Lesson 2: User Interface

INTRODUCTION

All apps require some type of user interface. This lesson points out the difference between Relative and Linear Layouts using XML with Java for Android applications. The process for changing, viewing and manipulating the text in various XML layouts will be discussed.

LESSON OBJECTIVES

By the end of this lesson, the student will be able to:

1. Identify differences between relative and linear app layouts.
2. Create a graphical user interface
3. Demonstrate how to modify XML layout files.
4. Demonstrate the use of Classes and Objects within an app.
5. Define Event Listeners used in an app.
6. Demonstrate how to set up the onClickListener Event Listener.

LEARNING SEQUENCE

Required Reading	Read the following: • Online Lesson Material
Resources	View the following: • Android Application Development Tutorial – 6 – Introduction to Layouts in XML (7:33) • Android Application Development Tutorial – 7 – Creating a Button in XML and Adding an ID (7:51) • Android Application Development Tutorial – 8 – Setting up Variables and Referencing XML ids (6:59) • Android Application Development Tutorial -9 – Set up a Button with OnClick Listener (4:36) Other Resources: • Layouts
Assignments	• Lab: XML Layouts • Lab: Main Activity • Quiz

INSTRUCTION

Home Screen

The Android home screen, as shown in Figure 1, is very similar to a computer desktop with the application icons functioning like desktop shortcuts. Deleting an app shortcut does not actually delete the app itself. To uninstall an app, a user needs to go to the Application manager in the Settings menu.

In addition to app icons, the home screen may also contain **widgets**. Widgets are small applications that run in a part of the home screen. Widgets allow the user to personalize the device by displaying weather updates or stock quotes, for example. An Android device can have multiple home screens that are accessed by swiping from one to the next.

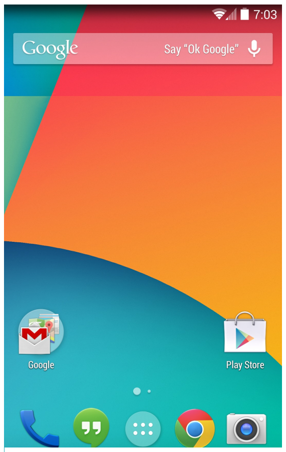


Figure 1: Android Home Screen

Basic Building Block

A Java package named android.view contains the interfaces and the classes used to draw on the screen. A class is a template for creating objects and implementing behaviors (i.e., methods). The foundation for the user interface is a View object. The View object is:

- Created from the View class
- A rectangular-shaped section on the screen
- Responsible for drawing and event handling
- The base class for widgets (widgets are used to create buttons or text fields)

There is a subclass, or second layer, of View called the ViewGroup. A ViewGroup provides a container that holds other Views (or ViewGroups) and defines the layout properties.

A third level exists which is a subclass of the ViewGroup class. The typical layout at this level defines the visual structure for the Android user interface. The XML file is used to declare the layout. This file is located in the res folder of the project.

A view has a location which is designated by left and top coordinates. A view also has two dimensions, designated by a width and a height. The unit for both location and dimension is the pixel. For more detailed information, go to the Android Developer website and read Layouts. This document provides example XML including information on attributes, id, layout parameters, and layout position.

Android Layout Types

Review the following table that lists the Layouts provided by Android and a description of each.

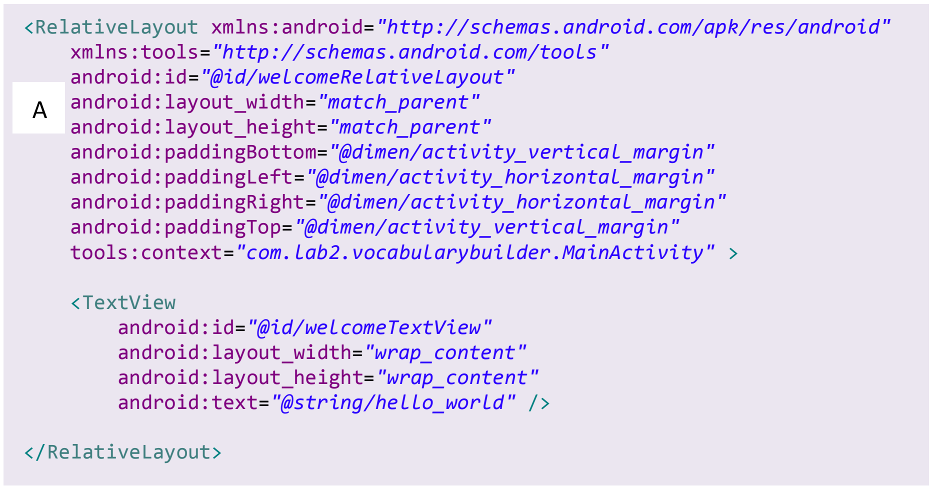
Android Layouts

Layout	Description
Linear Layout	Aligns all child views in one direction--vertically or horizontally

Relative Layout	Displays child views in relative positions
Table Layout	Groups views into rows and columns
Absolute Layout	Specifies the exact location for child views
Frame Layout	Provides a placeholder on the screen used to display a single view
List View	Displays a list of scrollable items
Grid View	Displays items in a two-dimensional grid that scrolls

Each layer uses attributes to define the properties of the layout. The xml file in Figure 2 was generated by Eclipse. Each layout contains one root element which is a View or ViewGroup object. The ViewGroup object, RelativeLayout, is Labeled A in Figure 2. The root element will always have the xmlns:android attribute. The android attributes define the location and dimension for the view.

The layout width and layout height are both using the value “match_parent”. This tells the view to be as big as its parent view group will allow (fill_parent was used before API Level 8). Another common value is “wrap_content”. This tells the view to size itself to the dimensions required by its content.

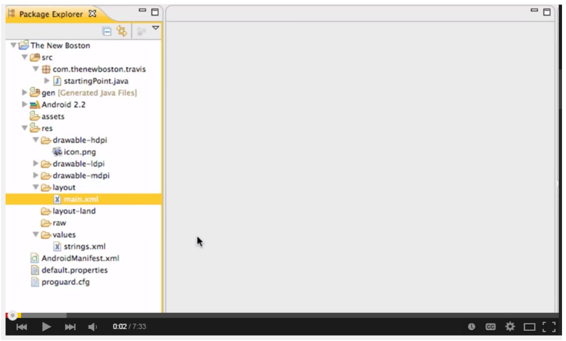


```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@id/welcomeRelativeLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.lab2.vocabularybuilder.MainActivity" >

    <TextView
        android:id="@id/welcomeTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />

</RelativeLayout>
```

Figure 2: XML file showing a Relative Layout

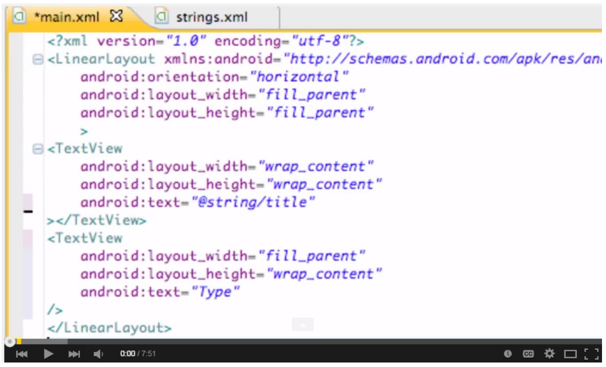
Remember, XML is a simple text-based format for representing structured information such as documents, data, configuration, books, transactions, and invoices to name a few. XML Essentials is the go-to reference to look up any questions or problems that are encountered.

Watch the video, Android Application Development Tutorial – 6 – Introduction to Layouts in XML (7:33), which explains the XML code associated with a Linear Layout in the main.xml file.



Creating a Button

Use the <Button> tag to create a button within the user interface. Eclipse should be open. The following example was adopted from the video, Android Application Development Tutorial – 7 – Creating A Button in XML and Adding an ID (7:51). Watch the video and then try it.



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

Open Eclipse and create a new project, if needed.

1. After the TextView close tag, type **<** (a less than sign). A popup menu appears. Select **Button**.
2. Type **android:** Again a popup menu appears. Select **layout_width** or type it in.
3. Finish out the code:

```
android:layout_width="250sp"
```

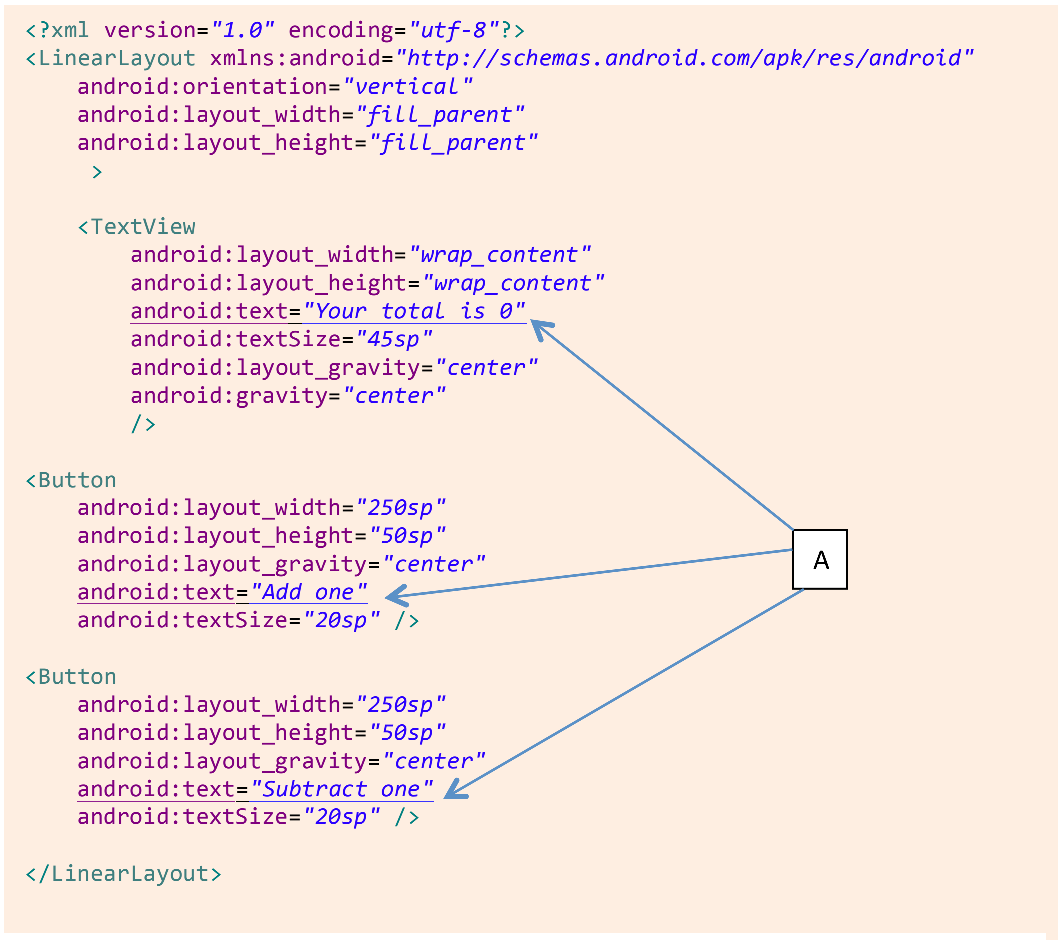
The abbreviation used above, *sp,* stands for scaled-independent pixels. The same unit of measure can be used with text. The following table, lists the other available units of measure.

Measurement Abbreviations used in XML Properties

Unit of Measure	Abbreviation
Inches	in
Millimeters	mm
Pixels	px
Density-independent pixels	dp
Scaled-independent pixels	sp

The benefit of using scaled-independent pixels is that the app will scale the text, as an example, based on how the user has set the phone to display text.

4. Continue to edit the main xml file as follows to add two buttons to the layout:



```
<?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="wrap_content"
        android:layout_height="wrap_content"
        android:text="Your total is 0"
        android:textSize="45sp"
        android:layout_gravity="center"
        android:gravity="center"
        />

<Button
    android:layout_width="250sp"
    android:layout_height="50sp"
    android:layout_gravity="center"
    android:text="Add one"
    android:textSize="20sp" />

<Button
    android:layout_width="250sp"
    android:layout_height="50sp"
    android:layout_gravity="center"
    android:text="Subtract one"
    android:textSize="20sp" />

</LinearLayout>
```

Figure 3: Layout XML file

The code in Figure 3 produces the graphical layout shown in Figure 4:

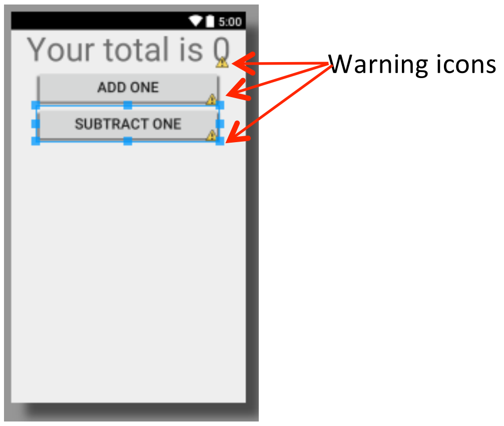


Figure 4: Graphical Layout after adding buttons

Notice the three Warnings icons indicated by the red arrows. The reason for this is that the code is currently using what is referred to as a "hardcoded string" meaning that the text has been typed in, or hardcoded, as indicated by Label A which points to the `android:text` attributes in Figure 3. The button labels, ADD ONE and SUBTRACT ONE describe the buttons' function when pressed. It is not a good practice to hard code strings into the layout file. Instead, add them to a string resource file and reference them from the layout. This practice allows the developer to update every occurrence of the word "Abtn" in all layouts at the same time by just editing the strings.xml file.

1. Edit the strings.xml file which can be found at res/values/strings.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="app_name">Calculations</string>
    <string name="Mtitle">Your total is 0</string>
    <string name="Abtn">Add Button</string>
    <string name="Sbtn">Subtract Button</string>
    <string name="action_settings">Settings</string>

</resources>
```

Figure 5: strings.xml File

2. Change the layout XML file by applying a string to each view as shown in Figure 6.

```
<TextView
        android:layout_width="wrap_content"

android:layout_height="wrap_content"
        android:text="@string/Mtitle"
        android:textSize="45sp"
        android:layout_gravity="center"
        android:gravity="center"
        />

<Button
    android:layout_width="250sp"
    android:layout_height="50sp"
    android:layout_gravity="center"
    android:text="@string/Abtn"
    android:textSize="20sp" />
<Button
    android:layout_width="250sp"
    android:layout_height="50sp"
    android:layout_gravity="center"
    android:text="@string/Sbtn"
    android:textSize="20sp" />
```

Figure 6: Changes made to the layout xml file

Verify the changes by selecting the Graphical Layout tab. All of the warnings are now gone.

The Java Files

The MainActivity.java file can be found within the package inside the src folder. This file contains the MainActivity class that opens the app's user interface. A class describes a group of objects. An object is an instance of a class. When an object is created, it is instantiated. This means that the object is given a name and memory in the system.

Each class requires a copy of an object. A class determines the data that an object holds and the ways the object behaves. It is a standard to start a class name with an uppercase letter.

The entry point of the Activity class is the onCreate() method. A method is a set of Java statements that is included in a Java class and defines what the method does. The code for the MainActivity.java file is shown in Figure 7.

```
package com.susands.calculations;

import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;

public class MainActivity extends ActionBarActivity {

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);    A
   }

   @Override
   public boolean onCreateOptionsMenu(Menu menu) {
      // Inflate the menu; this adds items to the action bar if it is present.
      getMenuInflater().inflate(R.menu.main, menu);
      return true;
   }

   @Override
   public boolean onOptionsItemSelected(MenuItem item) {
      // Handle action bar item clicks here. The action bar will
      // automatically handle clicks on the Home/Up button, so long
      // as you specify a parent activity in AndroidManifest.xml.
      int id = item.getItemId();
      if (id == R.id.action_settings) {
         return true;
      }
      return super.onOptionsItemSelected(item);
   }
}
```

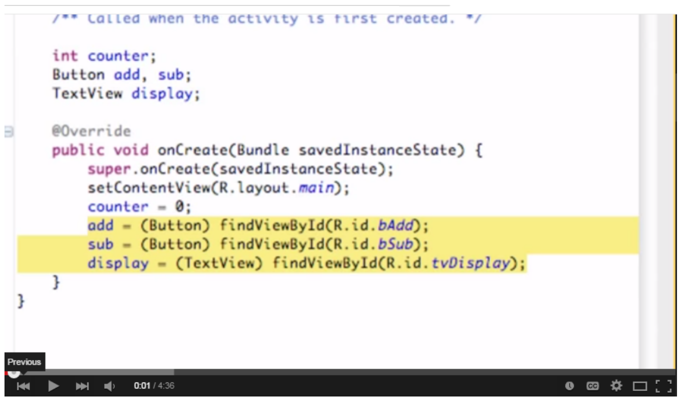
Figure 7: MainActivity.java code

When Eclipse automatically creates the MainActivity.java file, it contains the `onCreate()` method and `setContentView(R.layout.activity_main)` code as Label A shows. This will display the activity_main.xml layout when the application begins. If there is a second screen in the app, another xml file will be used. An `onCreate()` method will place the second Activity on top. The setContentView command then displays the layout from the second xml file.

Buttons

A button is one of the available input controls for an app. Other choices include text fields, checkboxes, radio buttons, toggle buttons, spinners, and pickers. A button is associated with an action that occurs when it is touched by the user. Read the section on Buttons at the Android Developer website to learn how to use the Button class and how a Button responds to a click event.

View the video, Android Application Development Tutorial – 8 – Setting up Variables and Referencing XML ids (6:59) to learn how to set up and add variables to the java file for the two buttons created in the layout. These variables will connect to the XML.



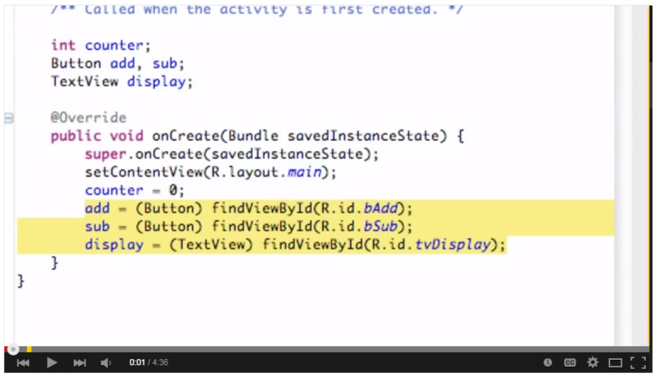
```
    /** Called when the activity is first created. */

    int counter;
    Button add, sub;
    TextView display;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        counter = 0;
        add = (Button) findViewById(R.id.bAdd);
        sub = (Button) findViewById(R.id.bSub);
        display = (TextView) findViewById(R.id.tvDisplay);
    }
}
```

Event Handling

Events collect data when a user interacts with an app by pressing a button or touching a screen. Within the Android framework, there is an event queue. Events are placed in the queue as they occur. Here are three components involved in Android Event Management:

- Event Listeners: The object that receives notice when an event happens
- Event Listeners Registration: The Event Handler gets registered with an Event Listener. The handler is called when the Event Listener fires the event.
- Event Handler: When an event happens, the event listener calls the Event Handlers which is the method that handles the event.

Watch the video, Android Application Development Tutorial -9 – Set up a Button with OnClick Listener (4:36), which sets up a Button with and calls OnClickListener when the button is pressed. The onclick Event Handler is the method that will handle the button press.



```
    /** Called when the activity is first created. */

    int counter;
    Button add, sub;
    TextView display;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        counter = 0;
        add = (Button) findViewById(R.id.bAdd);
        sub = (Button) findViewById(R.id.bSub);
        display = (TextView) findViewById(R.id.tvDisplay);
    }
}
```

The table below lists a few of the Event Listeners and Event Handlers.

Event Listeners and Descriptions

Event Listener	Description	Event Handler Used
onClickListener()	Called when user clicks or touches a button, text, or image.	onClick()

onLongClickListener ()	Called when user clicks or touches a button, text, or image for more than one second.	onLongClick()
onFocusChangeListener ()	Called when user goes away from the view item	onFocusChange()
onKeyListener ()	Called when the user presses or releases a key on the device	onKey()
onTouchListener ()	Called when the user presses or releases a key or uses a gesture on the screen	onTouch()
onMenuItemClickListener ()	Called when the user selects a menu item	onMenuItemClick()

SUMMARY

This lesson discussed how XML code is used to create the layouts of mobile applications programmed in Android. There are two very common layouts used—the Linear Layout and the Relative Layout. These layouts are incorporated differently into an application project. This lesson discussed how to add a button to the app's layout. The onCreate() method and onClickListener() Event Listener in the MainActivity.java file were also discussed.

ASSIGNMENTS

1. Lab: XML Layouts
2. Lab: Main Activity
3. Quiz