Tab Layout

To create a tabbed UI, you need to use a `TabHost` and a `TabWidget`. The `TabHost` must be the root node for the layout, which contains both the `TabWidget` for displaying the tabs and a `FrameLayout` for displaying the tab content.

You can implement your tab content in one of two ways: use the tabs to swap `View`s within the same `Activity`, or use the tabs to change between entirely separate activities. Which method you want for your application will depend on your demands, but if each tab provides a distinct user activity, then it probably makes sense to use a separate `Activity` for each tab, so that you can better manage the application in discrete groups, rather than one massive application and layout.

In this tutorial, you'll create a tabbed UI that uses a separate `Activity` for each tab.

1. Start a new project named *HelloTabWidget*.
2. First, create three separate `Activity` classes in your project: `ArtistsActivity`, `AlbumsActivity`, and `SongsActivity`. These will each represent a separate tab. For now, make each one display a simple message using a `TextView`. For example:

```
public class ArtistsActivity extends Activity {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        TextView textview = new TextView(this);
        textview.setText("This is the Artists tab");
        setContentView(textview);
    }
}
```

   Notice that this doesn't use a layout file. Just create a `TextView`, give it some text and set that as the content. Duplicate this for each of the three activities, and add the corresponding `<activity/>` tags to the Android Manifest file.
3. You need an icon for each of your tabs. For each icon, you should create two versions: one for when the tab is selected and one for when it is unselected. The general design recommendation is for the selected icon to be a dark color (grey), and the unselected icon to be a light color (white). (See the Icon Design Guidelines.) For example:

   For this tutorial, you can copy these images and use them for all three tabs. (When you create tabs in your own application, you should create customized tab icons.)

   Now create a state-list drawable that specifies which image to use for each tab state:

   1. Save the icon images in your project `res/drawable/` directory.
   2. Create a new XML file in `res/drawable/` named `ic_tab_artists.xml` and insert the following:

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- When selected, use grey -->
    <item android:drawable="@drawable/ic_tab_artists_grey"
          android:state_selected="true" />
    <!-- When not selected, use white-->
    <item android:drawable="@drawable/ic_tab_artists_white" />
</selector>
```

This is a state-list drawable, which you will apply as the tab image. When the tab state changes, the tab icon will automatically switch between the images defined here.

4. Open the `res/layout/main.xml` file and insert the following:

```
<?xml version="1.0" encoding="utf-8"?>
<TabHost xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@android:id/tabhost"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <LinearLayout
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:padding="5dp">
        <TabWidget
            android:id="@android:id/tabs"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content" />
        <FrameLayout
            android:id="@android:id/tabcontent"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:padding="5dp" />
    </LinearLayout>
</TabHost>
```

This is the layout that will display the tabs and provide navigation between each `Activity` created above.

The `TabHost` requires that a `TabWidget` and a `FrameLayout` both live somewhere within it. To position the `TabWidget` and `FrameLayout` vertically, a `LinearLayout` is used. The `FrameLayout` is where the content for each tab goes, which is empty now because the `TabHost` will automatically embed each `Activity` within it.

Notice that the `TabWidget` and the `FrameLayout` elements have the IDs `tabs` and `tabcontent`, respectively. These names must be used so that the `TabHost` can retrieve references to each of them. It expects exactly these names.

5. Now open `HelloTabWidget.java` and make it extend `TabActivity`:

```
public class HelloTabWidget extends TabActivity {
```

6. Use the following code for the `onCreate()` method:

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    Resources res = getResources(); // Resource object to get Drawables
    TabHost tabHost = getTabHost();  // The activity TabHost
    TabHost.TabSpec spec;  // Resusable TabSpec for each tab
    Intent intent;  // Reusable Intent for each tab

    // Create an Intent to launch an Activity for the tab (to be reused)
    intent = new Intent().setClass(this, ArtistsActivity.class);

    // Initialize a TabSpec for each tab and add it to the TabHost
    spec = tabHost.newTabSpec("artists").setIndicator("Artists",
                      res.getDrawable(R.drawable.ic_tab_artists))
                  .setContent(intent);
    tabHost.addTab(spec);

    // Do the same for the other tabs
```

```
    intent = new Intent().setClass(this, AlbumsActivity.class);
    spec = tabHost.newTabSpec("albums").setIndicator("Albums",
                      res.getDrawable(R.drawable.ic_tab_albums))
                  .setContent(intent);
    tabHost.addTab(spec);

    intent = new Intent().setClass(this, SongsActivity.class);
    spec = tabHost.newTabSpec("songs").setIndicator("Songs",
                      res.getDrawable(R.drawable.ic_tab_songs))
                  .setContent(intent);
    tabHost.addTab(spec);

    tabHost.setCurrentTab(2);
}
```

This sets up each tab with their text and icon, and assigns each one an `Activity`.

A reference to the `TabHost` is first captured with `getTabHost()`. Then, for each tab, a `TabHost.TabSpec` is created to define the tab properties. The `newTabSpec(String)` method creates a new `TabHost.TabSpec` identified by the given string tag. For each `TabHost.TabSpec`, `setIndicator(CharSequence, Drawable)` is called to set the text and icon for the tab, and `setContent(Intent)` is called to specify the `Intent` to open the appropriate `Activity`. Each `TabHost.TabSpec` is then added to the `TabHost` by calling `addTab(TabHost.TabSpec)`.

At the very end, `setCurrentTab(int)` opens the tab to be displayed by default, specified by the index position of the tab.

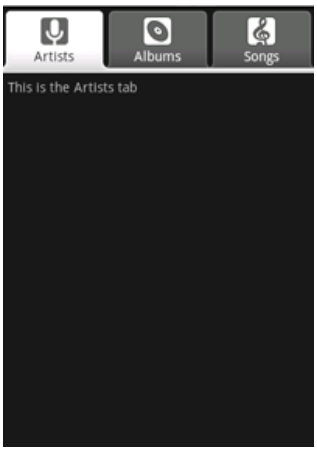
Notice that not once was the `TabWidget` object referenced. This is because a `TabWidget` must always be a child of a `TabHost`, which is what you use for almost all interaction with the tabs. So when a tab is added to the `TabHost`, it's automatically added to the child `TabWidget`.

7. Now open the Android Manifest file and add the `NoTitleBar` theme to the *HelloTabWidget*'s `<activity>` tag. This will remove the default application title from the top of the layout, leaving more space for the tabs, which effectively operate as their own titles. The `<activity>` tag should look like this:

```
<activity android:name=".HelloTabWidget" android:label="@string/app_name"
          android:theme="@android:style/Theme.NoTitleBar">
```

8. Run the application.

Your application should look like this (though your icons may be different):



References

- `TabWidget`
- `TabHost`

- `TabHost.TabSpec`
- `FrameLayout`

← Back to Hello, Views

↑ Go to top