Android UI Layout

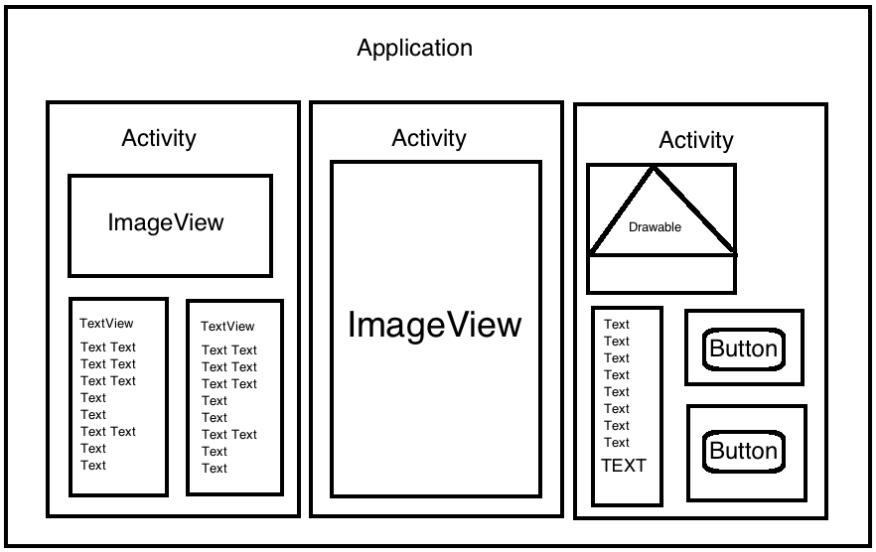
Android Layout is used to define the user interface that holds the UI controls or widgets that will appear on the screen of an android application or activity screen. The core components of every Android user interface: **layouts and views.**

Before you can add a view to your app, you need a layout, and a layout without any views isn't likely to win your app any fans.

What is a view?

As you are already aware, Android apps are made up of activities. Typically, one activity is displayed at a time and this Activity occupies the entire screen.

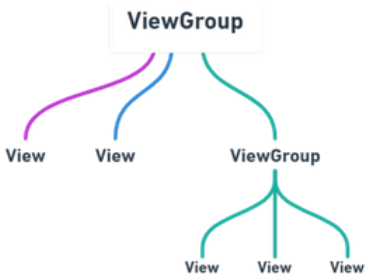
Each activity is made up of views, which are the most basic component of UI. Views always occupy a rectangular area, although a view can display content of any shape:



View is the base class for widgets, which are used to create interactive UI components like buttons, text fields, etc. Some examples of the most commonly used views are;

TextView, EditText, ImageView, Button, Image Button, Date Picker, RadioButton and CheckBox buttons.

A ViewGroup is a container that groups other child views & ViewGroup objects together. A View Group is a subclass of the View class. It is a container that holds views such as ImageView, TextView, etc.

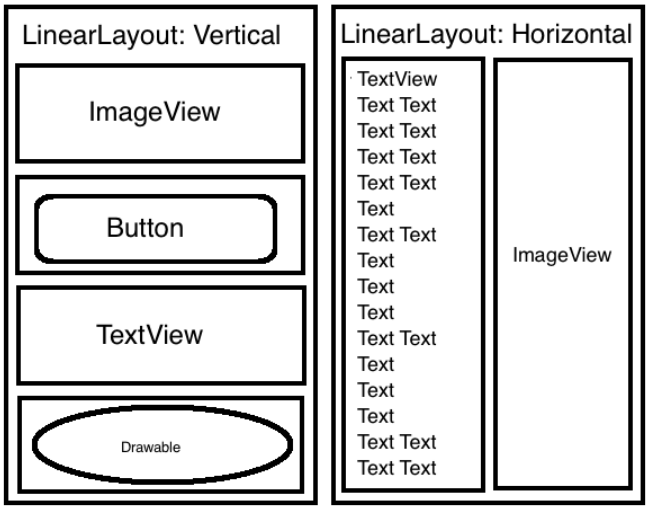


What is a layout?

Layouts are View Groups that are defined either by an xml file, or programmatically. Layout is xml file (or programmatically created) that includes multiple views to create UI.

layout is an invisible container that is responsible for positioning the child elements on the screen. A layout defines the structure for a user interface in your app, such as in an activity. All elements in the layout are built using a hierarchy of View and ViewGroup objects. A View usually draws something the user can see and interact with.

For example, LinearLayout is a ViewGroup (also sometimes known as layout manager) that arranges its child elements (views or ViewGroups) into vertical or horizontal rows:



Building your UI – XML or Java?

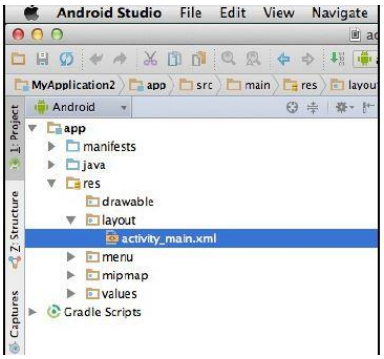
The easiest way of defining your user interface (and the views, ViewGroups, and layout elements that it contains) is via your **project's XML file**.

XML stands for **eXtensible Markup Language**. It is markup language much like HTML. XML's primary function is to create formats for data that is used to encode information for documentation, database records, transactions & many other types of data.

Advantages of XML include the following: XML uses human, not computer, language. XML is readable and understandable, even by novices, and no more difficult to code than HTML. XML is completely compatible with Java™ and 100% portable.

Declaring your UI with XML

Android provides a straightforward XML vocabulary that gives your user interface a **human-readable structure, and creates a separation between the code that defines your UI and the code that controls your app's behavior**. You define your layouts in XML in a dedicated layout resource file. This helps to keep both sets of code cleaner, and it gives you the ability to tweak and refine your UI without having to touch your app's underlying code. For example, you can update your layout to support an additional language without having to touch the previously-tested code.



Declaring your UI in XML also **makes it easier to provide alternate layouts**.' You'll find this layout resource file in your project's "res/layout" folder (Android project in Eclipse or Android Studio):

Android Layout Types

There are number of layouts provided by Android which are used in almost all Android applications to provide different view, look and feel. Few common layout types are,

1) **Linear Layout**

   Linear Layout is a view group that aligns all children in a single direction, vertically or horizontally.

2) **Relative Layout**

   RelativeLayout is a view group that displays child views in relative (specific) positions (Child A to the left of Child B).

3) **Table Layout**

   TableLayout is a view that groups views into rows and columns.

4) **Absolute Layout**

   AbsoluteLayout enables you to specify the exact location of its children.

5) **Frame Layout**

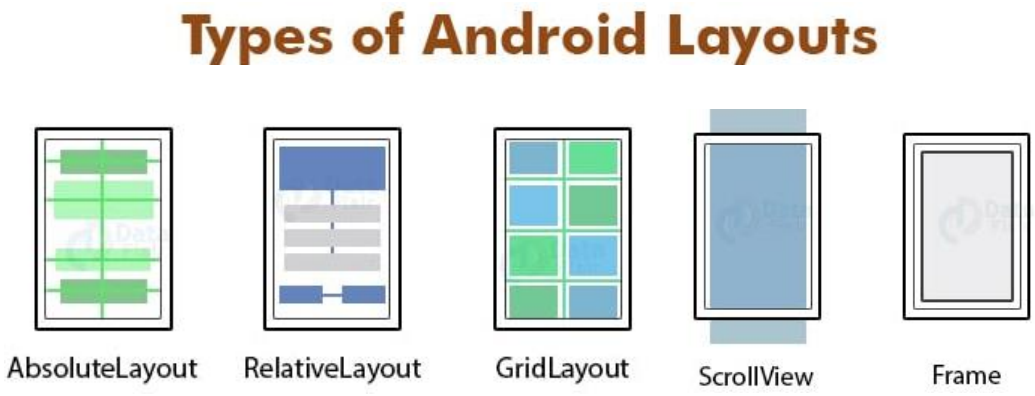
   This Layout is a placeholder on screen that you can use to display a single view.

6) **List View**

   ListView is a view group that displays a list of scrollable (single column list) items.

7) **Grid View**

   GridView is a ViewGroup that displays items in a two-dimensional, scrollable grid of columns and row.



Layout Attributes

Each layout has a set of attributes which define the visual properties of that layout. There are few common attributes among all the layouts and their are other attributes which are specific to that layout. Following are common attributes and will be applied to all the layouts:

1) **android:id:** This is the ID which uniquely identifies the view.
2) **android:layout_width:** This is the width of the layout.
3) **android:layout_height:** This is the height of the layout
4) **android:layout_marginTop:** This is the extra space on the top side of the layout.
5) **android:layout_marginBottom:**This is the extra space on the bottom side of the layout.

6) **android:layout_marginLeft:** This is the extra space on the left side of the layout.
7) **android:layout_marginRight:** This is the extra space on the right side of the layout.
8) **android:layout_gravity:** This specifies how child Views are positioned.
9) **android:layout_weight:** This specifies how much of the extra space in the layout should be allocated to the View.
10) **android:layout_x:** This specifies the x-coordinate of the layout.
11) **android:layout_y:** This specifies the y-coordinate of the layout.
12) **android:layout_width:** This is the width of the layout.
13) **android:paddingLeft:** This is the left padding filled for the layout.
14) **android:paddingRight:** This is the right padding filled for the layout.
15) **android:paddingTop:** This is the top padding filled for the layout.
16) **android:paddingBottom:** This is the bottom padding filled for the layout.