

Android Development 101

Now that we have the Android SDK, Eclipse and Phones all ready to go we can jump into actual Android development.

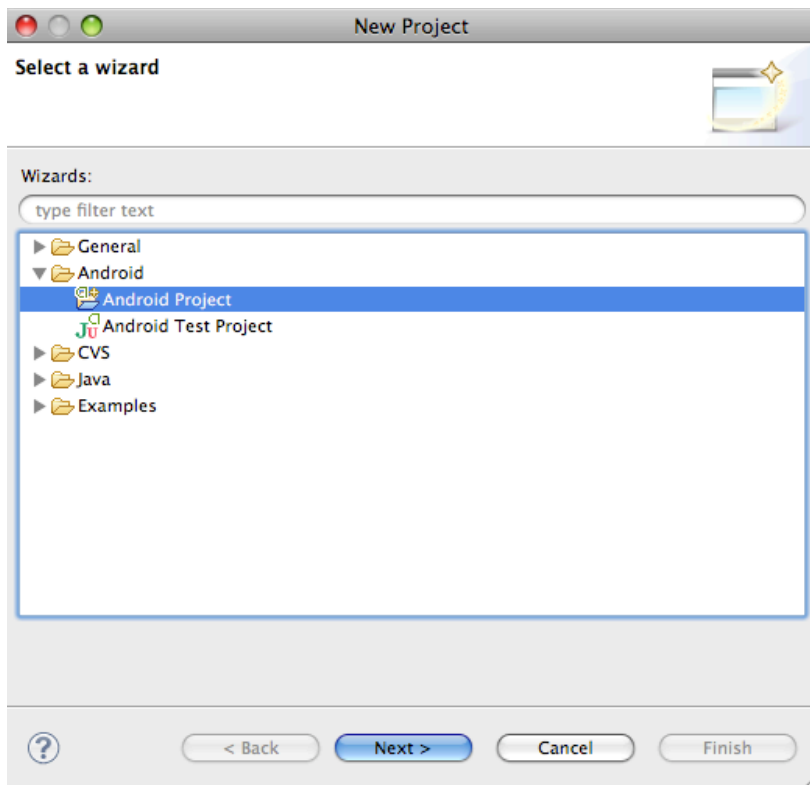
Activity

In Android, each application (and perhaps each screen in an application) is an Activity.

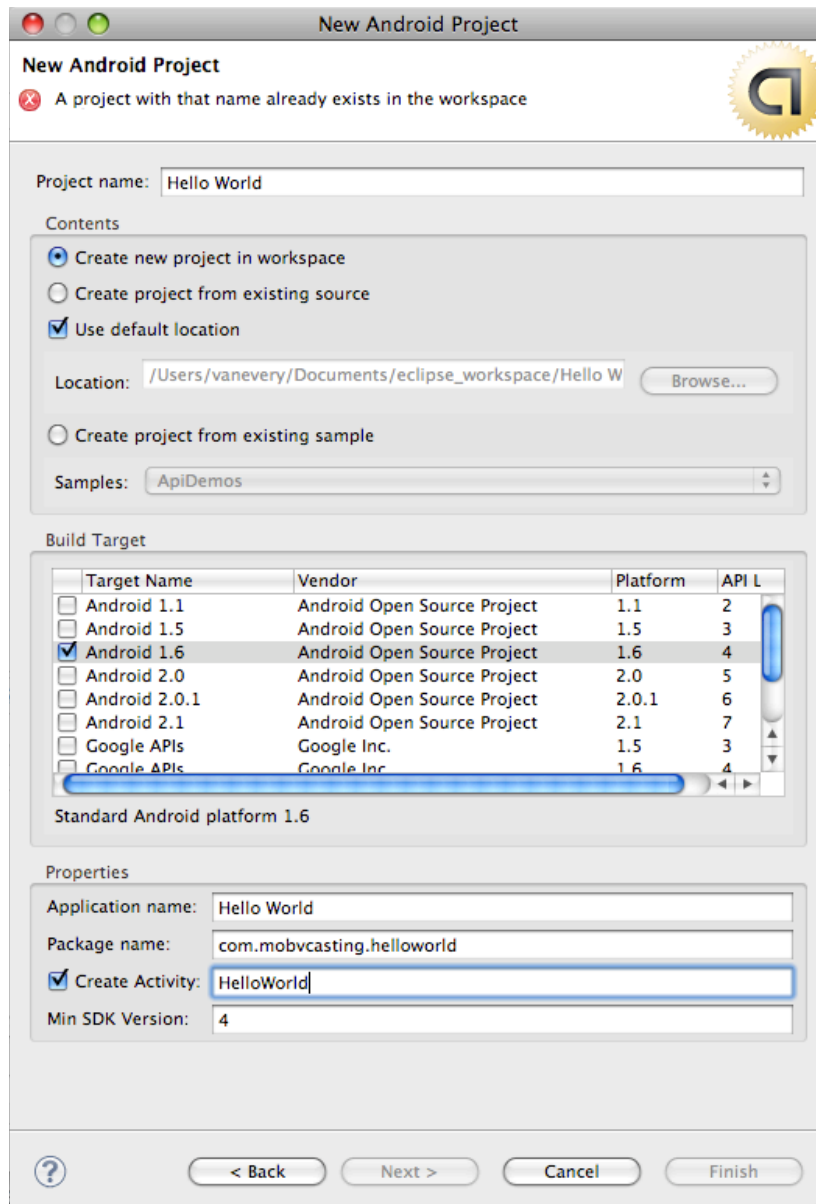
Hello World

Let's start with the proverbial Hello World example.

In Eclipse, choose "File", "New", "Project", "Android", "Android Project"



Following that, you should be given a "New Android Project" dialog:



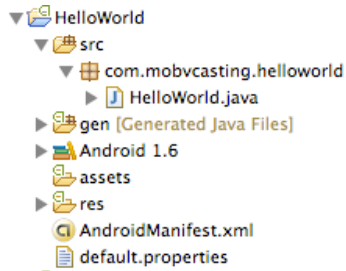
The details here are somewhat important. The "Project name" can be whatever you like it to be and will only be referenced in Eclipse. It should be "Create a new project in the workspace".

The "Build Target" is the next important bit. It should target the platform you intend to develop for. Android is backwards compatible in the running of applications (meaning you can run something targeted for 1.5 on a handset running 2.1) but you should develop for the lowest target you intend to support. Generally that will be 1.5 as that has the greatest number of users at the moment and those with later versions will still be able to run the application.

Next up is the "Application name". This will be the name of the application as seen on the device. "Package name" is a Java convention and generally needs to have at least two words (with periods between) and be unique (globally). The convention is that you use your domain name in reverse followed by something unique to the application itself.

Last is the "Activity". This is the name of the initial class that will be run or the default "Activity" that will be launched.

Clicking "Finish" should bring you back to the Eclipse workspace and your project should appear in your "Package Explorer" on the left.



Within the project you should see a "src" directory and inside there a ".java" file for the particular Activity you created. Double click on the ".java" file to bring up the code editor.

It should contain something like the following:

```
package com.mobvcasting.helloworld;

import android.app.Activity;
import android.os.Bundle;

public class HelloWorld extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

The top line "package..." should be the package name you specified previously. The two "import" statements bring in packages from the Android SDK that are used in this Activity.

The work begins with the "public class HelloWorld extends Activity" line. That is saying that this is a class named HelloWorld and it is derived from Activity (the base class). This means that this class has all of the properties and methods of the Activity class as well as what you might add to it.

You can examine the reference for the Activity class on developer.android.com:
<http://developer.android.com/reference/android/app/Activity.html>

The reference page gives an overview of the class, includes a list of what its base classes are, and what its methods and properties are.

You'll notice that in our HelloWorld class, we are "Overriding" the "onCreate" function. This is generally our starting point for any Activity based class. Generally you always want to call "super.onCreate(savedInstanceState)" first as this tells the derived classes (the "super") to run their onCreate methods.

Following that, is the "setContentView" method call. This actually determines what will be

displayed in the application. In this case, it is specifying "R.layout.main".

R.layout.main can be found within the "gen" directory in our Eclipse project. It is auto-generated by Eclipse and not meant to be edited. We can look at it though to see what is going on.

You should see a portion something like this:

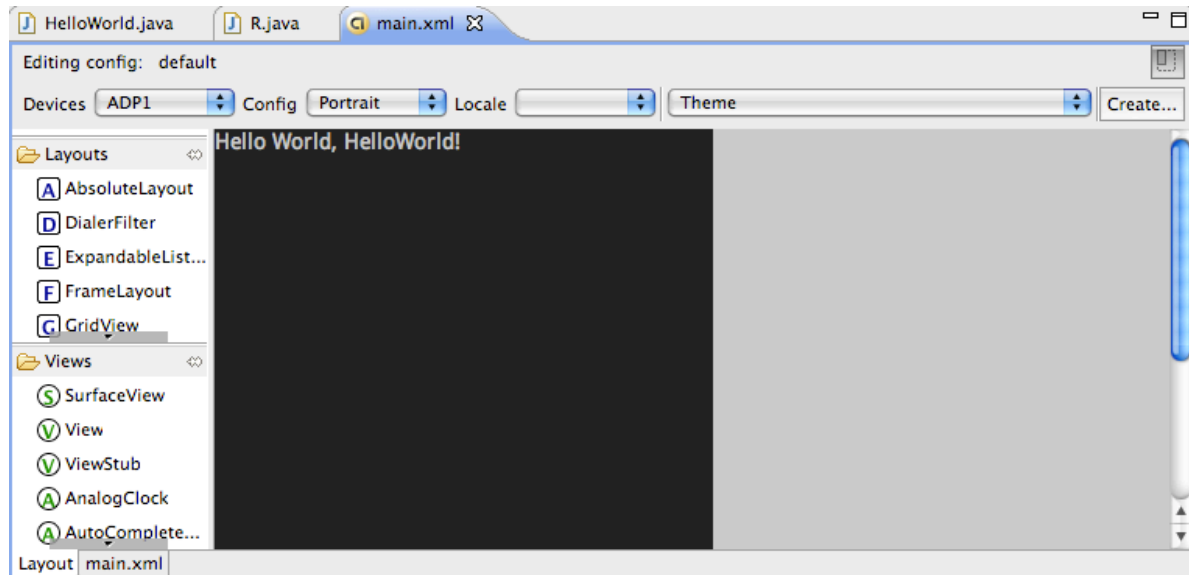
```
public static final class layout {  
    public static final int main=0x7f030000;  
}
```

Seeing that main is an "int" within the "layout" class. Behind the scenes Android is referencing our "layout" XML and making that be what is displayed.

In Android, in most cases, the UI (user interface) of an application is written in XML.

To edit this XML, in Eclipse, go to "res", "layout" and double click on the "main.xml" file.

You should see something that looks like this:



The tabs at the bottom determine your view. The default is a WYSIWYG (what you see is what you get) editor. If you click on "main.xml" you'll see the actual XML.

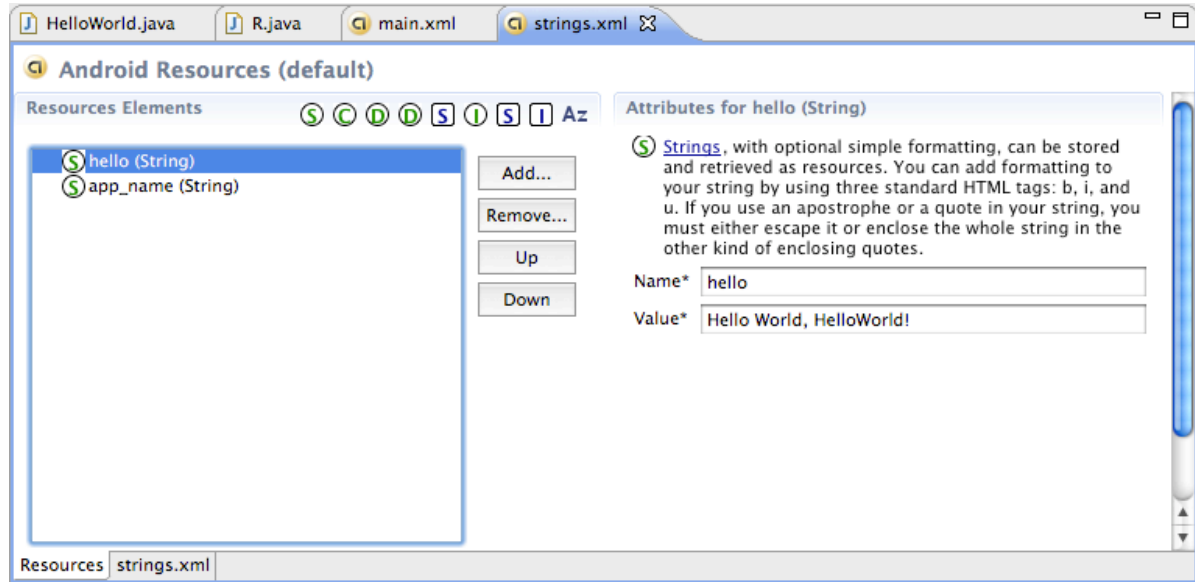
```
<?xml version="1.0" encoding="utf-8"?>  
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:orientation="vertical"  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent"  
    >  
<TextView  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:text="@string/hello"  
    />  
</LinearLayout>
```

As you can see, there is a "TextView" inside of a "Linear Layout". The "text" of the TextView is specified as "@string/hello". This means that Android will be referencing an additional file that

specifies the various strings used in the application.

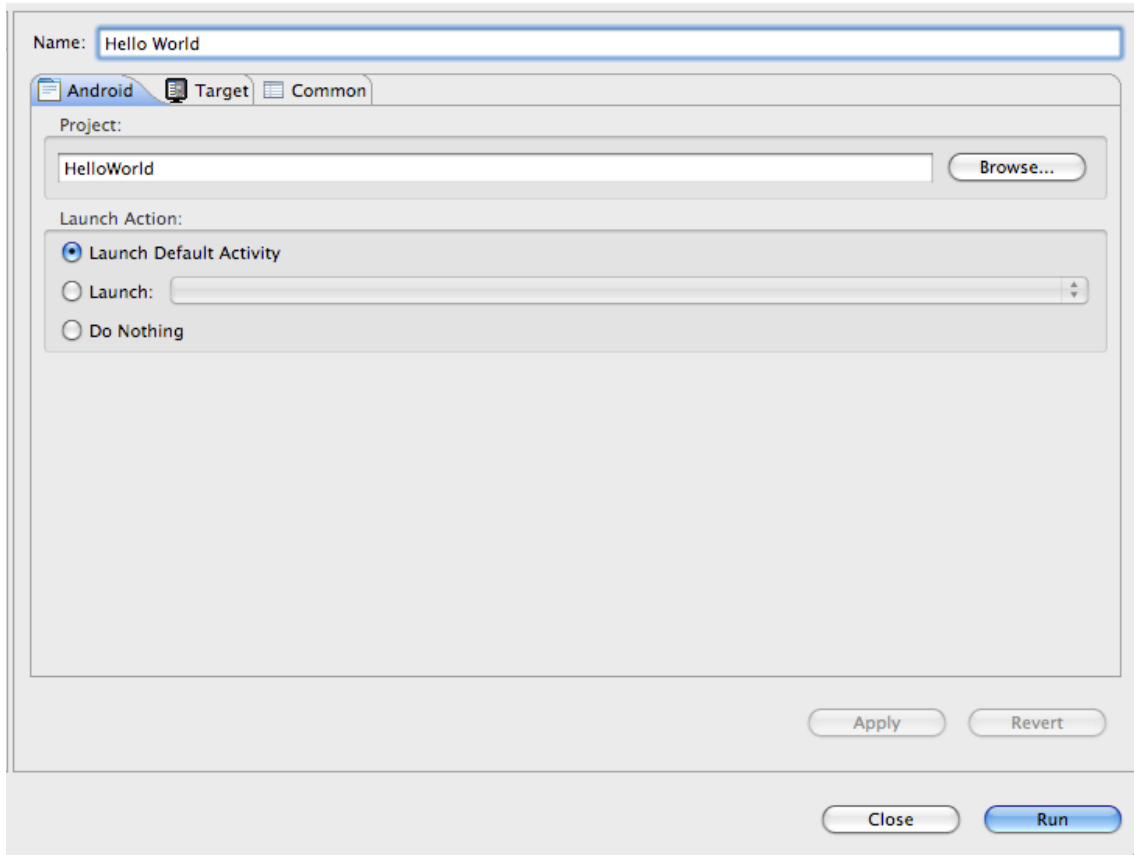
To change what this says, we need to edit that file and change the "hello" string. That file "strings.xml" is inside the "values" folder under "res" in your Eclipse project.

You can edit the value for "hello" by highlighting it, changing the "Value" and then saving it.



Once you have done that, it is time to test out the application on the Android Emulator.

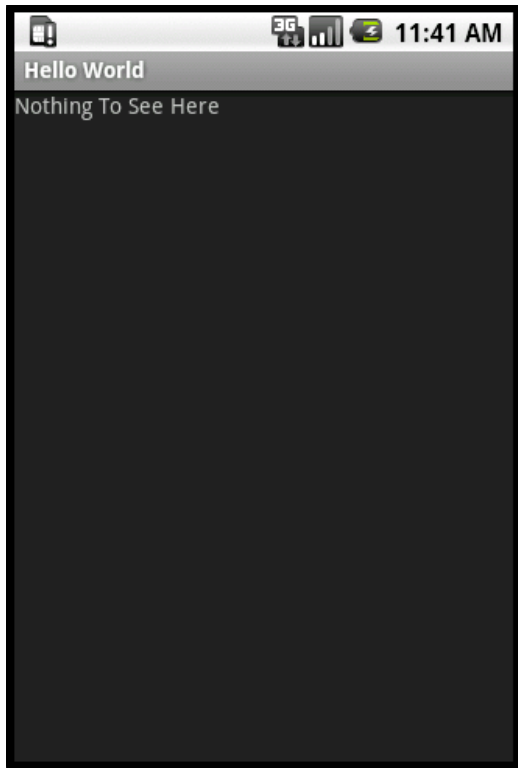
Click on "Run Configurations" under the "Run" menu, create a new configuration for this project (highlight Android Application on the left and click the paper with the plus symbol) and save it (clicking Apply).



Following that, click "Run" and your Emulator should launch and display your application.

(You may have to hit "Menu" on the emulator to get past the locked screen).

It should look something like this (the text should be what ever you filled in):



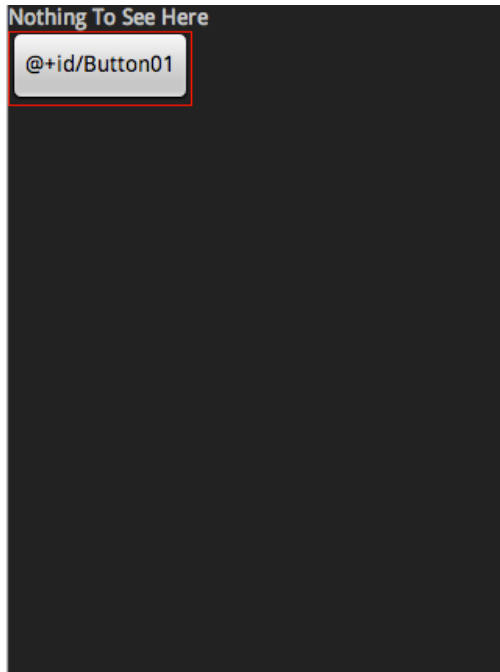
User Interface

As you see from the Hello World example, you can build user interfaces for Android in XML or with the WYSIWYG GUI editor in Eclipse.

Let's add a button to the Hello World example.

The first step is to go back to the "res", "layout", main.xml file and open it up. In the WYSIWYG editor on the left, highlight "Button" under "Views" and drag it onto the screen.

It should look something like this:



The properties of this button can be edited in the "Properties" pane at the bottom of the screen. One thing to keep in mind is that its "id" is "Button01". You can change this or leave it as is but remember what you have it set to be.

Let's change its "Text" in the "Properties" pane:

The default value is what you see on the button: "@+id/Button01" which is simply there to help you know what its "id" is. You can edit the value directly and change it to whatever you like OR you can put a new string into the "strings.xml" file and refer to it with this syntax: @string/stringname (stringname would be whatever you called the string in the strings.xml file).

Now that we have a button in our application, we have to make this button do something. In order to do that, we need to actually start writing some code so we have to start editing our .java file. Double click your "HelloWorld.java" file and add in the following code after the "super.onCreate" and the "setContentView" methods:

```
Button aButton = (Button) this.findViewById(R.id.Button01);
```

Substitute whatever you set for the "id" of the button in place of "Button01" in the line.

You should notice that "Button" is underlined in red by Eclipse. This means that there is a problem. The problem is probably that Eclipse doesn't know about the Button class. If you click error indication on the left hand side of the editor it should give you some possibilities for fixing the error. The top one will probably be to "Import 'Button'". Select that and the error should probably go away. You'll notice that an "import android.widget.Button;" line is added to your list of import statements in the top section of the code.

The statement we just added allows us to use the button in our code. Now we can tell our application what should respond to the button click itself. This is called a listener. In this case we need to make an OnClickListener.


```
aButton.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {
        // Here is where we tell it what to do
    }});
```

In addition, we have to tell our application about "OnClickListener" so we'll need an import statement:

```
import android.view.View.OnClickListener;
```

Let's change the text of the button when we click on it.

First step is to make our button be more global in scope. We need to declare it outside of the "onCreate" function. To do this, before the onCreate function, add this line:

```
Button aButton;
```

and inside the onCreate function change the "findViewById" line to this:

```
aButton = (Button) this.findViewById(R.id.Button01);
```

(just remove the "Button" from the beginning)

Inside the "onClick" function inside the OnClickListener add this line:

```
aButton.setText("You Clicked Me");
```

Your code should look like this:

```
package com.mobvcasting.helloworld;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class HelloWorld extends Activity {

    Button aButton;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        aButton = (Button) this.findViewById(R.id.Button01);
        aButton.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                aButton.setText("You Clicked Me");
            }});
    }
}
```

Now let's try it in the emulator, the same way we tested the "HelloWorld" application before except we don't need to create a new Run Configuration, we can just use the one we previously created.

Logging

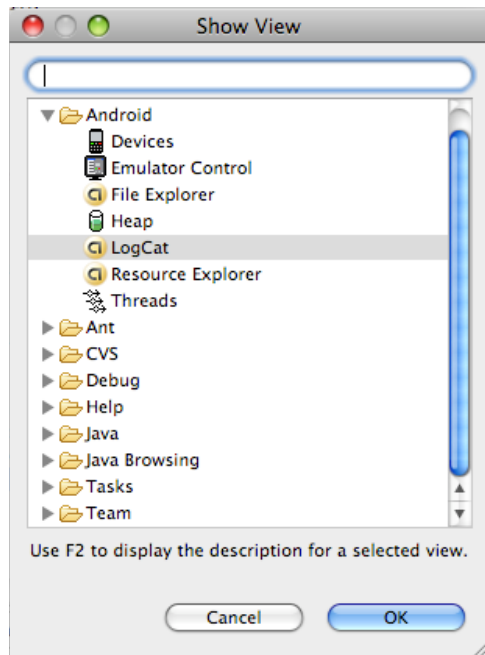
Add the following import statement to your code:

```
import android.util.Log;
```

And within the onClick method add a line like this:

```
Log.v("Clicker", "Button Clicked");
```

To see the log messages, we need to tell Eclipse to show them to us. There is a panel in Eclipse (part of the Android integration) called LogCat. To open LogCat, select Window: Show View: Other: Android: LogCat



You should see a new panel open at the bottom of your Eclipse Workspace called "LogCat". Now when you run the application, you can use the "LogCat" pane to view what is happening.

Making Toast

As proof that Android has a sense of humor there is a class called Toast which allows us to pop-up messages. Let's make our onClick method in Hello World pop-up some toast.

We'll need to import android.widget.Toast along with the rest of our import statements. Following that we can just use it. We use the static method "makeText" to create a new Toast view and then the "show" method to display it.

```
Toast t = Toast.makeText(this, "Button Clicked!", Toast.LENGTH_LONG);  
t.show();
```

Here is our full source code:

```
package com.mobvcasting.helloworld;  
  
import android.app.Activity;  
import android.os.Bundle;
```

```

import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.Toast;

public class HelloWorld extends Activity {

    Button aButton;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        aButton = (Button) this.findViewById(R.id.Button01);
        aButton.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                aButton.setText("You Clicked Me");

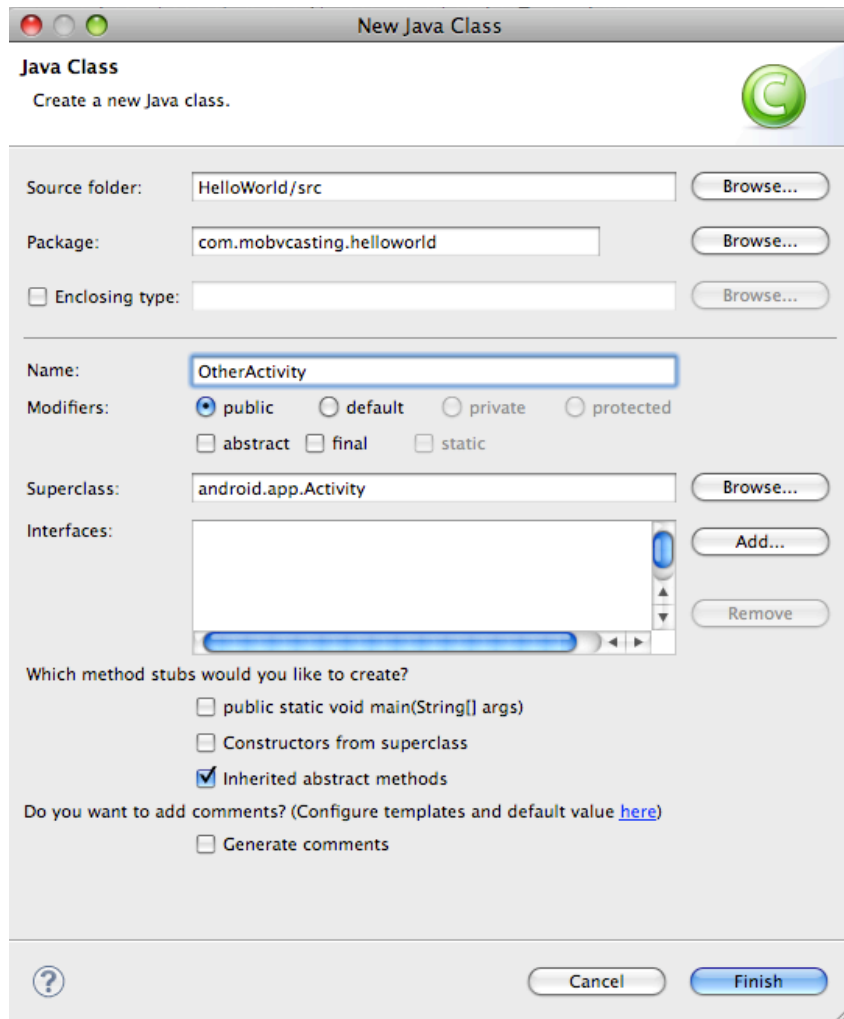
                Toast t = Toast.makeText(this, "Button Clicked!", Toast.LENGTH_LONG);
                t.show();
            }
        });
    }
}

```

Another Activity

One important thing in Android is the ability to switch between screens. The easiest way to do so is create another Activity that will display something different.

To create a new activity, choose "File", "New Class". You should get a New Java Class dialog box.



Make sure that it goes into the right project and the right directory: Project Name/src

Make sure the package name is the same as your Hello World package.

Give it a name (CamelCase is standard) and make sure it's super class is android.app.Activity.

Click Finish.

You'll then want to edit the file and add in some of the default Android methods:

```
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.other);
}
```

In the end, your class should look something like this:

```
package com.mobvcasting.helloworld;

import android.app.Activity;
import android.os.Bundle;
```

```

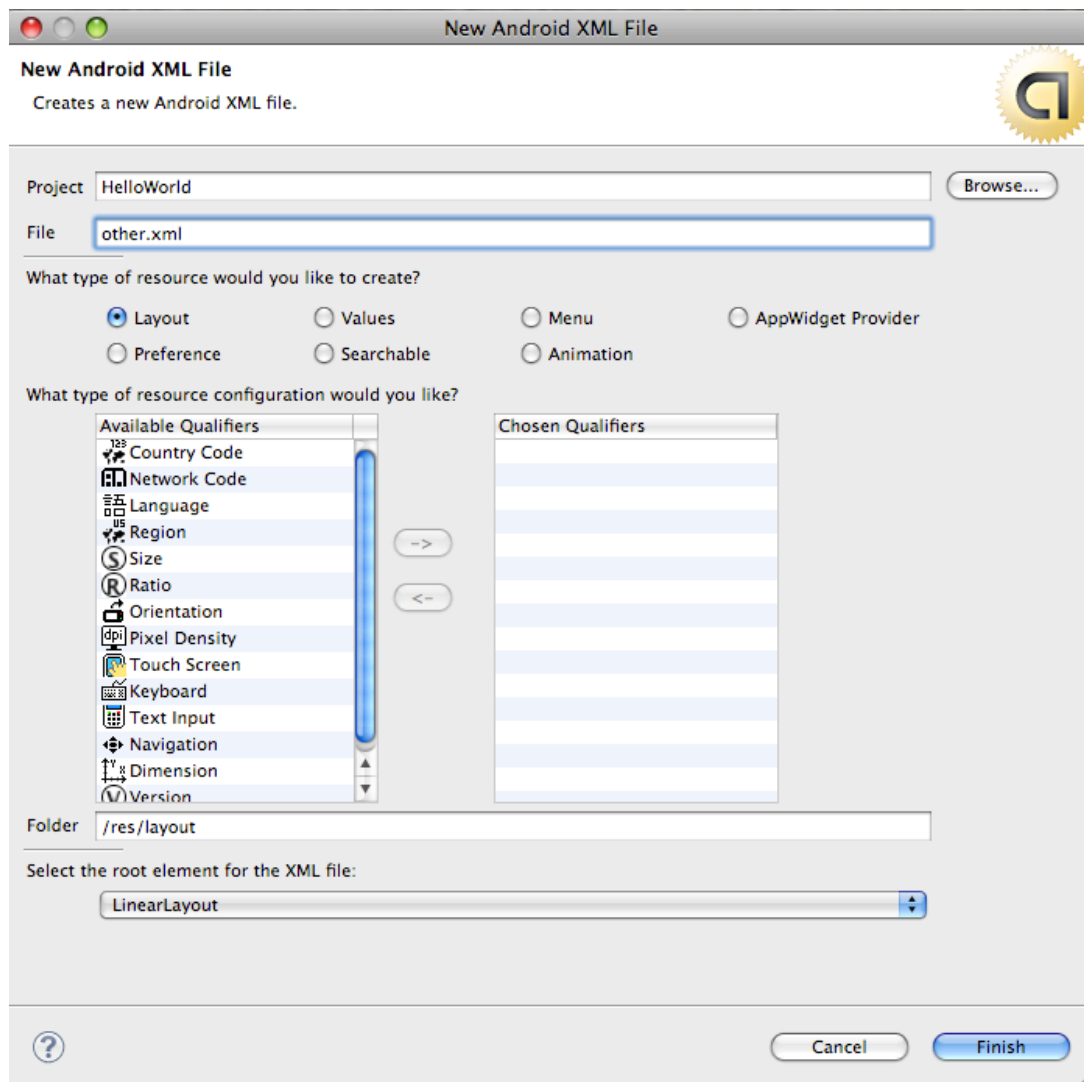
public class OtherActivity extends Activity {

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.other);
    }
}

```

You'll notice that `R.layout.other` is underlined. This means that Eclipse can't find it so we need to create a new layout for this activity.

Go to "File", "New", "Android XML File". Choose your Project, Name it: "other.xml", make it "Layout" and click Finish:



The new layout window should open in Android. Add something to it, perhaps a `TextView`.

Now you should be able to write code that allows the application to switch to your new view when

the button is pressed.

To do this, you need to use something called an Intent.

<http://developer.android.com/reference/android/content/Intent.html>

In order for your activity to launch based upon an Intent, we need to tell Android about it. To do so, we need to edit the AndroidManifest.xml file that is part of our project.

Within the XML, adding the following line:

```
<activity android:name=".OtherActivity"></activity>
```

Between the <application> tags should do it (assuming you named the class "OtherActivity").

Now you should be able to add code to the onClick method of your first Activity:

```
Intent i = new Intent(HelloWorld.this, OtherActivity.class);
startActivity(i);
```

Now when you run your application and click on the button it should show the "OtherActivity".

User Interface: Android Developers: <http://developer.android.com/guide/topics/ui/index.html>

Getting Data from another Activity

Use "startActivityForResult" passing in a constant for identifying the activity that is returning data later.

```
Intent i = new Intent(this, OtherActivity.class);
startActivityForResult(i, OTHER_ACTIVITY);
```

OTHER_ACTIVITY is an int constant defined elsewhere in the code:

```
public static final int OTHER_ACTIVITY = 0;
```

You could just as easily pass in the 0 in place of OTHER_ACTIVITY.

```
Intent i = new Intent(this, OtherActivity.class);
startActivityForResult(i, 0);
```

Previous to "finish()" in the OtherActivity Activity we can create an Intent with extra data to pass back to the original activity:

```
Bundle bundle = new Bundle();
bundle.putString("A Key", "A Value");
Intent mIntent = new Intent();
mIntent.putExtras(bundle);
setResult(RESULT_OK, mIntent);
finish();
```

In the original activity, the extra data can be pulled out as follows (in the onActivityResult method)

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent intent)
{
    super.onActivityResult(requestCode, resultCode, intent);
    Bundle extras = intent.getExtras();
```

```
switch(requestCode)
{
    // Could be case 0: or case OTHER_ACTIVITY: if we defined OTHER_ACTIVITY previously
    case OTHER_ACTIVITY:
        if (resultCode == RESULT_OK)
        {
            String theData = extras.getString("A Key");
        }
        break;
}
}
```