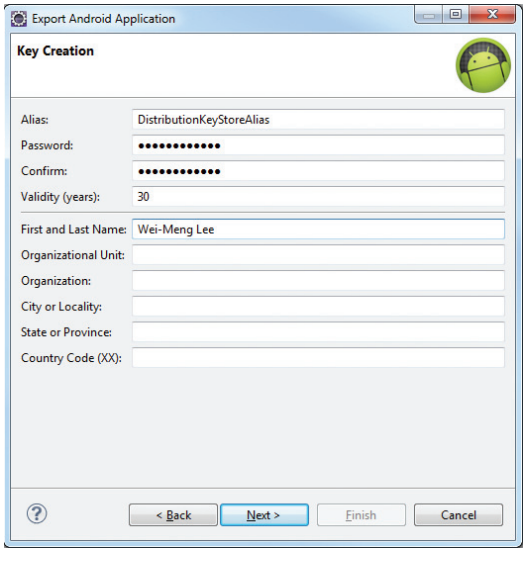

FIGURE 12-5

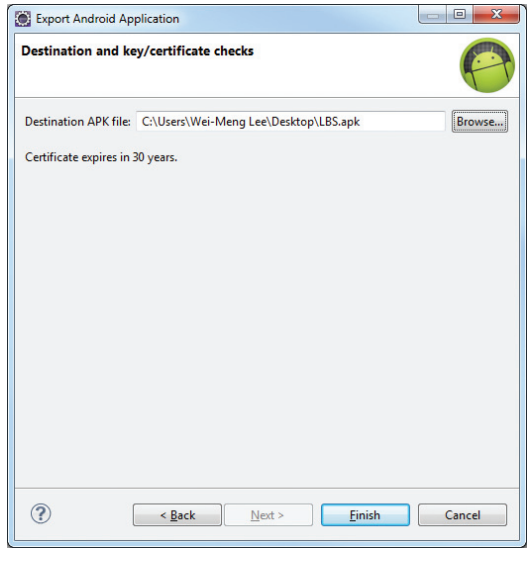

FIGURE 12-6

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
```

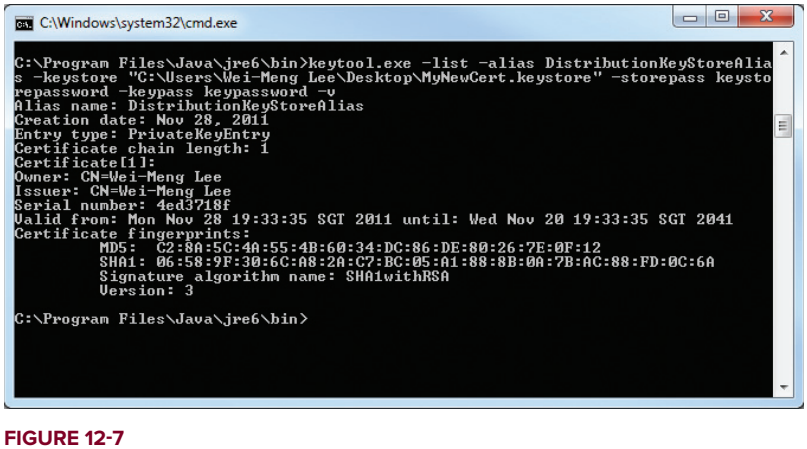

FIGURE 12-7

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.

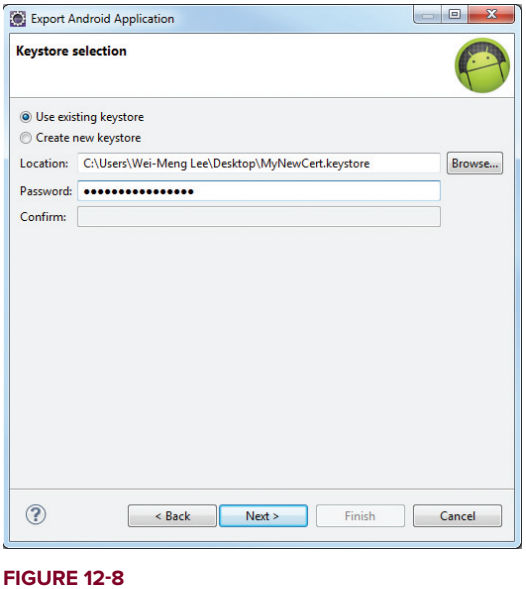

FIGURE 12-8

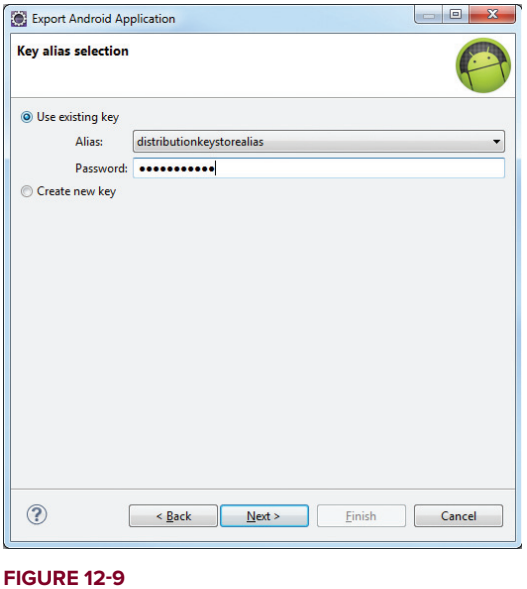

FIGURE 12-9

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.

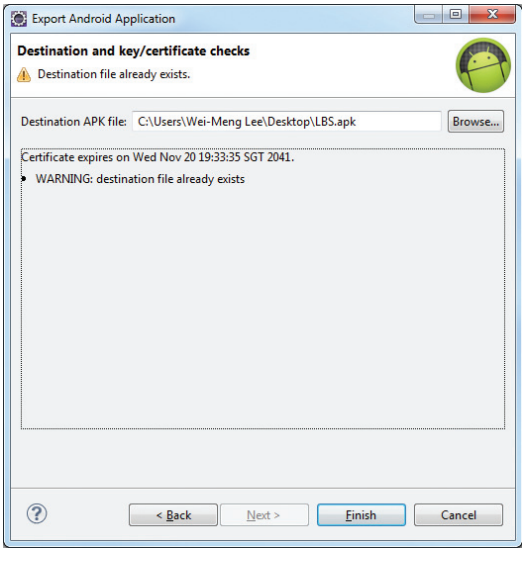

FIGURE 12-10

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).

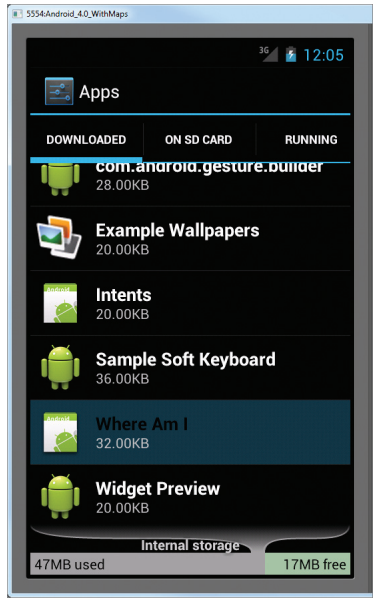

FIGURE 12-11

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.

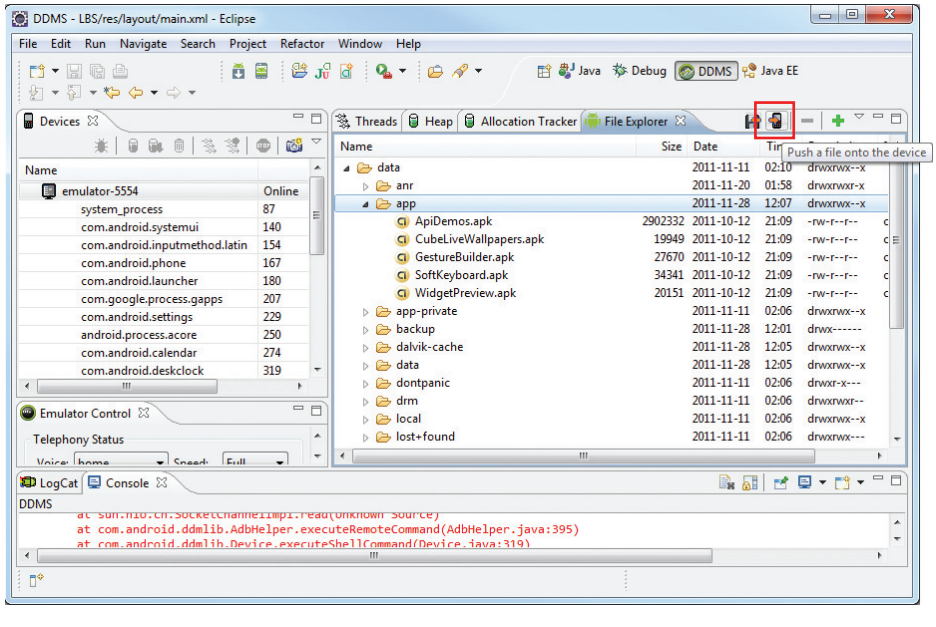

FIGURE 12-12

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.

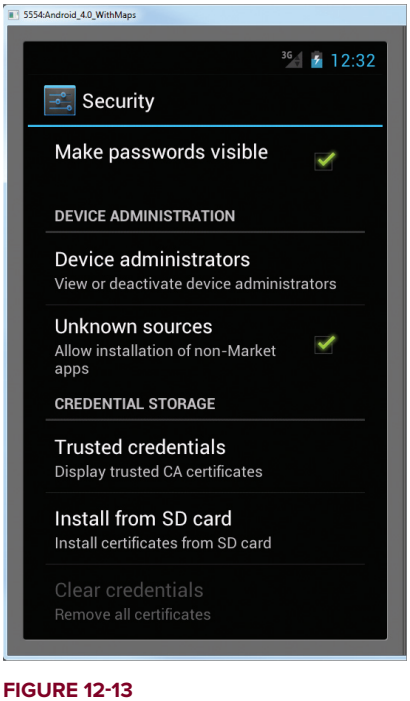

FIGURE 12-13

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.

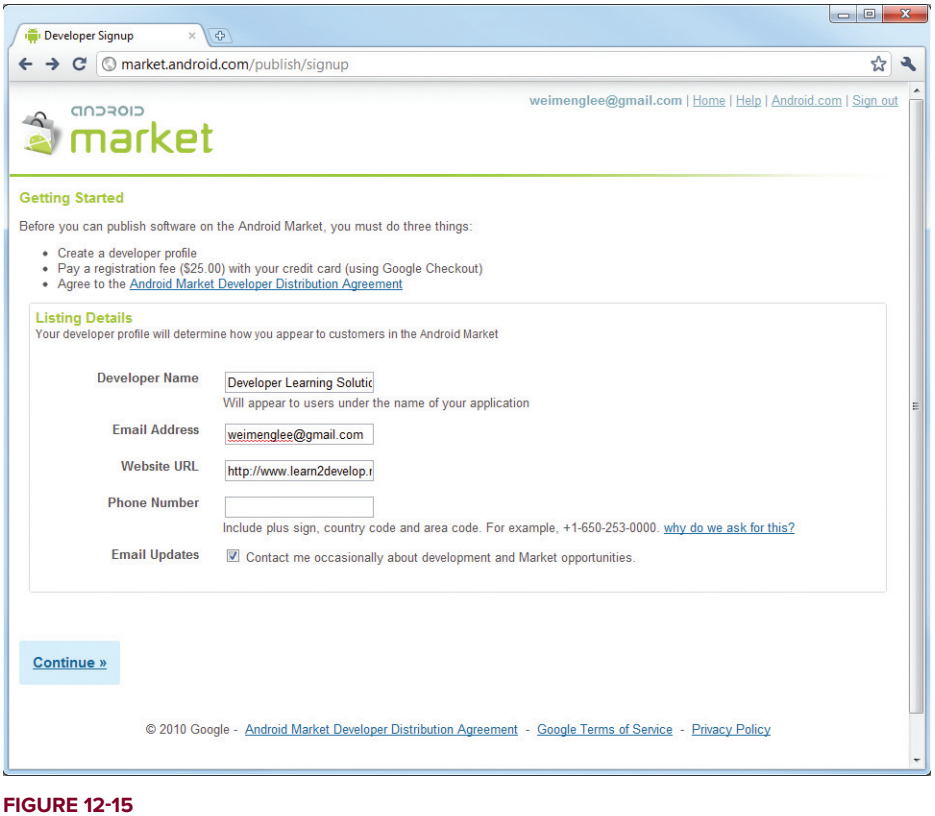

FIGURE 12-15

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.

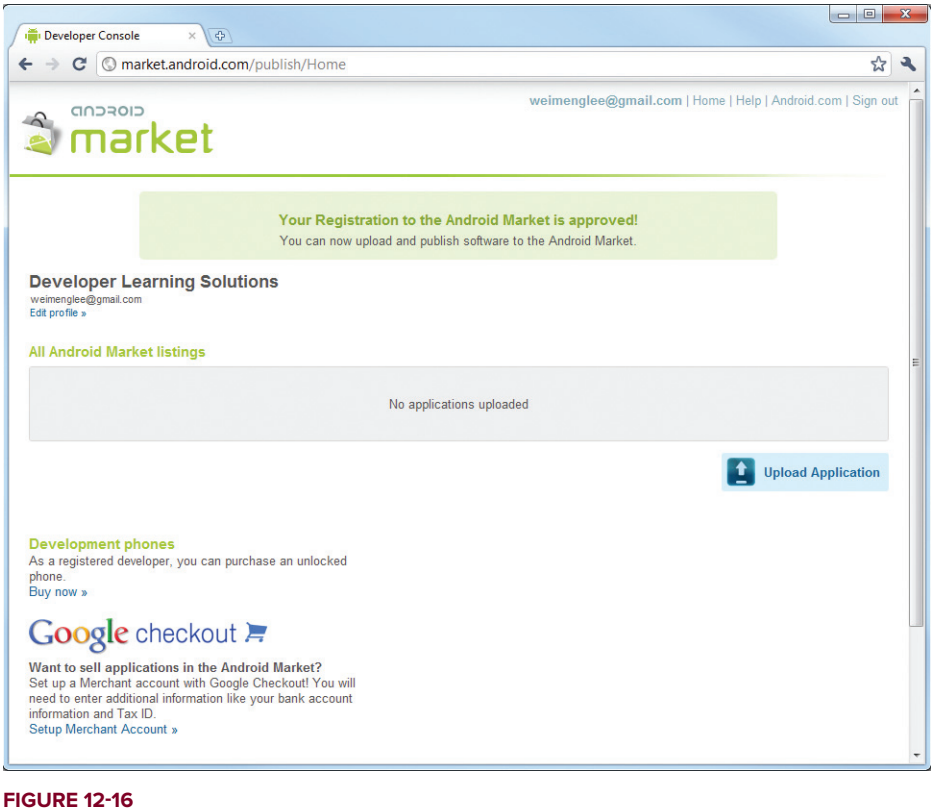

FIGURE 12-16

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.

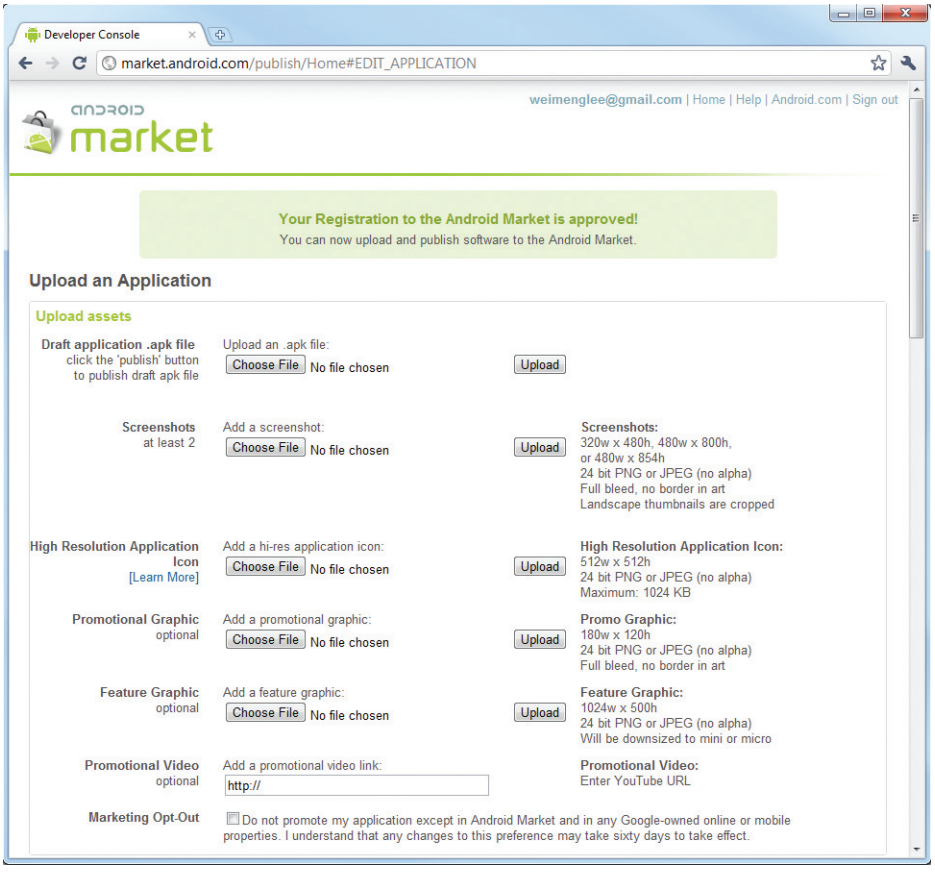

FIGURE 12-17

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.

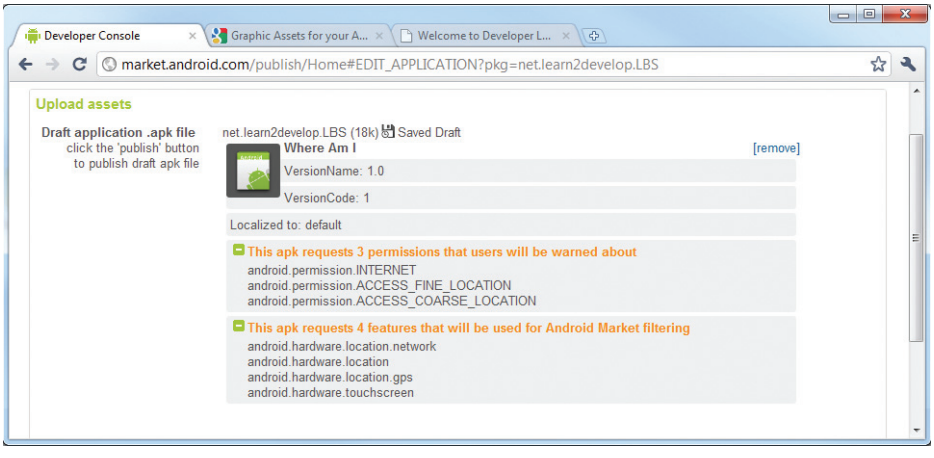

FIGURE 12-18

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.

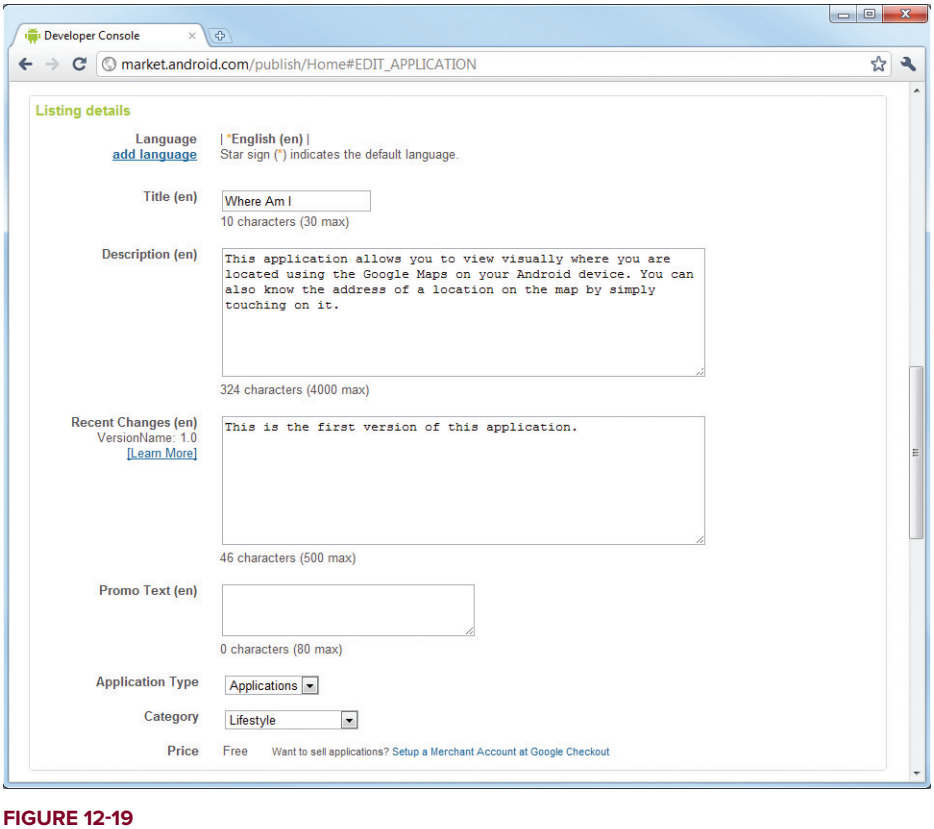

FIGURE 12-19

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.

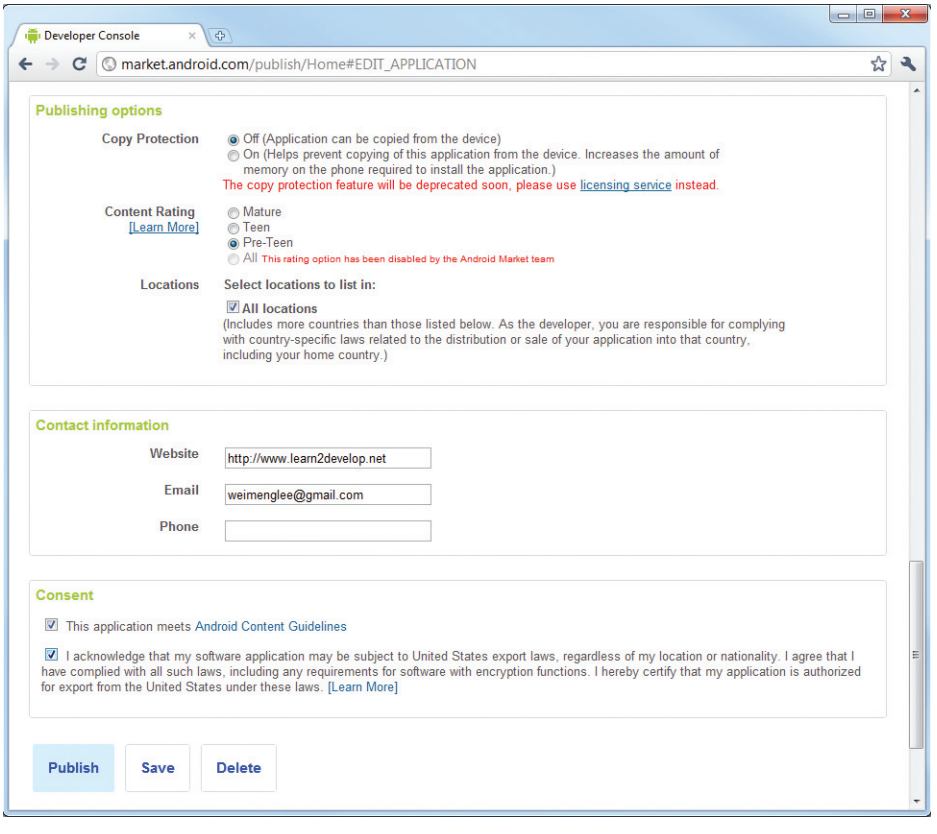

FIGURE 12-20

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!

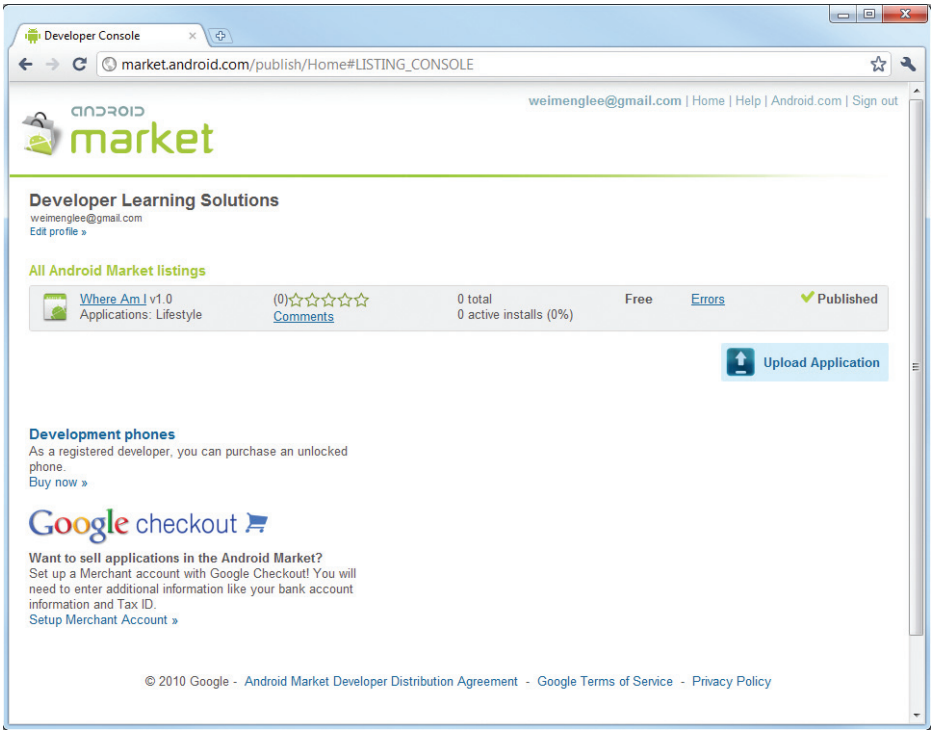

FIGURE 12-21

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.