Android Development

Week 05 Practical - Implicit Intents

# App Overview

In this practical we will create an application with one Activity and three options for actions:

* open a web site
* open a location on a map
* share a snippet of text.

All of the text fields are editable (EditText) but contain default values.

A screenshot of a cell phone

Description automatically generated

# Create the project and layout

Create a new project and app called Implicit Intents, with a new layout.

## Create the project

1. Start Android Studio and create a new Android Studio project. Name your app **Implicit Intents**.
2. Choose **Empty Views Activity** for the project template.
3. Click **Next.**
4. Accept the default Activity name (MainActivity). Make sure the **Generate Layout file** box is checked.
5. Click **Finish**.

## Create the Layout

Create the layout for the app, use Wizard defaults for min SDK version etc. We are going to use a LinearLayout and create a UI with three Button elements, and three EditText elements, like this:

A screenshot of a cell phone

Description automatically generated

1. Open app > res > values > strings.xml in the Project > Android pane, and add the following string resources:

**<string name="edittext\_uri">http://developer.android.com</string>**

**<string name="button\_uri">Open Website</string>**

**<string name="edittext\_loc">Golden Gate Bridge</string>**

**<string name="button\_loc">Open Location</string>**

**<string name="edittext\_share">\'Twas brillig and the slithy toves</string>**

**<string name="button\_share">Share This Text</string>**

1. Open res > layout > activity\_main.xml in the Project > Android pane. Right click the UI editor to bring up a context menu and switch to XML code.
2. Change android.support.constraint.ConstraintLayout to LinearLayout, as you learned in a previous practical (cut and paste the following to replace the opening XML

<**LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"**

**xmlns:app="http://schemas.android.com/apk/res-auto"**

**xmlns:tools="http://schemas.android.com/tools"**

**android:layout\_width="match\_parent"**

**android:layout\_height="match\_parent"**

**android:orientation="vertical"**

**android:padding="16dp"**

**tools:context="com.example.android.implicitintents.MainActivity">**

1. You may need to edit the tools context string (to match any explicit package you may have set in your wizard and you will have to modify the closing tag in your layout
2. Remove the TextView that displays "Hello World".
3. Add a set of UI elements to the layout for the Open Website button. You need an EditText element and a Button element. Use these attribute values:

|  |  |
| --- | --- |
| **EditText attribute** | **Value** |
| android:id | "@+id/website\_edittext" |
| android:layout\_width | "match\_parent" |
| android:layout\_height | "wrap\_content" |
| android:text | "@string/edittext\_uri" |
| **Button attribute** | **Value** |
| android:id | "@+id/open\_website\_button" |
| android:layout\_width | "wrap\_content" |
| android:layout\_height | "wrap\_content" |
| android:layout\_marginBottom | "24dp" |
| android:text | "@string/button\_uri" |
| android:onClick | "openWebsite" |

The value for the android:onClick attribute will remain underlined in red until you define the callback method later.

1. Add a set of UI elements (EditText and Button) to the layout for the **Open Location** button. Use the same attributes as in the previous step but modify them as shown below. (You can copy the values from the **Open Website** button and modify them.)

|  |  |
| --- | --- |
| **EditText attribute** | **Value** |
| android:id | "@+id/location\_edittext" |
| android:text | "@string/edittext\_loc" |
| **Button attribute** | **Value** |
| android:id | "@+id/open\_location\_button" |
| android:text | "@string/button\_loc" |
| android:onClick | "openLocation" |

The value for the android:onClick attribute will remain underlined in red until you define the callback method later.

1. Add a set of UI elements (EditText and Button) to the layout for the **Share This** button. Use the attributes shown below. (You can copy the values from the **Open Website** button and modify them.)

|  |  |
| --- | --- |
| **EditText attribute** | **Value** |
| android:id | "@+id/share\_edittext" |
| android:text | "@string/edittext\_share" |
| **Button attribute** | **Value** |
| android:id | "@+id/share\_text\_button" |
| android:text | "@string/button\_share" |
| android:onClick | "shareText" |

Depending on your version of Android Studio, your activity\_main.xml code should look something like the following. The values for the android:onClick attributes will remain underlined in red until you define the callback methods.

<**LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"**

**xmlns:app="http://schemas.android.com/apk/res-auto"**

**xmlns:tools="http://schemas.android.com/tools"**

**android:layout\_width="match\_parent"**

**android:layout\_height="match\_parent"**

**android:orientation="vertical"**

**android:padding="16dp"**

**tools:context="com.example.android.implicitintents.MainActivity">**

**<EditText**

**android:id="@+id/website\_edittext"**

**android:layout\_width="match\_parent"**

**android:layout\_height="wrap\_content"**

**android:text="@string/edittext\_uri"/>**

**<Button**

**android:id="@+id/open\_website\_button"**

**android:layout\_width="wrap\_content"**

**android:layout\_height="wrap\_content"**

**android:layout\_marginBottom="24dp"**

**android:text="@string/button\_uri"**

**android:onClick="openWebsite"/>**

**<EditText**

**android:id="@+id/location\_edittext"**

**android:layout\_width="match\_parent"**

**android:layout\_height="wrap\_content"**

**android:text="@string/edittext\_uri"/>**

**<Button**

**android:id="@+id/open\_location\_button"**

**android:layout\_width="wrap\_content"**

**android:layout\_height="wrap\_content"**

**android:layout\_marginBottom="24dp"**

**android:text="@string/button\_loc"**

**android:onClick="openLocation"/>**

<**EditText**

**android:id="@+id/share\_edittext"**

**android:layout\_width="match\_parent"**

**android:layout\_height="wrap\_content"**

**android:text="@string/edittext\_share"/>**

**<Button**

**android:id="@+id/share\_text\_button"**

**android:layout\_width="wrap\_content"**

**android:layout\_height="wrap\_content"**

**android:layout\_marginBottom="24dp"**

**android:text="@string/button\_share"**

**android:onClick="shareText"/>**

**</LinearLayout>**

# Manifest changes post SDK 30

On Android 10 and earlier, apps could query the full list of installed apps on the system using methods like queryIntentActivities(). In most cases, this is far broader access than is necessary for an app to implement its functionality. With our ongoing focus on privacy, we’re introducing changes on how apps can query and interact with other installed apps on the same device on Android 11. In particular, we’re bringing better scoped access to the list of apps installed on a given device.

To provide better accountability for access to installed apps on a device, apps targeting Android 11 (API level 30) will see a filtered list of installed apps by default. In order to access a broader list of installed apps, an app can specify information about apps they need to query and interact with directly. This can be done by adding a <queries> element in the Android manifest.

In order to find the browser using an implicit Intent you need to add a query to your manifest file (above the application tag).

**<queries>  
 <intent>  
 <action android:name="android.intent.action.VIEW" />  
 <category android:name="android.intent.category.BROWSABLE" />  
 <data android:scheme="https" />  
 </intent>  
</queries>**

# Implement the ‘Open Website’ button

## Define openWebsite()

Implement the on-click handler method for the first button in the layout, **Open Website**. This action uses an implicit Intent to send the given URI to an Activity that can handle that implicit Intent (such as a web browser).

1. Click "openWebsite" in the activity\_main.xml XML code.
2. Press Alt+Enter (Option+Enter on a Mac) and select **Create 'openWebsite(View)' in 'MainActivity.**

The MainActivity file opens, and Android Studio generates a skeleton method for the openWebsite() handler.

**public void openWebsite(View view) {**

**}**

1. In MainActivity, add a private variable at the top of the class to hold the EditText object for the web site URI.

**private EditText mWebsiteEditText;**

1. In the onCreate() method for MainActivity, use findViewById() to get a reference to the EditText instance and assign it to that private variable:

**mWebsiteEditText = findViewById(R.id.website\_edittext);**

## Add code to openWebsite()

1. Add a statement to the new openWebsite() method that gets the string value of the EditText:

**String url = mWebsiteEditText.getText().toString();**

1. Encode and parse that string into a Uri object:

**Uri webpage = Uri.parse(url);**

1. Create a new Intent with Intent.ACTION\_VIEW as the action and the URI as the data:

**Intent intent = new Intent(Intent.ACTION\_VIEW, webpage);**

This Intent constructor is different from the one you used to create an explicit Intent. In the previous constructor, you specified the current context and a specific component (Activity class) to send the Intent. In this constructor you specify an action and the data for that action. Actions are defined by the Intent class and can include ACTION\_VIEW (to view the given data), ACTION\_EDIT (to edit the given data), or ACTION\_DIAL (to dial a phone number). In this case the action is ACTION\_VIEW because you want to display the web page specified by the URI in the webpage variable.

1. Use the resolveActivity() method and the Android package manager to find an Activity that can handle your implicit Intent. Make sure that the request resolved successfully.

**if (intent.resolveActivity(getPackageManager()) != null) {**

**}**

This request that matches your Intent action and data with the Intent filters for installed apps on the device. You use it to make sure there is at least one Activity that can handle your requests.

1. Inside the if statement, call startActivity() to send the Intent.

**startActivity(intent);**

1. Add an else block to print a Log message if the Intent could not be resolved.

**} else {**

**Log.d("ImplicitIntents", "Can't handle this!");**

**}**

The openWebsite() method should now look as follows. (Comments added for clarity.)

**public void openWebsite(View view) {**

**// Get the URL text.**

**String url = mWebsiteEditText.getText().toString();**

**// Parse the URI and create the intent.**

**Uri webpage = Uri.parse(url);**

**Intent intent = new Intent(Intent.ACTION\_VIEW, webpage);**

**// Find an activity to hand the intent and start that activity.**

**if (intent.resolveActivity(getPackageManager()) != null) {**

**startActivity(intent);**

**} else {**

**Log.d("ImplicitIntents", "Can't handle this intent!");**

**}**

**}**

# Implement the ‘Open Location’ button

Implement the on-click handler method for the second button in the UI, **Open Location**. This method is almost identical to the openWebsite() method. The difference is the use of a geo URI to indicate a map location. You can use a geo URI with latitude and longitude or use a query string for a general location. In this example we've used the latter.

## Define openLocation ()

1. Click "openLocation" in the activity\_main.xml XML code.
2. Press Alt+Enter (Option+Enter on a Mac) and select **Create 'openLocation(View)' in MainActivity.**

Android Studio generates a skeleton method in MainActivity for the openLocation() handler.

**public void openLocation(View view) {**

**}**

1. Add a private variable at the top of MainActivity to hold the EditText object for the location URI.

**private EditText mLocationEditText;**

1. In the onCreate() method, use findViewByID() to get a reference to the EditText instance and assign it to that private variable:

**mLocationEditText = findViewById(R.id.location\_edittext);**

## Add code to openLocation()

1. In the new openLocation() method, add a statement to get the string value of the mLocationEditText EditText.

**String loc = mLocationEditText.getText().toString();**

1. Parse that string into a Uri object with a geo search query:

**Uri addressUri = Uri.parse("geo:0,0?q=" + loc);**

1. Create a new Intent with Intent.ACTION\_VIEW as the action and loc as the data.

**Intent intent = new Intent(Intent.ACTION\_VIEW, addressUri);**

1. Resolve the Intent and check to make sure that the Intent resolved successfully. If so, startActivity(), otherwise log an error message.

**if (intent.resolveActivity(getPackageManager()) != null) {**

**startActivity(intent);**

**} else {**

**Log.d("ImplicitIntents", "Can't handle this intent!");**

**}**

The openLocation() method should now look as follows (comments added for clarity):

**public void openLocation(View view) {**

**// Get the string indicating a location. Input is not validated; it is**

**// passed to the location handler intact.**

**String loc = mLocationEditText.getText().toString();**

**// Parse the location and create the intent.**

**Uri addressUri = Uri.parse("geo:0,0?q=" + loc);**

**Intent intent = new Intent(Intent.ACTION\_VIEW, addressUri);**

**// Find an activity to handle the intent, and start that activity.**

**if (intent.resolveActivity(getPackageManager()) != null) {**

**startActivity(intent);**

**} else {**

**Log.d("ImplicitIntents", "Can't handle this intent!");**

**}**

**}**

# Implement the ‘Share This’ button

A share action is an easy way for users to share items in your app with social networks and other apps. Although you could build a share action in your own app using an implicit Intent, Android provides the [ShareCompat.IntentBuilder](http://developer.android.com/reference/android/support/v4/app/ShareCompat.IntentBuilder.html) helper class to make implementing sharing easy. You can use ShareCompat.IntentBuilder to build an Intent and launch a chooser to let the user choose the destination app for sharing.

1. Click "shareText" in the activity\_main.xml XML code.
2. Press Alt+Enter (Option+Enter on a Mac) and select Create 'shareText(View)' in MainActivity.

Android Studio generates a skeleton method in MainActivity for the shareText() handler.

**public void shareText(View view) {**

**}**

1. Add a private variable at the top of MainActivity to hold the EditText.

**private EditText mShareTextEditText;**

1. In onCreate(), use findViewById() to get a reference to the EditText instance and assign it to that private variable:

**mShareTextEditText = findViewById(R.id.share\_edittext);**

## Add code to shareText()

1. In the new shareText() method, add a statement to get the string value of the mShareTextEditText EditText.

**String txt = mShareTextEditText.getText(). toString();**

1. Define the mime type of the text to share:

**String mimeType = "text/plain";**

1. Call ShareCompat.IntentBuilder with these ‘builder pattern’ methods:

**ShareCompat.IntentBuilder**

**.from(this)**

**.setType(mimeType)**

**.setChooserTitle("Share this text with: ")**

**.setText(txt)**

**.startChooser();**

1. Extract the value of .setChoosterTitle to a string resource.

The call to ShareCompat.IntentBuilder uses these methods:

|  |  |
| --- | --- |
| **Method** | **Description** |
| from() | The Activity that launches this share Intent (this). |
| setType() | The MIME type of the item to be shared. |
| setChooserTitle() | The title that appears on the system app chooser. |
| setText() | The actual text to be shared |
| startChooser() | Show the system app chooser and send the Intent. |

This format, with all the builder's setter methods strung together in one statement, is an easy shorthand way to create and launch the Intent. You can add any of the additional methods to this list.

The shareText() method should now look as follows:

**public void shareText(View view) {**

**String txt = mShareTextEditText.getText().toString();**

**String mimeType = "text/plain";**

**ShareCompat.IntentBuilder**

**.from(this)**

**.setType(mimeType)**

**.setChooserTitle(R.string.share\_text\_with)**

**.setText(txt)**

**.startChooser();**

**}**

# Run the App

1. Run the app.
2. Click the Open Website button to launch a browser with the website URL in the EditText above the Button. The browser and website should appear as shown below.

A screenshot of a cell phone

Description automatically generated

1. Click the Open Location button to launch the map with the location in the EditText above the Button. The map with the location should appear as shown below.

A screenshot of a cell phone

Description automatically generated

1. Click the Share This Text button to launch a dialog with choices for sharing the text. The dialog with choices should appear as shown below. If you’re running on a phone (rather than an emulator) play with it and send some stuff.

A screenshot of a cell phone

Description automatically generated