A Guide to Android Graphical Layouts

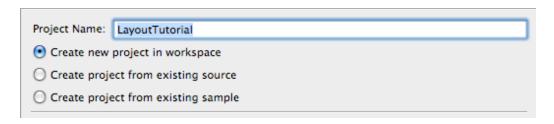
Michael Allon March 29, 2012

Abstract: This document is a guide for creating graphical layouts for Android phone application development. It is not intended to encompass every detail of the subject, but as a starting point for those interested in Android layouts and Android development. The reader will be guided through a tutorial program which gives examples of different layout elements and styles. It is assumed that the reader has Eclipse installed and running on his/her computer. It also assumes that the reader has a basic understanding of Java and object-oriented programming.

A Guide to Android Graphical Layouts

This document assumes that you have Eclipse installed and running on your machine. It also assumes you have a basic understanding of Java and object-oriented programming.

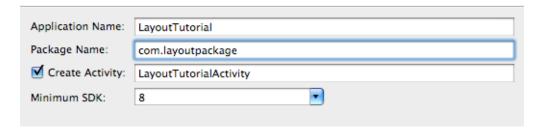
Open Eclipse and select a workspace. Click File New Other and click on Android Project. You will be prompted for a name for the project. Call it "LayoutTutorial" or any other name you want to call it. Make sure *Create new project in workspace* and *Use default location* are selected.



Click Next and select an SDK to use. In this tutorial we will be using Android 2.2

Target Name	Vendor	Platform	API Leve
✓ Android 2.2	Android Open Source Project	2.2	8
Google APIs	Google Inc.	2.2	8
DTS Add-On	KYOCERA Corporation	2.2	8
Real3D Add-On	LGE	2.2	8
GALAXY Tab Addon	Samsung Electronics Co., Ltd.	2.2	8

Now specify a package name for the application. We will call our package com.layoutpackage. If it is not checked already, click on the *Create Activity* box.



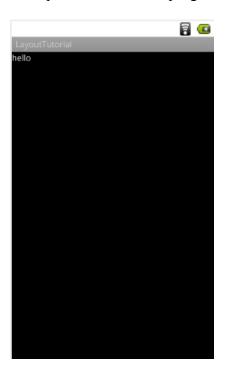
Eclipse has now created a new android project and generated several folders and files needed for the project. Open the file located in *LayoutTutorial / src / com.layoutpackage / LayoutTutorialActivity*.

This is the main activity which the program will run. We will be making very few changes to this file in this tutorial because we will focus on layouts more than functionality. For now, pay attention to the following line in the *onCreate* function:

setContentView(R.layout.main);

This line of code lets the program know to use the layout located in *res / layout / main.xml*. Next open the *main.xml* file.

Eclipse lets us view xml layout files in two ways; as a Graphical Layout or as a regular xml file. Click on the *Graphical Layout* tab. You should now see a black rectangle with some writing at the top. This is how the program will look when it is run.

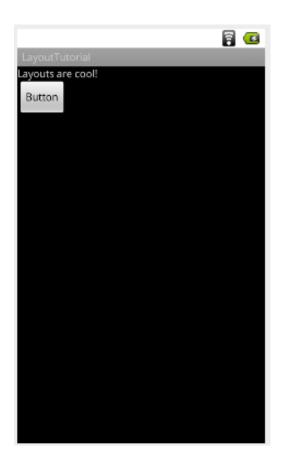


The line of text at the top displays the application name, and the line just below it shows a message. The application is called "LayoutTutorial" and the message we are displaying is "hello".

We can change what is displayed by editing the main.xml file. Switch views to regular xml by clicking the main.xml tab.

In *main.xml*, you will find an xml tag called *LinearLayout* and, within it, a tag called *TextView*. We can change the line within the *TextView* tag that says android:text=@string/hello to say android:text="Layouts are cool!". Save this and return to the Graphical Layout view. The message now reads "Layouts are cool!" instead of "hello".

Now lets add something else to the layout. In the Graphical Layout view, you will see a header on the left called *Palete* with a list of items beneath it. These items are standard andriod elements that we will use to create our layout. Click on *Form Widgets* and a list of elements will appear. Click and drag the *Button* element and drop it onto the black rectangle. The layout should now look like this:



The layout is starting to look better, but the button and text are not centered. We can fix in one of two ways. First, we can switch back to the *main.xml* tab. Add the following line to the TextView tag:

Android:gravity="center"

So for the TextView we now have:

<TextView

android:layout_width="fill_parent"

android:layout_height="wrap_content"

android:gravity="center"

android:text="Layouts are cool!" />

The text "Layouts are cool!" should now be centered horizontally in the layout. We will use a shortcut to do the same for the button. Click on the button to highlight it. Next click on the

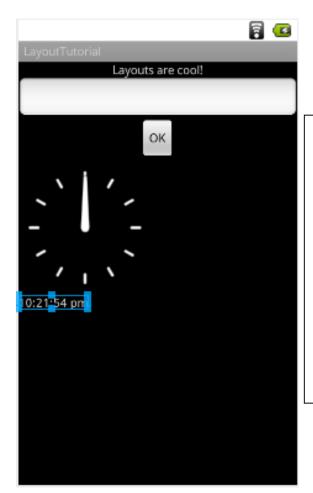
Change Gravity button near the top which looks like four arrows pointing outward. From the dropdown menu select Center Horizontal. The button is now centered just like the text. We can also change the text displayed by the button by editing the main.xml file:

android:text="OK"

Now lets add a text field. Under the *Palette* header select *Text Field*. Drag and drop the *Plain Text* item into the layout just above the Button. When the program runs, the user can edit this text field and the program can use the entered data.

Android provides many different components for use in applications. For example, lets add an analog clock to the xml layout. Under the *Palette* header click on *Time and Date*. Now select *AnalogClock*. Drag this onto the Graphical Layout just below the button.

If we have trouble reading the analog clock we can add a digital clock next to it. Under *Date and Time* select *DigitalClock*. Drag this anywhere onto the Graphical Layout. Our layout should now look like this:

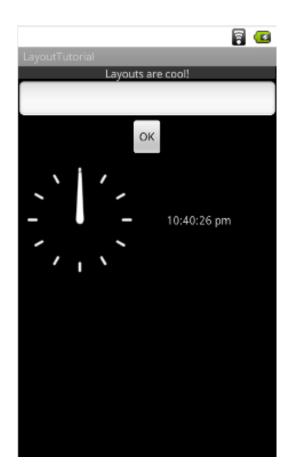


Up until this point we have been using a veritcal linear layout to display all our elements. Now, however, the application would look nicer if the two clocks were side-by-side horizontally.

Hold down Shift and select both of the clocks. Now right-click and select *Wrap in Container* from the dropdown menu. You will now be asked to specify a container type. Select LinearLayout(Horizontal). You must also give the layout an ID. This ID is unimportant for this tutorial and we will name it "LH1".

The two clocks are now displayed horizontally next to each other.

However, the digital clock is still offset and appears higher than the analog clock. We can fix this by highlighting the digital clock and clicking the *Change Gravity* button. Select center vertical and the digital clock is now vertically aligned with the analog clock. The clocks may appear too close together now, but we can fix this by highlighting the digital clock and clicking the *Change Margins* button. In the menu that pops up, change the "Left:" field to "30dp". This will offset the digital clock by 30 pixels. The application now looks like this:



The application is coming along nicely. We have a header message which says "Layouts are cool!", a text field for entering data, a button, and two clocks.

However the program doesn't actually do anything so far. Next we will add functionality to the button.

In the regular *main.xml* view, change the line:

Android:id ="@+id/button1"

To..

Android:id="@+id/bOK"

This is how we will refer to the button our main Java class.

Now go back to the LayoutTutorialActivity.java file. We need to let this Java class know about the OK button. Before the line "@Override" enter the following line:

Button okCommand;

We also need to import the Android Button package, so at the top of the file write:

Import android.widget.Button;

Now in the *onCreate* method write the following:

okCommand = (Button) findViewById(r.id.bOK); okCommand.setOnClickListener(this);

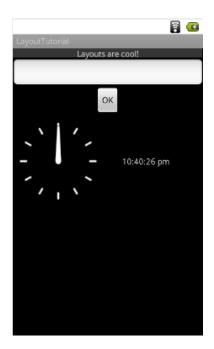
The first line locates the button in our resources folder and the second line helps to set up an event for when the button is clicked. Eclipse will show an error at this point because setOnClickListener is undefined. We need to first:

Import android.view.View;

Then we need our activity to implement View.OnClickListener. Change the class definition line to:

Eclipse will still give an error, but this is easily fixed by adding the View class' unimplemented method *onClick(View v)*. In this tutorial we will not go into detail on what to implement inside the *onClick* method. This will vary depending on the application and is not relevant to the topic of layouts. However, for future applications, this will be a key interface between Java code and the android xml layout.

So far the application still looks like this:



But what if we want a background to look like this \rightarrow



To change the background of our layout open the *main.xml* file you must save the PNG image you want to use in the *res / drawable-hdpi* folder and add the following line to the *LinearLayout* tag:

android:background="@drawable/spartan"

Replace "spartan" here with the name of the image you would like as a background.

This concludes the tutorial on Android layouts. We have only begun to scratch the surface of the topic in this document. For further reading and documentation see resources such as the following:

Android Developers: The Developers Guide http://developer.android.com/guide/index.html