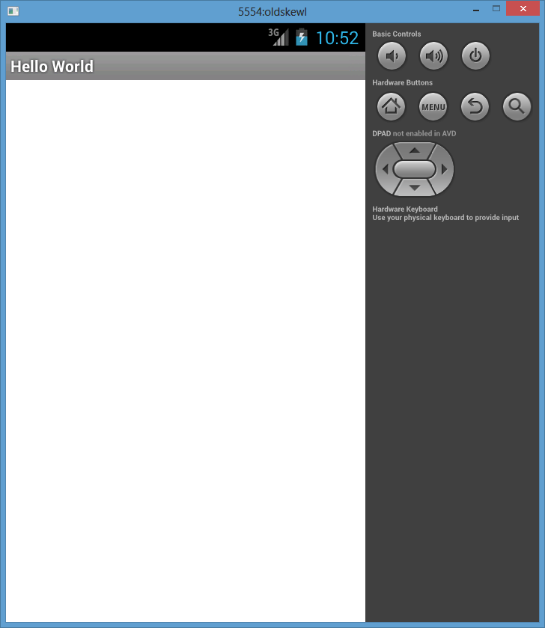
# Lab 1: Your First Android App

Welcome to the world of Android App development. Today we are going to jump right in and start building an Android app. First thing is first, **Open the Hello World Project** with Eclipse. Now let’s run our Android App in the Emulator. Right click the Hello World project folder and choose **Run As --> Android Application**. You will notice the Android Emulator starts up and Runs the Hello World App. Awesome!



Well, not so awesome yet. The app is very boring and doesn’t do anything yet. It’s just a blank screen with a title bar. Now let’s add some text to display to the user. Navigate to the **res/layout/activity\_main.xml** file. This file is known as a Layout File. Layout files define the visual structure for a User Interface. Using Android’s XML vocabulary, you can quickly design UI layouts with a series of nested elements. You can edit Layout Files in two ways, with the XML editor, or the Android Graphical UI Editor. Both display the same thing and you can toggle between the two by clicking on the “Graphical Layout” tab or the “activity\_main.xml” tab at the bottom. **Click on the “activity\_main.xml” tab** and edit your file. Insert a new TextView to display a message to the user.

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

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

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

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

android:orientation=*"vertical"* >

<TextView

android:id=*"@+id/my\_text"*

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

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

android:textSize=*"20sp"*

android:text=*"Android is fun!"*/>

</LinearLayout>

Now run the app again by choosing **Run As --> Android Application** or **Ctrl+F11** on the keyboard. Sweet, you make some text appear on the screen. Now let’s add a button to our App. Go back to the res/layout/activity\_main.xml file and add a Button element under the TextView.

<Button

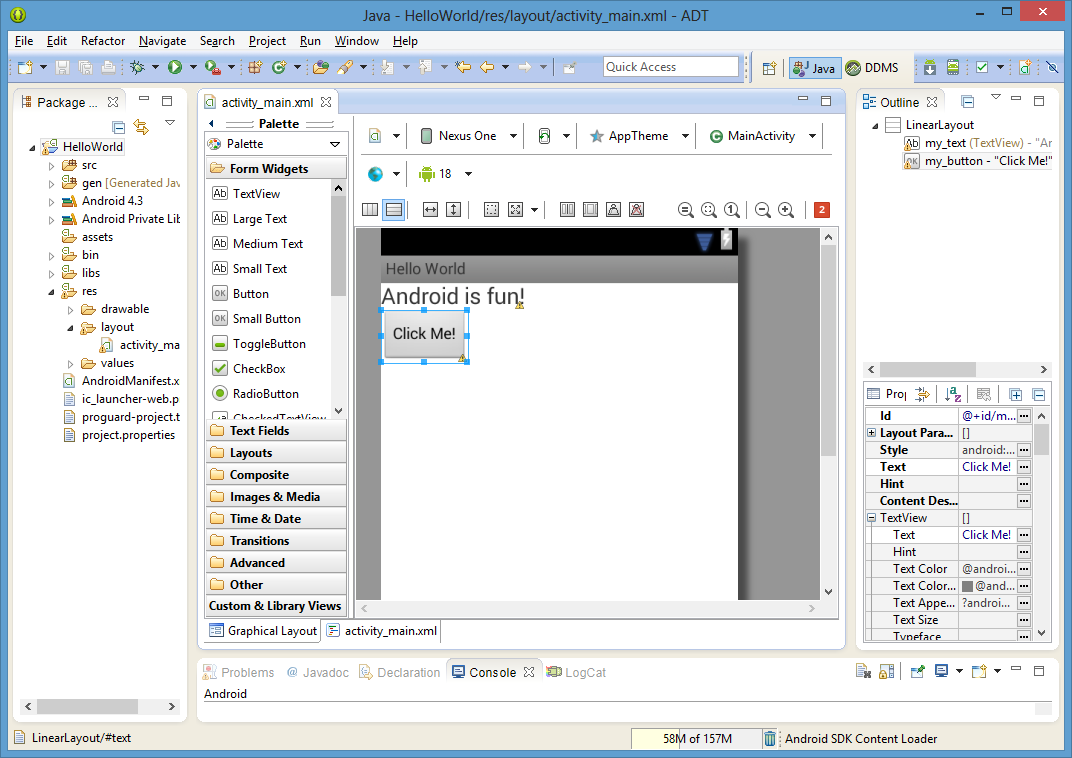
android:id=*"@+id/my\_button"*

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

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

android:text=*"Click Me!"* />

Run your app again. Checkout your new awesome button. You can get a preview of your layout before you run your app with the Android Graphical Layout tool. **Click on the Graphical Layout** tab at the bottom of the activity\_main.xml editor. The Graphical Layout editor is a really powerful tool to help you quickly build Android UI layouts.



You add new UI elements to your layout by dragging them from the Palette bar and positioning them right on your layout. Try out adding another TextView by **dragging TextView from the Palette** **onto your layout.** Pretty cool right? Now switch back to the XML editor by **clicking the activity\_main.xml tab.** Notice how a new XML TextView element has been added to the file. The graphical editor makes changes to the XML code in real time. As you move around elements and change attributes with the graphical editor, it’s actually changing your code too.

Alright that’s all well and good, but now let’s have some real fun. Let’s make the background of our TextView turn green when we click the button. To do so, we will need to write some Java code. **Open src/com.example.helloworld/MainActivity.java** and edit to look like this:

**package** com.example.helloworld;

**import** android.app.Activity;

**import** android.graphics.Color;

**import** android.os.Bundle;

**import** android.view.View;

**import** android.widget.Button;

**import** android.widget.TextView;

**public** **class** MainActivity **extends** Activity {

**private** Button mMyButton;

**private** TextView mMyText;

@Override

**protected** **void** onCreate(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

setContentView(R.layout.*activity\_main*);

mMyText = (TextView)findViewById(R.id.*my\_text*);

mMyButton = (Button) findViewById(R.id.*my\_button*);

mMyButton.setOnClickListener(**new** View.OnClickListener(){

@Override

**public** **void** onClick(View v) {

mMyText.setBackgroundColor(Color.*GREEN*);

mMyText.setText("Look at my green text");

}

});

}

}

Run your app and try it out.

You should now be able to change the text and the color when you click the button.

Alright, now let’s spice up our app even more. Let’s add a sweet image to our UI. We will use a new UI element called an ImageView to display our image. Go back to the activity\_main.xml and **add a new ImageView to the UI.** You can choose to add one by dragging one from the tool pallet or by editing the XML.

<ImageView

android:id=*"@+id/my\_image"*

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

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

android:src=*"@drawable/apple"* />

When you run your app again, you will notice that you now have a picture of an apple. Great. Let’s dive deeper and find out how that apple got there. The ImageView has a property called **src** that tells it what image to load and display at runtime. We set that in XML with the line: android:src=”@drawable/apple” This tells the android system to load up the apple resource image. All resources are found in the /res project folder. **Look in the /res/drawable** **folder** and notice it contains a few image files including apple.jpg. The Android build system turns all the files under the /res folder into resource. Resources can be referenced with the special notation that begins with ‘@’ in XML files. Also, resources files can be referenced in Java code.

Let’s see an example of using Java code to change the image when we click our button. **Switch back to MainActivity.java.** Add this code:

**...**

**import** android.widget.TextView;

...

**public** **class** MainActivity **extends** Activity {

**...**

**private** ImageView mMyImage;

**protected** **void** onCreate(Bundle savedInstanceState) {

**...**

mMyImage = (ImageView)findViewById(R.id.*my\_image*);

...

mMyButton.setOnClickListener(**new** View.OnClickListener(){

@Override

**public** **void** onClick(View v) {

...

mMyImage.setImageResource(R.drawable.*pizza*);

}

});

}

}

Ok so what’s going on here? First, we get a reference to the my\_image view with the findViewById(). Notice the use of R.id.my\_image. Every view element in an Android UI can optionally be assigned a unique ID. Our ImageView was declared with the XML: android:id="@+id/my\_image". Then, inside the onClick method, we change the ImageView’s image by calling setImageResource method. Notice we pass in the R.drawable.pizza. In Java code, all Android resources can be referred to with the R class. R is actually an automatically generated Java class that has references to all files under the /res project folder. To learn more about accessing resources, check out the Android documentation: <http://developer.android.com/guide/topics/resources/accessing-resources.html>

This concludes Lab 1. As a refresher, you have learned:

* How to start a brand new Android projects
* How to use the graphical and xml editor to add/move new UI elements
* How to add a onClick handler to a button
* How to refer to Android resource image files and change an ImageView image