42. An Overview of Android Intents

By this stage of the book, it should be clear that Android applications are comprised, among other things, of one or more activities. An area that has yet to be covered in extensive detail, however, is the mechanism by which one activity can trigger the launch of another activity. As outlined briefly in the chapter entitled *"The Anatomy of an Android Application"*, this is achieved primarily by using *Intents*.

Prior to working through some Android Studio based example implementations of intents in the following chapters, the goal of this chapter is to provide an overview of intents in the form of *explicit intents* and *implicit intents* together with an introduction to *intent filters*.

42.1 An Overview of Intents

Intents (*android.content.Intent*) are the messaging system by which one activity is able to launch another activity. An activity can, for example, issue an intent to request the launch of another activity contained within the same application. Intents also, however, go beyond this concept by allowing an activity to request the services of any other appropriately registered activity on the device for which permissions are configured. Consider, for example, an activity contained within an application that requires a web page to be loaded and displayed to the user. Rather than the application having to contain a second activity to perform this task, the code can simply send an intent to the Android runtime requesting the services of any activity that has registered the ability to display a web page. The runtime system will match the request to available activities on the device and either launch the activity that matches or, in the event of multiple matches, allow the user to decide which activity to use.

Intents also allow for the transfer of data from the sending activity to the receiving activity. In the previously outlined scenario, for example, the sending activity would need to send the URL of the web page to be displayed to the second activity. Similarly, the receiving activity may also be configured to return data to the sending activity when the required tasks are completed.

Though not covered until later chapters, it is also worth highlighting the fact that, in addition to launching activities, intents are also used to launch and

communicate with services and broadcast receivers.

Intents are categorized as either *explicit* or *implicit*.

42.2 Explicit Intents

An *explicit intent* requests the launch of a specific activity by referencing the *component name* (which is actually the class name) of the target activity. This approach is most common when launching an activity residing in the same application as the sending activity (since the class name is known to the application developer).

An explicit intent is issued by creating an instance of the Intent class, passing through the activity context and the component name of the activity to be launched. A call is then made to the *startActivity()* method, passing the intent object as an argument. For example, the following code fragment issues an intent for the activity with the class name ActivityB to be launched:

```
Intent i = new Intent(this, ActivityB.class);
startActivity(i);
```

Data may be transmitted to the receiving activity by adding it to the intent object before it is started via calls to the *putExtra()* method of the intent object. Data must be added in the form of key-value pairs. The following code extends the previous example to add String and integer values with the keys "myString" and "myInt" respectively to the intent:

```
Intent i = new Intent(this, ActivityB.class);
i.putExtra("myString", "This is a message for ActivityB");
i.putExtra("myInt", 100);

startActivity(i);
```

The data is received by the target activity as part of a Bundle object which can be obtained via a call to *getIntent().getExtras()*. The *getIntent()* method of the Activity class returns the intent that started the activity, while the *getExtras()* method (of the Intent class) returns a Bundle object containing the data. For example, to extract the data values passed to ActivityB:

```
Bundle extras = getIntent().getExtras();

if (extras != null) {
    String myString = extras.getString("myString");
    int myInt = extras.getInt("myInt");
}
```

When using intents to launch other activities within the same application, it is essential that those activities be listed in the application manifest file. The following *AndroidManifest.xml* contents are correctly configured for an application containing activities named ActivityA and ActivityB:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.ebookfrenzy.intent1.intent1" >

    <application
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:label="@string/app_name"
            android:name="com.ebookfrenzy.intent1.intent1.ActivityA" >
            <intent-filter>
              <action android:name="android.intent.action.MAIN" />
              <category android:name="android.intent.category.LAUNCHER
            </intent-filter>
        </activity>
        <activity
            android:name="ActivityB"
            android:label="ActivityB" >
        </activity>
    </application>
</manifest>
```

42.3 Returning Data from an Activity

As the example in the previous section stands, while data is transferred to ActivityB, there is no way for data to be returned to the first activity (which we will call ActivityA). This can, however, be achieved by launching ActivityB as a *sub-activity* of ActivityA. An activity is started as a sub-activity by starting the intent with a call to the *startActivityForResult()* method instead of using *startActivity()*. In addition to the intent object, this method is also passed a *request code* value which can be used to identify the return data when the sub-activity returns. For example:

```
startActivityForResult(i, REQUEST_CODE);
```

In order to return data to the parent activity, the sub-activity must implement the *finish()* method, the purpose of which is to create a new intent object containing the data to be returned, and then calling the *setResult()* method of

the enclosing activity, passing through a *result code* and the intent containing the return data. The result code is typically *RESULT_OK*, or *RESULT_CANCELED*, but may also be a custom value subject to the requirements of the developer. In the event that a sub-activity crashes, the parent activity will receive a *RESULT_CANCELED* result code.

The following code, for example, illustrates the code for a typical sub-activity *finish()* method:

```
public void finish() {
       Intent data = new Intent();

       data.putExtra("returnString1", "Message to parent activity");
       setResult(RESULT_OK, data);
       super.finish();
}
```

In order to obtain and extract the returned data, the parent activity must implement the *onActivityResult()* method, for example:

```
protected void onActivityResult(int requestCode, int resultCode,
Intent data)
{
   String returnString;
   if (requestCode == REQUEST_CODE && resultCode == RESULT_OK) {
      if (data.hasExtra("returnString1")) {
        returnString = data.getExtras().getString("returnString1");
       }
   }
}
```

Note that the above method checks the returned request code value to make sure that it matches that passed through to the *startActivityForResult()* method. When starting multiple sub-activities it is especially important to use the request code to track which activity is currently returning results, since all will call the same *onActivityResult()* method on exit.

42.4 Implicit Intents

Unlike explicit intents, which reference the class name of the activity to be launched, implicit intents identify the activity to be launched by specifying the action to be performed and the type of data to be handled by the receiving activity. For example, an action type of ACTION_VIEW accompanied by the URL of a web page in the form of a URI object will instruct the Android

system to search for, and subsequently launch, a web browser capable activity. The following implicit intent will, when executed on an Android device, result in the designated web page appearing in a web browser activity:

```
Intent intent = new Intent(Intent.ACTION_VIEW,
        Uri.parse("http://www.ebookfrenzy.com"));

startActivity(intent);
```

When the above implicit intent is issued by an activity, the Android system will search for activities on the device that have registered the ability to handle ACTION_VIEW requests on *http* scheme data using a process referred to as *intent resolution*. In the event that a single match is found, that activity will be launched. If more than one match is found, the user will be prompted to choose from the available activity options.

42.5 Using Intent Filters

Intent filters are the mechanism by which activities "advertise" supported actions and data handling capabilities to the Android intent resolution process. Continuing the example in the previous section, an activity capable of displaying web pages would include an intent filter section in its manifest file indicating support for the ACTION_VIEW type of intent requests on http scheme data.

It is important to note that both the sending and receiving activities must have requested permission for the type of action to be performed. This is achieved by adding *<uses-permission>* tags to the manifest files of both activities. For example, the following manifest lines request permission to access the internet and contacts database:

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

The following *AndroidManifest.xml* file illustrates a configuration for an activity named *WebViewActivity* with intent filters and permissions enabled for internet access*:*

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

```
    <uses-sdk android:minSdkVersion="10" />

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

    <application
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:label="@string/app_name"
            android:name=".WebViewActivity" >
            <intent-filter>
               <action android:name="android.intent.action.VIEW" />
               <category android:name="android.intent.category.DEFAUL
               <data android:scheme="http" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

42.6 Checking Intent Availability

It is generally unwise to assume that an activity will be available for a particular intent, especially since the absence of a matching action will typically result in the application crashing. Fortunately, it is possible to identify the availability of an activity for a specific intent before it is sent to the runtime system. The following method can be used to identify the availability of an activity for a specified intent action type:

```
public static boolean isIntentAvailable(Context context, String
action) {
    final PackageManager packageManager =
context.getPackageManager();
    final Intent intent = new Intent(action);
    List<ResolveInfo> list =
            packageManager.queryIntentActivities(intent,
                    PackageManager.MATCH_DEFAULT_ONLY);
    return list.size() > 0;
}
```

42.7 Summary

Intents are the messaging mechanism by which one Android activity can launch another. An explicit intent references a specific activity to be launched

by referencing the receiving activity by class name. Explicit intents are typically, though not exclusively, used when launching activities contained within the same application. An implicit intent specifies the action to be performed and the type of data to be handled, and lets the Android runtime find a matching activity to launch. Implicit intents are generally used when launching activities that reside in different applications.

An activity can send data to the receiving activity by bundling data into the intent object in the form of key-value pairs. Data can only be returned from an activity if it is started as a *sub-activity* of the sending activity.

Activities advertise capabilities to the Android intent resolution process through the specification of intent-filters in the application manifest file. Both sending and receiving activities must also request appropriate permissions to perform tasks such as accessing the device contact database or the internet.

Having covered the theory of intents, the next few chapters will work through the creation of some examples in Android Studio that put both explicit and implicit intents into action.