44. Android Implicit Intents – A Worked Example

In this chapter, an example application will be created in Android Studio designed to demonstrate a practical implementation of implicit intents. The goal will be to create and send an intent requesting that the content of a particular web page be loaded and displayed to the user. Since the example application itself will not contain an activity capable of performing this task, an implicit intent will be issued so that the Android intent resolution algorithm can be engaged to identify and launch a suitable activity from another application. This is most likely to be an activity from the Chrome web browser bundled with the Android operating system.

Having successfully launched the built-in browser, a new project will be created that also contains an activity capable of displaying web pages. This will be installed onto the device or emulator and used to demonstrate what happens when two activities match the criteria for an implicit intent.

44.1 Creating the Android Studio Implicit Intent Example Project

Launch Android Studio and create a new project, entering *ImplicitIntent* into the Application name field and *ebookfrenzy.com* as the Company Domain setting before clicking on the *Next* button.

On the form factors screen, enable the *Phone and Tablet* option and set the minimum SDK to API 14: Android 4.0 (IceCreamSandwich). Continue to proceed through the screens, requesting the creation of an Empty Activity named *ImplicitIntentActivity* with a corresponding layout resource file named *activity_implicit_intent.*

Click *Finish* to create the new project.

44.2 Designing the User Interface

The user interface for the *ImplicitIntentActivity* class is very simple, consisting solely of a ConstraintLayout and a Button object. Within the Project tool window, locate the *app -> res -> layout -> activity_implicit_intent.xml* file and double-click on it to load it into the

Layout Editor tool.

Delete the default TextView and, with Autoconnect mode enabled, position a Button widget so that it is centered within the layout. Note that the text on the button ("Show Web Page") has been extracted to a string resource named *show_web_page*.

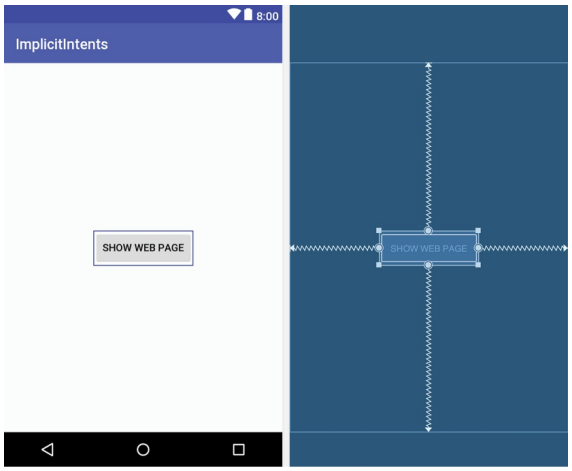

Figure 44-1

With the Button selected use the Attributes tool window to configure the *onClick* property to call a method named *showWebPage()*.

44.3 Creating the Implicit Intent

As outlined above, the implicit intent will be created and issued from within a method named *showWebPage()* which, in turn, needs to be implemented in the *ImplicitIntentActivity* class, the code for which resides in the *ImplicitIntentActivity.java* source file. Locate this file in the Project tool window and double-click on it to load it into an editing pane. Once loaded, modify the code to add the *showWebPage()* method together with a few requisite imports:

```
package com.ebookfrenzy.implicitintent;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.net.Uri;
import android.content.Intent;
import android.view.View;

public class ImplicitIntentActivity extends AppCompatActivity {
```

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

    public void showWebPage(View view) {
        Intent intent = new Intent(Intent.ACTION_VIEW,
                Uri.parse("http://www.ebookfrenzy.com"));

        startActivity(intent);
    }
}
```

The tasks performed by this method are actually very simple. First, a new intent object is created. Instead of specifying the class name of the intent, however, the code simply indicates the nature of the intent (to display something to the user) using the ACTION_VIEW option. The intent object also includes a URI containing the URL to be displayed. This indicates to the Android intent resolution system that the activity is requesting that a web page be displayed. The intent is then issued via a call to the *startActivity()* method.

Compile and run the application on either an emulator or a physical Android device and, once running, touch the *Show Web Page* button. When touched, a web browser view should appear and load the web page designated by the URL. A successful implicit intent has now been executed.

44.4 Adding a Second Matching Activity

The remainder of this chapter will be used to demonstrate the effect of the presence of more than one activity installed on the device matching the requirements for an implicit intent. To achieve this, a second application will be created and installed on the device or emulator. Begin, therefore, by creating a new project within Android Studio with the application name set to *MyWebView*, using the same SDK configuration options used when creating the ImplicitIntent project earlier in this chapter. Select an Empty Activity and, when prompted, name the activity *MyWebViewActivity* and the layout and resource file *activity_my_web_view*.

44.5 Adding the Web View to the UI

The user interface for the sole activity contained within the new *MyWebView* project is going to consist of an instance of the Android WebView widget. Within the Project tool window, locate the *activity_my_web_view.xml* file, which contains the user interface description for the activity, and double-click on it to load it into the Layout Editor tool.

With the Layout Editor tool in Design mode, select the default TextView widget and remove it from the layout by using the keyboard delete key.

Drag and drop a WebView object from the *Containers* section of the palette onto the existing ConstraintLayout view as illustrated in Figure 44-2:

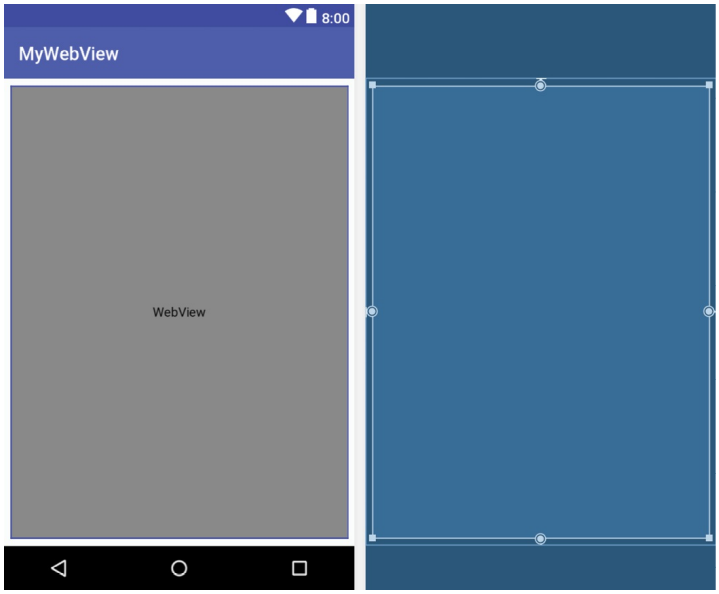

Figure 44-2

Before continuing, change the ID of the WebView instance to *webView1*.

44.6 Obtaining the Intent URL

When the implicit intent object is created to display a web browser window, the URL of the web page to be displayed will be bundled into the intent object within a Uri object. The task of the *onCreate()* method within the *MyWebViewActivity* class is to extract this Uri from the intent object, convert it into a URL string and assign it to the WebView object. To implement this functionality, modify the *MyWebViewActivity.java* file so that it reads as follows:

```
package com.ebookfrenzy.mywebview;

import android.support.v7.app.AppCompatActivity;
```

```
import android.os.Bundle;
import java.net.URL;
import android.net.Uri;
import android.content.Intent;
import android.webkit.WebView;

public class MyWebViewActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_my_web_view);

        handleIntent();
    }

    private void handleIntent() {
        Intent intent = getIntent();

        Uri data = intent.getData();
        URL url = null;

        try {
            url = new URL(data.getScheme(),
                    data.getHost(),
                    data.getPath());
        } catch (Exception e) {
            e.printStackTrace();
        }

        WebView webView = (WebView) findViewById(R.id.webView1);
        webView.loadUrl(url.toString());
    }
}
```

The new code added to the *onCreate()* method performs the following tasks:

- Obtains a reference to the intent which caused this activity to be launched
- Extracts the Uri data from the intent object
- Converts the Uri data to a URL object
- Obtains a reference to the WebView object in the user interface

- Loads the URL into the web view, converting the URL to a String in the process

The coding part of the MyWebView project is now complete. All that remains is to modify the manifest file.

44.7 Modifying the MyWebView Project Manifest File

There are a number of changes that must be made to the MyWebView manifest file before it can be tested. In the first instance, the activity will need to seek permission to access the internet (since it will be required to load a web page). This is achieved by adding the appropriate permission line to the manifest file:

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

Further, a review of the contents of the intent filter section of the *AndroidManifest.xml* file for the MyWebView project will reveal the following settings:

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

In the above XML, the *android.intent.action.MAIN* entry indicates that this activity is the point of entry for the application when it is launched without any data input. The *android.intent.category.LAUNCHER* directive, on the other hand, indicates that the activity should be listed within the application launcher screen of the device.

Since the activity is not required to be launched as the entry point to an application, cannot be run without data input (in this case a URL) and is not required to appear in the launcher, neither the MAIN nor LAUNCHER directives are required in the manifest file for this activity.

The intent filter for the *MyWebViewActivity* activity does, however, need to be modified to indicate that it is capable of handling ACTION_VIEW intent actions for http data schemes.

Android also requires that any activities capable of handling implicit intents that do not include MAIN and LAUNCHER entries, also include the so-called *default category* in the intent filter. The modified intent filter section should therefore read as follows:

```
<intent-filter>
      <action android:name="android.intent.action.VIEW" />
      <category android:name="android.intent.category.DEFAULT" />
      <data android:scheme="http" />
</intent-filter>
```

Bringing these requirements together results in the following complete *AndroidManifest.xml* file:

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

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MyWebViewActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.VIEW" />
                <category
                      android:name="android.intent.category.DEFAULT" />
                <data android:scheme="http" />
            </intent-filter>

        </activity>
    </application>

</manifest>
```

Load the *AndroidManifest.xml* file into the manifest editor by double-clicking on the file name in the Project tool window. Once loaded, modify the XML to match the above changes.

Having made the appropriate modifications to the manifest file, the new activity is ready to be installed on the device.

44.8 Installing the MyWebView Package on a Device

Before the MyWebViewActivity can be used as the recipient of an implicit

intent, it must first be installed onto the device. This is achieved by running the application in the normal manner. Because the manifest file contains neither the *android.intent.action.MAIN* nor the *android.intent.category.LAUNCHER* Android Studio needs to be instructed to install, but not launch, the app. To configure this behavior, select the *app -> Edit configurations…* menu from the toolbar as illustrated in Figure 44-3:

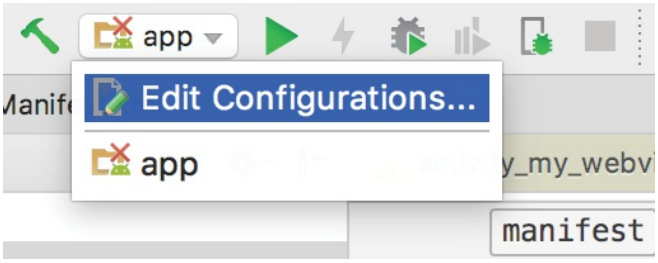

Figure 44-3

Within the Run/Debug Configurations dialog, change the Launch option located in the *Launch Options* section of the panel to *Nothing* and click on Apply followed by OK:

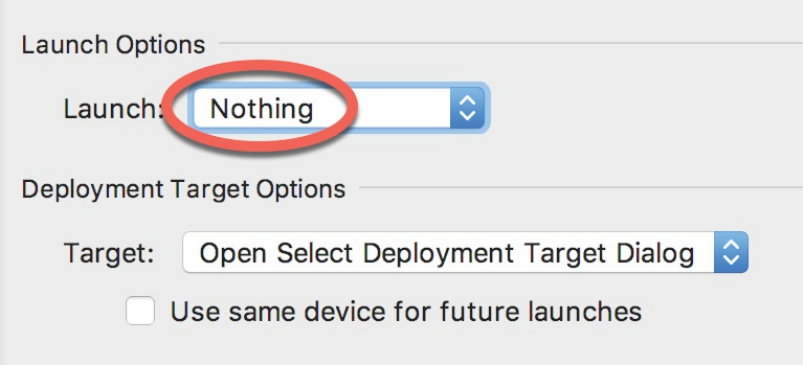

Figure 44-4

With this setting configured run the app as usual. Note that the app is installed on the device, but not launched.

44.9 Testing the Application

In order to test MyWebView, simply re-launch the *ImplicitIntent* application created earlier in this chapter and touch the *Show Web Page* button. This time, however, the intent resolution process will find two activities with intent filters matching the implicit intent. As such, the system will display a dialog (Figure 44-5) providing the user with the choice of activity to launch.

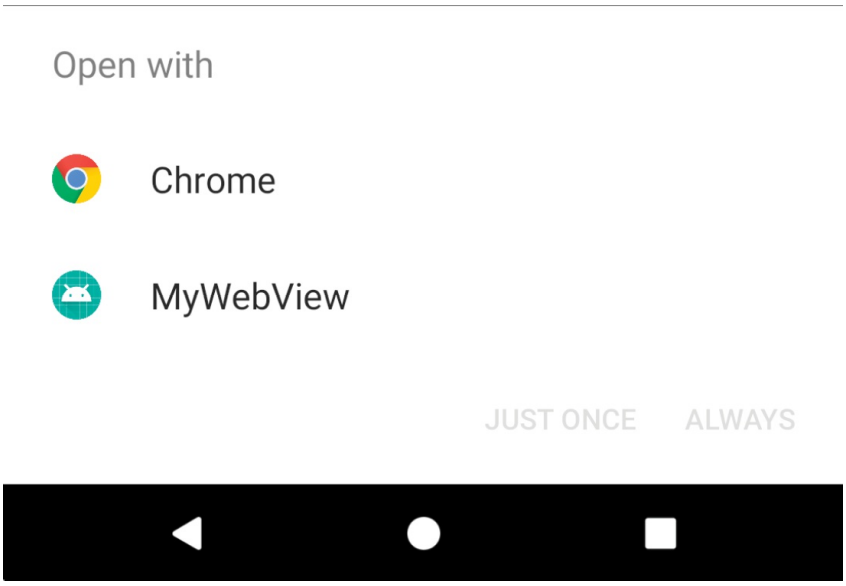

Figure 44-5

Selecting the *MyWebView* option followed by the *Just once* button should cause the intent to be handled by our new *MyWebViewActivity*, which will subsequently appear and display the designated web page.

If the web page loads into the Chrome browser without the above selection dialog appearing, it may be that Chrome has been configured as the default browser on the device. This can be changed by going to *Settings -> Apps & notifications* on the device followed *by App info*. Scroll down the list of apps and select *Chrome*. On the Chrome app info screen, tap the *Open by default* option followed by the *Clear Defaults* button.

44.10 Summary

Implicit intents provide a mechanism by which one activity can request the service of another, simply by specifying an action type and, optionally, the data on which that action is to be performed. In order to be eligible as a target candidate for an implicit intent, however, an activity must be configured to extract the appropriate data from the inbound intent object and be included in a correctly configured manifest file, including appropriate permissions and intent filters. When more than one matching activity for an implicit intent is found during an intent resolution search, the user is prompted to make a choice as to which to use.

Within this chapter an example was created to demonstrate both the issuing of an implicit intent, and the creation of an example activity capable of

handling such an intent.