1

Android Fundamentals

In this chapter, you learn about the fundamental topics in Android that most developers need to know, including how to link to other applications using the Intent object, how to communicate with other applications (or parts of the same application) using broadcast receivers, and how to pass data between activities.

RECIPE 1.1 LINKING ACTIVITIES

Android Versions

Level 1 and above

Permissions

None

Source Code to Download at Wrox.com

Linking.zip

Unless you are writing a Hello World application, chances are good that your application contains several activities that you need to connect in order to form a cohesive application. This recipe shows you the various ways to link to another activity in your Android application.

Solution

Suppose you have two activities in your application. The following `AndroidManifest.xml` file shows the two activities classes, `MainActivity` and `Activity2`:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.learn2develop.linking"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="15" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >

        <activity
            android:name=".MainActivity"
            android:label="@string/title_activity_main" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

        <activity
            android:name=".Activity2"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="net.learn2develop.Activity2" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>

    </application>

</manifest>
```

Assuming you are currently in the `MainActivity` activity, to link to `Activity2` you can use the following code snippet:

```
package net.learn2develop.linking;

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

```
public class MainActivity extends Activity {

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

        //---link to Activity2---
        Intent i = new Intent("net.learn2develop.Activity2");
        startActivity(i);
    }

}
```

To link to another activity, you create an `Intent` object and set its constructor to the name (as set in the `<action>` element in the `AndroidManifest.xml` file) of the target activity. Then, call the `startActivity()` method to launch that activity.

Alternatively, you can create an `Intent` object and then call its `setAction()` method to set the name of the target activity:

```
//---link to Activity2---
Intent i = new Intent();
i.setAction("net.learn2develop.Activity2");
startActivity(i);
```

The previous code snippets are useful for calling an activity that is within the same application, as well as for other applications to call your activity. If you want to call an activity that is internal to your application, you can also call it using its class name, like this:

```
//---link to Activity2---
Intent i = new Intent(this, Activity2.class);
```

If you do not want other activities to call your activity from outside your application, simply remove the `<action>` element within the `<intent-filter>` element:

```
<activity
    android:name=".Activity2"
    android:label="@string/app_name" >
    <intent-filter>
        <!--
        <action android:name="net.learn2develop.Activity2" />
        -->
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

If the activity you are trying to call does not exist on the device, your application will crash, displaying a message like the one shown in Figure 1-1.

To ensure that your application does not stop abruptly, call the `startActivity()` method together with the `Intent.createChooser()` method. The `createChooser()` method takes an `Intent` object and a string to display if an activity cannot be found (or if more than one activity has been found to satisfy your `Intent` object):

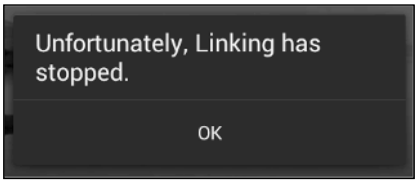

FIGURE 1-1

```
Intent i = new Intent("net.learn2develop.Activity2");
startActivity(Intent.createChooser(i, "Choose an application"));
```

Figure 1-2 shows the message that is displayed if an activity cannot be found.

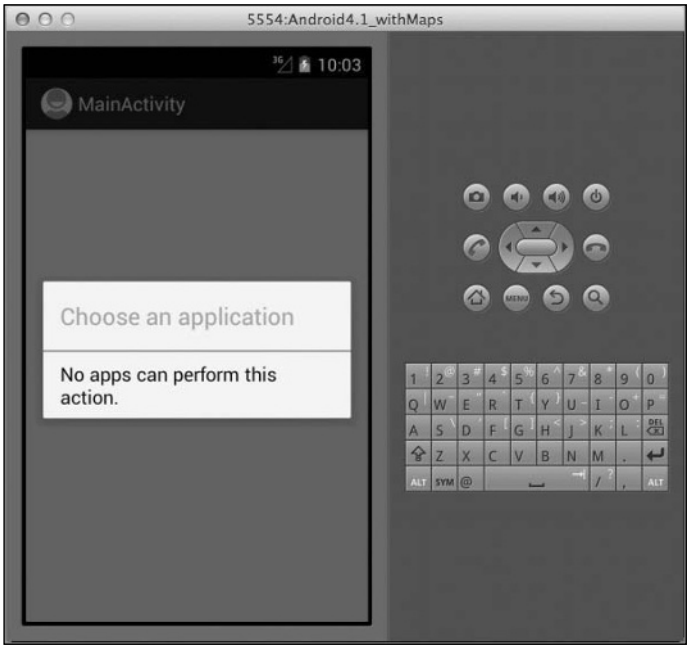

FIGURE 1-2

Figure 1-3 shows the message that is displayed when more than one activity has been found.

Note that when using the `createChooser()` method, you need to specify the name of the activity (such as `net.learn2develop.Activity2` as seen in the previous example) that you are launching, not its class name. The following code snippet will not work:

```
//---the following will never link to Activity2---
Intent i = new Intent(this, Activity2.class);
startActivity(Intent.createChooser(i, "Choose an application"));
```

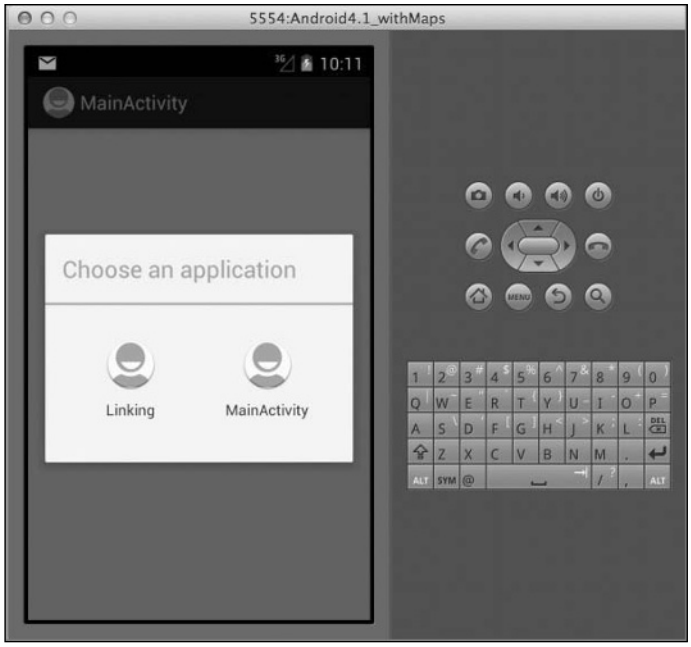

FIGURE 1-3

RECIPE 1.2 PASSING DATA BETWEEN ACTIVITIES

Android Versions

Level 1 and above

Permissions

None

Source Code to Download at Wrox.com

PassingData.zip

The preceding recipe demonstrated how to launch another activity using the `startActivity()` method. One common task you need to perform when launching another activity is passing some data to it. For example, you might want to launch another activity to collect some user-related data, so you pass the name of the user to another activity. When the user has finished collecting all the data, the data also needs to be passed back to the calling activity. Hence, you need to be able to pass data back and forth between activities. This recipe shows you how.

Solution

You can make use of the `Intent` class to pass data to another activity. To pass primitive data types to another activity, you can use the `putExtra()` method, as the following code snippet shows:

```
package net.learn2develop.passingdata;

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

public class MainActivity extends Activity {

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

    public void onClick(View view) {
        Intent i = new
                Intent("net.learn2develop.SecondActivity");

        //---use putExtra() to add new key/value pairs---
        i.putExtra("str1", "This is a string");
        i.putExtra("age1", 25);
    }
}
```

The preceding statements create an `Intent` object and then attach two values to it using the `putExtra()` method: one for a string and one for an integer.

You can also pass in a `Bundle` object using the `Intent` object. A `Bundle` object is like a dictionary object, as you can specify key/value pairs. To pass a `Bundle` object using an `Intent` object, create a `Bundle` object, populate it, and then attach it to the `Intent` object using the `putExtras()` method, as the following code shows:

```
public void onClick(View view) {
    Intent i = new
            Intent("net.learn2develop.SecondActivity");

    //---use putExtra() to add new key/value pairs---
    i.putExtra("str1", "This is a string");
    i.putExtra("age1", 25);

    //---use a Bundle object to add new key/values
    // pairs---
    Bundle extras = new Bundle();
    extras.putString("str2", "This is another string");
    extras.putInt("age2", 35);

    //---attach the Bundle object to the Intent object---
    i.putExtras(extras);
}
```

When you start another activity using the `Intent` object, the data attached to the `Intent` object is passed to the destination activity. To call another activity with the intention of getting some data back from it, use the `startActivityForResult()` method:

```
public void onClick(View view) {
    Intent i = new
            Intent("net.learn2develop.SecondActivity");

    //---use putExtra() to add new key/value pairs---
    i.putExtra("str1", "This is a string");
    i.putExtra("age1", 25);

    //---use a Bundle object to add new key/values
    // pairs---
    Bundle extras = new Bundle();
    extras.putString("str2", "This is another string");
    extras.putInt("age2", 35);

    //---attach the Bundle object to the Intent object---
    i.putExtras(extras);

    //---start the activity to get a result back---
    startActivityForResult(i, 1);
}
```

The `startActivityForResult()` method takes an `Intent` object as well as a request code. The request code is an integer value that is greater than or equal to zero. This request code is used to identify returning activities, as you may call more than one activity simultaneously. If you set the request code to -1, then the call of `startActivityForResult()` is equivalent to `startActivity()`. That is, you will not be able to obtain data passed back from the destination activity.

On the target activity, to retrieve the data that was passed to it, you use the `getIntent()` method to obtain the instance of the `Intent` object that was passed to it. To get the simple data type passed in through the `putExtra()` method, use the `get<type>Extra()` method, where the type may be `String`, `int`, `float`, and so on. The following code shows how the two primitive data types are retrieved:

```
package net.learn2develop.passingdata;

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

public class SecondActivity extends Activity {

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

        //---get the data passed in using getStringExtra()---
        Toast.makeText(this,getIntent().getStringExtra("str1"),
            Toast.LENGTH_SHORT).show();
```

```
        //---get the data passed in using getIntExtra()---
        Toast.makeText(this,Integer.toString(
            getIntent().getIntExtra("age1", 0)),
            Toast.LENGTH_SHORT).show();
    }

}
```

To retrieve the data passed in through the `Bundle` object, use the `getExtras()` method of the `Intent` object. The `getExtras()` method returns a `Bundle` object, which you can use to retrieve the various key/values using the `get<type>()` method, where type may be `String`, `int`, and so on:

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

    //---get the data passed in using getStringExtra()---
    Toast.makeText(this,getIntent().getStringExtra("str1"),
        Toast.LENGTH_SHORT).show();

    //---get the data passed in using getIntExtra()---
    Toast.makeText(this,Integer.toString(
        getIntent().getIntExtra("age1", 0)),
        Toast.LENGTH_SHORT).show();

    //---get the Bundle object passed in---
    Bundle bundle = getIntent().getExtras();

    //---get the data using the getString()---
    Toast.makeText(this, bundle.getString("str2"),
        Toast.LENGTH_SHORT).show();

    //---get the data using the getInt() method---
    Toast.makeText(this,Integer.toString(bundle.getInt("age2")),
        Toast.LENGTH_SHORT).show();
}
```

The destination may also pass data back to the calling activity. To pass data back, you can create another `Intent` object and set the values as described earlier. You can also use the `setData()` method to pass a `Uri` object through the `Intent` object. To pass the result back to the calling activity, use the `setResult()` method, as shown in the following code:

```
package net.learn2develop.passingdata;

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

public class SecondActivity extends Activity {
```

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

        //---get the data passed in using getStringExtra()---
        Toast.makeText(this,getIntent().getStringExtra("str1"),
            Toast.LENGTH_SHORT).show();

        //---get the data passed in using getIntExtra()---
        Toast.makeText(this,Integer.toString(
            getIntent().getIntExtra("age1", 0)),
            Toast.LENGTH_SHORT).show();

        //---get the Bundle object passed in---
        Bundle bundle = getIntent().getExtras();

        //---get the data using the getString()---
        Toast.makeText(this, bundle.getString("str2"),
            Toast.LENGTH_SHORT).show();

        //---get the data using the getInt() method---
        Toast.makeText(this,Integer.toString(bundle.getInt("age2")),
            Toast.LENGTH_SHORT).show();
    }

    public void onClick(View view) {
        //---use an Intent object to return data---
        Intent i = new Intent();

        //---use the putExtra() method to return some
        // value---
        i.putExtra("age3", 45);

        //---use the setData() method to return some value---
        i.setData(Uri.parse(
            "http://www.learn2develop.net"));

        //---set the result with OK and the Intent object---
        setResult(RESULT_OK, i);

        finish();
    }

}
```

The `RESULT_OK` constant enables you to indicate to the calling activity whether the data returned should be ignored. If you want the calling activity to ignore the result, you can use the `RESULT_CANCELLED` constant. How the calling activity interprets the result is really up to it, but the use of these two constants serves as an indication.

Back in the main calling activity, implement the `onActivityResult()` method. You need to check for the request code to ensure that you are getting the result from the correct activity. The request

code is the number that you earlier passed to the `startActivityForResult()` method, which is 1 in this example:

```
        //---start the activity to get a result back---
        startActivityForResult(i, 1);
```

You can also check the result code to see if it is `RESULT_OK`:

```
package net.learn2develop.passingdata;

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

public class MainActivity extends Activity {

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

    public void onClick(View view) {
        Intent i = new
                Intent("net.learn2develop.SecondActivity");

        //---use putExtra() to add new key/value pairs---
        i.putExtra("str1", "This is a string");
        i.putExtra("age1", 25);

        //---use a Bundle object to add new key/values
        // pairs---
        Bundle extras = new Bundle();
        extras.putString("str2", "This is another string");
        extras.putInt("age2", 35);

        //---attach the Bundle object to the Intent object---
        i.putExtras(extras);

        //---start the activity to get a result back---
        startActivityForResult(i, 1);
    }

    public void onActivityResult(int requestCode,
            int resultCode, Intent data)
    {
        //---check if the request code is 1---
        if (requestCode == 1) {

            //---if the result is OK---
            if (resultCode == RESULT_OK) {
```

```
                //---get the result using getIntExtra()---
                Toast.makeText(this, Integer.toString(
                        data.getIntExtra("age3", 0)),
                        Toast.LENGTH_SHORT).show();

                //---get the result using getData()---
                Uri url = data.getData();
                Toast.makeText(this, url.toString(),
                        Toast.LENGTH_SHORT).show();
            }
        }
    }
}
```

To retrieve the data sent using the `setData()` method, use the `getData()` method of the `Intent` object (passed in as the second argument of the `onActivityResult()` method).

RECIPE 1.3 PASSING OBJECTS BETWEEN ACTIVITIES

Android Versions

Level 1 and above

Permissions

None

Source Code to Download at Wrox.com

PassingData.zip

In the previous recipe, you saw how to pass simple data (such as strings and integers) between activities. This recipe demonstrates how to pass objects between activites. For example, you might have encapsulated the information of a customer (such as the customer ID, name, company, and so on) within an object and you need to pass it over to another activity for processing. Instead of passing the various pieces of the information of the customer individually, it would be easier to simply pass that object.

Solution

Besides passing simple data types using the `putExtra()` and `putExtras()` methods, you can also pass objects using an `Intent` object. If you have your own custom class, you need to ensure that your class implements the `Serializable` base class. The following `MyCustomClass` class is an example:

```
package net.learn2develop.passingdata;

import java.io.Serializable;
```

```
public class MyCustomClass implements Serializable {
    private static final long serialVersionUID = 1L;
    String _name;
    String _email;

    public void setName(String name) {
        _name = name;
    }

    public String Name() {
        return _name;
    }

    public void setEmail(String email) {
        _email = email;
    }

    public String Email() {
        return _email;
    }
}
```

To pass an object to another activity, use the `putExtra()` method:

```
public void onClick(View view) {
    Intent i = new
            Intent("net.learn2develop.SecondActivity");

    //---use putExtra() to add new key/value pairs---
    i.putExtra("str1", "This is a string");
    i.putExtra("age1", 25);

    //---use a Bundle object to add new key/values
    // pairs---
    Bundle extras = new Bundle();
    extras.putString("str2", "This is another string");
    extras.putInt("age2", 35);

    //---attach the Bundle object to the Intent object---
    i.putExtras(extras);

    //---create my own custom object---
    MyCustomClass myObject = new MyCustomClass();
    myObject.setName("Wei-Meng Lee");
    myObject.setEmail("weimenglee@learn2develop.net");
    i.putExtra("MyObject", myObject);

    //---start the activity to get a result back---
    startActivityForResult(i, 1);
}
```

To retrieve the object passed to another activity, use the `getSerializableExtra()` method of the `Intent` object, passing it the key that you set earlier in the `putExtra()` method. Then,

typecast the result returned by this method to the `MyCustomClass` class and assign it to a variable of this type:

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

    //---get the data passed in using getStringExtra()---
    Toast.makeText(this,getIntent().getStringExtra("str1"),
        Toast.LENGTH_SHORT).show();

    //---get the data passed in using getIntExtra()---
    Toast.makeText(this,Integer.toString(
        getIntent().getIntExtra("age1", 0)),
        Toast.LENGTH_SHORT).show();

    //---get the Bundle object passed in---
    Bundle bundle = getIntent().getExtras();

    //---get the data using the getString()---
    Toast.makeText(this, bundle.getString("str2"),
        Toast.LENGTH_SHORT).show();

    //---get the data using the getInt() method---
    Toast.makeText(this,Integer.toString(bundle.getInt("age2")),
        Toast.LENGTH_SHORT).show();

    //---get the custom object passed in---
    MyCustomClass obj = (MyCustomClass)
        getIntent().getSerializableExtra("MyObject");
    Toast.makeText(this, obj.Name(), Toast.LENGTH_SHORT).show();
    Toast.makeText(this, obj.Email(), Toast.LENGTH_SHORT).show();
}
```

RECIPE 1.4 SENDING AND RECEIVING BROADCASTS

Android Versions

Level 1 and above

Permissions

None

Source Code to Download at Wrox.com

UsingBroadcastReceiver.zip

In Android, a broadcast enables you to send a message to another part of your application (or another application) so that you can inform it of something happening. In this recipe, you learn how to create a broadcast receiver to listen for broadcasts, as well as send broadcasts to other applications.

Solution

There are two ways to create a broadcast receiver: programmatically through code and declaratively via the `AndroidManifest.xml` file. The following sections address each possible solution.

Programmatically Registering a Broadcast Receiver

Consider the following activity:

```
package net.learn2develop.usingbroadcastreceiver;

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 MainActivity extends Activity {
    MyBroadcastReceiver myReceiver;
    IntentFilter intentFilter;

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

        myReceiver = new MyBroadcastReceiver();
        intentFilter = new IntentFilter("MY_SPECIFIC_ACTION");
    }

    @Override
    public void onResume() {
        super.onResume();
        //---register the receiver---
        registerReceiver(myReceiver, intentFilter);
    }

    @Override
    public void onPause() {
        super.onPause();
        //---unregister the receiver---
        unregisterReceiver(myReceiver);
    }
```

```
    public void onClick(View view) {
        Intent i = new Intent("MY_SPECIFIC_ACTION");
        i.putExtra("key", "some value from intent");
        sendBroadcast(i);
    }

    public class MyBroadcastReceiver extends BroadcastReceiver {
        @Override
        public void onReceive(Context context, Intent i) {
            Toast.makeText(context,
                    "Received broadcast in MyBroadcastReceiver, " +
                    " value received: " + i.getStringExtra("key"),
                    Toast.LENGTH_LONG).show();
        }
    }

}
```

The preceding code snippet shows the inner class `MyBroadcastReceiver` extending from the `BroadcastReceiver` base class. In this class, you need to override the `onReceive()` method so that when the broadcast is received, you can perform the action that you want to perform. To get the data that is passed to the receiver, you can make use of the `Intent` object in the second argument of the `onReceive()` method.

To use this class, you need to create an instance of it, as well as create an `IntentFilter` object:

```
myReceiver = new MyBroadcastReceiver();
intentFilter = new IntentFilter("MY_SPECIFIC_ACTION");
```

You specify a user-defined action in the `IntentFilter`'s constructor, and use your own string to define this action.

To register the `BroadcastReceiver` object, use the `registerReceiver()` method, passing it the `BroadcastReceiver` object as well as the `IntentFilter` object:

```
registerReceiver(myReceiver, intentFilter);
```

Now that you have registered a `BroadcastReceiver` object, you can send a broadcast to test whether it works. To send a broadcast, you use the `sendBroadcast()` method, passing it an `Intent` object:

```
public void onClick(View view) {
    Intent i = new Intent("MY_SPECIFIC_ACTION");
    i.putExtra("key", "some value from intent");
    sendBroadcast(i);
}
```

If you want to pass data to the receiver, you can use the `putExra()` method. To unregister the broadcast receiver, use the `unregisterReceiver()` method:

```
unregisterReceiver(myReceiver);
```

Figure 1-4 shows the receiver receiving the broadcast.

FIGURE 1-4

The broadcast receiver will work even if the broadcast was sent by another application.

Registering the BroadcastReceiver in the AndroidManifest.xml File

In the previous example, if the application is in the background, the broadcast receiver will no longer work because you have unregistered the broadcast receiver when the application goes to the background:

```
@Override
public void onPause() {
    super.onPause();
    //---unregister the receiver---
    unregisterReceiver(myReceiver);
}
```

If you want a more persistent way to receive broadcasts, you need to register the `BroadcastReceiver` class in the `AndroidManifest.xml` file.

To do so, you create the `BroadcastReceiver` class in another Java class. The following code snippet shows the content of the `MySecondBroadcastReceiver.java` file:

```
package net.learn2develop.usingbroadcastreceiver;

import android.content.BroadcastReceiver;
import android.content.Context;
```

```
import android.content.Intent;
import android.widget.Toast;

public class MySecondBroadcastReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent i) {
        Toast.makeText(context,
                "Received broadcast in MySecondBroadcastReceiver; " +
                " value received: " + i.getStringExtra("key"),
                Toast.LENGTH_LONG).show();
    }
}
```

To register this receiver in the `AndroidManifest.xml` file, add the `<receiver>` element:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.learn2develop.usingbroadcastreceiver"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="15" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/title_activity_main" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

        <receiver android:name=".MySecondBroadcastReceiver" >
            <intent-filter>
                <action android:name="MY_SPECIFIC_ACTION" />
            </intent-filter>
        </receiver>

    </application>

</manifest>
```

Your application now has two `BroadcastReceiver` objects: one you registered programmatically in the `onResume()` method and one in the `AndroidManifest.xml` file. If you send a broadcast now, both receivers will be called. Figure 1-5 shows the second broadcast receiver being called.

FIGURE 1-5

RECIPE 1.5 ASSIGNING PRIORITIES TO BROADCAST RECEIVERS

Android Versions

Level 1 and above

Permissions

None

Source Code to Download at Wrox.com

UsingBroadcastReceiver.zip

When you send a broadcast using the `sendBroadcast()` method, all the broadcast receivers that match the specified action are called in random fashion. What if you want to assign a particular order to the broadcast receivers so that some broadcast receivers will be called before others? To do that, you need to assign a priority to the broadcast receivers.

Solution

To programmatically assign a priority to a broadcast receiver, use the `setPriority()` method:

```
public class MainActivity extends Activity {
    MyBroadcastReceiver myReceiver;
    IntentFilter intentFilter;

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

        myReceiver = new MyBroadcastReceiver();
        intentFilter = new IntentFilter("MY_SPECIFIC_ACTION");
    }

    @Override
    public void onResume() {
        super.onResume();
        intentFilter.setPriority(10);
        registerReceiver(myReceiver, intentFilter);
    }
```

The `setPriority()` method takes a priority value between 0 (default) and 1,000. The larger the number, the higher priority it has, and hence broadcast receivers with a higher priority are called before those with lower priority. If more than one broadcast receiver has the same priority, they are called randomly. The preceding code snippet sets the priority to 10.

To set the priority of broadcast receivers in the `AndroidManifest.xml` file, use the `android:priority` attribute:

```
        <receiver android:name=".MySecondBroadcastReceiver" >
            <intent-filter android:priority="50">
                <action android:name="MY_SPECIFIC_ACTION" />
            </intent-filter>
        </receiver>
```

The preceding example sets the priority to 50.

To send a broadcast that is delivered to broadcast receivers with higher priority first, you cannot use the `sendBroadcast()` method. Instead, you need to use the `sendOrderedBroadcast()` method, passing it an `Intent` object, plus any additional permission that the receiver must have in order to receive your broadcast:

```
    public void onClick(View view) {
        Intent i = new Intent("MY_SPECIFIC_ACTION");
        i.putExtra("key", "some value from intent");
        //sendBroadcast(i);

        //---allows broadcast to be aborted---
        //---allows broadcast receivers to set priority---
        sendOrderedBroadcast(i, null);
    }
```

If you try to send the broadcast now, you will notice that the broadcast receiver declared in the `AndroidManifest.xml` file is called first, before the one declared programmatically through code.

If you want to send a broadcast to only broadcast receivers with the permission to access the Internet, you will specify the permission in the second argument of the `sendOrderedBroadcast()` method, like this:

```
sendOrderedBroadcast(i, "android.permission.INTERNET");
```

Aborting a Broadcast

When broadcasts are sent using the `sendOrderedBroadcast()` method, broadcast receivers are called in the order of the priorities defined. When a broadcast receiver of higher priority receives the broadcast, it will handle that and the broadcast will be passed to the next broadcast receiver in line. In some scenarios, you might want to handle the broadcast and stop the broadcast from being propagated to the next receiver. To do that, you can use the `abortBroadcast()` method:

```
package net.learn2develop.UsingBroadcastReceiver;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.widget.Toast;

public class MySecondBroadcastReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent i) {
        Toast.makeText(context,
                "Received broadcast in MySecondBroadcastReceiver; " +
                "value received: " + i.getStringExtra("key"),
                 Toast.LENGTH_LONG).show();
        //---abort the broadcast----
        abortBroadcast();
    }
}
```

In the preceding code snippet, the `MySecondBroadcastReceiver` class aborts the broadcast after receiving it. Once it aborts the broadcast, other receivers who are waiting in line will not be able to receive it.

> **NOTE** *In order to call the* `abortBroadcast()` *method to abort a broadcast, you need to send the broadcast using the* `sendOrderedBroadcast()` *method. Using the* `sendBroadcast()` *method has no effect on the priority and will not cause a broadcast to be aborted.*

RECIPE 1.6 AUTO-LAUNCHING YOUR APPLICATION AT BOOT TIME

Android Versions

Level 1 and above

Permissions

android.permission.RECEIVE_BOOT_COMPLETED

Source Code to Download at Wrox.com

AutoStartApp.zip

If you need to automatically start your application whenever the device starts up, you need to register a `BroadcastReceiver`. This recipe shows you how.

Solution

To auto-launch your app during device boot-up, add a new class to your package and ensure that it extends the `BroadcastReceiver` base class. The following `BootupReceiver` class is an example:

```
package net.learn2develop.autostartapp;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.widget.Toast;

public class BootupReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        Toast.makeText(context, "App started", Toast.LENGTH_LONG).show();

        //---start the main activity of our app---
        Intent i = new Intent(context,MainActivity.class);
        i.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
        context.startActivity(i);
    }
}
```

When the device boots up, it will fire this broadcast receiver and call the `onReceiver()` method. To display your activity when the device boots up, you will use an `Intent` object. Remember to add the `FLAG_ACTIVITY_NEW_TASK` flag to the `Intent` object.

To register the broadcast receiver, you need to add the `<receiver>` element to the `AndroidManifest.xml` file. You also need the `RECEIVE_BOOT_COMPLETED` permission:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.learn2develop.autostartapp"
```

```
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="15" />

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

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/title_activity_main" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

        <receiver android:name=".BootupReceiver">
            <intent-filter>
                <action android:name="android.intent.action.BOOT_COMPLETED" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </receiver>

    </application>

</manifest>
```

Your application will now be automatically launched when the device has booted up.

RECIPE 1.7 CALLING BUILT-IN APPS

Android Versions

Level 1 and above

Permissions

None

Source Code to Download at Wrox.com

CallingApps.zip

One of the key features of Android is the functionality it provides for applications to call other applications seamlessly. This enable you to integrate various applications on the device to form a coherent experience for your users. This recipe shows you the various ways to call the applications on your device.

Solution

There are many ways to call built-in apps, and how depends on which application you are calling. For this recipe's solution, you will learn how to call some of the commonly installed applications on your Android device, such as:

- How to display maps
- How to direct the user to a particular application on Google Play
- How to send e-mails
- How to send text and graphic content to applications that can handle them

Displaying Maps

To display maps in your application, you can launch an activity using the `geo:` scheme, as shown in here:

```
package net.learn2develop.callingapps;

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

public class MainActivity extends Activity {

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

        Intent i = new Intent(android.content.Intent.ACTION_VIEW);
        i.setData(Uri.parse("geo:37.827500,-122.481670"));
        startActivity(i);
    }

}
```

Figure 1-6 shows the application displaying a list of applications able to handle the `geo:` scheme when the preceding code is run on an Android device (you may see more on your device).

To launch an application from the list, select the application (say, Google Earth) and select either Always or Just once. If you select Always, the application you have just selected (in this case, Google Earth) will always be launched automatically. If you select Just once, you will see this prompt (asking if you want to launch it Always or Just once) each time you run this application.

When you select the Earth application, the Google Earth application will launch (see Figure 1-7).

Similarly, selecting the Maps application launches the Google Maps application (see Figure 1-8).

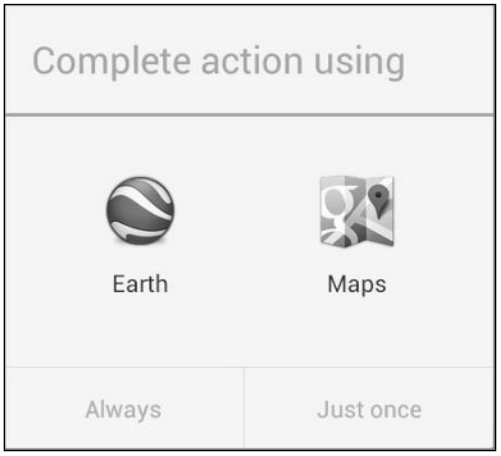

FIGURE 1-6

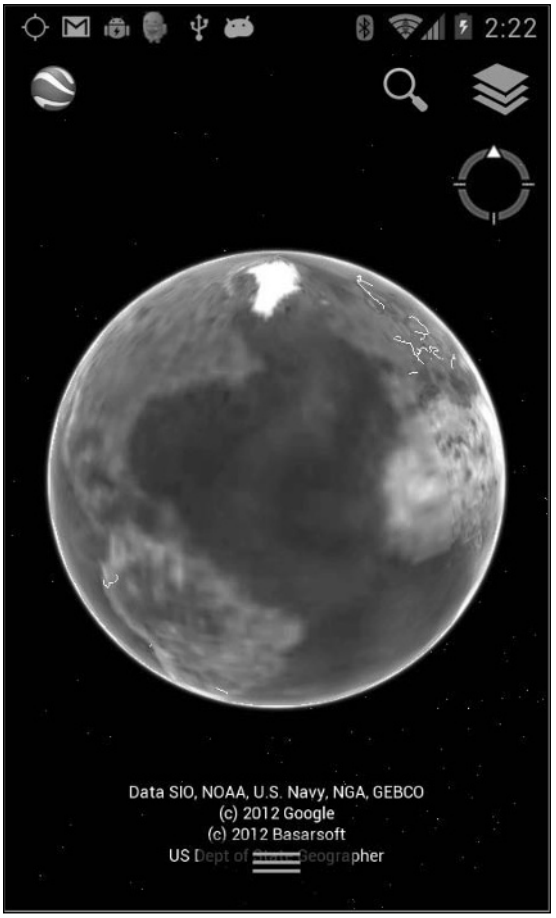

FIGURE 1-7

FIGURE 1-8

Launching Google Play

If you want to redirect the user to another application that is available on Google Play (formerly known as Google Market), use the `market:` scheme:

```
Intent i = new Intent(android.content.Intent.ACTION_VIEW);
i.setData(Uri.parse(
    "market://details?id=com.zinio.mobile.android.reader"));
startActivity(i);
```

The preceding code snippet will display the Zinio application available on Google Play (see Figure 1-9).

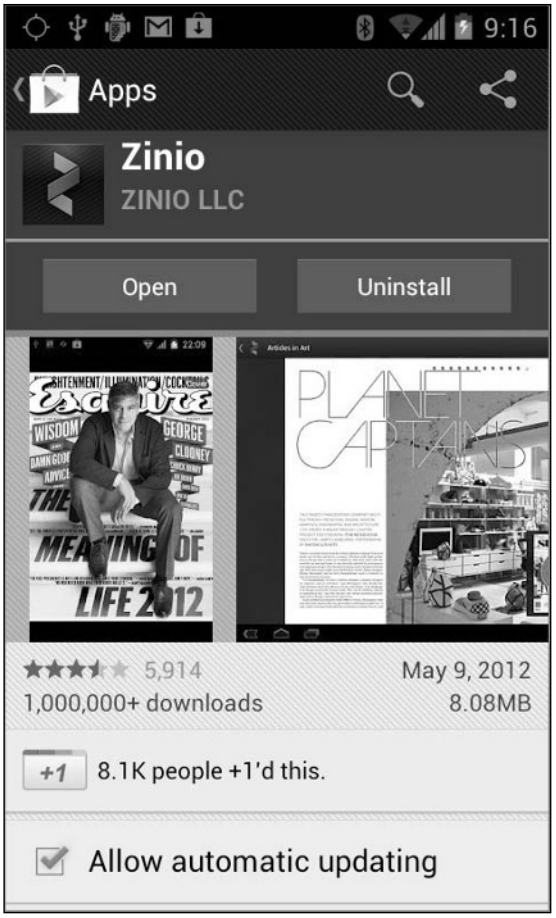

FIGURE 1-9

Sending E-mail

To send an e-mail message from within your application, use the following code snippet:

```
Intent i = new Intent(Intent.ACTION_SEND);
i.setData(Uri.parse("mailto:"));
String[] to = { "someone1@example.com" , "someone2@example.com" };
String[] cc = { "someone3@example.com" , "someone4@example.com" };
i.putExtra(Intent.EXTRA_EMAIL, to);
i.putExtra(Intent.EXTRA_CC, cc);
i.putExtra(Intent.EXTRA_SUBJECT, "Subject here...");
i.putExtra(Intent.EXTRA_TEXT, "Message here...");
i.setType("message/rfc822");
startActivity(Intent.createChooser(i, "Email"));
```

Figure 1-10 shows the E-mail application displaying the content of the e-mail.

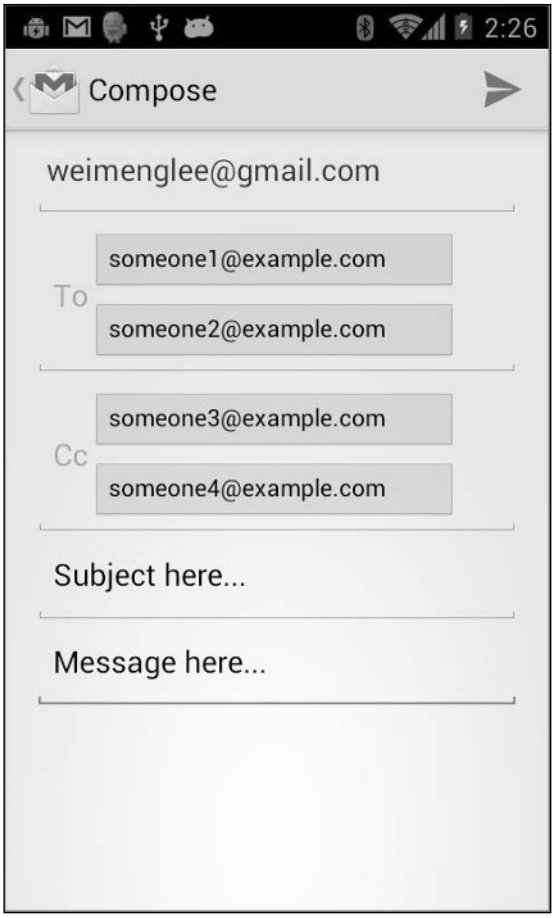

FIGURE 1-10

Sending Content to Other Apps

Occasionally, you may want to launch another activity and send some content to it. For example, you might want to send some content to Facebook as well as the E-mail application. In this case, instead of targeting a particular application to invoke, you can use the generic `ACTION_SEND` constant to invoke a list of applications from which to choose. Consider the following code snippet:

```
Intent i = new Intent(android.content.Intent.ACTION_SEND);
i.setType("text/plain");
i.putExtra(Intent.EXTRA_SUBJECT, "Subject...");
i.putExtra(Intent.EXTRA_TEXT, "Text...");
startActivity(Intent.createChooser(i, "Apps that can respond to this"));
```

When run on a real device, the preceding code might invoke the list of applications shown in Figure 1-11.

If you selected the Messaging app, the data you set in the `Intent` object will then be sent as an SMS message (see Figure 1-12).

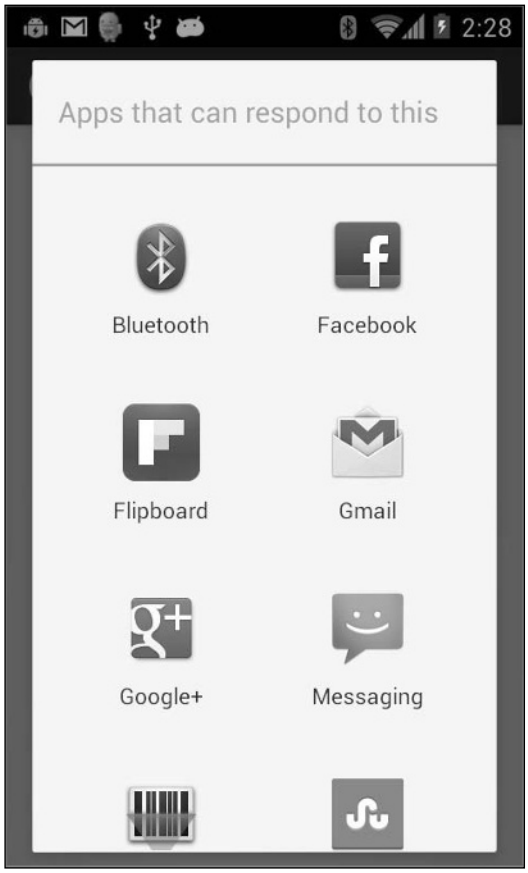

FIGURE 1-11

FIGURE 1-12

If you selected Gmail, the data you set in the `Intent` object will then be sent as an e-mail (see Figure 1-13).

If you selected Twitter (assuming you have Twitter installed on your device), the data you set in the `Intent` object will then be sent as a tweet (see Figure 1-14).

FIGURE 1-13

FIGURE 1-14

Sending Binary Content

If you have some images in your `drawable` folders, you can also send them to other applications using the following code snippet:

```
//---sending binary content---
Uri uriToImage =
    Uri.parse(
    "android.resource://net.learn2develop.CallingApps/drawable/" +
    Integer.toString(R.drawable.android));
```

```
Intent i = new Intent(android.content.Intent.ACTION_SEND);
i.setType("image/jpeg");
i.putExtra(Intent.EXTRA_STREAM, uriToImage);
i.putExtra(Intent.EXTRA_TEXT, "Text...");
startActivity(Intent.createChooser(i, "Apps that can respond to this"));
```

The preceding code assumes that you have a file named `android.jpg` located in one of the `drawable` folders in your project (see Figure 1-15). You set the image type to `image/jpeg` and then use the `putExtra()` method to put the image into the `Intent` object.

If you send the data to Twitter, the image will be used by Twitter as part of your tweet (see Figure 1-16).

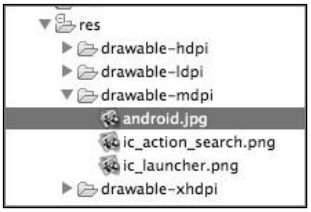

FIGURE 1-15

FIGURE 1-16

To send multiple images, you can use the following code snippet:

```
import java.util.ArrayList;
...
...
        Uri uriToImage1 =
            Uri.parse(
            "android.resource://net.learn2develop.CallingApps/drawable/" +
            Integer.toString(R.drawable.android));
        Uri uriToImage2 =
            Uri.parse(
            "android.resource://net.learn2develop.CallingApps/drawable/" +
            Integer.toString(R.drawable.google));

        ArrayList<Uri> urisToImages = new ArrayList<Uri>();
        urisToImages.add(uriToImage1);
        urisToImages.add(uriToImage2);
```

```
Intent i = new Intent(android.content.Intent.ACTION_SEND_MULTIPLE);
i.setType("image/*");
i.putExtra(Intent.EXTRA_STREAM, urisToImages);
i.putExtra(Intent.EXTRA_SUBJECT, "Subject...");
i.putExtra(Intent.EXTRA_TEXT, "Text...");
startActivity(Intent.createChooser(i, "Apps that can respond to this"));
```

Figure 1-17 shows the two images sent to the Gmail application.

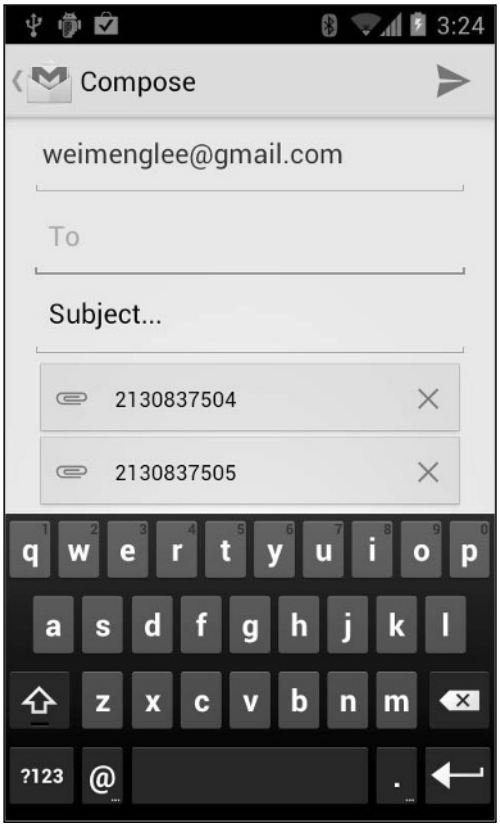

FIGURE 1-17

If you simply want to launch an application that enables you to view images, use the following code snippet:

```
Intent i = new Intent(android.content.Intent.ACTION_VIEW);
//---indicates the type that the target activity will handle---
i.setType("image/jpeg");
startActivity(i);
```

The preceding code snippet will list all the applications that allow you to view images (see Figure 1-18).

If you want to send an image (say located on the SD card) to an image viewer application, you can use the following code snippet:

```
Intent i = new Intent(android.content.Intent.ACTION_VIEW);
i.setDataAndType(
    Uri.parse("file:///storage/sdcard0/MyPhoto.jpg"), "image/*");
startActivity(i);
```

Figure 1-19 shows the Galley Viewer displaying the image located on the SD card.

FIGURE 1-18

FIGURE 1-19

RECIPE 1.8 MAKING YOUR APPLICATION CALLABLE BY OTHERS

Android Versions

Level 1 and above

Permissions

android.permission.INTERNET

Source Code to Download at Wrox.com

IntentFilters.zip

In all the previous recipes you have launched other apps by using the `Intent` object. What about your own apps? How do you allow other apps to call yours using the `Intent` object? This recipe shows you how.

Solution

Assume you have the following class in your application:

```
package net.learn2develop.intentfilters;

import android.app.Activity;
import android.net.Uri;
import android.os.Bundle;
import android.webkit.WebView;
import android.webkit.WebViewClient;

public class MyBrowserActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.browser);

        Uri url = getIntent().getData();
        WebView webView = (WebView) findViewById(R.id.webView);
        webView.setWebViewClient(new Callback());

        if (url!=null) {
            webView.loadUrl(url.toString());
        } else {
            webView.loadUrl("http://www.google.com");
        }
    }
```

```
        private class Callback extends WebViewClient {
            @Override
            public boolean shouldOverrideUrlLoading(
                    WebView view, String url) {
                return(false);
            }
        }
    }
```

Basically, the preceding class loads an XML file named `browser.xml` (shown next) that contains a `WebView`. It displays a web page based on the data passed in through the `Intent` object:

```
    <?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" >

        <WebView
            android:id="@+id/webView"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />

    </LinearLayout>
```

The class is also declared in your `AndroidManifest.xml` file as follows:

```
    <manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="net.learn2develop.intentfilters"
        android:versionCode="1"
        android:versionName="1.0" >

        <uses-sdk
            android:minSdkVersion="8"
            android:targetSdkVersion="15" />

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

        <application
            android:icon="@drawable/ic_launcher"
            android:label="@string/app_name"
            android:theme="@style/AppTheme" >
            <activity
                android:name=".MainActivity"
                android:label="@string/title_activity_main" >
                <intent-filter>
                    <action android:name="android.intent.action.MAIN" />

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

```
        <activity
            android:name=".MyBrowserActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.VIEW" />
                <action android:name="net.learn2develop.MyBrowser" />
                <category android:name="android.intent.category.DEFAULT" />
                <data android:scheme="http" />
            </intent-filter>
        </activity>

    </application>

</manifest>
```

Note the elements contained within the `<intent-filter>` element for the `MyBrowserActivity` class. It has two actions: `android.intent.action.VIEW` and `net.learn2develop.MyBrowser`. This means that it can be called using the `android.intent.action.VIEW` action constant, or directly using the `net.learn2develop.MyBrowser` name. It also specifies the data scheme of `http:`. This means that this activity requires data passed to it to have the `http:` prefix, such as `http://www.amazon.com`, `http://www.google.com`, and `http://www.wrox.com`.

To invoke the `MyBrowserActivity` class, you can use the following statements in your main activity:

```
package net.learn2develop.intentfilters;

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

public class MainActivity extends Activity {

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

        Intent i = new Intent("net.learn2develop.MyBrowser");
        i.setData(Uri.parse("http://www.amazon.com"));
        startActivity(i);
    }

}
```

Figure 1-20 shows the `MyBrowserActivity` class displaying the Amazon.com site. In this case, you are directly calling the `MyBrowserActivity` class using its "`net.learn2develop.MyBrowser`" action.

FIGURE 1-20

Alternatively, you can use the `android.content.Intent.ACTION_VIEW` constant (which evaluates to the same value as "`android.intent.action.VIEW`") to call it. However, this time you will see a dialog asking you to choose an application to complete the action (see Figure 1-21):

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

This is because now more than one application can handle this action.

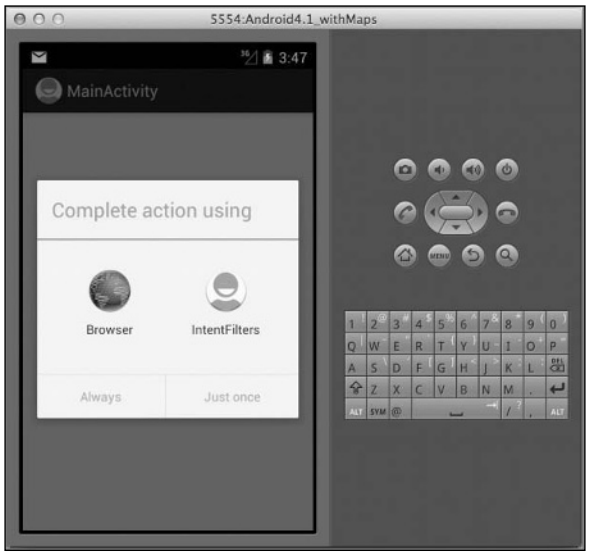

FIGURE 1-21

Now go to the `AndroidManifest.xml` file and modify the `<data>` element:

```
<activity
    android:name=".MyBrowserActivity"
    android:label="@string/app_name" >
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <action android:name="net.learn2develop.MyBrowser" />
        <category android:name="android.intent.category.DEFAULT" />
        <!--
           <data android:scheme="http" />
        -->
        <data android:mimeType="text/html" />
    </intent-filter>
</activity>
```

The `<data>` element in this case specifies the MIME media type that the activity is capable of handling.

To call the activity with this MIME type, you have to use the `setType()` method:

```
Intent i = new Intent(android.content.Intent.ACTION_VIEW);

//---if you are using setType(), no need to use setData()---
//i.setData(Uri.parse("http://www.amazon.com"));

//---indicates the type that the target activity will handle---
i.setType("text/html");
i.putExtra("URL", "http://www.amazon.com");
startActivity(i);
```

Note that the `setType()` method automatically clears any data you set using the `setData()` method; hence, you do not need to use the `setData()` method. To pass data to the activity in this case, you can use the `putExtra()` method.

The preceding code snippet will show a list of applications that are capable of handling the MIME type you specified (see Figure 1-22).

To retrieve the data passed in using the `putExtra()` method, you can use the `getStringExtra()` method:

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

    //Uri url = getIntent().getData();
    Uri url = Uri.parse(getIntent().getStringExtra("URL"));

    WebView webView = (WebView) findViewById(R.id.webView);
    webView.setWebViewClient(new Callback());
```

```
        if (url!=null) {
            webView.loadUrl(url.toString());
        } else {
            webView.loadUrl("http://www.google.com");
        }
    }
```

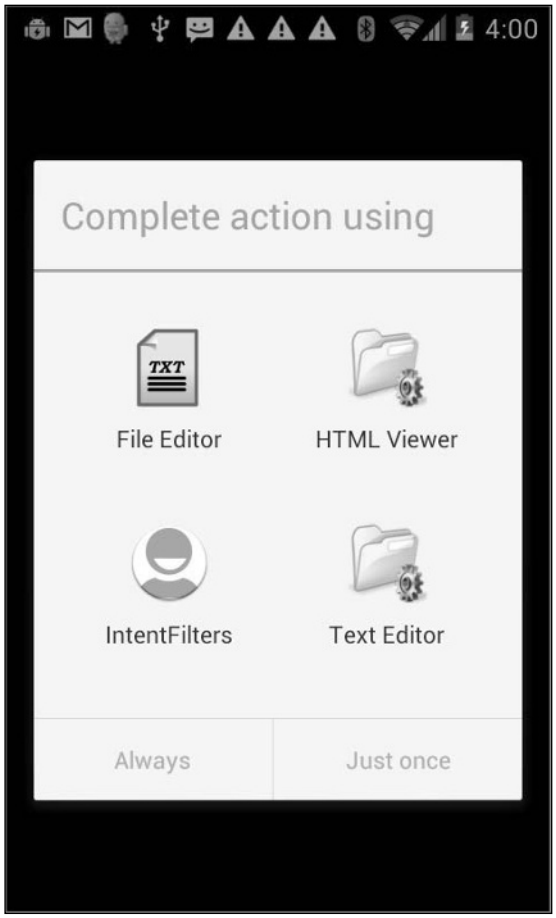

FIGURE 1-22