

# Layout

By OP Sahu

# Layout

- An Android layout is a class that handles arranging the way its children appear on the screen. Anything that is a View (or inherits from View) can be a child of a layout. All of the layouts inherit from ViewGroup (which inherits from View) so you can nest layouts. Every View and ViewGroup object supports their own variety of XML attributes.

# Basic XML Attributes in Android

- **ID:** Any View object may have an **integer ID** associated with it(see R.java), to uniquely identify the View within the tree. When the application is compiled, this ID is referenced as an integer, but the ID is typically **assigned in the layout XML file as a string**, in the id attribute. Syntax:  
`android:id="@+id/my_button"`

**@ indicates** that the XML parser should parse and expand the rest of the ID string and identify it as an ID resource.

**+ symbols** means that this is a new resource name that must be created and added to our resources (in the R.java file).

There are a number of other ID resources that are offered by the Android framework. When referencing an Android resource ID, you do not need the plus-symbol, but must add the android package namespace, like so:  
`android:id="@android:id/my_button"`

<u>Attribute</u>	<u>Description</u>
layout_width	Specifies the width of the View or ViewGroup
layout_height	Specifies the height of the View or ViewGroup
layout_marginTop	Specifies extra space on the top side of the View or ViewGroup.
layout_marginBottom	Specifies extra space on the bottom side of the View or ViewGroup.
layout_marginLeft	Specifies extra space on the left side of the View or ViewGroup.
layout_marginRight	Specifies extra space on the right side of the View or ViewGroup.
layout_x	Specifies the x-coordinate of the View or ViewGroup.
layout_y	Specifies the y-coordinate of the View or ViewGroup

- All view groups include a width and height (`layout_width` and `layout_height`), and each view is required to define them. Others are optional.
- You can specify width and height with exact measurements, though you probably won't want to do this often. More often, you will use one of these constants to set the width or height:
- ***wrap\_content*** tells your view to size itself to the dimensions required by its content
- ***fill\_parent*** constant is **deprecated** (renamed ***match\_parent*** in **API Level 8**) tells your view to become as big as its parent view group will allow.
- In general, specifying a layout width and height using absolute units such as pixels is not recommended. Instead, using relative measurements such as density-independent pixel units (*dp*), *wrap\_content*, or *match\_parent*, is a better approach, because it helps ensure that your application will display properly across a variety of device screen sizes.

# Types Of Layout

- Linear Layout
- Relative Layout
- Absolute Layout
- Frame Layout
- Table Layout

# Linear Layout

- `LinearLayout` organizes elements along a single line. You specify whether that line is vertical or horizontal using `android:orientation`. It's the default Layout in an Android Application, unlike in java where each **JPanel** object is initialized to use a **FlowLayout**, unless you specify differently when creating the JPanel. **Content panes use BorderLayout by default.**

# A sample Layout XML using LinearLayout.

- **<LinearLayout** xmlns:android="http://schemas.android.com/apk/res/android" android:orientation="horizontal" android:layout\_width="fill\_parent" android:layout\_height="fill\_parent">
- **<Button** android:id="@+id/backbutton" android:text="Back" android:layout\_width="wrap\_content" android:layout\_height="wrap\_content" />
- **<TextView** android:text="First Name" android:layout\_width="wrap\_content" android:layout\_height="wrap\_content" />
- **<EditText** android:width="100px" android:layout\_width="wrap\_content" android:layout\_height="wrap\_content" />
- **<TextView** android:text="Last Name" android:layout\_width="wrap\_content" android:layout\_height="wrap\_content" />
- **<EditText** android:width="100px" android:layout\_width="wrap\_content" android:layout\_height="wrap\_content" />
- **</LinearLayout>**



LinearLayout Demo

Back

First Name

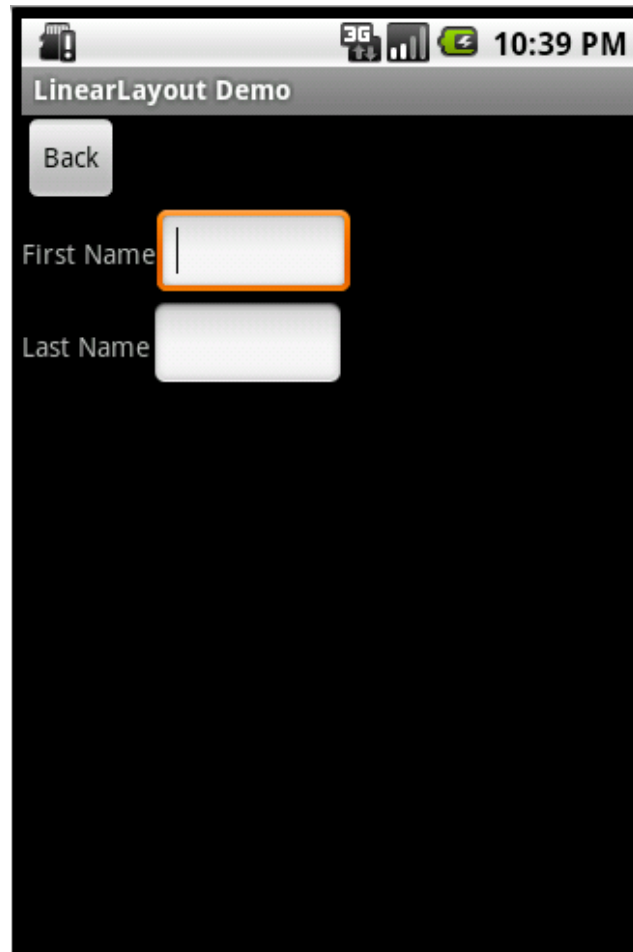
Last Name

- **<LinearLayout**

```
xmlns:android="http://schemas.android.com/apk/res/android"  
android:orientation="vertical" android:layout_width="fill_parent"  
android:layout_height="fill_parent">
```

- **<Button** android:id="@+id/backbutton" android:text="Back"  
android:layout\_width="wrap\_content"  
android:layout\_height="wrap\_content" /> **<LinearLayout**  
android:orientation="horizontal" android:layout\_width="fill\_parent"  
android:layout\_height="wrap\_content"> **<TextView** android:text="First  
Name" android:layout\_width="wrap\_content"  
android:layout\_height="wrap\_content" /> **<EditText**  
android:width="100px" android:layout\_width="wrap\_content"  
android:layout\_height="wrap\_content" /> **</LinearLayout>** **<LinearLayout**  
android:orientation="horizontal" android:layout\_width="fill\_parent"  
android:layout\_height="wrap\_content"> **<TextView** android:text="Last  
Name" android:layout\_width="wrap\_content"  
android:layout\_height="wrap\_content" /> **<EditText**  
android:width="100px" android:layout\_width="wrap\_content"  
android:layout\_height="wrap\_content" /> **</LinearLayout>**  
**</LinearLayout>**

O/P when  
android:orientation="vertical"



# Relative Layout

- RelativeLayout lays out elements based on their relationships with one another, and with the parent container. This is arguably the most complicated layout, and we need several properties to actually get the layout we want.

These properties will layout elements relative to the  
parent container.

- `android:layout_alignParentBottom` – Places the bottom of the element on the bottom of the container
- `android:layout_alignParentLeft` – Places the left of the element on the left side of the container
- `android:layout_alignParentRight` – Places the right of the element on the right side of the container
- `android:layout_alignParentTop` – Places the element at the top of the container
- `android:layout_centerHorizontal` – Centers the element horizontally within its parent container
- `android:layout_centerInParent` – Centers the element both horizontally and vertically within its container
- `android:layout_centerVertical` – Centers the element vertically within its parent container

# Relative To Other Elements

- These properties allow you to layout elements relative to other elements on screen. The value for each of these elements is the id of the element you are using to layout the new element. Each element that is used in this way must have an ID defined using `android:id="@+id/XXXXX"` where XXXXX is replaced with the desired id. You use `"@id/XXXXX"` to reference an element by its id. One thing to remember is that **referencing an element before it has been declared will produce an error.**

- `android:layout_above` – Places the element above the specified element
- `android:layout_below` – Places the element below the specified element
- `android:layout_toLeftOf` – Places the element to the left of the specified element
- `android:layout_toRightOf` – Places the element to the right of the specified element

# Alignment With Other Elements

- `android:layout_alignBaseline` – Aligns baseline of the new element with the baseline of the specified element
- `android:layout_alignBottom` – Aligns the bottom of new element in with the bottom of the specified element
- `android:layout_alignLeft` – Aligns left edge of the new element with the left edge of the specified element
- `android:layout_alignRight` – Aligns right edge of the new element with the right edge of the specified element
- `android:layout_alignTop` – Places top of the new element in alignment with the top of the specified element

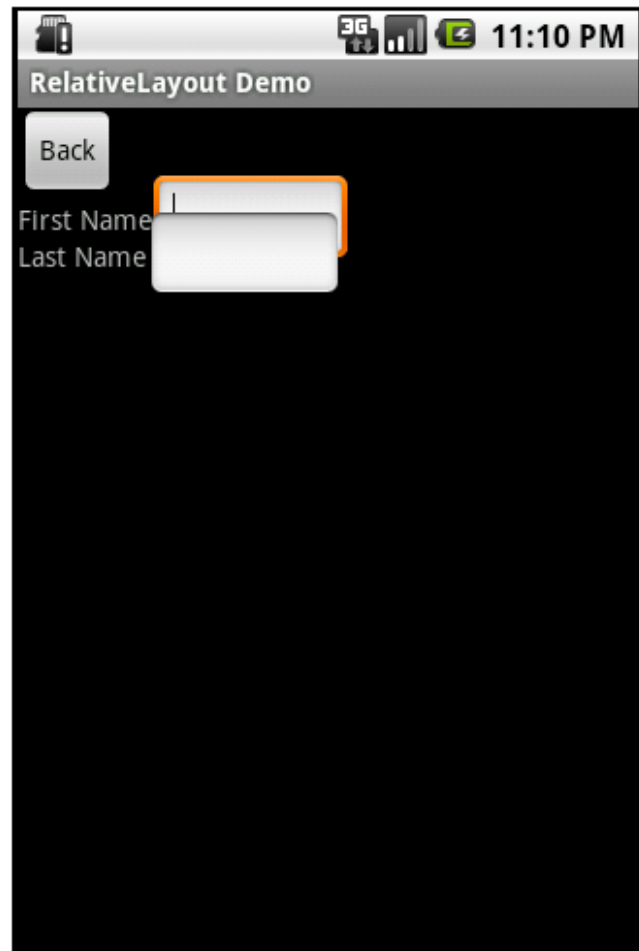


```
<RelativeLayout android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android">
    <Button android:id="@+id/backbutton" android:text="Back"
        android:layout_width="wrap_content" android:layout_height="wrap_content" />

    <TextView android:id="@+id/firstName" android:text="First Name"
        android:layout_width="wrap_content" android:layout_height="wrap_content"
        android:layout_below="@id/backbutton" />
    <EditText android:width="100px" android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toRightOf="@id/firstName"
        android:layout_alignBaseline="@id/firstName" />

    <TextView android:id="@+id/lastName" android:text="Last Name"
        android:layout_width="wrap_content" android:layout_height="wrap_content"
        android:layout_below="@id/firstName" />
    <EditText android:width="100px" android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toRightOf="@id/lastName"
        android:layout_alignBaseline="@id/lastName" />
    • </RelativeLayout>
```

# Ouch!!



```
<RelativeLayout android:layout_width="fill_parent" android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android">
```

```
<Button android:id="@+id/backbutton" android:text="Back"
    android:layout_width="wrap_content" android:layout_height="wrap_content" />
```





```
<TextView android:id="@+id/firstName" android:text="First Name"
    android:layout_width="wrap_content" android:layout_height="wrap_content"
    android:layout_below="@id/backbutton" />
```

```
<EditText android:id="@+id/editFirstName" android:width="100px"
    android:layout_width="wrap_content" android:layout_height="wrap_content"
    android:layout_toRightOf="@id/firstName" android:layout_below="@id/backbutton"/>
```

```
<EditText android:id="@+id/editLastName" android:width="100px"
    android:layout_width="wrap_content" android:layout_height="wrap_content"
    android:layout_below="@id/editFirstName"
    android:layout_alignLeft="@id/editFirstName"/>
```

```
<TextView android:id="@+id/lastName" android:text="Last Name"
    android:layout_width="wrap_content" android:layout_height="wrap_content"
    android:layout_toLeftOf="@id/editLastName" android:layout_below="@id/editFirstName"
    />
```

```
</RelativeLayout>
```

9:00 PM

RelativeLayout Demo

Back

First Name

Last Name

# Table Layout

- TableLayout organizes content into rows and columns. The rows are defined in the layout XML, and the columns are determined automatically by Android. This is done by creating at least one column for each element. So, for example, if you had a row with two elements and a row with five elements then you would have a layout with two rows and five columns.
- You can specify that an element should occupy more than one column using `android:layout_span`. This can increase the total column count as well, so if we have a row with two elements and each element has `android:layout_span="3"` then you will have at least six columns in your table.
- By default, Android places each element in the first unused column in the row. You can, however, specify the column an element should occupy using `android:layout_column`.

```
<TableLayout android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android">
    <TableRow> <Button android:id="@+id/backbutton" android:text="Back"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" /> </TableRow>
```

```
<TableRow> <TextView android:text="First Name"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" android:layout_column="1" />
    <EditText android:width="100px" android:layout_width="wrap_content"
    android:layout_height="wrap_content" /> </TableRow>
```

```
<TableRow> <TextView android:text="Last Name"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" android:layout_column="1" />
    <EditText android:width="100px" android:layout_width="wrap_content"
    android:layout_height="wrap_content" /> </TableRow>
```

```
</TableLayout>
```

3G

12:03 AM

TableLayout Demo

Back

First Name

Last Name





# Absolute Layout

- Absolute Layout is based on the simple idea of placing each control at an absolute position. You specify the exact x and y coordinates on the screen for each control. This is not recommended for most UI development (in fact Absolute Layout is currently deprecated) since absolutely positioning every element on the screen makes an inflexible UI that is much more difficult to maintain. Consider what happens if a control needs to be added to the UI. You would have to change the position of every single element that is shifted by the new control.



- **<AbsoluteLayout**

```
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="fill_parent"
android:layout_height="fill_parent"> <Button
android:id="@+id/backbutton" android:text="Back"
android:layout_x="10px" android:layout_y="5px"
android:layout_width="wrap_content"
android:layout_height="wrap_content" /> <TextView
android:layout_x="10px" android:layout_y="110px"
android:text="First Name" android:layout_width="wrap_content"
android:layout_height="wrap_content" /> <EditText
android:layout_x="150px" android:layout_y="100px"
android:width="100px" android:layout_width="wrap_content"
android:layout_height="wrap_content" /> <TextView
android:layout_x="10px" android:layout_y="160px"
android:text="Last Name" android:layout_width="wrap_content"
android:layout_height="wrap_content" /> <EditText
android:layout_x="150px" android:layout_y="150px"
android:width="100px" android:layout_width="wrap_content"
android:layout_height="wrap_content" /> </AbsoluteLayout>
```

10:11 PM

AbsoluteLayout Demo

Back

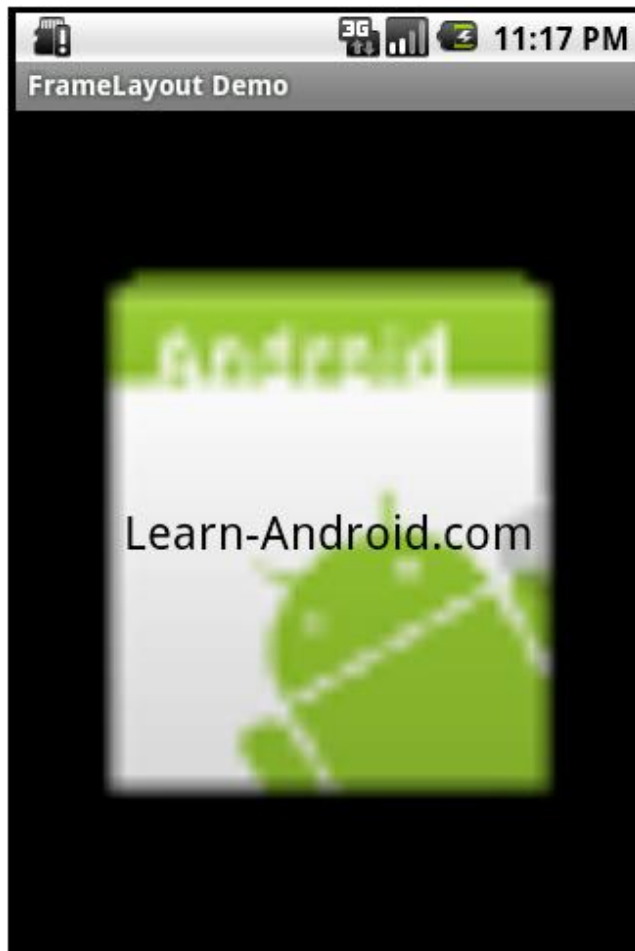
First Name

Last Name

# Frame Layout

- `FrameLayout` is designed to display a single item at a time. You can have multiple elements within a `FrameLayout` but each element will be positioned based on the top left of the screen. Elements that overlap will be displayed overlapping. I have created a simple XML layout using `FrameLayout` that shows how this works.

- **<FrameLayout** android:layout\_width="fill\_parent"  
android:layout\_height="fill\_parent"  
xmlns:android="http://schemas.android.com/apk/res/  
android"> **<ImageView** android:src="@drawable/icon"  
android:scaleType="fitCenter"  
android:layout\_height="fill\_parent"  
android:layout\_width="fill\_parent"/> **<TextView**  
android:text="Learn-Android.com"  
android:textSize="24sp" android:textColor="#000000"  
android:layout\_height="fill\_parent"  
android:layout\_width="fill\_parent"  
android:gravity="center"/> **</FrameLayout>**



- You can see I had both the ImageView and TextView fill the parent in both horizontal and vertical layout. Gravity specifies where the text appears within its container, so I set that to center. If I had not set a gravity then the text would have appeared at the top left of the screen.
- FrameLayout can become more useful when elements are hidden and displayed programmatically. You can use the attribute android:visibility in the XML to hide specific elements. You can call setVisibility from the code to accomplish the same thing. The three available visibility values are visible, invisible (does not display, but still takes up space in the layout), and gone (does not display, and does not take space in the layout).

**Thank You**