Android Implicit Intents – A Worked Example

From Techotopia

Previous
Android Explicit Intents – A Worked Example

Table of Contents

Next
Android Broadcast Intents and Broadcast Receivers

In this chapter, an example application will be created that is 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.

Contents

Creating the Implicit Intent Example Project

Within the Eclipse environment, create a new Android Application project. Name the project ImplicitIntent, with the appropriate package name and SDK selections.

Request the creation of a blank activity and the use of the default launcher icons. On the New Blank Activity screen of the New Android Application wizard, set the Activity Name to ImplicitIntentActivity and the Layout and Fragment name to activity_implicit_intent and fragment_implicit_intent.

Click Finish to create the new project.

Designing the User Interface

The user interface for the ImplicitIntentActivity class is very simple, consisting solely of a RelativeLayout view and a button. Within the Package Explorer panel of the Eclipse main window, locate the res -> layout -> fragment_implicit_intent.xml file and double click on it to load it into the editing pane. Select the fragment_implicit_intent.xml tab at the bottom of the editing area and replace the current XML with the following, or visually construct the user interface in the Graphical Layout tool so it resembles Figure 27-1. Note that in both cases, the text on the button has been assigned to a string resource named button_text:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".ImplicitIntentActivity" >

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="@string/button_text"
          android:onClick="showWebPage" />
</RelativeLayout>
```

Figure 27-1

Note the inclusion of the onClick property which will ensure that a method named showWebPage() (which has not yet been implemented) will be called when the button is "clicked" by the user.

Once the changes have been made, be sure to save the XML layout file before proceeding.

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 Package Explorer panel and double click on it to load it into an editing pane. Once loaded, modify the code to add the showWebPage() method together with the requisite imports:

```
package com.example.implicitintent;

import android.app.Activity;
import android.app.ActionBar;
import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;
import android.os.Build;
import android.content.Intent;
import android.net.Uri;

public class ImplicitIntentActivity extends Activity {
.
.
.

        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.

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 named MyWebView within Eclipse, using the usual SDK configuration options. When prompted, name the activity MyWebViewActivity and the layout and fragment activity_my_web_view and fragment_my_web_view.

Once the project has been created, delete the res/layout/fragment_my_web_view.xml file and edit the onCreate() method in the MyWebViewActivity.java file so that it reads as follows:

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

}
```

Remaining within the MyWebViewActivity.java file, also delete the PlaceholderFragment class declaration.

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 view. Within the Package Explorer panel, locate the activity_my_web_view.xml file containing the user interface description for the activity and double click on it to load it into the Graphical Layout tool. Click on the activity_my_web_view.xml tab located on the bottom edge of the layout tool panel to display the XML view of the layout. Edit the XML so that it reads as follows:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MyWebViewActivity" >
```

```
</RelativeLayout>
```

Return to the Graphical Layout view and drag and drop a WebView object from the Composite section of the palette onto the existing RelativeLayout view. By default, the new WebView object should automatically fill the entire display area as illustrated in Figure 30-2:

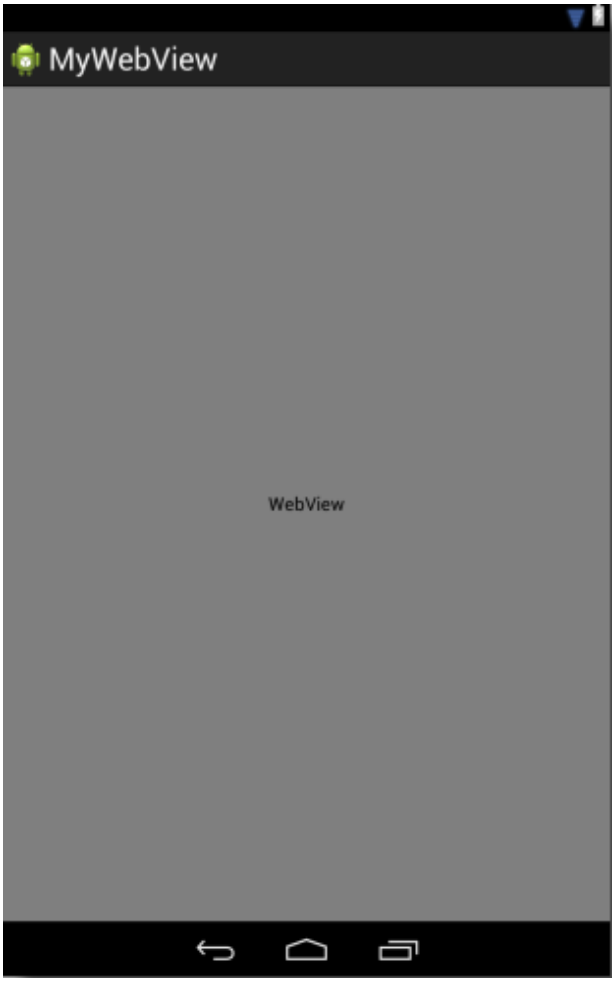

Figure 30-2

Be sure to save the user interface design before proceeding.

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 an 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 which, since we did not provide a specific name, will have a default ID of webView1. To implement this functionality, modify the onCreate() method in MyWebViewActivity.java so that it reads as follows:

```
package com.example.mywebview;

import android.app.Activity;
import android.app.ActionBar;
import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;
import android.os.Build;
import java.net.URL;
import android.net.Uri;
```

```
import android.content.Intent;
import android.webkit.WebView;

public class MyWebViewActivity extends Activity {

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

                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());
        }

        @Override
        public boolean onCreateOptionsMenu(Menu menu) {
                getMenuInflater().inflate(R.menu.activity_my_web_view, menu);
                return true;
        }
}
```

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.

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

    <uses-sdk
        android:minSdkVersion="19"
        android:targetSdkVersion="19" />

    <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=".MyWebViewActivity" >
            <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 Package Explorer panel. Once loaded, select the tab containing the file name located along the bottom edge of the manifest editor panel and 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.

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. In other words, right-click on the MyWebView entry in the Package Explorer panel and select the Run As -> Android Application menu option. Because the manifest file contains neither the android.intent.action.MAIN nor the android.intent.category.LAUNCHER directives, the application will install onto the device, but neither start up nor appear in the launcher. The successful installation of the package will, however, be reported in the Eclipse console panel:

```
[2012-07-02 13:05:27 - MyWebView] Uploading MyWebView.apk onto device '74CE000600000001'
[2012-07-02 13:05:27 - MyWebView] Installing MyWebView.apk...
[2012-07-02 13:05:29 - MyWebView] Success!
[2012-07-02 13:05:29 - MyWebView] \MyWebView\bin\MyWebView.apk installed on device
[2012-07-02 13:05:29 - MyWebView] Done!
```

Once the installation is completed, the activity is ready to be included in the Android framework's intent resolution process.

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 27-3) providing the user with the choice of activity to launch.

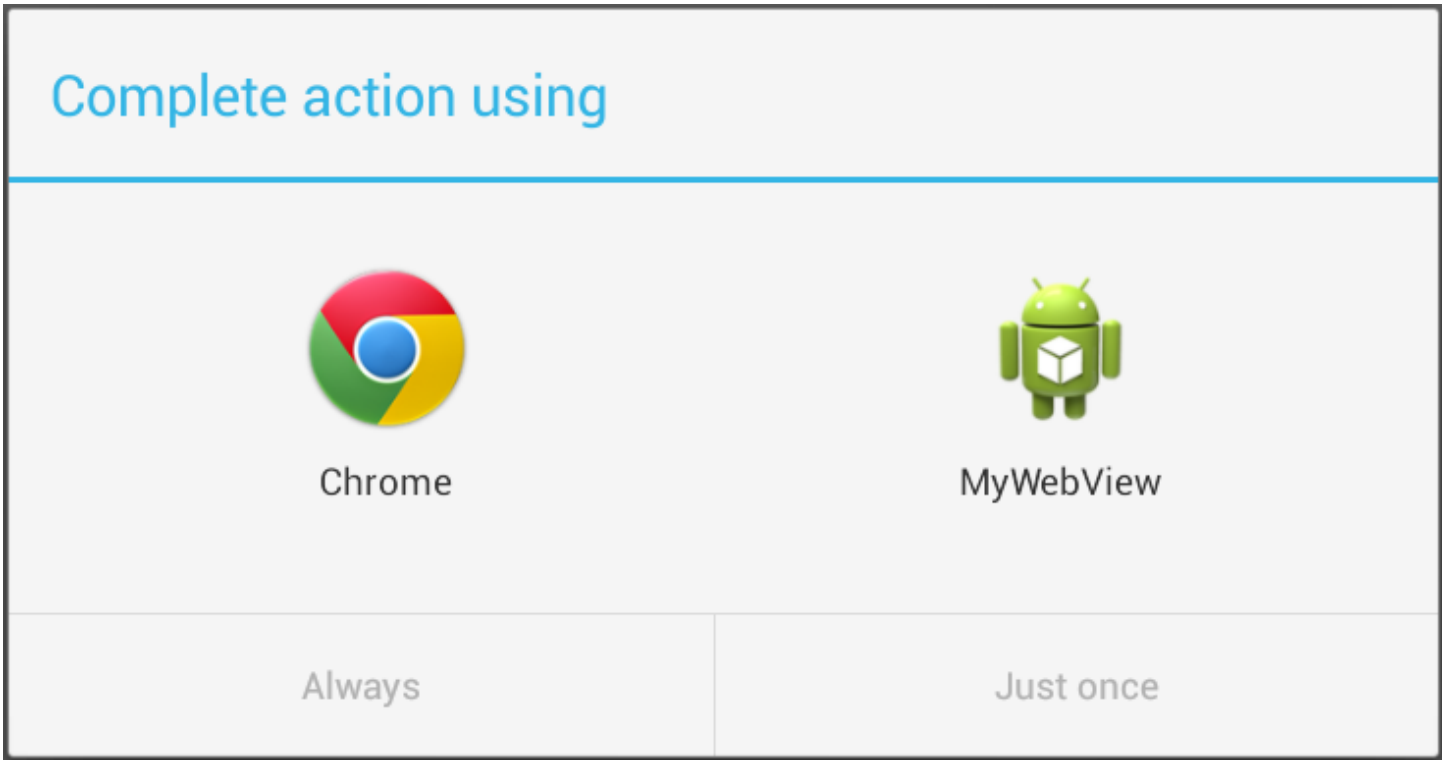

Figure 27-3

Selecting the MyWebView option 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 on the device and choosing the ALL category. Scroll down the list of apps and select Chrome. On the Chrome app info screen, touch the Clear Defaults button.

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 has been created to demonstrate both the issuing of an implicit intent, and the creation of an example activity capable of handling such an intent.

Previous | Table of Contents | Next

Android Explicit Intents – A Worked Example

Android Broadcast Intents and Broadcast Receivers