30. Using Fragments in Android Studio - An Example

As outlined in the previous chapter, fragments provide a convenient mechanism for creating reusable modules of application functionality consisting of both sections of a user interface and the corresponding behavior. Once created, fragments can be embedded within activities.

Having explored the overall theory of fragments in the previous chapter, the objective of this chapter is to create an example Android application using Android Studio designed to demonstrate the actual steps involved in both creating and using fragments, and also implementing communication between one fragment and another within an activity.

30.1 About the Example Fragment Application

The application created in this chapter will consist of a single activity and two fragments. The user interface for the first fragment will contain a toolbar of sorts consisting of an EditText view, a SeekBar and a Button, all contained within a RelativeLayout view. The second fragment will consist solely of a TextView object, also contained within a RelativeLayout view.

The two fragments will be embedded within the main activity of the application and communication implemented such that when the button in the first fragment is pressed, the text entered into the EditText view will appear on the TextView of the second fragment using a font size dictated by the position of the SeekBar in the first fragment.

Since this application is intended to work on earlier versions of Android, it will also be necessary to make use of the appropriate Android support library.

30.2 Creating the Example Project

Create a new project in Android Studio with , entering *FragmentExample* 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 setting to API 14: Android 4.0 (IceCreamSandwich). Continue to proceed through the screens, requesting the creation of an Empty

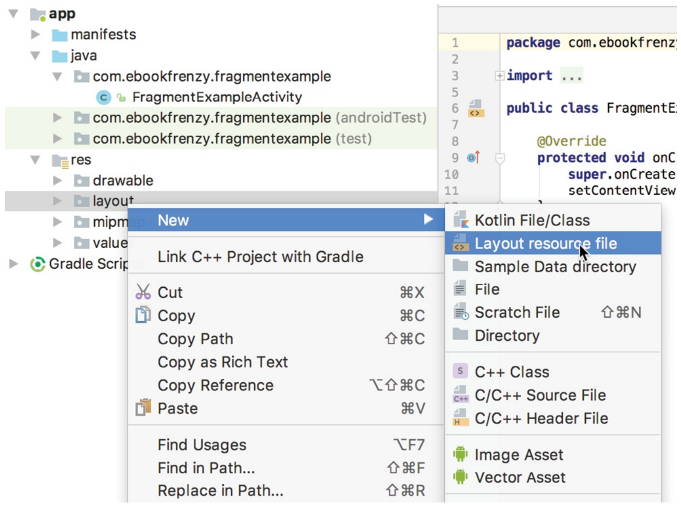
Activity named *FragmentExampleActivity* with a corresponding layout resource file named *activity_fragment_example.*

Click the *Finish* button to begin the project creation process.

30.3 Creating the First Fragment Layout

The next step is to create the user interface for the first fragment that will be used within our activity.

This user interface will, of course, reside in an XML layout file so begin by navigating to the *layout* folder located under *app* -> *res* in the Project tool window. Once located, right-click on the *layout* entry and select the *New* -> *Layout resource file* menu option as illustrated in Figure 30-1:



```
package com.ebookfrenz

import ...

public class FragmentE

    @Override
    protected void onC
        super.onCreate
        setContentView
```

Figure 30-1

In the resulting dialog, name the layout *toolbar_fragment* and change the root element to RelativeLayout before clicking on OK to create the new resource file.

The new resource file will appear within the Layout Editor tool ready to be designed. Switch the Layout Editor to Text mode and modify the XML so that it reads as outlined in the following listing to add three new view elements to the layout:

```
<?xml version="1.0" encoding="utf-8"?>

<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/seekBar1"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="17dp"
        android:text="Change Text" />

    <EditText
        android:id="@+id/editText1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="16dp"
        android:ems="10"
        android:inputType="text" >
        <requestFocus />
    </EditText>

    <SeekBar
        android:id="@+id/seekBar1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentStart="true"
        android:layout_below="@+id/editText1"
        android:layout_marginTop="14dp"
        android:layout_alignParentLeft="true" />

</RelativeLayout>
```

Once the changes have been made, switch the Layout Editor tool back to Design mode and click on the warning button in the top right-hand corner of the design area. Select the hardcoded text warning, click the *Fix* button and assign the string to a resource named *change_text*.

Upon completion of these steps, the user interface layout should resemble that of Figure 30-2:

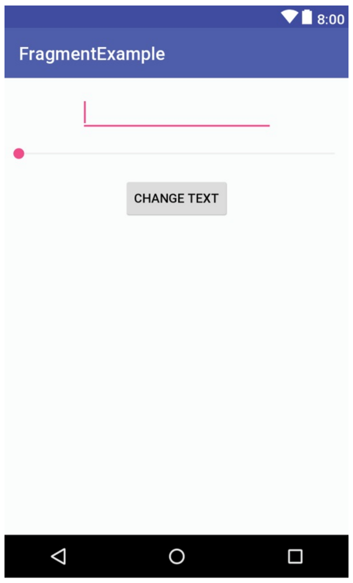


Figure 30-2

With the layout for the first fragment implemented, the next step is to create a class to go with it.

30.4 Creating the First Fragment Class

In addition to a user interface layout, a fragment also needs to have a class associated with it to do the actual work behind the scenes. Add a class for this purpose to the project by unfolding the *app* -> *java* folder in the Project tool window and right-clicking on the package name given to the project when it was created (in this instance *com.ebookfrenzy.fragmentexample*). From the resulting menu, select the *New* -> *Java Class* option. In the resulting *Create New Class* dialog, name the class *ToolbarFragment* and click on *OK* to create the new class.

Once the class has been created it should, by default, appear in the editing panel where it will read as follows:

```
package com.ebookfrenzy.fragmentexample;

/**
 * Created by <name> on <date>.
 */
public class ToolbarFragment {
}
```

For the time being, the only changes to this class are the addition of some import directives and the overriding of the *onCreateView()* method to make sure the layout file is inflated and displayed when the fragment is used within an activity. The class declaration also needs to indicate that the class extends the Android Fragment class:

```
package com.ebookfrenzy.fragmentexample;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class ToolbarFragment extends Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater,
                             ViewGroup container, Bundle
                            savedInstanceState) {

        // Inflate the layout for this fragment
        View view =  inflater.inflate(R.layout.toolbar_fragment,
                container, false);
        return view;
    }
}
```

Later in this chapter, more functionality will be added to this class. Before that, however, we need to create the second fragment.

30.5 Creating the Second Fragment Layout

Add a second new Android XML layout resource file to the project, once again selecting a RelativeLayout as the root element. Name the layout *text_fragment* and click *OK*. When the layout loads into the Layout Editor tool, change to Text mode and modify the XML to add a TextView to the fragment layout as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/and
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="Fragment Two"
        android:textAppearance="?android:attr/textAppearanceLarge" />
</RelativeLayout>
```

Once the XML changes have been made, switch back to Design mode and extract the string to a resource named *fragment_two*. Upon completion of these steps, the user interface layout for this second fragment should resemble that of Figure 30-3.

As with the first fragment, this one will also need to have a class associated with it. Right-click on *app -> java -> com.ebookfrenzy.fragmentexample* in the Project tool window. From the resulting menu, select the *New -> Java Class* option. Name the fragment *TextFragment* and click *OK* to create the class.

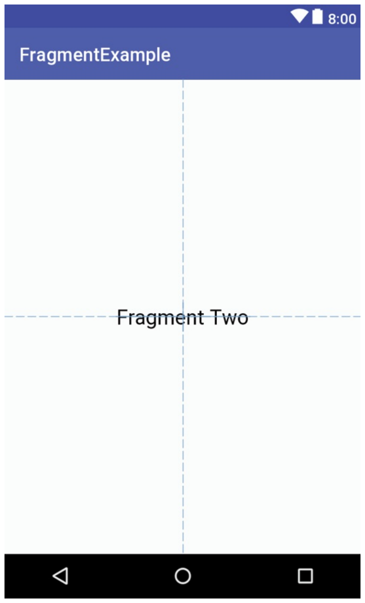


Figure 30-3

Edit the new *TextFragment.java* class file and modify it to implement the *onCreateView()* method and designate the class as extending the Android *Fragment* class:

```
package com.ebookfrenzy.fragmentexample;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class TextFragment extends Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater,
                             ViewGroup container,
                             Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.text_fragment,
                container, false);

        return view;
    }
}
```

Now that the basic structure of the two fragments has been implemented, they are ready to be embedded in the application's main activity.

30.6 Adding the Fragments to the Activity

The main activity for the application has associated with it an XML layout file named *activity_fragment_example.xml*. For the purposes of this example, the fragments will be added to the activity using the <fragment> element within this file. Using the Project tool window, navigate to the *app* -> *res* -> *layout* section of the *FragmentExample* project and double-click on the *activity_fragment_example.xml* file to load it into the Android Studio Layout Editor tool.

With the Layout Editor tool in Design mode, select and delete the default TextView object from the layout and select the *Layouts* category in the palette. Drag the <*fragment*> component from the list of layouts and drop it onto the layout so that it is centered horizontally and positioned such that the dashed line appears indicating the top layout margin:

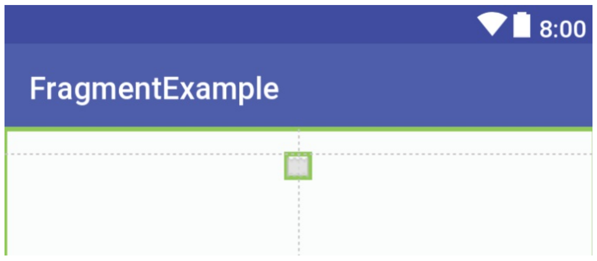
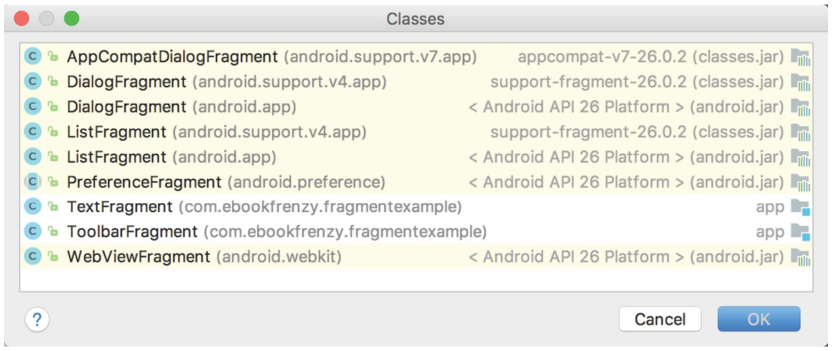


Figure 30-4

On dropping the fragment onto the layout, a dialog will appear displaying a list of Fragments available within the current project as illustrated in Figure 30-5:

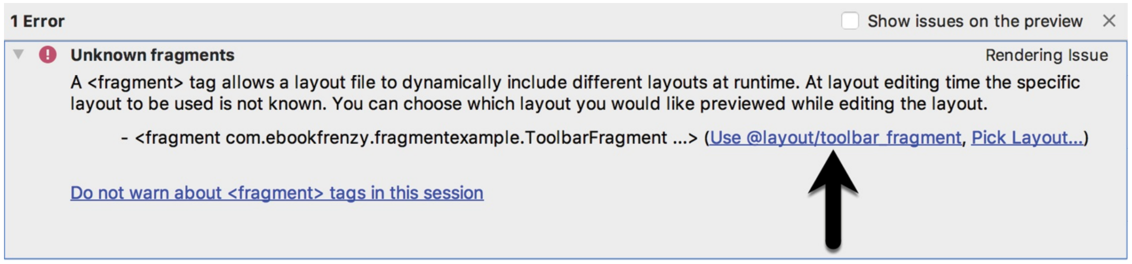


Classes

AppCompatDialogFragment (android.support.v7.app)	appcompat-v7-26.0.2 (classes.jar)
DialogFragment (android.support.v4.app)	support-fragment-26.0.2 (classes.jar)
DialogFragment (android.app)	< Android API 26 Platform > (android.jar)
ListFragment (android.support.v4.app)	support-fragment-26.0.2 (classes.jar)
ListFragment (android.app)	< Android API 26 Platform > (android.jar)
PreferenceFragment (android.preference)	< Android API 26 Platform > (android.jar)
TextFragment (com.ebookfrenzy.fragmentexample)	app
ToolbarFragment (com.ebookfrenzy.fragmentexample)	app
WebViewFragment (android.webkit)	< Android API 26 Platform > (android.jar)

Figure 30-5

Select the ToolbarFragment entry from the list and click on the OK button to dismiss the Fragments dialog. Once added, click on the red warning button in the top right-hand corner of the layout editor to display the warnings panel. An *unknown fragments* message (Figure 30-6) will be listed indicating that the Layout Editor tool needs to know which fragment to display during the preview session. Display the ToolbarFragment fragment by clicking on the *Use @layout/toolbar_fragment* link within the message:



Unknown fragments Rendering Issue

A <fragment> tag allows a layout file to dynamically include different layouts at runtime. At layout editing time the specific layout to be used is not known. You can choose which layout you would like previewed while editing the layout.

- <fragment com.ebookfrenzy.fragmentexample.ToolbarFragment ...> (Use @layout/toolbar_fragment, Pick Layout...)

Do not warn about <fragment> tags in this session

Figure 30-6

Click and drag another <fragment> entry from the panel and position it so that it is centered horizontally and positioned beneath the bottom edge of the first fragment. When prompted, select the *TextFragment* entry from the fragment dialog before clicking on the OK button. When the rendering message appears, click on the *Use @layout/text_fragment* option. Establish a constraint connection between the top edge of the TextFragment and the bottom edge of the ToolbarFragment.

Note that the fragments are now visible in the layout as demonstrated in Figure 30-7:

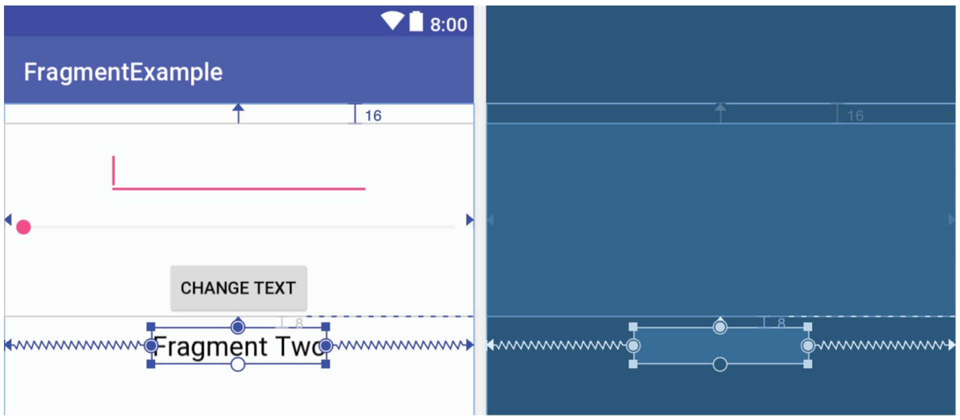


Figure 30-7

Before proceeding to the next step, select the TextFragment instance in the layout and, within the Attributes tool window, change the ID of the fragment to *text_fragment*.

30.7 Making the Toolbar Fragment Talk to the Activity

When the user touches the button in the toolbar fragment, the fragment class is going to need to get the text from the EditText view and the current value of the SeekBar and send them to the text fragment. As outlined in "An Introduction to Android Fragments", fragments should not communicate with each other directly, instead using the activity in which they are embedded as an intermediary.

The first step in this process is to make sure that the toolbar fragment responds to the button being clicked. We also need to implement some code to keep track of the value of the SeekBar view. For the purposes of this example, we will implement these listeners within the ToolbarFragment class. Select the *ToolbarFragment.java* file and modify it so that it reads as shown in the following listing:

```
package com.ebookfrenzy.fragmentexample;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.content.Context;
import android.widget.Button;
import android.widget.EditText;
import android.widget.SeekBar;
import android.widget.SeekBar.OnSeekBarChangeListener;

public class ToolbarFragment extends Fragment implements OnSeekBarCha

    private static int seekvalue = 10;
    private static EditText edittext;

    @Override
    public View onCreateView(LayoutInflater inflater,
                             ViewGroup container, Bundle
                            savedInstanceState) {

        // Inflate the layout for this fragment
        View view =  inflater.inflate(R.layout.toolbar_fragment,
                container, false);

        edittext = (EditText) view.findViewById(R.id.editText1);
        final SeekBar seekbar =
                (SeekBar) view.findViewById(R.id.seekBar1);

        seekbar.setOnSeekBarChangeListener(this);

        final Button button =
                (Button) view.findViewById(R.id.button1);

        button.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                buttonClicked(v);
            }
        });

        return view;
    }

    public void buttonClicked (View view) {

    }

    @Override
    public void onProgressChanged(SeekBar seekBar, int progress,
                                  boolean fromUser) {
        seekvalue = progress;
    }

    @Override
    public void onStartTrackingTouch(SeekBar arg0) {

    }

    @Override
    public void onStopTrackingTouch(SeekBar arg0) {

    }
}
```

Before moving on, we need to take some time to explain the above code changes. First, the class is declared as implementing the OnSeekBarChangeListener interface. This is because the user interface contains a SeekBar instance and the fragment needs to receive notifications when the user slides the bar to change the font size. Implementation of the OnSeekBarChangeListener interface requires that the *onProgressChanged()*, *onStartTrackingTouch()* and *onStopTrackingTouch()* methods be implemented. These methods have been implemented but only the *onProgressChanged()* method is actually required to perform a task, in this case storing the new value in a variable named seekvalue which has been declared at the start of the class. Also declared is a variable in which to store a reference to the EditText object.

The *onCreateView()* method has been modified to obtain references to the EditText, SeekBar and Button views in the layout. Once a reference to the button has been obtained it is used to set up an onClickListener on the button which is configured to call a method named *buttonClicked()* when a click event is detected. This method is also then implemented, though at this point it does not do anything.

The next phase of this process is to set up the listener that will allow the fragment to call the activity when the button is clicked. This follows the mechanism outlined in the previous chapter:

```
public class ToolbarFragment extends Fragment
    implements OnSeekBarChangeListener {

        private static int seekvalue = 10;
        private static EditText edittext;
```

```
ToolbarListener activityCallback;

public interface ToolbarListener {
      public void onButtonClick(int position, String text);
}

@Override
public void onAttach(Context context) {
      super.onAttach(context);
      try {
          activityCallback = (ToolbarListener) context;
      } catch (ClassCastException e) {
          throw new ClassCastException(context.toString()
              + " must implement ToolbarListener");
      }
 }

 @Override
  public View onCreateView(LayoutInflater inflater,
     ViewGroup container, Bundle savedInstanceState) {
      // Inflate the layout for this fragment

      View view =
            inflater.inflate(R.layout.toolbar_fragment,
                    container, false);

      edittext = (EditText)
            view.findViewById(R.id.editText1);
      final SeekBar seekbar =
             (SeekBar) view.findViewById(R.id.seekBar1);

      seekbar.setOnSeekBarChangeListener(this);

      final Button button =
        (Button) view.findViewById(R.id.button1);
      button.setOnClickListener(new View.OnClickListener() {
          public void onClick(View v) {
              buttonClicked(v);
          }
      });

      return view;
```

```
        }

        public void buttonClicked (View view) {
                activityCallback.onButtonClick(seekvalue,
                     edittext.getText().toString());
        }
.
.
.
}
```

The above implementation will result in a method named *onButtonClick()* belonging to the activity class being called when the button is clicked by the user. All that remains, therefore, is to declare that the activity class implements the newly created ToolbarListener interface and to implement the *onButtonClick()* method.

Since the Android Support Library is being used for fragment support in earlier Android versions, the activity also needs to be changed to subclass from *FragmentActivity* instead of *AppCompatActivity*. Bringing these requirements together results in the following modified *FragmentExampleActivity.java* file:

```
package com.ebookfrenzy.fragmentexample;

~~import android.support.v7.app.AppCompatActivity;~~
import android.support.v4.app.FragmentActivity;
import android.os.Bundle;

public class FragmentExampleActivity extends FragmentActivity implemen

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

    public void onButtonClick(int fontsize, String text) {

    }
}
```

With the code changes as they currently stand, the toolbar fragment will detect when the button is clicked by the user and call a method on the activity passing through the content of the EditText field and the current setting of the SeekBar view. It is now the job of the activity to communicate with the Text Fragment and to pass along these values so that the fragment can update the TextView object accordingly.

30.8 Making the Activity Talk to the Text Fragment

As outlined in *“An Introduction to Android Fragments”*, an activity can communicate with a fragment by obtaining a reference to the fragment class instance and then calling public methods on the object. As such, within the TextFragment class we will now implement a public method named *changeTextProperties()* which takes as arguments an integer for the font size and a string for the new text to be displayed. The method will then use these values to modify the TextView object. Within the Android Studio editing panel, locate and modify the *TextFragment.java* file to add this new method and to add code to the *onCreateView()* method to obtain the ID of the TextView object:

```
package com.ebookfrenzy.fragmentexample;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

public class TextFragment extends Fragment {

    private static TextView textview;

    @Override
    public View onCreateView(LayoutInflater inflater,
                             ViewGroup container,
                             Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.text_fragment,
                container, false);

        textview = (TextView) view.findViewById(R.id.textView1);

        return view;
    }
    public void changeTextProperties(int fontsize, String text)
    {
        textview.setTextSize(fontsize);
        textview.setText(text);
    }
}
```

When the TextFragment fragment was placed in the layout of the activity, it was given an ID of *text_fragment*. Using this ID, it is now possible for the activity to obtain a reference to the fragment instance and call the *changeTextProperties()* method on the object. Edit the *FragmentExampleActivity.java* file and modify the *onButtonClick()* method as follows:

```
public void onButtonClick(int fontsize, String text) {

   TextFragment textFragment =
    (TextFragment)
      getSupportFragmentManager().findFragmentById(R.id.text_fragment

   textFragment.changeTextProperties(fontsize, text);
}
```

30.9 Testing the Application

With the coding for this project now complete, the last remaining task is to run the application. When the application is launched, the main activity will start and will, in turn, create and display the two fragments. When the user touches the button in the toolbar fragment, the *onButtonClick()* method of the activity will be called by the toolbar fragment and passed the text from the EditText view and the current value of the SeekBar. The activity will then call the *changeTextProperties()* method of the second fragment, which will modify the TextView to reflect the new text and font size:

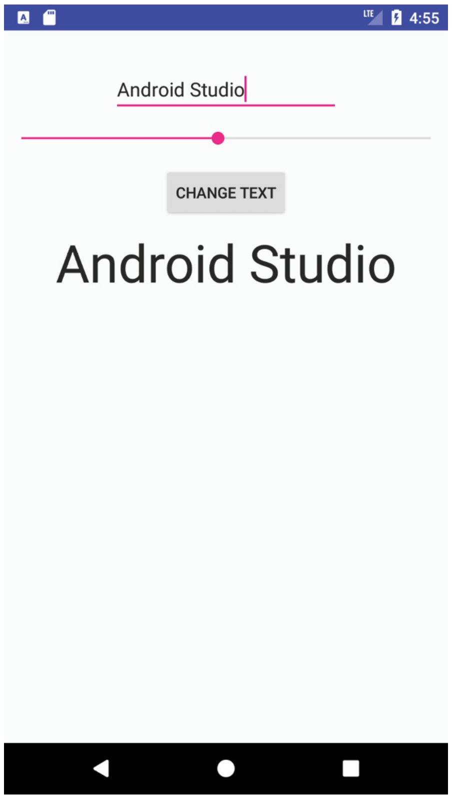


Figure 30-8

30.10 Summary

The goal of this chapter was to work through the creation of an example project intended specifically to demonstrate the steps involved in using fragments within an Android application. Topics covered included the use of the Android Support Library for compatibility with Android versions predating the introduction of fragments, the inclusion of fragments within an activity layout and the implementation of inter-fragment communication.