5

Android Basic XML Layouts

Victor Matos
Cleveland State University

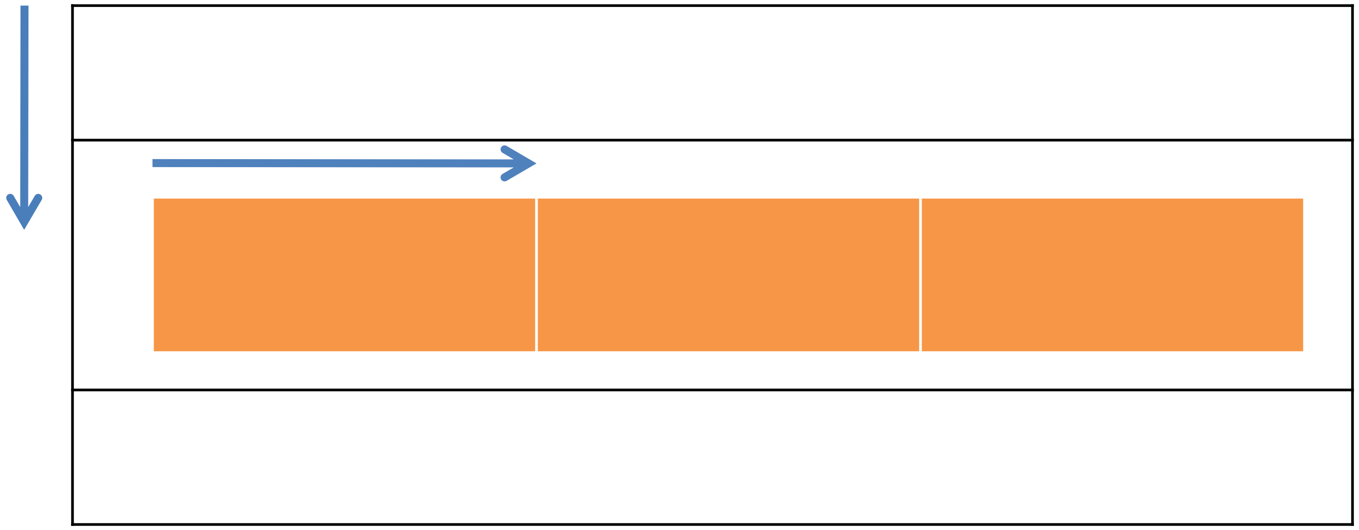
Notes are based on:
The Busy Coder's Guide to Android Development
by Mark L. Murphy
Copyright © 2008-2009 CommonsWare, LLC.
ISBN: 978-0-9816780-0-9
&
Android Developers
http://developer.android.com/index.html

Basic XML Layouts - Containers

Designing Complex Uis

- Arguably, **LinearLayout** is the most common modeling tool. It offers a "box" model similar to the Java-Swing *Box-Layout*.

- Generally, complex UI designs result from the combination of simpler *nested* boxes that show their inner pieces using a *horizontal* or *vertical* orientation.



Basic XML Layouts - Containers

Summary of Commonly-used Android containers

1. **LinearLayout** (the box model),
2. **RelativeLayout** (a rule-based model), and
3. **TableLayout** (the grid model), along with
4. **ScrollView**, a container designed to assist with implementing scrolling containers.
5. **Other** (ListView, GridView, WebView, MapView,…) discussed later

Basic XML Layouts - Containers

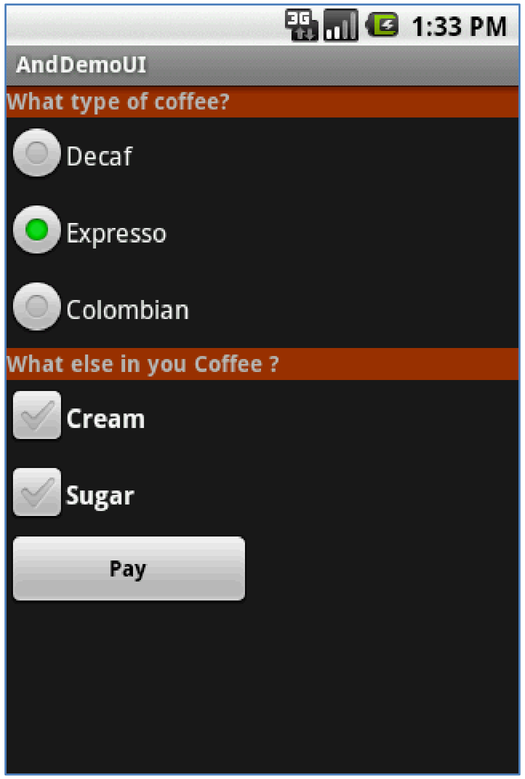
Before we get started …

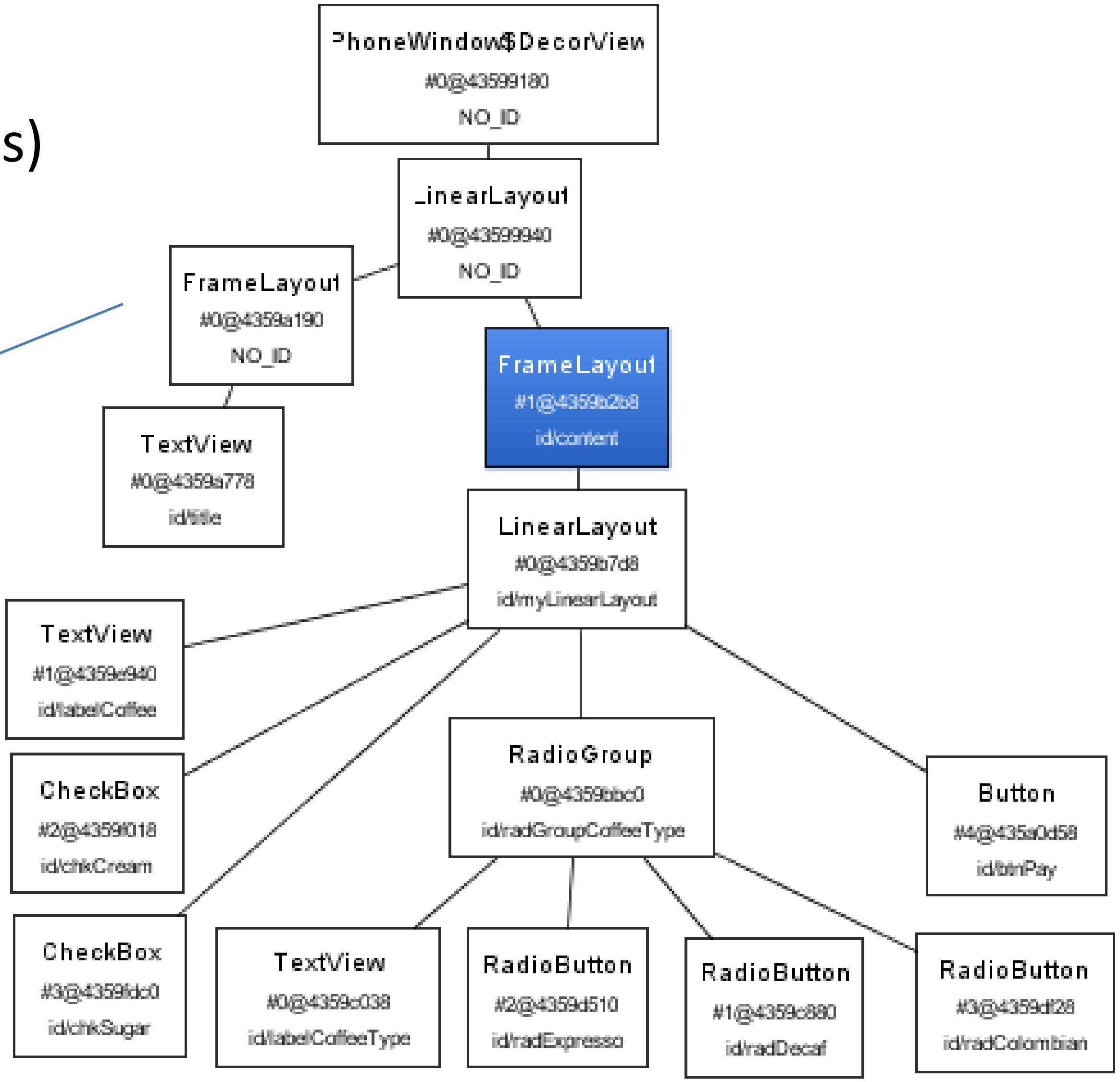
1. Android's simplest layout manager is called: **Frame Layout**.
2. A Frame Layout is a rectangular container that pins *each child* to its upper left corner.
3. Adding multiple views to a frame layout just stacks one on top of the other (overlapping the views)

Basic XML Layouts - Containers

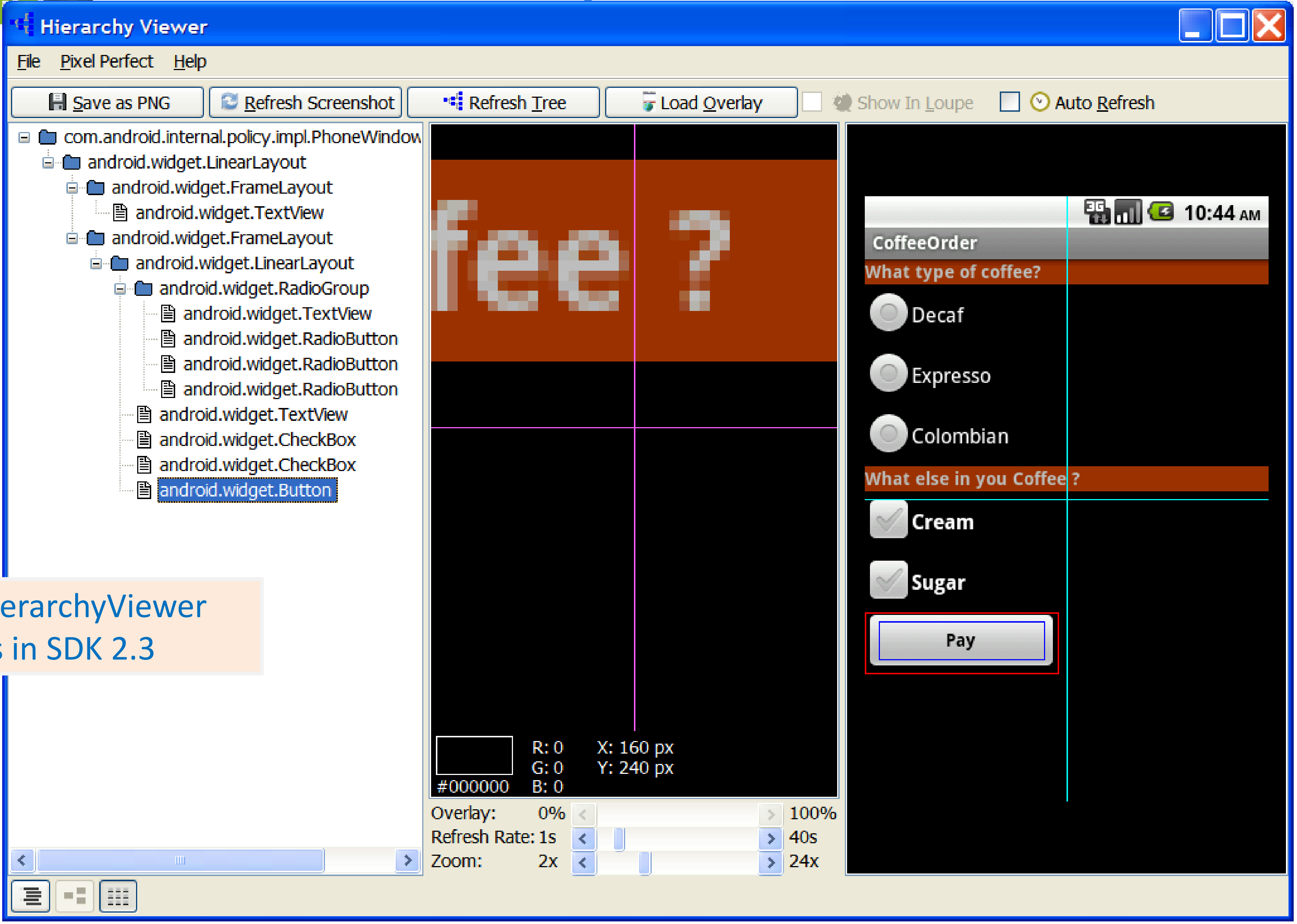
Before we get started …

Hierarchy Viewer (\tools)





Basic XML Layouts - Containers


Hierarchy Viewer

File Pixel Perfect Help

Save as PNG | Refresh Screenshot | Refresh Tree | Load Overlay | Show In Loupe | Auto Refresh

```
com.android.internal.policy.impl.PhoneWindow
  android.widget.LinearLayout
    android.widget.FrameLayout
      android.widget.TextView
    android.widget.FrameLayout
      android.widget.LinearLayout
        android.widget.RadioGroup
          android.widget.TextView
          android.widget.RadioButton
          android.widget.RadioButton
          android.widget.RadioButton
        android.widget.TextView
        android.widget.CheckBox
        android.widget.CheckBox
        android.widget.Button
```

R: 0 X: 160 px
G: 0 Y: 240 px
#000000 B: 0

Overlay: 0% 100%
Refresh Rate: 1s 40s
Zoom: 2x 24x

HierarchyViewer
As in SDK 2.3

Basic XML Layouts - Containers

1. Linear Layout

LinearLayout is a *box model* – widgets or child containers are lined up in a *column* or *row*, one after the next.

To configure a LinearLayout, you have five main areas of control besides the container's contents:

- orientation,
- fill model,
- weight,
- gravity,
- padding ,
- margin

Basic XML Layouts - Containers

1. Linear Layout

Orientation
indicates whether the LinearLayout represents a *row* or a *column*.

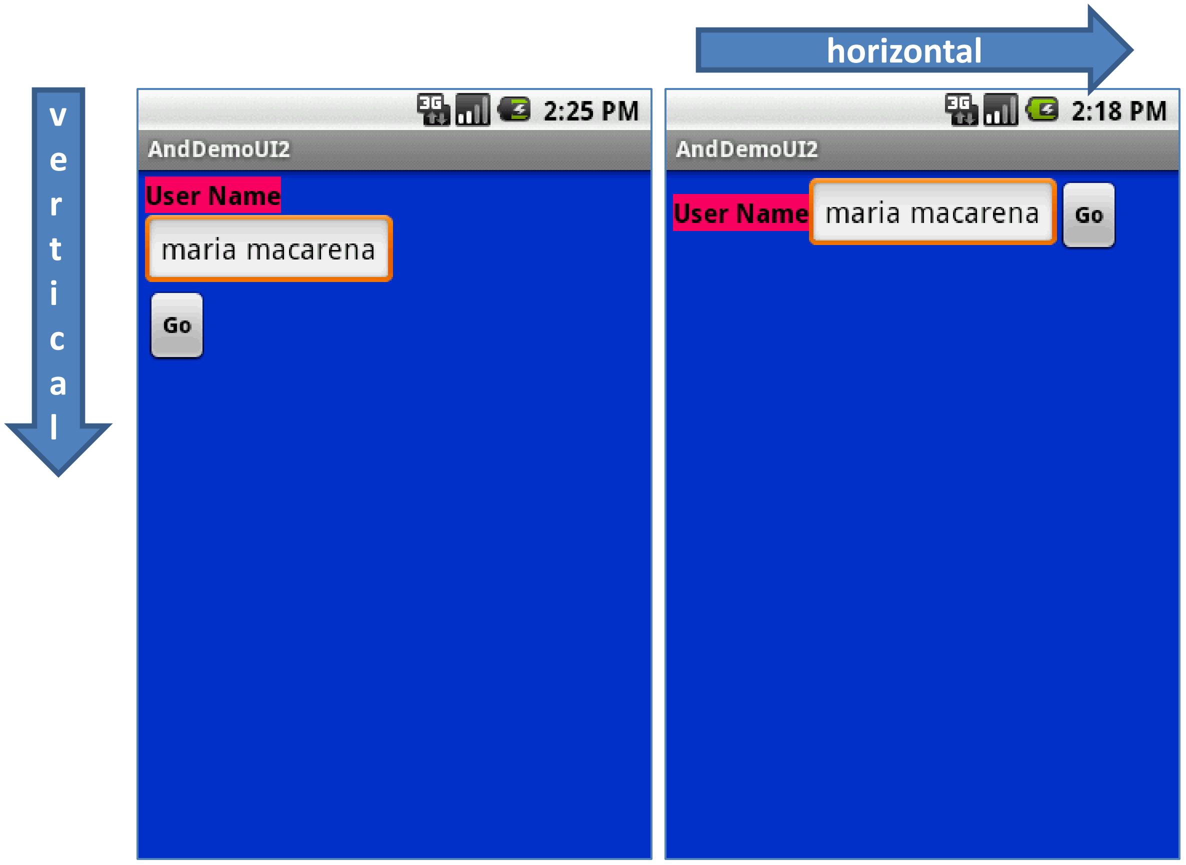
Add the android:orientation property to your LinearLayout element in your XML layout, setting the value to be **horizontal** for a row or **vertical** for a column.

The orientation can be modified at runtime by invoking *setOrientation()*

Basic XML Layouts - Containers

1.1 Linear Layout: Orientation

indicates whether the LinearLayout represents a *row* (HORIZONTAL) or a *column* (VERTICAL).

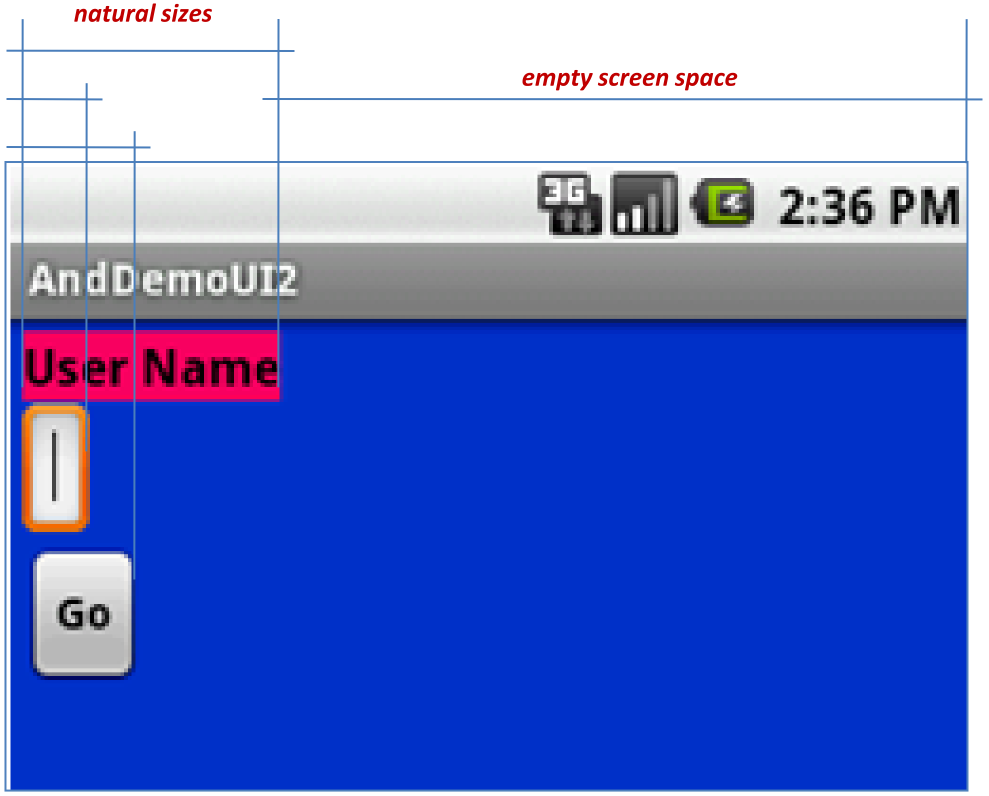


```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
android:id="@+id/myLinearLayout"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:background="#ff0033cc"
android:padding="4dip"
xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="horizontal" >
<TextView
android:id="@+id/labelUserName"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:background="#ffff0066"
android:text="User Name"
android:textSize="16sp"
android:textStyle="bold"
android:textColor="#ff000000"
>
</TextView>
<EditText
android:id="@+id/ediName"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textSize="18sp"
>
</EditText>
<Button
android:id="@+id/btnGo"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Go"
android:textStyle="bold"
>
</Button>
</LinearLayout>
```

Basic XML Layouts - Containers

1.2 Linear Layout: Fill Model

- Widgets have a "natural" size based on their accompanying text.
- When their combined sizes does not *exactly* match the width of the Android device's screen, we may have the issue of what to do with the remaining space.



Basic XML Layouts - Containers

1.2 Linear Layout: Fill Model

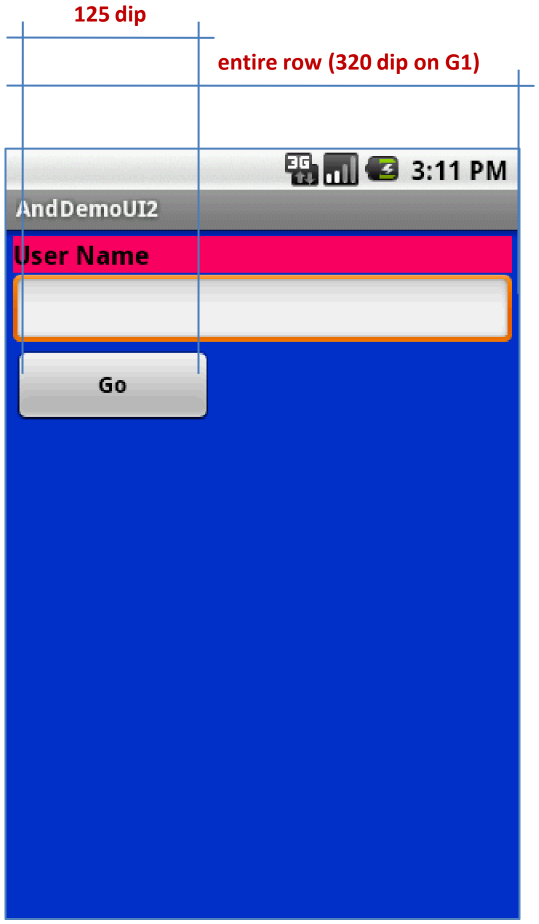
All widgets inside a LinearLayout **must** supply dimensional attributes
`android:layout_width` and `android:layout_height`
to help address the issue of empty space.

Values used in defining height and width are:

1. Specific a *particular dimension*, such as **125dip** (device independent pixels)
2. Provide **wrap_content**, which means the widget should fill up its natural space, unless that is too big, in which case Android can use **word-wrap** as needed to make it fit.
3. Provide **fill_parent**, which means the widget should fill up all available space in its enclosing container, after all other widgets are taken care of.

Basic XML Layouts - Containers

1.2 Linear Layout: Fill Model



G1 phone resolution is: 320 x 480 dip (3.2 in).

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
android:id="@+id/myLinearLayout"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:background="#ff0033cc"
android:padding="4dip"
android:orientation="vertical"
xmlns:android="http://schemas.android.com/apk/res/android"
>
<TextView
android:id="@+id/labelUserName"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:background="#ffff0066"
android:text="User Name"
android:textSize="16sp"
android:textStyle="bold"
android:textColor="#ff000000"
>
</TextView>
<EditText
android:id="@+id/ediName"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:textSize="18sp"
>
</EditText>
<Button
android:id="@+id/btnGo"
android:layout_width="125dip"
android:layout_height="wrap_content"
android:text="Go"
android:textStyle="bold"
>
</Button>
</LinearLayout>
```

Row-wise

Use all the row

Specific size: 125dip

Basic XML Layouts - Containers

1.2 Linear Layout: Weight

It is used to proportionally assign space to widgets in a view.

You set **android:layout_weight** to a value (1, 2, 3, …) to indicates what proportion of the free space should go to that widget.

Example

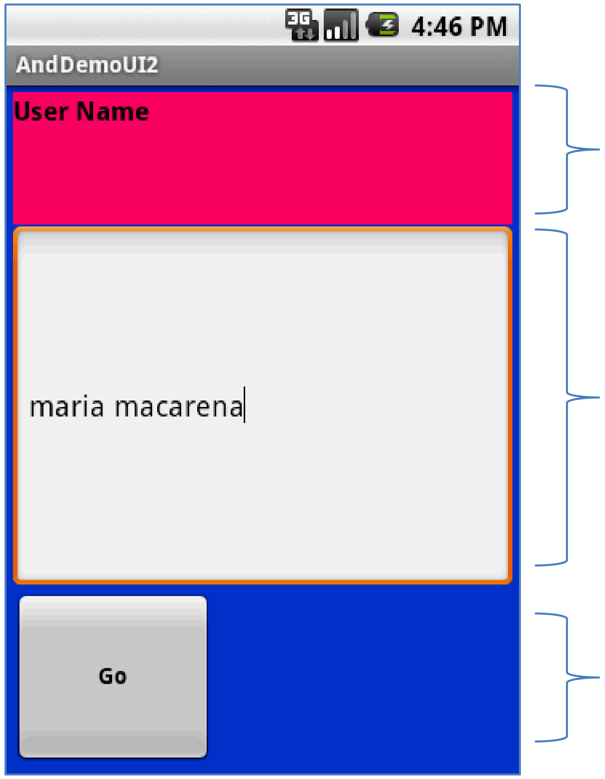
Both the *TextView* and the *Button* widgets have been set as in the previous example. Both have the additional property

`android:layout_weight="1"`

whereas the EditText control has

`android:layout_weight="2"`

Default value is 0

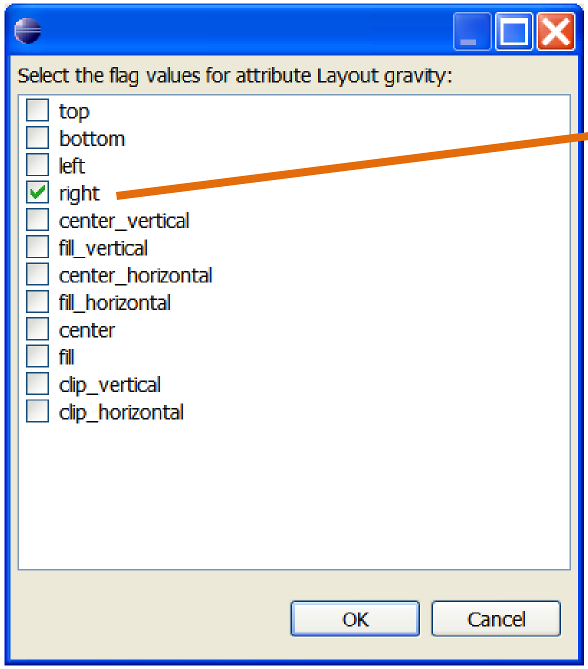


Takes: 2 /(1+1+2)
of the screen space

Basic XML Layouts - Containers

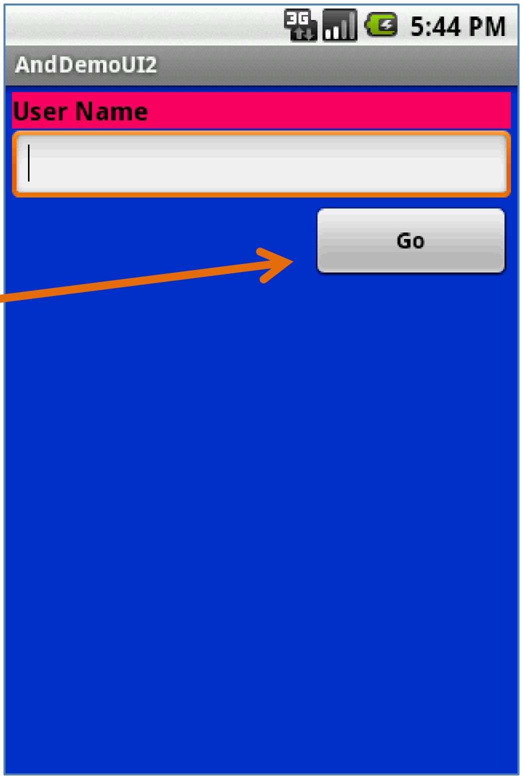
1.3 Linear Layout: Gravity

- It is used to indicate how a control will align on the screen.
- By default, widgets are left- and top-aligned.
- You may use the XML property **android:layout_gravity="…"** to set other possible arrangements: *left, center, right, top, bottom,* etc.



Select the flag values for attribute Layout gravity:

- [ ] top
- [ ] bottom
- [ ] left
- [✓] right
- [ ] center_vertical
- [ ] fill_vertical
- [ ] center_horizontal
- [ ] fill_horizontal
- [ ] center
- [ ] fill
- [ ] clip_vertical
- [ ] clip_horizontal



Button has **right** gravity

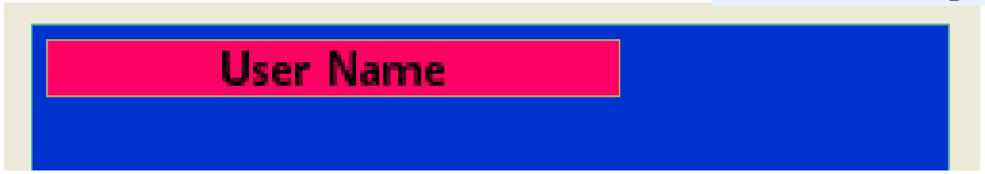
Basic XML Layouts - Containers

1.3 CAUTION: gravity vs. layout_gravity

The difference between:

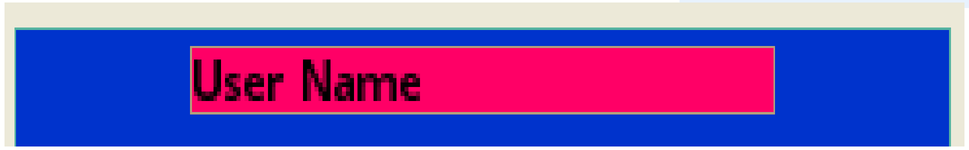
android:gravity
specifies how to place the content of an object, both on the x- and y-axis, within the object itself.

```
android:gravity="center"
```



android:layout_gravity
positions the view with respect to its parent (i.e. what the view is contained in).

```
android:layout_gravity="center"
```



Basic XML Layouts - Containers

1.4 Linear Layout: Padding

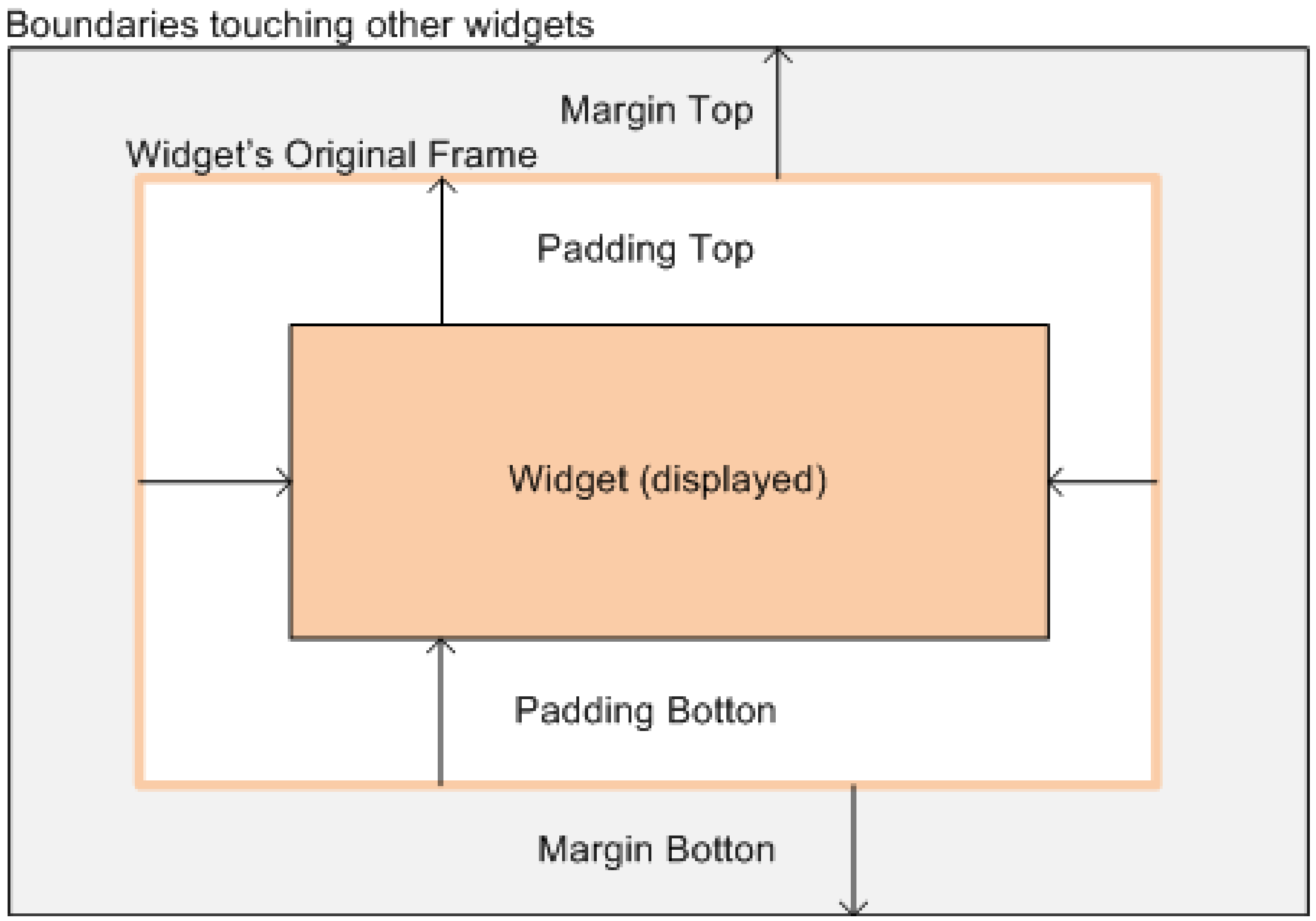
- The padding specifies how much space there is between the boundaries of the widget's "cell" and the actual widget contents.

- If you want to increase the *internal* whitespace between the edges of the and its contents, you will want to use the:
  - **android:padding** property
  - or by calling *setPadding*() at runtime on the widget's Java object.

Note: Padding is analogous to the margins on a word processing document.

Basic XML Layouts - Containers

1.3 Linear Layout: Padding and Marging

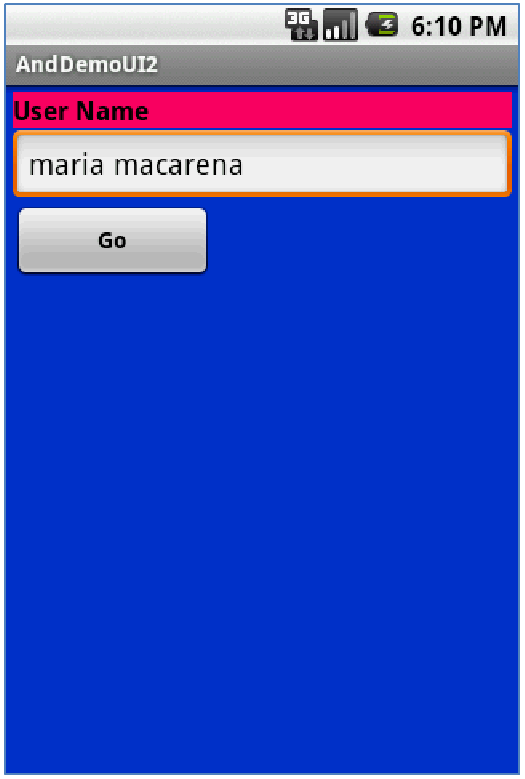


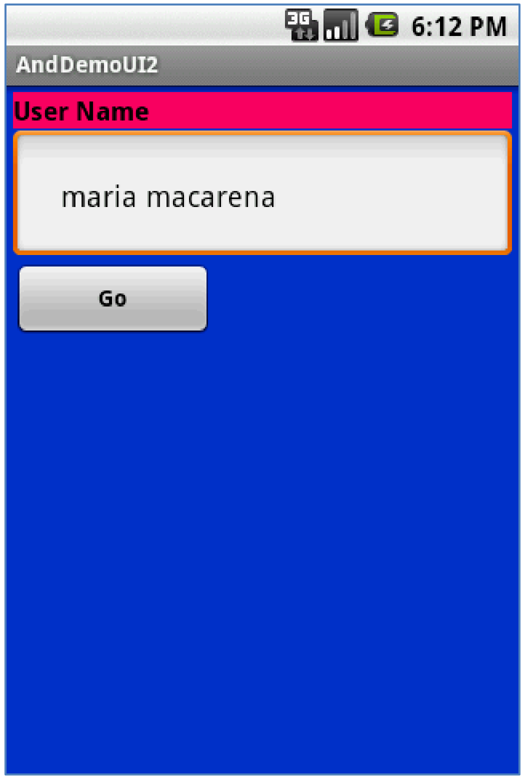
Basic XML Layouts - Containers

1.3 Linear Layout: Internal Margins Using Padding

Example:

The EditText box has been changed to display 30dip of padding all around



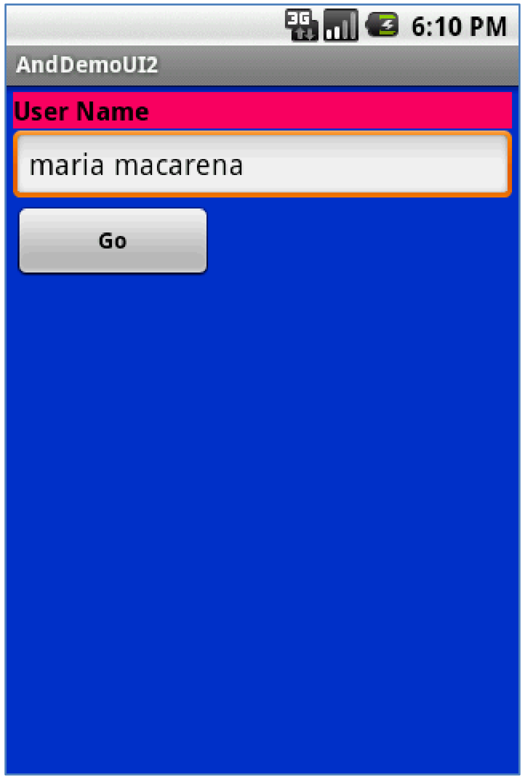


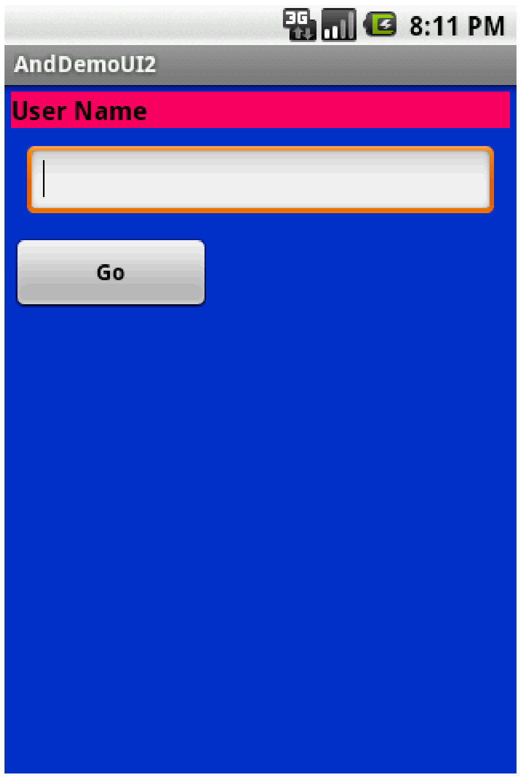
```
<EditText
android:id="@+id/ediName"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:textSize="18sp"

android:padding="30dip"

>
</EditText>
...
```

Basic XML Layouts - Containers

1.4 Linear Layout: (External) Marging

- By default, widgets are tightly packed next to each other.
- To increase space between them use the **android:layout_margin** attribute





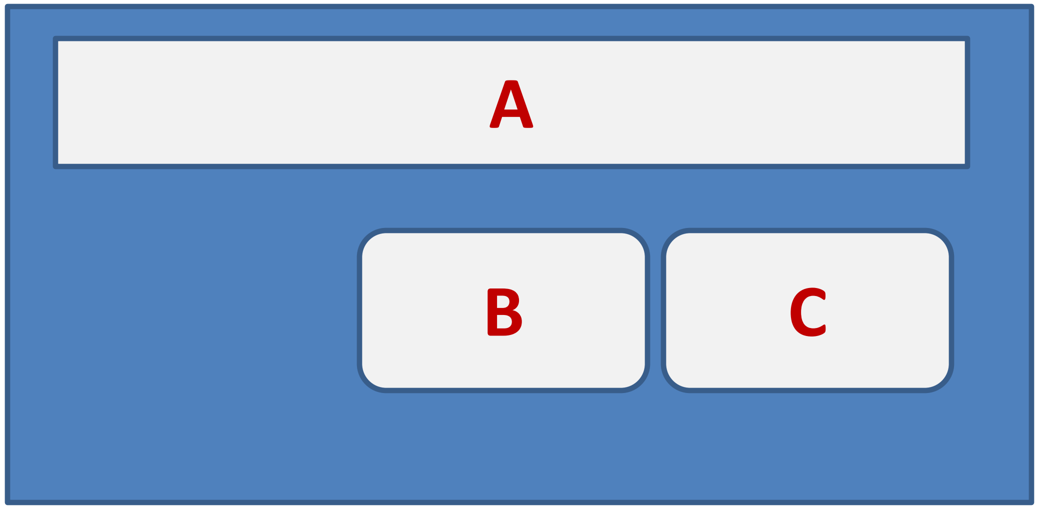
Increased inter-widget space

```
<EditText
android:id="@+id/ediName"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:textSize="18sp"

android:layout_margin="6dip"

>
</EditText>
...
```

Basic XML Layouts - Containers

2. Relative Layout

RelativeLayout places widgets based on their relationship to other widgets in the container and the parent container.



Example:
A is by the parent’s top
C is below A, to its right
B is below A, to the left of C

Basic XML Layouts - Containers

2. Relative Layout - Referring to the container

Some positioning XML (boolean) properties mapping a widget according to its location **respect to the parent's place** are:

- **android:layout_alignParentTop** says the widget's top should align with the top of the container
- **android:layout_alignParentBottom** the widget's bottom should align with the bottom of the container
- **android:layout_alignParentLeft** the widget's left side should align with the left side of the container
- **android:layout_alignParentRight** the widget's right side should align with the right side of the container

- **android:layout_centerInParent** the widget should be positioned both horizontally and vertically at the center of the container
- **android:layout_centerHorizontal** the widget should be positioned horizontally at the center of the container
- **android:layout_centerVertical** the widget should be positioned vertically at the center of the container

Basic XML Layouts - Containers

2. Relative Layout – Referring to other widgets

The following properties manage positioning of a widget **respect to other widgets:**

- **android:layout_above** indicates that the widget should be placed above the widget referenced in the property
- **android:layout_below** indicates that the widget should be placed below the widget referenced in the property
- **android:layout_toLeftOf** indicates that the widget should be placed to the left of the widget referenced in the property
- **android:layout_toRightOf** indicates that the widget should be placed to the right of the widget referenced in the property

Basic XML Layouts - Containers

2. Relative Layout – Referring to other widgets – cont.

- **android:layout_alignTop** indicates that the widget's top should be aligned with the top of the widget referenced in the property
- **android:layout_alignBottom** indicates that the widget's bottom should be aligned with the bottom of the widget referenced in the property
- **android:layout_alignLeft** indicates that the widget's left should be aligned with the left of the widget referenced in the property
- **android:layout_alignRight** indicates that the widget's right should be aligned with the right of the widget referenced in the property
- **android:layout_alignBaseline** indicates that the baselines of the two widgets should be aligned

Basic XML Layouts - Containers

2. Relative Layout – Referring to other widgets

In order to use Relative Notation in Properties you need to consistently:

1. Put identifiers (`android:id` attributes) on *all elements* that you will need to address.

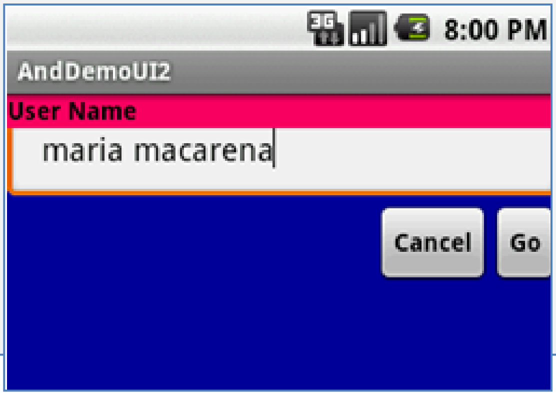
2. Syntax is: **@+id/...** (for instance an EditText box could be XML called: `android:id="@+id/ediUserName"`)

3. Reference other widgets using the same identifier value (**@+id/...**) already given to a widget. For instance a control below the EditText box could say: `android:layout_below="@+id/ediUserName"`

Basic XML Layouts - Containers

2. Relative Layout – Example

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
android:id="@+id/myRelativeLayout"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:background="#ff000099"
xmlns:android="http://schemas.android.com/apk/res/andr
oid">

<TextView
android:id="@+id/lblUserName"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:background="#ffff0066"
android:text="User Name"
android:textStyle="bold"
android:textColor="#ff000000"
android:layout_alignParentTop="true"
android:layout_alignParentLeft="true">
</TextView>

<EditText
android:id="@+id/ediUserName"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:layout_below="@+id/lblUserName"
android:layout_alignParentLeft="true"
android:layout_alignLeft="@+id/myRelativeLayout"
android:padding="20dip">
</EditText>

<Button
android:id="@+id/btnGo"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_below="@+id/ediUserName"
android:layout_alignRight="@+id/ediUserName"
android:text="Go"
android:textStyle="bold">
</Button>

<Button
android:id="@+id/btnCancel"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_toLeftOf="@+id/btnGo"
android:layout_below="@+id/ediUserName"
android:text="Cancel"
android:textStyle="bold">
</Button>
</RelativeLayout>
```

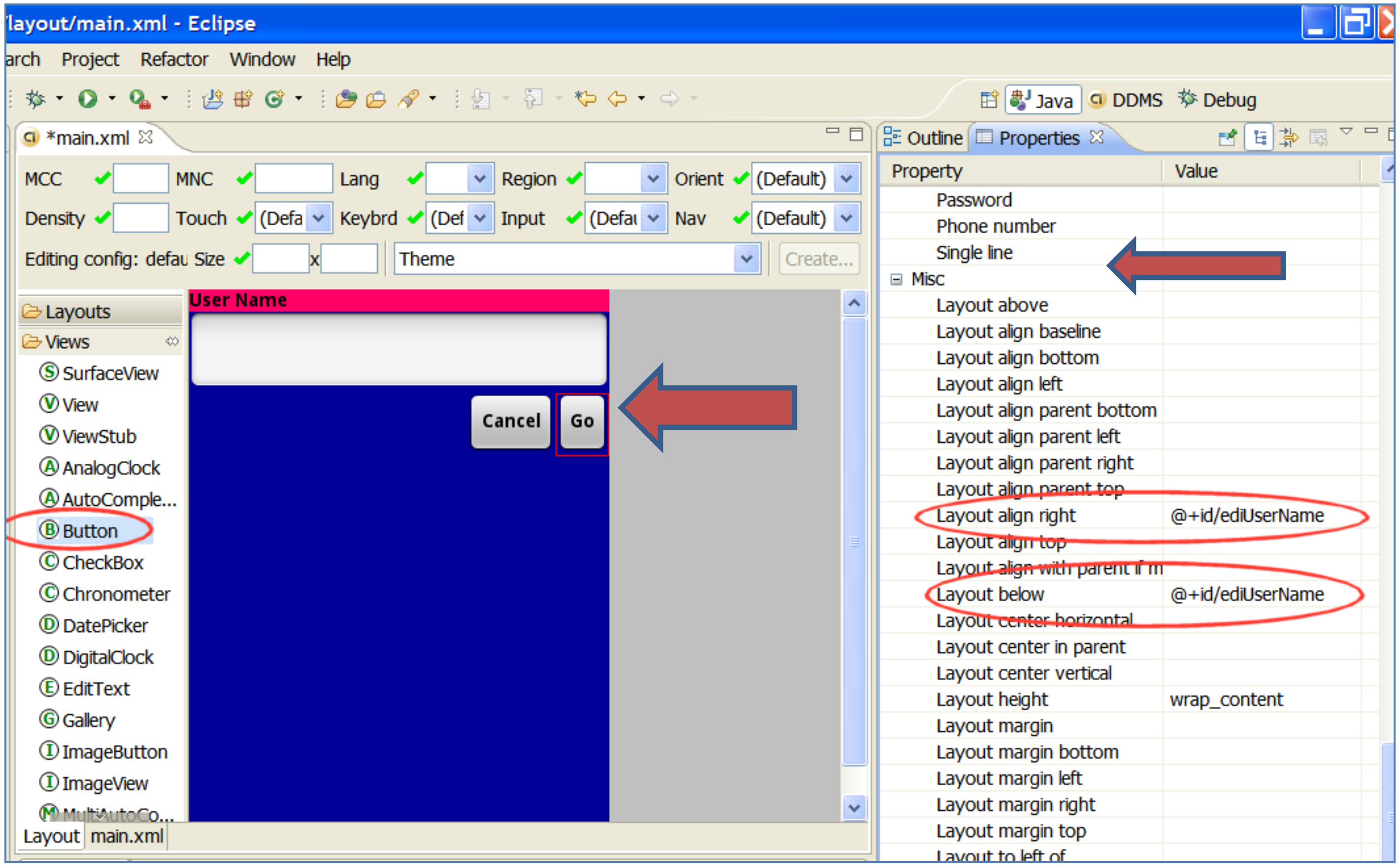


Basic XML Layouts - Containers

2. Relative Layout – Comment (as of Aug. 2009)

Use the **Eclipse ADT Layout Editor** for laying out *RelativeLayouts*.

DroidDraw is of very little help in this respect.

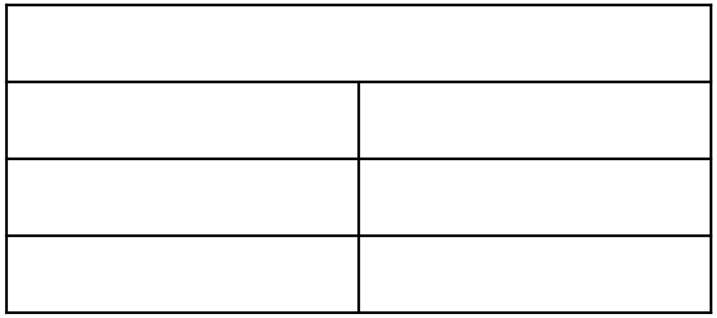


Property	Value
Password	
Phone number	
Single line	
Misc	
Layout above	
Layout align baseline	
Layout align bottom	
Layout align left	
Layout align parent bottom	
Layout align parent left	
Layout align parent right	
Layout align parent top	
Layout align right	@+id/ediUserName
Layout align top	
Layout align with parent if m	
Layout below	@+id/ediUserName
Layout center horizontal	
Layout center in parent	
Layout center vertical	
Layout height	wrap_content
Layout margin	
Layout margin bottom	
Layout margin left	
Layout margin right	
Layout margin top	
Layout to left of	

Basic XML Layouts - Containers

3. Table Layout

1. Android's **TableLayout** allows you to position your widgets in a *grid* made of identifiable *rows* and *columns*.
2. Columns might *shrink* or *stretch* to accommodate their contents.
3. TableLayout works in conjunction with *TableRow*.
4. TableLayout controls the overall behavior of the container, with the widgets themselves positioned into one or more *TableRow* containers, one per row in the grid.



Basic XML Layouts - Containers

3. Table Layout

Rows are declared by you by putting widgets as children of a **TableRow** inside the overall *TableLayout*.

The *number of columns is determined by Android* (you control the number of columns in an indirect way).

So if you have three rows, one with two widgets, one with three widgets, and one with four widgets, there will be at least four columns.

0		1	
0		1	2
0	1	2	3

Basic XML Layouts - Containers

3. Table Layout

However, a single widget can take up more than one column by including the **android:layout_span** property, indicating the number of columns the widget spans (this is similar to the **colspan** attribute one finds in table cells in **HTML**)

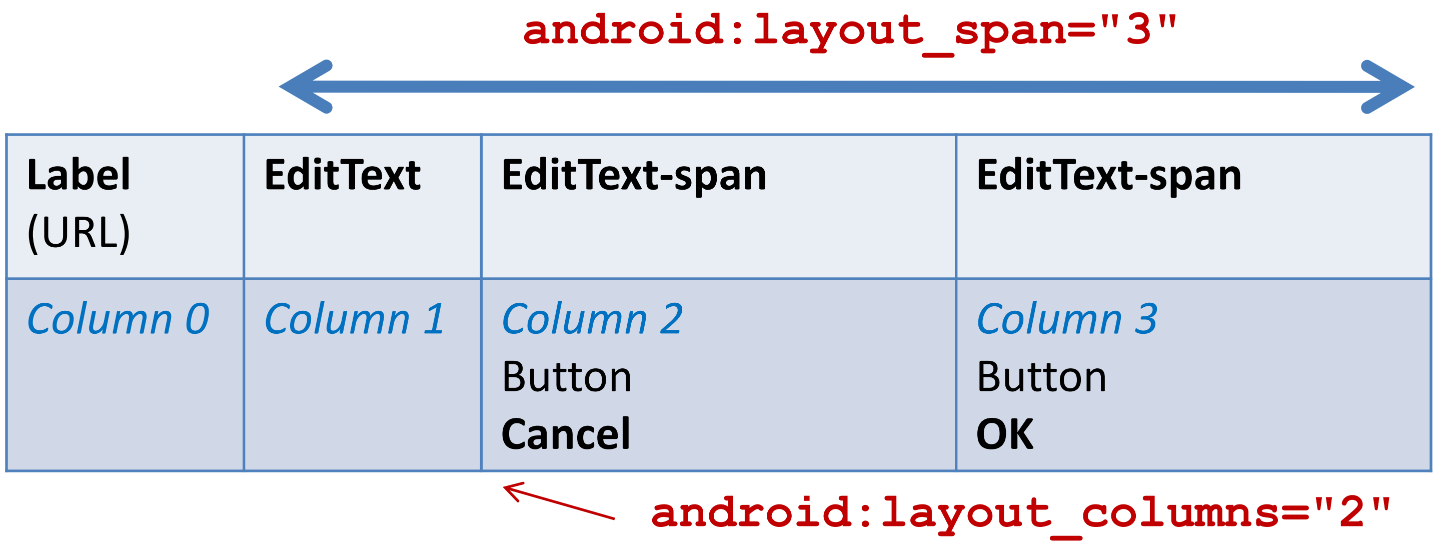
```
<TableRow>
   <TextView android:text="URL:" />
   <EditText
   android:id="@+id/entry"
   android:layout_span="3" />
</TableRow>
```

Basic XML Layouts - Containers

3. Table Layout

Ordinarily, widgets are put into the first available column of each row.

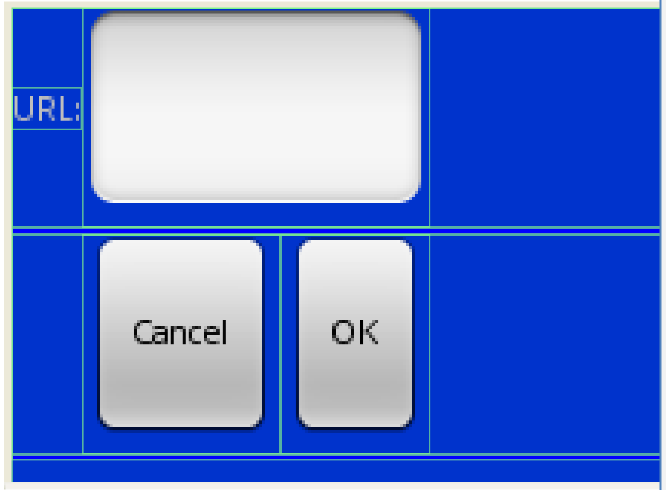
In the example below, the label (“*URL*”) would go in the first column (*column 0, as columns are counted starting from 0*), and the TextField would go into a spanned set of three columns (columns 1 through 3).



Label (URL)	**EditText**	**EditText-span**	**EditText-span**
Column 0	*Column 1*	*Column 2* Button **Cancel**	*Column 3* Button **OK**

Basic XML Layouts - Containers

3. Table Layout – Example



Note to the reader:
Experiment changing layout_span to 1, 2, 3

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
android:id="@+id/myTableLayout"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:background="#ff0033cc"
android:orientation="vertical"
xmlns:android="http://schemas.android.com/apk/res/android"
>
<TableRow>
<TextView
android:text="URL:" />
<EditText android:id="@+id/ediUrl"
android:layout_span="3"/>
</TableRow>
<View
android:layout_height="3dip"
android:background="#0000FF" />
<TableRow>
<Button android:id="@+id/cancel"
android:layout_column="2"
android:text="Cancel" />
<Button android:id="@+id/ok"
android:text="OK" />
</TableRow>
<View
android:layout_height="3dip"
android:background="#0000FF" />

</TableLayout>
```

Strech up to column 3

Skip columns: 0, 1

Basic XML Layouts - Containers

3. Table Layout

By default, each column will be sized according to the "*natural*" size of the widest widget in that column.

If your content is narrower than the available space, you can use the *TableLayout* property:

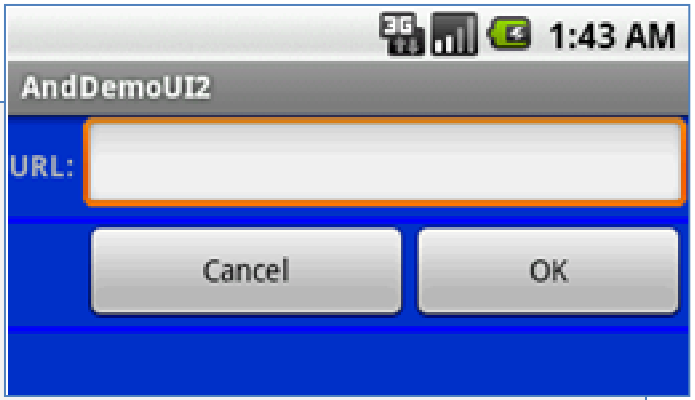
```
android:stretchColumns =“…”
```

Its value should be a single column number (0-based) or a comma-delimited list of column numbers. Those columns will be stretched to take up any available space yet on the row.

Basic XML Layouts - Containers

3. Table Layout

In our running example we stretch columns 2, 3, and 4 to fill the rest of the row.



```
. . .
<TableLayout
android:id="@+id/myTableLayout"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:background="#ff0033cc"
android:orientation="vertical"
android:stretchColumns ="2,3,4"
xmlns:android="http://schemas.android.com/apk/res/android"
>

 . . .
```

***TODO**: try to stretch one column at the time 1, then 2, and so on.*

Basic XML Layouts - Containers

4. ScrollView Layout

When we have more data than what can be shown on a single screen you may use the **ScrollView** control.

It provides a sliding or scrolling access to the data. This way the user can only see part of your layout at one time, but the rest is available via scrolling.

This is similar to browsing a large web page that forces the user to scroll up the page to see the bottom part of the form.

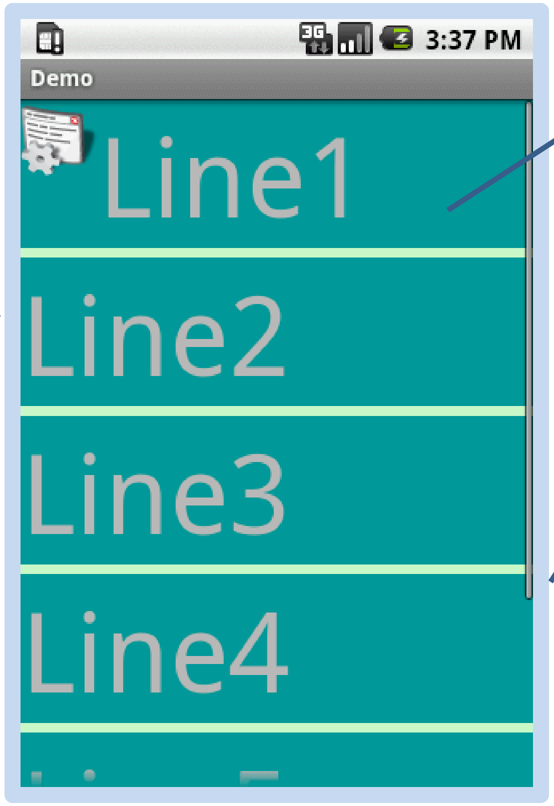
Basic XML Layouts - Containers

4. Example ScrollView Layout

```xml
<?xml version="1.0" encoding="utf-8"?>
<ScrollView
android:id="@+id/myScrollView1"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:background="#ff009999"
xmlns:android="http://schemas.android.com/apk/res/android"
>
<LinearLayout
android:id="@+id/myLinearLayoutVertical"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:orientation="vertical"
>

<LinearLayout
android:id="@+id/myLinearLayoutHorizontal1"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:orientation="horizontal"
>
<ImageView
        android:id="@+id/myPicture"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/icon" />
<TextView
        android:id="@+id/textView1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Line1"
        android:textSize="70dip" />
</LinearLayout>

<View
        android:layout_width="fill_parent"
        android:layout_height="6dip"
        android:background="#ffccffcc" />
<TextView
        android:id="@+id/textView2"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Line2"
        android:textSize="70dip" />
<View
        android:layout_width="fill_parent"
        android:layout_height="6dip"
        android:background="#ffccffcc" />
<TextView
        android:id="@+id/textView3"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Line3"
        android:textSize="70dip" />
<View
        android:layout_width="fill_parent"
        android:layout_height="6dip"
        android:background="#ffccffcc" />
<TextView
        android:id="@+id/textView4"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Line4"
        android:textSize="70dip" />
<View
        android:layout_width="fill_parent"
        android:layout_height="6dip"
        android:background="#ffccffcc" />
<TextView
        android:id="@+id/textView5"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Line5"
        android:textSize="70dip" />
</LinearLayout>

</ScrollView>
```

Basic XML Layouts - Containers

4. Example ScrollView Layout



Simple TextView

Combining an ImageView & TextView in a horizontal Linear Layout

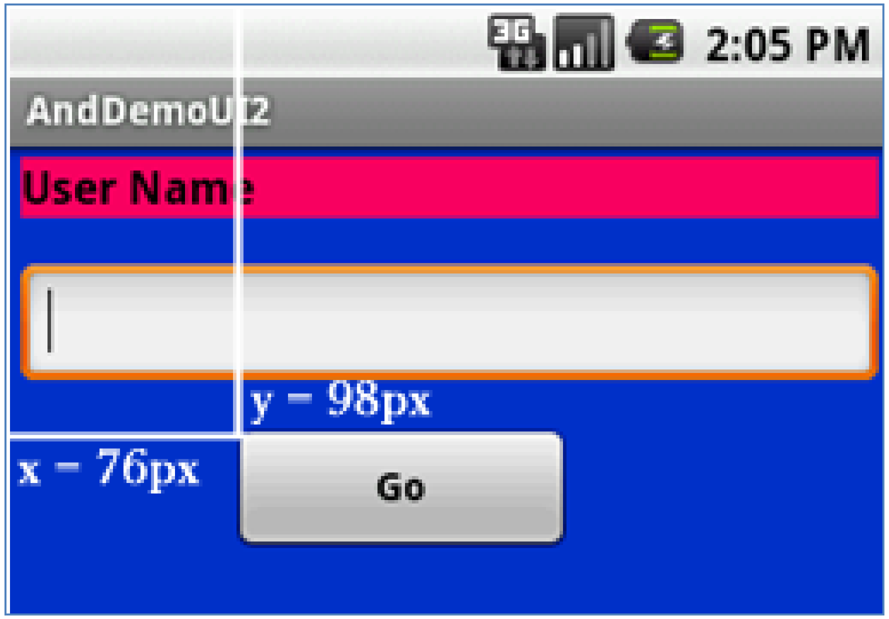
Scroller

Basic XML Layouts - Containers

5. Miscellaneous.
Absolute Layout

- A layout that lets you specify exact locations (x/y coordinates) of its children.

- Absolute layouts are *less flexible* and harder to maintain than other types of layouts without absolute positioning.



Basic XML Layouts - Containers

5. Miscellaneous Absolute Layout (cont.)

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout
android:id="@+id/myLinearLayout"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:background="#ff0033cc"
android:padding="4dip"
xmlns:android="http://schemas.android.com
/apk/res/android"
>

<TextView
android:id="@+id/tvUserName"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:background="#ffff0066"
android:text="User Name"
android:textSize="16sp"
android:textStyle="bold"
android:textColor="#ff000000"
android:layout_x="0dip"
android:layout_y="10dip"
>
</TextView>
<EditText
android:id="@+id/etName"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:textSize="18sp"
android:layout_x="0dip"
android:layout_y="38dip"
>
</EditText>

<Button
android:layout_width="120dip"
android:text="Go"
android:layout_height="wrap_content"
android:textStyle="bold"
android:id="@+id/btnGo"
android:layout_x="100dip"
android:layout_y="170dip"  />
</AbsoluteLayout>
```

Button location

Basic XML Layouts - Containers

Questions?