6

Data Persistence

WHAT YOU WILL LEARN IN THIS CHAPTER

- How to save simple data using the SharedPreferences object
- Enabling users to modify preferences using a PreferenceActivity class
- How to write and read files in internal and external storage
- Creating and using a SQLite database

In this chapter, you will learn how to persist data in your Android applications. Persisting data is an important topic in application development, as users typically expect to reuse data in the future. For Android, there are primarily three basic ways of persisting data:

- A lightweight mechanism known as *shared preferences* to save small chunks of data
- Traditional file systems
- A relational database management system through the support of SQLite databases

The techniques discussed in this chapter enable applications to create and access their own private data. In the next chapter you'll learn how you can share data across applications.

SAVING AND LOADING USER PREFERENCES

Android provides the `SharedPreferences` object to help you save simple application data. For example, your application may have an option that enables users to specify the font size of the text displayed in your application. In this case, your application needs to remember the size set by the user so that the next time he or she uses the application again, it can set the size appropriately. In order to do so, you have several options. You can save the data to a file, but you have to perform some file management routines, such as writing the data to

the file, indicating how many characters to read from it, and so on. Also, if you have several pieces of information to save, such as text size, font name, preferred background color, and so on, then the task of writing to a file becomes more onerous.

An alternative to writing to a text file is to use a database, but saving simple data to a database is overkill, both from a developer's point of view and in terms of the application's run-time performance.

Using the `SharedPreferences` object, however, you save the data you want through the use of name/value pairs — specify a name for the data you want to save, and then both it and its value will be saved automatically to an XML file for you.

Accessing Preferences Using an Activity

In the following Try It Out, you learn how to use the `SharedPreferences` object to store application data. You will also learn how the stored application data can be modified directly by the user through a special type of activity provided by the Android OS.

TRY IT OUT Saving Data Using the SharedPreferences Object

codefile SharedPreferences.zip available for download at Wrox.com

1. Using Eclipse, create an Android project and name it **UsingPreferences**.
2. Create a new subfolder in the res folder and name it **xml**. In this newly created folder, add a file and name it **myapppreferences.xml** (see Figure 6-1).
3. Populate the `myapppreferences.xml` file as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen
    xmlns:android="http://schemas.android.com/apk/res/android">
    <PreferenceCategory android:title="Category 1">
        <CheckBoxPreference
            android:title="Checkbox"
            android:defaultValue="false"
            android:summary="True or False"
            android:key="checkboxPref" />
        </PreferenceCategory>
    <PreferenceCategory android:title="Category 2">
        <EditTextPreference
            android:summary="Enter a string"
            android:defaultValue="[Enter a string here]"
            android:title="Edit Text"
            android:key="editTextPref" />
        <RingtonePreference
            android:summary="Select a ringtone"
            android:title="Ringtones"
            android:key="ringtonePref" />
        <PreferenceScreen
            android:title="Second Preference Screen"
```

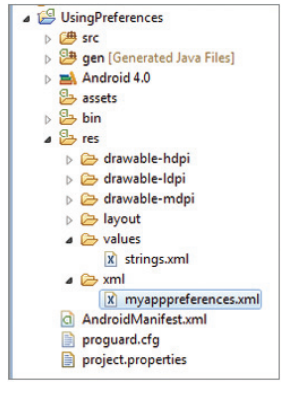

FIGURE 6-1

```
            android:summary=
                "Click here to go to the second Preference Screen"
            android:key="secondPrefScreenPref" >
            <EditTextPreference
                android:summary="Enter a string"
                android:title="Edit Text (second Screen)"
                android:key="secondEditTextPref" />
        </PreferenceScreen>
    </PreferenceCategory>
</PreferenceScreen>
```

4. Under the package name, add a new Class file and name it **AppPreferenceActivity**.

5. Populate the `AppPreferenceActivity.java` file as follows:

```
package net.learn2develop.UsingPreferences;

import android.os.Bundle;
import android.preference.PreferenceActivity;

public class AppPreferenceActivity extends PreferenceActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //---load the preferences from an XML file---
        addPreferencesFromResource(R.xml.myapppreferences);
    }
}
```

6. In the `AndroidManifest.xml` file, add the new entry for the `AppPreferenceActivity` class:

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

    <uses-sdk android:minSdkVersion="14" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:label="@string/app_name"
            android:name=".UsingPreferencesActivity" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />

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

```
                <action
                    android:name="net.learn2develop.AppPreferenceActivity" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

7. In the `main.xml` file, add the following code in bold (replacing the existing `TextView`):

```
<?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" >

<Button
    android:id="@+id/btnPreferences"
    android:text="Load Preferences Screen"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:onClick="onClickLoad"/>

<Button
    android:id="@+id/btnDisplayValues"
    android:text="Display Preferences Values"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:onClick="onClickDisplay"/>

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

<Button
    android:id="@+id/btnModifyValues"
    android:text="Modify Preferences Values"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:onClick="onClickModify"/>

</LinearLayout>
```

8. Add the following lines in bold to the `UsingPreferencesActivity.java` file:

```
package net.learn2develop.UsingPreferences;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class UsingPreferencesActivity extends Activity {
```

```
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void onClickLoad(View view) {
        Intent i = new Intent("net.learn2develop.AppPreferenceActivity");
        startActivity(i);
    }
}
```

9. Press F11 to debug the application on the Android emulator. Click the Load Preferences Screen button to see the preferences screen, as shown in Figure 6-2.

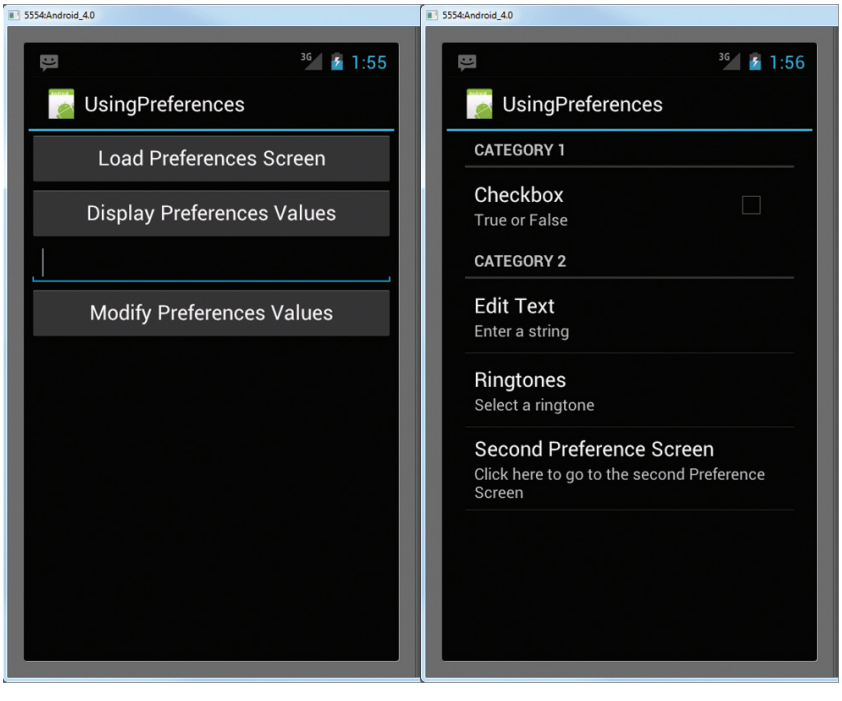

FIGURE 6-2

10. Clicking the Checkbox item toggles the checkbox's value between checked and unchecked. Note the two categories: Category 1 and Category 2. Click the Edit Text item and enter some values as shown in Figure 6-3. Click OK to dismiss the dialog.

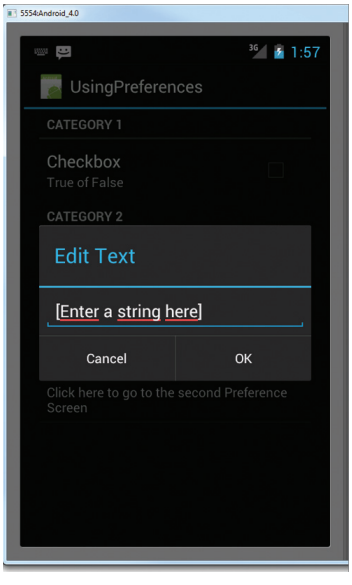

FIGURE 6-3

11. Click the Ringtones item to select either the default ringtone or silent mode (see Figure 6-4). If you test the application on a real Android device, you can select from a more comprehensive list of ringtones.

12. Clicking the Second Preference Screen item will navigate to the next screen (see Figure 6-5).

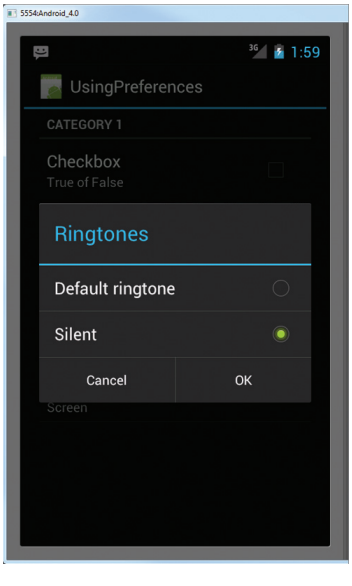

FIGURE 6-4

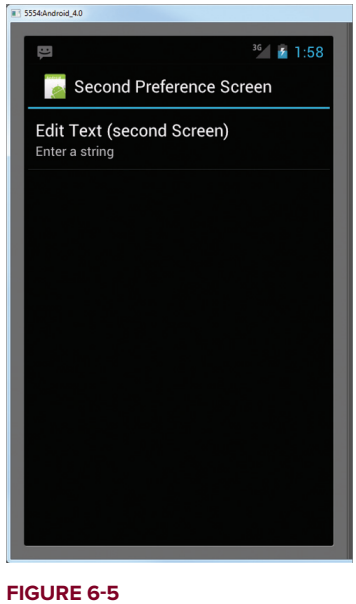

FIGURE 6-5

13. To go back to the previous screen, click the Back button. To dismiss the preferences screen, you also click the Back button.

14. Once you have modified the value of at least one of the preferences, a file is created in the `/data/data/net.learn2develop.UsingPreferences/shared_prefs` folder of the Android emulator. To verify this, switch to the DDMS perspective in Eclipse and look at the File Explorer tab (see Figure 6-6); you will see an XML file named `net.learn2develop.UsingPreferences_preferences.xml`.

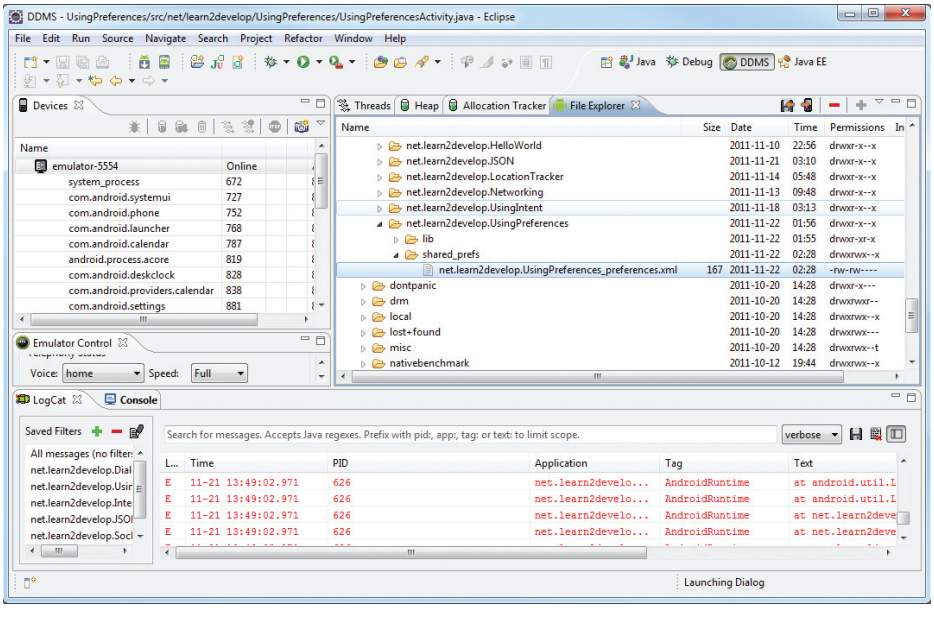

FIGURE 6-6

15. If you extract this file and examine its content, you will see something like the following:

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
<string name="editTextPref">[Enter a string here]</string>
<string name="ringtonePref"></string>
</map>
```

How It Works

You first created an XML file named `myapppreferences.xml` to store the types of preferences you want to save for your application:

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen
    xmlns:android="http://schemas.android.com/apk/res/android">
    <PreferenceCategory android:title="Category 1">
```

```
        <CheckBoxPreference
            android:title="Checkbox"
            android:defaultValue="false"
            android:summary="True or False"
            android:key="checkboxPref" />
        </PreferenceCategory>
    <PreferenceCategory android:title="Category 2">
        <EditTextPreference
            android:summary="Enter a string"
            android:defaultValue="[Enter a string here]"
            android:title="Edit Text"
            android:key="editTextPref" />
        <RingtonePreference
            android:summary="Select a ringtone"
            android:title="Ringtones"
            android:key="ringtonePref" />
        <PreferenceScreen
            android:title="Second Preference Screen"
            android:summary=
                "Click here to go to the second Preference Screen"
            android:key="secondPrefScreenPref" >
            <EditTextPreference
                android:summary="Enter a string"
                android:title="Edit Text (second Screen)"
                android:key="secondEditTextPref" />
        </PreferenceScreen>
    </PreferenceCategory>
</PreferenceScreen>
```

In the preceding snippet, you created the following:

- Two preference categories for grouping different types of preferences
- Two checkbox preferences with keys named `checkboxPref` and `secondEditTextPref`
- A ringtone preference with a key named `ringtonePref`
- A preference screen to contain additional preferences

The `android:key` attribute specifies the key that you can programmatically reference in your code to set or retrieve the value of that particular preference.

To get the OS to display all these preferences for users to edit, you create an activity that extends the `PreferenceActivity` base class, and then call the `addPreferencesFromResource()` method to load the XML file containing the preferences:

```
public class AppPreferenceActivity extends PreferenceActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //---load the preferences from an XML file---
        addPreferencesFromResource(R.xml.myapppreferences);
    }
}
```

The `PreferenceActivity` class is a specialized type of activity that displays a hierarchy of preferences to the user.

To display the activity for the preferences, you invoke it using an `Intent` object:

```
Intent i = new Intent("net.learn2develop.AppPreferenceActivity");
startActivity(i);
```

All the changes made to the preferences are automatically persisted to an XML file in the `shared_prefs` folder of the application.

Programmatically Retrieving and Modifying the Preferences Values

In the previous section, you saw how the `PreferenceActivity` class both enables developers to easily create preferences and enables users to modify them during runtime. To make use of these preferences in your application, you use the `SharedPreferences` class. The following Try It Out shows you how.

TRY IT OUT Retrieving and Modifying Preferences

1. Using the same project created in the previous section, add the following lines in bold to the `UsingPreferencesActivity.java` file:

```
package net.learn2develop.UsingPreferences;

import android.app.Activity;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

public class UsingPreferencesActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void onClickLoad(View view) {
        Intent i = new Intent("net.learn2develop.AppPreferenceActivity");
        startActivity(i);
    }

    public void onClickDisplay(View view) {
```

```
        SharedPreferences appPrefs =
                getSharedPreferences("net.learn2develop.UsingPreferences_preferences",
                        MODE_PRIVATE);
        DisplayText(appPrefs.getString("editTextPref", ""));
    }

    public void onClickModify(View view) {
        SharedPreferences appPrefs =
                getSharedPreferences("net.learn2develop.UsingPreferences_preferences",
                        MODE_PRIVATE);
        SharedPreferences.Editor prefsEditor = appPrefs.edit();
        prefsEditor.putString("editTextPref",
                ((EditText) findViewById(R.id.txtString)).getText().toString());
        prefsEditor.commit();
    }

    private void DisplayText(String str) {
        Toast.makeText(getBaseContext(), str, Toast.LENGTH_LONG).show();
    }
}
```

2. Press F11 to rerun the application on the Android emulator again. This time, clicking the Display Preferences Values button will display the value shown in Figure 6-7.

3. Enter a string in the EditText view and click the Modify Preferences Values button (see Figure 6-8).

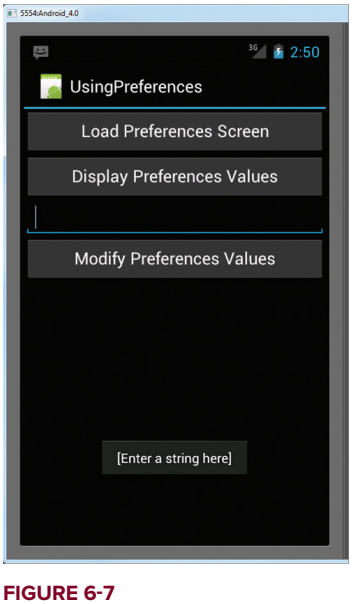

FIGURE 6-7

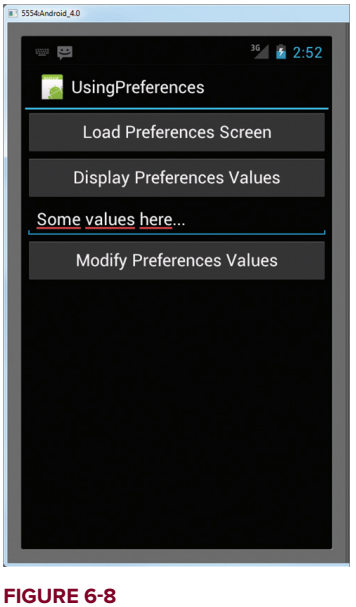

FIGURE 6-8

4. Now click the Display Preferences Values button again. Note that the new value is saved.

How It Works

In the `onClickDisplay()` method, you first used the `getSharedPreferences()` method to obtain an instance of the `SharedPreferences` class. You do so by specifying the name of the XML file (in this case it is "`net.learn2develop.UsingPreferences_preferences`," using the format: *`<PackageName>_preferences`*). To retrieve a string preference, you used the `getString()` method, passing it the key to the preference that you want to retrieve:

```
public void onClickDisplay(View view) {
    SharedPreferences appPrefs =
            getSharedPreferences("net.learn2develop.UsingPreferences_preferences",
                    MODE_PRIVATE);
    DisplayText(appPrefs.getString("editTextPref", ""));
}
```

The `MODE_PRIVATE` constant indicates that the preference file can only be opened by the application that created it.

In the `onClickModify()` method, you created a `SharedPreferences.Editor` object through the `edit()` method of the `SharedPreferences` object. To change the value of a string preference, use the `putString()` method. To save the changes to the preferences file, use the `commit()` method:

```
public void onClickModify(View view) {
    SharedPreferences appPrefs =
            getSharedPreferences("net.learn2develop.UsingPreferences_preferences",
                    MODE_PRIVATE);
    SharedPreferences.Editor prefsEditor = appPrefs.edit();
    prefsEditor.putString("editTextPref",
            ((EditText) findViewById(R.id.txtString)).getText().toString());
    prefsEditor.commit();
}
```

Changing the Default Name of the Preferences File

Notice that by default the name of the preferences file saved on the device is `net.learn2develop.UsingPreferences_preferences.xml`, with the package name used as the prefix. However, sometimes it is useful to give the preferences file a specific name. In this case, you can do the following.

Add the following code in bold to the `AppPreferenceActivity.java` file:

```
package net.learn2develop.UsingPreferences;

import android.os.Bundle;
import android.preference.PreferenceActivity;
```

```
import android.preference.PreferenceManager;

public class AppPreferenceActivity extends PreferenceActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        PreferenceManager prefMgr = getPreferenceManager();
        prefMgr.setSharedPreferencesName("appPreferences");

        //---load the preferences from an XML file---
        addPreferencesFromResource(R.xml.myapppreferences);
    }
}
```

Here, you make use of the `PreferenceManager` class to set the shared preferences file name to `appPreferences.xml`.

Modify the `UsingPreferencesActivity.java` file as follows:

```
    public void onClickDisplay(View view) {
        /*
        SharedPreferences appPrefs =
                getSharedPreferences("net.learn2develop.UsingPreferences_preferences",
                        MODE_PRIVATE);
        */
        SharedPreferences appPrefs =
                getSharedPreferences("appPreferences", MODE_PRIVATE);

        DisplayText(appPrefs.getString("editTextPref", ""));
    }

    public void onClickModify(View view) {
        /*
        SharedPreferences appPrefs =
                getSharedPreferences("net.learn2develop.UsingPreferences_preferences",
                        MODE_PRIVATE);
        */
        SharedPreferences appPrefs =
                getSharedPreferences("appPreferences", MODE_PRIVATE);

        SharedPreferences.Editor prefsEditor = appPrefs.edit();
        prefsEditor.putString("editTextPref",
                ((EditText) findViewById(R.id.txtString)).getText().toString());
        prefsEditor.commit();
    }
```

When you rerun the application and make changes to the preferences, you will notice that the `appPreferences.xml` file is now created (see Figure 6-9).

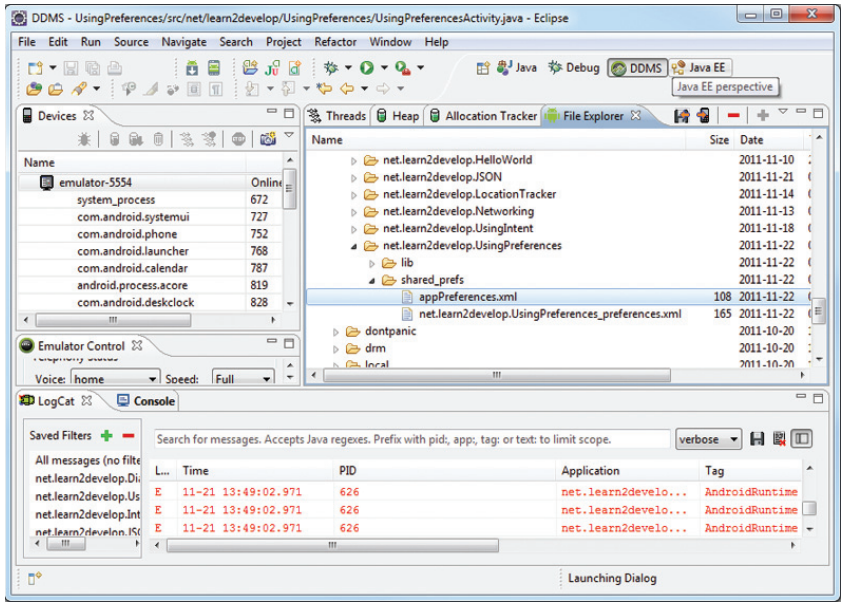

FIGURE 6-9

PERSISTING DATA TO FILES

The `SharedPreferences` object enables you to store data that is best stored as name/value pairs — for example, user ID, birth date, gender, driving license number, and so on. However, sometimes you might prefer to use the traditional file system to store your data. For example, you might want to store the text of poems you want to display in your applications. In Android, you can use the classes in the `java.io` package to do so.

Saving to Internal Storage

The first way to save files in your Android application is to write to the device's internal storage. The following Try It Out demonstrates how to save a string entered by the user to the device's internal storage.

TRY IT OUT Saving Data to Internal Storage

codefile Files.zip available for download at Wrox.com

1. Using Eclipse, create an Android project and name it **Files**.
2. In the `main.xml` file, add the following statements in bold:

```
<?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" >

<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Please enter some text" />

<EditText
    android:id="@+id/txtText1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />

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

<Button
    android:id="@+id/btnLoad"
    android:text="Load"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:onClick="onClickLoad" />

</LinearLayout>
```

3. In the `FilesActivity.java` file, add the following statements in bold:

```
package net.learn2develop.Files;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

public class FilesActivity extends Activity {
    EditText textBox;
    static final int READ_BLOCK_SIZE = 100;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        textBox = (EditText) findViewById(R.id.txtText1);
    }

    public void onClickSave(View view) {
```

```
        String str = textBox.getText().toString();
        try
        {
            FileOutputStream fOut =
                    openFileOutput("textfile.txt",
                            MODE_WORLD_READABLE);
            OutputStreamWriter osw = new
                    OutputStreamWriter(fOut);

            //---write the string to the file---
            osw.write(str);
            osw.flush();
            osw.close();

            //---display file saved message---
            Toast.makeText(getBaseContext(),
                    "File saved successfully!",
                    Toast.LENGTH_SHORT).show();

            //---clears the EditText---
            textBox.setText("");
        }
        catch (IOException ioe)
        {
            ioe.printStackTrace();
        }

    }

    public void onClickLoad(View view) {
        try
        {
            FileInputStream fIn =
                    openFileInput("textfile.txt");
            InputStreamReader isr = new
                    InputStreamReader(fIn);

            char[] inputBuffer = new char[READ_BLOCK_SIZE];
            String s = "";

            int charRead;
            while ((charRead = isr.read(inputBuffer))>0)
            {
                //---convert the chars to a String---
                String readString =
                        String.copyValueOf(inputBuffer, 0,
                                charRead);
                s += readString;

                inputBuffer = new char[READ_BLOCK_SIZE];
            }
            //---set the EditText to the text that has been
            // read---
            textBox.setText(s);

            Toast.makeText(getBaseContext(),
                    "File loaded successfully!",
                    Toast.LENGTH_SHORT).show();
```

```
            }
            catch (IOException ioe) {
                ioe.printStackTrace();
            }

        }

}
```

4. Press F11 to debug the application on the Android emulator.
5. Type some text into the EditText view (see Figure 6-10) and then click the Save button.
6. If the file is saved successfully, you will see the `Toast` class displaying the "File saved successfully!" message. The text in the `EditText` view should disappear.
7. Click the Load button and you should see the string appearing in the `EditText` view again. This confirms that the text is saved correctly.

FIGURE 6-10

How It Works

To save text into a file, you use the `FileOutputStream` class. The `openFileOutput()` method opens a named file for writing, with the mode specified. In this example, you used the `MODE_WORLD_READABLE` constant to indicate that the file is readable by all other applications:

```
FileOutputStream fOut =
        openFileOutput("textfile.txt",
                MODE_WORLD_READABLE);
```

Apart from the `MODE_WORLD_READABLE` constant, you can select from the following: `MODE_PRIVATE` (the file can only be accessed by the application that created it), `MODE_APPEND` (for appending to an existing file), and `MODE_WORLD_WRITEABLE` (all other applications have write access to the file).

To convert a character stream into a byte stream, you use an instance of the `OutputStreamWriter` class, by passing it an instance of the `FileOutputStream` object:

```
OutputStreamWriter osw = new
        OutputStreamWriter(fOut);
```

You then use its `write()` method to write the string to the file. To ensure that all the bytes are written to the file, use the `flush()` method. Finally, use the `close()` method to close the file:

```
//---write the string to the file---
osw.write(str);
osw.flush();
osw.close();
```

To read the content of a file, you use the `FileInputStream` class, together with the `InputStreamReader` class:

```
FileInputStream fIn =
        openFileInput("textfile.txt");
```

```
InputStreamReader isr = new
        InputStreamReader(fIn);
```

Because you do not know the size of the file to read, the content is read in blocks of 100 characters into a buffer (character array). The characters read are then copied into a `String` object:

```
char[] inputBuffer = new char[READ_BLOCK_SIZE];
String s = "";

int charRead;
while ((charRead = isr.read(inputBuffer))>0)
{
    //---convert the chars to a String---
    String readString =
            String.copyValueOf(inputBuffer, 0,
                    charRead);
    s += readString;

    inputBuffer = new char[READ_BLOCK_SIZE];
}
```

The `read()` method of the `InputStreamReader` object checks the number of characters read and returns -1 if the end of the file is reached.

When testing this application on the Android emulator, you can use the DDMS perspective to verify that the application did indeed save the file into the application's files directory (see Figure 6-11; the entire path is `/data/data/net.learn2develop.Files/files`)

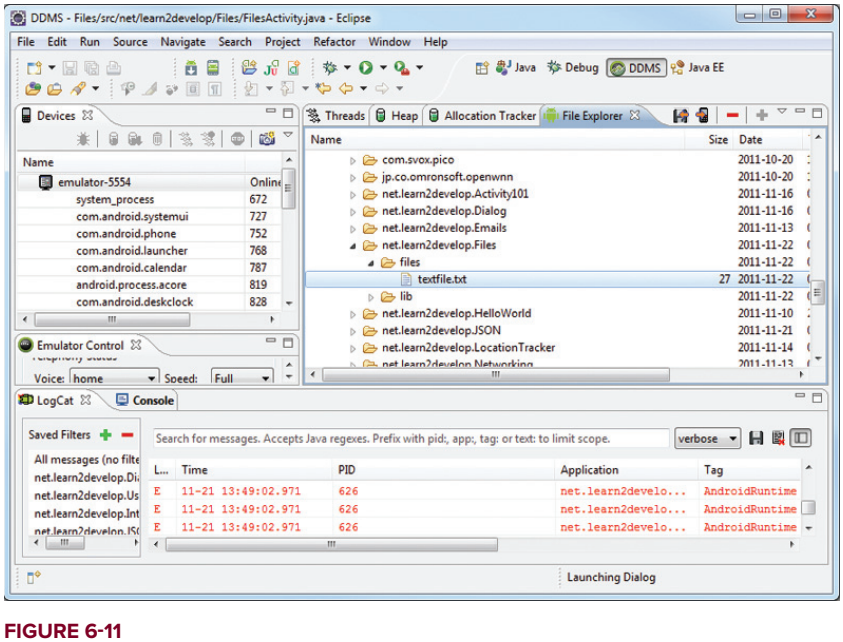

FIGURE 6-11

Saving to External Storage (SD Card)

The previous section showed how you can save your files to the internal storage of your Android device. Sometimes, it would be useful to save them to external storage (such as an SD card) because of its larger capacity, as well as the capability to share the files easily with other users (by removing the SD card and passing it to somebody else).

Using the project created in the previous section as the example, to save the text entered by the user in the SD card, modify the `onClick()` method of the Save button as shown in bold here:

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;

import android.app.Activity;
import android.os.Bundle;
import android.os.Environment;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

    public void onClickSave(View view) {
        String str = textBox.getText().toString();
        try
        {
            //---SD Card Storage---
            File sdCard = Environment.getExternalStorageDirectory();
            File directory = new File (sdCard.getAbsolutePath() +
                "/MyFiles");
            directory.mkdirs();
            File file = new File(directory, "textfile.txt");
            FileOutputStream fOut = new FileOutputStream(file);

            /*
            FileOutputStream fOut =
                    openFileOutput("textfile.txt",
                            MODE_WORLD_READABLE);
            */

            OutputStreamWriter osw = new
                    OutputStreamWriter(fOut);

            //---write the string to the file---
            osw.write(str);
            osw.flush();
            osw.close();

            //---display file saved message---
```

```
            Toast.makeText(getBaseContext(),
                    "File saved successfully!",
                    Toast.LENGTH_SHORT).show();

            //---clears the EditText---
            textBox.setText("");
        }
        catch (IOException ioe)
        {
            ioe.printStackTrace();
        }

    }
```

The preceding code uses the `getExternalStorageDirectory()` method to return the full path to the external storage. Typically, it should return the "`/sdcard`" path for a real device, and "`/mnt/sdcard`" for an Android emulator. However, you should never try to hardcode the path to the SD card, as manufacturers may choose to assign a different path name to the SD card. Hence, be sure to use the `getExternalStorageDirectory()` method to return the full path to the SD card.

You then create a directory called `MyFiles` in the SD card. Finally, you save the file into this directory.

To load the file from the external storage, modify the `onClickLoad()` method for the Load button:

```
public void onClickLoad(View view) {
    try
    {
        //---SD Storage---
        File sdCard = Environment.getExternalStorageDirectory();
        File directory = new File (sdCard.getAbsolutePath() +
            "/MyFiles");
        File file = new File(directory, "textfile.txt");
        FileInputStream fIn = new FileInputStream(file);
        InputStreamReader isr = new InputStreamReader(fIn);

        /*
        FileInputStream fIn =
                openFileInput("textfile.txt");
        InputStreamReader isr = new
                InputStreamReader(fIn);
        */

        char[] inputBuffer = new char[READ_BLOCK_SIZE];
        String s = "";

        int charRead;
        while ((charRead = isr.read(inputBuffer))>0)
        {
            //---convert the chars to a String---
            String readString =
```

```
                        String.copyValueOf(inputBuffer, 0,
                                charRead);
                    s += readString;

                    inputBuffer = new char[READ_BLOCK_SIZE];
                }
                //---set the EditText to the text that has been
                // read---
                textBox.setText(s);

                Toast.makeText(getBaseContext(),
                        "File loaded successfully!",
                        Toast.LENGTH_SHORT).show();
            }
            catch (IOException ioe) {
                ioe.printStackTrace();
            }

        }
```

Note that in order to write to the external storage, you need to add the `WRITE_EXTERNAL_STORAGE` permission in your `AndroidManifest.xml` file:

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

    <uses-sdk android:minSdkVersion="14" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:label="@string/app_name"
            android:name=".FilesActivity" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

If you run the preceding modified code, you will see the text file created in the `/mnt/sdcard/MyFiles/` folder (see Figure 6-12).

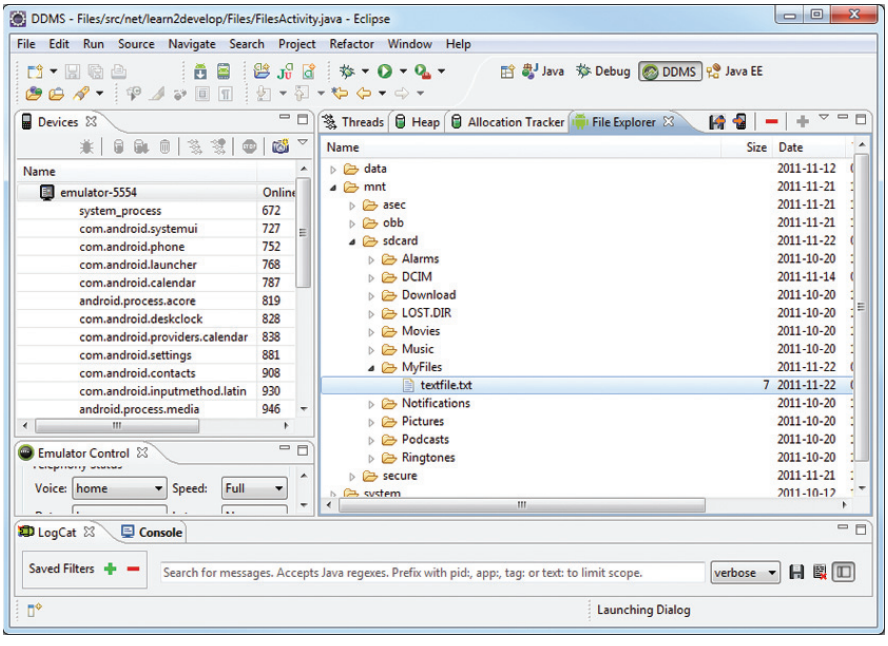

FIGURE 6-12

Choosing the Best Storage Option

The previous sections described three main ways to save data in your Android applications: the `SharedPreferences` object, internal storage, and external storage. Which one should you use in your applications? Here are some guidelines:

- If you have data that can be represented using name/value pairs, then use the `SharedPreferences` object. For example, if you want to store user preference data such as user name, background color, date of birth, or last login date, then the `SharedPreferences` object is the ideal way to store this data. Moreover, you don't really have to do much to store data this way; just use the `SharedPreferences` object to store and retrieve it.
- If you need to store ad-hoc data, then using the internal storage is a good option. For example, your application (such as an RSS reader) may need to download images from the web for display. In this scenario, saving the images to internal storage is a good solution. You may also need to persist data created by the user, such as when you have a note-taking application that enables users to take notes and save them for later use. In both of these scenarios, using the internal storage is a good choice.
- There are times when you need to share your application data with other users. For example, you may create an Android application that logs the coordinates of the locations that a user has been to, and you want to share all this data with other users. In this scenario, you can

store your files on the SD card of the device so that users can easily transfer the data to other devices (and computers) for use later.

Using Static Resources

Besides creating and using files dynamically during runtime, it is also possible to add files to your package during design time so that you can use it during runtime. For example, you may want to bundle some help files with your package so that you can display some help messages when users need them. In this case, you can add the files to your package's `res/raw` folder (you need to create this folder yourself). Figure 6-13 shows the `res/raw` folder containing a file named `textfile.txt`.

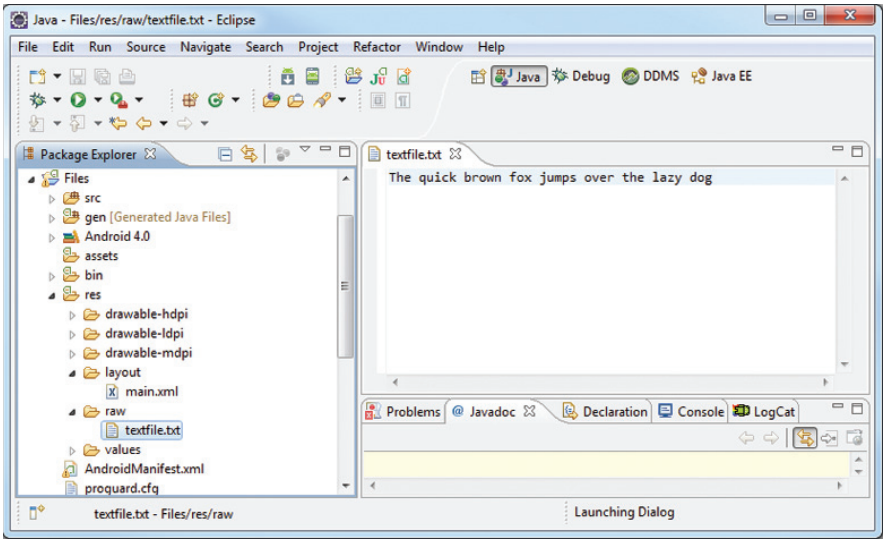

FIGURE 6-13

To make use of the file in code, use the `getResources()` method (of the `Activity` class) to return a `Resources` object, and then use its `openRawResource()` method to open the file contained in the `res/raw` folder:

```
import java.io.BufferedReader;
import java.io.InputStream;

public class FilesActivity extends Activity {
    EditText textBox;
    static final int READ_BLOCK_SIZE = 100;

    /** Called when the activity is first created. */
    @Override
```

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    textBox = (EditText) findViewById(R.id.txtText1);

    InputStream is = this.getResources().openRawResource(R.raw.textfile);
    BufferedReader br = new BufferedReader(new InputStreamReader(is));
    String str = null;
    try {
        while ((str = br.readLine()) != null) {
            Toast.makeText(getBaseContext(),
                str, Toast.LENGTH_SHORT).show();
        }
        is.close();
        br.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

The resource ID of the resource stored in the `res/raw` folder is named after its filename without its extension. For example, if the text file is `textfile.txt`, then its resource ID is `R.raw.textfile`.

CREATING AND USING DATABASES

So far, all the techniques you have seen are useful for saving simple sets of data. For saving relational data, using a database is much more efficient. For example, if you want to store the test results of all the students in a school, it is much more efficient to use a database to represent them because you can use database querying to retrieve the results of specific students. Moreover, using databases enables you to enforce data integrity by specifying the relationships between different sets of data.

Android uses the SQLite database system. The database that you create for an application is only accessible to itself; other applications will not be able to access it.

In this section, you will learn how to programmatically create a SQLite database in your Android application. For Android, the SQLite database that you create programmatically in an application is always stored in the `/data/data/<package_name>/databases` folder.

Creating the DBAdapter Helper Class

A good practice for dealing with databases is to create a helper class to encapsulate all the complexities of accessing the data so that it is transparent to the calling code. Hence, for this section, you will create a helper class called `DBAdapter` that creates, opens, closes, and uses a SQLite database.

In this example, you are going to create a database named `MyDB` containing one table named `contacts`. This table will have three columns: `_id`, `name`, and `email` (see Figure 6-14).

_id	name	email

FIGURE 6-14

TRY IT OUT Creating the Database Helper Class

codefile Databases.zip available for download at Wrox.com

1. Using Eclipse, create an Android project and name it **Databases**.
2. Add a new Java Class file to the package and name it **DBAdapter** (see Figure 6-15).
3. Add the following statements in bold to the `DBAdapter.java` file:

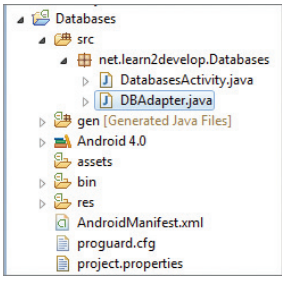

FIGURE 6-15

```
package net.learn2develop.Databases;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.util.Log;

public class DBAdapter {
    static final String KEY_ROWID = "_id";
    static final String KEY_NAME = "name";
    static final String KEY_EMAIL = "email";
    static final String TAG = "DBAdapter";

    static final String DATABASE_NAME = "MyDB";
    static final String DATABASE_TABLE = "contacts";
    static final int DATABASE_VERSION = 1;

    static final String DATABASE_CREATE =
        "create table contacts (_id integer primary key autoincrement, "
        + "name text not null, email text not null);";

    final Context context;

    DatabaseHelper DBHelper;
    SQLiteDatabase db;

    public DBAdapter(Context ctx)
    {
        this.context = ctx;
        DBHelper = new DatabaseHelper(context);
    }

    private static class DatabaseHelper extends SQLiteOpenHelper
    {
        DatabaseHelper(Context context)
        {
            super(context, DATABASE_NAME, null, DATABASE_VERSION);
        }

        @Override
        public void onCreate(SQLiteDatabase db)
```

```
    {
        try {
            db.execSQL(DATABASE_CREATE);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)
    {
        Log.w(TAG, "Upgrading database from version " + oldVersion + " to "
                + newVersion + ", which will destroy all old data");
        db.execSQL("DROP TABLE IF EXISTS contacts");
        onCreate(db);
    }
}

//---opens the database---
public DBAdapter open() throws SQLException
{
    db = DBHelper.getWritableDatabase();
    return this;
}

//---closes the database---
public void close()
{
    DBHelper.close();
}

//---insert a contact into the database---
public long insertContact(String name, String email)
{
    ContentValues initialValues = new ContentValues();
    initialValues.put(KEY_NAME, name);
    initialValues.put(KEY_EMAIL, email);
    return db.insert(DATABASE_TABLE, null, initialValues);
}

//---deletes a particular contact---
public boolean deleteContact(long rowId)
{
    return db.delete(DATABASE_TABLE, KEY_ROWID + "=" + rowId, null) > 0;
}

//---retrieves all the contacts---
public Cursor getAllContacts()
{
    return db.query(DATABASE_TABLE, new String[] {KEY_ROWID, KEY_NAME,
            KEY_EMAIL}, null, null, null, null, null);
}

//---retrieves a particular contact---
public Cursor getContact(long rowId) throws SQLException
{
```

```
            Cursor mCursor =
                    db.query(true, DATABASE_TABLE, new String[] {KEY_ROWID,
                    KEY_NAME, KEY_EMAIL}, KEY_ROWID + "=" + rowId, null,
                    null, null, null, null);
            if (mCursor != null) {
                mCursor.moveToFirst();
            }
            return mCursor;
        }

        //---updates a contact---
        public boolean updateContact(long rowId, String name, String email)
        {
            ContentValues args = new ContentValues();
            args.put(KEY_NAME, name);
            args.put(KEY_EMAIL, email);
            return db.update(DATABASE_TABLE, args, KEY_ROWID + "=" + rowId, null) > 0;
        }
    }
```

How It Works

You first defined several constants to contain the various fields for the table that you are going to create in your database:

```
    static final String KEY_ROWID = "_id";
    static final String KEY_NAME = "name";
    static final String KEY_EMAIL = "email";
    static final String TAG = "DBAdapter";

    static final String DATABASE_NAME = "MyDB";
    static final String DATABASE_TABLE = "contacts";
    static final int DATABASE_VERSION = 1;

    static final String DATABASE_CREATE =
        "create table contacts (_id integer primary key autoincrement, "
        + "name text not null, email text not null);";
```

In particular, the `DATABASE_CREATE` constant contains the SQL statement for creating the `contacts` table within the `MyDB` database.

Within the `DBAdapter` class, you also added a private class that extended the `SQLiteOpenHelper` class, which is a helper class in Android to manage database creation and version management. In particular, you overrode the `onCreate()` and `onUpgrade()` methods:

```
    private static class DatabaseHelper extends SQLiteOpenHelper
    {
        DatabaseHelper(Context context)
        {
            super(context, DATABASE_NAME, null, DATABASE_VERSION);
        }

        @Override
        public void onCreate(SQLiteDatabase db)
```

```
        {
            try {
                db.execSQL(DATABASE_CREATE);
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        @Override
        public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)
        {
            Log.w(TAG, "Upgrading database from version " + oldVersion + " to "
                    + newVersion + ", which will destroy all old data");
            db.execSQL("DROP TABLE IF EXISTS contacts");
            onCreate(db);
        }
    }
```

The `onCreate()` method creates a new database if the required database is not present. The `onUpgrade()` method is called when the database needs to be upgraded. This is achieved by checking the value defined in the `DATABASE_VERSION` constant. For this implementation of the `onUpgrade()` method, you simply drop the table and create it again.

You can then define the various methods for opening and closing the database, as well as the methods for adding/editing/deleting rows in the table:

```
//---opens the database---
public DBAdapter open() throws SQLException
{
    db = DBHelper.getWritableDatabase();
    return this;
}

//---closes the database---
public void close()
{
    DBHelper.close();
}

//---insert a contact into the database---
public long insertContact(String name, String email)
{
    ContentValues initialValues = new ContentValues();
    initialValues.put(KEY_NAME, name);
    initialValues.put(KEY_EMAIL, email);
    return db.insert(DATABASE_TABLE, null, initialValues);
}

//---deletes a particular contact---
public boolean deleteContact(long rowId)
{
    return db.delete(DATABASE_TABLE, KEY_ROWID + "=" + rowId, null) > 0;
}

//---retrieves all the contacts---
```

```
public Cursor getAllContacts()
{
    return db.query(DATABASE_TABLE, new String[] {KEY_ROWID, KEY_NAME,
            KEY_EMAIL}, null, null, null, null, null);
}

//---retrieves a particular contact---
public Cursor getContact(long rowId) throws SQLException
{
    Cursor mCursor =
            db.query(true, DATABASE_TABLE, new String[] {KEY_ROWID,
            KEY_NAME, KEY_EMAIL}, KEY_ROWID + "=" + rowId, null,
            null, null, null, null);
    if (mCursor != null) {
        mCursor.moveToFirst();
    }
    return mCursor;
}

//---updates a contact---
public boolean updateContact(long rowId, String name, String email)
{
    ContentValues args = new ContentValues();
    args.put(KEY_NAME, name);
    args.put(KEY_EMAIL, email);
    return db.update(DATABASE_TABLE, args, KEY_ROWID + "=" + rowId, null) > 0;
}
```

Notice that Android uses the `Cursor` class as a return value for queries. Think of the `Cursor` as a pointer to the result set from a database query. Using `Cursor` enables Android to more efficiently manage rows and columns as needed.

You use a `ContentValues` object to store name/value pairs. Its `put()` method enables you to insert keys with values of different data types.

To create a database in your application using the `DBAdapter` class, you create an instance of the `DBAdapter` class:

```
public DBAdapter(Context ctx)
{
    this.context = ctx;
    DBHelper = new DatabaseHelper(context);
}
```

The constructor of the `DBAdapter` class will then create an instance of the `DatabaseHelper` class to create a new database:

```
    DatabaseHelper(Context context)
    {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }
```

Using the Database Programmatically

With the `DBAdapter` helper class created, you are now ready to use the database. In the following sections, you will learn how to perform the regular CRUD (create, read, update and delete) operations commonly associated with databases.

Adding Contacts

The following Try It Out demonstrates how you can add a contact to the table.

TRY IT OUT **Adding Contacts to a Table**

codefile Databases.zip available for download at Wrox.com

1. Using the same project created earlier, add the following statements in bold to the `DatabasesActivity.java` file:

```
package net.learn2develop.Databases;

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

public class DatabasesActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        DBAdapter db = new DBAdapter(this);

        //---add a contact---
        db.open();
        long id = db.insertContact("Wei-Meng Lee", "weimenglee@learn2develop.net");
        id = db.insertContact("Mary Jackson", "mary@jackson.com");
        db.close();
    }
}
```

2. Press F11 to debug the application on the Android emulator.

How It Works

In this example, you first created an instance of the `DBAdapter` class:

```
DBAdapter db = new DBAdapter(this);
```

The `insertContact()` method returns the ID of the inserted row. If an error occurs during the operation, it returns -1.

If you examine the file system of the Android device/emulator using DDMS, you can see that the `MyDB` database is created under the `databases` folder (see Figure 6-16).

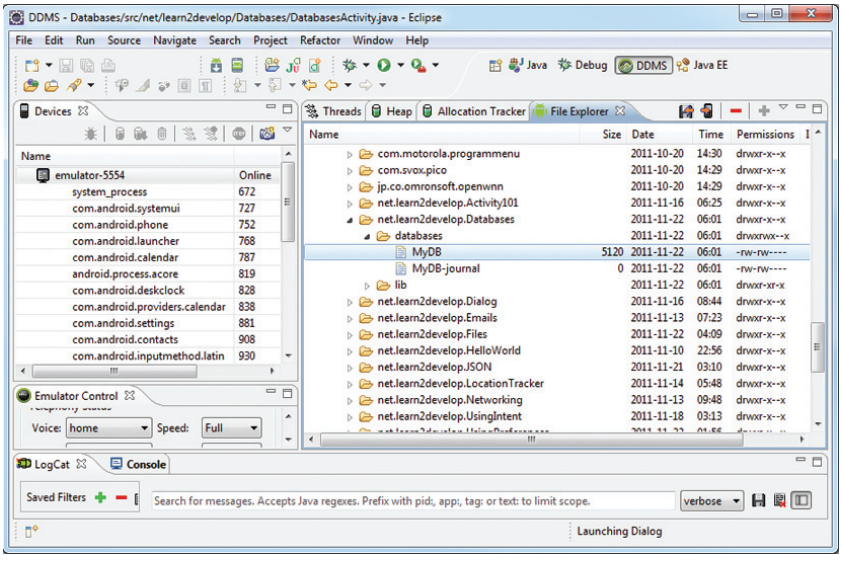

FIGURE 6-16

Retrieving All the Contacts

To retrieve all the contacts in the `contacts` table, use the `getAllContacts()` method of the `DBAdapter` class, as the following Try It Out shows.

TRY IT OUY Retrieving All Contacts from a Table

codefile Databases.zip available for download at Wrox.com

1. Using the same project created earlier, add the following statements in bold to the `DatabasesActivity.java` file:

```
package net.learn2develop.Databases;

import android.app.Activity;
import android.database.Cursor;
import android.os.Bundle;
import android.widget.Toast;

public class DatabasesActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        DBAdapter db = new DBAdapter(this);

        /*
        //---add a contact---
```

```
        db.open();
        ...
        db.close();
        */

        //---get all contacts---
        db.open();
        Cursor c = db.getAllContacts();
        if (c.moveToFirst())
        {
            do {
                DisplayContact(c);
            } while (c.moveToNext());
        }
        db.close();
    }

    public void DisplayContact(Cursor c)
    {
        Toast.makeText(this,
                "id: " + c.getString(0) + "\n" +
                "Name: " + c.getString(1) + "\n" +
                "Email:  " + c.getString(2),
                Toast.LENGTH_LONG).show();
    }
}
```

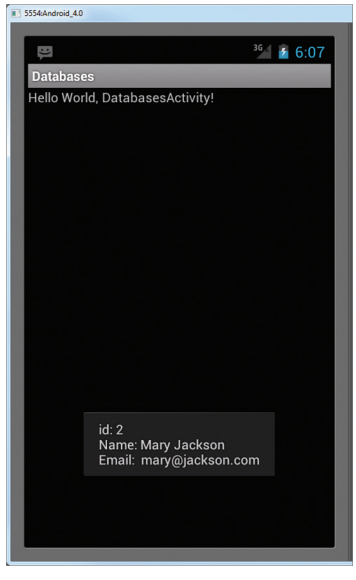

FIGURE 6-17

2. Press F11 to debug the application on the Android emulator. Figure 6-17 shows the `Toast` class displaying the contacts retrieved from the database.

How It Works

The `getAllContacts()` method of the `DBAdapter` class retrieves all the contacts stored in the database. The result is returned as a `Cursor` object. To display all the contacts, you first need to call the `moveToFirst()` method of the `Cursor` object. If it succeeds (which means at least one row is available), then you display the details of the contact using the `DisplayContact()` method. To move to the next contact, call the `moveToNext()` method of the `Cursor` object.

Retrieving a Single Contact

To retrieve a single contact using its ID, call the `getContact()` method of the `DBAdapter` class, as the following Try It Out shows.

TRY IT OUT Retrieving a Contact from a Table

codefile Databases.zip available for download at Wrox.com

1. Using the same project created earlier, add the following statements in bold to the `DatabasesActivity.java` file:

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

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

        DBAdapter db = new DBAdapter(this);

        /*
        //---add a contact---
        ...
        //--get all contacts---
        ...
        db.close();
        */

        //---get a contact---
        db.open();
        Cursor c = db.getContact(2);
        if (c.moveToFirst())
            DisplayContact(c);
        else
            Toast.makeText(this, "No contact found", Toast.LENGTH_LONG).show();
        db.close();
    }
```

2. Press F11 to debug the application on the Android emulator. The details of the second contact will be displayed using the `Toast` class.

How It Works

The `getContact()` method of the `DBAdapter` class retrieves a single contact using its ID. You passed in the ID of the contact; in this case, you passed in an ID of 2 to indicate that you want to retrieve the second contact:

```
Cursor c = db.getContact(2);
```

The result is returned as a `Cursor` object. If a row is returned, you display the details of the contact using the `DisplayContact()` method; otherwise, you display a message using the `Toast` class.

Updating a Contact

To update a particular contact, call the `updateContact()` method in the `DBAdapter` class by passing the ID of the contact you want to update, as the following Try It Out shows.

TRY IT OUT Updating a Contact in a Table

codefile Databases.zip available for download at Wrox.com

1. Using the same project created earlier, add the following statements in bold to the `DatabasesActivity.java` file:

```
@Override
public void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        DBAdapter db = new DBAdapter(this);

        /*
        //---add a contact---
        ...
        //--get all contacts---
        ...
        //---get a contact---
        ...
        db.close();
        */

        //---update contact---
        db.open();
        if (db.updateContact(1, "Wei-Meng Lee", "weimenglee@gmail.com"))
            Toast.makeText(this, "Update successful.", Toast.LENGTH_LONG).show();
        else
            Toast.makeText(this, "Update failed.", Toast.LENGTH_LONG).show();
        db.close();
    }
```

2. Press F11 to debug the application on the Android emulator. A message will be displayed if the update is successful.

How It Works

The `updateContact()` method in the `DBAdapter` class updates a contact's details by using the ID of the contact you want to update. It returns a Boolean value, indicating whether the update was successful.

Deleting a Contact

To delete a contact, use the `deleteContact()` method in the `DBAdapter` class by passing the ID of the contact you want to update, as the following Try It Out shows.

TRY IT OUT Deleting a Contact from a Table

codefile Databases.zip available for download at Wrox.com

1. Using the same project created earlier, add the following statements in bold to the `DatabasesActivity.java` file:

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

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

        DBAdapter db = new DBAdapter(this);

        /*
        //---add a contact---
        ...
        //--get all contacts---
        ...
        //---get a contact---
        ...
        //---update contact---
        ...
        db.close();
        */

        //---delete a contact---
        db.open();
        if (db.deleteContact(1))
            Toast.makeText(this, "Delete successful.", Toast.LENGTH_LONG).show();
        else
            Toast.makeText(this, "Delete failed.", Toast.LENGTH_LONG).show();
        db.close();
    }
```

2. Press F11 to debug the application on the Android emulator. A message is displayed if the deletion was successful.

How It Works

The `deleteContact()` method in the `DBAdapter` class deletes a contact using the ID of the contact you want to delete. It returns a Boolean value, indicating whether the deletion was successful.

Upgrading the Database

Sometimes, after creating and using the database, you may need to add additional tables, change the schema of the database, or add columns to your tables. In this case, you need to migrate your existing data from the old database to a newer one.

To upgrade the database, change the `DATABASE_VERSION` constant to a value higher than the previous one. For example, if its previous value was 1, change it to 2:

```
public class DBAdapter {
    static final String KEY_ROWID = "_id";
    static final String KEY_NAME = "name";
    static final String KEY_EMAIL = "email";
    static final String TAG = "DBAdapter";

    static final String DATABASE_NAME = "MyDB";
    static final String DATABASE_TABLE = "contacts";
    static final int DATABASE_VERSION = 2;
```

NOTE *Before you run this example, be sure to comment out the block of delete statements described in the previous section. If not, the deletion will fail as the table in the database will be dropped (deleted).*

When you run the application one more time, you will see the following message in the LogCat window of Eclipse:

```
DBAdapter(8705): Upgrading database from version 1 to 2, which
will destroy all old data
```

In this example, for simplicity you simply drop the existing table and create a new one. In real life, you usually back up your existing table and then copy it over to the new table.

Pre-Creating the Database

In real-life applications, sometimes it would be more efficient to pre-create the database at design time rather than runtime. For example, you want to create an application to log the coordinates of all the places that you have been to. In this case, it is much easier to pre-create the database during the design time and simply use it during runtime.

To pre-create a SQLite database, you can use many of the free tools available on the Internet. One such tool is the SQLite Database Browser, which is available free for different platforms (`http://sourceforge.net/projects/sqlitebrowser/`).

Once you have installed the SQLite Database Browser, you can create a database visually. Figure 6-18 shows that I have created a `contacts` table with the fields indicated.

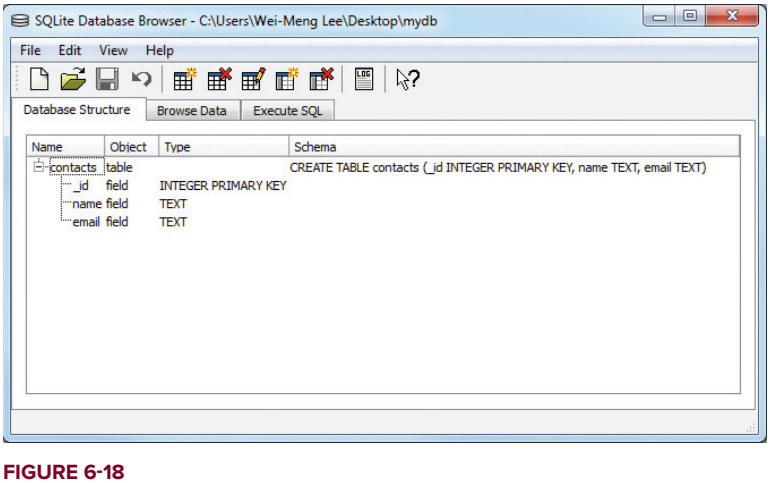

FIGURE 6-18

Populating the table with rows is also straightforward. Figure 6-19 shows how you can fill the table with data using the Browse Data tab.

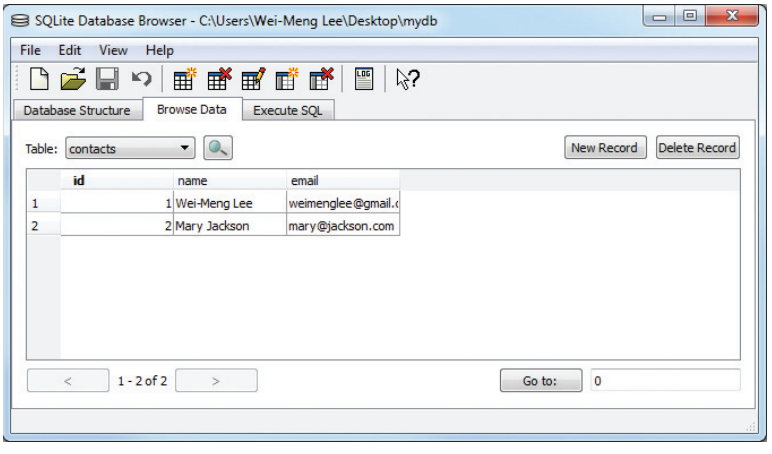

FIGURE 6-19

With the database created at design time, the next thing you should do is bundle it together with your application so that you can use it in your application. The following Try It Out shows you how.

TRY IT OUT Bundling a Database

1. Using the same project created earlier, drag and drop the SQLite database file that you have created in the previous section into the `assets` folder in your Android project in Eclipse (see Figure 6-20).

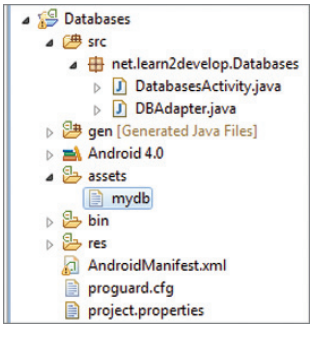

FIGURE 6-20

NOTE *Note that a filename for files added to the assets folder must be in lowercase letters. As such, a filename such as* `MyDB` *is invalid, whereas* `mydb` *is fine.*

2. Add the following statements in bold to the `DatabasesActivity.java` file:

```
package net.learn2develop.Databases;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

import android.app.Activity;
import android.database.Cursor;
import android.os.Bundle;
import android.widget.Toast;

public class DatabasesActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        DBAdapter db = new DBAdapter(this);
        try {
            String destPath = "/data/data/" + getPackageName() +
                "/databases";
            File f = new File(destPath);
            if (!f.exists()) {
                f.mkdirs();
                f.createNewFile();

                //---copy the db from the assets folder into
                // the databases folder---
                CopyDB(getBaseContext().getAssets().open("mydb"),
                    new FileOutputStream(destPath + "/MyDB"));
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }

        //---get all contacts---
        db.open();
        Cursor c = db.getAllContacts();
        if (c.moveToFirst())
        {
            do {
                DisplayContact(c);
            } while (c.moveToNext());
        }
        db.close();
```

```
    }

    public void CopyDB(InputStream inputStream,
    OutputStream outputStream) throws IOException {
        //---copy 1K bytes at a time---
        byte[] buffer = new byte[1024];
        int length;
        while ((length = inputStream.read(buffer)) > 0) {
            outputStream.write(buffer, 0, length);
        }
        inputStream.close();
        outputStream.close();
    }

    public void DisplayContact(Cursor c)
    {
        Toast.makeText(this,
                "id: " + c.getString(0) + "\n" +
                "Name: " + c.getString(1) + "\n" +
                "Email:  " + c.getString(2),
                Toast.LENGTH_LONG).show();
    }
}
```

3. Press F11 to debug the application on the Android emulator. When the application runs, it will copy the `mydb` database file into the `/data/data/net.learn2develop.Databases/databases/` folder with the name `MyDB`.

How It Works

You first defined the `CopyDB()` method to copy the database file from one location to another:

```
public void CopyDB(InputStream inputStream,
OutputStream outputStream) throws IOException {
    //---copy 1K bytes at a time---
    byte[] buffer = new byte[1024];
    int length;
    while ((length = inputStream.read(buffer)) > 0) {
        outputStream.write(buffer, 0, length);
    }
    inputStream.close();
    outputStream.close();
}
```

Note that in this case you used the `InputStream` object to read from the source file, and then wrote it to the destination file using the `OutputStream` object.

When the activity is created, you copy the database file located in the `assets` folder into the `/data/data/net.learn2develop.Databases/databases/` folder on the Android device (or emulator):

```
    try {
        String destPath = "/data/data/" + getPackageName() +
            "/databases";
        File f = new File(destPath);
```

```
            if (!f.exists()) {
                f.mkdirs();
                f.createNewFile();

                //---copy the db from the assets folder into
                // the databases folder---
                CopyDB(getBaseContext().getAssets().open("mydb"),
                    new FileOutputStream(destPath + "/MyDB"));
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
```

You copy the database file only if it does not exist in the destination folder. If you don't perform this check, every time the activity is created you will overwrite the database file with the one in the `assets` folder. This may not be desirable, as your application may make changes to the database file during runtime, and this will overwrite all the changes you have made so far.

To ensure that the database file is indeed copied, be sure to delete the database file in your emulator (if it already existed) prior to testing the application. You can delete the database using DDMS (see Figure 6-21).

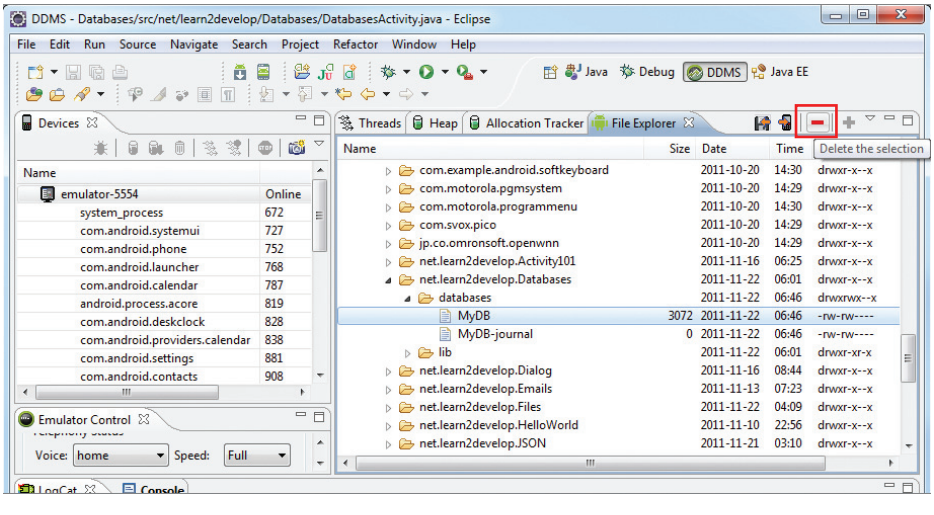

FIGURE 6-21

SUMMARY

In this chapter, you learned the different ways to save persistent data to your Android device. For simple unstructured data, using the `SharedPreferences` object is the ideal solution. If you need to store bulk data, then consider using the traditional file system. Finally, for structured data, it is

more efficient to store it in a relational database management system. For this, Android provides the SQLite database, which you can access easily using the APIs exposed.

Note that for the `SharedPreferences` object and the SQLite database, the data is accessible only by the application that creates it. In other words, it is not shareable. If you need to share data among different applications, you need to create a *content provider*. Content providers are discussed in more detail in the next chapter.

EXERCISES

1. How do you display the preferences of your application using an activity?
2. Name the method that enables you to obtain the path of the external storage of an Android device.
3. What is the permission you need to declare when writing files to external storage?

Answers to the exercises can be found in Appendix C.

WHAT YOU LEARNED IN THIS CHAPTER

TOPIC	KEY CONCEPTS
Saving simple user data	Use the `SharedPreferences` object.
Sharing data among activities in the same application	Use the `getSharedPreferences()` method.
Saving to a file	Use the `FileOutputStream` and `OutputStreamReader` classes.
Reading from a file	Use the `FileInputStream` and `InputStreamReader` classes.
Saving to external storage	Use the `getExternalStorageDirectory()` method to return the path to the external storage.
Accessing files in the `res/raw` folder	Use the `openRawResource()` method in the `Resources` object (obtained via the `getResources()` method).
Creating a database helper class	Extend the `SQLiteOpenHelper` class.

7

Content Providers

WHAT YOU WILL LEARN IN THIS CHAPTER

- What are content providers?
- How to use a content provider in Android
- How to create and use your own content provider

In the previous chapter, you learned about the various ways to persist data — using shared preferences, files, as well as SQLite databases. While using the database approach is the recommended way to save structured and complex data, sharing data is a challenge because the database is accessible to only the package that created it.

In this chapter, you will learn Android's way of sharing data through the use of *content providers*. You will learn how to use the built-in content providers, as well as implement your own content providers to share data across packages.

SHARING DATA IN ANDROID

In Android, using a content provider is the recommended way to share data across packages. Think of a content provider as a data store. How it stores its data is not relevant to the application using it; what is important is how packages can access the data stored in it using a consistent programming interface. A content provider behaves very much like a database — you can query it, edit its content, as well as add or delete content. However, unlike a database, a content provider can use different ways to store its data. The data can be stored in a database, in files, or even over a network.

Android ships with many useful content providers, including the following:

- **Browser** — Stores data such as browser bookmarks, browser history, and so on
- **CallLog** — Stores data such as missed calls, call details, and so on

- **Contacts** — Stores contact details
- **MediaStore** — Stores media files such as audio, video, and images
- **Settings** — Stores the device's settings and preferences

Besides the many built-in content providers, you can also create your own content providers.

To query a content provider, you specify the query string in the form of a URI, with an optional specifier for a particular row. The format of the query URI is as follows:

```
<standard_prefix>://<authority>/<data_path>/<id>
```

The various parts of the URI are as follows:

- The *standard prefix* for content providers is always `content://`.
- The *authority* specifies the name of the content provider. An example would be `contacts` for the built-in Contacts content provider. For third-party content providers, this could be the fully qualified name, such as `com.wrox.provider` or `net.learn2develop.provider`.
- The *data path* specifies the kind of data requested. For example, if you are getting all the contacts from the Contacts content provider, then the data path would be `people`, and the URI would look like this: `content://contacts/people`.
- The *id* specifies the specific record requested. For example, if you are looking for contact number 2 in the Contacts content provider, the URI would look like this: `content://contacts/people/2`.

Table 7-1 shows some examples of query strings.

TABLE 7-1: Example Query Strings

QUERY STRING	DESCRIPTION
`content://media/internal/images`	Returns a list of all the internal images on the device
`content://media/external/images`	Returns a list of all the images stored on the external storage (e.g., SD card) on the device
`content://call_log/calls`	Returns a list of all calls registered in the Call Log
`content://browser/bookmarks`	Returns a list of bookmarks stored in the browser

USING A CONTENT PROVIDER

The best way to understand content providers is to actually use one. The following Try It Out shows how you can use a content provider from within your Android application.

TRY IT OUT Using the Contacts Content Provider

codefile Provider.zip available for download at Wrox.com

1. Using Eclipse, create a new Android project and name it **Provider**.
2. Add the following statements in bold to 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" >

<ListView
    android:id="@+id/android:list"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:stackFromBottom="false"
    android:transcriptMode="normal" />

<TextView
    android:id="@+id/contactName"
    android:textStyle="bold"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />

<TextView
    android:id="@+id/contactID"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />

</LinearLayout>
```

3. In the `ProviderActivity.java` class, code the following:

```
package net.learn2develop.Provider;

import android.app.ListActivity;
import android.content.CursorLoader;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.provider.ContactsContract;
import android.widget.CursorAdapter;
import android.widget.SimpleCursorAdapter;

public class ProviderActivity extends ListActivity  {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

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

        Uri allContacts = Uri.parse("content://contacts/people");

        Cursor c;
        if (android.os.Build.VERSION.SDK_INT <11) {
            //---before Honeycomb---
            c = managedQuery(allContacts, null, null, null, null);
        } else {
            //---Honeycomb and later---
            CursorLoader cursorLoader = new CursorLoader(
                    this,
                    allContacts,
                    null,
                    null,
                    null ,
                    null);
            c = cursorLoader.loadInBackground();
        }

        String[] columns = new String[] {
            ContactsContract.Contacts.DISPLAY_NAME,
            ContactsContract.Contacts._ID};

        int[] views = new int[] {R.id.contactName, R.id.contactID};

        SimpleCursorAdapter adapter;

        if (android.os.Build.VERSION.SDK_INT <11) {
            //---before Honeycomb---
            adapter = new SimpleCursorAdapter(
                    this, R.layout.main, c, columns, views);
        } else {
            //---Honeycomb and later---
            adapter = new SimpleCursorAdapter(
                    this, R.layout.main, c, columns, views,
                    CursorAdapter.FLAG_REGISTER_CONTENT_OBSERVER);
        }

        this.setListAdapter(adapter);   }
}
```

4. Add the following statements in bold to the `AndroidManifest.xml` file:

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

    <uses-sdk android:minSdkVersion="14" />
    <uses-permission android:name="android.permission.READ_CONTACTS"/>

    <application
```

```
            android:icon="@drawable/ic_launcher"
            android:label="@string/app_name" >
            <activity
                android:label="@string/app_name"
                android:name=".ProviderActivity" >
                <intent-filter >
                    <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

5. Launch an AVD and create a few contacts in the Android Emulator. To add a contact, go to the Phone application and click the Star icon at the top (see Figure 7-1). Click the MENU button on the emulator and click the New contact menu item. You will be warned about backing up your contacts. Click the Keep Local button and enter the name, phone number, and e-mail address of a few people.

6. Press F11 to debug the application on the Android emulator. Figure 7-2 shows the activity displaying the list of contacts you just created.

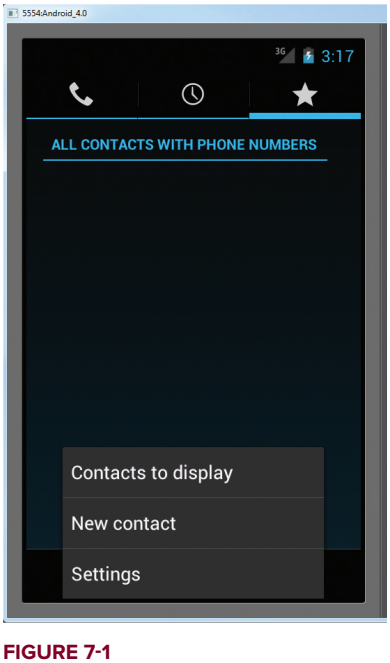

FIGURE 7-1

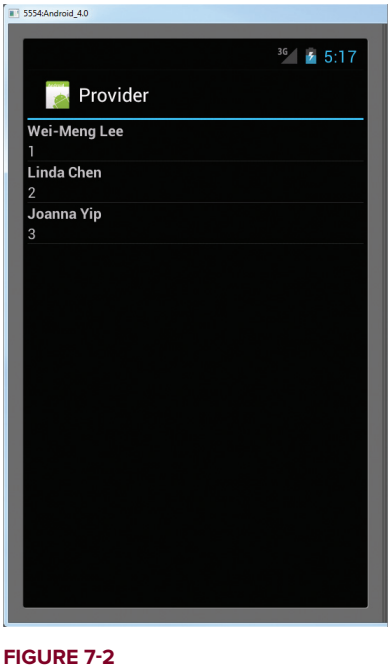

FIGURE 7-2

How It Works

In this example, you retrieved all the contacts stored in the Contacts application and displayed them in the `ListView`.

First, you specified the URI for accessing the Contacts application:

```
Uri allContacts = Uri.parse("content://contacts/people");
```

Next, observe that you have a conditional check to detect the version of the device on which the application is currently running:

```
Cursor c;
if (android.os.Build.VERSION.SDK_INT <11) {
    //---before Honeycomb---
    c = managedQuery(allContacts, null, null, null, null);
} else {
    //---Honeycomb and later---
    CursorLoader cursorLoader = new CursorLoader(
            this,
            allContacts,
            null,
            null,
            null ,
            null);
    c = cursorLoader.loadInBackground();
}
```

If the application is running on pre-Honeycomb devices (the value of the `android.os.Build.VERSION.SDK_INT` variable is lower than 11), you can use the `managedQuery()` method of the `Activity` class to retrieve a managed cursor. A *managed cursor* handles all the work of unloading itself when the application pauses and requerying itself when the application restarts. The statement

```
Cursor c = managedQuery(allContacts, null, null, null, null);
```

is equivalent to

```
Cursor c = getContentResolver().query(allContacts, null, null, null, null);
//---allows the activity to manage the Cursor's
// lifecyle based on the activity's lifecycle---
startManagingCursor(c);
```

The `getContentResolver()` method returns a `ContentResolver` object, which helps to resolve a content URI with the appropriate content provider.

However, beginning with Android API level 11 (Honeycomb and later), the `managedQuery()` method is deprecated (still available but not recommended for use). For Honeycomb or later devices, use the `CursorLoader` class:

```
    CursorLoader cursorLoader = new CursorLoader(
            this,
            allContacts,
```

```
                null,
                null,
                null ,
                null);
        c = cursorLoader.loadInBackground();
```

The `CursorLoader` class (only available beginning with Android API level 11 and later) performs the cursor query on a background thread and hence does not block the application UI.

The `SimpleCursorAdapter` object maps a cursor to `TextViews` (or `ImageViews`) defined in your XML file (`main.xml`). It maps the data (as represented by `columns`) to views (as represented by `views`):

```
String[] columns = new String[] {
    ContactsContract.Contacts.DISPLAY_NAME,
    ContactsContract.Contacts._ID};

int[] views = new int[] {R.id.contactName, R.id.contactID};

SimpleCursorAdapter adapter;

if (android.os.Build.VERSION.SDK_INT <11) {
    //---before Honeycomb---
    adapter = new SimpleCursorAdapter(
            this, R.layout.main, c, columns, views);
} else {
    //---Honeycomb and later---
    adapter = new SimpleCursorAdapter(
            this, R.layout.main, c, columns, views,
            CursorAdapter.FLAG_REGISTER_CONTENT_OBSERVER);
}

this.setListAdapter(adapter);
```

Like the `managedQuery()` method, one of the constructors for the `SimpleCursorAdapter` class has been deprecated. For Honeycomb or later devices, you need to use the new constructor for the `SimpleCursorAdapter` class with one additional argument:

```
    //---Honeycomb and later---
    adapter = new SimpleCursorAdapter(
            this, R.layout.main, c, columns, views,
            CursorAdapter.FLAG_REGISTER_CONTENT_OBSERVER);
```

The flag registers the adapter to be informed when there is a change in the content provider.

Note that in order for your application to access the Contacts application, you need to have the `READ_CONTACTS` permission in your `AndroidManifest.xml` file.

Predefined Query String Constants

Besides using the query URI, you can use a list of predefined query string constants in Android to specify the URI for the different data types. For example, besides using the query `content://contacts/people`, you can rewrite the statement

```
Uri allContacts = Uri.parse("content://contacts/people");
```

using one of the predefined constants in Android, as follows:

```
Uri allContacts = ContactsContract.Contacts.CONTENT_URI;
```

NOTE *For Android 2.0 and later, to query the base Contacts records you need to use the* `ContactsContract.Contacts.CONTENT_URI` *URI.*

Some examples of predefined query string constants are as follows:

- `Browser.BOOKMARKS_URI`
- `Browser.SEARCHES_URI`
- `CallLog.CONTENT_URI`
- `MediaStore.Images.Media.INTERNAL_CONTENT_URI`
- `MediaStore.Images.Media.EXTERNAL_CONTENT_URI`
- `Settings.CONTENT_URI`

If you want to retrieve the first contact, specify the ID of that contact, like this:

```
Uri allContacts = Uri.parse("content://contacts/people/1");
```

Alternatively, use the predefined constant together with the `withAppendedId()` method of the `ContentUris` class:

```
import android.content.ContentUris;
...
        Uri allContacts = ContentUris.withAppendedId(
            ContactsContract.Contacts.CONTENT_URI, 1);
```

Besides binding to a `ListView`, you can also print out the results using the `Cursor` object, as shown here:

```
package net.learn2develop.Provider;

import android.app.ListActivity;
import android.content.CursorLoader;
import android.database.Cursor;
```

```
import android.net.Uri;
import android.os.Bundle;
import android.provider.ContactsContract;
import android.widget.CursorAdapter;
import android.widget.SimpleCursorAdapter;
import android.util.Log;
public class ProviderActivity extends ListActivity  {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        Uri allContacts = ContactsContract.Contacts.CONTENT_URI;
        ...
        ...
        if (android.os.Build.VERSION.SDK_INT <11) {
            //---before Honeycomb---
            adapter = new SimpleCursorAdapter(
                    this, R.layout.main, c, columns, views);
        } else {
            //---Honeycomb and later---
            adapter = new SimpleCursorAdapter(
                    this, R.layout.main, c, columns, views,
                    CursorAdapter.FLAG_REGISTER_CONTENT_OBSERVER);
        }

        this.setListAdapter(adapter);
        PrintContacts(c);
    }

    private void PrintContacts(Cursor c)
    {
        if (c.moveToFirst()) {
            do{
            String contactID = c.getString(c.getColumnIndex(
                   ContactsContract.Contacts._ID));
            String contactDisplayName =
                  c.getString(c.getColumnIndex(
                      ContactsContract.Contacts.DISPLAY_NAME));
                Log.v("Content Providers", contactID + ", " +
                    contactDisplayName);
            } while (c.moveToNext());
        }
    }
}
```

NOTE *If you don't know how to view the LogCat window, refer to Appendix A for a quick tour of the Eclipse IDE.*

The `PrintContacts()` method will print out the following in the LogCat window:

```
12-13 08:32:50.471: V/Content Providers(12346): 1, Wei-Meng Lee
12-13 08:32:50.471: V/Content Providers(12346): 2, Linda Chen
12-13 08:32:50.471: V/Content Providers(12346): 3, Joanna Yip
```

It prints out the ID and name of each contact stored in the Contacts application. In this case, you access the `ContactsContract.Contacts._ID` field to obtain the ID of a contact, and `ContactsContract.Contacts.DISPLAY_NAME` for the name of a contact. If you want to display the phone number of a contact, you need to query the content provider again, as the information is stored in another table:

```
private void PrintContacts(Cursor c)
{
    if (c.moveToFirst()) {
        do{
            String contactID = c.getString(c.getColumnIndex(
                    ContactsContract.Contacts._ID));
            String contactDisplayName =
                    c.getString(c.getColumnIndex(
                            ContactsContract.Contacts.DISPLAY_NAME));
            Log.v("Content Providers", contactID + ", " +
                    contactDisplayName);

            //---get phone number---
            int hasPhone =
                    c.getInt(c.getColumnIndex(
                            ContactsContract.Contacts.HAS_PHONE_NUMBER));
            if (hasPhone == 1) {
                Cursor phoneCursor =
                    getContentResolver().query(
                        ContactsContract.CommonDataKinds.Phone.CONTENT_URI, null,
                        ContactsContract.CommonDataKinds.Phone.CONTACT_ID + " = " +
                        contactID, null, null);
                while (phoneCursor.moveToNext()) {
                    Log.v("Content Providers",
                        phoneCursor.getString(
                            phoneCursor.getColumnIndex(
                                ContactsContract.CommonDataKinds.Phone.NUMBER)));
                }
                phoneCursor.close();
            }

        } while (c.moveToNext());
    }
}
```

NOTE *To access the phone number of a contact, you need to query against the URI stored in* `ContactsContract.CommonDataKinds.Phone.CONTENT_URI`.

In the preceding code snippet, you first check whether a contact has a phone number using the `ContactsContract.Contacts.HAS_PHONE_NUMBER` field. If the contact has at least a phone number, you then query the content provider again based on the ID of the contact. Once the phone number(s) are retrieved, you then iterate through them and print out the numbers. You should see something like this:

```
12-13 08:59:31.881: V/Content Providers(13351): 1, Wei-Meng Lee
12-13 08:59:32.311: V/Content Providers(13351): +651234567
12-13 08:59:32.321: V/Content Providers(13351): 2, Linda Chen
12-13 08:59:32.511: V/Content Providers(13351): +1 876-543-21
12-13 08:59:32.545: V/Content Providers(13351): 3, Joanna Yip
12-13 08:59:32.641: V/Content Providers(13351): +239 846 5522
```

Projections

The second parameter of the `managedQuery()` method (third parameter for the `CursorLoader` class) controls how many columns are returned by the query; this parameter is known as the *projection*. Earlier, you specified `null`:

```
Cursor c;
if (android.os.Build.VERSION.SDK_INT <11) {
    //---before Honeycomb---
    c = managedQuery(allContacts, null, null, null, null);
} else {
    //---Honeycomb and later---
    CursorLoader cursorLoader = new CursorLoader(
            this,
            allContacts,
            null,
            null,
            null ,
            null);
    c = cursorLoader.loadInBackground();
}
```

You can specify the exact columns to return by creating an array containing the name of the column to return, like this:

```
String[] projection = new String[]
        {ContactsContract.Contacts._ID,
         ContactsContract.Contacts.DISPLAY_NAME,
         ContactsContract.Contacts.HAS_PHONE_NUMBER};

Cursor c;
if (android.os.Build.VERSION.SDK_INT <11) {
    //---before Honeycomb---
    c = managedQuery(allContacts, projection, null, null, null);
} else {
    //---Honeycomb and later---
    CursorLoader cursorLoader = new CursorLoader(
            this,
```

```
                allContacts,
                projection,
                null,
                null ,
                null);
        c = cursorLoader.loadInBackground();
}
```

In the above case, the `_ID`, `DISPLAY_NAME`, and `HAS_PHONE_NUMBER` fields will be retrieved.

Filtering

The third and fourth parameters of the `managedQuery()` method (fourth and fifth parameters for the `CursorLoader` class) enable you to specify a SQL `WHERE` clause to filter the result of the query. For example, the following statement retrieves only the people whose name ends with "Lee":

```
Cursor c;
if (android.os.Build.VERSION.SDK_INT <11) {
    //---before Honeycomb---
    c = managedQuery(allContacts, projection,
            ContactsContract.Contacts.DISPLAY_NAME + " LIKE '%Lee'", null, null);
} else {
    //---Honeycomb and later---
    CursorLoader cursorLoader = new CursorLoader(
            this,
            allContacts,
            projection,
            ContactsContract.Contacts.DISPLAY_NAME + " LIKE '%Lee'",
            null ,
            null);
    c = cursorLoader.loadInBackground();
}
```

Here, the third parameter (for the `managedQuery()` method, and the fourth parameter for the `CursorLoader` constructor) contains a SQL statement containing the name to search for ("Lee"). You can also put the search string into the next argument of the method/constructor, like this:

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

```
                new String[] {"%Lee"},
                null);
        c = cursorLoader.loadInBackground();
    }
```

Sorting

The last parameter of the `managedQuery()` method (and constructor for the `CursorLoader` class) enables you to specify a SQL `ORDER BY` clause to sort the result of the query. For example, the following statement sorts the contact names in ascending order:

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

CREATING YOUR OWN CONTENT PROVIDERS

Creating your own content provider in Android is relatively simple. All you need to do is extend the abstract `ContentProvider` class and override the various methods defined within it.

In this section, you will learn how to create a simple content provider that stores a list of books. For ease of illustration, the content provider stores the books in a database table containing three fields, as shown in Figure 7-3.

_id	title	isbn

FIGURE 7-3

The following Try It Out shows you the steps.

TRY IT OUT Creating Your Own Content Provider

codefile ContentProviders.zip available for download at Wrox.com

1. Using Eclipse, create a new Android project and name it **ContentProviders**.
2. In the `src` folder of the project, add a new Java class file and name it **BooksProvider**.

3. Populate the `BooksProvider.java` file as follows:

```
package net.learn2develop.ContentProviders;

import android.content.ContentProvider;
import android.content.ContentUris;
import android.content.ContentValues;
import android.content.Context;
import android.content.UriMatcher;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteQueryBuilder;
import android.net.Uri;
import android.text.TextUtils;
import android.util.Log;

public class BooksProvider extends ContentProvider {
    static final String PROVIDER_NAME =
        "net.learn2develop.provider.Books";

    static final Uri CONTENT_URI =
        Uri.parse("content://"+ PROVIDER_NAME + "/books");

    static final String _ID = "_id";
    static final String TITLE = "title";
    static final String ISBN = "isbn";

    static final int BOOKS = 1;
    static final int BOOK_ID = 2;

    private static final UriMatcher uriMatcher;
    static{
        uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        uriMatcher.addURI(PROVIDER_NAME, "books", BOOKS);
        uriMatcher.addURI(PROVIDER_NAME, "books/#", BOOK_ID);
    }

    //---for database use---
    SQLiteDatabase booksDB;
    static final String DATABASE_NAME = "Books";
    static final String DATABASE_TABLE = "titles";
    static final int DATABASE_VERSION = 1;
    static final String DATABASE_CREATE =
        "create table " + DATABASE_TABLE +
        " (_id integer primary key autoincrement, "
        + "title text not null, isbn text not null);";

    private static class DatabaseHelper extends SQLiteOpenHelper
    {
        DatabaseHelper(Context context) {
```

```
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db)
    {
        db.execSQL(DATABASE_CREATE);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion,
            int newVersion) {
        Log.w("Content provider database",
                "Upgrading database from version " +
                        oldVersion + " to " + newVersion +
                ", which will destroy all old data");
        db.execSQL("DROP TABLE IF EXISTS titles");
        onCreate(db);
    }
}

@Override
public int delete(Uri arg0, String arg1, String[] arg2) {
    // arg0 = uri
    // arg1 = selection
    // arg2 = selectionArgs
    int count=0;
    switch (uriMatcher.match(arg0)){
    case BOOKS:
        count = booksDB.delete(
                DATABASE_TABLE,
                arg1,
                arg2);
        break;
    case BOOK_ID:
        String id = arg0.getPathSegments().get(1);
        count = booksDB.delete(
                DATABASE_TABLE,
                _ID + " = " + id +
                (!TextUtils.isEmpty(arg1) ? " AND (" +
                        arg1 + ')' : ""),
                        arg2);
        break;
    default: throw new IllegalArgumentException("Unknown URI " + arg0);
    }
    getContext().getContentResolver().notifyChange(arg0, null);
    return count;
}

@Override
public String getType(Uri uri) {
    switch (uriMatcher.match(uri)){
    //---get all books---
    case BOOKS:
```

```
            return "vnd.android.cursor.dir/vnd.learn2develop.books ";

        //---get a particular book---
        case BOOK_ID:
            return "vnd.android.cursor.item/vnd.learn2develop.books ";

        default:
            throw new IllegalArgumentException("Unsupported URI: " + uri);
        }
    }

    @Override
    public Uri insert(Uri uri, ContentValues values) {
        //---add a new book---
        long rowID = booksDB.insert(
                DATABASE_TABLE,
                "",
                values);

        //---if added successfully---
        if (rowID>0)
        {
            Uri _uri = ContentUris.withAppendedId(CONTENT_URI, rowID);
            getContext().getContentResolver().notifyChange(_uri, null);
            return _uri;
        }
        throw new SQLException("Failed to insert row into " + uri);
    }

    @Override
    public boolean onCreate() {
        Context context = getContext();
        DatabaseHelper dbHelper = new DatabaseHelper(context);
        booksDB = dbHelper.getWritableDatabase();
        return (booksDB == null)? false:true;
    }

    @Override
    public Cursor query(Uri uri, String[] projection, String selection,
            String[] selectionArgs, String sortOrder) {
        SQLiteQueryBuilder sqlBuilder = new SQLiteQueryBuilder();
        sqlBuilder.setTables(DATABASE_TABLE);

        if (uriMatcher.match(uri) == BOOK_ID)
            //---if getting a particular book---
            sqlBuilder.appendWhere(
                    _ID + " = " + uri.getPathSegments().get(1));

        if (sortOrder==null || sortOrder=="")
            sortOrder = TITLE;

        Cursor c = sqlBuilder.query(
            booksDB,
            projection,
```

```
            selection,
            selectionArgs,
            null,
            null,
            sortOrder);

        //---register to watch a content URI for changes---
        c.setNotificationUri(getContext().getContentResolver(), uri);
        return c;
    }

    @Override
    public int update(Uri uri, ContentValues values, String selection,
            String[] selectionArgs) {
        int count = 0;
        switch (uriMatcher.match(uri)){
        case BOOKS:
            count = booksDB.update(
                    DATABASE_TABLE,
                    values,
                    selection,
                    selectionArgs);
            break;
        case BOOK_ID:
            count = booksDB.update(
                    DATABASE_TABLE,
                    values,
                    _ID + " = " + uri.getPathSegments().get(1) +
                    (!TextUtils.isEmpty(selection) ? " AND (" +
                            selection + ')' : ""),
                            selectionArgs);
            break;
        default: throw new IllegalArgumentException("Unknown URI " + uri);
        }
        getContext().getContentResolver().notifyChange(uri, null);
        return count;
    }
}
```

4. Add the following statements in bold to the `AndroidManifest.xml` file:

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

    <uses-sdk android:minSdkVersion="14" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:label="@string/app_name"
            android:name=".ContentProvidersActivity" >
```

```
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <provider android:name="BooksProvider"
            android:authorities="net.learn2develop.provider.Books">
        </provider>
    </application>

</manifest>
```

How It Works

In this example, you first created a class named `BooksProvider` that extends the `ContentProvider` base class. The various methods to override in this class are as follows:

- `getType()` — Returns the MIME type of the data at the given URI
- `onCreate()` — Called when the provider is started
- `query()` — Receives a request from a client. The result is returned as a `Cursor` object.
- `insert()` — Inserts a new record into the content provider
- `delete()` — Deletes an existing record from the content provider
- `update()` — Updates an existing record from the content provider

Within your content provider, you are free to choose how you want to store your data — a traditional file system, XML, a database, or even through web services. For this example, you used the SQLite database approach discussed in the previous chapter.

You then defined the following constants within the `BooksProvider` class:

```
static final String PROVIDER_NAME =
    "net.learn2develop.provider.Books";

static final Uri CONTENT_URI =
    Uri.parse("content://"+ PROVIDER_NAME + "/books");

static final String _ID = "_id";
static final String TITLE = "title";
static final String ISBN = "isbn";

static final int BOOKS = 1;
static final int BOOK_ID = 2;

private static final UriMatcher uriMatcher;
static{
    uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
    uriMatcher.addURI(PROVIDER_NAME, "books", BOOKS);
    uriMatcher.addURI(PROVIDER_NAME, "books/#", BOOK_ID);
}

//---for database use---
SQLiteDatabase booksDB;
```

```
static final String DATABASE_NAME = "Books";
static final String DATABASE_TABLE = "titles";
static final int DATABASE_VERSION = 1;
static final String DATABASE_CREATE =
    "create table " + DATABASE_TABLE +
    " (_id integer primary key autoincrement, "
    + "title text not null, isbn text not null);";
```

Observe in the preceding code that you used an `UriMatcher` object to parse the content URI that is passed to the content provider through a `ContentResolver`. For example, the following content URI represents a request for all books in the content provider:

```
content://net.learn2develop.provider.Books/books
```

The following represents a request for a particular book with `_id` `5`:

```
content://net.learn2develop.provider.Books/books/5
```

Your content provider uses a SQLite database to store the books. Note that you used the `SQLiteOpenHelper` helper class to help manage your database:

```
private static class DatabaseHelper extends SQLiteOpenHelper
{
    DatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db)
    {
        db.execSQL(DATABASE_CREATE);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion,
            int newVersion) {
        Log.w("Content provider database",
                "Upgrading database from version " +
                        oldVersion + " to " + newVersion +
                ", which will destroy all old data");
        db.execSQL("DROP TABLE IF EXISTS titles");
        onCreate(db);
    }
}
```

Next, you override the `getType()` method to uniquely describe the data type for your content provider. Using the `UriMatcher` object, you returned `vnd.android.cursor.item/vnd.learn2develop.books` for a single book, and `vnd.android.cursor.dir/vnd.learn2develop.books` for multiple books:

```
@Override
public String getType(Uri uri) {
    switch (uriMatcher.match(uri)){
    //---get all books---
    case BOOKS:
```

```
            return "vnd.android.cursor.dir/vnd.learn2develop.books ";

        //---get a particular book---
        case BOOK_ID:
            return "vnd.android.cursor.item/vnd.learn2develop.books ";

        default:
            throw new IllegalArgumentException("Unsupported URI: " + uri);
        }
    }
```

Next, you overrode the `onCreate()` method to open a connection to the database when the content provider is started:

```
    @Override
    public boolean onCreate() {
        Context context = getContext();
        DatabaseHelper dbHelper = new DatabaseHelper(context);
        booksDB = dbHelper.getWritableDatabase();
        return (booksDB == null)? false:true;
    }
```

You overrode the `query()` method to allow clients to query for books:

```
    @Override
    public Cursor query(Uri uri, String[] projection, String selection,
            String[] selectionArgs, String sortOrder) {
        SQLiteQueryBuilder sqlBuilder = new SQLiteQueryBuilder();
        sqlBuilder.setTables(DATABASE_TABLE);

        if (uriMatcher.match(uri) == BOOK_ID)
            //---if getting a particular book---
            sqlBuilder.appendWhere(
                    _ID + " = " + uri.getPathSegments().get(1));

        if (sortOrder==null || sortOrder=="")
            sortOrder = TITLE;

        Cursor c = sqlBuilder.query(
            booksDB,
            projection,
            selection,
            selectionArgs,
            null,
            null,
            sortOrder);

        //---register to watch a content URI for changes---
        c.setNotificationUri(getContext().getContentResolver(), uri);
        return c;
    }
```

By default, the result of the query is sorted using the `title` field. The resulting query is returned as a `Cursor` object.

To allow a new book to be inserted into the content provider, you override the `insert()` method:

```
@Override
public Uri insert(Uri uri, ContentValues values) {
    //---add a new book---
    long rowID = booksDB.insert(
            DATABASE_TABLE,
            "",
            values);

    //---if added successfully---
    if (rowID>0)
    {
        Uri _uri = ContentUris.withAppendedId(CONTENT_URI, rowID);
        getContext().getContentResolver().notifyChange(_uri, null);
        return _uri;
    }
    throw new SQLException("Failed to insert row into " + uri);
}
```

Once the record is inserted successfully, you call the `notifyChange()` method of the `ContentResolver`. This notifies registered observers that a row was updated.

To delete a book, you override the `delete()` method:

```
@Override
public int delete(Uri arg0, String arg1, String[] arg2) {
    // arg0 = uri
    // arg1 = selection
    // arg2 = selectionArgs
    int count=0;
    switch (uriMatcher.match(arg0)){
    case BOOKS:
        count = booksDB.delete(
                DATABASE_TABLE,
                arg1,
                arg2);
        break;
    case BOOK_ID:
        String id = arg0.getPathSegments().get(1);
        count = booksDB.delete(
                DATABASE_TABLE,
                _ID + " = " + id +
                (!TextUtils.isEmpty(arg1) ? " AND (" +
                        arg1 + ')' : ""),
                        arg2);
        break;
    default: throw new IllegalArgumentException("Unknown URI " + arg0);
    }
    getContext().getContentResolver().notifyChange(arg0, null);
    return count;
}
```

Likewise, call the `notifyChange()` method of the `ContentResolver` after the deletion. This will notify registered observers that a row was deleted.

Finally, to update a book, you override the `update()` method:

```
@Override
public int update(Uri uri, ContentValues values, String selection,
        String[] selectionArgs) {
    int count = 0;
    switch (uriMatcher.match(uri)){
    case BOOKS:
        count = booksDB.update(
                DATABASE_TABLE,
                values,
                selection,
                selectionArgs);
        break;
    case BOOK_ID:
        count = booksDB.update(
                DATABASE_TABLE,
                values,
                _ID + " = " + uri.getPathSegments().get(1) +
                (!TextUtils.isEmpty(selection) ? " AND (" +
                        selection + ')' : ""),
                        selectionArgs);
        break;
    default: throw new IllegalArgumentException("Unknown URI " + uri);
    }
    getContext().getContentResolver().notifyChange(uri, null);
    return count;
}
```

As with the `insert()` and `delete()` methods, you called the `notifyChange()` method of the `ContentResolver` after the update. This notifies registered observers that a row was updated.

Finally, to register your content provider with Android, modify the `AndroidManifest.xml` file by adding the `<provider>` element.

USING THE CONTENT PROVIDER

Now that you have built your new content provider, you can test it from within your Android application. The following Try It Out demonstrates how to do that.

TRY IT OUT Using the Newly Created Content Provider

1. Using the same project created in the previous section, add the following statements in bold to 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" >

<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="ISBN" />

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

<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Title" />

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

<Button
    android:text="Add title"
    android:id="@+id/btnAdd"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:onClick="onClickAddTitle" />

<Button
    android:text="Retrieve titles"
    android:id="@+id/btnRetrieve"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:onClick="onClickRetrieveTitles"  />

</LinearLayout>
```

2. In the `ContentProvidersActivity.java` file, add the following statements in bold:

```
package net.learn2develop.ContentProviders;

import android.app.Activity;
import android.content.ContentValues;
import android.content.CursorLoader;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

public class ContentProvidersActivity extends Activity {
    /** Called when the activity is first created. */
```

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void onClickAddTitle(View view) {
        //---add a book---
        ContentValues values = new ContentValues();
        values.put(BooksProvider.TITLE, ((EditText)
                findViewById(R.id.txtTitle)).getText().toString());
        values.put(BooksProvider.ISBN, ((EditText)
                findViewById(R.id.txtISBN)).getText().toString());
        Uri uri = getContentResolver().insert(
                BooksProvider.CONTENT_URI, values);
        Toast.makeText(getBaseContext(),uri.toString(),
                Toast.LENGTH_LONG).show();
    }

    public void onClickRetrieveTitles(View view) {
        //---retrieve the titles---
        Uri allTitles = Uri.parse(
                "content://net.learn2develop.provider.Books/books");
        Cursor c;
        if (android.os.Build.VERSION.SDK_INT <11) {
            //---before Honeycomb---
            c = managedQuery(allTitles, null, null, null,
                    "title desc");
        } else {
            //---Honeycomb and later---
            CursorLoader cursorLoader = new CursorLoader(
                    this,
                    allTitles, null, null, null,
                    "title desc");
            c = cursorLoader.loadInBackground();
        }
        if (c.moveToFirst()) {
            do{
                Toast.makeText(this,
                    c.getString(c.getColumnIndex(
                        BooksProvider._ID)) + ", " +
                    c.getString(c.getColumnIndex(
                        BooksProvider.TITLE)) + ", " +
                    c.getString(c.getColumnIndex(
                        BooksProvider.ISBN)),
                    Toast.LENGTH_SHORT).show();
            } while (c.moveToNext());
        }
    }

}
```

3. Press F11 to debug the application on the Android emulator.

4. Enter an ISBN and title for a book and click the Add title button. Figure 7-4 shows the `Toast` class displaying the URI of the book added to the content provider. To retrieve all the titles stored in the content provider, click the Retrieve titles button and observe the values displayed using the `Toast` class.

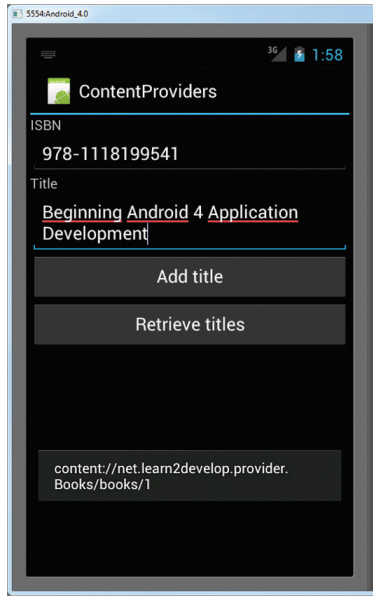

FIGURE 7-4

How It Works

First, you modified the activity so that users can enter a book's ISBN and title to add to the content provider that you have just created.

To add a book to the content provider, you create a new `ContentValues` object and then populate it with the various information about a book:

```
//---add a book---
ContentValues values = new ContentValues();
values.put(BooksProvider.TITLE, ((EditText)
        findViewById(R.id.txtTitle)).getText().toString());
values.put(BooksProvider.ISBN, ((EditText)
        findViewById(R.id.txtISBN)).getText().toString());
Uri uri = getContentResolver().insert(
        BooksProvider.CONTENT_URI, values);
```

Notice that because your content provider is in the same package, you can use the `BooksProvider.TITLE` and the `BooksProvider.ISBN` constants to refer to the `"title"` and `"isbn"` fields, respectively. If you were accessing this content provider from another package, then you would not be able to use these constants. In that case, you need to specify the field name directly, like this:

```
ContentValues values = new ContentValues();
values.put("title", ((EditText)
    findViewById(R.id.txtTitle)).getText().toString());
values.put("isbn", ((EditText)
    findViewById(R.id.txtISBN)).getText().toString());
Uri uri = getContentResolver().insert(
        Uri.parse(
            "content://net.learn2develop.provider.Books/books"),
            values);
```

Also note that for external packages, you need to refer to the content URI using the fully qualified content URI:

```
        Uri.parse(
            "content://net.learn2develop.provider.Books/books"),
```

To retrieve all the titles in the content provider, you used the following code snippets:

```
//---retrieve the titles---
Uri allTitles = Uri.parse(
        "content://net.learn2develop.provider.Books/books");
Cursor c;
if (android.os.Build.VERSION.SDK_INT <11) {
    //---before Honeycomb---
    c = managedQuery(allTitles, null, null, null,
            "title desc");
} else {
    //---Honeycomb and later---
    CursorLoader cursorLoader = new CursorLoader(
            this,
            allTitles, null, null, null,
            "title desc");
    c = cursorLoader.loadInBackground();
}
if (c.moveToFirst()) {
    do{
        Toast.makeText(this,
            c.getString(c.getColumnIndex(
                BooksProvider._ID)) + ", " +
            c.getString(c.getColumnIndex(
                BooksProvider.TITLE)) + ", " +
            c.getString(c.getColumnIndex(
                BooksProvider.ISBN)),
            Toast.LENGTH_SHORT).show();
    } while (c.moveToNext());
}
```

The preceding query will return the result sorted in descending order based on the `title` field.

If you want to update a book's detail, call the `update()` method with the content URI, indicating the book's ID:

```
ContentValues editedValues = new ContentValues();
editedValues.put(BooksProvider.TITLE, "Android Tips and Tricks");
getContentResolver().update(
    Uri.parse(
        "content://net.learn2develop.provider.Books/books/2"),
        editedValues,
        null,
        null);
```

To delete a book, use the `delete()` method with the content URI, indicating the book's ID:

```
//---delete a title---
getContentResolver().delete(
        Uri.parse("content://net.learn2develop.provider.Books/books/2"),
        null, null);
```

To delete all books, simply omit the book's ID in your content URI:

```
//---delete all titles---
getContentResolver().delete(
        Uri.parse("content://net.learn2develop.provider.Books/books"),
        null, null);
```

SUMMARY

In this chapter, you learned what content providers are and how to use some of the built-in content providers in Android. In particular, you have seen how to use the Contacts content provider. Google's decision to provide content providers enables applications to share data through a standard set of programming interfaces. In addition to the built-in content providers, you can also create your own custom content provider to share data with other packages.

EXERCISES

1. Write the query to retrieve all contacts from the Contacts application that contain the word "jack."
2. Name the methods that you need to override in your own implementation of a content provider.
3. How do you register a content provider in your `AndroidManifest.xml` file?

Answers to the exercises can be found in Appendix C.

▶ WHAT YOU LEARNED IN THIS CHAPTER

TOPIC	KEY CONCEPTS
Retrieving a managed cursor	Use the `managedQuery()` method (for pre-Honeycomb devices) or use the `CursorLoader` class (for Honeycomb or later devices).
Two ways to specify a query for a content provider	Use either a query URI or a predefined query string constant.
Retrieving the value of a column in a content provider	Use the `getColumnIndex()` method.
Query URI for accessing a contact's name	`ContactsContract.Contacts.CONTENT_URI`
Query URI for accessing a contact's phone number	`ContactsContract.CommonDataKinds.Phone.CONTENT_URI`
Creating your own content provider	Create a class and extend the `ContentProvider` class.

8

Messaging

WHAT YOU WILL LEARN IN THIS CHAPTER

- How to send SMS messages programmatically from within your application
- How to send SMS messages using the built-in Messaging application
- How to receive incoming SMS messages
- How to send e-mail messages from your application

Once your basic Android application is up and running, the next interesting thing you can add to it is the capability to communicate with the outside world. You may want your application to send an SMS message to another phone when an event happens (such as when a particular geographical location is reached), or you may wish to access a web service that provides certain services (such as currency exchange, weather, etc.).

In this chapter, you learn how to send and receive SMS messages programmatically from within your Android application. You will also learn how to invoke the Mail application from within your Android application to send e-mail messages to other users.

SMS MESSAGING

SMS messaging is one of the main *killer applications* on a mobile phone today — for some users as necessary as the phone itself. Any mobile phone you buy today should have at least SMS messaging capabilities, and nearly all users of any age know how to send and receive such messages. Android comes with a built-in SMS application that enables you to send and receive SMS messages. However, in some cases you might want to integrate SMS capabilities into your

own Android application. For example, you might want to write an application that automatically sends an SMS message at regular time intervals. For example, this would be useful if you wanted to track the location of your kids — simply give them an Android device that sends out an SMS message containing its geographical location every 30 minutes. Now you know if they really went to the library after school! (Of course, such a capability also means you would have to pay the fees incurred from sending all those SMS messages…)

This section describes how you can programmatically send and receive SMS messages in your Android applications. The good news for Android developers is that you don't need a real device to test SMS messaging: The free Android emulator provides that capability.

Sending SMS Messages Programmatically

You will first learn how to send SMS messages programmatically from within your application. Using this approach, your application can automatically send an SMS message to a recipient without user intervention. The following Try It Out shows you how.

TRY IT OUT Sending SMS Messages

codefile SMS.zip available for download at Wrox.com

1. Using Eclipse, create a new Android project and name it **SMS**.
2. Replace the `TextView` with the following statements in bold 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" >

<Button
    android:id="@+id/btnSendSMS"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Send SMS"
    android:onClick="onClick" />

</LinearLayout>
```

3. In the `AndroidManifest.xml` file, add the following statements in bold:

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

    <uses-sdk android:minSdkVersion="14" />
```

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

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:label="@string/app_name"
            android:name=".SMSActivity" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

4. Add the following statements in bold to the `SMSActivity.java` file:

```
package net.learn2develop.SMS;

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

import android.telephony.SmsManager;
import android.view.View;

public class SMSActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void onClick(View v) {
        sendSMS("5556", "Hello my friends!");
    }

    //---sends an SMS message to another device---
    private void sendSMS(String phoneNumber, String message)
    {
        SmsManager sms = SmsManager.getDefault();
        sms.sendTextMessage(phoneNumber, null, message, null, null);
    }
}
```

5. Press F11 to debug the application on the Android emulator. Using the Android SDK and AVD Manager, launch another AVD.

6. On the first Android emulator (5554), click the Send SMS button to send an SMS message to the second emulator (5556). Figure 8-1 shows the SMS message received by the second emulator (note the notification bar at the top).

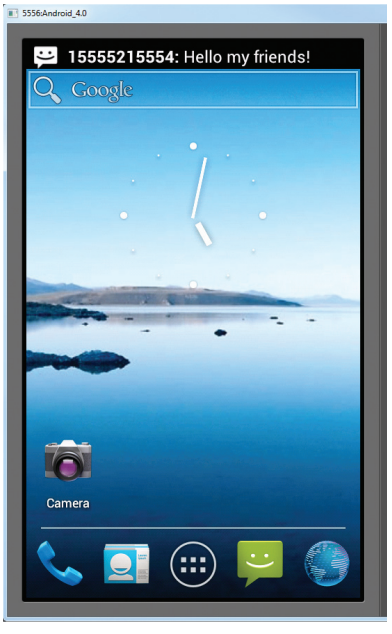

FIGURE 8-1

How It Works

Android uses a permissions-based policy whereby all the permissions needed by an application must be specified in the `AndroidManifest.xml` file. This ensures that when the application is installed, the user knows exactly which access permissions it requires.

Because sending SMS messages incurs additional costs on the user's end, indicating the SMS permissions in the `AndroidManifest.xml` file enables users to decide whether to allow the application to install or not.

To send an SMS message programmatically, you use the `SmsManager` class. Unlike other classes, you do not directly instantiate this class; instead, you call the `getDefault()` static method to obtain an `SmsManager` object. You then send the SMS message using the `sendTextMessage()` method:

```
//---sends an SMS message to another device---
private void sendSMS(String phoneNumber, String message)
{
    SmsManager sms = SmsManager.getDefault();
    sms.sendTextMessage(phoneNumber, null, message, null, null);
}
```

Following are the five arguments to the `sendTextMessage()` method:

- `destinationAddress` — Phone number of the recipient
- `scAddress` — Service center address; use `null` for default SMSC
- `text` — Content of the SMS message
- `sentIntent` — Pending intent to invoke when the message is sent (discussed in more detail in the next section)
- `deliveryIntent` — Pending intent to invoke when the message has been delivered (discussed in more detail in the next section)

NOTE *If you send an SMS message programmatically using the* `SmsManager` *class, the message sent will not appear in the built-in Messaging application of the sender.*

Getting Feedback after Sending a Message

In the previous section, you learned how to programmatically send SMS messages using the `SmsManager` class; but how do you know that the message has been sent correctly? To do so, you can create two `PendingIntent` objects to monitor the status of the SMS message-sending process. These two `PendingIntent` objects are passed to the last two arguments of the `sendTextMessage()` method. The following code snippets show how you can monitor the status of the SMS message being sent:

```
package net.learn2develop.SMS;

import android.app.Activity;
import android.app.PendingIntent;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;

import android.telephony.SmsManager;
import android.view.View;
import android.widget.Toast;

public class SMSActivity extends Activity {
    String SENT = "SMS_SENT";
    String DELIVERED = "SMS_DELIVERED";
    PendingIntent sentPI, deliveredPI;
    BroadcastReceiver smsSentReceiver, smsDeliveredReceiver;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        sentPI = PendingIntent.getBroadcast(this, 0,
                new Intent(SENT), 0);

        deliveredPI = PendingIntent.getBroadcast(this, 0,
                new Intent(DELIVERED), 0);
    }

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

        //---create the BroadcastReceiver when the SMS is sent---
        smsSentReceiver = new BroadcastReceiver(){
            @Override
            public void onReceive(Context arg0, Intent arg1) {
                switch (getResultCode())
```

```
                {
                case Activity.RESULT_OK:
                    Toast.makeText(getBaseContext(), "SMS sent",
                            Toast.LENGTH_SHORT).show();
                    break;
                case SmsManager.RESULT_ERROR_GENERIC_FAILURE:
                    Toast.makeText(getBaseContext(), "Generic failure",
                            Toast.LENGTH_SHORT).show();
                    break;
                case SmsManager.RESULT_ERROR_NO_SERVICE:
                    Toast.makeText(getBaseContext(), "No service",
                            Toast.LENGTH_SHORT).show();
                    break;
                case SmsManager.RESULT_ERROR_NULL_PDU:
                    Toast.makeText(getBaseContext(), "Null PDU",
                            Toast.LENGTH_SHORT).show();
                    break;
                case SmsManager.RESULT_ERROR_RADIO_OFF:
                    Toast.makeText(getBaseContext(), "Radio off",
                            Toast.LENGTH_SHORT).show();
                    break;
                }
            }
        };

        //---create the BroadcastReceiver when the SMS is delivered---
        smsDeliveredReceiver = new BroadcastReceiver(){
            @Override
            public void onReceive(Context arg0, Intent arg1) {
                switch (getResultCode())
                {
                case Activity.RESULT_OK:
                    Toast.makeText(getBaseContext(), "SMS delivered",
                            Toast.LENGTH_SHORT).show();
                    break;
                case Activity.RESULT_CANCELED:
                    Toast.makeText(getBaseContext(), "SMS not delivered",
                            Toast.LENGTH_SHORT).show();
                    break;
                }
            }
        };

        //---register the two BroadcastReceivers---
        registerReceiver(smsDeliveredReceiver, new IntentFilter(DELIVERED));
        registerReceiver(smsSentReceiver, new IntentFilter(SENT));
    }

    @Override
    public void onPause() {
        super.onPause();
        //---unregister the two BroadcastReceivers---
        unregisterReceiver(smsSentReceiver);
```

```
        unregisterReceiver(smsDeliveredReceiver);
    }

    public void onClick(View v) {
        sendSMS("5556", "Hello my friends!");
    }

    //---sends an SMS message to another device---
    private void sendSMS(String phoneNumber, String message)
    {
        SmsManager sms = SmsManager.getDefault();
        sms.sendTextMessage(phoneNumber, null, message, sentPI, deliveredPI);
    }
}
```

The preceding example created two `PendingIntent` objects in the `onCreate()` method:

```
sentPI = PendingIntent.getBroadcast(this, 0,
        new Intent(SENT), 0);

deliveredPI = PendingIntent.getBroadcast(this, 0,
        new Intent(DELIVERED), 0);
```

These two `PendingIntent` objects will be used to send broadcasts later when an SMS message has been sent ("`SMS_SENT`") and delivered ("`SMS_DELIVERED`").

In the `onResume()` method, you then created and registered two `BroadcastReceivers`. These two `BroadcastReceivers` listen for intents that match "`SMS_SENT`" and "`SMS_DELIVERED`" (which are fired by the `SmsManager` when the message has been sent and delivered, respectively):

```
//---register the two BroadcastReceivers---
registerReceiver(smsDeliveredReceiver, new IntentFilter(DELIVERED));
registerReceiver(smsSentReceiver, new IntentFilter(SENT));
```

Within each `BroadcastReceiver` you override the `onReceive()` method and get the current result code.

The two `PendingIntent` objects are passed into the last two arguments of the `sendTextMessage()` method:

```
SmsManager sms = SmsManager.getDefault();
sms.sendTextMessage(phoneNumber, null, message, sentPI, deliveredPI);
```

In this case, whether a message has been sent correctly or failed to be delivered, you will be notified of its status via the two `PendingIntent` objects.

Finally, in the `onPause()` method, you unregister the two `BroadcastReceivers` objects.

NOTE *If you test the application on the Android emulator, only the* `sentPI PendingIntent` *object will be fired, but not the* `deliveredPI PendingIntent` *object. On a real device, both* `PendingIntent` *objects will fire.*

Sending SMS Messages Using Intent

Using the `SmsManager` class, you can send SMS messages from within your application without the need to involve the built-in Messaging application. However, sometimes it would be easier if you could simply invoke the built-in Messaging application and let it do all the work of sending the message.

To activate the built-in Messaging application from within your application, you can use an `Intent` object together with the MIME type `"vnd.android-dir/mms-sms"`, as shown in the following code snippet:

```
Intent i = new
        Intent(android.content.Intent.ACTION_VIEW);
i.putExtra("address", "5556; 5558; 5560");

i.putExtra("sms_body", "Hello my friends!");
i.setType("vnd.android-dir/mms-sms");
startActivity(i);
```

This will invoke the Messaging application, as shown in Figure 8-2. Note that you can send your SMS to multiple recipients by simply separating each phone number with a semi-colon (in the `putExtra()` method). The numbers will be separated using commas in the Messaging application.

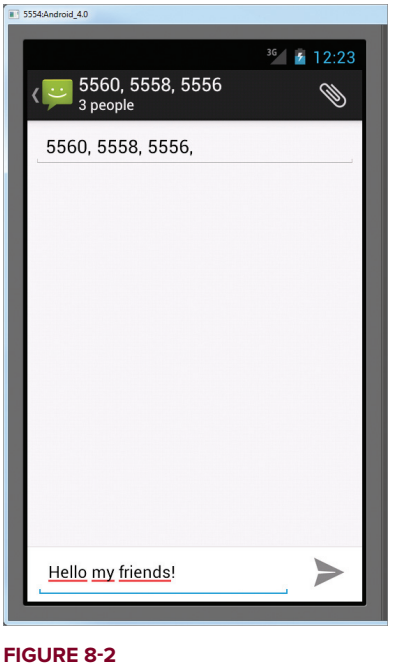

FIGURE 8-2

NOTE *If you use this method to invoke the Messaging application, there is no need to ask for the* `SEND_SMS` *permission in* `AndroidManifest.xml` *because your application is ultimately not the one sending the message.*

Receiving SMS Messages

Besides sending SMS messages from your Android applications, you can also receive incoming SMS messages from within your applications by using a `BroadcastReceiver` object. This is useful when you want your application to perform an action when a certain SMS message is received. For example, you might want to track the location of your phone in case it is lost or stolen. In this case, you can write an application that automatically listens for SMS messages containing some secret code. Once that message is received, you can then send an SMS message containing the location's coordinates back to the sender.

The following Try It Out shows how to programmatically listen for incoming SMS messages.

TRY IT OUT Receiving SMS Messages

1. Using the same project created in the previous section, add the following statements in bold to the `AndroidManifest.xml` file:

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

    <uses-sdk android:minSdkVersion="10" />
    <uses-permission android:name="android.permission.SEND_SMS"/>
    <uses-permission android:name="android.permission.RECEIVE_SMS"/>

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:label="@string/app_name"
            android:name=".SMSActivity" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <receiver android:name=".SMSReceiver">
            <intent-filter>
                <action android:name=
                    "android.provider.Telephony.SMS_RECEIVED" />
            </intent-filter>
        </receiver>
    </application>

</manifest>
```

2. In the `src` folder of the project, add a new Class file to the package name and call it **SMSReceiver** (see Figure 8-3).

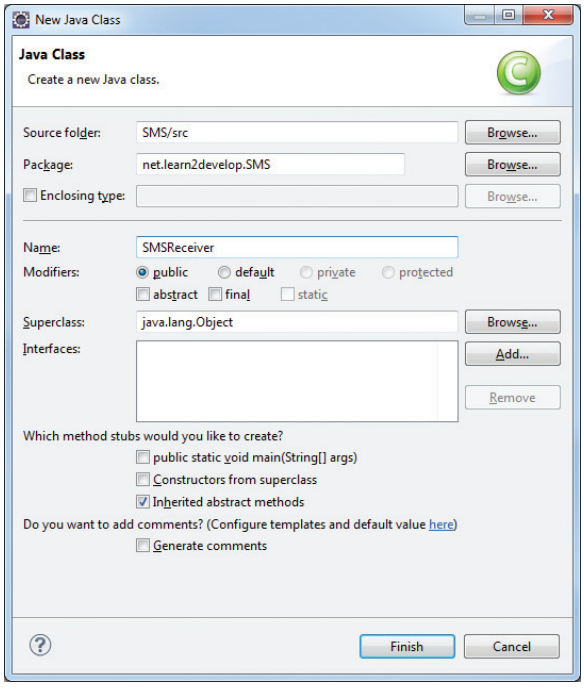

FIGURE 8-3

3. Code the `SMSReceiver.java` file as follows:

```
package net.learn2develop.SMS;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.telephony.SmsMessage;
import android.util.Log;
import android.widget.Toast;

public class SMSReceiver extends BroadcastReceiver
{
    @Override
    public void onReceive(Context context, Intent intent)
    {
        //---get the SMS message passed in---
        Bundle bundle = intent.getExtras();
        SmsMessage[] msgs = null;
        String str = "SMS from ";
        if (bundle != null)
```

```
        {
            //---retrieve the SMS message received---
            Object[] pdus = (Object[]) bundle.get("pdus");
            msgs = new SmsMessage[pdus.length];
            for (int i=0; i<msgs.length; i++){
                msgs[i] = SmsMessage.createFromPdu((byte[])pdus[i]);
                if (i==0) {
                    //---get the sender address/phone number---
                    str += msgs[i].getOriginatingAddress();
                    str += ": ";
                }
                //---get the message body---
                str += msgs[i].getMessageBody().toString();
            }
            //---display the new SMS message---
            Toast.makeText(context, str, Toast.LENGTH_SHORT).show();
            Log.d("SMSReceiver", str);
        }
    }
}
```

4. Press F11 to debug the application on the Android emulator.

5. Using the DDMS, send a message to the emulator. Your application should be able to receive the message and display it using the `Toast` class (see Figure 8-4).

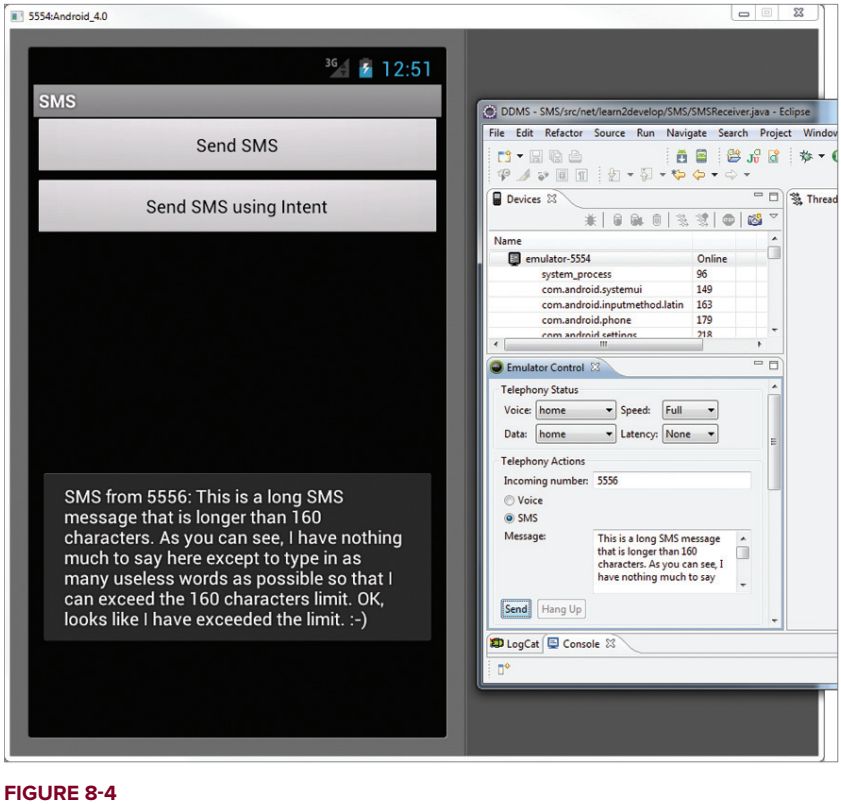

FIGURE 8-4

How It Works

To listen for incoming SMS messages, you create a `BroadcastReceiver` class. The `BroadcastReceiver` class enables your application to receive intents sent by other applications using the `sendBroadcast()` method. Essentially, it enables your application to handle events raised by other applications. When an intent is received, the `onReceive()` method is called; hence, you need to override this.

When an incoming SMS message is received, the `onReceive()` method is fired. The SMS message is contained in the `Intent` object (`intent`; the second parameter in the `onReceive()` method) via a `Bundle` object. Note that each SMS message received will invoke the `onReceive()` method. If your device receives five SMS messages, then the `onReceive()` method will be called five times.

Each SMS message is stored in an `Object` array in the PDU format. If the SMS message is fewer than 160 characters, then the array will have one element. If an SMS message contains more than 160 characters, then the message will be split into multiple smaller messages and stored as multiple elements in the array.

To extract the content of each message, you use the static `createFromPdu()` method from the `SmsMessage` class. The phone number of the sender is obtained via the `getOriginatingAddress()` method; therefore, if you need to send an autoreply to the sender, this is the method to obtain the sender's phone number. To extract the body of the message, you use the `getMessageBody()` method.

One interesting characteristic of the `BroadcastReceiver` is that your application will continue to listen for incoming SMS messages even if it is not running; as long as the application is installed on the device, any incoming SMS messages will be received by the application.

Preventing the Messaging Application from Receiving a Message

In the previous section, you may have noticed that every time you send an SMS message to the emulator (or device), both your application and the built-in application receive it. This is because when an SMS message is received, all applications (including the Messaging application) on the Android device take turns handling the incoming message. Sometimes, however, this is not the behavior you want — for example, you might want your application to receive the message and prevent it from being sent to other applications. This is very useful, especially if you are building some kind of tracking application.

The solution is very simple. To prevent an incoming message from being handled by the built-in Messaging application, your application just needs to handle the message before the Messaging app has the chance to do so. To do this, add the `android:priority` attribute to the `<intent-filter>` element, like this:

```
<receiver android:name=".SMSReceiver">
    <intent-filter android:priority="100">
        <action android:name=
            "android.provider.Telephony.SMS_RECEIVED" />
    </intent-filter>
</receiver>
```

Set this attribute to a high number, such as 100. The higher the number, the earlier Android executes your application. When an incoming message is received, your application will execute

first, and you can decide what to do with the message. To prevent other applications from seeing the message, simply call the `abortBroadcast()` method in your `BroadcastReceiver` class:

```
@Override
public void onReceive(Context context, Intent intent)
{
    //---get the SMS message passed in---
    Bundle bundle = intent.getExtras();
    SmsMessage[] msgs = null;
    String str = "SMS from ";
    if (bundle != null)
    {
        //---retrieve the SMS message received---
        Object[] pdus = (Object[]) bundle.get("pdus");
        msgs = new SmsMessage[pdus.length];
        for (int i=0; i<msgs.length; i++){
            msgs[i] = SmsMessage.createFromPdu((byte[])pdus[i]);
            if (i==0) {
                //---get the sender address/phone number---
                str += msgs[i].getOriginatingAddress();
                str += ": ";
            }
            //---get the message body---
            str += msgs[i].getMessageBody().toString();
        }

        //---display the new SMS message---
        Toast.makeText(context, str, Toast.LENGTH_SHORT).show();
        Log.d("SMSReceiver", str);

        //---stop the SMS message from being broadcasted---
        this.abortBroadcast();
    }
}
```

Once you do this, no other applications will be able to receive your SMS messages.

NOTE *Be aware that after the preceding application is installed on your device, all incoming SMS messages will be intercepted by your application and will not appear in your Messaging application ever again.*

Updating an Activity from a BroadcastReceiver

The previous section demonstrated how you can use a `BroadcastReceiver` class to listen for incoming SMS messages and then use the `Toast` class to display the received SMS message. Often, you'll want to send the SMS message back to the main activity of your application. For example, you might wish to display the message in a `TextView`. The following Try It Out demonstrates how you can do this.

TRY IT OUT Creating a View-Based Application Project

1. Using the same project from the previous section, add the following lines in bold to 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" >

<Button
    android:id="@+id/btnSendSMS"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Send SMS"
    android:onClick="onClick" />

<TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />

</LinearLayout>
```

2. Add the following statements in bold to the `SMSReceiver.java` file:

```
package net.learn2develop.SMS;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.telephony.SmsMessage;
import android.util.Log;
import android.widget.Toast;

public class SMSReceiver extends BroadcastReceiver
{
    @Override
    public void onReceive(Context context, Intent intent)
    {
        //---get the SMS message passed in---
        Bundle bundle = intent.getExtras();
        SmsMessage[] msgs = null;
        String str = "SMS from ";
        if (bundle != null)
        {
            //---retrieve the SMS message received---
            Object[] pdus = (Object[]) bundle.get("pdus");
                 msgs = new SmsMessage[pdus.length];
            for (int i=0; i<msgs.length; i++){
                     msgs[i] = SmsMessage.createFromPdu((byte[])pdus[i]);
                if (i==0) {
```

```
                    //---get the sender address/phone number---
                    str += msgs[i].getOriginatingAddress();
                    str += ": ";
                }
                //---get the message body---
                str += msgs[i].getMessageBody().toString();
            }
            //---display the new SMS message---
            Toast.makeText(context, str, Toast.LENGTH_SHORT).show();
            Log.d("SMSReceiver", str);

            //---send a broadcast intent to update the SMS received in the activity---
            Intent broadcastIntent = new Intent();
            broadcastIntent.setAction("SMS_RECEIVED_ACTION");
            broadcastIntent.putExtra("sms", str);
            context.sendBroadcast(broadcastIntent);
        }
    }
}
```

3. Add the following statements in bold to the `SMSActivity.java` file:

```
package net.learn2develop.SMS;

import android.app.Activity;
import android.app.PendingIntent;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;

import android.telephony.SmsManager;
import android.view.View;
import android.widget.TextView;
import android.widget.Toast;

public class SMSActivity extends Activity {
    String SENT = "SMS_SENT";
    String DELIVERED = "SMS_DELIVERED";
    PendingIntent sentPI, deliveredPI;
    BroadcastReceiver smsSentReceiver, smsDeliveredReceiver;
    IntentFilter intentFilter;

    private BroadcastReceiver intentReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            //---display the SMS received in the TextView---
            TextView SMSes = (TextView) findViewById(R.id.textView1);
            SMSes.setText(intent.getExtras().getString("sms"));
        }
    };

    /** Called when the activity is first created. */
```

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    sentPI = PendingIntent.getBroadcast(this, 0,
            new Intent(SENT), 0);

    deliveredPI = PendingIntent.getBroadcast(this, 0,
            new Intent(DELIVERED), 0);

    //---intent to filter for SMS messages received---
    intentFilter = new IntentFilter();
    intentFilter.addAction("SMS_RECEIVED_ACTION");
}

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

    //---register the receiver---
    registerReceiver(intentReceiver, intentFilter);

    //---create the BroadcastReceiver when the SMS is sent---
    smsSentReceiver = new BroadcastReceiver(){
        @Override
        public void onReceive(Context arg0, Intent arg1) {
            switch (getResultCode())
            {
            case Activity.RESULT_OK:
                Toast.makeText(getBaseContext(), "SMS sent",
                        Toast.LENGTH_SHORT).show();
                break;
            case SmsManager.RESULT_ERROR_GENERIC_FAILURE:
                Toast.makeText(getBaseContext(), "Generic failure",
                        Toast.LENGTH_SHORT).show();
                break;
            case SmsManager.RESULT_ERROR_NO_SERVICE:
                Toast.makeText(getBaseContext(), "No service",
                        Toast.LENGTH_SHORT).show();
                break;
            case SmsManager.RESULT_ERROR_NULL_PDU:
                Toast.makeText(getBaseContext(), "Null PDU",
                        Toast.LENGTH_SHORT).show();
                break;
            case SmsManager.RESULT_ERROR_RADIO_OFF:
                Toast.makeText(getBaseContext(), "Radio off",
                        Toast.LENGTH_SHORT).show();
                break;
            }
        }
    };

    //---create the BroadcastReceiver when the SMS is delivered---
```

```
        smsDeliveredReceiver = new BroadcastReceiver(){
            @Override
            public void onReceive(Context arg0, Intent arg1) {
                switch (getResultCode())
                {
                case Activity.RESULT_OK:
                    Toast.makeText(getBaseContext(), "SMS delivered",
                            Toast.LENGTH_SHORT).show();
                    break;
                case Activity.RESULT_CANCELED:
                    Toast.makeText(getBaseContext(), "SMS not delivered",
                            Toast.LENGTH_SHORT).show();
                    break;
                }
            }
        };

        //---register the two BroadcastReceivers---
        registerReceiver(smsDeliveredReceiver, new IntentFilter(DELIVERED));
        registerReceiver(smsSentReceiver, new IntentFilter(SENT));
    }

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

        //---unregister the receiver---
        unregisterReceiver(intentReceiver);

        //---unregister the two BroadcastReceivers---
        unregisterReceiver(smsSentReceiver);
        unregisterReceiver(smsDeliveredReceiver);
    }

    public void onClick(View v) {
        sendSMS("5556", "Hello my friends!");
    }

    public void onSMSIntentClick (View v) {
        Intent i = new
                Intent(android.content.Intent.ACTION_VIEW);
        i.putExtra("address", "5556; 5558; 5560");

        i.putExtra("sms_body", "Hello my friends!");
        i.setType("vnd.android-dir/mms-sms");
        startActivity(i);
    }

    //--sends an SMS message to another device--
    private void sendSMS(String phoneNumber, String message)
    {
        SmsManager sms = SmsManager.getDefault();
        sms.sendTextMessage(phoneNumber, null, message, sentPI, deliveredPI);
    }
}
```

4. Press F11 to debug the application on the Android emulator. Using the DDMS, send an SMS message to the emulator. Figure 8-5 shows the `Toast` class displaying the message received, and the `TextView` showing the message received.

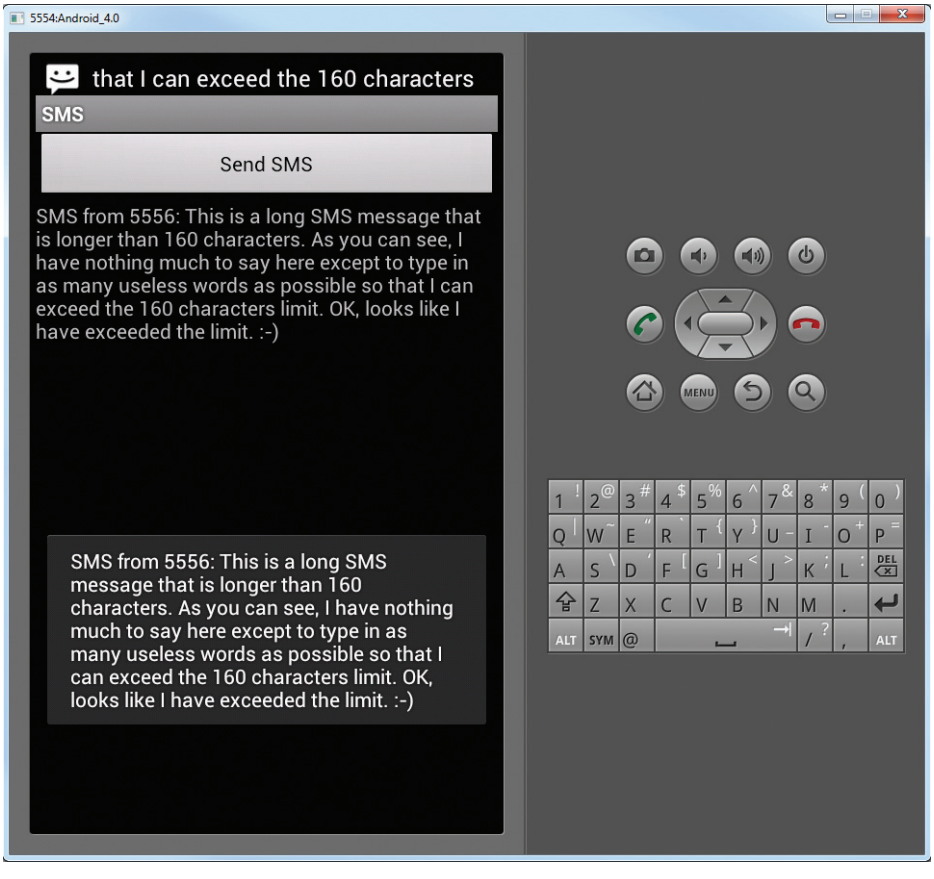

FIGURE 8-5

How It Works

You first added a `TextView` to your activity so that it can be used to display the received SMS message.

Next, you modified the `SMSReceiver` class so that when it receives an SMS message, it broadcasts another `Intent` object so that any applications listening for this intent can be notified (which you will implement in the activity next). The SMS received is also sent out via this intent:

```
//---send a broadcast intent to update the SMS received in the activity---
Intent broadcastIntent = new Intent();
broadcastIntent.setAction("SMS_RECEIVED_ACTION");
broadcastIntent.putExtra("sms", str);
context.sendBroadcast(broadcastIntent);
```

Next, in your activity you created a `BroadcastReceiver` object to listen for broadcast intents:

```
private BroadcastReceiver intentReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        //---display the SMS received in the TextView---
        TextView SMSes = (TextView) findViewById(R.id.textView1);
        SMSes.setText(intent.getExtras().getString("sms"));
    }
};
```

When a broadcast intent is received, you update the SMS message in the `TextView`.

You need to create an `IntentFilter` object so that you can listen for a particular intent. In this case, the intent is `"SMS_RECEIVED_ACTION"`:

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    sentPI = PendingIntent.getBroadcast(this, 0,
            new Intent(SENT), 0);

    deliveredPI = PendingIntent.getBroadcast(this, 0,
            new Intent(DELIVERED), 0);

    //---intent to filter for SMS messages received---
    intentFilter = new IntentFilter();
    intentFilter.addAction("SMS_RECEIVED_ACTION");
}
```

Finally, you register the `BroadcastReceiver` in the activity's `onResume()` event and unregister it in the `onPause()` event:

```
@Override
protected void onResume() {
    //--register the receiver--
    registerReceiver(intentReceiver, intentFilter);
    super.onResume();
}

@Override
protected void onPause() {
    //--unregister the receiver--
    unregisterReceiver(intentReceiver);
    super.onPause();
}


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

    //---register the receiver---
```

```
        registerReceiver(intentReceiver, intentFilter);

        //---create the BroadcastReceiver when the SMS is sent---
        //...
    }

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

        //---unregister the receiver---
        unregisterReceiver(intentReceiver);

        //---unregister the two BroadcastReceivers---
        //...
    }
```

This means that the `TextView` will display the SMS message only when the message is received while the activity is visible on the screen. If the SMS message is received when the activity is not in the foreground, the `TextView` will not be updated.

Invoking an Activity from a BroadcastReceiver

The previous example shows how you can pass the SMS message received to be displayed in the activity. However, in many situations your activity may be in the background when the SMS message is received. In this case, it would be useful to be able to bring the activity to the foreground when a message is received. The following Try It Out shows you how.

TRY IT OUT Invoking an Activity

1. Using the same project used in the previous section, add the following lines in bold to the `SMSActivity.java` file:

```
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        sentPI = PendingIntent.getBroadcast(this, 0,
                new Intent(SENT), 0);

        deliveredPI = PendingIntent.getBroadcast(this, 0,
                new Intent(DELIVERED), 0);

        //---intent to filter for SMS messages received---
        intentFilter = new IntentFilter();
        intentFilter.addAction("SMS_RECEIVED_ACTION");

        //---register the receiver---
```

```
        registerReceiver(intentReceiver, intentFilter);
    }

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

        //---register the receiver---
        //registerReceiver(intentReceiver, intentFilter);

        //---create the BroadcastReceiver when the SMS is sent---
        smsSentReceiver = new BroadcastReceiver(){
            @Override
            public void onReceive(Context arg0, Intent arg1) {
                switch (getResultCode())
                {
                case Activity.RESULT_OK:
                    Toast.makeText(getBaseContext(), "SMS sent",
                            Toast.LENGTH_SHORT).show();
                    break;
                case SmsManager.RESULT_ERROR_GENERIC_FAILURE:
                    Toast.makeText(getBaseContext(), "Generic failure",
                            Toast.LENGTH_SHORT).show();
                    break;
                case SmsManager.RESULT_ERROR_NO_SERVICE:
                    Toast.makeText(getBaseContext(), "No service",
                            Toast.LENGTH_SHORT).show();
                    break;
                case SmsManager.RESULT_ERROR_NULL_PDU:
                    Toast.makeText(getBaseContext(), "Null PDU",
                            Toast.LENGTH_SHORT).show();
                    break;
                case SmsManager.RESULT_ERROR_RADIO_OFF:
                    Toast.makeText(getBaseContext(), "Radio off",
                            Toast.LENGTH_SHORT).show();
                    break;
                }
            }
        };

        //---create the BroadcastReceiver when the SMS is delivered---
        smsDeliveredReceiver = new BroadcastReceiver(){
            @Override
            public void onReceive(Context arg0, Intent arg1) {
                switch (getResultCode())
                {
                case Activity.RESULT_OK:
                    Toast.makeText(getBaseContext(), "SMS delivered",
                            Toast.LENGTH_SHORT).show();
                    break;
                case Activity.RESULT_CANCELED:
                    Toast.makeText(getBaseContext(), "SMS not delivered",
                            Toast.LENGTH_SHORT).show();
                    break;
```

```
            }
        }
    };

    //---register the two BroadcastReceivers---
    registerReceiver(smsDeliveredReceiver, new IntentFilter(DELIVERED));
    registerReceiver(smsSentReceiver, new IntentFilter(SENT));
}

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

    //---unregister the receiver---
    //unregisterReceiver(intentReceiver);

    //---unregister the two BroadcastReceivers---
    unregisterReceiver(smsSentReceiver);
    unregisterReceiver(smsDeliveredReceiver);
}

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

    //---unregister the receiver---
    unregisterReceiver(intentReceiver);
}
```

2. Add the following statements in bold to the `SMSReceiver.java` file:

```
@Override
public void onReceive(Context context, Intent intent)
{
    //---get the SMS message passed in---
    Bundle bundle = intent.getExtras();
    SmsMessage[] msgs = null;
    String str = "SMS from ";
    if (bundle != null)
    {
        //---retrieve the SMS message received---
        Object[] pdus = (Object[]) bundle.get("pdus");
        msgs = new SmsMessage[pdus.length];
        for (int i=0; i<msgs.length; i++){
            msgs[i] = SmsMessage.createFromPdu((byte[])pdus[i]);
            if (i==0) {
                //---get the sender address/phone number---
                str += msgs[i].getOriginatingAddress();
                str += ": ";
            }
            //---get the message body---
            str += msgs[i].getMessageBody().toString();
        }
        //---display the new SMS message---
```

```
            Toast.makeText(context, str, Toast.LENGTH_SHORT).show();
            Log.d("SMSReceiver", str);

            //---launch the SMSActivity---
            Intent mainActivityIntent = new Intent(context, SMSActivity.class);
            mainActivityIntent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
            context.startActivity(mainActivityIntent);

            //---send a broadcast intent to update the SMS received in the activity---
            Intent broadcastIntent = new Intent();
            broadcastIntent.setAction("SMS_RECEIVED_ACTION");
            broadcastIntent.putExtra("sms", str);
            context.sendBroadcast(broadcastIntent);
        }
    }
```

3. Modify the `AndroidManifest.xml` file as follows:

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

4. Press F11 to debug the application on the Android emulator. When the `SMSActivity` is shown, click the Home button to send the activity to the background.

5. Use the DDMS to send an SMS message to the emulator again. This time, note that the activity will be brought to the foreground, displaying the SMS message received.

How It Works

In the `SMSActivity` class, you first registered the `BroadcastReceiver` in the activity's `onCreate()` event, instead of the `onResume()` event; and instead of unregistering it in the `onPause()` event, you unregister it in the `onDestroy()` event. This ensures that even if the activity is in the background, it will still be able to listen for the broadcast intent.

Next, you modified the `onReceive()` event in the `SMSReceiver` class by using an intent to bring the activity to the foreground before broadcasting another intent:

```
            //---launch the SMSActivity---
            Intent mainActivityIntent = new Intent(context, SMSActivity.class);
            mainActivityIntent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
            context.startActivity(mainActivityIntent);

            //---send a broadcast intent to update the SMS received in the activity---
            Intent broadcastIntent = new Intent();
            broadcastIntent.setAction("SMS_RECEIVED_ACTION");
            broadcastIntent.putExtra("sms", str);
            context.sendBroadcast(broadcastIntent);
```

The `startActivity()` method launches the activity and brings it to the foreground. Note that you needed to set the `Intent.FLAG_ACTIVITY_NEW_TASK` flag because calling `startActivity()` from outside of an activity context requires the `FLAG_ACTIVITY_NEW_TASK` flag.

You also needed to set the `launchMode` attribute of the `<activity>` element in the `AndroidManifest.xml` file to `singleTask`:

```
<activity
    android:label="@string/app_name"
    android:name=".SMSActivity"
    android:launchMode="singleTask" >
```

If you don't set this, multiple instances of the activity will be launched as your application receives SMS messages.

Note that in this example, when the activity is in the background (such as when you click the Home button to show the home screen), the activity is brought to the foreground and its `TextView` is updated with the SMS received. However, if the activity was killed (such as when you click the Back button to destroy it), the activity is launched again but the `TextView` is not updated.

Caveats and Warnings

While the capability to send and receive SMS messages makes Android a very compelling platform for developing sophisticated applications, this flexibility comes with a price. A seemingly innocent application may send SMS messages behind the scene without the user knowing, as demonstrated by a recent case of an SMS-based Trojan Android application (see `http://forum.vodafone.co.nz/topic/5719-android-sms-trojan-warning/`). Claiming to be a media player, once installed, the application sends SMS messages to a premium-rate number, resulting in huge phone bills for the user.

While the user needs to explicitly give permissions (such as accessing the Internet, sending and receiving SMS messages, etc.) to your application, the request for permissions is shown only at installation time. If the user clicks the Install button, he or she is considered to have granted the application permission to send and receive SMS messages. This is dangerous, as after the application is installed it can send and receive SMS messages without ever prompting the user again.

In addition to this, the application can also "sniff" for incoming SMS messages. For example, based on the techniques you learned from the previous section, you can easily write an application that checks for certain keywords in the SMS message. When an SMS message contains the keyword you are looking for, you can then use the Location Manager (discussed in Chapter 9) to obtain your geographical location and then send the coordinates back to the sender of the SMS message. The sender could then easily track your location. All these tasks can be done easily without the user knowing it! That said, users should try to avoid installing Android applications that come from dubious sources, such as from unknown websites or strangers.

SENDING E-MAIL

Like SMS messaging, Android also supports e-mail. The Gmail/Email application on Android enables you to configure an e-mail account using POP3 or IMAP. Besides sending and receiving e-mails using the Gmail/Email application, you can also send e-mail messages programmatically from within your Android application. The following Try It Out shows you how.

TRY IT OUT Sending E-mail Programmatically

codefile Emails.zip available for download at Wrox.com

1. Using Eclipse, create a new Android project and name it **Emails**.
2. Add the following statements in bold to the `main.xml` file, replacing the `TextView`:

```
<?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" >

<Button
    android:id="@+id/btnSendEmail"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Send Email"
    android:onClick="onClick" />

</LinearLayout>
```

3. Add the following statements in bold to the `SMSActivity.java` file:

```
package net.learn2develop.Emails;

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

public class EmailsActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void onClick(View v) {
        //---replace the following email addresses with real ones---
        String[] to =
```

```
            {"someguy@example.com",
             "anotherguy@example.com"};
        String[] cc = {"busybody@example.com"};
        sendEmail(to, cc, "Hello", "Hello my friends!");
    }

    //---sends an SMS message to another device---
    private void sendEmail(String[] emailAddresses, String[] carbonCopies,
    String subject, String message)
    {
        Intent emailIntent = new Intent(Intent.ACTION_SEND);
        emailIntent.setData(Uri.parse("mailto:"));
        String[] to = emailAddresses;
        String[] cc = carbonCopies;
        emailIntent.putExtra(Intent.EXTRA_EMAIL, to);
        emailIntent.putExtra(Intent.EXTRA_CC, cc);
        emailIntent.putExtra(Intent.EXTRA_SUBJECT, subject);
        emailIntent.putExtra(Intent.EXTRA_TEXT, message);
        emailIntent.setType("message/rfc822");
        startActivity(Intent.createChooser(emailIntent, "Email"));
    }
}
```

4. Press F11 to test the application on the Android emulator/device (ensure that you have configured your e-mail before trying this example). Click the Send Email button and you should see the Email application launched in your emulator/device, as shown in Figure 8-6.

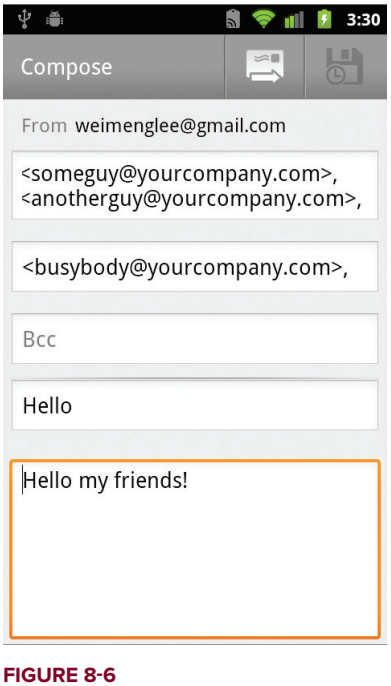

FIGURE 8-6

AN E-MAIL TIP FOR TESTING PURPOSES

If you are a gmail user, for testing purposes you can take advantage of the fact that you can append the plus sign (+) and any string to the user name of your account. You will continue to receive the e-mail as usual, but it can be used for filtering. For example, my Gmail account is weimenglee@gmail.com. When I am doing testing I will configure the recipient address as "weimenglee+android_book@gmail.com" and then I can easily filter out those messages for deletion later. In addition, I can also add periods to my Gmail address. For example, if you want to test sending an e-mail to a couple of people, you can send to weimeng.lee@gmail.com, wei.meng .lee@gmail.com, and wei.menglee@gmail.com; an e-mail to all these e-mail addresses will end up in my weimenglee@gmail.com account as a single message. This is very useful for testing!

Finally, take a look at `http://smtp4dev.codeplex.com`, which contains a dummy SMTP server that allows you to debug e-mail messages.

How It Works

In this example, you are launching the built-in Email application to send an e-mail message. To do so, you use an `Intent` object and set the various parameters using the `setData()`, `putExtra()`, and `setType()` methods:

```
Intent emailIntent = new Intent(Intent.ACTION_SEND);
emailIntent.setData(Uri.parse("mailto:"));
String[] to = emailAddresses;
String[] cc = carbonCopies;
emailIntent.putExtra(Intent.EXTRA_EMAIL, to);
emailIntent.putExtra(Intent.EXTRA_CC, cc);
emailIntent.putExtra(Intent.EXTRA_SUBJECT, subject);
emailIntent.putExtra(Intent.EXTRA_TEXT, message);
emailIntent.setType("message/rfc822");
startActivity(Intent.createChooser(emailIntent, "Email"));
```

SUMMARY

This chapter described the two key ways for your application to communicate with the outside world. You first learned how to send and receive SMS messages. Using SMS, you can build a wide variety of applications that rely on the service provided by your mobile operator. Chapter 9 shows you a good example of how to use SMS messaging to build a location tracker application.

You also learned how to send e-mail messages from within your Android application. You do that by invoking the built-in Email application through the use of an `Intent` object.

EXERCISES

1. Name the two ways in which you can send SMS messages in your Android application.

2. Name the permissions you need to declare in your `AndroidManifest.xml` file for sending and receiving SMS messages.

3. How do you notify an activity from a `BroadcastReceiver`?

Answers to the exercises can be found in Appendix C.

▶ WHAT YOU LEARNED IN THIS CHAPTER

TOPIC	KEY CONCEPTS
Programmatically sending SMS messages	Use the `SmsManager` class.
Getting feedback on messages sent	Use two **`PendingIntent`** objects in the `sendTextMessage()` method.
Sending SMS messages using Intent	Set the intent type to `vnd.android-dir/mms-sms`.
Receiving SMS messages	Implement a `BroadcastReceiver` and set it in the **`AndroidManifest.xml`** file.
Sending e-mail using Intent	Set the intent type to `message/rfc822`.

9

Location-Based Services

WHAT YOU WILL LEARN IN THIS CHAPTER

- Displaying Google Maps in your Android application
- Displaying zoom controls on the map
- Switching between the different map views
- Adding markers to maps
- How to get the address location touched on the map
- Performing geocoding and reverse geocoding
- Obtaining geographical data using GPS, Cell-ID, and Wi-Fi triangulation
- How to monitor for a location
- Building a Location Tracker application

You have all seen the explosive growth of mobile apps in recent years. One category of apps that is very popular is location-based services, commonly known as LBS. LBS apps track your location, and may offer additional services such as locating amenities nearby, as well as offer suggestions for route planning, and so on. Of course, one of the key ingredients in an LBS app is maps, which present a visual representation of your location.

In this chapter, you will learn how to make use of Google Maps in your Android application, and how to manipulate it programmatically. In addition, you will learn how to obtain your geographical location using the `LocationManager` class available in the Android SDK. This chapter ends with a project to build a Location Tracker application that you can install on an Android device and can use to track the location of friends and relatives using SMS messaging.

DISPLAYING MAPS

Google Maps is one of the many applications bundled with the Android platform. In addition to simply using the Maps application, you can also embed it into your own applications and make it do some very cool things. This section describes how to use Google Maps in your Android applications and programmatically perform the following:

- Change the views of Google Maps.
- Obtain the latitude and longitude of locations in Google Maps.
- Perform geocoding and reverse geocoding (translating an address to latitude and longitude and vice versa).
- Add markers to Google Maps.

Creating the Project

To get started, you need to first create an Android project so that you can display Google Maps in your activity.

TRY IT OUT Creating the Google APIs Project

codefile LBS.zip available for download at Wrox.com

1. Using Eclipse, create an Android project and name it **LBS**.

NOTE *In order to use Google Maps in your Android application, you need to ensure that you check the Google APIs as your build target. Google Maps is not part of the standard Android SDK, so you need to find it in the Google APIs add-on.*

2. Once the project is created, note the additional JAR file (`maps.jar`) located under the Google APIs folder (see Figure 9-1).

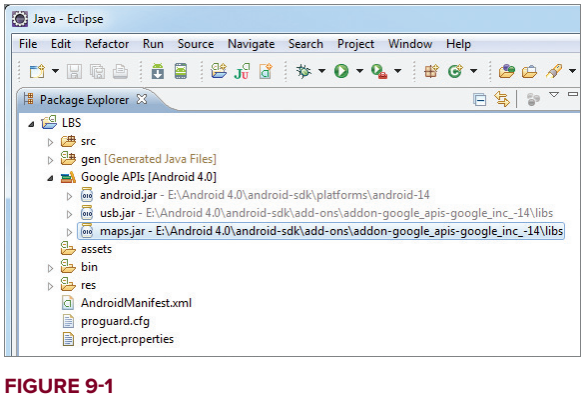

FIGURE 9-1

How It Works

This simple activity created an Android project that uses the Google APIs add-on. The Google APIs add-on, besides including the standard Android and USB libraries, also includes the Maps library, packaged within the `maps.jar` file.

Obtaining the Maps API Key

Beginning with the Android SDK release v1.0, you need to apply for a free Google Maps API key before you can integrate Google Maps into your Android application. When you apply for the key, you must also agree to Google's terms of use, so be sure to read them carefully.

To apply for a key, follow the series of steps outlined next.

NOTE *Google provides detailed documentation about applying for a Maps API key at* `http://code.google.com/android/add-ons/google-apis/mapkey.html`.

First, if you are testing the application on the Android emulator or an Android device directly connected to your development machine, locate the SDK debug certificate located in the default folder (`C:\Users\<username>\.android` for Windows 7 users). You can verify the existence of the debug certificate by going to Eclipse and selecting Window ➪ Preferences. Expand the Android item and select Build (see Figure 9-2). On the right side of the window, you can see the debug certificate's location.

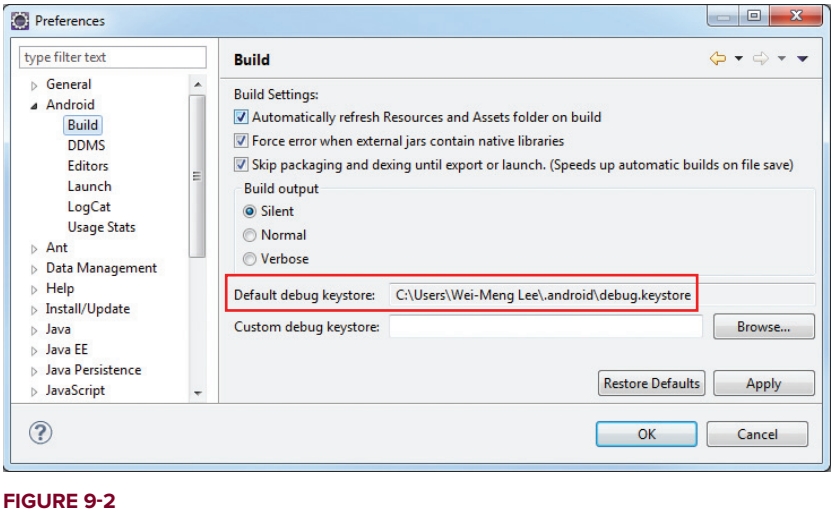

FIGURE 9-2

NOTE *For Windows XP users, the default Android folder is* `C:\Documents and Settings\<username>\Local Settings\Application Data\Android`.

The filename of the debug keystore is `debug.keystore`. This is the certificate that Eclipse uses to sign your application so that it may be run on the Android emulator or devices.

Using the debug keystore, you need to extract its MD5 fingerprint using the `Keytool.exe` application included with your JDK installation. This fingerprint is needed to apply for the free Google Maps key. You can usually find the `Keytool.exe` in the `C:\Program Files\Java\<JDK_version_number>\bin` folder.

Issue the following command (see also Figure 9-3) to extract the MD5 fingerprint:

```
keytool.exe -list -alias androiddebugkey -keystore
"C:\Users\<username>\.android\debug.keystore" -storepass android
-keypass android -v
```

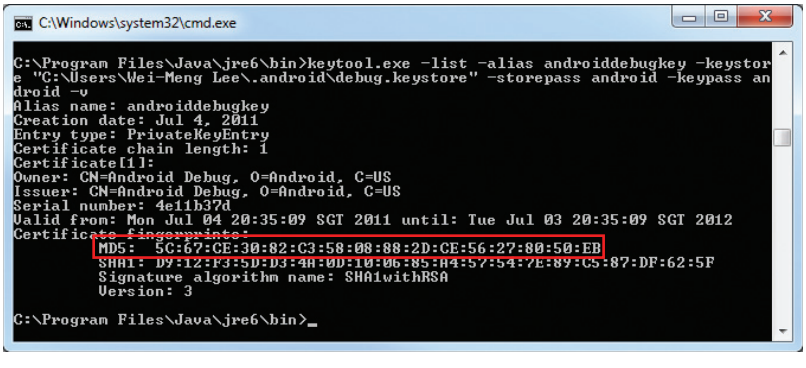

FIGURE 9-3

In this example, my MD5 fingerprint is `5C:67:CE:30:82:C3:58:08:88:2D:CE:56:27:80:50:EB`.

In the preceding command, the following arguments are used:

- `-list` — Shows details about the specified keystore
- `-alias` — The alias for the keystore, which is "`androiddebugkey`" for `debug.keystore`
- `-keystore` — Specifies the location of the keystore
- `-storepass` — Specifies the password for the keystore, which is "`android`" for `debug.keystore`
- `-keypass` — Specifies the password for the key in the keystore, which is "`android`" for `debug.keystore`

Copy the MD5 certificate fingerprint and navigate your web browser to: `http://code.google.com/android/maps-api-signup.html`. Follow the instructions on the page to complete the

application and obtain the Google Maps key. When you are done, you should see something similar to what is shown in Figure 9-4.

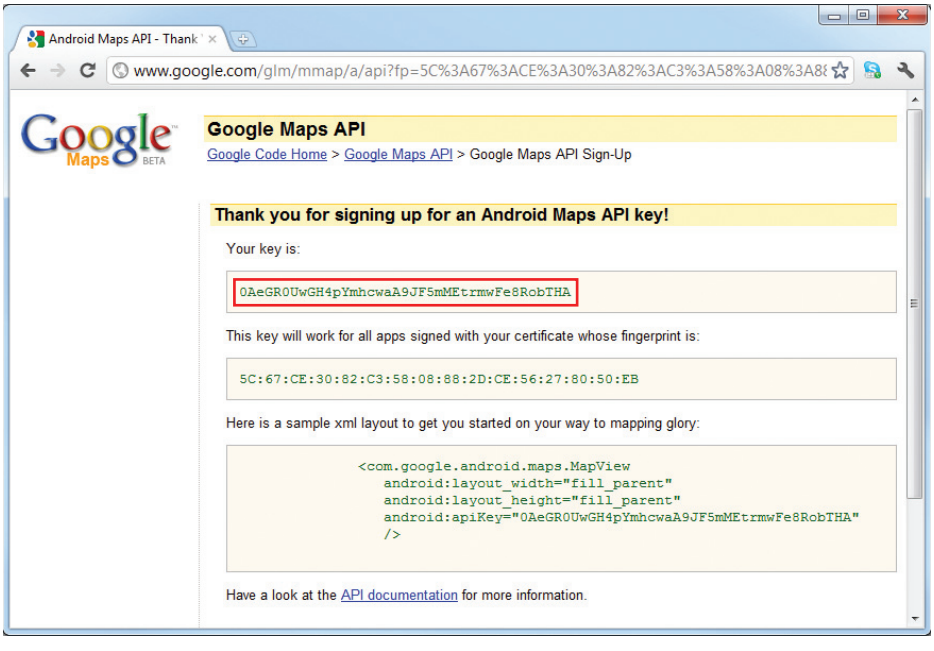

FIGURE 9-4

NOTE *Although you can use the MD5 fingerprint of the debug keystore to obtain the Maps API key for debugging your application on the Android emulator or devices, the key will not be valid if you try to deploy your Android application as an APK file. Once you are ready to deploy your application to the Android Market (or other methods of distribution), you need to reapply for a Maps API key using the certificate that will be used to sign your application. Chapter 12 discusses this topic in more detail.*

Displaying the Map

You are now ready to display Google Maps in your Android application. This involves two main tasks:

- Modify your `AndroidManifest.xml` file by adding both the `<uses-library>` element and the `INTERNET` permission.
- Add the `MapView` element to your UI.

The following Try It Out shows you how.

TRY IT OUT Displaying Google Maps

1. Using the project created in the previous section, replace the `TextView` with the following lines in bold in the `main.xml` file. (Be sure to replace the value of the `android:apiKey` attribute with the API key you obtained earlier):

```
<?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="0AeGR0UwGH4pYmhcwaA9JF5mMEtrmwFe8RobTHA" />

</LinearLayout>
```

2. Add the following lines in bold to the `AndroidManifest.xml` file:

```
<?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"/>

    <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>
```

3. Add the following statements in bold to the `LBSActivity.java` file. Note that `LBSActivity` should now extend the `MapActivity` base class.

```
package net.learn2develop.LBS;

import com.google.android.maps.MapActivity;
import android.os.Bundle;

public class LBSActivity extends MapActivity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    @Override
    protected boolean isRouteDisplayed() {
        // TODO Auto-generated method stub
        return false;
    }
}
```

4. Press F11 to debug the application on the Android emulator. Figure 9-5 shows Google Maps displaying in the activity of the application.

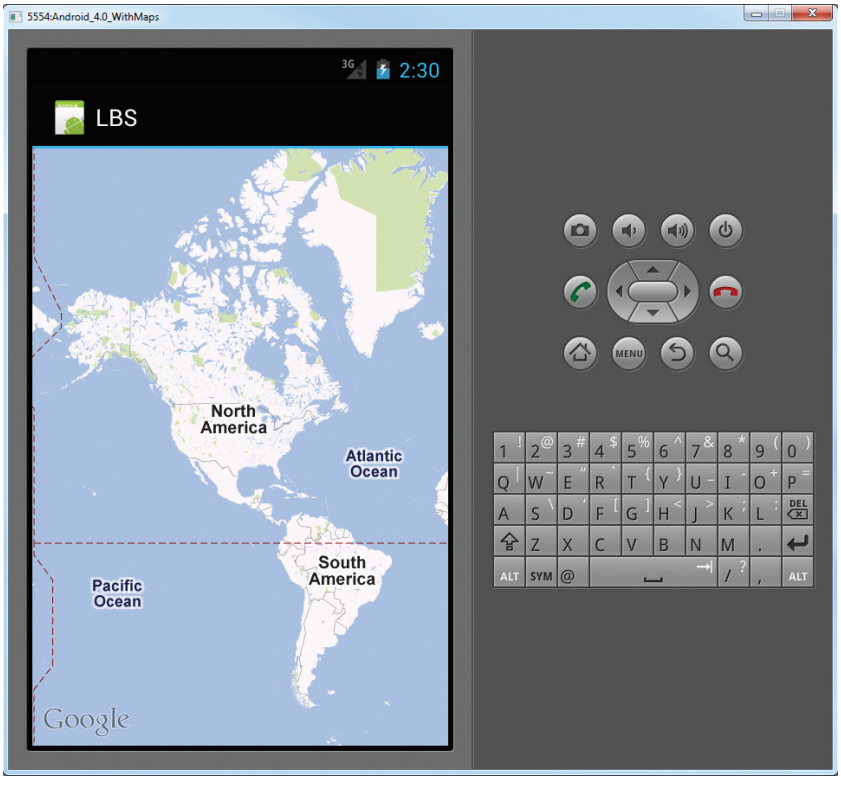

FIGURE 9-5

How It Works

In order to display Google Maps in your application, you first needed the `INTERNET` permission in your manifest file. You then added the `<com.google.android.maps.MapView>` element to your UI file to embed the map within your activity. It is very important that your activity extends the `MapActivity` base class, which itself is an extension of the `Activity` class. For the `MapActivity` class, you need to implement one method: `isRouteDisplayed()`. This method is used for Google's accounting purposes, and you should return `true` for this method if you are displaying routing information on the map. For most simple cases, you can simply return `false`.

In order to test your application on the Android emulator, be sure to create an AVD with the Google Maps API chosen as the selected target.

CAN'T SEE THE MAP?

If instead of seeing Google Maps displayed you see an empty screen with grids, then most likely you are using the wrong API key in the `main.xml` file. It is also possible that you omitted the `INTERNET` permission in your `AndroidManifest.xml` file. Finally, ensure that you have Internet access on your emulator/devices.

If your program does not run (i.e., it crashes), then you probably forgot to add the following statement to the `AndroidManifest.xml` file:

```
<uses-library android:name="com.google.android.maps" />
```

Note its placement in the file; it should be located within the `<Application>` element.

Displaying the Zoom Control

The previous section showed how you can display Google Maps in your Android application. You can pan the map to any desired location and it will be updated on-the-fly. However, on the emulator there is no way to zoom in or out from a particular location (on a real Android device you can pinch the map to zoom it). Thus, in this section, you will learn how you can enable users to zoom in or out of the map using the built-in zoom controls.

TRY IT OUT Displaying the Built-In Zoom Controls

1. Using the project created in the previous section, add the following statements in bold:

```
package net.learn2develop.LBS;

import com.google.android.maps.MapActivity;
import com.google.android.maps.MapView;

import android.os.Bundle;

public class LBSActivity extends MapActivity {
```

```
    MapView mapView;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        mapView = (MapView) findViewById(R.id.mapView);
        mapView.setBuiltInZoomControls(true);
    }

    @Override
    protected boolean isRouteDisplayed() {
        // TODO Auto-generated method stub
        return false;
    }
}
```

2. Press F11 to debug the application on the Android emulator. Observe the built-in zoom controls that appear at the bottom of the map when you click and drag the map (see Figure 9-6). You can click the minus (–) icon to zoom out of the map, and the plus (+) icon to zoom into the map.

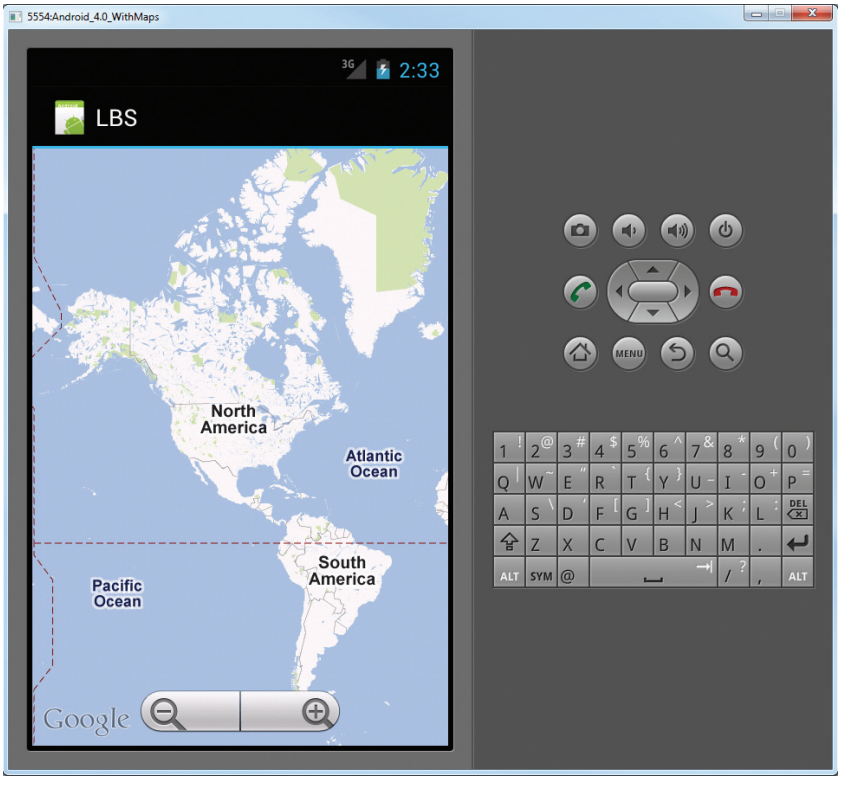

FIGURE 9-6

How It Works

To display the built-in zoom controls, you first get a reference to the MapView and then call the `setBuiltInZoomControls()` method:

```
mapView = (MapView) findViewById(R.id.mapView);
mapView.setBuiltInZoomControls(true);
```

Besides displaying the zoom controls, you can also programmatically zoom in or out of the map using the `zoomIn()` or `zoomOut()` method of the `MapController` class. The following Try It Out shows you how to achieve this.

TRY IT OUT Programmatically Zooming In or Out of the Map

1. Using the project created in the previous section, add the following statements in bold to the `LBSActivity.java` file:

```
package net.learn2develop.LBS;

import com.google.android.maps.MapActivity;
import com.google.android.maps.MapController;
import com.google.android.maps.MapView;

import android.os.Bundle;
import android.view.KeyEvent;

public class LBSActivity extends MapActivity {
    MapView mapView;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        mapView = (MapView) findViewById(R.id.mapView);
        mapView.setBuiltInZoomControls(true);
    }

    public boolean onKeyDown(int keyCode, KeyEvent event)
    {
        MapController mc = mapView.getController();
        switch (keyCode)
        {
            case KeyEvent.KEYCODE_3:
                mc.zoomIn();
                break;
            case KeyEvent.KEYCODE_1:
                mc.zoomOut();
                break;
        }
```

```
        return super.onKeyDown(keyCode, event);
    }

    @Override
    protected boolean isRouteDisplayed() {
        // TODO Auto-generated method stub
        return false;
    }
}
```

2. Press F11 to debug the application on the Android emulator. You can now zoom into the map by clicking the numeric 3 key on the emulator. To zoom out of the map, click the numeric 1 key.

How It Works

To handle key presses on your activity, you handle the `onKeyDown` event:

```
public boolean onKeyDown(int keyCode, KeyEvent event)
{
    MapController mc = mapView.getController();
    switch (keyCode)
    {
        case KeyEvent.KEYCODE_3:
            mc.zoomIn();
            break;
        case KeyEvent.KEYCODE_1:
            mc.zoomOut();
            break;
    }
    return super.onKeyDown(keyCode, event);
}
```

To manage the panning and zooming of the map, you need to obtain an instance of the `MapController` class from the `MapView` object. The `MapController` class contains the `zoomIn()` and `zoomOut()` methods (plus some other methods to control the map) to enable users to zoom in or out of the map, respectively.

Note that if you deploy the application on a real Android device, you may not be able to test the zooming feature, as most Android devices today do not have a physical keyboard.

Changing Views

By default, Google Maps is displayed in *map view*, which is basically drawings of streets and places of interest. You can also set Google Maps to display in *satellite view* using the `setSatellite()` method of the `MapView` class:

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

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

        mapView = (MapView) findViewById(R.id.mapView);
        mapView.setBuiltInZoomControls(true);
        mapView.setSatellite(true);
    }
```

Figure 9-7 shows Google Maps displayed in satellite view.

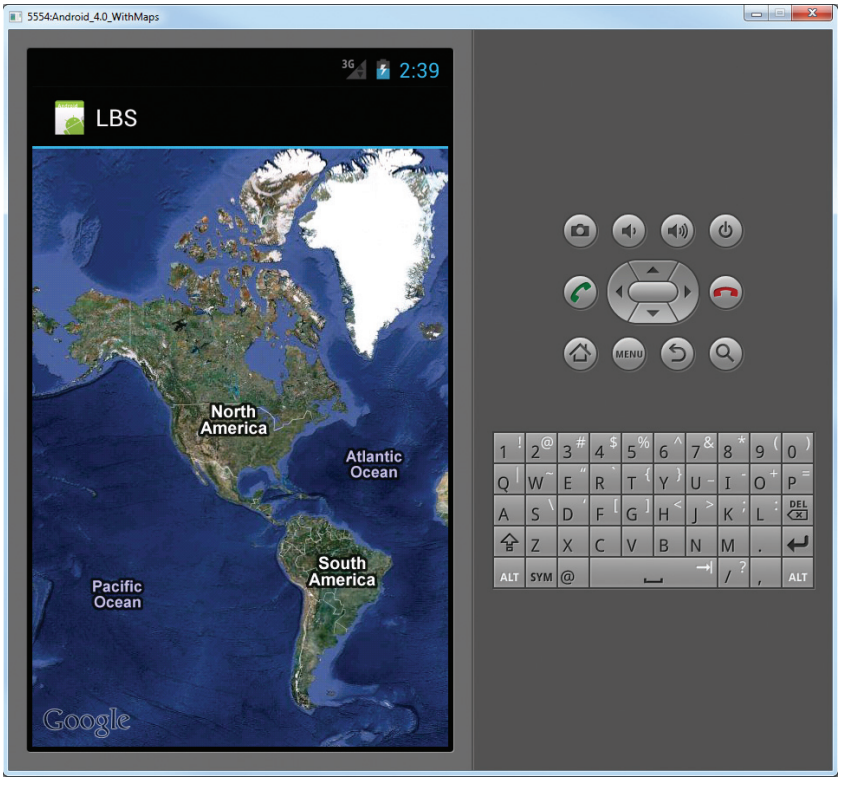

FIGURE 9-7

If you want to display traffic conditions on the map, use the `setTraffic()` method:

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        mapView = (MapView) findViewById(R.id.mapView);
        mapView.setBuiltInZoomControls(true);
        mapView.setSatellite(true);
        mapView.setTraffic(true);
    }
```

Figure 9-8 shows the map displaying the current traffic conditions (you have to zoom in to see the roads). The different colors reflect the varying traffic conditions. In general, green equates to smooth traffic of about 50 miles per hour, yellow equates to moderate traffic of about 25–50 miles per hour, and red equates to slow traffic about less than 25 miles per hour.

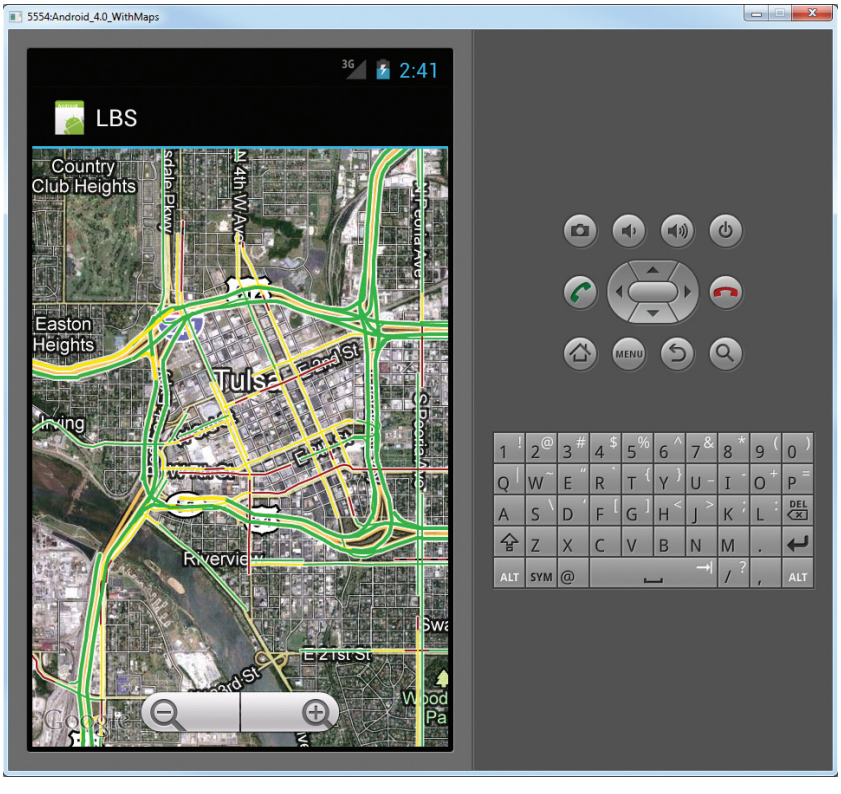

FIGURE 9-8

Note that the traffic information is available only in major cities in the United States, France, Britain, Australia, and Canada, with new cities and countries frequently added.

Navigating to a Specific Location

By default, Google Maps displays the map of the United States when it is first loaded. However, you can set Google Maps to display a particular location. To do so, you can use the `animateTo()` method of the `MapController` class.

The following Try It Out shows how you can programmatically animate Google Maps to a particular location.

TRY IT OUT Navigating the Map to Display a Specific Location

1. Using the project created in the previous section, add the following statements in bold to the `LBSActivity.java` file:

```
package net.learn2develop.LBS;

import com.google.android.maps.GeoPoint;
import com.google.android.maps.MapActivity;
import com.google.android.maps.MapController;
import com.google.android.maps.MapView;

import android.os.Bundle;
import android.view.KeyEvent;

public class LBSActivity extends MapActivity {
    MapView mapView;
    MapController mc;
    GeoPoint p;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        mapView = (MapView) findViewById(R.id.mapView);
        mapView.setBuiltInZoomControls(true);
        mapView.setSatellite(true);
        mapView.setTraffic(true);

        mc = mapView.getController();
        String coordinates[] = {"1.352566007", "103.78921587"};
        double lat = Double.parseDouble(coordinates[0]);
        double lng = Double.parseDouble(coordinates[1]);

        p = new GeoPoint(
            (int) (lat * 1E6),
            (int) (lng * 1E6));

        mc.animateTo(p);
        mc.setZoom(13);
        mapView.invalidate();
    }

    public boolean onKeyDown(int keyCode, KeyEvent event)
    {
        //...
    }

    @Override
    protected boolean isRouteDisplayed() {
        //...
    }
}
```

2. Press F11 to debug the application on the Android emulator. When the map is loaded, observe that it now animates to a particular location in Singapore (see Figure 9-9).

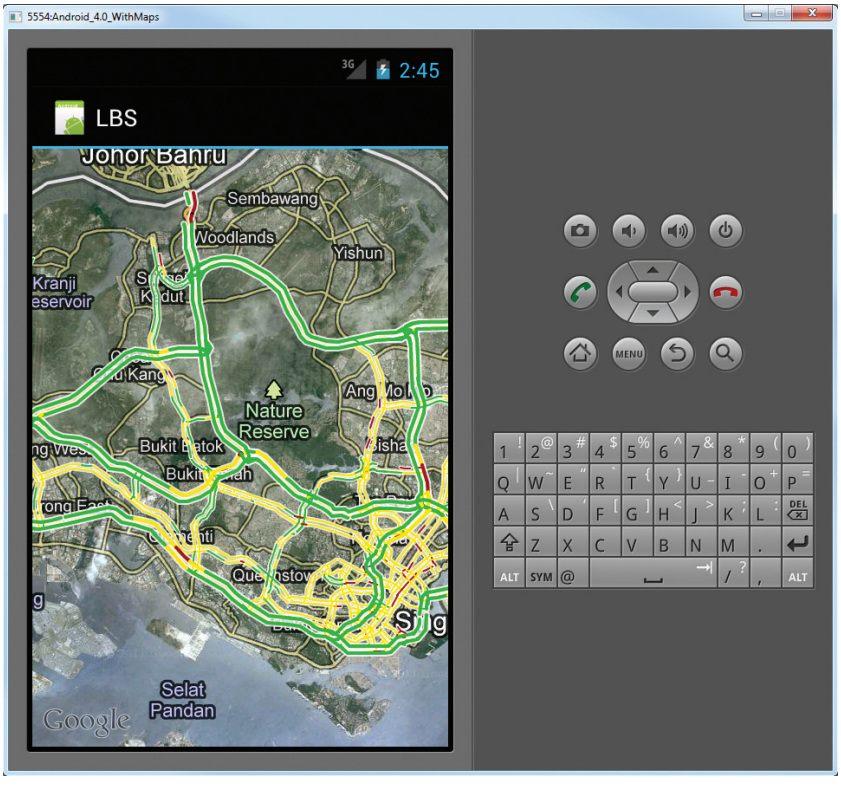

FIGURE 9-9

How It Works

In the preceding code, you first obtained a map controller from the `MapView` instance and assigned it to a `MapController` object (`mc`). You then used a `GeoPoint` object to represent a geographical location. Note that for this class, the latitude and longitude of a location are represented in micro degrees. This means that they are stored as integer values. For a latitude value of 40.747778, for example, you need to multiply it by 1e6 (which is one million) to obtain 40747778.

To navigate the map to a particular location, you can use the `animateTo()` method of the `MapController` class. The `setZoom()` method enables you to specify the zoom level at which the map is displayed (the bigger the number, the more details you see on the map). The `invalidate()` method forces the `MapView` to be redrawn.

Adding Markers

Adding markers to a map to indicate places of interest enables your users to easily locate the places they are looking for. The following Try It Out shows you how to add a marker to Google Maps.

TRY IT OUT Adding Markers to the Map

1. Create a GIF image containing a pushpin (see Figure 9-10) and copy it into the `res/drawable-mdpi` folder of the existing project. For the best effect, make the background of the image transparent so that it does not block parts of the map when the image is added to the map.

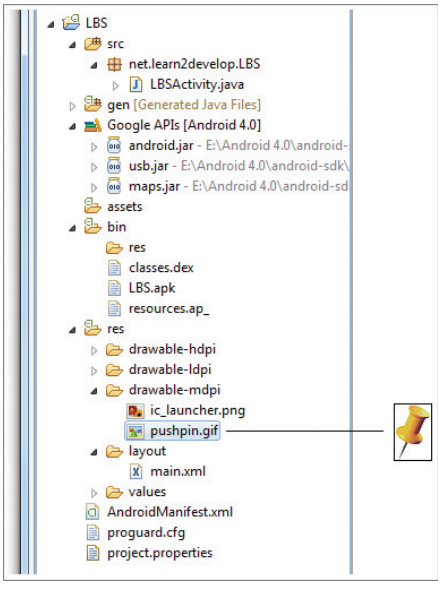

FIGURE 9-10

2. Using the project created in the previous activity, add the following statements in bold to the `LBSActivity.java` file:

```
package net.learn2develop.LBS;

import java.util.List;

import com.google.android.maps.GeoPoint;
import com.google.android.maps.MapActivity;
import com.google.android.maps.MapController;
import com.google.android.maps.MapView;
import com.google.android.maps.Overlay;

import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
import android.graphics.Point;
import android.os.Bundle;
import android.view.KeyEvent;

public class LBSActivity extends MapActivity {
    MapView mapView;
    MapController mc;
    GeoPoint p;

    private class MapOverlay extends com.google.android.maps.Overlay
    {
        @Override
        public boolean draw(Canvas canvas, MapView mapView,
        boolean shadow, long when)
        {
            super.draw(canvas, mapView, shadow);

            //---translate the GeoPoint to screen pixels---
            Point screenPts = new Point();
            mapView.getProjection().toPixels(p, screenPts);

            //---add the marker---
```

```
            Bitmap bmp = BitmapFactory.decodeResource(
                getResources(), R.drawable.pushpin);
            canvas.drawBitmap(bmp, screenPts.x, screenPts.y-50, null);
            return true;
        }
    }

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        mapView = (MapView) findViewById(R.id.mapView);
        mapView.setBuiltInZoomControls(true);
        mapView.setSatellite(true);
        mapView.setTraffic(true);

        mc = mapView.getController();
        String coordinates[] = {"1.352566007", "103.78921587"};
        double lat = Double.parseDouble(coordinates[0]);
        double lng = Double.parseDouble(coordinates[1]);

        p = new GeoPoint(
            (int) (lat * 1E6),
            (int) (lng * 1E6));

        mc.animateTo(p);
        mc.setZoom(13);

        //---Add a location marker---
        MapOverlay mapOverlay = new MapOverlay();
        List<Overlay> listOfOverlays = mapView.getOverlays();
        listOfOverlays.clear();
        listOfOverlays.add(mapOverlay);

        mapView.invalidate();
    }

    public boolean onKeyDown(int keyCode, KeyEvent event)
    {
        //...
    }

    @Override
    protected boolean isRouteDisplayed() {
        //...
    }
}
```

3. Press F11 to debug the application on the Android emulator. Figure 9-11 shows the marker added to the map.

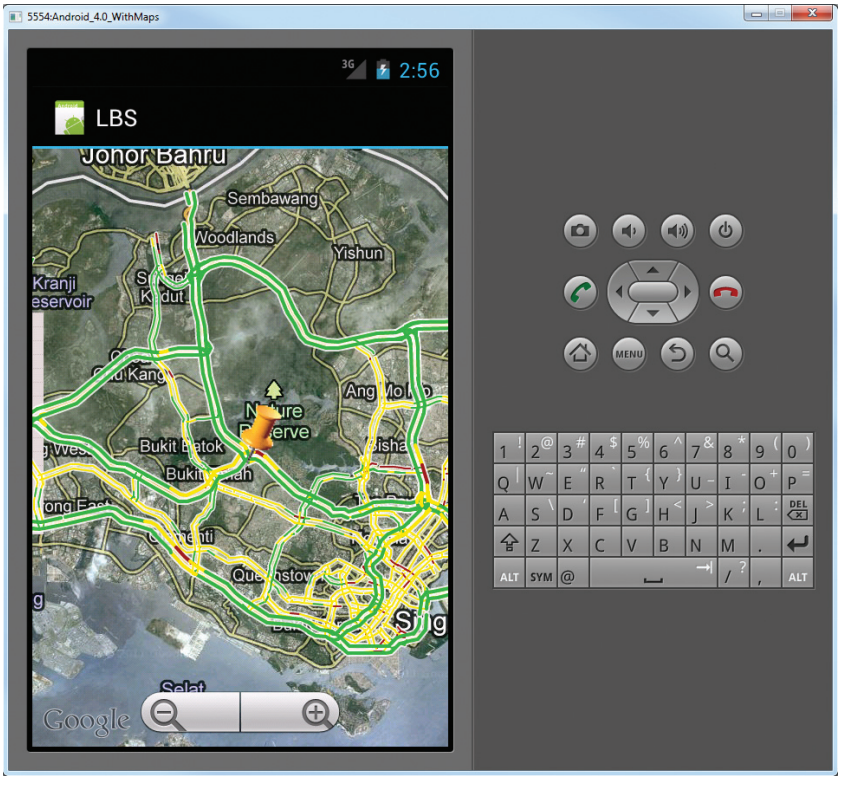

FIGURE 9-11

How It Works

To add a marker to the map, you first needed to define a class that extends the `Overlay` class:

```
private class MapOverlay extends com.google.android.maps.Overlay
    {
        @Override
        public boolean draw(Canvas canvas, MapView mapView,
        boolean shadow, long when)
        {
            //...
        }
    }
```

An overlay represents an individual item that you can draw on the map. You can add as many overlays as you want. In the `MapOverlay` class, you overrode the `draw()` method so that you could draw the pushpin image on the map. In particular, note that you needed to translate the geographical location (represented by a `GeoPoint` object, `p`) into screen coordinates:

```
//---translate the GeoPoint to screen pixels---
Point screenPts = new Point();
mapView.getProjection().toPixels(p, screenPts);
```

Because you want the pointed tip of the pushpin to indicate the position of the location, you need to deduct the height of the image (which is 50 pixels) from the *y* coordinate of the point (see Figure 9-12) and draw the image at that location:

```
//---add the marker---
Bitmap bmp = BitmapFactory.decodeResource(
    getResources(), R.drawable.pushpin);
canvas.drawBitmap(bmp, screenPts.x, screenPts.y-50, null);
```

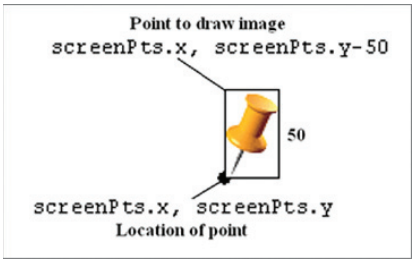

FIGURE 9-12

To add the marker, you created an instance of the `MapOverlay` class and added it to the list of overlays available on the `MapView` object:

```
//---Add a location marker---
MapOverlay mapOverlay = new MapOverlay();
List<Overlay> listOfOverlays = mapView.getOverlays();
listOfOverlays.clear();
listOfOverlays.add(mapOverlay);
```

Getting the Location That Was Touched

After using Google Maps for a while, you may want to know the latitude and longitude of a location corresponding to the position on the screen that was just touched. Knowing this information is very useful, as you can determine a location's address, a process known as *reverse geocoding* (you will learn how this is done in the next section).

If you have added an overlay to the map, you can override the `onTouchEvent()` method within the `MapOverlay` class. This method is fired every time the user touches the map. The method has two parameters: `MotionEvent` and `MapView`. Using the `MotionEvent` parameter, you can determine whether the user has lifted his or her finger from the screen using the `getAction()` method. In the following code snippet, if the user has touched and then lifted the finger, you display the latitude and longitude of the location touched:

```
package net.learn2develop.LBS;

import java.util.List;

import com.google.android.maps.GeoPoint;
```

```
import com.google.android.maps.MapActivity;
import com.google.android.maps.MapController;
import com.google.android.maps.MapView;
import com.google.android.maps.Overlay;

import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
import android.graphics.Point;
import android.os.Bundle;
import android.view.KeyEvent;
import android.view.MotionEvent;
import android.widget.Toast;

public class LBSActivity extends MapActivity {
    MapView mapView;
    MapController mc;
    GeoPoint p;

    private class MapOverlay extends com.google.android.maps.Overlay
    {
        @Override
        public boolean draw(Canvas canvas, MapView mapView,
        boolean shadow, long when)
        {
            //...
        }

        @Override
        public boolean onTouchEvent(MotionEvent event, MapView mapView)
        {
            //---when user lifts his finger---
            if (event.getAction() == 1) {
                GeoPoint p = mapView.getProjection().fromPixels(
                    (int) event.getX(),
                    (int) event.getY());
                    Toast.makeText(getBaseContext(),
                        "Location: "+
                        p.getLatitudeE6() / 1E6 + "," +
                        p.getLongitudeE6() /1E6 ,
                        Toast.LENGTH_SHORT).show();
            }
            return false;
        }
    }
    //...
}
```

The `getProjection()` method returns a projection for converting between screen-pixel coordinates and latitude/longitude coordinates. The `fromPixels()` method then converts the screen coordinates into a `GeoPoint` object.

Figure 9-13 shows the map displaying a set of coordinates when the user clicks a location on the map.

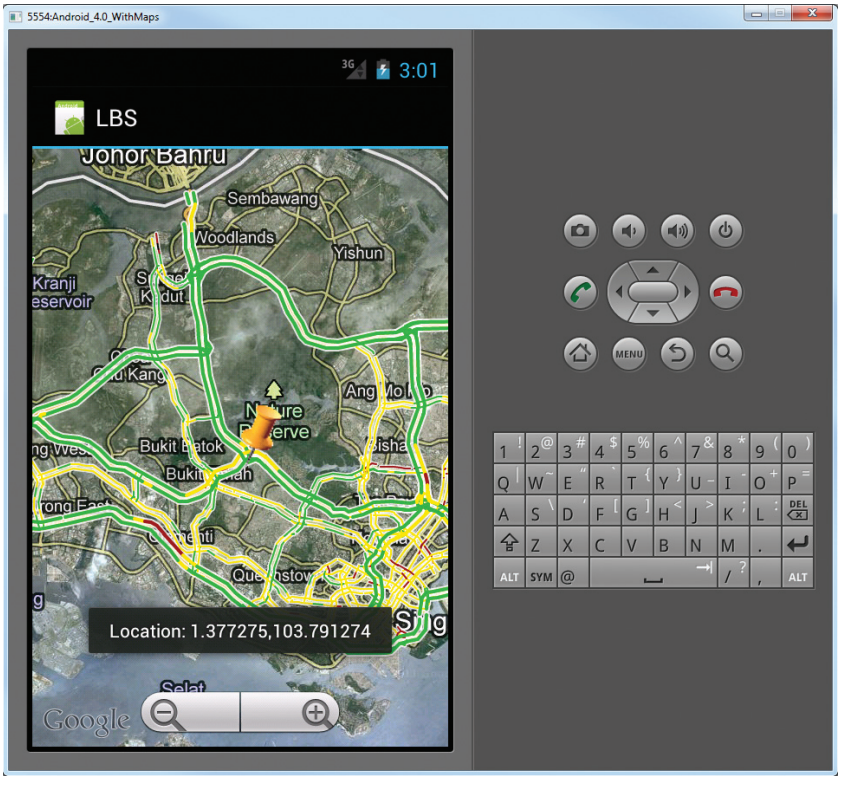

FIGURE 9-13

Geocoding and Reverse Geocoding

As mentioned in the preceding section, if you know the latitude and longitude of a location, you can find out its address using a process known as reverse geocoding. Google Maps in Android supports this via the `Geocoder` class. The following code snippet shows how you can retrieve the address of a location just touched using the `getFromLocation()` method:

```
package net.learn2develop.LBS;

import java.io.IOException;
import java.util.List;
import java.util.Locale;

import com.google.android.maps.GeoPoint;
import com.google.android.maps.MapActivity;
import com.google.android.maps.MapController;
import com.google.android.maps.MapView;
import com.google.android.maps.Overlay;

import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
```

```
import android.graphics.Point;
import android.location.Address;
import android.location.Geocoder;
import android.os.Bundle;
import android.view.KeyEvent;
import android.view.MotionEvent;
import android.widget.Toast;

public class LBSActivity extends MapActivity {
    MapView mapView;
    MapController mc;
    GeoPoint p;

    private class MapOverlay extends com.google.android.maps.Overlay
    {
        @Override
        public boolean draw(Canvas canvas, MapView mapView,
        boolean shadow, long when)
        {
            //...
        }

        @Override
        public boolean onTouchEvent(MotionEvent event, MapView mapView)
        {
            //---when user lifts his finger---
            if (event.getAction() == 1) {
                GeoPoint p = mapView.getProjection().fromPixels(
                        (int) event.getX(),
                        (int) event.getY());

                /*
                    Toast.makeText(getBaseContext(),
                        "Location: "+
                        p.getLatitudeE6() / 1E6 + "," +
                        p.getLongitudeE6() /1E6 ,
                        Toast.LENGTH_SHORT).show();
                 */

                Geocoder geoCoder = new Geocoder(
                        getBaseContext(), Locale.getDefault());
                try {
                    List<Address> addresses = geoCoder.getFromLocation(
                            p.getLatitudeE6()  / 1E6,
                            p.getLongitudeE6() / 1E6, 1);

                    String add = "";
                    if (addresses.size() > 0)
                    {
                        for (int i=0; i<addresses.get(0).getMaxAddressLineIndex();
                                i++)
                            add += addresses.get(0).getAddressLine(i) + "\n";
                    }
                    Toast.makeText(getBaseContext(), add, Toast.LENGTH_SHORT).show();
```

```
                    }
                    catch (IOException e) {
                        e.printStackTrace();
                    }
                    return true;
                }
                return false;
            }
        }
        //...
    }
```

The `Geocoder` object converts the latitude and longitude into an address using the `getFromLocation()` method. Once the address is obtained, you display it using the `Toast` class. Figure 9-14 shows the application displaying the address of a location that was touched on the map.

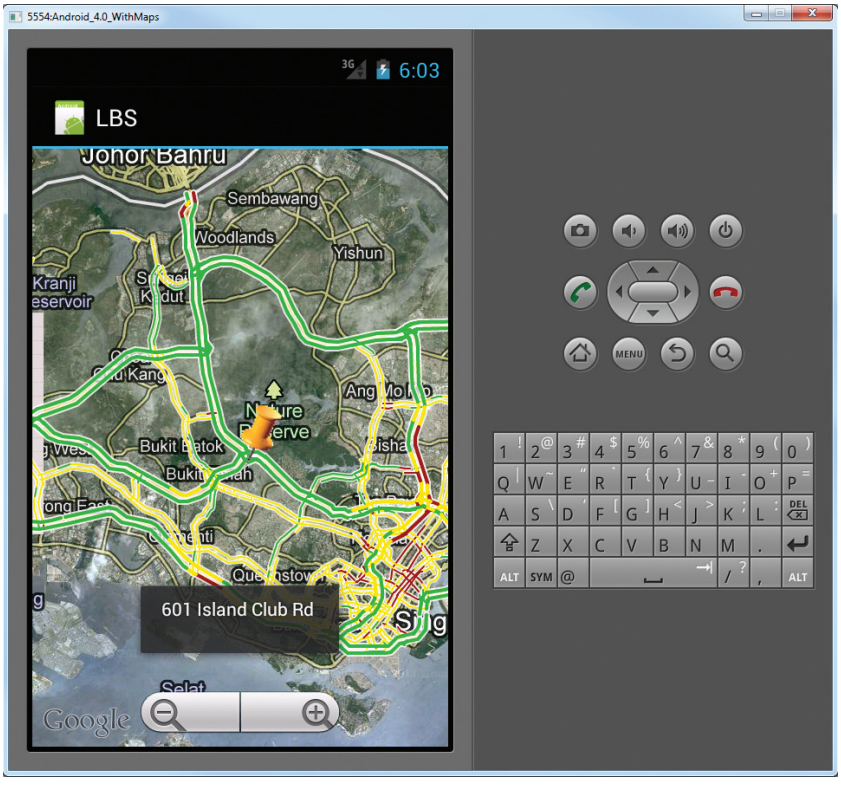

FIGURE 9-14

If you know the address of a location but want to know its latitude and longitude, you can do so via geocoding. Again, you can use the `Geocoder` class for this purpose. The following code shows how

you can find the exact location of the Empire State Building by using the `getFromLocationName()` method:

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    mapView = (MapView) findViewById(R.id.mapView);
    mapView.setBuiltInZoomControls(true);
    mapView.setSatellite(true);
    mapView.setTraffic(true);

    mc = mapView.getController();

    /*
    String coordinates[] = {"1.352566007", "103.78921587"};
    double lat = Double.parseDouble(coordinates[0]);
    double lng = Double.parseDouble(coordinates[1]);

    p = new GeoPoint(
        (int) (lat * 1E6),
        (int) (lng * 1E6));

    mc.animateTo(p);
    mc.setZoom(13);
    */

    //---geo-coding---
    Geocoder geoCoder = new Geocoder(this, Locale.getDefault());
    try {
        List<Address> addresses = geoCoder.getFromLocationName(
            "empire state building", 5);

        if (addresses.size() > 0) {
            p = new GeoPoint(
                    (int) (addresses.get(0).getLatitude() * 1E6),
                    (int) (addresses.get(0).getLongitude() * 1E6));
            mc.animateTo(p);
            mc.setZoom(20);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }

    //---Add a location marker---
    MapOverlay mapOverlay = new MapOverlay();
    List<Overlay> listOfOverlays = mapView.getOverlays();
    listOfOverlays.clear();
    listOfOverlays.add(mapOverlay);

    mapView.invalidate();
}
```

Figure 9-15 shows the map navigating to the location of the Empire State Building.

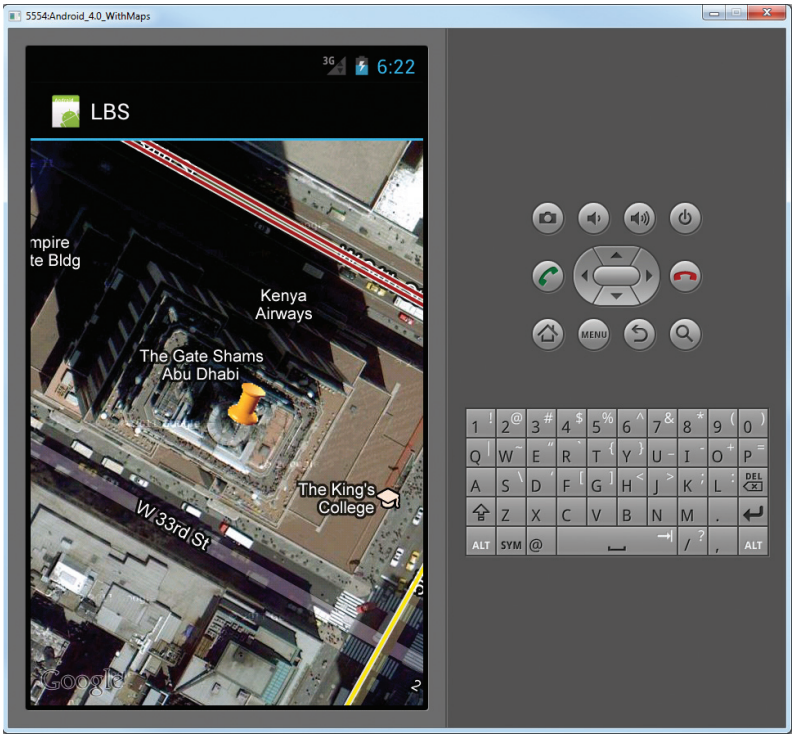

FIGURE 9-15

GETTING LOCATION DATA

Nowadays, mobile devices are commonly equipped with GPS receivers. Because of the many satellites orbiting the earth, you can use a GPS receiver to find your location easily. However, GPS requires a clear sky to work and hence does not always work indoors or where satellites can't penetrate (such as a tunnel through a mountain).

Another effective way to locate your position is through *cell tower triangulation*. When a mobile phone is switched on, it is constantly in contact with base stations surrounding it. By knowing the identity of cell towers, it is possible to translate this information into a physical location through the use of various databases containing the cell towers' identities and their exact geographical locations. The advantage of cell tower triangulation is that it works indoors, without the need to obtain information from satellites. However, it is not as precise as GPS because its accuracy depends on overlapping signal coverage, which varies quite a bit. Cell tower triangulation works best in densely populated areas where the cell towers are closely located.

A third method of locating your position is to rely on Wi-Fi triangulation. Rather than connect to cell towers, the device connects to a Wi-Fi network and checks the service provider against databases to determine the location serviced by the provider. Of the three methods described here, Wi-Fi triangulation is the least accurate.

On the Android platform, the SDK provides the `LocationManager` class to help your device determine the user's physical location. The following Try It Out shows how this is done in code.

TRY IT OUT Navigating the Map to a Specific Location

1. Using the same project created in the previous section, add the following statements in bold to the `LBSActivity.java` file:

```
package net.learn2develop.LBS;

import java.io.IOException;
import java.util.List;
import java.util.Locale;

import com.google.android.maps.GeoPoint;
import com.google.android.maps.MapActivity;
import com.google.android.maps.MapController;
import com.google.android.maps.MapView;
import com.google.android.maps.Overlay;

import android.content.Context;

import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
import android.graphics.Point;
import android.location.Address;
import android.location.Geocoder;

import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;

import android.os.Bundle;
import android.view.KeyEvent;
import android.view.MotionEvent;
import android.widget.Toast;

public class LBSActivity extends MapActivity {
    MapView mapView;
    MapController mc;
    GeoPoint p;

    LocationManager lm;
    LocationListener locationListener;

    private class MapOverlay extends com.google.android.maps.Overlay
    {
        //...
    }

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        mapView = (MapView) findViewById(R.id.mapView);
```

```
        mapView.setBuiltInZoomControls(true);
        mapView.setSatellite(true);
        mapView.setTraffic(true);

        mc = mapView.getController();

        /*
        String coordinates[] = {"1.352566007", "103.78921587"};
        double lat = Double.parseDouble(coordinates[0]);
        double lng = Double.parseDouble(coordinates[1]);

        p = new GeoPoint(
            (int) (lat * 1E6),
            (int) (lng * 1E6));

        mc.animateTo(p);
        mc.setZoom(13);
        */

        //---geo-coding---
        Geocoder geoCoder = new Geocoder(this, Locale.getDefault());
        try {
            //...
        }

        //---Add a location marker---
        MapOverlay mapOverlay = new MapOverlay();
        List<Overlay> listOfOverlays = mapView.getOverlays();
        listOfOverlays.clear();
        listOfOverlays.add(mapOverlay);

        mapView.invalidate();

        //---use the LocationManager class to obtain locations data---
        lm = (LocationManager)
            getSystemService(Context.LOCATION_SERVICE);

        locationListener = new MyLocationListener();
    }

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

        //---request for location updates---
        lm.requestLocationUpdates(
                LocationManager.GPS_PROVIDER,
                0,
                0,
                locationListener);
    }

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

        //---remove the location listener---
        lm.removeUpdates(locationListener);
```

```
    }

    private class MyLocationListener implements LocationListener
    {
        public void onLocationChanged(Location loc) {
            if (loc != null) {
                Toast.makeText(getBaseContext(),
                        "Location changed : Lat: " + loc.getLatitude() +
                        " Lng: " + loc.getLongitude(),
                        Toast.LENGTH_SHORT).show();

                p = new GeoPoint(
                        (int) (loc.getLatitude() * 1E6),
                        (int) (loc.getLongitude() * 1E6));

                mc.animateTo(p);
                mc.setZoom(18);
            }
        }

        public void onProviderDisabled(String provider) {
        }

        public void onProviderEnabled(String provider) {
        }

        public void onStatusChanged(String provider, int status,
            Bundle extras) {
        }
    }

    public boolean onKeyDown(int keyCode, KeyEvent event)
    {
        //...
    }

    @Override
    protected boolean isRouteDisplayed() {
        //...
    }
}
```

2. Add the following line in bold to the `AndroidManifest.xml` file:

```
<?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"/>

    <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>
```

3. Press F11 to debug the application on the Android emulator.

4. To simulate GPS data received by the Android emulator, you use the Location Controls tool (see Figure 9-16) located in the DDMS perspective of Eclipse.

5. Ensure that you have first selected the emulator in the Devices tab. Then, in the Emulator Control tab, locate the Location Controls tool and select the Manual tab. Enter a latitude and longitude and click the Send button.

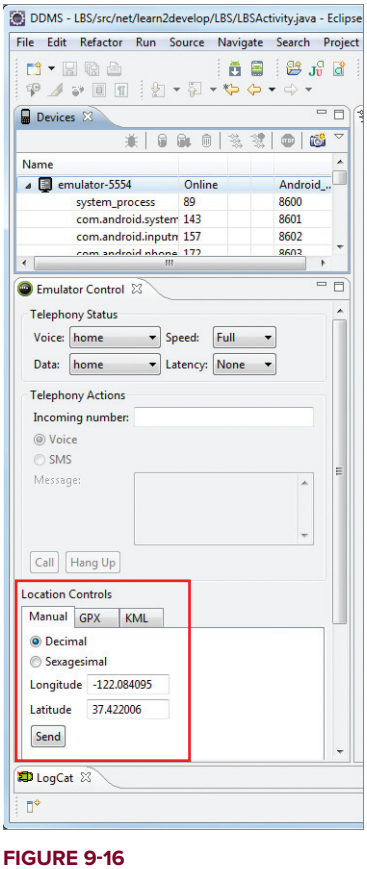

FIGURE 9-16

6. Observe that the map on the emulator now animates to another location (see Figure 9-17). This proves that the application has received the GPS data.

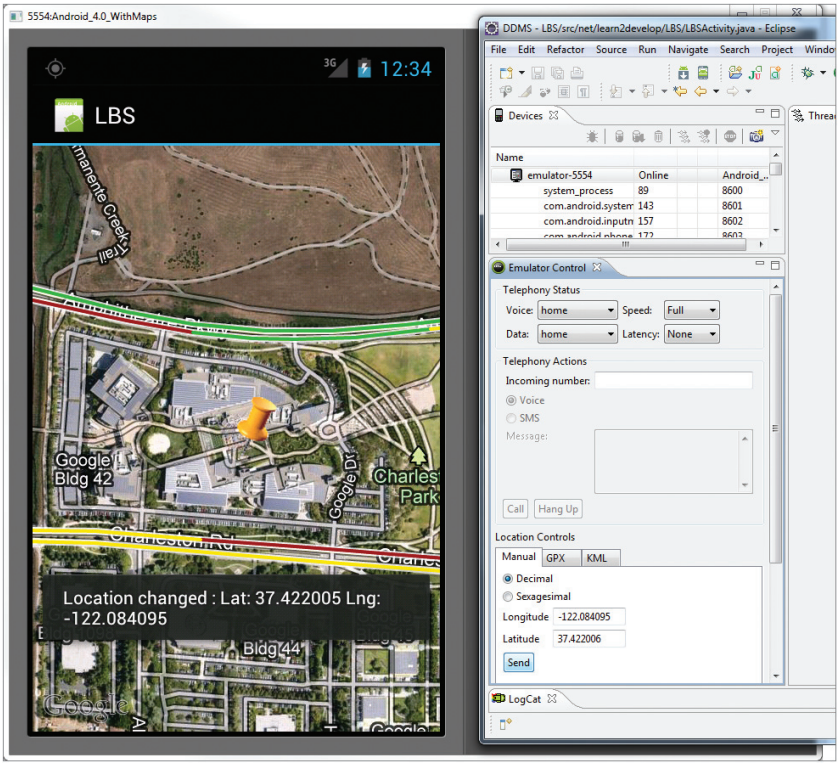

FIGURE 9-17

How It Works

In Android, location-based services are provided by the `LocationManager` class, located in the `android.location` package. Using the `LocationManager` class, your application can obtain periodic updates of the device's geographical locations, as well as fire an intent when it enters the proximity of a certain location.

In the `LBSActivity.java` file, you first obtained a reference to the `LocationManager` class using the `getSystemService()` method. You did this in the `onCreate()` method of the `LBSActivity`:

```
//---use the LocationManager class to obtain locations data---
lm = (LocationManager)
    getSystemService(Context.LOCATION_SERVICE);

locationListener = new MyLocationListener();
```

Next, you created an instance of the `MyLocationListener` class, which you defined next.

The `MyLocationListener` class implements the `LocationListener` abstract class. You need to override four methods in this implementation:

- `onLocationChanged(Location location)` — Called when the location has changed
- `onProviderDisabled(String provider)` — Called when the provider is disabled by the user
- `onProviderEnabled(String provider)` — Called when the provider is enabled by the user
- `onStatusChanged(String provider, int status, Bundle extras)` — Called when the provider status changes

In this example, you're more interested in what happens when a location changes, so you wrote some code in the `onLocationChanged()` method. Specifically, when a location changes, you display a small dialog on the screen showing the new location information: latitude and longitude. You show this dialog using the `Toast` class:

```
public void onLocationChanged(Location loc) {
    if (loc != null) {
        Toast.makeText(getBaseContext(),
                "Location changed : Lat: " + loc.getLatitude() +
                " Lng: " + loc.getLongitude(),
                Toast.LENGTH_SHORT).show();

        p = new GeoPoint(
                (int) (loc.getLatitude() * 1E6),
                (int) (loc.getLongitude() * 1E6));

        mc.animateTo(p);
        mc.setZoom(18);
    }
}
```

In the preceding method, you also navigate the map to the location that you have received.

To be notified whenever there is a change in location, you needed to register a request for location changes so that your program can be notified periodically. This is done via the `requestLocationUpdates()` method. You did this in the `onResume()` method of the activity:

```
@Override
public void onResume() {
    super.onResume();

    //---request for location updates---
    lm.requestLocationUpdates(
            LocationManager.GPS_PROVIDER,
            0,
            0,
            locationListener);
}
```

The `requestLocationUpdates()` method takes four arguments:

- `provider` — The name of the provider with which you register. In this case, you are using GPS to obtain your geographical location data.

- `minTime` — The minimum time interval for notifications, in milliseconds. 0 indicates that you want to be continually informed of location changes.
- `minDistance` — The minimum distance interval for notifications, in meters. 0 indicates that you want to be continually informed of location changes.
- `listener` — An object whose `onLocationChanged()` method will be called for each location update

Finally, in the `onPause()` method, you remove the listener when the activity is destroyed or goes into the background (so that the application no longer listens for changes in location, thereby saving the battery of the device). You did that using the `removeUpdates()` method:

```
@Override
public void onPause() {
    super.onPause();

    //---remove the location listener---
    lm.removeUpdates(locationListener);
}
```

If you want to use Cell-ID and Wi-Fi triangulation (important for indoor use) to obtain your location data, you can use the network provider, like this:

```
@Override
public void onResume() {
    super.onResume();

    //---request for location updates---
    lm.requestLocationUpdates(
            LocationManager.GPS_PROVIDER,
            0,
            0,
            locationListener);
}
```

To use the network provider, you need to add the `ACCESS_COARSE_LOCATION` permission to the `AndroidManifest.xml` file:

```
<?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>
```

NOTE *The network provider will not work on the Android emulator. If you test the preceding code on the emulator, it will result in an illegal argument exception. You need to test the code on a real device.*

You can combine both the GPS location provider with the network location provider within your application:

```
@Override
public void onResume() {
    super.onResume();

    //---request for location updates---
    lm.requestLocationUpdates(
            LocationManager.GPS_PROVIDER,
            0,
            0,
            locationListener);

    //---request for location updates---
    lm.requestLocationUpdates(
            LocationManager.NETWORK_PROVIDER,
            0,
            0,
            locationListener);
}
```

However, be aware that doing so will cause your application to receive two different sets of coordinates, as both the GPS provider and the NETWORK provider will try to get your location using their own methods (GPS versus Wi-Fi and Cell ID triangulation). Hence, it is important that you monitor the status of the two providers in your device and use the appropriate one. You can check the status of the two providers by implementing the following three methods (shown in bold) of the `MyLocationListener` class:

```
private class MyLocationListener implements LocationListener
{
    @Override
    public void onLocationChanged(Location loc) {
        if (loc != null) {
            Toast.makeText(getBaseContext(),
                    "Location changed : Lat: " + loc.getLatitude() +
                    " Lng: " + loc.getLongitude(),
```

```
                    Toast.LENGTH_SHORT).show();


            p = new GeoPoint(
                    (int) (loc.getLatitude() * 1E6),
                    (int) (loc.getLongitude() * 1E6));

            mc.animateTo(p);
            mc.setZoom(18);
        }
    }

    //---called when the provider is disabled---
    public void onProviderDisabled(String provider) {
        Toast.makeText(getBaseContext(),
                provider + " disabled",
                Toast.LENGTH_SHORT).show();
    }

    //---called when the provider is enabled---
    public void onProviderEnabled(String provider) {
        Toast.makeText(getBaseContext(),
                provider + " enabled",
                Toast.LENGTH_SHORT).show();
    }

    //---called when there is a change in the provider status---
    public void onStatusChanged(String provider, int status,
        Bundle extras) {
        String statusString = "";
        switch (status) {
            case android.location.LocationProvider.AVAILABLE:
                statusString = "available";
            case android.location.LocationProvider.OUT_OF_SERVICE:
                statusString = "out of service";
            case android.location.LocationProvider.TEMPORARILY_UNAVAILABLE:
                statusString = "temporarily unavailable";
        }

        Toast.makeText(getBaseContext(),
                provider + " " + statusString,
                Toast.LENGTH_SHORT).show();
    }
}
```

MONITORING A LOCATION

One very cool feature of the `LocationManager` class is its ability to monitor a specific location. This is achieved using the `addProximityAlert()` method.

The following code snippet shows how to monitor a particular location such that if the user is within a five-meter radius from that location, your application will fire an intent to launch the web browser:

```
import android.app.PendingIntent;
import android.content.Intent;
import android.net.Uri;

        //---use the LocationManager class to obtain locations data---
        lm = (LocationManager)
            getSystemService(Context.LOCATION_SERVICE);

        //---PendingIntent to launch activity if the user is within
        // some locations---
        PendingIntent pendingIntent = PendingIntent.getActivity(
            this, 0, new
            Intent(android.content.Intent.ACTION_VIEW,
              Uri.parse("http://www.amazon.com")), 0);

        lm.addProximityAlert(37.422006, -122.084095, 5, -1, pendingIntent);
```

The `addProximityAlert()` method takes five arguments: latitude, longitude, radius (in meters), expiration (duration for which the proximity alert is valid, after which it is deleted; -1 for no expiration), and the pending intent.

Note that if the Android device's screen goes to sleep, the proximity is also checked once every four minutes in order to preserve the battery life of the device.

PROJECT — BUILDING A LOCATION TRACKER

Now that you have seen how to build a location-based Android application, you can put that knowledge to good use by combining the techniques covered in this chapter with the techniques covered in Chapter 8, "Messaging," to build a cool working application. You will build a location tracker application that can be installed on a user's Android device. You can then send an SMS message containing a specific code to the user's device, and it will automatically return the location of the device through a return SMS message. This type of location tracker application could be used to keep track of your child or any elderly relative who lives alone (and it can be done without the person's knowledge).

WARNING *Before you roll out this application to your users, note that in some countries it is illegal to track the location of a person without his or her knowledge. If you install the location tracker application on a user's phone, that device will automatically return its location information to whomever sends it an SMS message beginning with the words "Where are you?" Therefore, if you want to use this project in real life, you must alert potential users about the application's functionality, so that they have the option to not reveal their location.*

TRY IT OUT Invoking an Activity

1. Using Eclipse, create a new Android project and name it **LocationTracker**.
2. Add the following lines in bold to the `AndroidManifest.xml` file:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="net.learn2develop.LocationTracker"
      android:versionCode="1"
      android:versionName="1.0">
    <uses-sdk android:minSdkVersion="14" />

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

    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".LocationTrackerActivity"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <!-- put this here so that even if the app is not running,
        your app can be woken up when there is an incoming SMS message -->
        <receiver android:name=".SMSReceiver">
            <intent-filter android:priority="100">
                <action
                    android:name="android.provider.Telephony.SMS_RECEIVED" />
            </intent-filter>
        </receiver>
    </application>
</manifest>
```

3. Add a new Java class to the package name of the project and name it `SMSReceiver`. You should now have a Java file named `SMSReceiver.java` under the package name of your project (see Figure 9-18).

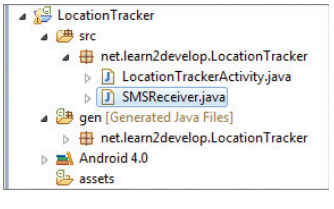

FIGURE 9-18

4. Populate the `SMSReceiver.java` file with the following lines in bold:

```
package net.learn2develop.LocationTracker;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.telephony.SmsManager;
import android.telephony.SmsMessage;

public class SMSReceiver extends BroadcastReceiver
```

```
{
    LocationManager lm;
    LocationListener locationListener;
    String senderTel;

    @Override
    public void onReceive(Context context, Intent intent)
    {
        //---get the SMS message that was received---
        Bundle bundle = intent.getExtras();
        SmsMessage[] msgs = null;
        String str="";
        if (bundle != null)
        {
            senderTel = "";
            //---retrieve the SMS message received---
            Object[] pdus = (Object[]) bundle.get("pdus");
            msgs = new SmsMessage[pdus.length];
            for (int i=0; i<msgs.length; i++){
                msgs[i] = SmsMessage.createFromPdu((byte[])pdus[i]);
                if (i==0) {
                    //---get the sender address/phone number---
                    senderTel = msgs[i].getOriginatingAddress();
                }
                //---get the message body---
                str += msgs[i].getMessageBody().toString();
            }

            if (str.startsWith("Where are you?")) {
                //---use the LocationManager class to obtain locations data---
                lm = (LocationManager)
                        context.getSystemService(Context.LOCATION_SERVICE);

                //---request location updates---
                locationListener = new MyLocationListener();
                lm.requestLocationUpdates(
                        LocationManager.NETWORK_PROVIDER,
                        60000,
                        1000,
                        locationListener);

                //---abort the broadcast; SMS messages won't be broadcasted---
                this.abortBroadcast();
            }
        }
    }

    private class MyLocationListener implements LocationListener
    {
        @
        public void onLocationChanged(Location loc) {
            if (loc != null) {
                //---send a SMS containing the current location---
                SmsManager sms = SmsManager.getDefault();
                sms.sendTextMessage(senderTel, null,
                        "http://maps.google.com/maps?q=" + loc.getLatitude() + "," +
```

```
                                    loc.getLongitude(), null, null);

                        //---stop listening for location changes---
                        lm.removeUpdates(locationListener);
                    }
                }

                public void onProviderDisabled(String provider) {
                }

                public void onProviderEnabled(String provider) {
                }

                public void onStatusChanged(String provider, int status,
                        Bundle extras) {
                }
            }
        }
```

5. To test the application, first deploy it onto a real Android device. Then, use another phone (any type of mobile phone that can send SMS messages) to send an SMS message to it with the message "Where are you?"

6. After the SMS message has been sent, wait for the Android device to send back an SMS message. Figure 9-19 shows the reply from the Android device, containing its location (shown here using an iPhone).

7. Most smartphones recognize URL data in the SMS message; therefore, if you click the URL in the SMS message, you will see the location through Google Maps (see Figure 9-20).

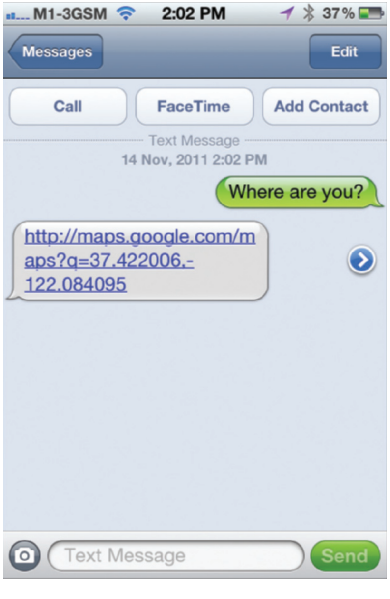

FIGURE 9-19

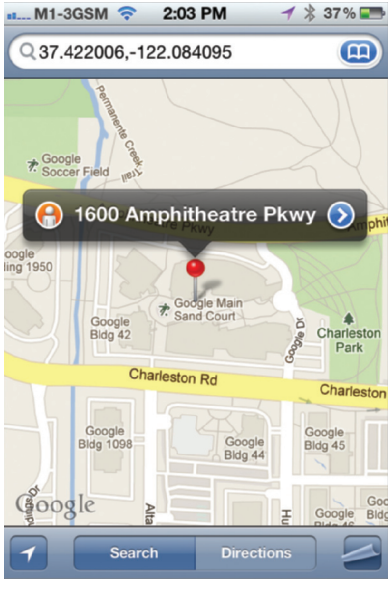

FIGURE 9-20

How It Works

This project combined the concepts that you learned in this chapter with the material about SMS messaging covered in Chapter 8 into one complete application. Once installed, the application will listen for incoming SMS messages that contain the text "Where are you?" It then intercepts these SMS messages, so the user won't be able to see the message in the Messaging application on the Android device.

When an SMS message is received, you first extract the phone number of the sender so that you can use it later to send a reply containing the location of the device:

```
//---retrieve the SMS message received---
senderTel = "";

//---retrieve the SMS message received---
Object[] pdus = (Object[]) bundle.get("pdus");
msgs = new SmsMessage[pdus.length];
for (int i=0; i<msgs.length; i++){
    msgs[i] = SmsMessage.createFromPdu((byte[])pdus[i]);
    if (i==0) {
        //---get the sender address/phone number---
        senderTel = msgs[i].getOriginatingAddress();
    }
    //---get the message body---
    str += msgs[i].getMessageBody().toString();
}
```

You then examine the content of the SMS message. If it starts with the sentence "Where are you?", you then request for location updates using the `LocationManager` class:

```
if (str.startsWith("Where are you?")) {
    //---use the LocationManager class to obtain locations data---
    lm = (LocationManager)
            context.getSystemService(Context.LOCATION_SERVICE);

    //---request location updates---
    locationListener = new MyLocationListener();
    lm.requestLocationUpdates(
            LocationManager.NETWORK_PROVIDER,
            60000,
            1000,
            locationListener);

    this.abortBroadcast();
}
```

Note that in this example, I have used the network provider to obtain my location because it does not require a line of sight to the sky (which is required by the GPS provider). However, using the network provider does require the device to have an Internet connection, so your application will not work if the device lacks one.

Once the location is obtained, you send a returning SMS message containing the location of the device, using a URL that points to Google Maps:

```
public void onLocationChanged(Location loc) {
    if (loc != null) {
```

```
            //---send a SMS containing the current location---
            SmsManager sms = SmsManager.getDefault();
            sms.sendTextMessage(senderTel, null,
                    "http://maps.google.com/maps?q=" + loc.getLatitude() + "," +
                            loc.getLongitude(), null, null);

            //---stop listening for location changes---
            lm.removeUpdates(locationListener);
        }
    }
```

Once the SMS message is sent, you immediately remove the location updates so that you do not listen for location changes anymore.

Note that I have used 60,000 ms and 1,000 meters for the `minTime` and `minDistance` arguments of the `requestLocationUpdates()` method, respectively. If you recall from the earlier part of this chapter, I used 0 for both arguments, because I wanted to be continually informed of location changes. However, in this project you should not do this, as it would cause the application to continuously send several SMS messages back to the sender at once. This is because by the time you stop listening for location updates, the Location Manager would have called the `onLocationChanged()` method several times, reporting the device's smallest changes in location and resulting in multiple SMS messages. In real testing, the number of SMS messages ranges from 10 to 60. Thus, you should set the `minTime` and `minDistance` arguments to a more reasonable value so that the Location Manager does not have a chance to fire the `onLocationChanged()` method repeatedly.

SUMMARY

This chapter took a whirlwind tour of the `MapView` object, which displays Google Maps in your Android application. You have learned the various ways in which the map can be manipulated, and you have also seen how you can obtain geographical location data using the various network providers: GPS, Cell-ID, or Wi-Fi triangulation. Finally, you learned to build a Location Tracker application that enables you to track a user's location using SMS messaging.

EXERCISES

1. If you have embedded the Google Maps API into your Android application but it does not show the map when the application is loaded, what could be the likely reasons?
2. What is the difference between geocoding and reverse geocoding?
3. Name the two location providers that you can use to obtain your location data.
4. What method is used for monitoring a location?

Answers to the exercises can be found in Appendix C.

▶ WHAT YOU LEARNED IN THIS CHAPTER

TOPIC	KEY CONCEPTS

Displaying the MapView	`<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_MAPS_API_KEY" />`

Referencing the Map library	`<uses-library android:name="com.google.android.maps" />`
Displaying the zoom controls	`mapView.setBuiltInZoomControls(true);`
Programmatically zooming in or out of the map	`mc.zoomIn();` `mc.zoomOut();`
Changing views	`mapView.setSatellite(true);` `mapView.setTraffic(true);`

Animating to a particular location	`mc = mapView.getController();`
	`String coordinates[] = {"1.352566007", "103.78921587"};`
	`double lat = Double.parseDouble(coordinates[0]);`
	`double lng = Double.parseDouble(coordinates[1]);`
	`p = new GeoPoint(`
	`(int) (lat * 1E6),`
	`(int) (lng * 1E6));`
	`mc.animateTo(p);`

Adding markers	`Implement an Overlay class and override the draw() method`

Getting the location of the map touched	`GeoPoint p = mapView.getProjection().fromPixels(`
	`(int) event.getX(),`
	`(int) event.getY());`

Geocoding and reverse geocoding	`Use the Geocoder class`

continues

(continued)

TOPIC	KEY CONCEPTS

Obtaining location data	`private LocationManager lm;`
	
	`//...`
	
	`lm = (LocationManager)`
	`getSystemService(Context.LOCATION_SERVICE);`
	
	`locationListener = new MyLocationListener();`
	
	`lm.requestLocationUpdates(`
	`LocationManager.GPS_PROVIDER,`
	`0,`
	`0,`
	`locationListener);`
	
	`//...`
	
	`private class MyLocationListener implements LocationListener`
	`{`
	
	`public void onLocationChanged(Location loc) {`
	`if (loc != null) {`
	`}`
	`}`
	
	`public void onProviderDisabled(String provider) {`
	`}`
	
	`public void onProviderEnabled(String provider) {`
	`}`
	
	`public void onStatusChanged(String provider, int status,`
	`Bundle extras) {`
	`}`
	`}`

Monitoring a location	`lm.addProximityAlert(37.422006, -122.084095, 5, -1,` `pendingIntent);`

10

Networking

WHAT YOU WILL LEARN IN THIS CHAPTER

- How to connect to the web using HTTP
- How to consume XML web services
- How to consume JSON web services
- How to connect to a Socket server

In Chapter 8, you learned about how your application can talk to the outside world through the use of SMS messaging and e-mails. Another way to communicate with the outside world is through the wireless network available on your Android device. Therefore, in this chapter you will learn how to use the HTTP protocol to talk to web servers so that you can download text and binary data. You will also learn how to parse XML files to extract the relevant parts of an XML document — a technique that is useful if you are accessing web services. Besides XML web services, this chapter also covers JSON (JavaScript Object Notation), which is a lightweight alternative to XML. You will make use of the classes available in the Android SDK to manipulate JSON content.

Finally, this chapter also demonstrates how to write an Android application to connect to servers using TCP sockets. Using sockets programming, you can write sophisticated, interesting networked applications.

CONSUMING WEB SERVICES USING HTTP

One common way to communicate with the outside world is through HTTP. HTTP is no stranger to most people; it is the protocol that drives much of the web's success. Using the HTTP protocol, you can perform a wide variety of tasks, such as downloading web pages from a web server, downloading binary data, and more.

The following Try It Out creates an Android project so you can use the HTTP protocol to connect to the web to download all sorts of content.

TRY IT OUT Creating the Base Project for HTTP Connection

codefile Networking.zip available for download at Wrox.com

1. Using Eclipse, create a new Android project and name it **Networking**.
2. Add the following statement in bold to the `AndroidManifest.xml` file:

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

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

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:label="@string/app_name"
            android:name=".NetworkingActivity" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

3. Import the following packages in the `NetworkingActivity.java` file:

```
package net.learn2develop.Networking;

import android.app.Activity;
import android.os.Bundle;
import java.io.IOException;
import java.io.InputStream;
import java.net.HttpURLConnection;
import java.net.URL;
import java.net.URLConnection;
import android.util.Log;
public class NetworkingActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

4. Define the `OpenHttpConnection()` method in the `NetworkingActivity.java` file:

```
public class NetworkingActivity extends Activity {
    private InputStream OpenHttpConnection(String urlString) throws IOException
    {
        InputStream in = null;
        int response = -1;

        URL url = new URL(urlString);
        URLConnection conn = url.openConnection();

        if (!(conn instanceof HttpURLConnection))
            throw new IOException("Not an HTTP connection");
        try{
            HttpURLConnection httpConn = (HttpURLConnection) conn;
            httpConn.setAllowUserInteraction(false);
            httpConn.setInstanceFollowRedirects(true);
            httpConn.setRequestMethod("GET");
            httpConn.connect();
            response = httpConn.getResponseCode();
            if (response == HttpURLConnection.HTTP_OK) {
                in = httpConn.getInputStream();
            }
        }
        catch (Exception ex)
        {
            Log.d("Networking", ex.getLocalizedMessage());
            throw new IOException("Error connecting");
        }
        return in;
    }

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

How It Works

Because you are using the HTTP protocol to connect to the web, your application needs the `INTERNET` permission; hence, the first thing you did was add the permission in the `AndroidManifest.xml` file.

You then defined the `OpenHttpConnection()` method, which takes a URL string and returns an `InputStream` object. Using an `InputStream` object, you can download the data by reading bytes from the stream object. In this method, you made use of the `HttpURLConnection` object to open an HTTP connection with a remote URL. You set all the various properties of the connection, such as the request method, and so on:

```
HttpURLConnection httpConn = (HttpURLConnection) conn;
httpConn.setAllowUserInteraction(false);
httpConn.setInstanceFollowRedirects(true);
httpConn.setRequestMethod("GET");
```

After trying to establish a connection with the server, the HTTP response code is returned. If the connection is established (via the response code `HTTP_OK`), then you proceed to get an `InputStream` object from the connection:

```
httpConn.connect();
response = httpConn.getResponseCode();
if (response == HttpURLConnection.HTTP_OK) {
    in = httpConn.getInputStream();
}
```

Using the `InputStream` object, you can then start to download the data from the server.

Downloading Binary Data

A common task you need to perform is downloading binary data from the web. For example, you may want to download an image from a server so that you can display it in your application. The following Try It Out shows how this is done.

TRY IT OUT Downloading Binary Data

1. Using the same project created earlier, replace the default `TextView` with the following statements in bold to 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" >

<ImageView
    android:id="@+id/img"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center" />

</LinearLayout>
```

2. Add the following statements in bold to the `NetworkingActivity.java` file:

```
import android.widget.ImageView;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.os.AsyncTask;

public class NetworkingActivity extends Activity {
    ImageView img;

    private InputStream OpenHttpConnection(String urlString) throws IOException
```

```
{
    InputStream in = null;
    int response = -1;

    URL url = new URL(urlString);
    URLConnection conn = url.openConnection();

    if (!(conn instanceof HttpURLConnection))
        throw new IOException("Not an HTTP connection");
    try{
        HttpURLConnection httpConn = (HttpURLConnection) conn;
        httpConn.setAllowUserInteraction(false);
        httpConn.setInstanceFollowRedirects(true);
        httpConn.setRequestMethod("GET");
        httpConn.connect();
        response = httpConn.getResponseCode();
        if (response == HttpURLConnection.HTTP_OK) {
            in = httpConn.getInputStream();
        }
    }
    catch (Exception ex)
    {
        Log.d("Networking", ex.getLocalizedMessage());
        throw new IOException("Error connecting");
    }
    return in;
}

private Bitmap DownloadImage(String URL)
{
    Bitmap bitmap = null;
    InputStream in = null;
    try {
        in = OpenHttpConnection(URL);
        bitmap = BitmapFactory.decodeStream(in);
        in.close();
    } catch (IOException e1) {
        Log.d("NetworkingActivity", e1.getLocalizedMessage());
    }
    return bitmap;
}

private class DownloadImageTask extends AsyncTask<String, Void, Bitmap> {
    protected Bitmap doInBackground(String... urls) {
        return DownloadImage(urls[0]);
    }

    protected void onPostExecute(Bitmap result) {
        ImageView img = (ImageView) findViewById(R.id.img);
        img.setImageBitmap(result);
    }
}

/** Called when the activity is first created. */
```

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        new DownloadImageTask().execute(
            "http://www.mayoff.com/5-01cablecarDCP01934.jpg");
    }
}
```

3. Press F11 to debug the application on the Android emulator. Figure 10-1 shows the image downloaded from the web and then displayed in the `ImageView`.

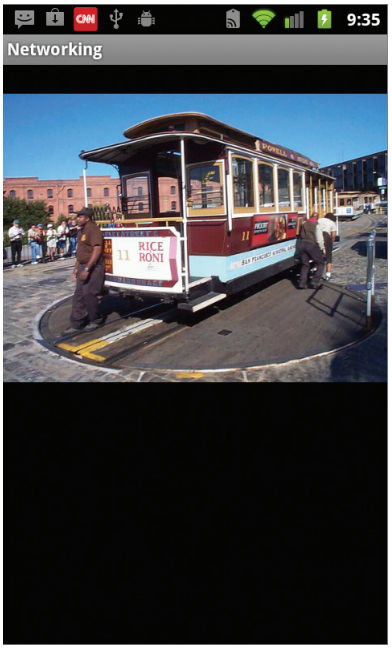

FIGURE 10-1

How It Works

The `DownloadImage()` method takes the URL of the image to download and then opens the connection to the server using the `OpenHttpConnection()` method that you have defined earlier. Using the `InputStream` object returned by the connection, the `decodeStream()` method from the `BitmapFactory` class is used to download and decode the data into a `Bitmap` object. The `DownloadImage()` method returns a `Bitmap` object.

To download an image and display it on the activity, you call the `DownloadImage()` method. However, starting with Android 3.0, synchronous operations can no longer be run directly from a UI thread. If you try to call the `DownloadImage()` method directly in your `onCreate()` method (as shown in the following code snippet), your application will crash when it is run on a device running Android 3.0 and later:

```
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    //---download an image---
    //---code will not run in Android 3.0 and beyond---
    Bitmap bitmap =
        DownloadImage("http://www.mayoff.com/5-01cablecarDCP01934.jpg");
    img = (ImageView) findViewById(R.id.img);
    img.setImageBitmap(bitmap);
}
```

Because the `DownloadImage()` method is synchronous — that is, it will not return control until the image is downloaded — calling it directly will freeze the UI of your activity. This is not allowed in Android 3.0 and later; all synchronous code must be wrapped using an `AsyncTask` class. Using `AsyncTask` enables you to perform background tasks in a separate thread and then return the result in a UI thread. That way, you can perform background operations without needing to handle complex threading issues.

To call the `DownloadImage()` method asynchronously, you need to wrap the code in a subclass of the `AsyncTask` class, as shown here:

```
private class DownloadImageTask extends AsyncTask<String, Void, Bitmap> {
    protected Bitmap doInBackground(String... urls) {
        return DownloadImage(urls[0]);
    }
    protected void onPostExecute(Bitmap result) {
        ImageView img = (ImageView) findViewById(R.id.img);
        img.setImageBitmap(result);
    }
}
```

Basically, you defined a class (`DownloadImageTask`) that extends the `AsyncTask` class. In this case, there are two methods within the `DownloadImageTask` class: `doInBackground()` and `onPostExecute()`.

You put all the code that needs to be run asynchronously in the `doInBackground()` method. When the task is completed, the result is passed back via the `onPostExecute()` method. In this case, you use the `ImageView` to display the image downloaded.

RUNNING SYNCHRONOUS OPERATIONS IN A UI THREAD

To be specific, if you set the `android:minSdkVersion` attribute in your `AndroidManifest.xml` file to a value of 9 or less and then run your application on an Android 3.0 or later device, your synchronous code will still work in a UI thread (though not recommended). However, if the `android:minSdkVersion` attribute value is set to 10 or above, your synchronous code will not work in a UI thread.

NOTE *Chapter 11 discusses the* `AsyncTask` *class in more detail.*

To call the `DownloadImageTask` class, create an instance of it and then call its `execute()` method, passing it the URL of the image to download:

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    new DownloadImageTask().execute(
        "http://www.mayoff.com/5-01cablecarDCP01934.jpg");
}
```

If you want to download a series of images asynchronously, you can modify the `DownloadImageTask` class as follows:

```
…
import android.widget.Toast;
…
    private class DownloadImageTask extends AsyncTask
    <String, Bitmap, Long> {
        //---takes in a list of image URLs in String type---
        protected Long doInBackground(String... urls) {
            long imagesCount = 0;
            for (int i = 0; i < urls.length; i++) {
                //---download the image---
                Bitmap imageDownloaded = DownloadImage(urls[i]);
                if (imageDownloaded != null)  {
                    //---increment the image count---
                    imagesCount++;
                    try {
                        //---insert a delay of 3 seconds---
                        Thread.sleep(3000);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                    //---return the image downloaded---
                    publishProgress(imageDownloaded);
                }
            }
            //---return the total images downloaded count---
            return imagesCount;
        }

        //---display the image downloaded---
        protected void onProgressUpdate(Bitmap... bitmap) {
            img.setImageBitmap(bitmap[0]);
        }

        //---when all the images have been downloaded---
        protected void onPostExecute(Long imagesDownloaded) {
            Toast.makeText(getBaseContext(),
                  "Total " + imagesDownloaded + " images downloaded" ,
                  Toast.LENGTH_LONG).show();
        }
    }
```

Note that in this example, the `DownloadImageTask` class has one more method: `onProgressUpdate()`. Because the task to be performed inside an `AsyncTask` class can be lengthy, you call the `publishProgress()` method to update the progress of the operation. This will trigger the `onProgressUpdate()` method, which in this case displays the image to be downloaded. The `onProgressUpdate()` method is executed on the UI thread; hence it is thread-safe to update the `ImageView` with the bitmap downloaded from the server.

To download a series of images asynchronously in the background, create an instance of the `BackgroundTask` class and call its `execute()` method, like this:

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    /* new DownloadImageTask().execute(
         "http://www.mayoff.com/5-01cablecarDCP01934.jpg");
    */

    img = (ImageView) findViewById(R.id.img);
    new DownloadImageTask().execute(
            "http://www.mayoff.com/5-01cablecarDCP01934.jpg",
            "http://www.hartiesinfo.net/greybox/Cable_Car_
               Hartbeespoort.jpg",
            "http://mcmanuslab.ucsf.edu/sites/default/files/
               imagepicker/m/mmcmanus/
               CaliforniaSanFranciscoPaintedLadiesHz.jpg",
            "http://www.fantom-xp.com/wallpapers/63/San_Francisco
               _-_Sunset.jpg",
            "http://travel.roro44.com/europe/france/
               Paris_France.jpg",
            "http://wwp.greenwichmeantime.com/time-zone/usa/nevada
                /las-vegas/hotel/the-strip/paris-las-vegas/paris-
                las-vegas-hotel.jpg",
            "http://designheaven.files.wordpress.com/2010/04/
                eiffel_tower_paris_france.jpg");
}
```

When you run the preceding code, the images are downloaded in the background and displayed at an interval of three seconds. When the last image has been downloaded, the `Toast` class displays the total number of images downloaded.

REFERRING TO LOCALHOST FROM YOUR EMULATOR

When working with the Android emulator, you may frequently need to access data hosted on the local web server using `localhost`. For example, your own web services are likely to be hosted on your local computer during development, and you'll want to test them on the same development machine you use to write your Android applications. In such cases, you should use the special IP address of 10.0.2.2 (not 127.0.0.1) to refer to the host computer's loopback interface. From the Android emulator's perspective, `localhost` (127.0.0.1) refers to its own loopback interface.

Downloading Text Content

Besides downloading binary data, you can also download plain-text content. For example, you might want to access a web service that returns a string of random quotes. The following Try It Out shows how you can download a string from a web service in your application.

TRY IT OUT Downloading Plain-Text Content

1. Using the same project created earlier, add the following statements in bold to the `NetworkingActivity.java` file:

```
import java.io.InputStreamReader;

    private String DownloadText(String URL)
    {
        int BUFFER_SIZE = 2000;
        InputStream in = null;
        try {
            in = OpenHttpConnection(URL);
        } catch (IOException e) {
            Log.d("Networking", e.getLocalizedMessage());
            return "";
        }

        InputStreamReader isr = new InputStreamReader(in);
        int charRead;
        String str = "";
        char[] inputBuffer = new char[BUFFER_SIZE];
        try {
            while ((charRead = isr.read(inputBuffer))>0) {
                //---convert the chars to a String---
                String readString =
                    String.copyValueOf(inputBuffer, 0, charRead);
                str += readString;
                inputBuffer = new char[BUFFER_SIZE];
            }
            in.close();
        } catch (IOException e) {
            Log.d("Networking", e.getLocalizedMessage());
            return "";
        }
        return str;
    }

    private class DownloadTextTask extends AsyncTask<String, Void, String> {
        protected String doInBackground(String... urls) {
            return DownloadText(urls[0]);
        }

        @Override
        protected void onPostExecute(String result) {
            Toast.makeText(getBaseContext(), result, Toast.LENGTH_LONG).show();
        }
```

```
}

/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    //---download text---
    new DownloadTextTask().execute(
       "http://iheartquotes.com/api/v1/random?max_characters=256&max_lines=10");
}
```

2. Press F11 to debug the application on the Android emulator. Figure 10-2 shows the random quote downloaded and displayed using the `Toast` class.

How It Works

The `DownloadText()` method takes the URL of the text file to download and then returns the string of the text file downloaded. It basically opens an HTTP connection to the server and then uses an `InputStreamReader` object to read each character from the stream and save it in a `String` object. As shown in the previous section, you had to create a subclass of the `AsyncTask` class in order to call the `DownloadText()` method asynchronously.

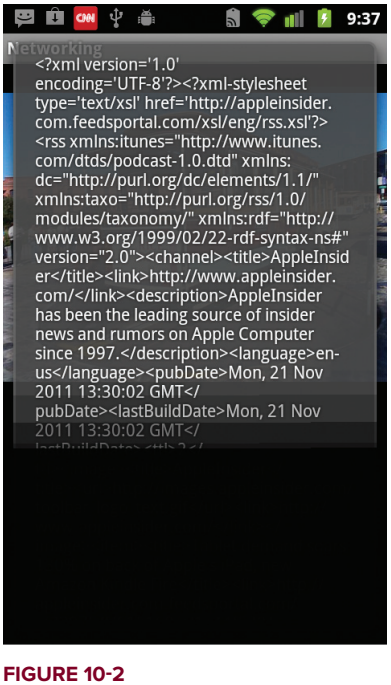

FIGURE 10-2

Accessing Web Services Using the GET Method

So far, you have learned how to download images and text from the web. The previous section showed how to download some plain text from a server. Very often, you need to download XML files and parse the contents (a good example of this is consuming web services). Therefore, in this section you learn how to connect to a web service using the HTTP `GET` method. Once the web service returns a result in XML, you extract the relevant parts and display its content using the `Toast` class.

In this example, you use the web method from `http://services.aonaware.com/DictService/DictService.asmx?op=Define`. This web method is from a dictionary web service that returns the definition of a given word.

The web method takes a request in the following format:

```
GET /DictService/DictService.asmx/Define?word=string HTTP/1.1
Host: services.aonaware.com
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length
```

It returns a response in the following format:

```
<?xml version="1.0" encoding="utf-8"?>
<WordDefinition xmlns="http://services.aonaware.com/webservices/">
  <Word>string</Word>
  <Definitions>
    <Definition>
      <Word>string</Word>
      <Dictionary>
        <Id>string</Id>
        <Name>string</Name>
      </Dictionary>
      <WordDefinition>string</WordDefinition>
    </Definition>
    <Definition>
      <Word>string</Word>
      <Dictionary>
        <Id>string</Id>
        <Name>string</Name>
      </Dictionary>
      <WordDefinition>string</WordDefinition>
    </Definition>
  </Definitions>
</WordDefinition>
```

Hence, to obtain the definition of a word, you need to establish an HTTP connection to the web method and then parse the XML result that is returned. The following Try It Out shows you how.

TRY IT OUT Consuming Web Services

1. Using the same project created earlier, add the following statements in bold to the `NetworkingActivity.java` file:

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

    private String WordDefinition(String word) {
        InputStream in = null;
        String strDefinition = "";
        try {
            in = OpenHttpConnection(
"http://services.aonaware.com/DictService/DictService.asmx/Define?word=" + word);
            Document doc = null;
            DocumentBuilderFactory dbf =
                DocumentBuilderFactory.newInstance();
            DocumentBuilder db;
            try {
                db = dbf.newDocumentBuilder();
                doc = db.parse(in);
            } catch (ParserConfigurationException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            doc.getDocumentElement().normalize();

            //---retrieve all the <Definition> elements---
            NodeList definitionElements =
                doc.getElementsByTagName("Definition");

            //---iterate through each <Definition> elements---
            for (int i = 0; i < definitionElements.getLength(); i++) {
                Node itemNode = definitionElements.item(i);
                if (itemNode.getNodeType() == Node.ELEMENT_NODE)
                {
                    //---convert the Definition node into an Element---
                    Element definitionElement = (Element) itemNode;

                    //---get all the <WordDefinition> elements under
                    // the <Definition> element---
                    NodeList wordDefinitionElements =
                        (definitionElement).getElementsByTagName(
```

```
                    "WordDefinition");

                strDefinition = "";
                //---iterate through each <WordDefinition> elements---
                for (int j = 0; j < wordDefinitionElements.getLength(); j++) {
                    //---convert a <WordDefinition> node into an Element---
                    Element wordDefinitionElement =
                        (Element) wordDefinitionElements.item(j);

                    //---get all the child nodes under the
                    // <WordDefinition> element---
                    NodeList textNodes =
                        ((Node) wordDefinitionElement).getChildNodes();

                    strDefinition +=
                        ((Node) textNodes.item(0)).getNodeValue() + ". \n";
                }

            }
        }
    } catch (IOException e1) {
        Log.d("NetworkingActivity", e1.getLocalizedMessage());
    }
    //---return the definitions of the word---
    return strDefinition;
}

private class AccessWebServiceTask extends AsyncTask<String, Void, String> {
    protected String doInBackground(String... urls) {
        return WordDefinition(urls[0]);
    }

    protected void onPostExecute(String result) {
        Toast.makeText(getBaseContext(), result, Toast.LENGTH_LONG).show();
    }
}

/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    //---access a Web Service using GET---
    new AccessWebServiceTask().execute("apple");
}
```

2. Press F11 to debug the application on the Android emulator. Figure 10-3 shows the result of the web service call being parsed and then displayed using the `Toast` class.

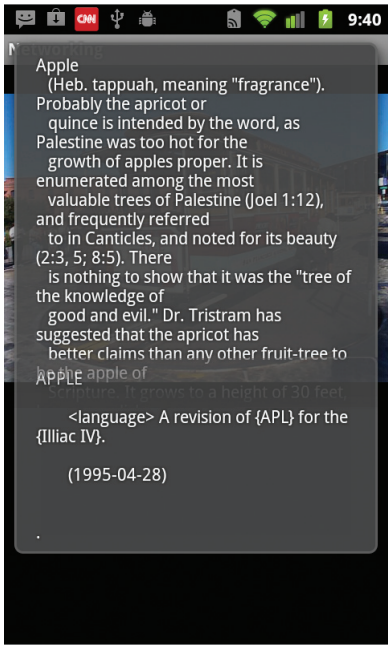

FIGURE 10-3

How It Works

The `WordDefinition()` method first opens an HTTP connection to the web service, passing in the word that you are interested in:

```
            in = OpenHttpConnection(
"http://services.aonaware.com/DictService/DictService.asmx/Define?word=" + word);
```

It then uses the `DocumentBuilderFactory` and `DocumentBuilder` objects to obtain a `Document` (DOM) object from an XML file (which is the XML result returned by the web service):

```
Document doc = null;
DocumentBuilderFactory dbf =
    DocumentBuilderFactory.newInstance();
DocumentBuilder db;
try {
    db = dbf.newDocumentBuilder();
    doc = db.parse(in);
} catch (ParserConfigurationException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
} catch (Exception e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
doc.getDocumentElement().normalize();
```

Once the `Document` object is obtained, you find all the elements with the `<Definition>` tag:

```
//---retrieve all the <Definition> elements---
NodeList definitionElements =
    doc.getElementsByTagName("Definition");
```

Figure 10-4 shows the structure of the XML document returned by the web service.

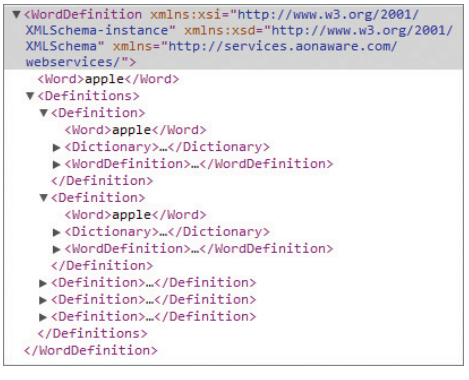

FIGURE 10-4

Because the definition of a word is contained within the `<WordDefinition>` element, you then proceed to extract all the definitions:

```
//---iterate through each <Definition> elements---
for (int i = 0; i < definitionElements.getLength(); i++) {
    Node itemNode = definitionElements.item(i);
    if (itemNode.getNodeType() == Node.ELEMENT_NODE)
    {
        //---convert the Definition node into an Element---
        Element definitionElement = (Element) itemNode;

        //---get all the <WordDefinition> elements under
        // the <Definition> element---
        NodeList wordDefinitionElements =
            (definitionElement).getElementsByTagName(
            "WordDefinition");

        strDefinition = "";
        //---iterate through each <WordDefinition> elements---
        for (int j = 0; j < wordDefinitionElements.getLength(); j++) {
            //---convert a <WordDefinition> node into an Element---
            Element wordDefinitionElement =
                (Element) wordDefinitionElements.item(j);

            //---get all the child nodes under the
            // <WordDefinition> element---
            NodeList textNodes =
                ((Node) wordDefinitionElement).getChildNodes();

            strDefinition +=
```

```
                        ((Node) textNodes.item(0)).getNodeValue() + ". \n";
                }               }
        }
```

The preceding code loops through all the `<Definition>` elements looking for a child element named `<WordDefinition>`. The text content of the `<WordDefinition>` element contains the definition of a word, and the definitions of a word are then concatenated and returned by the `WordDefinition()` method:

```
        //---return the definitions of the word---
        return strDefinition;
```

As usual, you need to create a subclass of the `AsyncTask` class to call the `WordDefinition()` method asynchronously:

```
private class AccessWebServiceTask extends AsyncTask<String, Void, String> {
    protected String doInBackground(String... urls) {
        return WordDefinition(urls[0]);
    }

    protected void onPostExecute(String result) {
        Toast.makeText(getBaseContext(), result, Toast.LENGTH_LONG).show();
    }
}
```

Finally, you access the web service asynchronously using the `execute()` method:

```
        //---access a Web Service using GET---
        new AccessWebServiceTask().execute("apple");
```

CONSUMING JSON SERVICES

In the previous section, you learned how to consume XML web services by using HTTP to connect to the web server and then obtain the results in XML. You also learned how to use DOM to parse the result of the XML document. However, manipulating XML documents is a computationally expensive operation for mobile devices, for the following reasons:

- XML documents are lengthy. They use tags to embed information, and the size of an XML document can get very big pretty quickly. A large XML document means that your device has to use more bandwidth to download it, which translates into higher cost.
- XML documents are more difficult to process. As shown earlier, you have to use DOM to traverse the tree in order to locate the information you want. In addition, DOM itself has to build the entire document in memory as a tree structure before you can traverse it. This is both memory and CPU intensive.

A much more efficient way to represent information exists in the form of JSON (JavaScript Object Notation). JSON is a lightweight data-interchange format that is easy for humans to read and write.

It is also easy for machines to parse and generate. The following lines of code show what a JSON message looks like:

```
[
    {
        "appeId":"1",
        "survId":"1",
        "location":"",
        "surveyDate":"2008-03 14",
        "surveyTime":"12:19:47",
        "inputUserId":"1",
        "inputTime":"2008-03-14 12:21:51",
        "modifyTime":"0000-00-00 00:00:00"
    },
    {
        "appeId":"2",
        "survId":"32",
        "location":"",
        "surveyDate":"2008-03-14",
        "surveyTime":"22:43:09",
        "inputUserId":"32",
        "inputTime":"2008-03-14 22:43:37",
        "modifyTime":"0000-00-00 00:00:00"
    },
    {
        "appeId":"3",
        "survId":"32",
        "location":"",
        "surveyDate":"2008-03-15",
        "surveyTime":"07:59:33",
        "inputUserId":"32",
        "inputTime":"2008-03-15 08:00:44",
        "modifyTime":"0000-00-00 00:00:00"
    },
    {
        "appeId":"4",
        "survId":"1",
        "location":"",
        "surveyDate":"2008-03-15",
        "surveyTime":"10:45:42",
        "inputUserId":"1",
        "inputTime":"2008-03-15 10:46:04",
        "modifyTime":"0000-00-00 00:00:00"
    },
    {
        "appeId":"5",
        "survId":"32",
        "location":"",
        "surveyDate":"2008-03-16",
        "surveyTime":"08:04:49",
        "inputUserId":"32",
        "inputTime":"2008-03-16 08:05:26",
        "modifyTime":"0000-00-00 00:00:00"
    },
```

```
    {
        "appeId":"6",
        "survId":"32",
        "location":"",
        "surveyDate":"2008-03-20",
        "surveyTime":"20:19:01",
        "inputUserId":"32",
        "inputTime":"2008-03-20 20:19:32",
        "modifyTime":"0000-00-00 00:00:00"
    }
]
```

The preceding block of lines represents a set of data taken for a survey. Note that the information is represented as a collection of key/value pairs; and each key/value pair is grouped into an ordered list of objects. Unlike XML, there are no lengthy tag names, only brackets and braces.

The following Try It Out demonstrates how to process JSON messages easily using the `JSONArray` and `JSONObject` classes available in the Android SDK.

TRY IT OUT Consuming JSON Services

1. Using Eclipse, create a new Android project and name it **JSON**.
2. Add the following line in bold to the `AndroidManifest.xml` file:

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

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

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:label="@string/app_name"
            android:name=".JSONActivity" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

3. Add the following lines of code in bold to the `JSONActivity.java` file:

```
package net.learn2develop.JSON;

import java.io.BufferedReader;
import java.io.IOException;
```

```
import java.io.InputStream;
import java.io.InputStreamReader;

import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.StatusLine;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.DefaultHttpClient;
import org.json.JSONArray;
import org.json.JSONObject;

import android.app.Activity;
import android.os.AsyncTask;
import android.os.Bundle;
import android.util.Log;
import android.widget.Toast;

public class JSONActivity extends Activity  {

    public String readJSONFeed(String URL) {
        StringBuilder stringBuilder = new StringBuilder();
        HttpClient client = new DefaultHttpClient();
        HttpGet httpGet = new HttpGet(URL);
        try {
            HttpResponse response = client.execute(httpGet);
            StatusLine statusLine = response.getStatusLine();
            int statusCode = statusLine.getStatusCode();
            if (statusCode == 200) {
                HttpEntity entity = response.getEntity();
                InputStream content = entity.getContent();
                BufferedReader reader = new BufferedReader(
                        new InputStreamReader(content));
                String line;
                while ((line = reader.readLine()) != null) {
                    stringBuilder.append(line);
                }
            } else {
                Log.e("JSON", "Failed to download file");
            }
        } catch (ClientProtocolException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return stringBuilder.toString();
    }

    private class ReadJSONFeedTask extends AsyncTask<String, Void, String> {
        protected String doInBackground(String... urls) {
            return readJSONFeed(urls[0]);
        }

        protected void onPostExecute(String result) {
            try {
```

```
                JSONArray jsonArray = new JSONArray(result);
                Log.i("JSON", "Number of surveys in feed: " +
                        jsonArray.length());

                //---print out the content of the json feed---
                for (int i = 0; i < jsonArray.length(); i++) {
                    JSONObject jsonObject = jsonArray.getJSONObject(i);
                    Toast.makeText(getBaseContext(), jsonObject.getString("appeId") +
                            " - " + jsonObject.getString("inputTime"),
                             Toast.LENGTH_SHORT).show();
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        new ReadJSONFeedTask().execute(
            "http://extjs.org.cn/extjs/examples/grid/survey.html");
    }
}
```

4. Press F11 to debug the application on the Android emulator. You will see the `Toast` class appear a couple of times, displaying the information (see Figure 10-5).

FIGURE 10-5

How It Works

The first thing you did in this project was define the `readJSONFeed()` method:

```
public String readJSONFeed(String URL) {
    StringBuilder stringBuilder = new StringBuilder();
    HttpClient client = new DefaultHttpClient();
    HttpGet httpGet = new HttpGet(URL);
    try {
        HttpResponse response = client.execute(httpGet);
        StatusLine statusLine = response.getStatusLine();
        int statusCode = statusLine.getStatusCode();
        if (statusCode == 200) {
            HttpEntity entity = response.getEntity();
            InputStream content = entity.getContent();
            BufferedReader reader = new BufferedReader(
                    new InputStreamReader(content));
            String line;
            while ((line = reader.readLine()) != null) {
                stringBuilder.append(line);
            }
        } else {
            Log.e("JSON", "Failed to download file");
        }
    } catch (ClientProtocolException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return stringBuilder.toString();
}
```

This method simply connects to the specified URL and then reads the response from the web server. It returns a string as the result.

To call the `readJSONFeed()` method asynchronously, you created a subclass of the `AsyncTask` class:

```
private class ReadJSONFeedTask extends AsyncTask<String, Void, String> {
    protected String doInBackground(String... urls) {
        return readJSONFeed(urls[0]);
    }

    protected void onPostExecute(String result) {
        try {
            JSONArray jsonArray = new JSONArray(result);
            Log.i("JSON", "Number of surveys in feed: " +
                    jsonArray.length());

            //---print out the content of the json feed---
            for (int i = 0; i < jsonArray.length(); i++) {
                JSONObject jsonObject = jsonArray.getJSONObject(i);
                Toast.makeText(getBaseContext(), jsonObject.getString("appeId") +
                        " - " + jsonObject.getString("inputTime"),
                        Toast.LENGTH_SHORT).show();
            }
        } catch (Exception e) {
```

```
                e.printStackTrace();
            }
        }
    }
```

You called the `readJSONFeed()` method in the `doInBackground()` method, and the JSON string that you have fetched is passed in through the `onPostExecute()` method. The JSON string used in this example (and as illustrated earlier) is from `http://extjs.org.cn/extjs/examples/grid/survey.html`.

To obtain the list of objects in the JSON string, you used the `JSONArray` class, passing it the JSON feed as the constructor for the class:

```
JSONArray jsonArray = new JSONArray(result);
Log.i("JSON", "Number of surveys in feed: " +
        jsonArray.length());
```

The `length()` method returns the number of objects in the `jsonArray` object. With the list of objects stored in the `jsonArray` object, you iterated through it to obtain each object using the `getJSONObject()` method:

```
//---print out the content of the json feed---
for (int i = 0; i < jsonArray.length(); i++) {
    JSONObject jsonObject = jsonArray.getJSONObject(i);

    Toast.makeText(this, jsonObject.getString("appeId") +
            " - " + jsonObject.getString("inputTime"),
            Toast.LENGTH_SHORT).show();
}
```

The `getJSONObject()` method returns an object of type `JSONObject`. To obtain the value of the key/value pair stored inside the object, you used the `getString()` method (you can also use the `getInt()`, `getLong()`, and `getBoolean()` methods for other data types).

Finally, you accessed the JSON feed asynchronously using the `execute()` method:

```
new ReadJSONFeedTask().execute(
    "http://extjs.org.cn/extjs/examples/grid/survey.html");
```

This example showed how you can consume a JSON service and quickly parse its result. A much more interesting example is to use a real-life scenario: Twitter. The following changes make the application fetch my latest tweets from Twitter and then display the tweets in the `Toast` class (see Figure 10-6):

```
private class ReadJSONFeedTask extends AsyncTask<String, Void, String> {
    protected String doInBackground(String... urls) {
        return readJSONFeed(urls[0]);
    }

    protected void onPostExecute(String result) {
        try {
            JSONArray jsonArray = new JSONArray(result);
            Log.i("JSON", "Number of surveys in feed: " +
                    jsonArray.length());

            //---print out the content of the json feed---
```

```
            for (int i = 0; i < jsonArray.length(); i++) {
                JSONObject jsonObject = jsonArray.getJSONObject(i);
                /*
                 Toast.makeText(getBaseContext(), jsonObject.getString("appeId") +
                         " - " + jsonObject.getString("inputTime"),
                         Toast.LENGTH_SHORT).show();
                 */

                Toast.makeText(getBaseContext(), jsonObject.getString("text") +
                        " - " + jsonObject.getString("created_at"),
                        Toast.LENGTH_SHORT).show();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    /*
     new ReadJSONFeedTask().execute(
          "http://extjs.org.cn/extjs/examples/grid/survey.html");
     */
    new ReadJSONFeedTask().execute(
        "https://twitter.com/statuses/user_timeline/weimenglee.json");
}
```

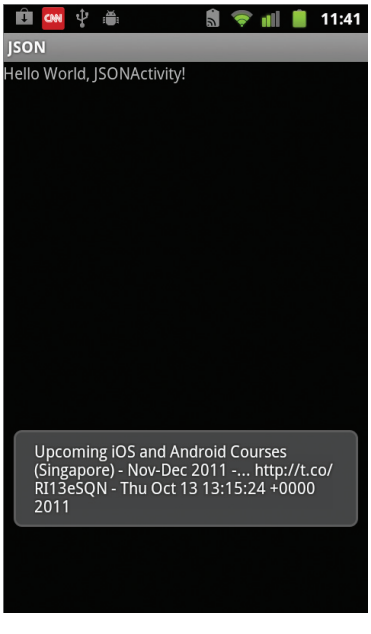

FIGURE 10-6

SOCKETS PROGRAMMING

So far, you have seen the use of HTTP to consume XML and JSON web services. While most web services use HTTP for communication, they inherently suffer from one huge disadvantage: They are stateless. When you connect to a web service using HTTP, every connection is treated as a new connection — the web server does not maintain a persistent connection with the clients.

Consider a scenario in which your application connects to a web service that books cinema seats. When seats are booked on the server by a client, all the other clients are not aware of this until they connect to the web service again to obtain the new seat allocations. This constant polling by the clients incurs unnecessary bandwidth and makes your application inefficient. A much better solution is to have the server maintain individual connections to each client, and send a message to each one whenever seats are booked by another client.

If you want your application to maintain a persistent connection to the server and be notified by the server whenever changes occur, you need to use a programming technique known as *sockets programming*. Sockets programming is a technique through which you use to establish a connection between a client and a server. The following Try It Out demonstrates how you can build an Android chat client application that connects to a socket server. Multiple applications can connect to the server and chat at the same time.

TRY IT OUT Connecting to a Socket Server

1. Using Eclipse, create a new Android project and name it **Sockets**.
2. Add the following line in bold to the `AndroidManifest.xml` file:

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

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

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:label="@string/app_name"
            android:name=".SocketsActivity" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

3. Add the following lines in bold to the `main.xml` file, replacing the `TextView`:

```
<?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" >

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

<Button
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Send Message"
    android:onClick="onClickSend"/>

<TextView
    android:id="@+id/txtMessagesReceived"
    android:layout_width="fill_parent"
    android:layout_height="200dp"
    android:scrollbars = "vertical" />

</LinearLayout>
```

4. Add a new Java Class file to the package and name it `CommsThread`. Populate the `CommsThread.java` file as follows:

```
package net.learn2develop.Sockets;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.Socket;
import android.util.Log;

public class CommsThread extends Thread {
    private final Socket socket;
    private final InputStream inputStream;
    private final OutputStream outputStream;

    public CommsThread(Socket sock) {
        socket = sock;
        InputStream tmpIn = null;
        OutputStream tmpOut = null;
        try {
            //---creates the inputstream and outputstream objects
            // for reading and writing through the sockets---
            tmpIn = socket.getInputStream();
            tmpOut = socket.getOutputStream();
        } catch (IOException e) {
            Log.d("SocketChat", e.getLocalizedMessage());
        }
```

```
            inputStream = tmpIn;
            outputStream = tmpOut;
        }

        public void run() {
            //---buffer store for the stream---
            byte[] buffer = new byte[1024];

            //---bytes returned from read()---
            int bytes;

            //---keep listening to the InputStream until an
            // exception occurs---
            while (true) {
                try {
                    //---read from the inputStream---
                    bytes = inputStream.read(buffer);

                    //---update the main activity UI---
                    SocketsActivity.UIupdater.obtainMessage(
                        0,bytes, -1, buffer).sendToTarget();
                } catch (IOException e) {
                    break;
                }
            }
        }

        //---call this from the main activity to
        // send data to the remote device---
        public void write(byte[] bytes) {
            try {
                outputStream.write(bytes);
            } catch (IOException e) { }
        }

        //---call this from the main activity to
        // shutdown the connection---
        public void cancel() {
            try {
                socket.close();
            } catch (IOException e) { }
        }
    }
```

5. In the `SocketsActivity.java` file, add the following lines in bold:

```
package net.learn2develop.Sockets;

import java.io.IOException;
import java.net.InetAddress;
import java.net.Socket;
import java.net.UnknownHostException;

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

```
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;

import android.util.Log;

public class SocketsActivity extends Activity {
    static final String NICKNAME = "Wei-Meng";
    InetAddress serverAddress;
    Socket socket;

    //---all the Views---
    static TextView txtMessagesReceived;
    EditText txtMessage;

    //---thread for communicating on the socket---
    CommsThread commsThread;

    //---used for updating the UI on the main activity---
    static Handler UIupdater = new Handler() {
        @Override
        public void handleMessage(Message msg) {
            int numOfBytesReceived = msg.arg1;
            byte[] buffer = (byte[]) msg.obj;

            //---convert the entire byte array to string---
            String strReceived = new String(buffer);

            //---extract only the actual string received---
            strReceived = strReceived.substring(
                    0, numOfBytesReceived);

            //---display the text received on the TextView---
            txtMessagesReceived.setText(
                    txtMessagesReceived.getText().toString() +
                    strReceived);
        }
    };

    private class CreateCommThreadTask extends AsyncTask
    <Void, Integer, Void> {
        @Override
        protected Void doInBackground(Void... params) {
            try {
                //---create a socket---
                serverAddress =
                    InetAddress.getByName("192.168.1.142");
                    //--remember to change the IP address above to match your own--
                socket = new Socket(serverAddress, 500);
                commsThread = new CommsThread(socket);
                commsThread.start();
                //---sign in for the user; sends the nick name---
                sendToServer(NICKNAME);
```

```
            } catch (UnknownHostException e) {
                Log.d("Sockets", e.getLocalizedMessage());
            } catch (IOException e) {
                Log.d("Sockets", e.getLocalizedMessage());
            }
            return null;
        }
    }

    private class WriteToServerTask extends AsyncTask
    <byte[], Void, Void> {
        protected Void doInBackground(byte[]...data) {
            commsThread.write(data[0]);
            return null;
        }
    }

    private class CloseSocketTask extends AsyncTask
    <Void, Void, Void> {
        @Override
        protected Void doInBackground(Void... params) {
            try {
                socket.close();
            } catch (IOException e) {
                Log.d("Sockets", e.getLocalizedMessage());
            }
            return null;
        }
    }

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        //---get the views---
        txtMessage = (EditText) findViewById(R.id.txtMessage);
        txtMessagesReceived = (TextView)
                findViewById(R.id.txtMessagesReceived);
    }

    public void onClickSend(View view) {
        //---send the message to the server---
        sendToServer(txtMessage.getText().toString());
    }

    private void sendToServer(String message) {
        byte[] theByteArray =
                message.getBytes();
        new WriteToServerTask().execute(theByteArray);
    }

    @Override
    public void onResume() {
        super.onResume();
```

```
            new CreateCommThreadTask().execute();
        }

        @Override
        public void onPause() {
            super.onPause();
            new CloseSocketTask().execute();
        }
}
```

6. For testing, you will be using a socket server application that I have written (you can obtain this application through the source code download for this book at wrox.com). This application is a multi-user console application (for Windows) that simply listens at port 500 of the local computer and then broadcasts all the messages it receives to all the other clients connected to it. To run the server, open a Command window in Windows and type in the following command: **C:\>Server.exe *Your_IP_Address***. For example, if your computer had an IP address of 192.168.1.142, then you would enter the following:

```
C:\>Server.exe 192.168.1.142
```

7. Before you deploy the application onto a real device, ensure that the device is connected to the same network as your computer running the server described in the previous step. In a common setup, your computer is connected to the wireless router (either wired or wirelessly) and your device is connected wirelessly to the same wireless router. Once this is done, press F11 to deploy the application onto the Android device.

8. Type a message and tap the Send Message button (see Figure 10-7).

9. You will be able to see the message received by the server, as shown in Figure 10-8.

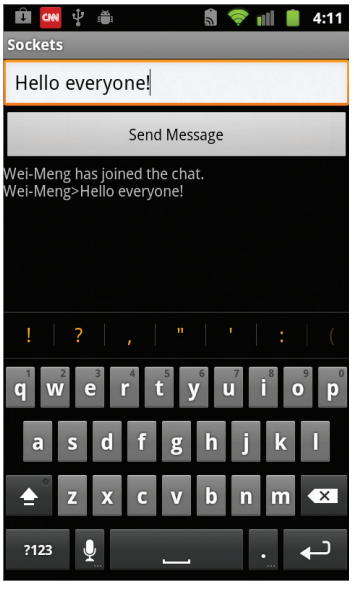

FIGURE 10-7

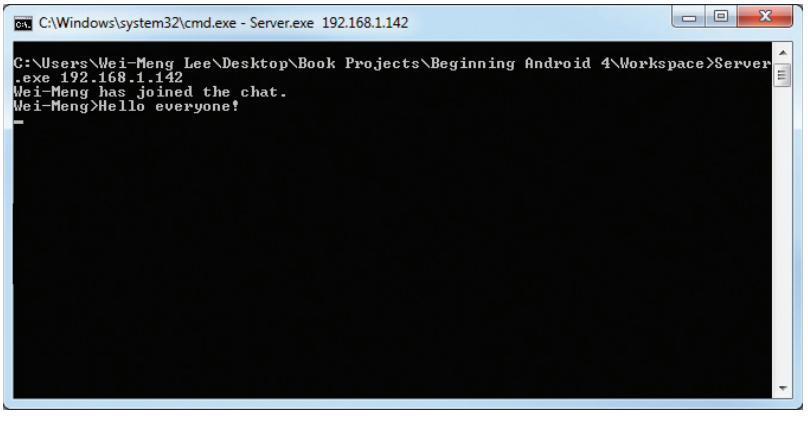

FIGURE 10-8

How It Works

To handle the intricacies of sockets communication, you created a separate class and called it the `CommsThread` (for communication thread). This class extends the `Thread` class so that all the sockets communication can be performed on a different thread, separate from the main UI thread:

```
public class CommsThread extends Thread {
}
```

Within this class, you declared three objects:

```
private final Socket socket;
private final InputStream inputStream;
private final OutputStream outputStream;
```

The first is a `Socket` object, which provides a client-side TCP socket. The `InputStream` object helps to read data from the socket connection. The `OutputStream` object helps to write data to the socket connection.

The constructor for the `CommsThread` class takes in a `Socket` instance and then tries to obtain an `InputStream` and `OutputStream` objects from the socket connection:

```
public CommsThread(Socket sock) {
    socket = sock;
    InputStream tmpIn = null;
    OutputStream tmpOut = null;
    try {
        //---creates the inputstream and outputstream objects
        // for reading and writing through the sockets---
        tmpIn = socket.getInputStream();
        tmpOut = socket.getOutputStream();
    } catch (IOException e) {
        Log.d("SocketChat", e.getLocalizedMessage());
    }
    inputStream = tmpIn;
    outputStream = tmpOut;
}
```

The `run()` method (which is called when you call the `start()` method of this thread) keeps listening for incoming data by reading perpetually using the `InputStream` object. When data is received, it updates the main activity's UI by passing it a message containing the data received:

```
public void run() {
    //---buffer store for the stream---
    byte[] buffer = new byte[1024];

    //---bytes returned from read()---
    int bytes;

    //---keep listening to the InputStream until an
    // exception occurs---
    while (true) {
```

```
            try {
                //---read from the inputStream---
                bytes = inputStream.read(buffer);

                //---update the main activity UI---
                SocketsActivity.UIupdater.obtainMessage(
                    0,bytes, -1, buffer).sendToTarget();
            } catch (IOException e) {
                break;
            }
        }
    }
```

The `write()` method helps to write data to the socket connection:

```
    //---call this from the main activity to
    // send data to the remote device---
    public void write(byte[] bytes) {
        try {
            outputStream.write(bytes);
        } catch (IOException e) { }
    }
```

Finally, the `cancel()` method closes the socket connection:

```
    //---call this from the main activity to
    // shutdown the connection---
    public void cancel() {
        try {
            socket.close();
        } catch (IOException e) { }
    }
```

In the `SocketsActivity.java` file, you created three subclasses that extended the `AsyncTask` class:

```
    private class CreateCommThreadTask extends AsyncTask
    <Void, Integer, Void> {
        @Override
        protected Void doInBackground(Void... params) {
            try {
                //---create a socket---
                serverAddress =
                    InetAddress.getByName("192.168.1.142");
                socket = new Socket(serverAddress, 500);
                commsThread = new CommsThread(socket);
                commsThread.start();
                //---sign in for the user; sends the nick name---
                sendToServer(NICKNAME);
            } catch (UnknownHostException e) {
                Log.d("Sockets", e.getLocalizedMessage());
            } catch (IOException e) {
                Log.d("Sockets", e.getLocalizedMessage());
            }
            return null;
```

```
    }
}

private class WriteToServerTask extends AsyncTask
<byte[], Void, Void> {
    protected Void doInBackground(byte[]...data) {
        commsThread.write(data[0]);
        return null;
    }
}

private class CloseSocketTask extends AsyncTask
<Void, Void, Void> {
    @Override
    protected Void doInBackground(Void... params) {
        try {
            socket.close();
        } catch (IOException e) {
            Log.d("Sockets", e.getLocalizedMessage());
        }
        return null;
    }
}
```

The `CreateCommThreadTask` class asynchronously creates a socket connection with the server. For the socket server, the first string sent by the client after the connection is established will be treated as the nickname for the client. Hence, after the socket connection is started, you immediately send a message to the server containing the nickname you want to use for your client:

```
//---sign in for the user; sends the nick name---
sendToServer(NICKNAME);
```

The `WriteToServerTask` class enables you to send data to the server asynchronously, while the `CloseSocketTask` class closes a socket connection.

The `sendToServer()` method takes in a `String` argument and converts it into a byte array. It then calls the `execute()` method of the `WriteToServerTask` class to send the data to the server asynchronously:

```
private void sendToServer(String message) {
    byte[] theByteArray =
            message.getBytes();
    new WriteToServerTask().execute(theByteArray);
}
```

Finally, when the activity is paused, you close the socket connection; and when it is resumed, you establish the connection again:

```
@Override
public void onPause() {
    super.onPause();
    new CloseSocketTask().execute();
}

@Override
```

```
public void onResume() {
    super.onResume();
    new CreateCommThreadTask().execute();
}
```

SUMMARY

In this chapter, you learned how your application can connect with the outside world through the use of the HTTP protocol. Using the HTTP protocol, you can download various types of data from web servers. One good application of this is to talk to web services, whereby you need to parse XML files. In addition to XML web services, you also saw how to consume JSON services, which are more lightweight than XML web services. Finally, you saw an alternative to HTTP: using sockets for communication. Sockets enable your application to remain connected to a server so that it can receive data as and when it becomes available. A very important lesson learned in this chapter is that all synchronous operations must be encapsulated using the `AsyncTask` class; otherwise, your application will not work on devices running Honeycomb or later.

EXERCISES

1. Name the permissions you need to declare in your `AndroidManifest.xml` file for an HTTP connection.
2. Name the classes used for dealing with JSON messages.
3. Name the class for performing background asynchronous tasks.

Answers to the exercises can be found in Appendix C.

▶ WHAT YOU LEARNED IN THIS CHAPTER

TOPIC	KEY CONCEPTS
Establishing an HTTP connection	Use the `HttpURLConnection` class.
Accessing XML web services	Use the `Document`, `DocumentBuilderFactory`, and `DocumentBuilder` classes to parse the XML result returned by the web service.
Dealing with JSON messages	Use the `JSONArray` and `JSONObject` classes.
Sockets programming	Use the `Socket` class to establish a TCP connection. Use the `InputStream` and `OutputStream` objects for receiving and sending data, respectively.
The three methods in an `AsyncTask` class	The three methods are `doInBackground()`, `onProgressUpdate()`, and `onPostExecute()`.

11

Developing Android Services

WHAT YOU WILL LEARN IN THIS CHAPTER

- How to create a service that runs in the background
- How to perform long-running tasks in a separate thread
- How to perform repeated tasks in a service
- How an activity and a service communicate

A service is an application in Android that runs in the background without needing to interact with the user. For example, while using an application, you may want to play some background music at the same time. In this case, the code that is playing the background music has no need to interact with the user, and hence it can be run as a service. Services are also ideal for situations in which there is no need to present a UI to the user. A good example of this scenario is an application that continually logs the geographical coordinates of the device. In this case, you can write a service to do that in the background. In this chapter, you will learn how to create your own services and use them to perform background tasks asynchronously.

CREATING YOUR OWN SERVICES

The best way to understand how a service works is by creating one. The following Try It Out shows you the steps to create a simple service. Subsequent sections add more functionality to this service. For now, you will learn how to start and stop a service.

TRY IT OUT Creating a Simple Service

codefile Services.zip available for download at Wrox.com

1. Using Eclipse, create a new Android project and name it **Services**.
2. Add a new Java Class file to the project and name it **MyService**. Populate the `MyService.java` file with the following code:

```
package net.learn2develop.Services;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.widget.Toast;

public class MyService extends Service {

    @Override
    public IBinder onBind(Intent arg0) {
        return null;
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        // We want this service to continue running until it is explicitly
        // stopped, so return sticky.
        Toast.makeText(this, "Service Started", Toast.LENGTH_LONG).show();
        return START_STICKY;
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        Toast.makeText(this, "Service Destroyed", Toast.LENGTH_LONG).show();
    }
}
```

3. In the `AndroidManifest.xml` file, add the following statement in bold:

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

    <uses-sdk android:minSdkVersion="14" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
```

```
            android:label="@string/app_name"
            android:name=".ServicesActivity" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

4. In the `main.xml` file, add the following statements in bold, replacing `TextView`:

```
<?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" >

<Button android:id="@+id/btnStartService"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Start Service"
    android:onClick="startService"/>

<Button android:id="@+id/btnStopService"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Stop Service"
    android:onClick="stopService" />

</LinearLayout>
```

5. Add the following statements in bold to the `ServicesActivity.java` file:

```
package net.learn2develop.Services;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class ServicesActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void startService(View view) {
```

```
        startService(new Intent(getBaseContext(), MyService.class));
    }

    public void stopService(View view) {
        stopService(new Intent(getBaseContext(),
MyService.class));
    }

}
```

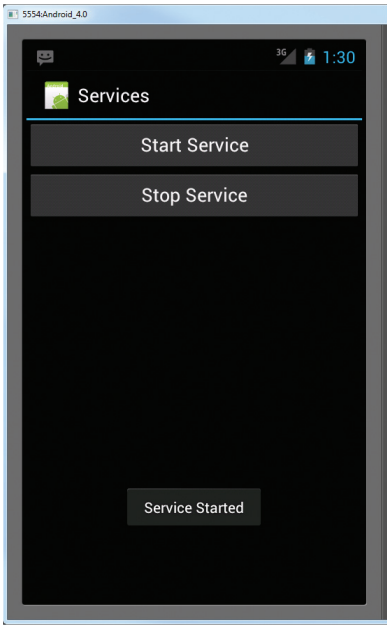

FIGURE 11-1

6. Press F11 to debug the application on the Android emulator.
7. Clicking the Start Service button will start the service (see Figure 11-1). To stop the service, click the Stop Service button.

How It Works

This example demonstrated the simplest service that you can create. The service itself is not doing anything useful, of course, but it serves to illustrate the creation process.

First, you defined a class that extends the `Service` base class. All services extend the `Service` class:

```
public class MyService extends Service {
}
```

Within the `MyService` class, you implemented three methods:

```
@Override
public IBinder onBind(Intent arg0) {  ...  }

@Override
public int onStartCommand(Intent intent, int flags, int startId) { ... }

@Override
public void onDestroy() { ... }
```

The `onBind()` method enables you to bind an activity to a service. This in turn enables an activity to directly access members and methods inside a service. For now, you simply return a `null` for this method. Later in this chapter you will learn more about binding.

The `onStartCommand()` method is called when you start the service explicitly using the `startService()` method (discussed shortly). This method signifies the start of the service, and you code it to do the things you need to do for your service. In this method, you returned the constant `START_STICKY` so that the service will continue to run until it is explicitly stopped.

The `onDestroy()` method is called when the service is stopped using the `stopService()` method. This is where you clean up the resources used by your service.

All services that you have created must be declared in the `AndroidManifest.xml` file, like this:

```
<service android:name=".MyService" />
```

If you want your service to be available to other applications, you can always add an intent filter with an action name, like this:

```
<service android:name=".MyService">
    <intent-filter>
        <action android:name="net.learn2develop.MyService" />
    </intent-filter>
</service>
```

To start a service, you use the `startService()` method, like this:

```
startService(new Intent(getBaseContext(), MyService.class));
```

If you are calling this service from an external application, then the call to the `startService()` method looks like this:

```
startService(new Intent("net.learn2develop.MyService"));
```

To stop a service, use the `stopService()` method:

```
stopService(new Intent(getBaseContext(), MyService.class));
```

Performing Long-Running Tasks in a Service

Because the service you created in the previous section does not do anything useful, in this section you will modify it so that it performs a task. In the following Try It Out, you will simulate the service of downloading a file from the Internet.

TRY IT OUT Making Your Service Useful

1. Using the Services project created in the first example, add the following statements in bold to the `ServicesActivity.java` file:

```
package net.learn2develop.Services;

import java.net.MalformedURLException;
import java.net.URL;

import android.app.Service;
```

```
import android.content.Intent;
import android.os.IBinder;
import android.widget.Toast;

public class MyService extends Service {

    @Override
    public IBinder onBind(Intent arg0) {
        return null;
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        // We want this service to continue running until it is explicitly
        // stopped, so return sticky.
        //Toast.makeText(this, "Service Started", Toast.LENGTH_LONG).show();

        try {
            int result = DownloadFile(new URL("http://www.amazon.com/somefile.pdf"));
            Toast.makeText(getBaseContext(),
                "Downloaded " + result + " bytes",
                Toast.LENGTH_LONG).show();
        } catch (MalformedURLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return START_STICKY;
    }

    private int DownloadFile(URL url) {
        try {
            //---simulate taking some time to download a file---
            Thread.sleep(5000);
        } catch (InterruptedException e) {
             e.printStackTrace();
        }
        //---return an arbitrary number representing
        // the size of the file downloaded---
        return 100;
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        Toast.makeText(this, "Service Destroyed", Toast.LENGTH_LONG).show();
    }
```

2. Press F11 to debug the application on the Android emulator.
3. Click the Start Service button to start the service to download the file. Note that the activity is frozen for a few seconds before the `Toast` class displays the "Downloaded 100 bytes" message (see Figure 11-2).

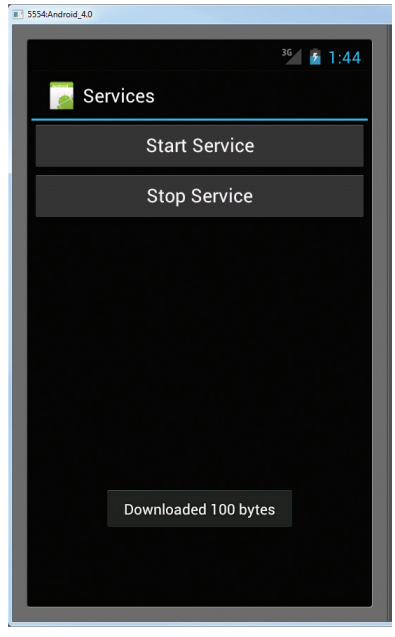

FIGURE 11-2

How It Works

In this example, your service calls the `DownloadFile()` method to simulate downloading a file from a given URL. This method returns the total number of bytes downloaded (which you have hardcoded as 100). To simulate the delays experienced by the service when downloading the file, you used the `Thread.Sleep()` method to pause the service for five seconds (5,000 milliseconds).

As you start the service, note that the activity is suspended for about five seconds, which is the time taken for the file to be downloaded from the Internet. During this time, the entire activity is not responsive, demonstrating a very important point: The service runs on the same thread as your activity. In this case, because the service is suspended for five seconds, so is the activity.

Hence, for a long-running service, it is important that you put all long-running code into a separate thread so that it does not tie up the application that calls it. The following Try It Out shows you how.

TRY IT OUT Performing Tasks in a Service Asynchronously

codefile Services.zip available for download at Wrox.com

1. Using the Services project created in the first example, add the following statements in bold to the `MyService.java` file:

```
package net.learn2develop.Services;

import java.net.MalformedURLException;
import java.net.URL;

import android.app.Service;
import android.content.Intent;
import android.os.AsyncTask;
import android.os.IBinder;
import android.util.Log;
import android.widget.Toast;

public class MyService extends Service {

    @Override
    public IBinder onBind(Intent arg0) {
```

```
        return null;
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        // We want this service to continue running until it is explicitly
        // stopped, so return sticky.
        //Toast.makeText(this, "Service Started", Toast.LENGTH_LONG).show();

        try {
            new DoBackgroundTask().execute(
                    new URL("http://www.amazon.com/somefiles.pdf"),
                    new URL("http://www.wrox.com/somefiles.pdf"),
                    new URL("http://www.google.com/somefiles.pdf"),
                    new URL("http://www.learn2develop.net/somefiles.pdf"));

        } catch (MalformedURLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return START_STICKY;
    }

    private int DownloadFile(URL url) {
        try {
            //---simulate taking some time to download a file---
            Thread.sleep(5000);
        } catch (InterruptedException e) {
             e.printStackTrace();
        }
        //---return an arbitrary number representing
        // the size of the file downloaded---
        return 100;
    }

    private class DoBackgroundTask extends AsyncTask<URL, Integer, Long> {
        protected Long doInBackground(URL... urls) {
            int count = urls.length;
            long totalBytesDownloaded = 0;
            for (int i = 0; i < count; i++) {
                totalBytesDownloaded += DownloadFile(urls[i]);
                //---calculate percentage downloaded and
                // report its progress---
                publishProgress((int) (((i+1) / (float) count) * 100));
            }
            return totalBytesDownloaded;
        }

        protected void onProgressUpdate(Integer... progress) {
            Log.d("Downloading files",
                    String.valueOf(progress[0]) + "% downloaded");
            Toast.makeText(getBaseContext(),
                String.valueOf(progress[0]) + "% downloaded",
```

```
                Toast.LENGTH_LONG).show();
        }

        protected void onPostExecute(Long result) {
            Toast.makeText(getBaseContext(),
                    "Downloaded " + result + " bytes",
                    Toast.LENGTH_LONG).show();
            stopSelf();
        }
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        Toast.makeText(this, "Service Destroyed", Toast.LENGTH_LONG).show();
    }
}
```

2. Press F11 to debug the application on the Android emulator.
3. Click the Start Service button. The `Toast` class will display a message indicating what percentage of the download is completed. You should see four of them: 25%, 50%, 75%, and 100%.
4. You can see output similar to the following in the LogCat window:

```
12-06 01:58:24.967: D/Downloading files(6020): 25% downloaded
12-06 01:58:30.019: D/Downloading files(6020): 50% downloaded
12-06 01:58:35.078: D/Downloading files(6020): 75% downloaded
12-06 01:58:40.096: D/Downloading files(6020): 100% downloaded
```

How It Works

This example illustrates one way in which you can execute a task asynchronously within your service. You do so by creating an inner class that extends the `AsyncTask` class. The `AsyncTask` class enables you to perform background execution without needing to manually handle threads and handlers.

The `DoBackgroundTask` class extends the `AsyncTask` class by specifying three generic types:

```
private class DoBackgroundTask extends AsyncTask<URL, Integer, Long> {
```

In this case, the three types specified are `URL`, `Integer` and `Long`. These three types specify the data type used by the following three methods that you implement in an `AsyncTask` class:

- `doInBackground()` — This method accepts an array of the first generic type specified earlier. In this case, the type is `URL`. This method is executed in the background thread and is where you put your long-running code. To report the progress of your task, you call the `publishProgress()` method, which invokes the next method, `onProgressUpdate()`, which you implement in an `AsyncTask` class. The return type of this method takes the third generic type specified earlier, which is `Long` in this case.

- `onProgressUpdate()` — This method is invoked in the UI thread and is called when you call the `publishProgress()` method. It accepts an array of the second generic type specified earlier. In this case, the type is `Integer`. Use this method to report the progress of the background task to the user.
- `onPostExecute()` — This method is invoked in the UI thread and is called when the `doInBackground()` method has finished execution. This method accepts an argument of the third generic type specified earlier, which in this case is a `Long`.

Figure 11-3 summarizes the types specified and their relationship to the three methods inside a subclass of the `AsyncTask` class.

```
private class DoBackgroundTask extends AsyncTask<URL, Integer, Long> {
    protected Long doInBackground(URL... urls) {
        int count = urls.length;
        long totalBytesDownloaded = 0;
        for (int i = 0; i < count; i++) {
            totalBytesDownloaded += DownloadFile(urls[i]);
            //---calculate percentage downloaded and
            // report its progress---
            publishProgress((int) (((i+1) / (float) count) * 100));
        }
        return totalBytesDownloaded;
    }

    protected void onProgressUpdate(Integer... progress) {
        Log.d("Downloading files",
                String.valueOf(progress[0]) + "% downloaded");
        Toast.makeText(getBaseContext(),
            String.valueOf(progress[0]) + "% downloaded",
            Toast.LENGTH_LONG).show();
    }

    protected void onPostExecute(Long result) {
        Toast.makeText(getBaseContext(),
                "Downloaded " + result + " bytes",
                Toast.LENGTH_LONG).show();
        stopSelf();
    }
}
```

FIGURE 11-3

To download multiple files in the background, you created an instance of the `DoBackgroundTask` class and then called its `execute()` method by passing in an array of `URL`s:

```
try {
    new DoBackgroundTask().execute(
            new URL("http://www.amazon.com/somefiles.pdf"),
            new URL("http://www.wrox.com/somefiles.pdf"),
            new URL("http://www.google.com/somefiles.pdf"),
            new URL("http://www.learn2develop.net/somefiles.pdf"));

} catch (MalformedURLException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

The preceding causes the service to download the files in the background, and reports the progress as a percentage of files downloaded. More important, the activity remains responsive while the files are downloaded in the background, on a separate thread.

Note that when the background thread has finished execution, you can manually call the `stopSelf()` method to stop the service:

```
protected void onPostExecute(Long result) {
    Toast.makeText(getBaseContext(),
            "Downloaded " + result + " bytes",
            Toast.LENGTH_LONG).show();
    stopSelf();
}
```

The `stopSelf()` method is the equivalent of calling the `stopService()` method to stop the service.

Performing Repeated Tasks in a Service

In addition to performing long-running tasks in a service, you might also perform some repeated tasks in a service. For example, you may write an alarm clock service that runs persistently in the background. In this case, your service may need to periodically execute some code to check whether a prescheduled time has been reached so that an alarm can be sounded. To execute a block of code to be executed at a regular time interval, you can use the `Timer` class within your service. The following Try It Out shows you how.

TRY IT OUT Running Repeated Tasks Using the Timer Class

codefile Services.zip available for download at Wrox.com

1. Using the Services project again, add the following statements in bold to the `MyService.java` file:

```
package net.learn2develop.Services;

import java.net.MalformedURLException;
import java.net.URL;
import java.util.Timer;
import java.util.TimerTask;

import android.app.Service;
import android.content.Intent;
import android.os.AsyncTask;
import android.os.IBinder;
import android.util.Log;
import android.widget.Toast;

public class MyService extends Service {
    int counter = 0;
    static final int UPDATE_INTERVAL = 1000;
    private Timer timer = new Timer();

    @Override
    public IBinder onBind(Intent arg0) {
        return null;
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        // We want this service to continue running until it is explicitly
        // stopped, so return sticky.
        //Toast.makeText(this, "Service Started", Toast.LENGTH_LONG).show();

        doSomethingRepeatedly();

        try {
            new DoBackgroundTask().execute(
                    new URL("http://www.amazon.com/somefiles.pdf"),
                    new URL("http://www.wrox.com/somefiles.pdf"),
                    new URL("http://www.google.com/somefiles.pdf"),
                    new URL("http://www.learn2develop.net/somefiles.pdf"));

        } catch (MalformedURLException e) {
```

```
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    return START_STICKY;
}

private void doSomethingRepeatedly() {
    timer.scheduleAtFixedRate(new TimerTask() {
        public void run() {
            Log.d("MyService", String.valueOf(++counter));
        }
    }, 0, UPDATE_INTERVAL);
}

private int DownloadFile(URL url) {
    try {
        //---simulate taking some time to download a file---
        Thread.sleep(5000);
    } catch (InterruptedException e) {
         e.printStackTrace();
    }
    //---return an arbitrary number representing
    // the size of the file downloaded---
    return 100;
}

private class DoBackgroundTask extends AsyncTask<URL, Integer, Long> {
    protected Long doInBackground(URL... urls) {
        int count = urls.length;
        long totalBytesDownloaded = 0;
        for (int i = 0; i < count; i++) {
            totalBytesDownloaded += DownloadFile(urls[i]);
            //---calculate percentage downloaded and
            // report its progress---
            publishProgress((int) (((i+1) / (float) count) * 100));
        }
        return totalBytesDownloaded;
    }

    protected void onProgressUpdate(Integer... progress) {
        Log.d("Downloading files",
                String.valueOf(progress[0]) + "% downloaded");
        Toast.makeText(getBaseContext(),
            String.valueOf(progress[0]) + "% downloaded",
            Toast.LENGTH_LONG).show();
    }

    protected void onPostExecute(Long result) {
        Toast.makeText(getBaseContext(),
                "Downloaded " + result + " bytes",
                Toast.LENGTH_LONG).show();
        stopSelf();
    }
}

@Override
```

```
    public void onDestroy() {
        super.onDestroy();

        if (timer != null){
            timer.cancel();
        }

        Toast.makeText(this, "Service Destroyed", Toast.LENGTH_LONG).show();
    }
}
```

2. Press F11 to debug the application on the Android emulator.
3. Click the Start Service button.
4. Observe the output displayed in the LogCat window. It will be similar to the following:

```
12-06 02:37:54.118: D/MyService(7752): 1
12-06 02:37:55.109: D/MyService(7752): 2
12-06 02:37:56.120: D/MyService(7752): 3
12-06 02:37:57.111: D/MyService(7752): 4
12-06 02:37:58.125: D/MyService(7752): 5
12-06 02:37:59.137: D/MyService(7752): 6
```

How It Works

In this example, you created a `Timer` object and called its `scheduleAtFixedRate()` method inside the `doSomethingRepeatedly()` method that you have defined:

```
private void doSomethingRepeatedly() {
    timer.scheduleAtFixedRate( new TimerTask() {
        public void run() {
            Log.d("MyService", String.valueOf(++counter));
        }
    }, 0, UPDATE_INTERVAL);
}
```

You passed an instance of the `TimerTask` class to the `scheduleAtFixedRate()` method so that you can execute the block of code within the `run()` method repeatedly. The second parameter to the `scheduleAtFixedRate()` method specifies the amount of time, in milliseconds, before first execution. The third parameter specifies the amount of time, in milliseconds, between subsequent executions.

In the preceding example, you essentially print out the value of the counter every second (1,000 milliseconds). The service repeatedly prints the value of `counter` until the service is terminated:

```
@Override
public void onDestroy() {
    super.onDestroy();

    if (timer != null){
        timer.cancel();
    }

    Toast.makeText(this, "Service Destroyed", Toast.LENGTH_LONG).show();
}
```

For the `scheduleAtFixedRate()` method, your code is executed at fixed time intervals, regardless of how long each task takes. For example, if the code within your `run()` method takes two seconds to complete, then your second task will start immediately after the first task has ended. Similarly, if your delay is set to three seconds and the task takes two seconds to complete, then the second task will wait for one second before starting.

Also, observe that you call the `doSomethingRepeatedly()` method directly in the `onStartCommand()` method, without needing to wrap it in a subclass of the `AsyncTask` class. This is because the `TimerTask` class itself implements the `Runnable` interface, which allows it to run on a separate thread.

Executing Asynchronous Tasks on Separate Threads Using IntentService

Earlier in this chapter, you learned how to start a service using the `startService()` method and stop a service using the `stopService()` method. You have also seen how you should execute long-running task on a separate thread — not the same thread as the calling activities. It is important to note that once your service has finished executing a task, it should be stopped as soon as possible so that it does not unnecessarily hold up valuable resources. That's why you use the `stopSelf()` method to stop the service when a task has been completed. Unfortunately, a lot of developers often forgot to terminate a service when it is done performing its task. To easily create a service that runs a task asynchronously and terminates itself when it is done, you can use the `IntentService` class.

The `IntentService` class is a base class for `Service` that handles asynchronous requests on demand. It is started just like a normal service; and it executes its task within a worker thread and terminates itself when the task is completed. The following Try It Out demonstrates how to use the `IntentService` class.

TRY IT OUT Using the IntentService Class to Auto-Stop a Service

codefile Services.zip available for download at Wrox.com

1. Using the Services project created in the first example, add a new Class file named **MyIntentService.java**.
2. Populate the `MyIntentService.java` file as follows:

```
package net.learn2develop.Services;

import java.net.MalformedURLException;
import java.net.URL;

import android.app.IntentService;
import android.content.Intent;
import android.util.Log;

public class MyIntentService extends IntentService {

    public MyIntentService() {
        super("MyIntentServiceName");
    }

    @Override
```

```
    protected void onHandleIntent(Intent intent) {
        try {
            int result =
                DownloadFile(new URL("http://www.amazon.com/somefile.pdf"));
            Log.d("IntentService", "Downloaded " + result + " bytes");
        } catch (MalformedURLException e) {
            e.printStackTrace();
        }
    }

    private int DownloadFile(URL url) {
        try {
            //---simulate taking some time to download a file---
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        return 100;
    }
}
```

3. Add the following statement in bold to the `AndroidManifest.xml` file:

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

    <uses-sdk android:minSdkVersion="14" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:label="@string/app_name"
            android:name=".ServicesActivity" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />

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

        <service android:name=".MyService">
            <intent-filter>
                <action android:name="net.learn2develop.MyService" />
            </intent-filter>
        </service>
        <service android:name=".MyIntentService" />

    </application>

</manifest>
```

4. Add the following statement in bold to the `ServicesActivity.java` file:

```
public void startService(View view) {
    //startService(new Intent(getBaseContext(), MyService.class));
    //OR
    //startService(new Intent("net.learn2develop.MyService"));
    startService(new Intent(getBaseContext(), MyIntentService.class));
}
```

5. Press F11 to debug the application on the Android emulator.

6. Click the Start Service button. After about five seconds, you should see something similar to the following statement in the LogCat window:

```
12-06 13:35:32.181: D/IntentService(861): Downloaded 100 bytes
```

How It Works

First, you defined the `MyIntentService` class, which extends the `IntentService` class instead of the `Service` class:

```
public class MyIntentService extends IntentService {
}
```

You needed to implement a constructor for the class and call its superclass with the name of the intent service (setting it with a string):

```
public MyIntentService() {
    super("MyIntentServiceName");
}
```

You then implemented the `onHandleIntent()` method, which is executed on a worker thread:

```
@Override
protected void onHandleIntent(Intent intent) {
    try {
        int result =
            DownloadFile(new URL("http://www.amazon.com/somefile.pdf"));
        Log.d("IntentService", "Downloaded " + result + " bytes");
    } catch (MalformedURLException e) {
        e.printStackTrace();
    }
}
```

The `onHandleIntent()` method is where you place the code that needs to be executed on a separate thread, such as downloading a file from a server. When the code has finished executing, the thread is terminated and the service is stopped automatically.

ESTABLISHING COMMUNICATION BETWEEN A SERVICE AND AN ACTIVITY

Often a service simply executes in its own thread, independently of the activity that calls it. This doesn't pose any problem if you simply want the service to perform some tasks periodically and the activity does not need to be notified about the service's status. For example, you may have a service that periodically logs the geographical location of the device to a database. In this case, there is no need for your service to interact with any activities, because its main purpose is to save the coordinates into a database. However, suppose you want to monitor for a particular location. When the service logs an address that is near the location you are monitoring, it might need to communicate that information to the activity. If so, you need to devise a way for the service to interact with the activity.

The following Try It Out demonstrates how a service can communicate with an activity using a `BroadcastReceiver`.

TRY IT OUT Invoking an Activity from a Service

codefile Services.zip available for download at Wrox.com

1. Using the Services project created earlier, add the following statements in bold to the `MyIntentService.java` file:

```
package net.learn2develop.Services;

import java.net.MalformedURLException;
import java.net.URL;

import android.app.IntentService;
import android.content.Intent;
import android.util.Log;

public class MyIntentService extends IntentService {

    public MyIntentService() {
        super("MyIntentServiceName");
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        try {
            int result =
                DownloadFile(new URL("http://www.amazon.com/somefile.pdf"));
            Log.d("IntentService", "Downloaded " + result + " bytes");

            //---send a broadcast to inform the activity
            // that the file has been downloaded---
            Intent broadcastIntent = new Intent();
            broadcastIntent.setAction("FILE_DOWNLOADED_ACTION");
            getBaseContext().sendBroadcast(broadcastIntent);

        } catch (MalformedURLException e) {
            e.printStackTrace();
        }
```

```
    }

    private int DownloadFile(URL url) {
        try {
            //---simulate taking some time to download a file---
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return 100;
    }
}
```

2. Add the following statements in bold to the `ServicesActivity.java` file:

```
package net.learn2develop.Services;

import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class ServicesActivity extends Activity {
    IntentFilter intentFilter;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

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

        //---intent to filter for file downloaded intent---
        intentFilter = new IntentFilter();
        intentFilter.addAction("FILE_DOWNLOADED_ACTION");

        //---register the receiver---
        registerReceiver(intentReceiver, intentFilter);
    }

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

        //---unregister the receiver---
```

```
        unregisterReceiver(intentReceiver);
    }

    public void startService(View view) {
        //startService(new Intent(getBaseContext(), MyService.class));
        //OR
        //startService(new Intent("net.learn2develop.MyService"));
        startService(new Intent(getBaseContext(), MyIntentService.class));
    }

    public void stopService(View view) {
        stopService(new Intent(getBaseContext(), MyService.class));
    }

    private BroadcastReceiver intentReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            Toast.makeText(getBaseContext(), "File downloaded!",
                    Toast.LENGTH_LONG).show();
        }
    };

}
```

3. Press F11 to debug the application on the Android emulator.
4. Click the Start Service button. After about five seconds, the `Toast` class will display a message indicating that the file has been downloaded (see Figure 11-4).

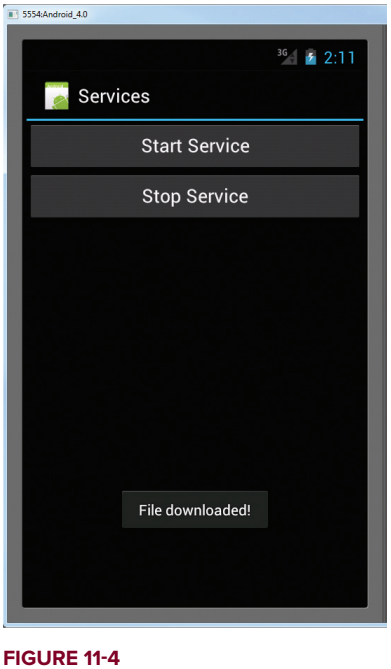

FIGURE 11-4

How It Works

To notify an activity when a service has finished its execution, you broadcast an intent using the `sendBroadcast()` method:

```
@Override
protected void onHandleIntent(Intent intent) {
    try {
        int result =
            DownloadFile(new URL("http://www.amazon.com/somefile.pdf"));
        Log.d("IntentService", "Downloaded " + result + " bytes");

        //---send a broadcast to inform the activity
        // that the file has been downloaded---
        Intent broadcastIntent = new Intent();
        broadcastIntent.setAction("FILE_DOWNLOADED_ACTION");
        getBaseContext().sendBroadcast(broadcastIntent);

    } catch (MalformedURLException e) {
        e.printStackTrace();
    }
}
```

The action of this intent that you are broadcasting is set to `"FILE_DOWNLOADED_ACTION"`, which means any activity that is listening for this intent will be invoked. Hence, in your `ServicesActivity.java` file, you listen for this intent using the `registerReceiver()` method from the `IntentFilter` class:

```
@Override
public void onResume() {
    super.onResume();

    //---intent to filter for file downloaded intent---
    intentFilter = new IntentFilter();
    intentFilter.addAction("FILE_DOWNLOADED_ACTION");

    //---register the receiver---
    registerReceiver(intentReceiver, intentFilter);
}
```

When the intent is received, it invokes an instance of the `BroadcastReceiver` class that you have defined:

```
private BroadcastReceiver intentReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        Toast.makeText(getBaseContext(), "File downloaded!",
                Toast.LENGTH_LONG).show();
    }
  };
```

NOTE *Chapter 8 discusses the* `BroadcastReceiver` *class in more detail.*

In this case, you displayed the message "File downloaded!" Of course, if you need to pass some data from the service to the activity, you can make use of the `Intent` object. The next section discusses this.

BINDING ACTIVITIES TO SERVICES

So far, you have seen how services are created and how they are called and terminated when they are done with their task. All the services that you have seen are simple — either they start with a counter and increment at regular intervals or they download a fixed set of files from the Internet. However, real-world services are usually much more sophisticated, requiring the passing of data so that they can do the job correctly for you.

Using the service demonstrated earlier that downloads a set of files, suppose you now want to let the calling activity determine what files to download, instead of hardcoding them in the service. Here is what you need to do.

First, in the calling activity, you create an `Intent` object, specifying the service name:

```
public void startService(View view) {
    Intent intent = new Intent(getBaseContext(), MyService.class);
}
```

You then create an array of `URL` objects and assign it to the `Intent` object through its `putExtra()` method. Finally, you start the service using the `Intent` object:

```
public void startService(View view) {
    Intent intent = new Intent(getBaseContext(), MyService.class);
    try {
        URL[] urls = new URL[] {
                new URL("http://www.amazon.com/somefiles.pdf"),
                new URL("http://www.wrox.com/somefiles.pdf"),
                new URL("http://www.google.com/somefiles.pdf"),
                new URL("http://www.learn2develop.net/somefiles.pdf")};
        intent.putExtra("URLs", urls);
    } catch (MalformedURLException e) {
        e.printStackTrace();
    }
    startService(intent);
}
```

Note that the `URL` array is assigned to the `Intent` object as an `Object` array.

On the service's end, you need to extract the data passed in through the `Intent` object in the `onStartCommand()` method:

```
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    // We want this service to continue running until it is explicitly
    // stopped, so return sticky.
    Toast.makeText(this, "Service Started", Toast.LENGTH_LONG).show();
    Object[] objUrls = (Object[]) intent.getExtras().get("URLs");
    URL[] urls = new URL[objUrls.length];
    for (int i=0; i<objUrls.length-1; i++) {
        urls[i] = (URL) objUrls[i];
    }
    new DoBackgroundTask().execute(urls);
    return START_STICKY;
}
```

The preceding first extracts the data using the `getExtras()` method to return a `Bundle` object. It then uses the `get()` method to extract out the `URL` array as an `Object` array. Because in Java you cannot directly cast an array from one type to another, you have to create a loop and cast each member of the array individually. Finally, you execute the background task by passing the URL array into the `execute()` method.

This is one way in which your activity can pass values to the service. As you can see, if you have relatively complex data to pass to the service, you have to do some additional work to ensure that the data is passed correctly. A better way to pass data is to bind the activity directly to the service so that the activity can call any public members and methods on the service directly. The following Try It Out shows you how to bind an activity to a service.

TRY IT OUT Accessing Members of a Property Directly through Binding

codefile Services.zip available for download at Wrox.com

1. Using the Services project created earlier, add the following statements in bold to the `MyService.java` file (note that you are modifying the existing `onStartCommand()`):

```
import android.os.Binder;

import android.os.IBinder;

public class MyService extends Service {
    int counter = 0;
    URL[] urls;
    static final int UPDATE_INTERVAL = 1000;
    private Timer timer = new Timer();
    private final IBinder binder = new MyBinder();

    public class MyBinder extends Binder {
        MyService getService() {
            return MyService.this;
        }
    }

    @Override
    public IBinder onBind(Intent arg0) {
```

```
        return binder;
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        // We want this service to continue running until it is explicitly
        // stopped, so return sticky.
        Toast.makeText(this, "Service Started", Toast.LENGTH_LONG).show();
        new DoBackgroundTask().execute(urls);
        return START_STICKY;
    }


    private void doSomethingRepeatedly() { … }

    private int DownloadFile(URL url) { ... }

    private class DoBackgroundTask extends AsyncTask<URL, Integer, Long> { ... }

    @Override
    public void onDestroy() { ... }
}
```

2. In the `ServicesActivity.java` file, add the following statements in bold (note the change to the existing `startService()` method):

```
import android.content.ComponentName;
import android.os.IBinder;
import android.content.ServiceConnection;
import java.net.MalformedURLException;
import java.net.URL;

public class ServicesActivity extends Activity {
    IntentFilter intentFilter;

    MyService serviceBinder;
    Intent i;

    private ServiceConnection connection = new ServiceConnection() {
        public void onServiceConnected(
            ComponentName className, IBinder service) {
            //--called when the connection is made--
            serviceBinder = ((MyService.MyBinder)service).getService();
            try {
                URL[] urls = new URL[] {
                    new URL("http://www.amazon.com/somefiles.pdf"),
                    new URL("http://www.wrox.com/somefiles.pdf"),
                    new URL("http://www.google.com/somefiles.pdf"),
                    new URL("http://www.learn2develop.net/somefiles.pdf")};
                    //---assign the URLs to the service through the
                    // serviceBinder object---
                    serviceBinder.urls = urls;
                } catch (MalformedURLException e) {
                    e.printStackTrace();
                }
```

```
                startService(i);
            }
            public void onServiceDisconnected(ComponentName className) {
                //---called when the service disconnects---
                serviceBinder = null;
            }
        };

        public void startService(View view) {
            i = new Intent(ServicesActivity.this, MyService.class);
            bindService(i, connection, Context.BIND_AUTO_CREATE);
        }

        @Override
        public void onCreate(Bundle savedInstanceState) { ... }

        @Override
        public void onResume() { ... }

        @Override
        public void onPause() { ... }

        public void stopService(View view) { ... }

        private BroadcastReceiver intentReceiver = new BroadcastReceiver() {
            ...
        };

    }
```

3. Press F11 to debug the application. Clicking the Start Service button will start the service as normal.

How It Works

To bind activities to a service, you must first declare an inner class in your service that extends the `Binder` class:

```
public class MyBinder extends Binder {
    MyService getService() {
        return MyService.this;
    }
}
```

Within this class you implemented the `getService()` method, which returns an instance of the service.

You then created an instance of the `MyBinder` class:

```
private final IBinder binder = new MyBinder();
```

You also modified the `onBind()` method to return the `MyBinder` instance:

```
@Override
public IBinder onBind(Intent arg0) {
    return binder;
}
```

In the `onStartCommand()` method, you then called the `execute()` method using the `urls` array, which you declared as a public member in your service:

```
public class MyService extends Service {
    int counter = 0;
    URL[] urls;
...
...
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        // We want this service to continue running until it is explicitly
        // stopped, so return sticky.
        Toast.makeText(this, "Service Started", Toast.LENGTH_LONG).show();
        new DoBackgroundTask().execute(urls);
        return START_STICKY;
    }
}
```

This `URL` array can be set directly from your activity, which you did next.

In the `ServicesActivity.java` file, you first declared an instance of your service and an `Intent` object:

```
MyService serviceBinder;
Intent i;
```

The `serviceBinder` object will be used as a reference to the service, which you accessed directly.

You then created an instance of the `ServiceConnection` class so that you could monitor the state of the service:

```
private ServiceConnection connection = new ServiceConnection() {
    public void onServiceConnected(
        ComponentName className, IBinder service) {
        //---called when the connection is made---
        serviceBinder = ((MyService.MyBinder)service).getService();
        try {
            URL[] urls = new URL[] {
                new URL("http://www.amazon.com/somefiles.pdf"),
                new URL("http://www.wrox.com/somefiles.pdf"),
                new URL("http://www.google.com/somefiles.pdf"),
                new URL("http://www.learn2develop.net/somefiles.pdf")};
                //---assign the URLs to the service through the
                // serviceBinder object---
                serviceBinder.urls = urls;
            } catch (MalformedURLException e) {
                e.printStackTrace();
            }
            startService(i);
    }
    public void onServiceDisconnected(ComponentName className) {
        //---called when the service disconnects---
        serviceBinder = null;
    }
};
```

You need to implement two methods: `onServiceConnected()` and `onServiceDisconnected()`. The `onServiceConnected()` method is called when the activity is connected to the service; the `onServiceDisconnected()` method is called when the service is disconnected from the activity.

In the `onServiceConnected()` method, when the activity is connected to the service, you obtained an instance of the service by using the `getService()` method of the `service` argument and then assigning it to the `serviceBinder` object. The `serviceBinder` object is a reference to the service, and all the members and methods in the service can be accessed through this object. Here, you created a URL array and then directly assigned it to the public member in the service:

```
URL[] urls = new URL[] {
    new URL("http://www.amazon.com/somefiles.pdf"),
    new URL("http://www.wrox.com/somefiles.pdf"),
    new URL("http://www.google.com/somefiles.pdf"),
    new URL("http://www.learn2develop.net/somefiles.pdf")};
    //---assign the URLs to the service through the
    // serviceBinder object---
    serviceBinder.urls = urls;
```

You then started the service using an `Intent` object:

```
startService(i);
```

Before you can start the service, you have to bind the activity to the service. This you did in the `startService()` method of the Start Service button:

```
public void startService(View view) {
    i = new Intent(ServicesActivity.this, MyService.class);
    bindService(i, connection, Context.BIND_AUTO_CREATE);
}
```

The `bindService()` method enables your activity to be connected to the service. It takes three arguments: an `Intent` object, a `ServiceConnection` object, and a flag to indicate how the service should be bound.

UNDERSTANDING THREADING

So far, you have seen how services are created and why it is important to ensure that your long-running tasks are properly handled, especially when updating the UI thread. Earlier in this chapter (as well as in Chapter 10), you also saw how to use the `AsyncTask` class for executing long-running code in the background. This section briefly summarizes the various ways to handle long-running tasks correctly using a variety of methods available.

For this discussion, assume that you have an Android project named **Threading**. The `main.xml` file contains a Button and TextView:

```
<?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" >

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello" />

    <Button
        android:id="@+id/btnStartCounter"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Start"
        android:onClick="startCounter" />

    <TextView
        android:id="@+id/textView1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="TextView" />

</LinearLayout>
```

Suppose you want to display a counter on the activity, from 0 to 1,000. In your `ThreadingActivity` class, you have the following code:

```
package net.learn2develop.Threading;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.TextView;

public class ThreadingActivity extends Activity {
    TextView txtView1;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        txtView1 = (TextView) findViewById(R.id.textView1);
    }

    public void startCounter(View view) {
        for (int i=0; i<=1000; i++) {
            txtView1.setText(String.valueOf(i));
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
```

```
                Log.d("Threading", e.getLocalizedMessage());
            }
        }
    }
}
```

When you run the application and click the Start button, the application is briefly frozen, and after a while you may see the message shown in Figure 11-5.

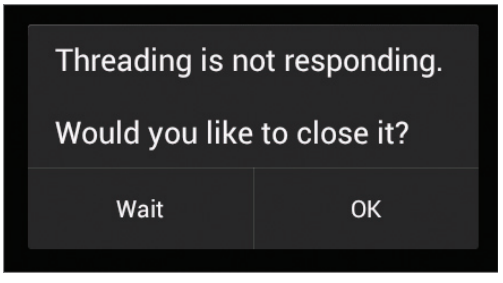

FIGURE 11-5

The UI freezes because the application is continuously trying to display the value of the counter at the same time it is pausing for one second after it has been displayed. This ties up the UI, which is waiting for the display of the numbers to be completed. The result is a nonresponsive application that will frustrate your users.

To solve this problem, one option is to wrap the part of the code that contains the loop using a `Thread` and `Runnable` class, like this:

```
public void startCounter(View view) {
    new Thread(new Runnable() {
        public void run() {
            for (int i=0; i<=1000; i++) {
                txtView1.setText(String.valueOf(i));
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    Log.d("Threading", e.getLocalizedMessage());
                }
            }
        }
    }).start();
}
```

In the preceding code, you first create a class that implements the `Runnable` interface. Within this class, you put your long-running code within the `run()` method. The `Runnable` block is then started using the `Thread` class.

NOTE *A `Runnable` is a block of code that can be executed by a thread.*

However, the preceding application will not work, and it will crash if you try to run it. This code that is placed inside the `Runnable` block is on a separate thread, and in the preceding example you are trying to update the UI from another thread, which is not a safe thing to do because Android UIs are not thread-safe. To resolve this, you need to use the `post()` method of a `View` to create

another `Runnable` block to be added to the message queue. In short, the new `Runnable` block created will be executed in the UI thread, so it would now be safe to execute your application:

```
public void startCounter(View view) {
    new Thread(new Runnable() {
        @Override
        public void run() {
            for (int i=0; i<=1000; i++) {
                final int valueOfi = i;

                //---update UI---
                txtView1.post(new Runnable() {
                    public void run() {
                        //---UI thread for updating---
                        txtView1.setText(String.valueOf(valueOfi));
                    }
                });

                //---insert a delay
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    Log.d("Threading", e.getLocalizedMessage());
                }
            }
        }
    }).start();
}
```

This application will now work correctly, but it is complicated and makes your code difficult to maintain.

A second option to update the UI from another thread is to use the `Handler` class. A `Handler` enables you to send and process messages, similar to using the `post()` method of a `View`. The following code snippets shows a `Handler` class called `UIupdater` that updates the UI using the message that it receives:

NOTE *For the following code to work, you need to import the* `android.os.Handler` *package as well as add the* `static` *modifier to* `txtView1`.

```
//---used for updating the UI on the main activity---
static Handler UIupdater = new Handler() {
    @Override
    public void handleMessage(Message msg) {
        byte[] buffer = (byte[]) msg.obj;

        //---convert the entire byte array to string---
        String strReceived = new String(buffer);

        //---display the text received on the TextView---
        txtView1.setText(strReceived);
```

```
            Log.d("Threading", "running");
        }
    };

    public void startCounter(View view) {
        new Thread(new Runnable() {
            @Override
            public void run() {
                for (int i=0; i<=1000; i++) {
                    //---update the main activity UI---
                    ThreadingActivity.UIupdater.obtainMessage(
                        0,  String.valueOf(i).getBytes() ).sendToTarget();
                    //---insert a delay
                    try {
                        Thread.sleep(1000);
                    } catch (InterruptedException e) {
                        Log.d("Threading", e.getLocalizedMessage());
                    }
                }
            }
        }).start();
    }

}
```

A detailed discussion of the `Handler` class is beyond the scope of this book. For more details, check out the documentation at `http://developer.android.com/reference/android/os/Handler.html`.

So far, the two methods just described enable you to update the UI from a separate thread. In Android, you could use the simpler `AsyncTask` class to do this. Using the `AsyncTask`, you could rewrite the preceding code as follows:

```
    private class DoCountingTask extends AsyncTask<Void, Integer, Void> {
        protected Void doInBackground(Void... params) {
            for (int i = 0; i < 1000; i++) {
                //---report its progress---
                publishProgress(i);
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    Log.d("Threading", e.getLocalizedMessage());
                }
            }
            return null;
        }

        protected void onProgressUpdate(Integer... progress) {
            txtView1.setText(progress[0].toString());
            Log.d("Threading", "updating...");
        }
    }

    public void startCounter(View view) {
        new DoCountingTask().execute();
    }
```

The preceding code will update the UI safely from another thread. What about stopping the task? If you run the preceding application and then click the Start button, the counter will start to display from zero. However, if you press the back button on the emulator/device, the task continues to run even though the activity has been destroyed. You can verify this through the LogCat window. If you want to stop the task, use the following code snippets:

```
public class ThreadingActivity extends Activity {
    static TextView txtView1;

    DoCountingTask task;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        txtView1 = (TextView) findViewById(R.id.textView1);
    }

    public void startCounter(View view) {
        task = (DoCountingTask) new DoCountingTask().execute();
    }

    public void stopCounter(View view) {
        task.cancel(true);
    }

    private class DoCountingTask extends AsyncTask<Void, Integer, Void> {
        protected Void doInBackground(Void... params) {
            for (int i = 0; i < 1000; i++) {
                //---report its progress---
                publishProgress(i);
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    Log.d("Threading", e.getLocalizedMessage());
                }
                if (isCancelled()) break;
            }
            return null;
        }

        protected void onProgressUpdate(Integer... progress) {
            txtView1.setText(progress[0].toString());
            Log.d("Threading", "updating...");
        }
    }

    @Override
    protected void onPause() {
        super.onPause();
        stopCounter(txtView1);
    }
}
```

To stop the `AsyncTask` subclass, you need to get an instance of it first. To stop the task, call its `cancel()` method. Within the task, you call the `isCancelled()` method to check whether the task should be terminated.

SUMMARY

In this chapter, you learned how to create a service in your Android project to execute long-running tasks. You have seen the many approaches you can use to ensure that the background task is executed in an asynchronous fashion, without tying up the main calling activity. You have also learned how an activity can pass data into a service, and how you can alternatively bind to an activity so that it can access a service more directly.

EXERCISES

1. Why is it important to put long-running code in a service on a separate thread?
2. What is the purpose of the `IntentService` class?
3. Name the three methods you need to implement in an `AsyncTask` class.
4. How can a service notify an activity of an event happening?
5. For threading, what is the recommended method to ensure that your code runs without tying up the UI of your application?

Answers to the exercises can be found in Appendix C.

▶ WHAT YOU LEARNED IN THIS CHAPTER

TOPIC	KEY CONCEPTS
Creating a service	Create a class and extend the `Service` class.
Implementing the methods in a service	Implement the following methods: `onBind()`, `onStartCommand()`, and `onDestroy()`.
Starting a service	Use the `startService()` method.
Stopping a service	Use the `stopService()` method.
Performing long-running tasks	Use the `AsyncTask` class and implement three methods: `doInBackground()`, `onProgressUpdate()`, and `onPostExecute()`.
Performing repeated tasks	Use the `Timer` class and call its `scheduleAtFixedRate()` method.
Executing tasks on a separate thread and auto-stopping a service	Use the `IntentService` class.
Enabling communication between an activity and a service	Use the `Intent` object to pass data into the service. For a service, broadcast an `Intent` to notify an activity.
Binding an activity to a service	Use the `Binder` class in your service and implement the `ServiceConnection` class in your calling activity.
Updating the UI from a `Runnable` block	Use the `post()` method of a view to update the UI. Alternatively, you can also use a `Handler` class. The recommended way is to use the `AsyncTask` class.

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.