About This Article

This article provides information about how to use Fragments to build User Interfaces in Android applications. Android application developers can use this article as reference to implement the UI in their applications from Android 3.0(Honeycomb).

Scope:

This article is intended for Android developers wishing to develop mobile applications. It assumes basic knowledge of Android and Java programming languages.

To find out more about Android, please refer to the Knowledge Base under Samsung Developers.
http://developer.samsung.com/android/all-resources

Introduction

The Fragment API introduced in Android 3.0, allows developers to more easily design dynamic user interfaces. It helps to organize user interface components in a reusable manner across Activities.

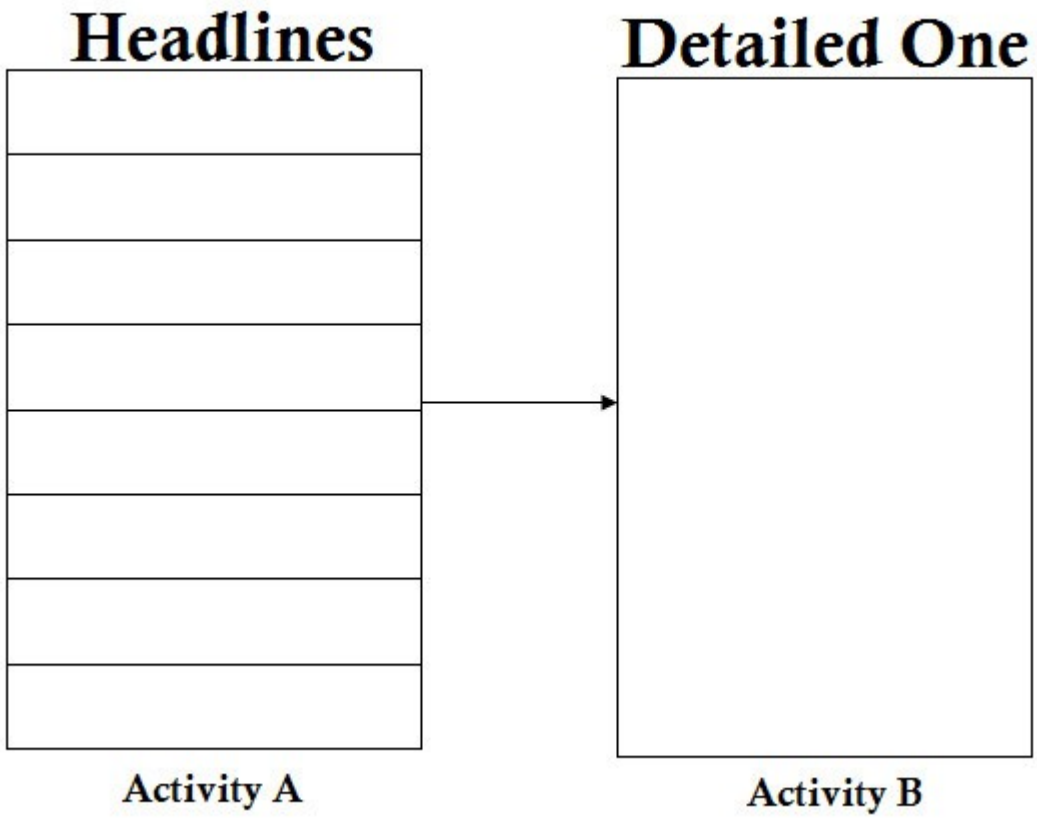
A Fragment represents a portion of user interface in an Activity. Multiple fragments can be used in a single activity to build a multi-pane UI, and reuse it as a single fragment in multiple activities. In simple terms, a Fragment is a chunk of user interface with its own life cycle. This article discusses the use of fragments and its implementation in projects.

Redesigning the Screens

Normal Approach

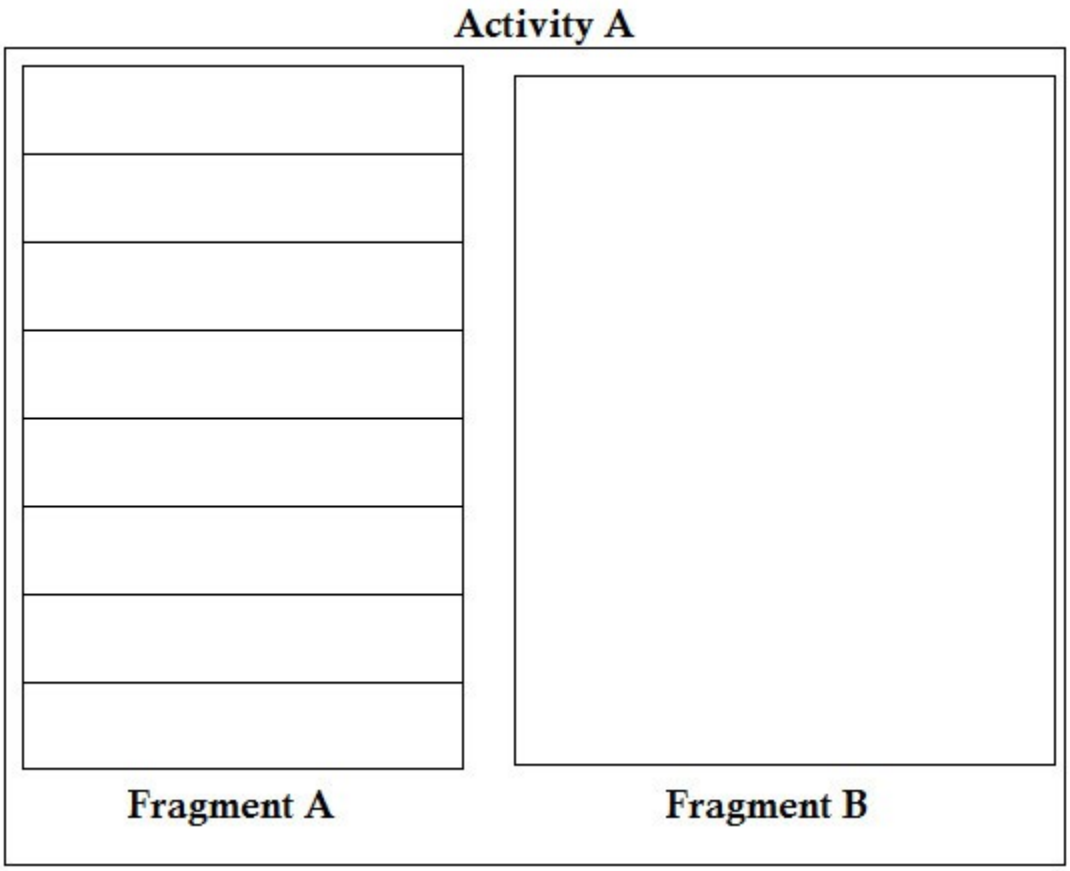
Consider a news application which lists headlines and articles in brief. If user selects one of the headlines, the application shows the article in detail. The design of the screen before Android 3.0 is as follows:

http://developer.samsung.com/android/technical-docs/Using-Fragments-to-Build-User-Interfaces-in-Android



Better Approach

Starting with Android 3.0, the Fragments API is available to design the screens. On larger screens, Listview can help to design a more effective user interface to display the detailed article beside the list on the same screen. When the user clicks a specific ListView element, the detailed article view on the right side updates to display the appropriate content. The following figure illustrates such a redesign:



Creating Fragments

A Fragment has its own life cycle similar to any other application components in Android. Create a fragment by creating a subclass of the Fragment class. The Fragment class contains few callback methods like an Activity. A few of them are onCreate (), onStart(), onPause() and onStop(). Almost all applications should implement at least three methods for every fragment: onCreate(), onCreateView(),onPause(). There are several other callback methods to handle various stages of the fragment lifecycle. For more details regarding Fragment lifecycle see: http://developer.android.com/reference/android/app/Fragment.html#LifecycleFollowing is example code for creating an application with two fragments that shows one listview on the left side and one detailed view on right side as shown in above figure.

Walkthrough

Create two Java classes to represent the two fragments: the ListView and the DetailedView screens. Name themFragmentA and FragmentB. FragmentA extends the ListFragment class and FragmentB extends the Fragment class. The following code snippet shows the skeleton of both classes.

FragmentA class

```
public class FragmentA extends ListFragment
{
        // need to implement the class here
}
```

FragmentB class

```
public class FragmentB extends Fragment
```

```
        {
                // need to implement the class here
        }
```

Let us explore the FragmentA class first that uses list widget to show some elements.

```
private final String[] tutorialList = {"RED","GREEN","BLUE",};
```

Now implement onCreate() method and onListItemClick() methods in FragmentA class

```
        @Override
        public void onCreate(Bundle savedInstanceState) {
                // TODO Auto-generated method stub
                super.onCreate(savedInstanceState);
                setListAdapter(new ArrayAdapter(getActivity().
        getApplicationContext(), android.R.layout.simple_list_item_activated_1,
tutorialList));
        }

        @Override
        public void onListItemClick(ListView l, View v, int position, long id) {
                // TODO Auto-generated method stub
                super.onListItemClick(l, v, position, id);
                   Intent launchingIntent = new
Intent(getActivity(),DetailActivity.class);
                   launchingIntent.putExtra("COLOR_TYPE", tutorialList[position]);
                   startActivity(launchingIntent);
        }
```

Next, look at the FragmentB class to receive the color sent from the FragmentA. Implement an onCreateView()method in the FragmentB class.

viewer.xml

In the FragmentB class, use an xml file to show the color for the selection from FragmentA. Following is theviewer.xml file:

```
        <?xml version="1.0" encoding="utf-8"?>
        <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
                android:layout_width="fill_parent"
                android:layout_height="fill_parent"
                android:gravity="center"
                android:orientation="horizontal" >

                <TextView
                        android:id="@+id/textSample"
                        android:layout_width="fill_parent"
                        android:layout_height="fill_parent"
                        android:background="#CC9988"
                        android:gravity="center"
                        android:text="This is the sample text"
```

```
                        android:textColor="#000000" />


        </LinearLayout>
```

FragmentB class

```
        public class FragmentB extends Fragment{
        TextView mtext;
        @Override
        public View onCreateView(LayoutInflater inflater, ViewGroup container,
                                 Bundle savedInstanceState) {
                 // TODO Auto-generated method stub
                 Intent launchingIntent = getActivity().getIntent();
                 String color = launchingIntent.getStringExtra("COLOR_TYPE");
                    View viewer = (View) inflater.inflate(R.layout.viewer, container,
false);
                 mtext  = (TextView) viewer.findViewById(R.id.textSample);
                 updateColor(color);
                 return viewer;
                 }

        /**
        * for updating the color from the given input
        * @param color
        */
        public void updateColor(String color)
        {
        if(color != null)
                    if(color.equals("RED"))
                                 mtext.setBackgroundColor(Color.RED);
                    else if(color.equals("BLUE"))
                             mtext.setBackgroundColor(Color.BLUE);
                    else if(color.equals("GREEN"))
                             mtext.setBackgroundColor(Color.GREEN);
        }
        }
```

Adding Fragments to Activities

After creating both fragments and its functionalities, now add these fragments to activities. Implement two activities for showing the fragments, one for each. Before implementing activities, create xml files for both of the fragments. The following code snippets show the fragments declaration in xml files:

fragmenta.xml

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

```
                android:id="@+id/titles"
                android:name="com.sample.frags.FragmentA"
                android:layout_width="match_parent"
                android:layout_height="match_parent" />
```

fragmentb.xml

```
        <?xml version="1.0" encoding="utf-8"?>
        <fragment xmlns:android="http://schemas.android.com/apk/res/android"
                android:id="@+id/viewers"
                android:name="com.sample.frags.FragmentB"
                android:layout_width="match_parent"
                android:layout_height="match_parent" />
```

Now create two activities to host these two fragments. Let us name it as TitleActivity and DetailActivity forFragmentA and FragmentB, respectively.

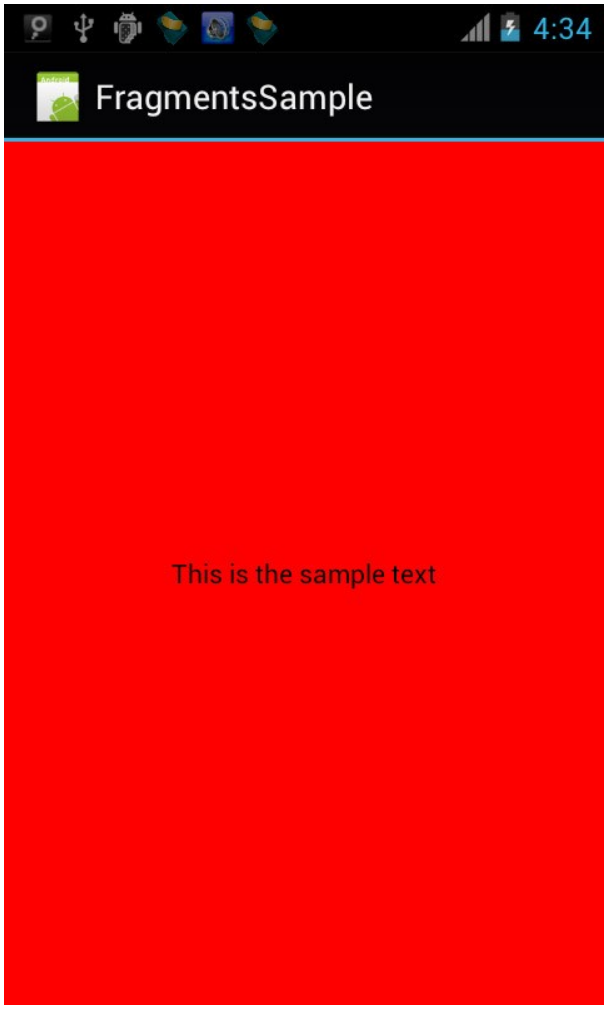
TitleActivity

```
        public class TitleActivity extends Activity{
                @Override
                protected void onCreate(Bundle savedInstanceState) {
                        // TODO Auto-generated method stub
                        super.onCreate(savedInstanceState);
                        setContentView(R.layout.fragmenta);
                }
        }
```

DetailActivity

```
        public class DetailActivity extends Activity{
                @Override
                protected void onCreate(Bundle savedInstanceState) {
                        // TODO Auto-generated method stub
                        super.onCreate(savedInstanceState);
                        setContentView(R.layout.fragmentb);
                }
        }
```

When you run the application you observe the same result as if you had used two different activities. However, here the same functionality is implemented with fragments.



Working in Landscape mode

Now modify the code to use the fragments feature in landscape mode. Copy the fragmenta xml file and place it in the/res/layout-land folder. Modify the fragmenta.xml file as shown below:

the fragmenta.xml file under /res/layout remains unchanged.

fragmenta.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="horizontal">
        <fragment
                android:name="com.sample.frags.FragmentA"
                android:layout_width="0dp"
                android:layout_height="match_parent"
                android:id="@+id/titles"
                android:layout_weight="45">
        </fragment>
```

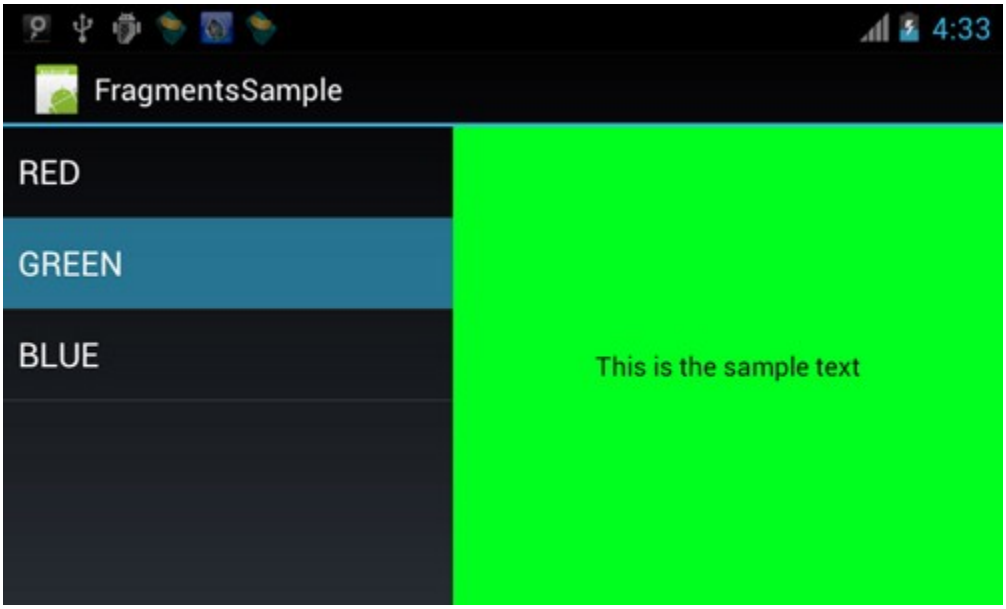
```
            <fragment
                    android:name="com.sample.frags.FragmentB"
                    android:layout_width="0dp"
                    android:layout_height="match_parent"
                    android:id="@+id/viewers"
                    android:layout_weight="55">
            </fragment>
    </LinearLayout>
```

Change the code snippet shown below for onListItemClick() method in FragmentA class to achieve the fragment functionality.

FragmentA class

```
    @Override
    public void onListItemClick(ListView l, View v, int position, long id) {
            // TODO Auto-generated method stub
            super.onListItemClick(l, v, position, id);
    FragmentB viewer = (FragmentB) getFragmentManager().findFragmentById(R.id.viewers);
            if (viewer == null || !viewer.isInLayout()) {
              Intent launchingIntent = new Intent(getActivity(),DetailActivity.class);
              launchingIntent.putExtra("COLOR_TYPE", tutorialList[position]);
              startActivity(launchingIntent);
            } else {
              viewer.updateColor(tutorialList[position]);
    }
    }
```

Now run the application and the following output appears in landscape mode. In portrait mode it remains unchanged.



Fragment Transactions

One of the most important features of fragment is fragment transactions. By using fragment transactions, developers can add, remove and replace the fragments in response to end user interaction. Each set of operations that are performed is called a transaction. Developers can also save each transaction to a back stack managed by the activity for future use. To implement the fragment transactions in the sample, modify the fragmenta.xml file in layout-landfolder.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="horizontal"
   android:id="@+id/fragment_container">
<fragment
   android:name="com.sample.frags.FragmentA"
   android:layout_width="100dp"
   android:layout_height="match_parent"
   android:id="@+id/titles">
</fragment>
<fragment
   android:name="com.sample.frags.FragmentB"
   android:layout_width="200dp"
   android:layout_height="match_parent"
   android:id="@+id/viewers">
</fragment>
</LinearLayout>
```

Add one more fragment to the existing linear layout when the user clicks the "Blue" item. For this, a FragmentC class is necessary.

FragmentC class

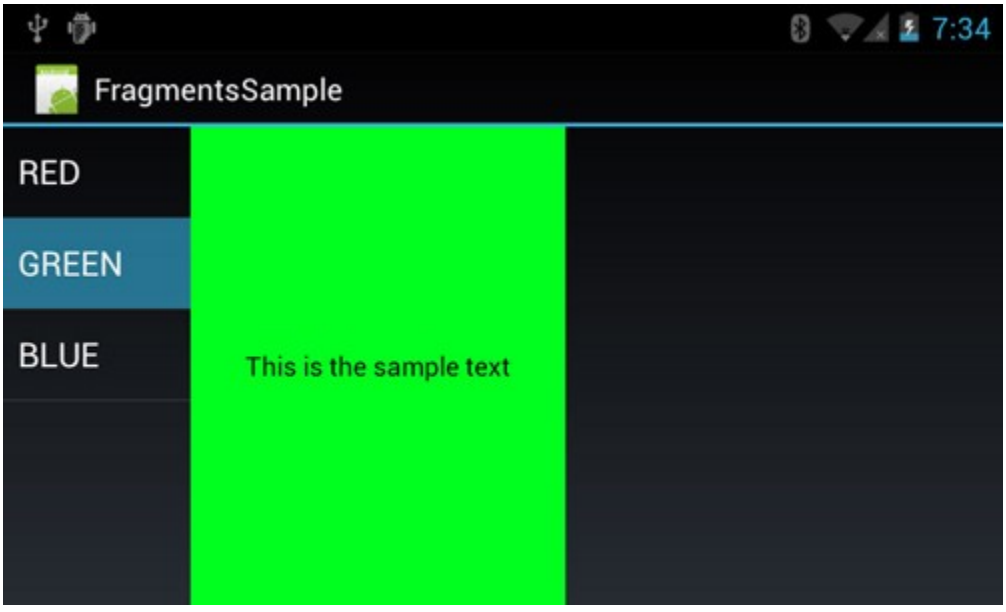
```
public class FragmentC extends Fragment{

TextView mtext;
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
        // TODO Auto-generated method stub
        TextView mtext  = new TextView(getActivity());
        mtext.setText("Fragment C added. \n Click back button to go
previous state");
        mtext.setBackgroundColor(Color.MAGENTA);
        mtext.setLayoutParams(new LayoutParams(LayoutParams.MATCH_PARENT,
 LayoutParams.MATCH_PARENT));
        return mtext;
```

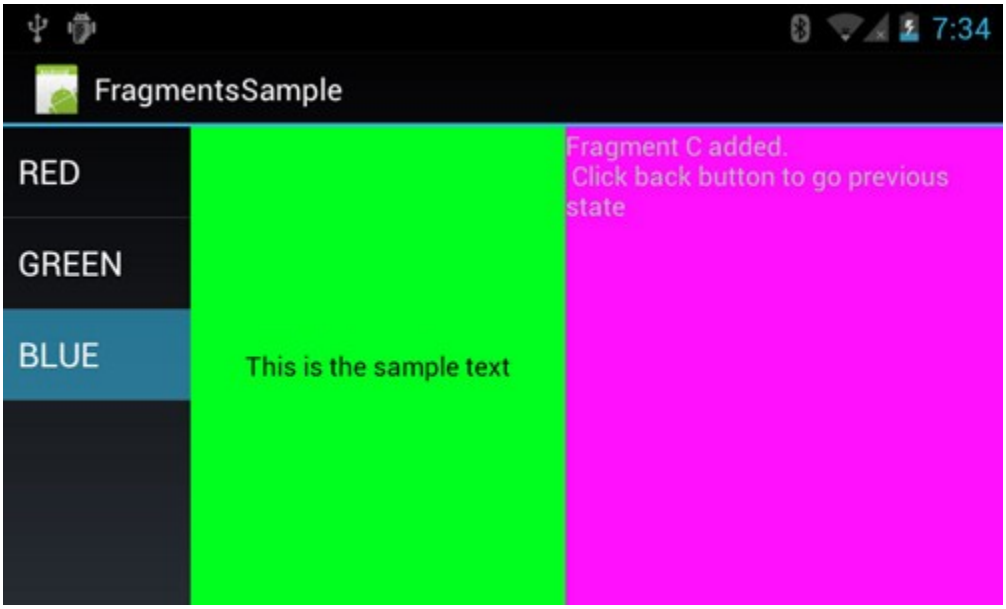
```
        }
        }
```

When user clicks the "Blue" item, the fragment c will be added to the existing activity. To do this, modify theonListItemClick() in the FragmentA class.

```
        @Override
        public void onListItemClick(ListView l, View v, int position, long id) {
                // TODO Auto-generated method stub
                super.onListItemClick(l, v, position, id);
                if(position == 2)
                {
                // reference to Fragmentc
                Fragment newFrag = new FragmentC();
                // starting the fragment transaction
                FragmentTransaction trans = getFragmentManager().beginTransaction();
                // adding the fragment reference to the existing linearlayout
                trans.add(R.id.fragment_container, newFrag);
                // adding transation to the history stack
                trans.addToBackStack(null);
                //coming the trasaction finally.
                trans.commit();
        }
        else
        {
        ...
        }
        }
```

If you run the application now, you will get the following screens.





ref:http://developer.samsung.com/android/technical-docs/Using-Fragments-to-Build-User-Interfaces-in-Android