GUI Design for Android Applications

SE3A04 – Tutorial

Jason Jaskolka

Department of Computing and Software
Faculty of Engineering
McMaster University
Hamilton, Ontario, Canada
jaskolj@mcmaster.ca

November 28, 2011

Outline

Declarative User Interface

- Declarative approach involves using XML to declare what the UI will look like
- Similar to creating a web page using HTML
- We write tags and specify elements to appear on the screen
- If you ever hand-coded an HTML page, you did pretty much the same work as creating an Android screen

Declarative User Interface

- **Advantages**:
  - Can use *what-you-see-is-what-you-get* (WYSIWYG) tools
  - Some WYSIWYG tools ship with the Eclipse Android Development Tools (ADT) extension, others come from third parties
  - XML is fairly human readable and even people unfamiliar with Android can determine the intent of the user interface
- **Disadvantages**:
  - XML is great for declaring the look and feel of your user interface, but that doesn't provide a good way of handling user input

Programmatic User Interface

- Programmatic approach involves writing Java code to develop user interfaces
- If you have ever worked with any Java AWT or Java Swing development, Android is pretty much the same in that respect
- It is also similar to many other UI toolkits in other languages as well

Programmatic User Interface

- **Advantages**:
  - Everything you can do declaratively, you can also do programmatically
  - Additionally, Java also allows you to specify what happens when that button is actually clicked
- **Disadvantages**:
  - We end up writing quite a few lines of Java to do something quite simple

Which to Use?

- The best practice is to use both
- Use declarative (XML) approach to declare everything about the user interface that is static, such as the layout of the screen, all the widgets, etc.
- Then switch to programmatic (Java) approach to define what goes on when user interacts with various widgets of the user interface
- In other words, use XML to declare what a button looks like, and Java to specify what it does

Views and Layouts

- Android organizes its UI elements into layouts and views
- Everything you see, such as a button, label, or text box, is a view
- Layouts organize views, such as grouping together a button and label or a group of them
- Layouts are similar to Java containers and views are similar to Java components (for those with Java AWT or Java Swing experience)
- Views in Android are sometimes referred to as widgets

Views and Layouts

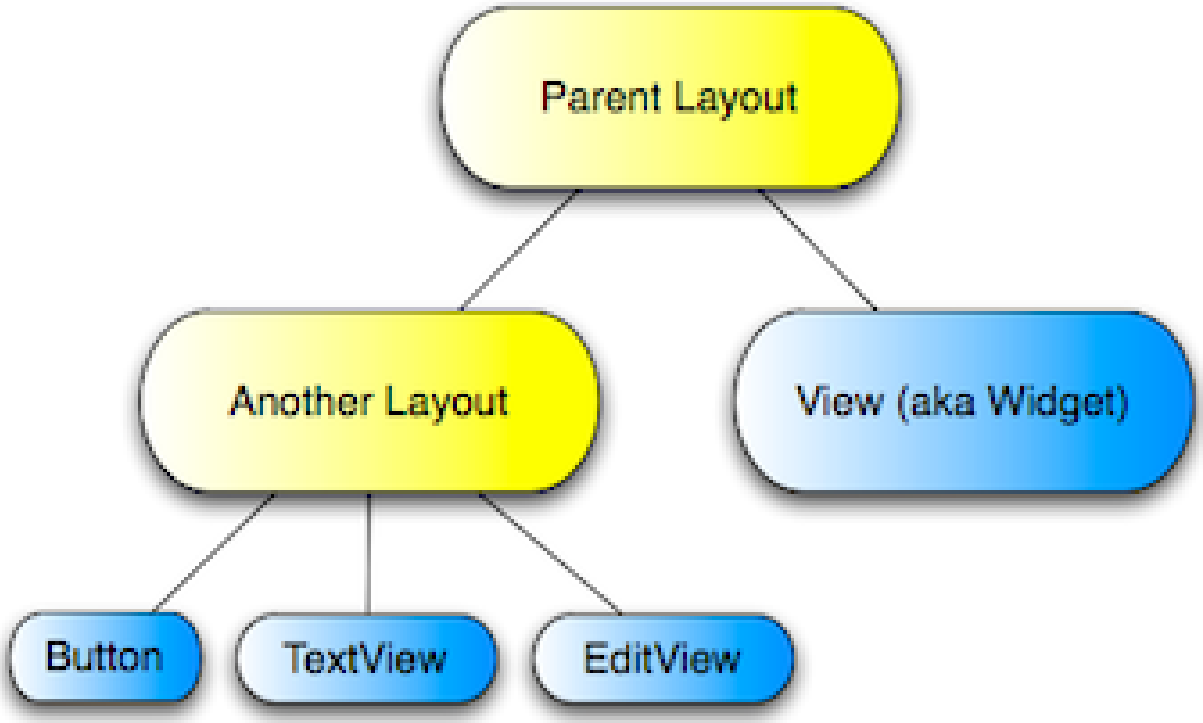


Figure: Relationship between Android layouts and views

Views and Layouts

- A layout can contain other children
- Those children can furthermore be layouts themselves allowing for complex user interface structure
- A layout is responsible for allocating space for each child
- Different layouts use different approaches to laying out their child widgets
- There are couple of main layouts that are used more frequently than others

LinearLayout

- LinearLayout is one of the simplest and most common layouts
- It simply lays out its children next to each other, either horizontally or vertically
- The order of children matters
- Since LinearLayout asks its children for how much space they need, it allocates desired space to each child in the order they are added
- This means that if an "older" child comes along and asks for all the space on the screen, there won't be much left for the subsequent widgets in this layout

TableLayout

- TableLayout lays out its children in a table
- TableLayout consists of only other TableRow widgets
- TableRow represents a row in a table and can contain other UI widgets
- TableRow widgets are laid out next to each other horizontally, sort of like LinearLayout with horizontal orientation

FrameLayout

- FrameLayout places its children on top of each other so that latest child is covering the previous, like a deck of cards
- This layout is useful for tabs, for example
- FrameLayout is also used as placeholder for other widgets to be added to it programmatically at some later point in time

RelativeLayout

- RelativeLayout lays out its children relative to each other
- It is very powerful as it doesn't require you to nest unnecessary layouts in order to achieve certain look
- RelativeLayout can minimize the total number of widgets that need to be drawn thus improving the overall performance of your application
- It requires each of its child views to have an ID set so that we can position it relatively to other children

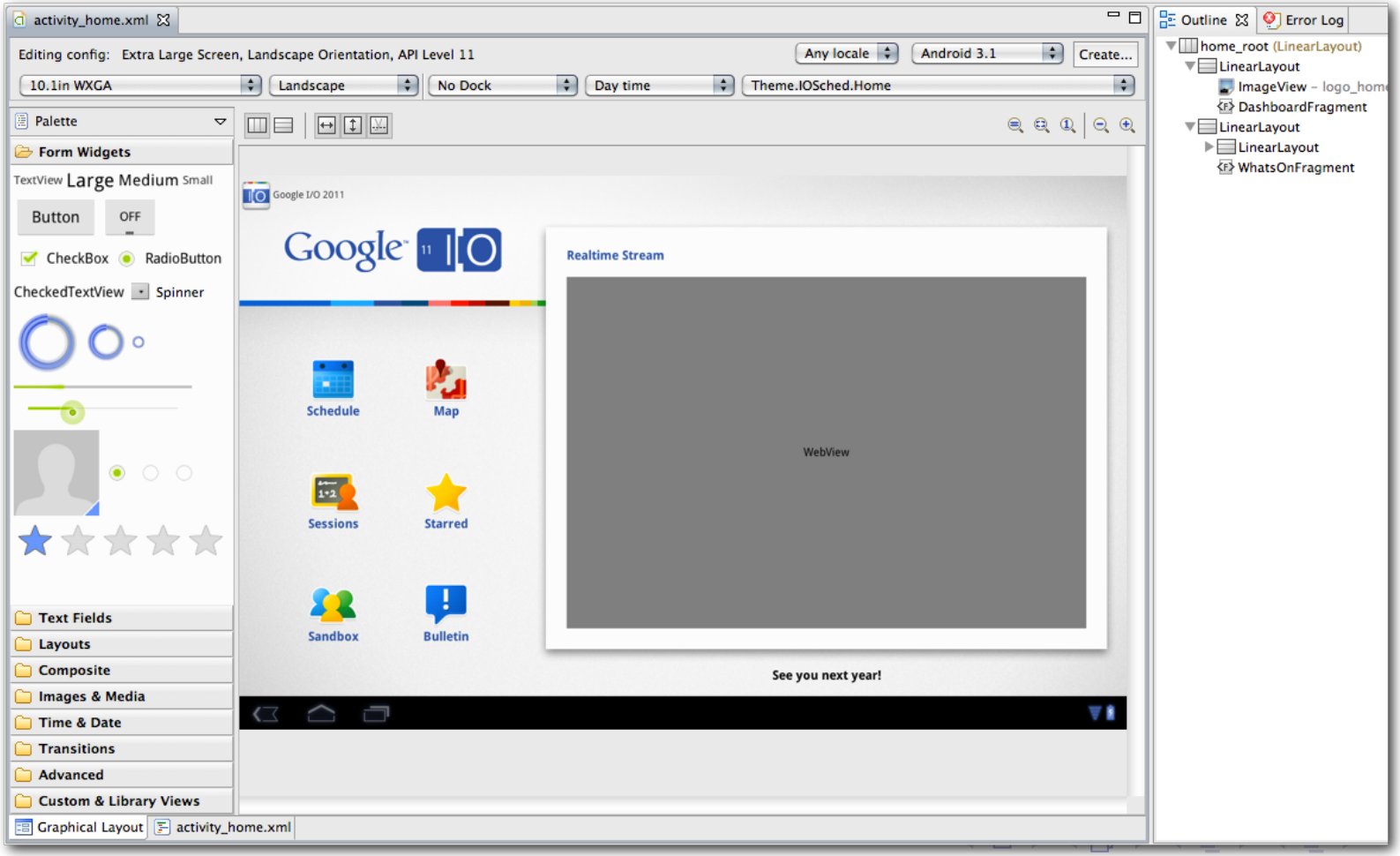
AbsoluteLayout

- AbsoluteLayout positions its children at absolute coordinates on the screen
- It is popular layout for WYSIWYG tools that automatically generate the UI
- While very simple, it is not very flexible
- The user interface can look good on one particular screen but as soon as the screen size, orientation, or density changes, AbsoluteLayout would not be able to adjust

ADT Graphical Layout Editor

- ADT provides many features to allow you to design and build your application's user interface
- Many of these features are in the graphical layout editor, which you can access by opening one of your application's XML layout files in Eclipse
- The graphical layout editor is split up into the following parts:
  - Canvas
  - Outline
  - Palette
  - Configuration Chooser

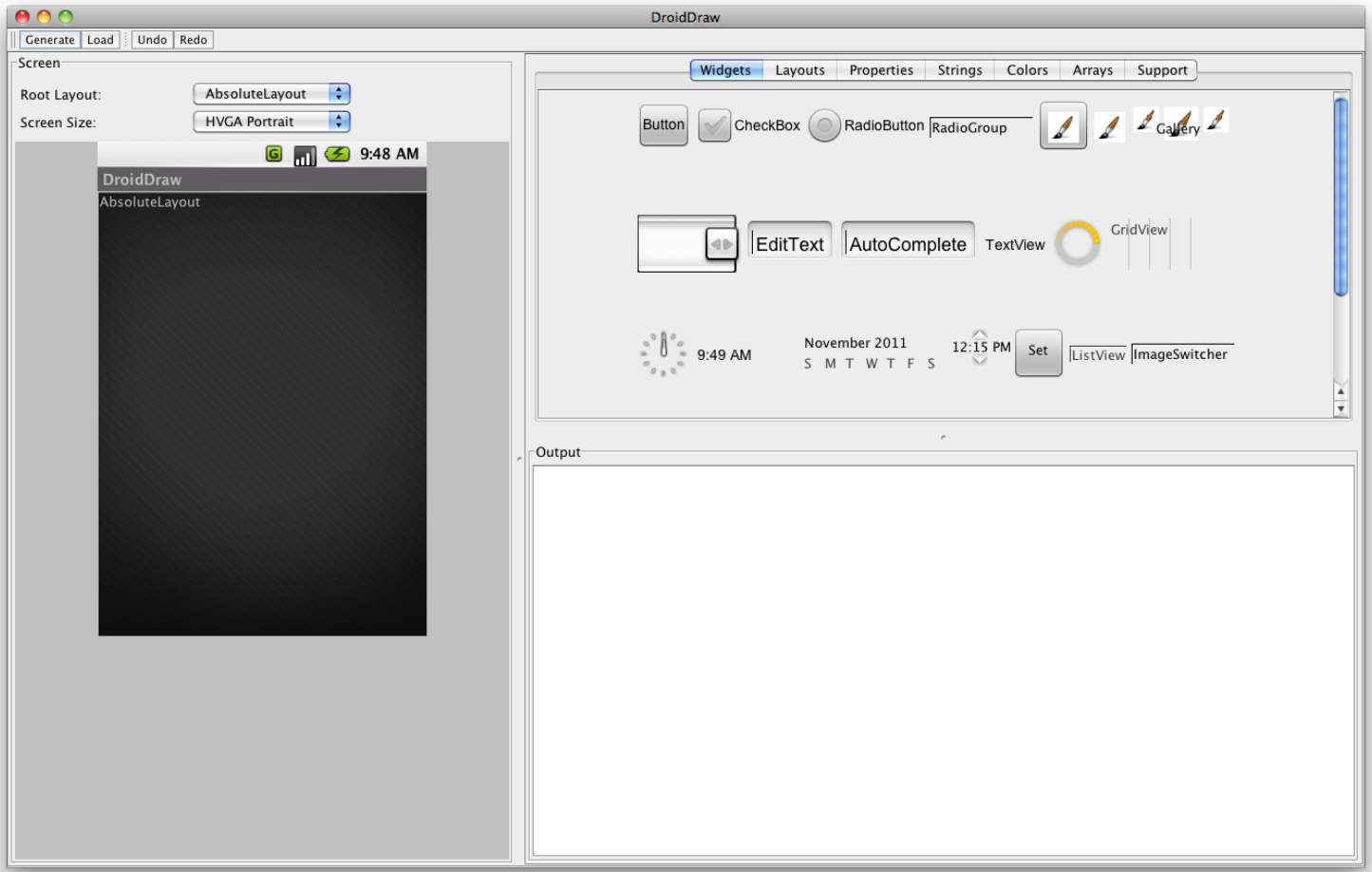
ADT Graphical Layout Editor



DroidDraw

- DroidDraw is a WYSIWYG tool for building user interfaces for the Android platform
- Very similar to the ADT Graphical Layout Editor
- Generates XML which needs to be included with your Android project

DroidDraw


DroidDraw
Generate
Load
Undo
Redo
Screen
Root Layout:
AbsoluteLayout
Screen Size:
HVGA Portrait
9:48 AM
DroidDraw
AbsoluteLayout
Widgets
Layouts
Properties
Strings
Colors
Arrays
Support
Button
CheckBox
RadioButton
RadioGroup
Gallery
EditText
AutoComplete
TextView
GridView
9:49 AM
November 2011
S M T W T F S
12:15 PM
Set
ListView
ImageSwitcher
Output

Some Implementation Details

- The XML file for your interface should reside in the `res/layout` directory of your Android project
- Mixing the declarative and programmatic approaches is nothing more than referencing your declared views and layouts and programming the business logic for each
- GUI elements can be looked up with:

```
this.findViewById(R.id.<id>)
```

Code Fragments for a Button

Example

Suppose you have a button named with name "Start" and with ID is `@+id/start` in the file `main.xml` in `res/layout`.

- You will import the following packages:

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

Code Fragments for a Button

Example (continued)

- You will have a class that *extends* `Activity` and *implements* `OnClickListener`:

```
public class <Name> extends Activity
    implements OnClickListener {...}
```

- You will have a class variable:

```
Button start;
```

Code Fragments for a Button

Example (continued)

- You will have an `onCreate(...)` method for the Activity:

```
@Override
public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
    setContentView(R.layout.main);
        ...
    start = (Button)this.findViewById(R.id.start);
    start.setOnClickListener(this);
}
```

Code Fragments for a Button

Example (continued)

- You will have an `onClick(...)` method which will handle what the application needs to do when the button is clicked (event-handler):

```
public void onClick(View v) {
    // Do something;
}
```

Summary

- There are a number of ways to approach the user interface design
- We have mentioned just a few of the wide variety of options
- There are numerous online tutorials, screen-casts, and code samples demonstrating the use of the tools that we have talked about

References


Marko Gargenta
Learning Android: Chapter 6: Android User Interface
O'Reilly OFPS, 2010.
http://ofps.oreilly.com/titles/9781449390501/Android_User_Interface.html


Android Developers
Android Developer Tools
http://developer.android.com/guide/developing/tools/adt.html


Brendan Burns
DroidDraw
http://www.droiddraw.org/

Questions

- Questions?