



Graphical User Interfaces

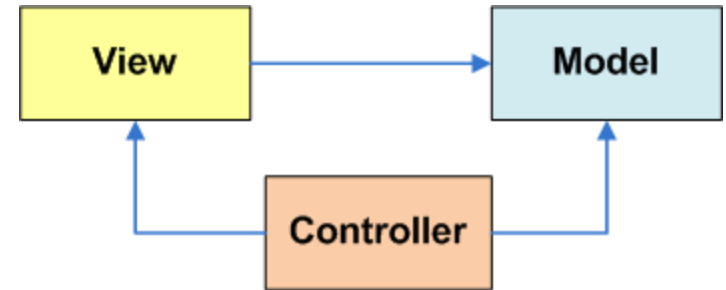
Victor Matos

Cleveland State University

Portions of this page are reproduced from work created and [shared by Google](#) and used according to terms described in the [Creative Commons 3.0 Attribution License](#).

The Model-View-Control (MVC) Pattern

The *Model-View-Controller (MVC)* is an important software design pattern whose main goal is to separate the (1) user interface, (2) business, and (3) input logic.



How is this seen by the Android developer?

- **Model.** Consists of the Java code and API objects used to manage the behavior and data of the application.
- **View.** Set of screens the user sees and interacts with.
- **Controller.** Implemented through the Android OS, responsible for interpretation of the user and system inputs. Input may come from a variety of sources such as the trackball, keyboard, touchscreen, GPS chip, background services, etc, and tells the Model and/or the View (usually through callbacks and registered listeners) to change as appropriate.

[Burbeck92] Burbeck, Steve. "Application Programming in Smalltalk-80: How to use Model-View-Controller (MVC)." *University of Illinois in Urbana-Champaign (UIUC) Smalltalk Archive*. Available at: <http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html>.

The Model-View-Control (MVC) Pattern

Getting ready to create MVC conforming solutions

The Android developer should be aware of ...

- **Inputs** could be sent to the application from various physical/logical components. Reacting to those signals is typically handled by **callback methods**. Usually there are many of them, you want to learn how to choose the appropriate one.
- Moving to states in the **lifecycle** is tied to logic in the model. For instance, if forced to *Pause* you may want to save uncommitted data.
- A **notification** mechanism is used to inform the user of important events happening *outside* the current application (such as arrival of a text message or email, fluctuations of the stock market, etc) and consequently choose how to proceed.
- **Views** are unlimited in terms of aesthetic and functionality. However physical constraints such as size, and hardware acceleration (or lack of) may affect how graphical components are managed.

Android & the MVC Pattern

The View - User Interfaces (Uis)

Android **graphical interfaces** are usually implemented as XML files (although they could also be dynamically created from Java code).

An Android UI is conceptually similar to a common HTML page

- **In a manner similar to a web page interaction**, when the Android user touches the screen, the controller interprets the input and determines what specific portion of the screen and gestures were involved. Based on this information it tells the model about the interaction in such a way that the appropriate “callback listener” or lifecycle state could be called into action.
- **Unlike a web application** (which refreshes its pages after explicit requests from the user) an asynchronous Android background service could quietly notify the controller about some change of state (such as reaching a given coordinate on a map) and in turn a change of the view’s state could be triggered; all of these without user intervention.

UI Design Patterns

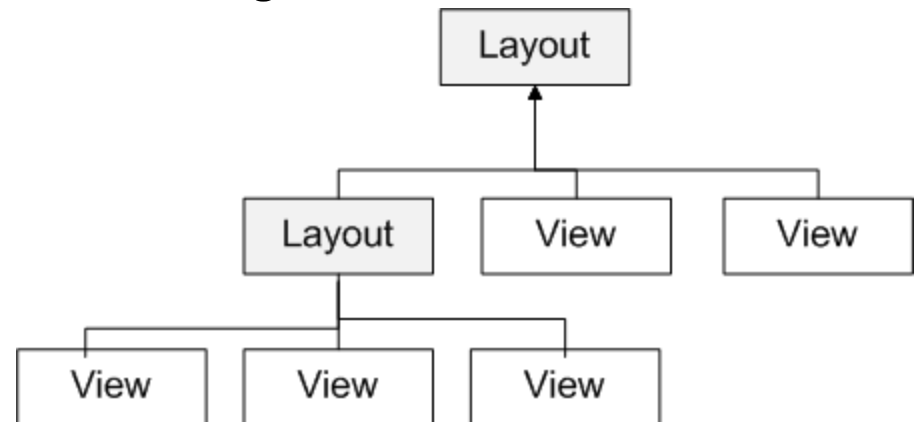
For a detailed discussion on **Android UI Design Patterns** see video:

http://www.youtube.com/watch?v=M1ZBjICRfz0&feature=player_embedded

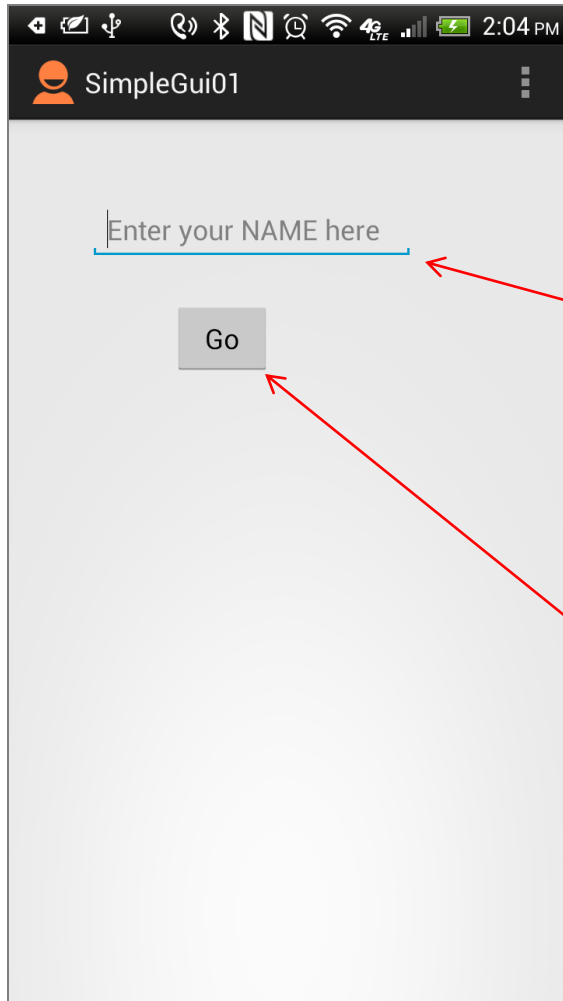


The View Class

- The **View class** is the Android's most basic component from which users interfaces can be created. It acts as a container of displayable elements.
- A **View** occupies a rectangular area on the screen and is responsible for *drawing* and *event handling*.
- **Widgets** are subclasses of View. They are used to create interactive UI components such as buttons, checkboxes, labels, text fields, etc.
- **Layouts** are invisible containers used for holding other Views and nested layouts.



Graphical UI ↔ XML Layout



Actual UI displayed by the app

Text version: *activity_main.xml* file



```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
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=".MainActivity" >

<EditText
    android:id="@+id/editText1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentTop="true"
    android:layout_marginLeft="35dp"
    android:layout_marginTop="35dp"
    android:ems="10"
    android:hint="Enter your NAME here" />

<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/editText1"
    android:layout_below="@+id/editText1"
    android:layout_marginLeft="54dp"
    android:layout_marginTop="26dp"
    android:text="Go" />

</RelativeLayout>
```

Using Views

- An Android's **XML** view file consists of a **layout** holding a hierarchical arrangement of its contained elements.
- The inner elements could be simple widgets or nested layouts holding some complex viewgroups.
- An Activity uses the `setContentView(R.layout.xmlfilename)` method to render a view on the device's screen.

```
<LinearLayout  
  xmlns:android="http://schemas.android.com/apk/res/android"  
  android:layout_width="match_parent"  
  android:layout_height="wrap_content"  
  android:orientation="horizontal" >
```

Widgets and other nested layouts

```
</LinearLayout>
```

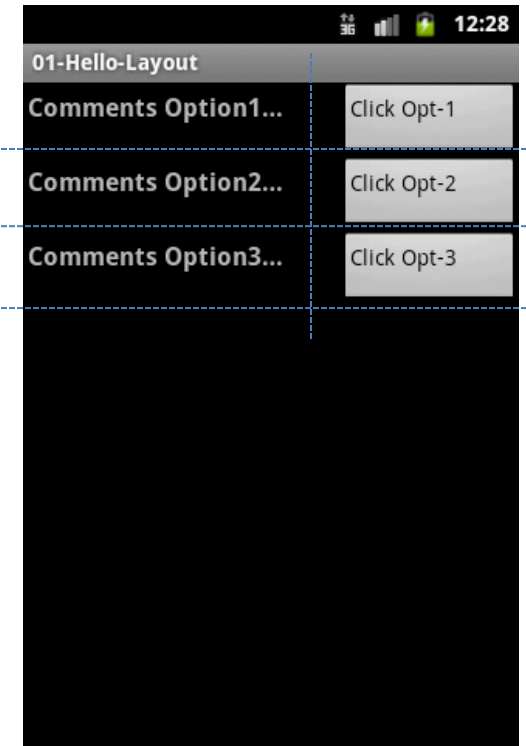
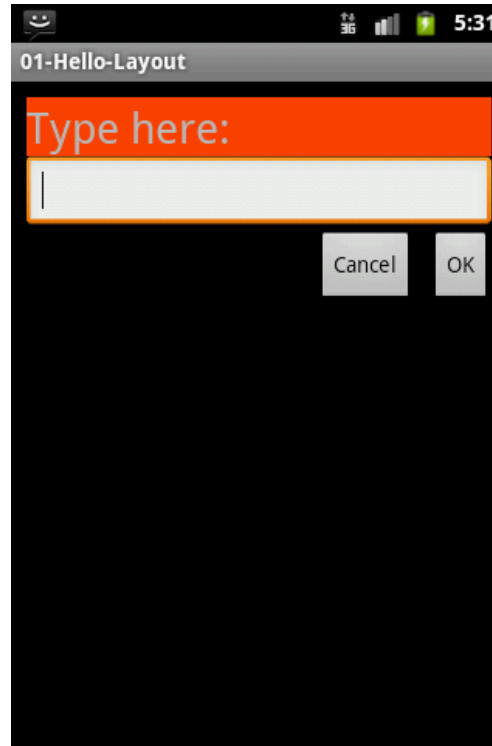
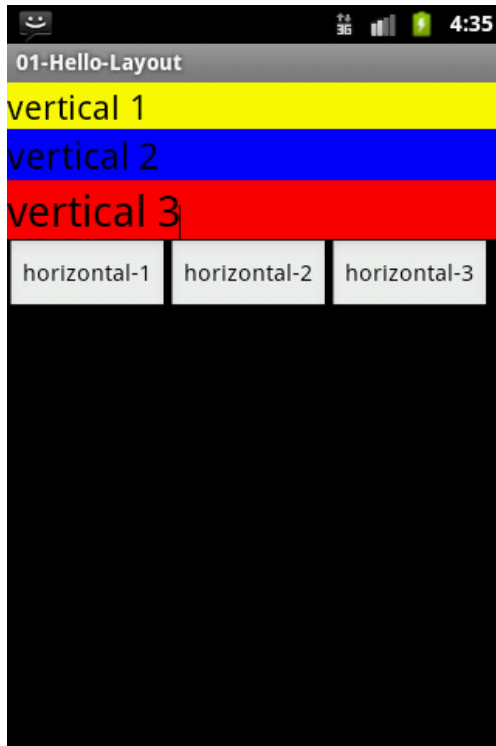

Using Views

Dealing with widgets & layouts typically involves the following operations

1. **Set properties:** For example setting the background color, text, font and size of a *TextView*.
2. **Set up listeners:** For example, an image could be programmed to respond to various events such as: click, long-tap, mouse-over, etc.
3. **Set focus:** To set focus on a specific view, you call the method `requestFocus()` or use XML tag `<requestFocus />`
4. **Set visibility:** You can hide or show views using `setVisibility(...)`.

A brief sample of UI components

Layouts



Linear Layout

A LinearLayout places its inner views either in horizontal or vertical disposition.

Relative Layout

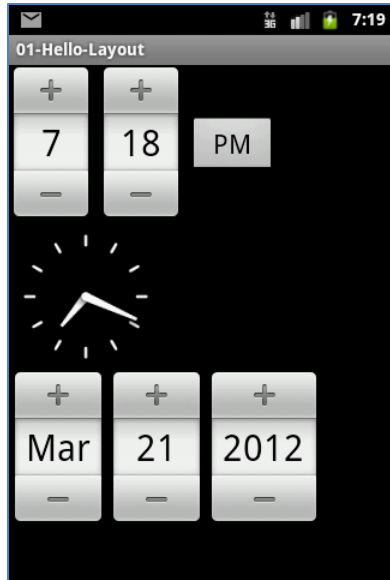
A RelativeLayout is a ViewGroup that allows you to position elements relative to each other.

Table Layout

A TableLayout is a ViewGroup that places elements using a row & column disposition.

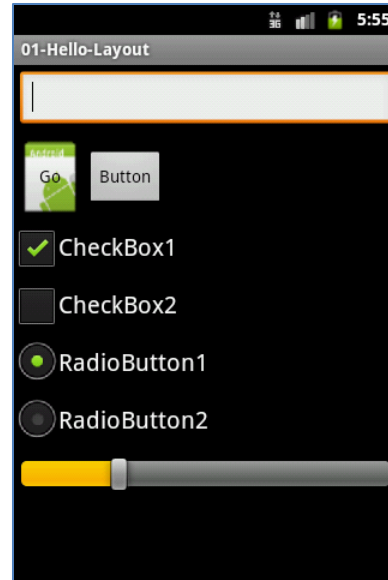
A brief sample of UI components

Widgets



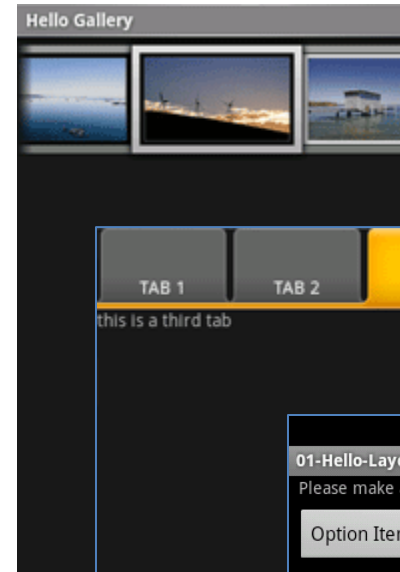
TimePicker AnalogClock DatePicker

A *DatePicke* is a widget that allows the user to select a month, day and year.



Form Controls

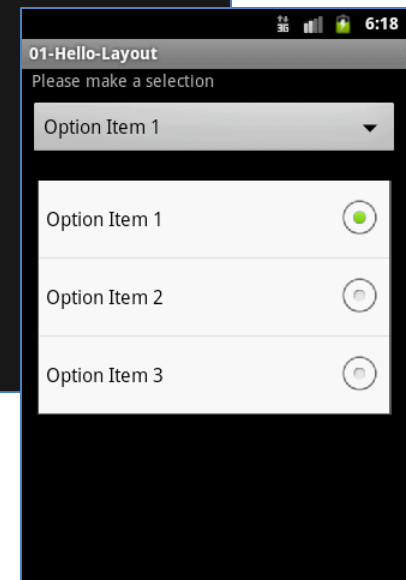
Includes a variety of typical form widgets, like: *image buttons*, *text fields*, *checkboxes* and *radio buttons*.



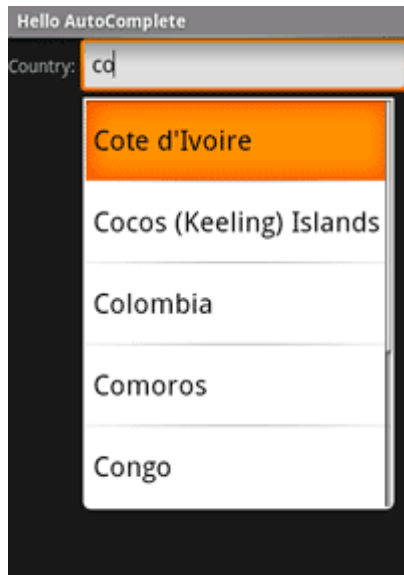
GalleryView

TabWidget

Spinner

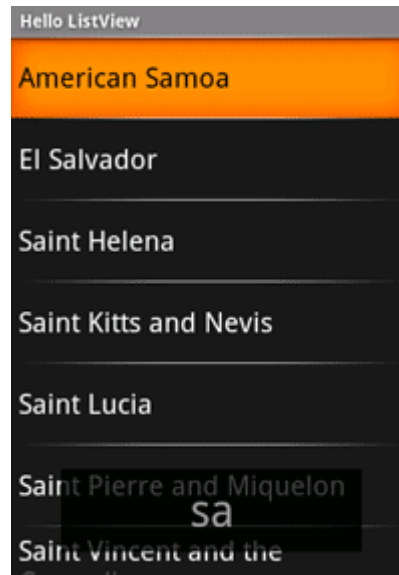


A brief sample of UI components



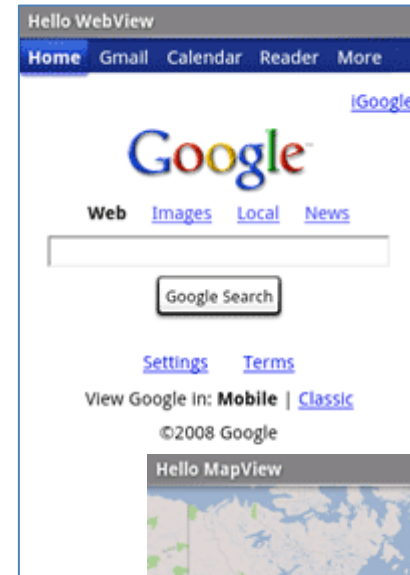
AutoCompleteTextView

It is a version of the *EditText* widget that will provide auto-complete suggestions as the user types. The suggestions are extracted from a collection of strings.

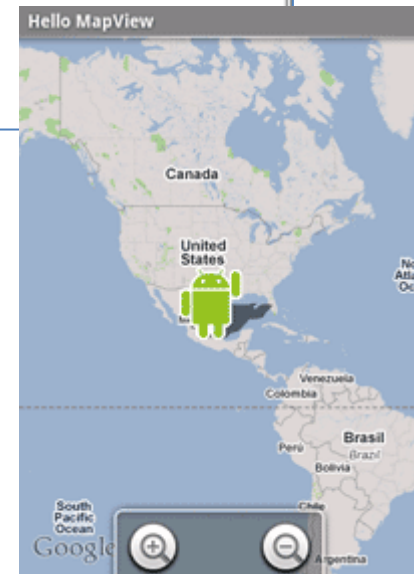


ListView

A *ListView* is a View that shows items in a vertically scrolling list. The items are acquired from a *ListAdapter*.



WebView



MapView

XML Layouts in Eclipse

Android considers XML-based layouts to be *resources*, consequently layout files are stored in the **res/layout** directory inside your Android project.

The screenshot shows the Eclipse IDE interface. On the left, the Package Explorer displays the project structure. A red circle highlights the 'res' folder, and a red arrow points to it from the label 'Resource folder'. Another red circle highlights the 'layout' subfolder, and a red arrow points to it from the label 'Package Explorer'. A blue box with the text 'XML version of a window' has a blue arrow pointing to the 'activity_main.xml' file in the Package Explorer. The main editor window shows the XML code for 'activity_main.xml', with a red circle around the file name in the editor's title bar. The code defines a RelativeLayout containing an EditText and a Button.

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

    <EditText
        android:id="@+id/editText1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/button1"
        android:layout_alignParentRight="true"
        android:layout_alignParentTop="true"
        android:ems="10"
        android:hint="Enter you name here..." >

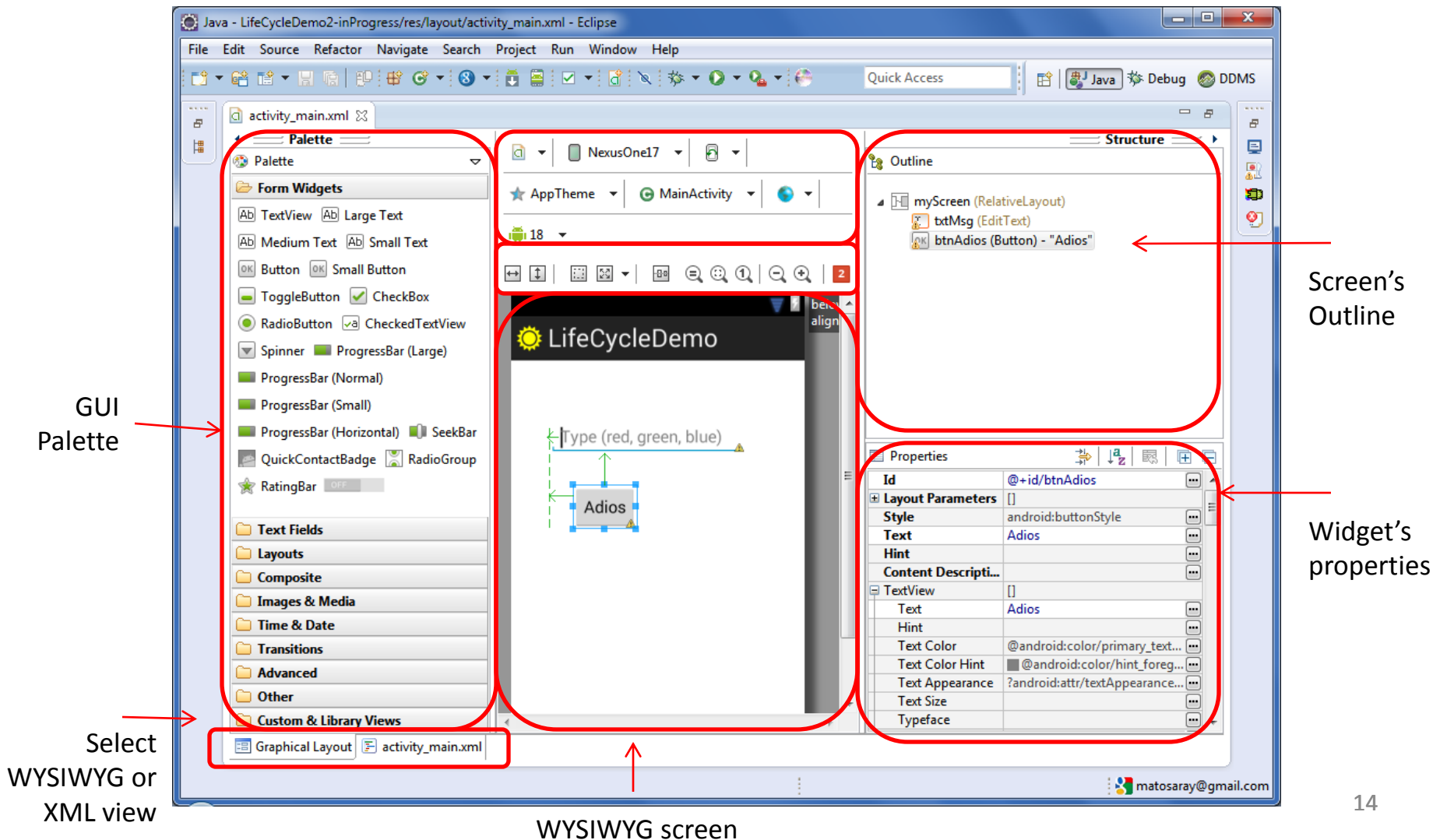
        <requestFocus />
    </EditText>

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_below="@+id/editText1"
        android:layout_marginLeft="15dp"
        android:text="Click me!" />

</RelativeLayout>
```

XML Layouts in Eclipse

The **Screen Designer Tool** included in Eclipse+ADT allows a dual view -WYSIWIG and XML- of each layout design held in the resource folder **res/layout/**



Tools for Android UI/XML Design



ASIDE ...

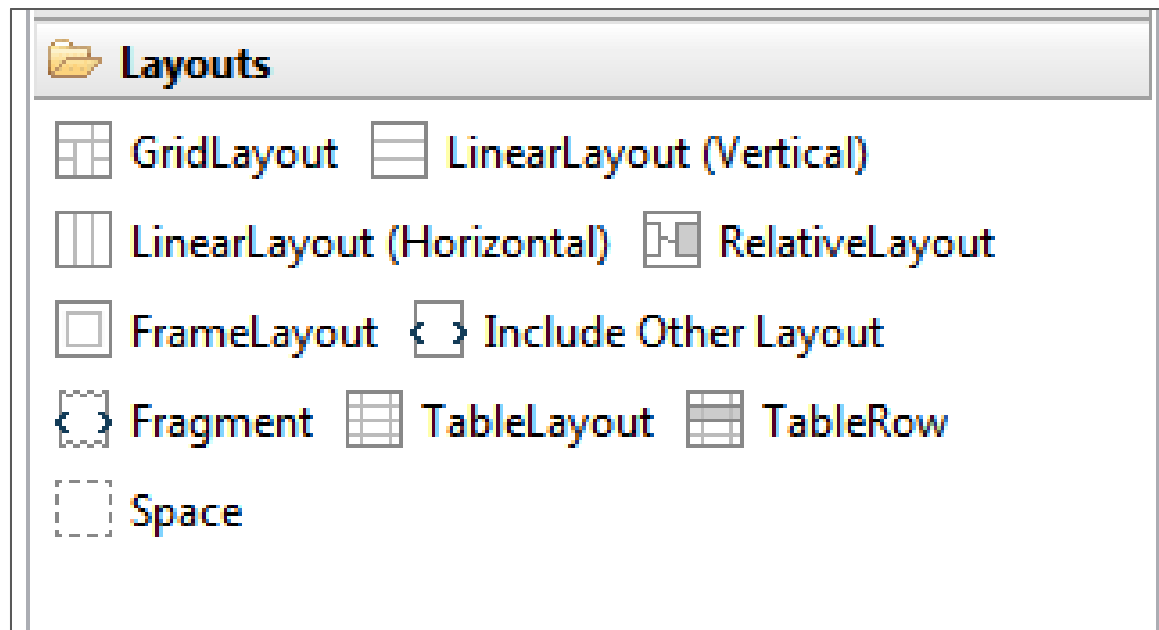
Alternative tools for creating Android apps:

- **Android Studio.** Based on IntelliJ IDEA. Similar to Eclipse with the ADT Plugin.
<http://developer.android.com/sdk/installing/studio.html>
- **Android SDK.** Streamlined workbench based on Eclipse+ADT in a simpler to install package.
<http://developer.android.com/sdk/index.html>
- **DroidDraw** Very simple GUI designer, incomplete, not integrated to the Eclipse IDE, aging! <http://www.droiddraw.org/>
- **App Inventor** (very promising & ambitious, 'hides' coding ...)
<http://appinventor.mit.edu/>

More on this issue later...

How to create Android GUIs?

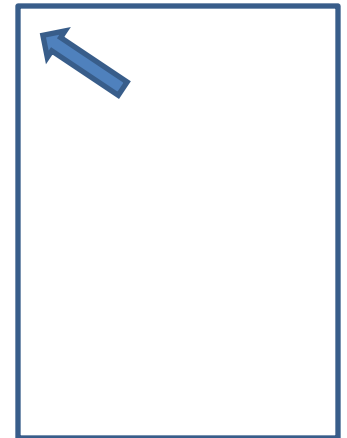
- Android *Layouts* are GUI containers having a predefined structure and placement policy.
- **Layouts can be nested**, therefore a cell, row, or column of a given layout could be another layout.
- The Eclipse+ADT workbench offers the following base types:



Common Layouts

FrameLayout

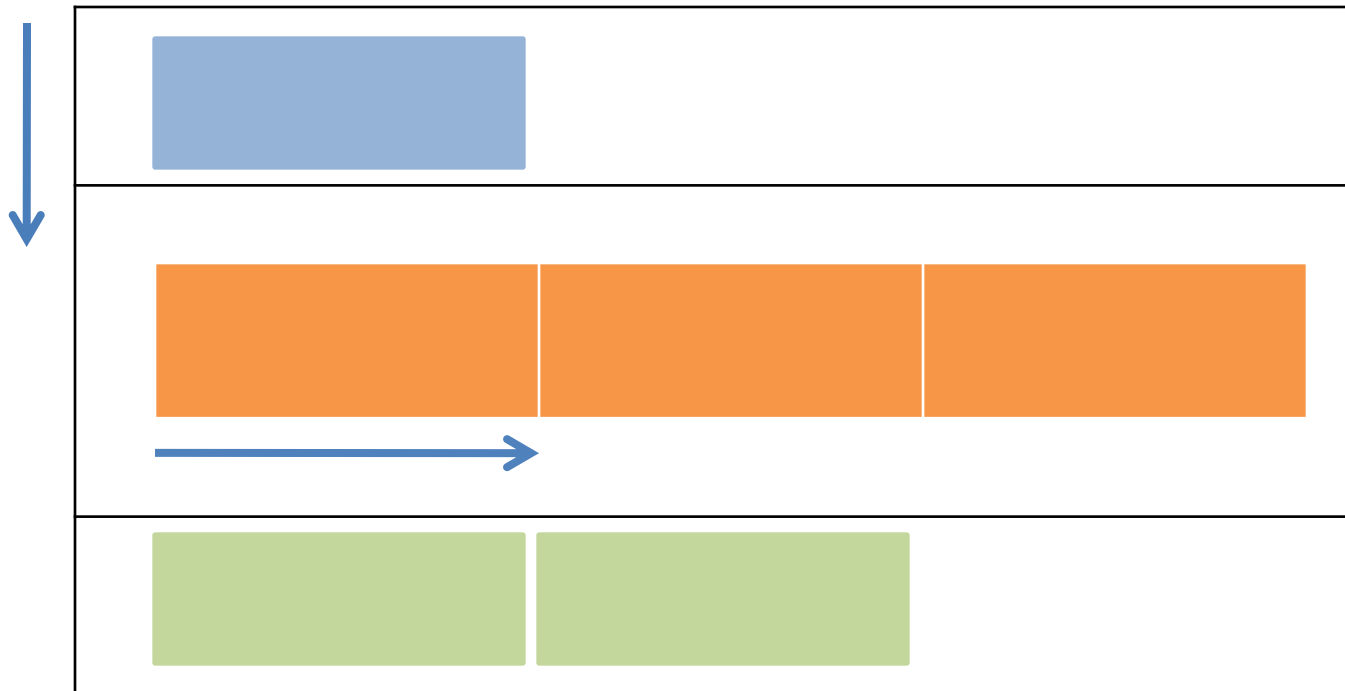
- FrameLayout is the simplest type of GUI container.
- Useful as outermost container holding a window.
- Allows you to define how much of the screen (high, width) is to be used.
- All its children elements are *aligned to the top left corner of the screen.*;



The Linear Layout

1. Linear Layout

- The **LinearLayout** supports a filling strategy in which new elements are stacked either in a **horizontal** or **vertical** fashion.
- If the layout has a vertical orientation new *rows* are placed one on top of the other.
- A horizontal layout uses a side-by-side *column* placement policy.



The Linear Layout

1. LinearLayout: Setting Attributes

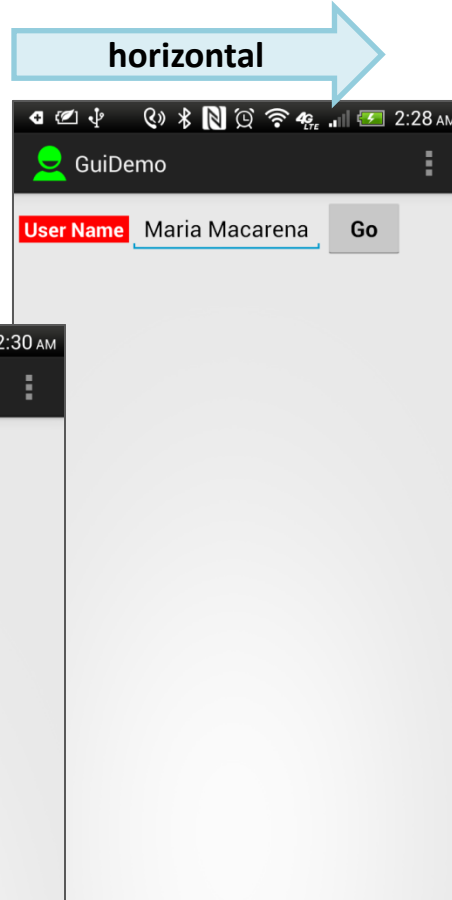
Configuring a **LinearLayout** usually requires you to set the following attributes:

- orientation (*vertical, horizontal*)
- fill model (*match_parent, wrap_contents*)
- weight (*0, 1, 2, ...n*)
- gravity (*top, bottom, center,...*)
- padding (*dp – dev. independent pixels*)
- margin (*dp – dev. independent pixels*)

The LinearLayout - Orientation

1.1 Attribute: Orientation

The **android:orientation** property can be set to: **horizontal** for columns, or **vertical** for rows. Use *setOrientation()* for runtime changes.



```
<LinearLayout
xmlns:android="http://schemas.android.com/ap
k/res/android"
    android:id="@+id/myLinearLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal"
    android:padding="4dp" >

    <TextView
        android:id="@+id/LabelUserName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="#ffff0000"
        android:text=" User Name "
        android:textColor="#ffffff"
        android:textSize="16sp"
        android:textStyle="bold" />

    <EditText
        android:id="@+id/ediName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Maria Macarena"
        android:textSize="18sp" />

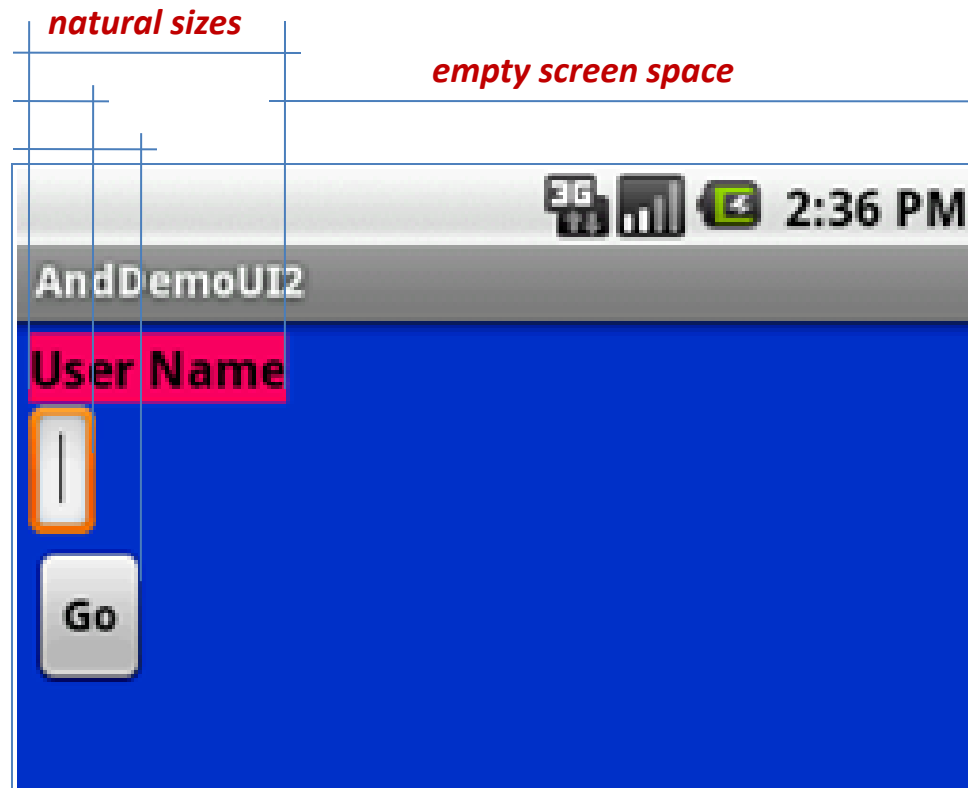
    <Button
        android:id="@+id/btnGo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Go"
        android:textStyle="bold" />

</LinearLayout>
```

The LinearLayout – Fill Model

1.2 Fill Model

- Widgets have a "natural size" based on their included text (rubber band effect).
- On occasions you may want your widget to have a specific space allocation (height, width) even if no text is initially provided (as is the case of the empty text box shown below).



The LinearLayout – Fill Model

1.2 Fill Model

All widgets inside a LinearLayout **must** include ‘width’ and ‘height’ attributes.

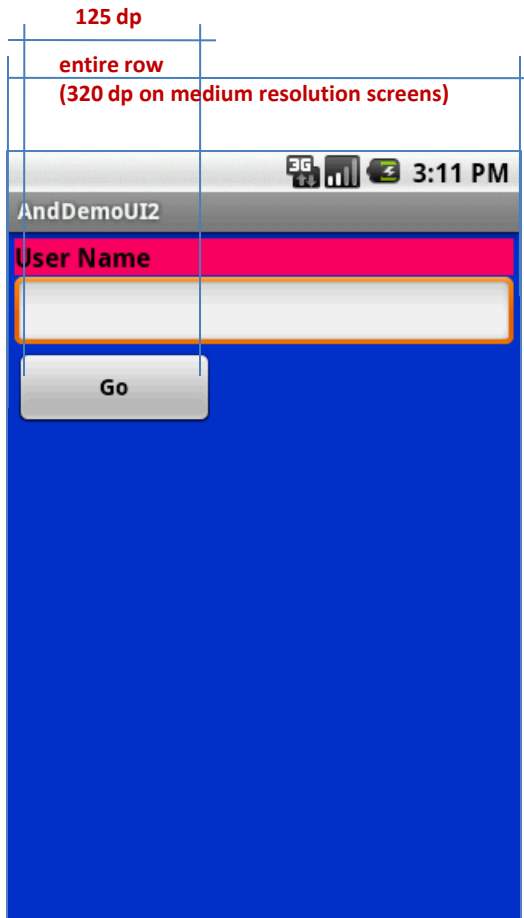
```
android:layout_width  
android:layout_height
```

Values used in defining height and width can be:

1. A specific dimension such as **125dp** (device independent pixels, a.k.a. **dip**)
2. **wrap_content** indicates the widget should just fill up its natural space.
3. **match_parent** (previously called ‘**fill_parent**’) indicates the widget wants to be as big as the enclosing parent.

The LinearLayout – Fill Model

1.2 Fill Model



Medium resolution is: 320 x 480 dpi.

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

Row-wise

```
<TextView
```

```
  android:id="@+id/labelUserName"
  android:layout_width="_parent"
  android:layout_height="wrap_content"
  android:background="#ffff0066"
  android:text="User Name"
  android:textColor="#ff000000"
  android:textSize="16sp"
  android:textStyle="bold" />
```

Use all the row

```
<EditText
```

```
  android:id="@+id/ediName"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:textSize="18sp" />
```

```
<Button
```

```
  android:id="@+id/btnGo"
  android:layout_width="125dp"
  android:layout_height="wrap_content"
  android:text="Go"
  android:textStyle="bold" />
```

Specific size: 125dp

```
</LinearLayout>
```

The LinearLayout – Weight

1.2 Weight

Indicates how much of the extra space in the LinearLayout will be allocated to the view. Use **0** if the view should not be stretched. The bigger the weight the larger the extra space given to that widget.

Example

The XML specification for this window is similar to the previous example.

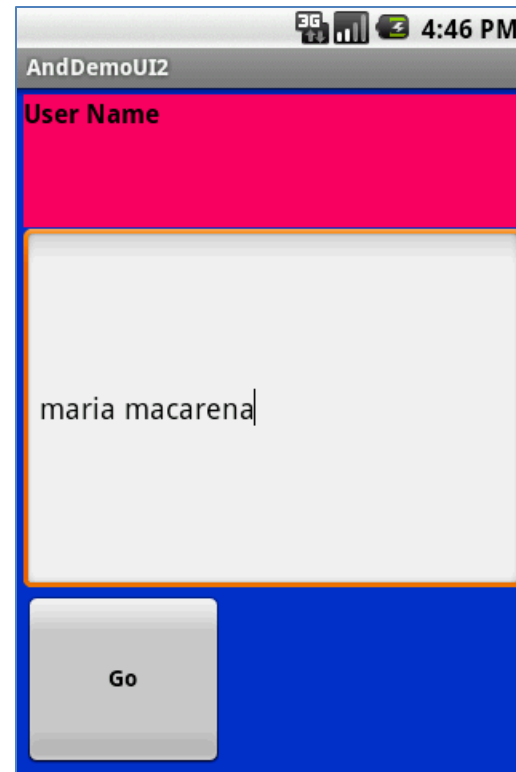
The TextView and Button controls have the additional property

```
android:layout_weight="1"
```

whereas the EditText control has

```
android:layout_weight="2"
```

Default value is 0

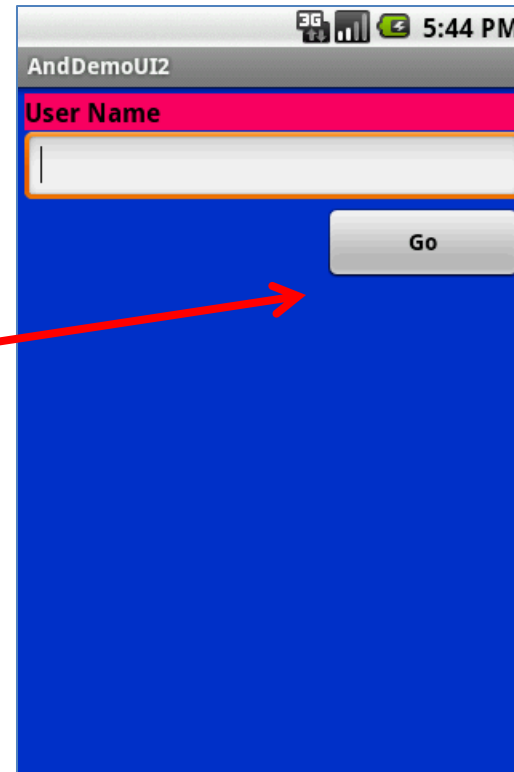
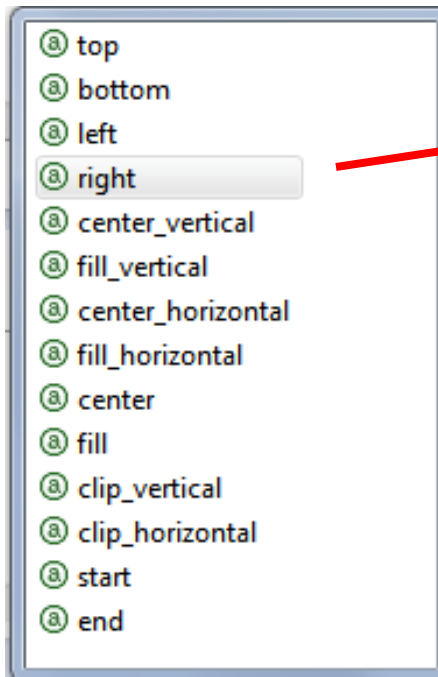


**Takes: 2 / (1+1+2)
of the screen space**

The LinearLayout – Gravity

1.3 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.



Button has
right
layout_gravity

The LinearLayout – Gravity



1.3 CAUTION: gravity vs. layout_gravity

The difference between:

android:gravity

indicates how to place an object within a container. In the example the text is centered

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



android:layout_gravity

positions the view with respect to its

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



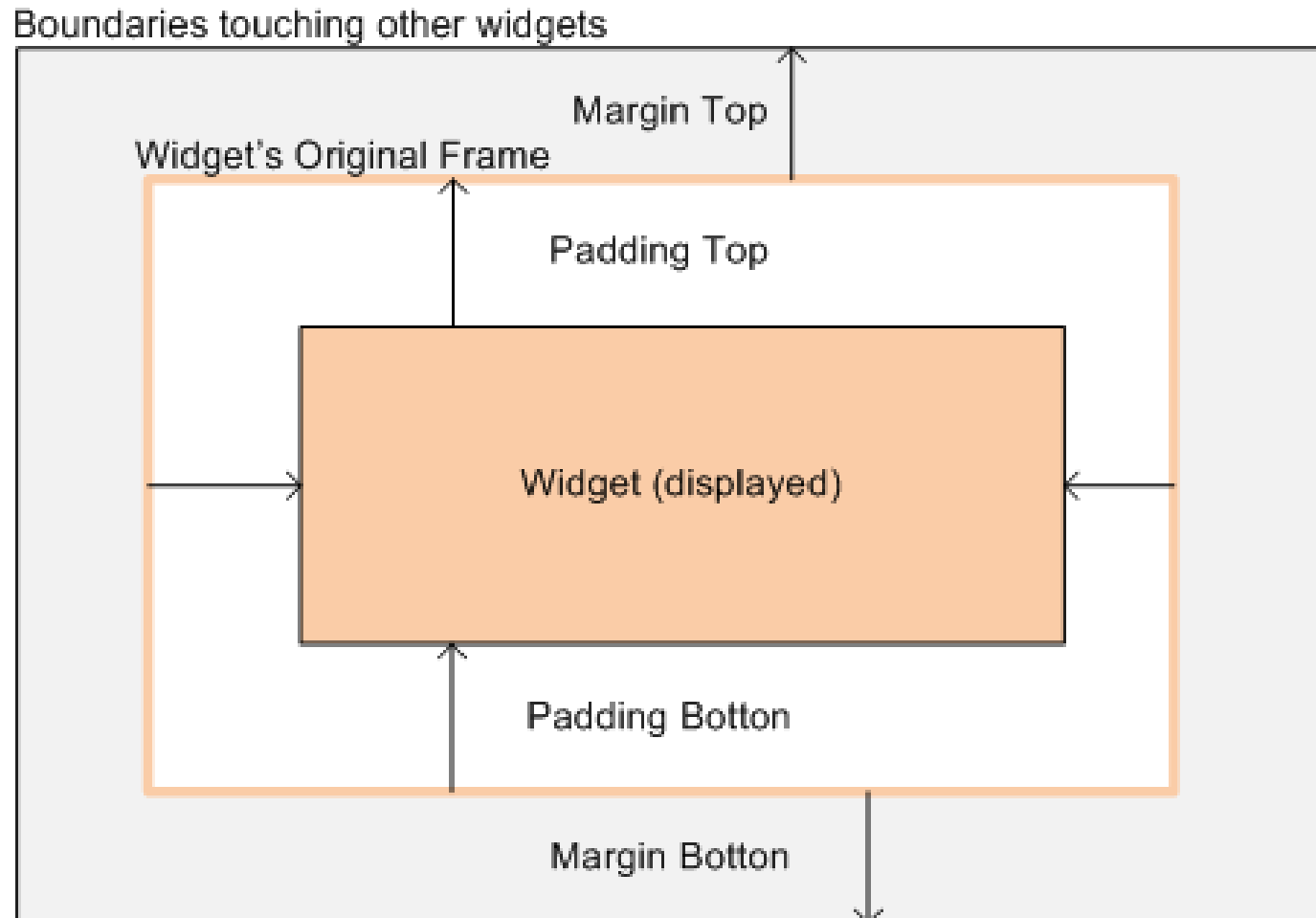
The LinearLayout – Padding

1.4 Padding

- The **padding** attribute specifies the widget's internal margin (in **dp** units).
- The internal margin is the extra space between the borders of the widget's "cell" and the actual widget contents.
- Either use
 - `android:padding` property
 - or call method `setPadding()` at runtime.

The LinearLayout – Padding

1.3 Padding and Marging

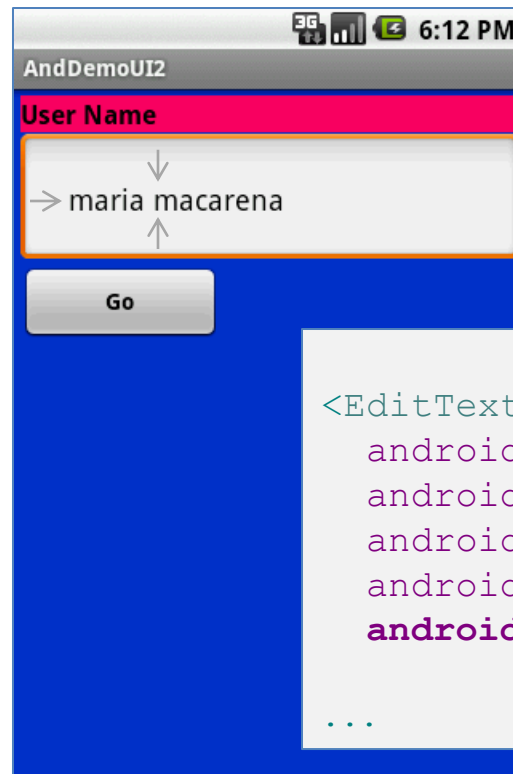


The LinearLayout – Padding

1.3 Internal Margins Using Padding

Example:

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



The LinearLayout – Margin

1.4 (External) Margin

- Widgets –by default– are tightly packed next to each other.
- To increase space between them use the **android:layout_margin** attribute

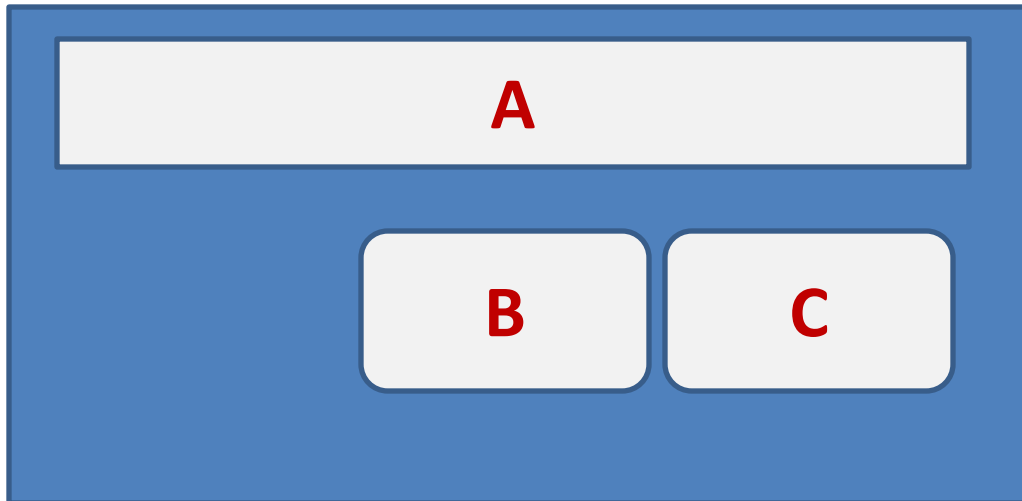
```
<EditText
  android:id="@+id/ediName"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:textSize="18sp"

  android:layout_margin="6dp"
>
</EditText>
...
```

The Relative Layout

2. Relative Layout

The placement of widgets in a **RelativeLayout** is based on their *positional relationship* to other widgets in the container and the parent container.



Example:

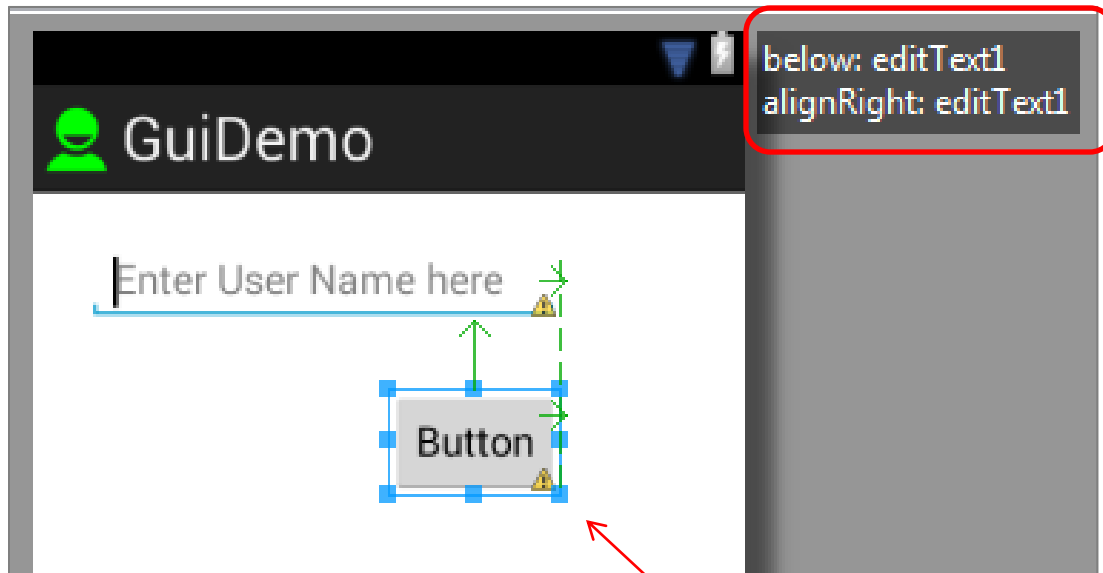
A is by the parent's top

C is below A, to its right

B is below A, to the left of C

The Relative Layout

2. Example: Relative Layout



Location of the button is expressed in reference to its relative position with respect to the EditText box.

The Relative Layout

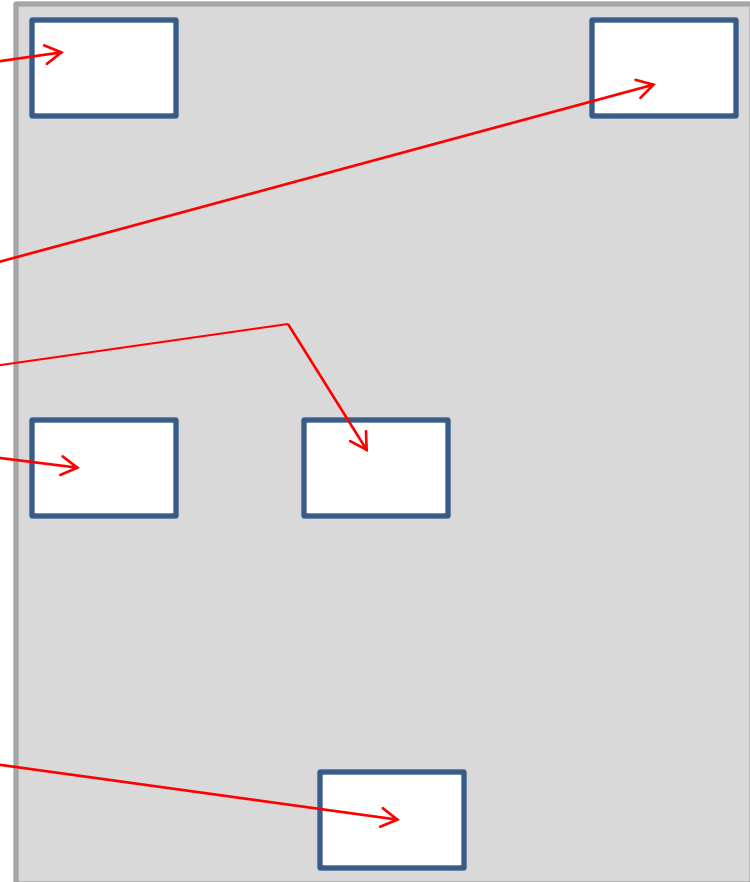
2. Referring to the container

Below there is a list of some positioning XML **boolean** properties (=“true/false”) useful for collocating a widget based on the location of its **parent** container.

`android:layout_alignParentTop`
`android:layout_alignParentBottom`

`android:layout_alignParentLeft`
`android:layout_alignParentRight`

`android:layout_centerInParent`
`android:layout_centerVertical`
`android:layout_centerHorizontal`



The Relative Layout

2. Referring to other widgets

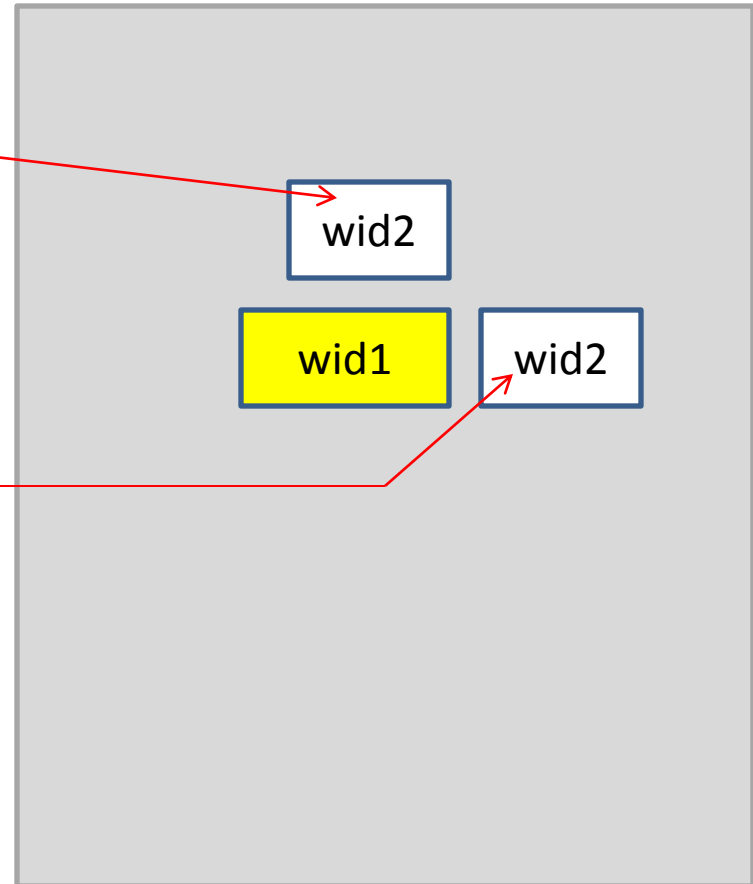
The following properties manage the positioning of a widget **respect to other widgets**:

```
android:layout_above="@+id/wid1"
```

```
android:layout_below
```

```
android:layout_toLeftOf
```

```
android:layout_toRightOf
```



In this example widget “wid2” is map relative to wid1 (known as “@+id/wid1”)

The Relative Layout

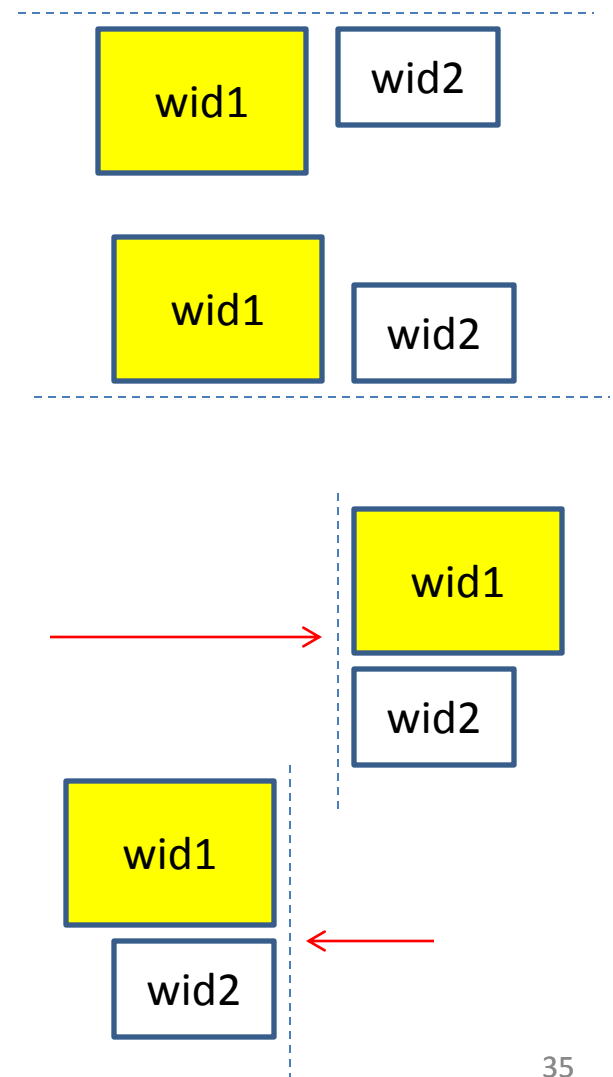
2. Referring to other widgets – cont.

```
android:layout_alignTop="@+id/wid1"
```

```
android:layout_alignBottom="@+id/wid1"
```

```
android:layout_alignLeft="@+id/wid1"
```

```
android:layout_alignRight="@+id/wid1"
```



The Relative Layout

2. Referring to other widgets

When using relative positioning you need to:

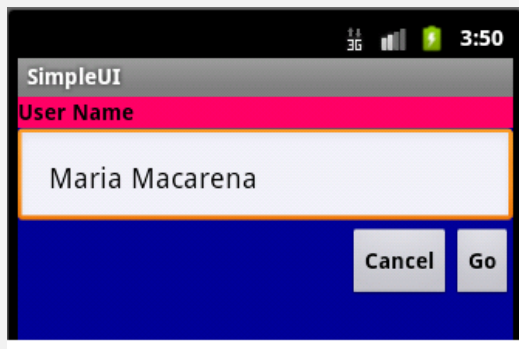
1. Use identifiers (`android:id` attributes) on *all elements* that you will be referring to.
2. XML elements are named using the prefix: `@+id/...` For instance an EditText box could be called: `android:id="@+id/txtUserName"`
3. You must refer only to widgets that have been already defined. For instance a new control to be positioned below the `txtUserName` EditText box could refer to it using: `android:layout_below="@+id/txtUserName"`

The Relative Layout

2. Example

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/myRelativeLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#ff000099" >

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



```
<EditText
    android:id="@+id/txtUserName"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_below="@+id/lblUserName"
    android:padding="20dp" >
</EditText>

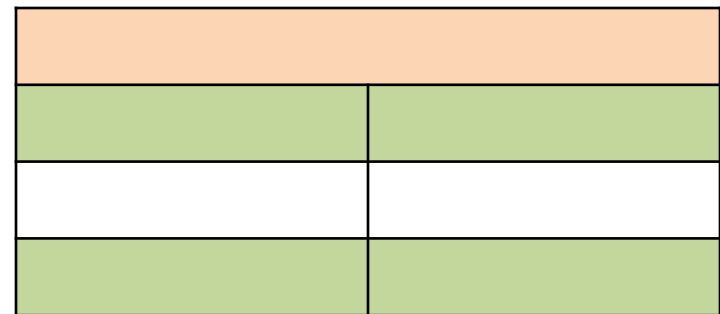
<Button
    android:id="@+id/btnGo"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignRight="@+id/txtUserName"
    android:layout_below="@+id/txtUserName"
    android:text="Go"
    android:textStyle="bold" >
</Button>

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

The Table Layout

3. Table Layout

1. Android's **TableLayout** uses a grid to position your widgets.
2. Like in a matrix, cells in the grid are identifiable by *rows* and *columns*.
3. Columns are flexible, they could *shrink* or *stretch* to accommodate their contents.
4. The element **TableRow** is used to define a new row in which widgets can be allocated.
5. The number of columns in a TableRow is determined by the total of side-by-side widgets placed on the row.



Orange	
Green	Green
White	White
Green	Green

Basic XML Layouts - Containers

3. Table Layout – Setting Number of Columns

The number of columns in a row is determined by Android.

Example: If your TableLayout 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 – Stretching a Column

- A single widget in a `TableLayout` can occupy more than one column.
- The `android:layout_span` property indicates the number of columns the widget is allowed to expand.

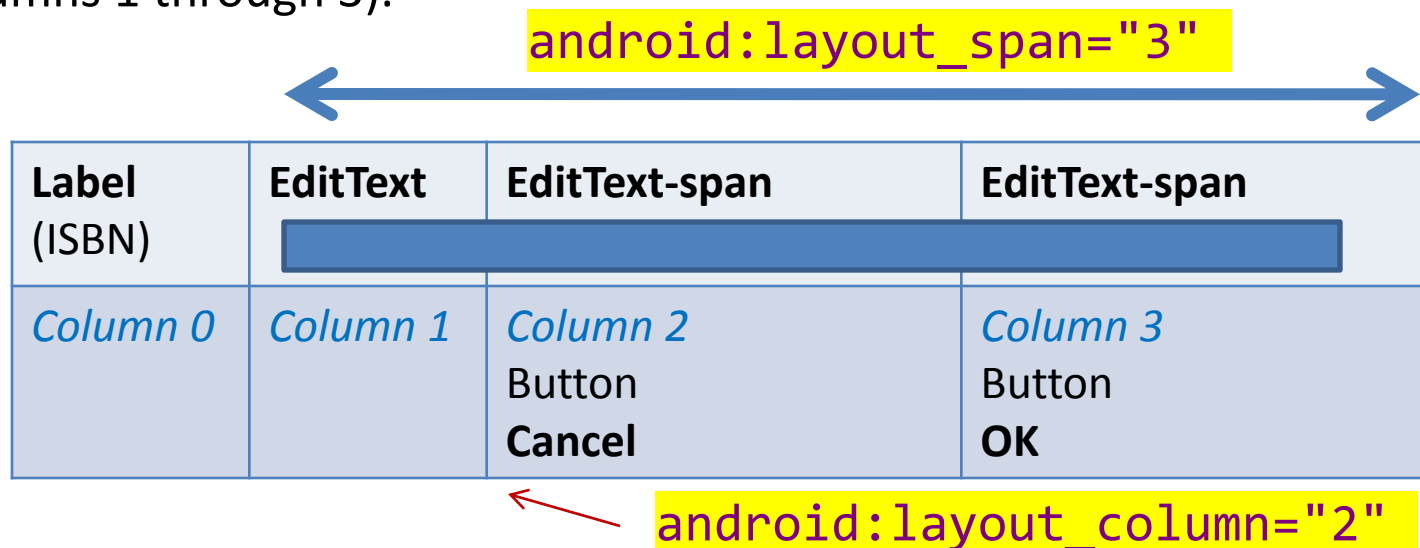
```
<TableRow>
  <TextView android:text="URL:" />
  <EditText
    android:id="@+id/entry"
    android:layout_span="3" />
</TableRow>
```


Basic XML Layouts - Containers

3. Table Layout – Stretching a Column

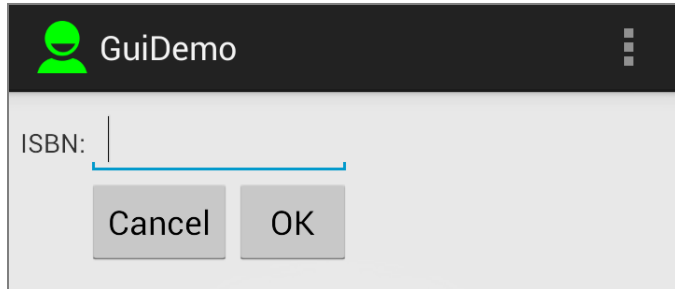
Widgets on a table's row are placed from left to right, beginning with the first available column. Each column in the table stretches to accommodate its hosted widgets.

Example: The table shown below has four columns (*indices: 0,1,2,3*). The label ("ISBN") goes in the first column (*index 0*). The EditText box uses the `layout_span` attribute to be placed into a spanned set of three columns (columns 1 through 3).



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
xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/myTableLayout"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:padding="6dp"
  android:orientation="vertical" >
  <TableRow>
    <TextView android:text="ISBN:" />
    <EditText
      android:id="@+id/ediISBN"
      android:layout_span="3" />
  </TableRow>

  <TableRow>
    <Button
      android:id="@+id/cancel"
      android:layout_column="2"
      android:text="Cancel" />
    <Button
      android:id="@+id/ok"
      android:text="OK" />
  </TableRow>
</TableLayout>
```

Occupy 3
columns

Skip
columns 0, 1

Basic XML Layouts - Containers

3. Stretching the Entire Table

- By default, a column is as wide as the “natural” size of the widest widget collocated in this column (e.g. a column holding a button showing the caption “Go” is narrower than other column holding a button with the caption “Cancel”).
- A table does not necessarily take all the horizontal space available.
- If you want the table to (horizontally) match its container use the property:

```
android:stretchColumns="column(s)"
```

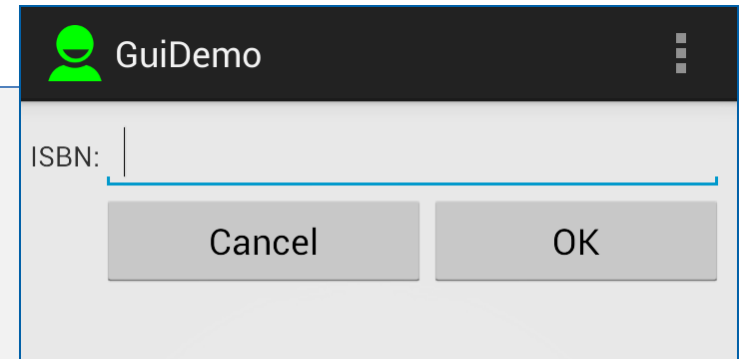
Its value is the column-index (or comma-separated column indices) to be stretched to take up any space still available on the row.

Basic XML Layouts - Containers

3. Example: Stretching the Entire Table

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="match_parent"  
    android:layout_height="match_parent"  
    android:orientation="vertical"  
    android:stretchColumns="2,3"  
    xmlns:android="http://schemas.android.com/apk/res/android"  
>  
...
```

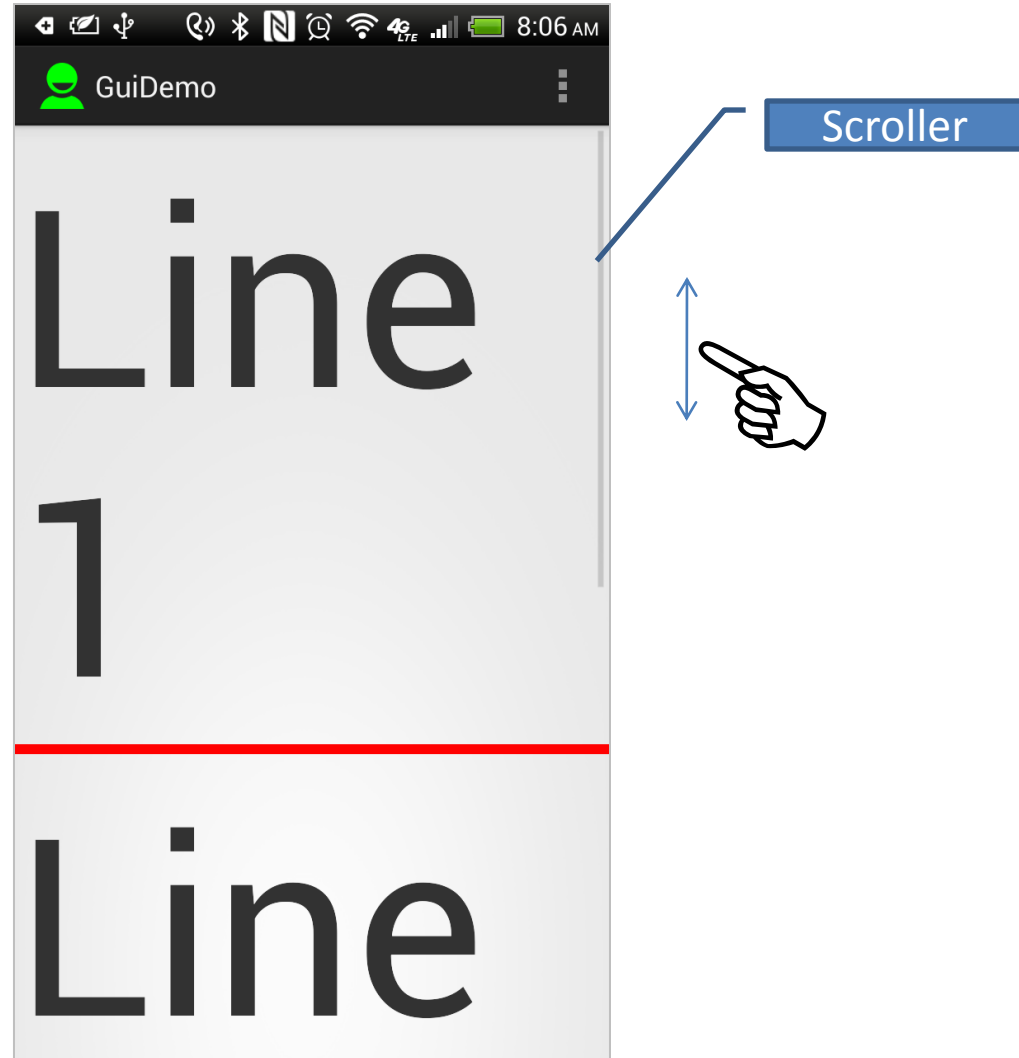


TODO: try to stretch one column at the time 1, then 2, and so on.

Basic XML Layouts - Containers

4. ScrollView Layout

- The **ScrollView** control is useful in situations in which we have *more data to show* than what a single screen could display.
- ScrollViews provide a sliding access to the data.
- Only a portion of the user's data can be seen at one time, however the rest is available via scrolling.



Basic XML Layouts - Containers

4. Example: ScrollView Layout

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

<ScrollView
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/myScroLLView1"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <LinearLayout
        android:id="@+id/myLinearLayoutVertical"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical" >

        <TextView
            android:id="@+id/textView1"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Line1"
            android:textSize="150dp" />

        <View
            android:layout_width="match_parent"
            android:layout_height="6dp"
            android:background="#ffff0000" />

        <TextView
            android:id="@+id/textView2"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Line2"
            android:textSize="150dp" />

        <View
            android:layout_width="match_parent"
            android:layout_height="6dp"
            android:background="#ffff0000" />

        <TextView
            android:id="@+id/textView3"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Line3"
            android:textSize="150dp" />

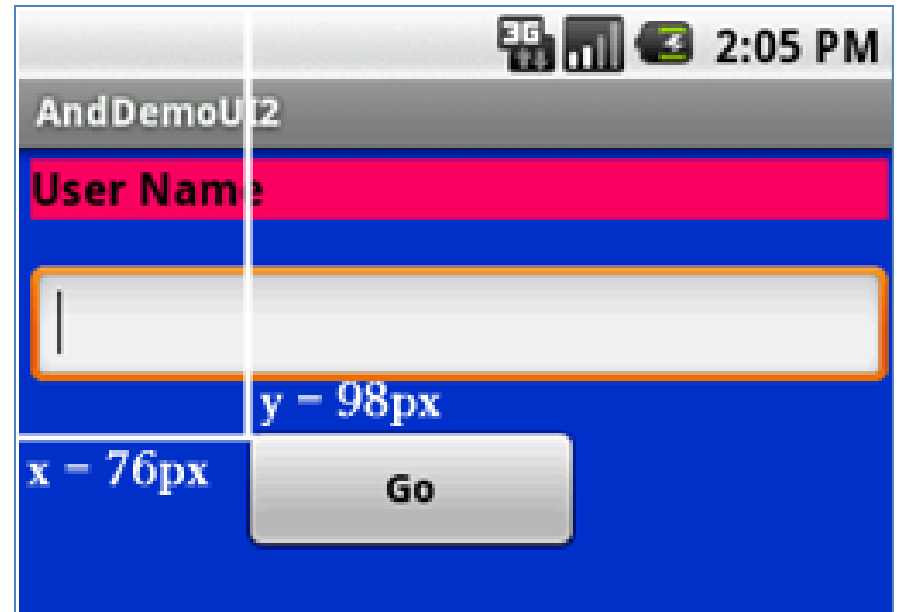
    </LinearLayout>

</ScrollView>
```

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="match_parent"
  android:layout_height="match_parent"
  android:background="#ff0033cc"
  android:padding="4dp"
  xmlns:android="http://schemas.android.com/apk/res/android"
  >
  <TextView
    android:id="@+id/tvUserNm"
    android:layout_width="match_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="0dp"
    android:layout_y="10dp"
  >
  </TextView>
  <EditText
    android:id="@+id/etName"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textSize="18sp"
    android:layout_x="0dp"
    android:layout_y="38dp"
  >
  </EditText>
  <Button
    android:layout_width="120dp"
    android:text="Go"
    android:layout_height="wrap_content"
    android:textStyle="bold"
    android:id="@+id/btnGo"
    android:layout_x="100dp"
    android:layout_y="170dp" />
</AbsoluteLayout>
```



Attaching Layouts to Java Code

PLUMBING. You must 'connect' the XML elements with equivalent objects in your Java activity. This allows you to manipulate the UI with code.

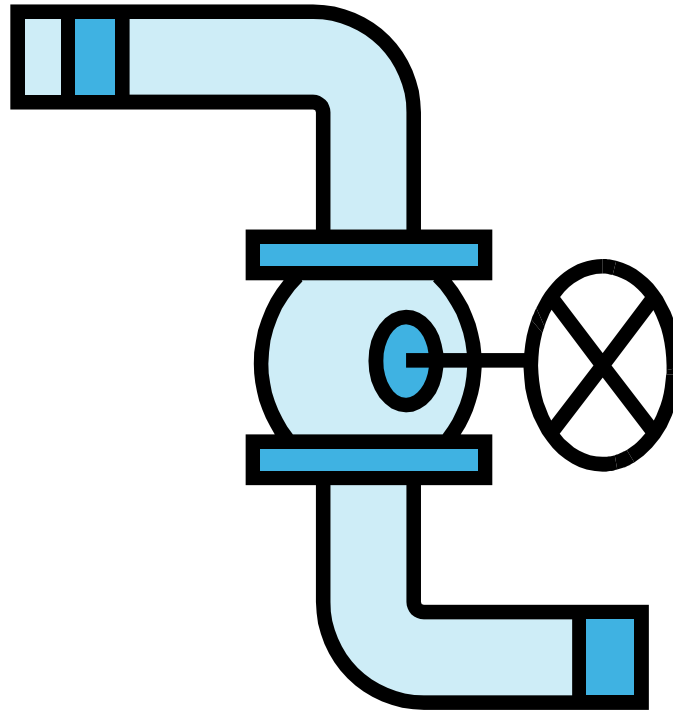
XML Layout

```
<xml....
```

```
...
```

```
...
```

```
</xml>
```



JAVA code

```
public class ....
```

```
{
```

```
...
```

```
...
```

```
}
```

Attaching Layouts to Java Code

Assume the UI in *res/layout/main.xml* has been created. This layout could be called by an application using the statement

```
setContentView(R.layout.main);
```

Individual widgets, such as *myButton* could be accessed by the application using the statement `findViewById(...)` as in

```
Button btn= (Button) findViewById(R.id.myButton);
```

Where **R** is a class automatically generated to keep track of resources available to the application. In particular **R.id...** is the collection of widgets defined in the XML layout.

Attaching Layouts to Java Code

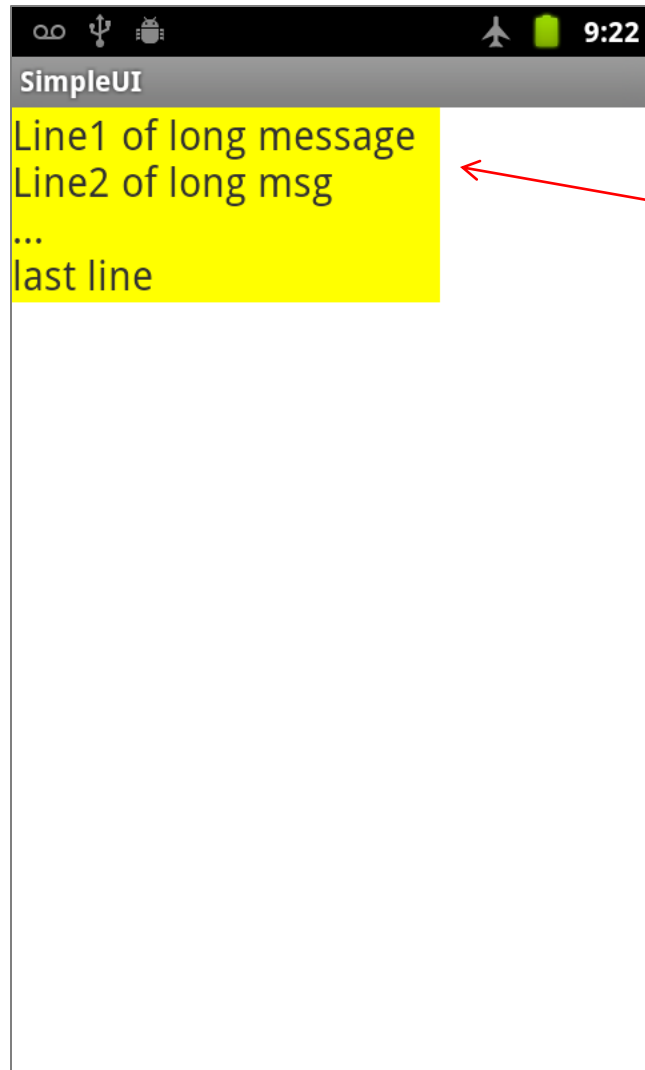
Attaching Listeners to the Widgets

The button of our example could now be used, for instance a listener for the click event could be written as:

```
btn.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        updateTime();
    }
});

private void updateTime() {
    btn.setText(new Date().toString());
}
```

Basic Widgets: Labels



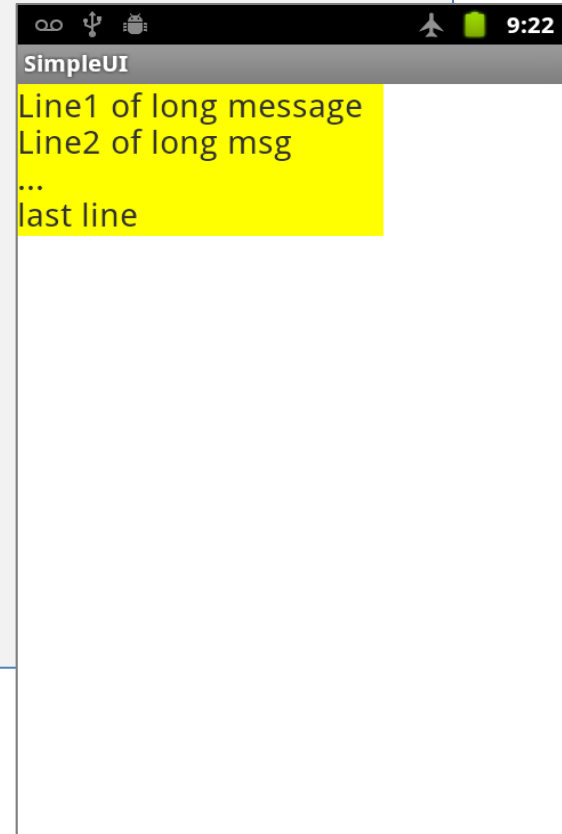
- A **label** is called in android a **TextView**.
- TextViews are typically used for output to display a caption.
- TextViews are *not* editable, therefore they take no input.

Basic Widgets: Labels

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/widget32"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/txt1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="#ffffff00"
        android:inputType="none"
        android:text="@string/Long_msg_1"
        android:textSize="20sp" />

</LinearLayout>
```



Hint on Better Programming Style:

Add to the [res/values/stringg.xml](#) the entry

```
<string name="long_msg_1">Line1 of long message\nLine2 of long msg\n...\nlast line</string>
```

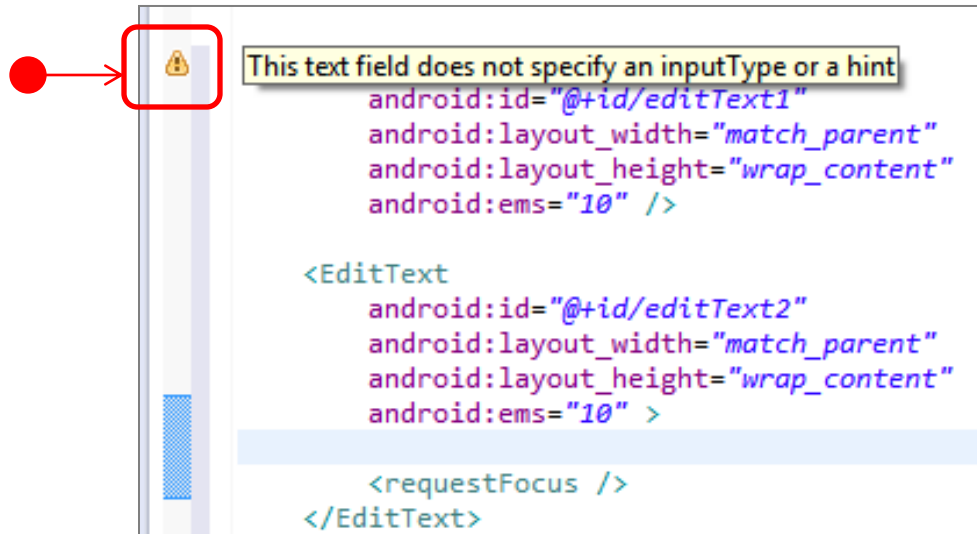
EditText Caution

WARNING



This text field does not specify an InputType or a hint

is just a warning requesting your help to improve the working of a TextView. Add the clause **android:hint="...some hint here..."** and/or **android:InputType="...choice..."** where choices are

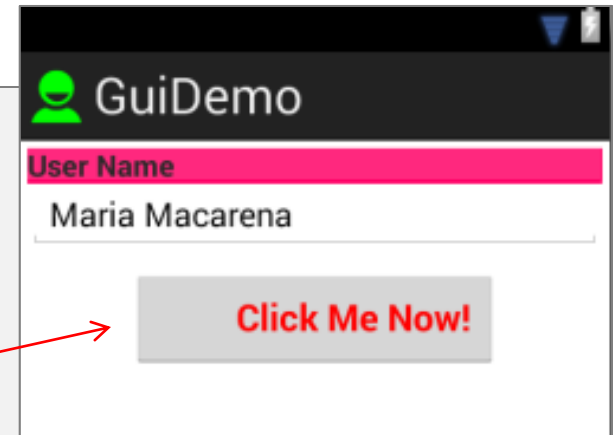


Basic Widgets: Buttons

- A **Button** widget allows the simulation of a clicking action on a GUI.
- **Button** is a subclass of **TextView**. Therefore formatting a button's face is similar to the setting of a **TextView**.

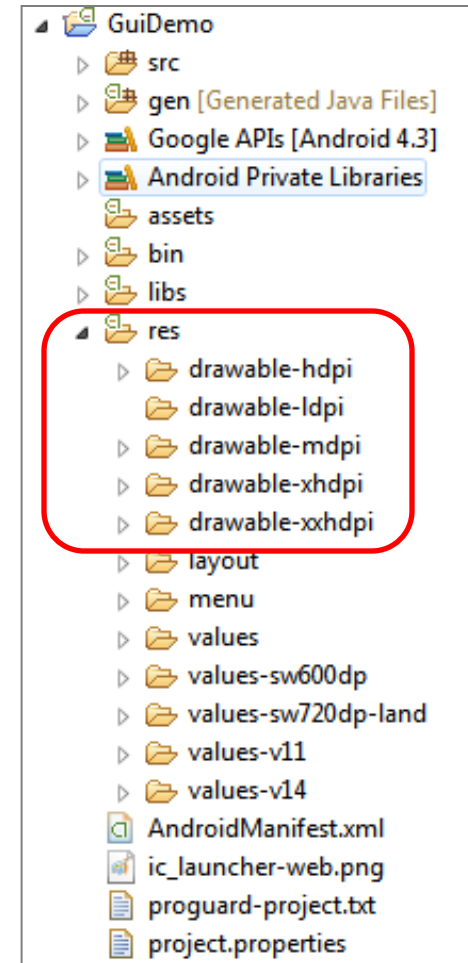
```
<Button
```

```
    android:id="@+id/button1"  
    android:layout_width="300dp"  
    android:layout_height="wrap_content"  
    android:layout_gravity="center"  
    android:layout_marginTop="5dp"  
    android:gravity="right"  
    android:padding="5dp"  
    android:text="@string/button1_caption"  
    android:textColor="#ffff0000"  
    android:textSize="20sp"  
    android:textStyle="bold" />
```



Basic Widgets: Images

- **ImageView** and **ImageButton** are two Android widgets that allow embedding of images in your applications.
- Analogue to *TextView* and *Button* controls (respectively).
- Each widget takes an **android:src** or **android:background** attribute (in an XML layout) to specify what picture to use.
- Pictures are usually stored in the **res/drawable** folder (optionally a low, medium, and high definition version of the same image could be stored to later be used with different types of screens)



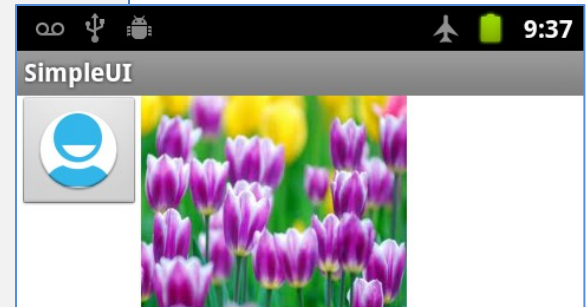
Basic Widgets: Images

```
<LinearLayout
  . . .

  <ImageButton
    android:id="@+id/myImageBtn1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/ic_launcher" >
  </ImageButton>

  <ImageView
    android:id="@+id/myImageView1"
    android:layout_width="150dp"
    android:layout_height="120dp"
    android:scaleType="fitXY"
    android:src="@drawable/flower1" >
  </ImageView>

</LinearLayout>
```



This is a jpg,
gif, png,... file

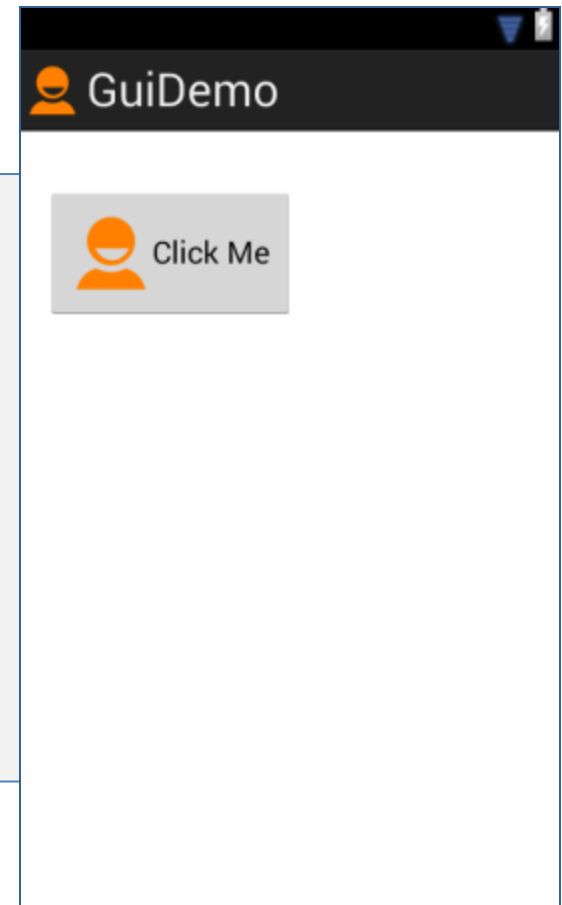
Basic Widgets: Combining Images & Text

A common **Button** could display text and a simple image as shown below

```
<LinearLayout
    . . .

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:drawableLeft="@drawable/ic_happy_face"
        android:gravity="left|center_vertical"
        android:padding="15dp"
        android:text="@string/click_me" />

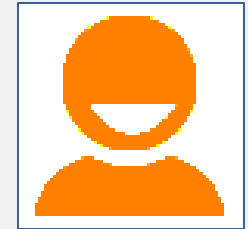
</LinearLayout>
```



Basic Widgets: Images

Icons are small images used to graphically represent your application and/or parts of it. They may appear in different places of the device including:

- Home screen
- Launcher window.
- Options menu
- Action Bar
- Status bar
- Multi-tab interface.
- Pop-up dialog boxes
- List view



Detailed information at:

http://developer.android.com/guide/practices/ui_guidelines/icon_design.html

HINT

Several websites allow you to convert your pictures to image files under a variety of formats & sizes (.png, .jpg, .gif, etc). For instance try:

<http://www.prodraw.net/favicon/index.php>

<http://converticon.com/>

Basic Widgets: EditText

- The **EditText** (or *textBox*) widget is an extension of *TextView* that allows user's input.
- The control can display *editable* text (uses HTML-styles: bold, ...).
- Important Java methods are:

```
textBox.setText("someValue")
```

and

```
textBox.getText().toString()
```



Basic Widgets: EditText

- The **EditText** (or *textBox*) widget is an extension of *TextView* that allows user's input.



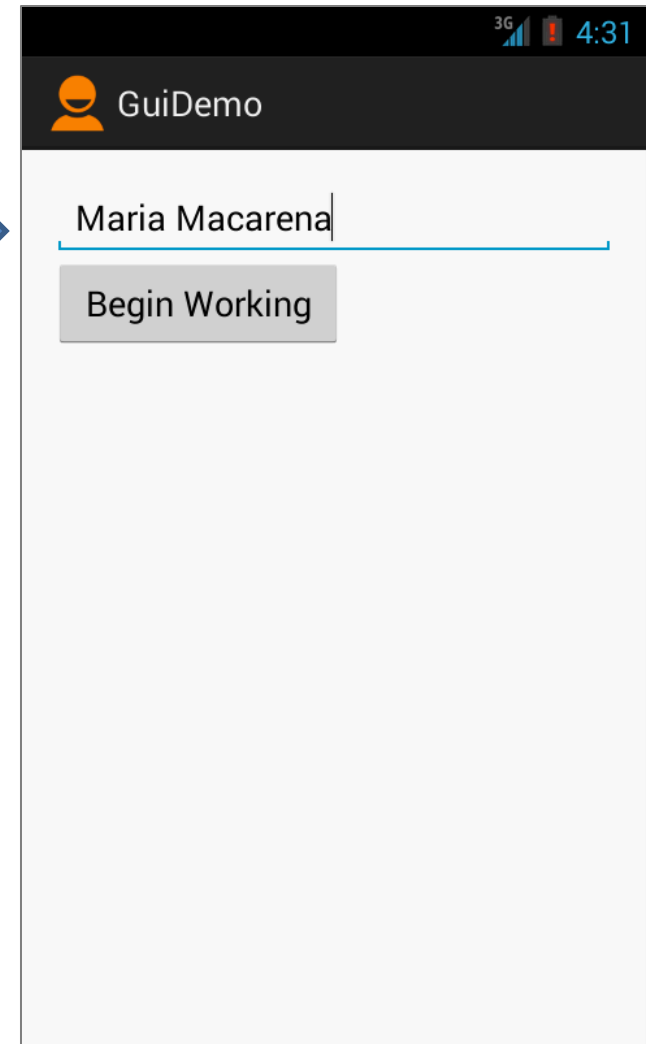
- Important Java I/O methods are:

```
textBox.setText("someValue")
```

and

```
textBox.getText().toString()
```

- The control can display *editable* or *HTML-formatted* text by means of `Html.fromHtml(text)`



Basic Widgets: EditText

CAUTION: Deprecated Methods

DEPRECATED

- `android:autoText`
- `android:capitalize`
- `android:digits`
- `android:singleLine`
- `android:password`
- `android:numeric`
- `android:phoneNumber`

Instead use the newer attribute:



`android:inputType="...choices..."`

where choices include

- Ⓐ "none"
- Ⓐ "text"
- Ⓐ "textCapCharacters"
- Ⓐ "textCapWords"
- Ⓐ "textCapSentences"
- Ⓐ "textAutoCorrect"
- Ⓐ "textAutoComplete"
- Ⓐ "textMultiLine"
- Ⓐ "textTimeMultiLine"
- Ⓐ "textNoSuggestions"
- Ⓐ "textUri"
- Ⓐ "textEmailAddress"
- Ⓐ "textEmailSubject"
- Ⓐ "textShortMessage"
- Ⓐ "textLongMessage"
- Ⓐ "textPersonName"
- Ⓐ "textPostalAddress"
- Ⓐ "textPassword"
- Ⓐ "textVisiblePassword"
- Ⓐ "textWebEditText"
- Ⓐ "textFilter"
- Ⓐ "textPhonetic"
- Ⓐ "number"
- Ⓐ "numberSigned"
- Ⓐ "numberDecimal"
- Ⓐ "phone"
- Ⓐ "datetime"
- Ⓐ "date"
- Ⓐ "time"

Basic Widgets: EditTexts

Example

...

```
<EditText
  android:id="@+id/txtUserName"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"

  android:inputType="textCapWords|textAutoCorrect"

  android:hint="@string/enter_your_first_and_last_name"

  android:textSize="18sp" />
```

...

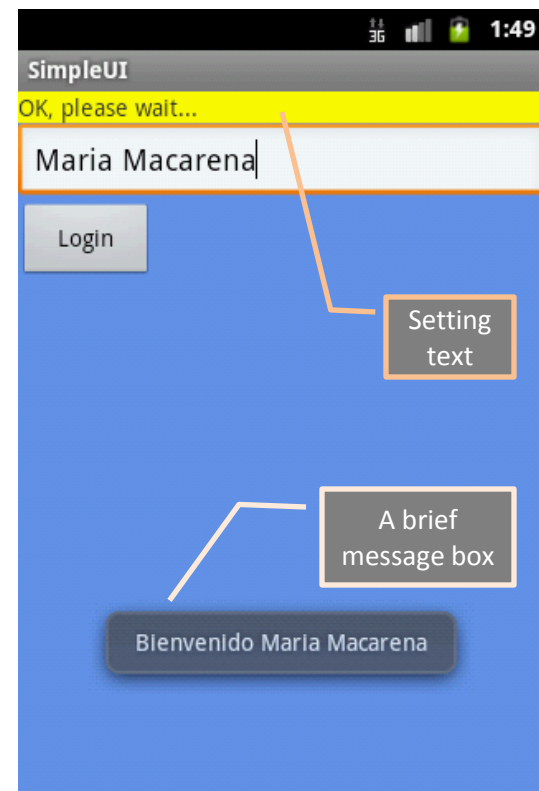
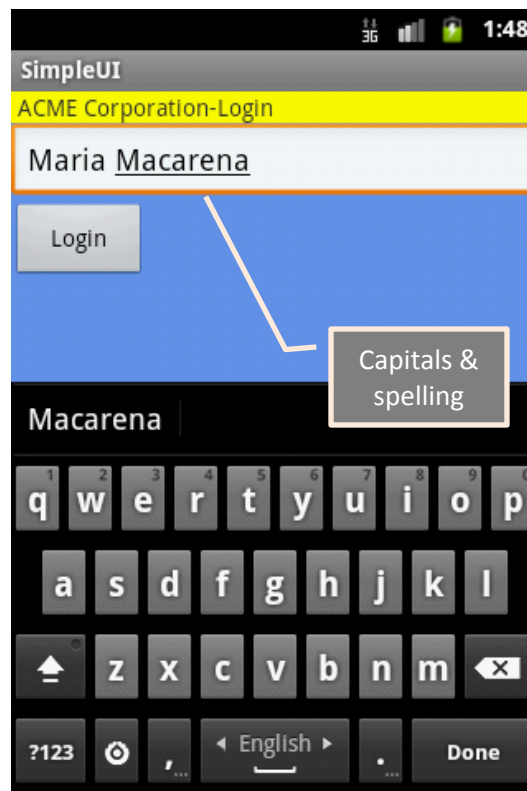
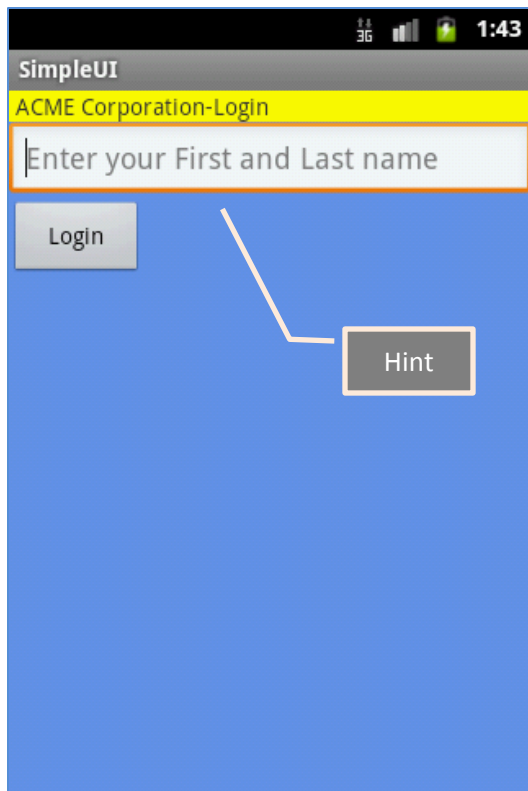
Enter "teh" It will be changed to: "the"

Each word is capitalized

Suggestion (grey out)

Example 1: Login Screen

In this example we will create and use a simple login screen holding a label(**TextView**), a textBox (**EditText**), and a **Button**. A fragment of its functionality is shown below.



Example 1: Login Screen

Layout Design 1 of 2

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#886495ed"
    android:orientation="vertical"
    android:padding="2dp" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="1dp"
        android:background="#ffffff00"
        android:text="@string/ACME_Corp_Caption" />

    <EditText
        android:id="@+id/txtUserName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="1dp"
        android:hint="@string/Enter_your_First_and_Last_name"
        android:inputType="textCapWords|textAutoCorrect"
        android:textSize="18sp" >

        <requestFocus />
    </EditText>
```

Example 1: Login Screen

Layout Design 2 of 2

```
<Button
    android:id="@+id/button1"
    android:layout_width="82dp"
    android:layout_height="wrap_content"
    android:layout_marginTop="1dp"
    android:text="@string/Login" />

</LinearLayout>
```

Resource Captions: res/values/strings

```
<?xml version="1.0" encoding="utf-8"?>
<!-- this is the res/values/strings.xml file -->
<resources>

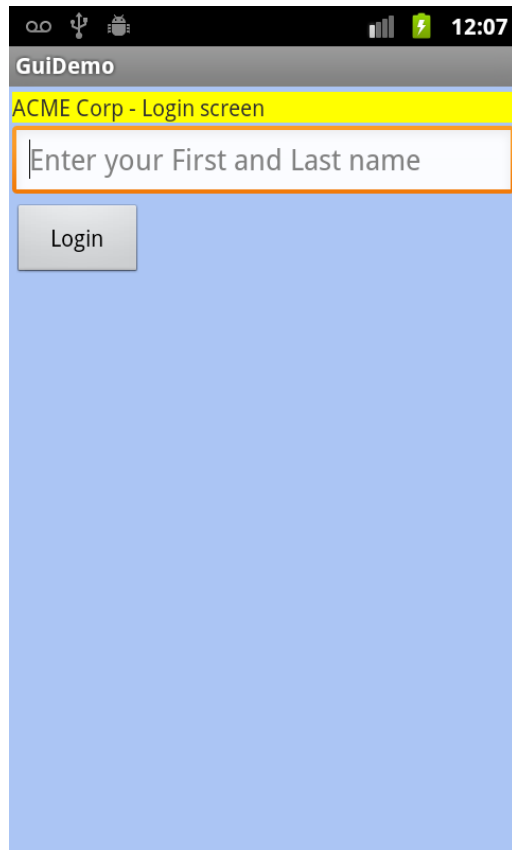
    <string name="app_name">GuiDemo</string>
    <string name="action_settings">Settings</string>
    <string name="Login">Login</string>
    <string name="ACME_Corp_Caption">Login</string>
    <string name="Enter_your_First_and_Last_name">Enter your First and Last name</string>

</resources>
```

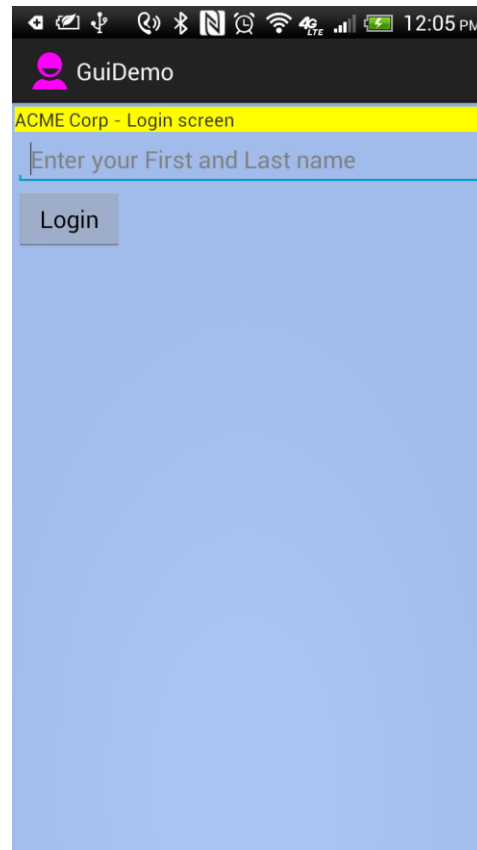
Example 1: Login Screen

Rendering the Layout

The images below show the previously defined login screen displayed by two different devices running SDK2.3 (Gingerbread) and SDK4.3 (Ice Cream)



GingerBread SDK



Ice Cream SDK

Example 1: Login Screen

MainActivity.java Class (1 of 2)

```
package csu.matos.guidemo;
import ...
// "LOGIN" - a gentle introduction to UI controls

public class MainActivity extends Activity {

    //class variables representing UI controls to be controlled from the program
    TextView labelUserName;
    EditText txtUserName;
    Button btnBegin;

    //variables used with the Toast message class
    private Context context;
    private int duration = Toast.LENGTH_SHORT;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //show the login screen
        setContentView(R.layout.activity_main);
        context = getApplicationContext();
    }
}
```

Example 1: Login Screen

MainActivity.java Class (2 of 2)

```
//binding the UI's controls defined in "main.xml" to Java code
labelUserName = (TextView) findViewById(R.id.textView1);
txtUserName = (EditText) findViewById(R.id.txtUserName);
btnBegin = (Button) findViewById(R.id.button1);

//LISTENER: allowing the button widget to react to user interaction
btnBegin.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        String userName = txtUserName.getText().toString();
        if (userName.compareTo("Maria Macarena")==0){
            labelUserName.setText("OK, please wait...");
            Toast.makeText(context,
                "Bienvenido " + userName,
                duration).show();
        }
        Toast.makeText(context,
            userName + " is not a valid USER" ,
            duration).show();
    }
}); // onClick

} // onCreate

} // class
```

Your turn!

Implement any/all of the following projects
Using simple text boxes (EditText, TextView)
and buttons:

1. Currency calculator
2. Tip Calculator
3. Simple Flashlight



Example 2: Wiring Multiple Button Widgets

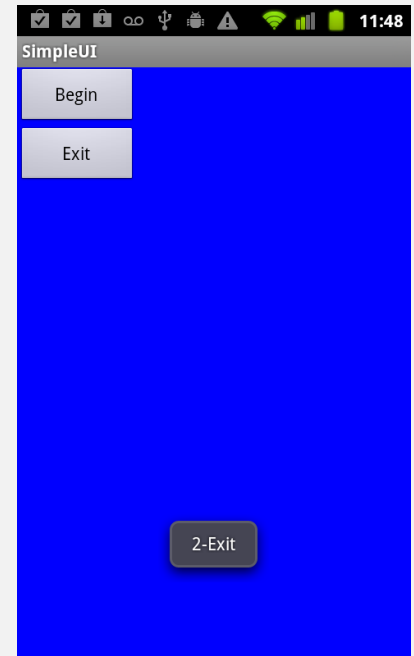
Note: The example below shows an alternative way of defining a single Listener for multiple buttons.

```
public class SimpleUI extends Activity implements OnClickListener {
    Button btnBegin;
    Button btnExit;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

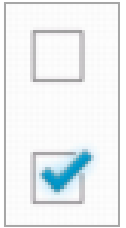
        btnBegin = (Button) findViewById(R.id.btnBegin);
        btnExit = (Button) findViewById(R.id.btnExit);

        btnBegin.setOnClickListener(this);
        btnExit.setOnClickListener(this);
    } //onCreate

    @Override
    public void onClick(View v) {
        if (v.getId() == btnBegin.getId()) {
            Toast.makeText(getApplicationContext(), "1-Begin", 1).show();
        }
        if (v.getId() == btnExit.getId()) {
            Toast.makeText(getApplicationContext(), "2-Exit", 1).show();
        }
    } //onClick
} //class
```



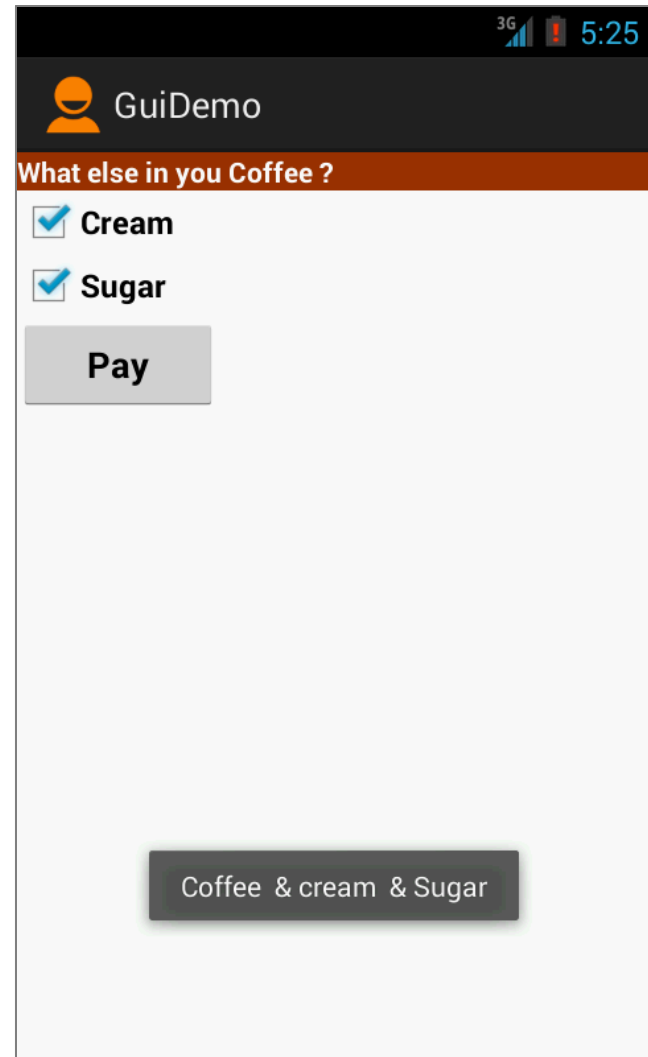
Basic Widgets: CheckBox



A checkbox is a special two-states button that can be either *checked* or *unchecked*.

The screen displays two CheckBox controls for selecting 'Cream' and 'Sugar' options. In this image both boxes are 'checked'.

When the user pushes the 'Pay' button a Toast-message is issue telling what is the current combination of choices held by the checkboxes.



Example 3: CheckBox



The following Coffee-App shows us how to use CheckBoxes.

Layout 1 of 2

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="5dp"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/labelCoffee"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="#ff993300"
        android:text="@string/coffee_addons"
        android:textColor="@android:color/white"
        android:textStyle="bold" />

    <CheckBox
        android:id="@+id/chkCream"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/cream"
        android:textStyle="bold" />
```

Example 3: CheckBox



Coffee-App

Layout 2 of 2

```
<CheckBox
    android:id="@+id/chkSugar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/sugar"
    android:textStyle="bold" />

<Button
    android:id="@+id/btnPay"
    android:layout_width="153dp"
    android:layout_height="wrap_content"
    android:text="@string/pay"
    android:textStyle="bold" />

</LinearLayout>
```

Example 3: CheckBox



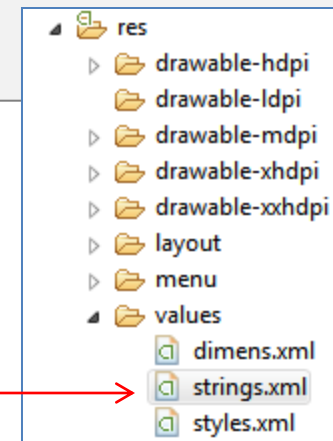
Coffee-App

Resources: res/values/strings

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

  <string name="app_name">GuiDemo</string>
  <string name="action_settings">Settings</string>

  <string name="click_me">Click Me</string>
  <string name="sugar">Sugar</string>
  <string name="cream">Cream</string>
  <string name="coffee_addons">What else do you like in your coffee?</string>
  <string name="pay">Pay</string>
</resources>
```



Example 2: CheckBox



Java Code – 1 of 2

```
public class MainActivity extends Activity {
    CheckBox chkCream;
    CheckBox chkSugar;
    Button btnPay;

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

        //binding XML controls with Java code
        chkCream = (CheckBox)findViewById(R.id.chkCream);
        chkSugar = (CheckBox)findViewById(R.id.chkSugar);
        btnPay = (Button) findViewById(R.id.btnPay);
    }
}
```

Example 2: CheckBox



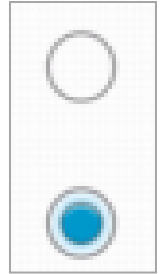
Complete code for the checkBox demo (3 of 3)

```
//LISTENER: wiring button-events-&-code
    btnPay.setOnClickListener(new OnClickListener() {

@Override
public void onClick(View v) {
    String msg = "Coffee ";
    if (chkCream.isChecked()) {
        msg += " & cream ";
    }
    if (chkSugar.isChecked()){
        msg += " & Sugar";
    }
    Toast.makeText(getApplicationContext(),
        msg, Toast.LENGTH_SHORT).show();
    //go now and compute cost...

    }//onClick
    });
} //onCreate
} //class
```

Basic Widgets: RadioButtons

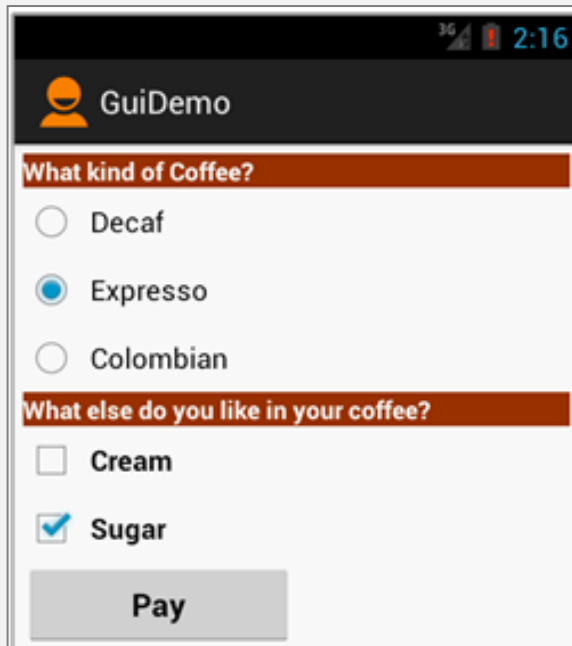


- A radio button is a two-states button that can be either *checked* or *unchecked*.
- When the radio button is unchecked, the user can press or click it to check it.
- Radio buttons are normally used together in a **RadioGroup**.
- When several radio buttons live inside a radio group, checking one radio button *unchecks* all the others.
- RadioButton inherits from ... TextView. Hence, all the standard TextView properties for *font face, style, color*, etc. are available for controlling the look of radio buttons.
- Similarly, you can call *isChecked()* on a RadioButton to see if it is selected, *toggle()* to select it, and so on, like you can with a CheckBox.

Example 4: RadioButtons

We extend the previous example by adding a *RadioGroup* and three *RadioButtons*. Only new XML and Java code is shown:

```
<TextView
    android:id="@+id/textView1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="#ff993300"
    android:text="@string/kind_of_coffee"
    android:textColor="#ffffff"
    android:textStyle="bold" />
```



```
<RadioGroup
    android:id="@+id/radioGroupCoffeeType"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" >



    <RadioButton
        android:id="@+id/radDecaf"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/decaf" />

    <RadioButton
        android:id="@+id/radEspresso"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/espresso" />

    <RadioButton
        android:id="@+id/radColombian"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:checked="true"
        android:text="@string/colombian" />

</RadioGroup>
```

Example 4: RadioButtons

```
public class MainActivity extends Activity {  
    CheckBox chkCream;   
    CheckBox chkSugar;  
    Button btnPay;  
  
    RadioGroup radCoffeeType;   
    RadioButton radDecaf;  
    RadioButton radEspresso;  
    RadioButton radColombian;  
  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
        chkCream = (CheckBox) findViewById(R.id.chkCream);  
        chkSugar = (CheckBox) findViewById(R.id.chkSugar);  
        btnPay = (Button) findViewById(R.id.btnPay);  
        radCoffeeType = (RadioGroup) findViewById(R.id.radioGroupCoffeeType);  
        radDecaf = (RadioButton) findViewById(R.id.radDecaf);  
        radEspresso = (RadioButton) findViewById(R.id.radEspresso);  
        radColombian = (RadioButton) findViewById(R.id.radColombian);  
    }  
}
```


Example 4: RadioButtons

```
// LISTENER: wiring button-events-&-code
btnPay.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        String msg = "Coffee ";
        if (chkCream.isChecked())
            msg += " & cream ";
        if (chkSugar.isChecked())
            msg += " & Sugar";

        // get radio buttons ID number
        int radioId = radCoffeeType.getCheckedRadioButtonId();

        // compare selected's Id with individual RadioButtons ID
        if (radColombian.getId() == radioId)
            msg = "Colombian " + msg;
        // similarly you may use .isChecked() on each RadioButton
        if (radEspresso.isChecked())
            msg = "Espresso " + msg;
        // similarly you may use .isChecked() on each RadioButton
        if (radDecaf.isChecked())
            msg = "Decaf " + msg;

        Toast.makeText(getApplicationContext(), msg, 1).show();
        // go now and compute cost...
    } // onClick
});
} // onCreate

} // class
```

Example 4: RadioButtons

Example

This UI uses
RadioButtons
and
CheckBoxes
to define choices

RadioGroup

Summary of choices

The screenshot shows a mobile application interface titled "GuiDemo". The status bar at the top indicates 3G connectivity, a battery icon, and the time 2:32. The main content area has a dark header with an orange smiley face icon and the text "GuiDemo". Below the header, there are two sections with brown headers: "What kind of Coffee?" and "What else do you like in your coffee?". Under the first section, there are three radio button options: "Decaf", "Espresso" (which is selected), and "Colombian". Under the second section, there are two checkbox options: "Cream" (unchecked) and "Sugar" (checked). Below these options is a grey button labeled "Pay". At the bottom of the screen, there is a dark grey button with white text that reads "Espresso Coffee & Sugar".

Miscellaneous:

UI Attributes & Java Methods

XML Controls the focus sequence:

```
android:visibility  
android:background  
<requestFocus />
```

Java methods

```
myButton.requestFocus()  
myTextBox.isFocused()  
myWidget.setEnabled()  
myWidget.isEnabled()
```

User Interfaces

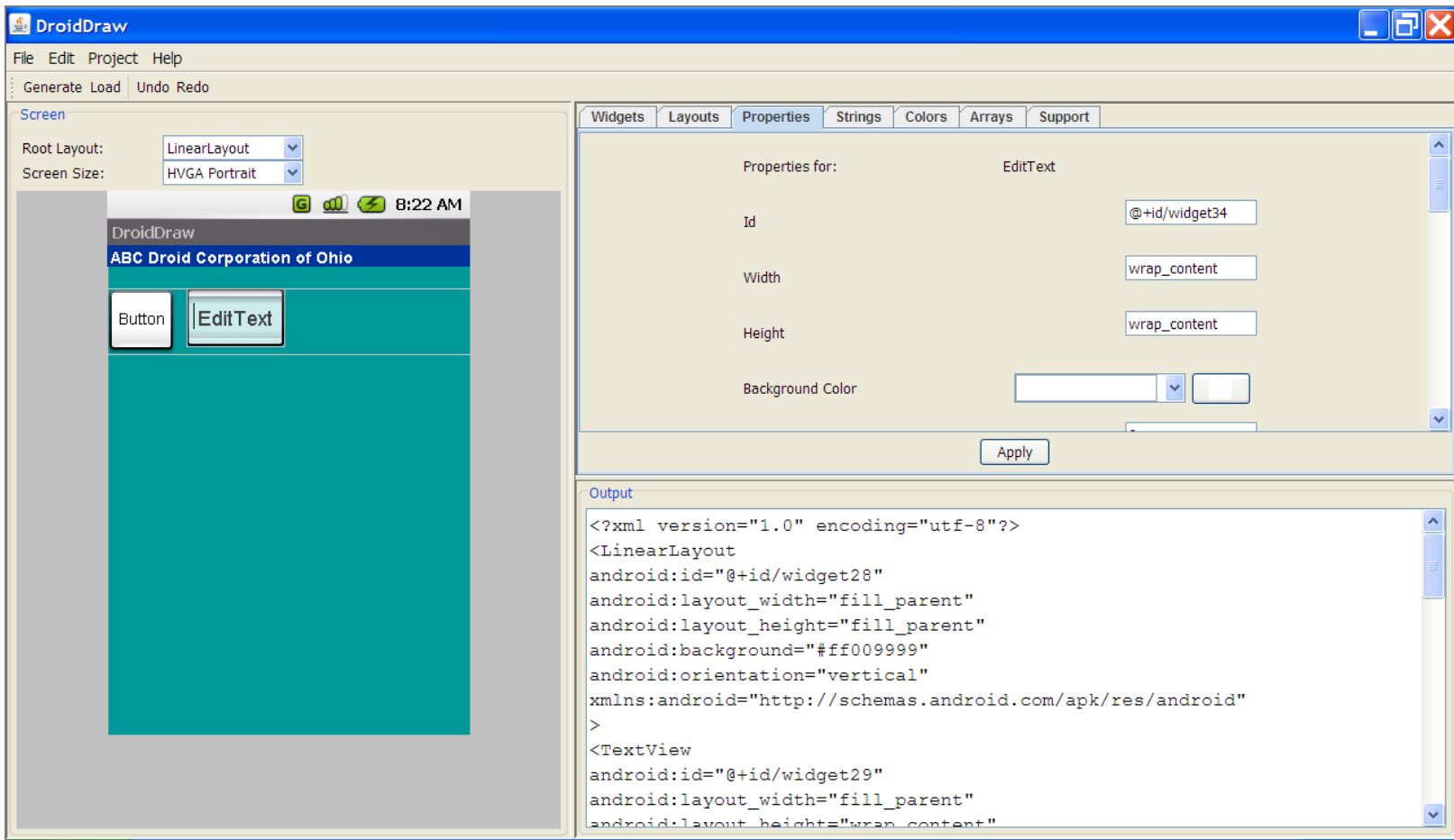


This image was made using the Device Frame Generator, which is part of the Android Asset Studio tool

Appendix A.

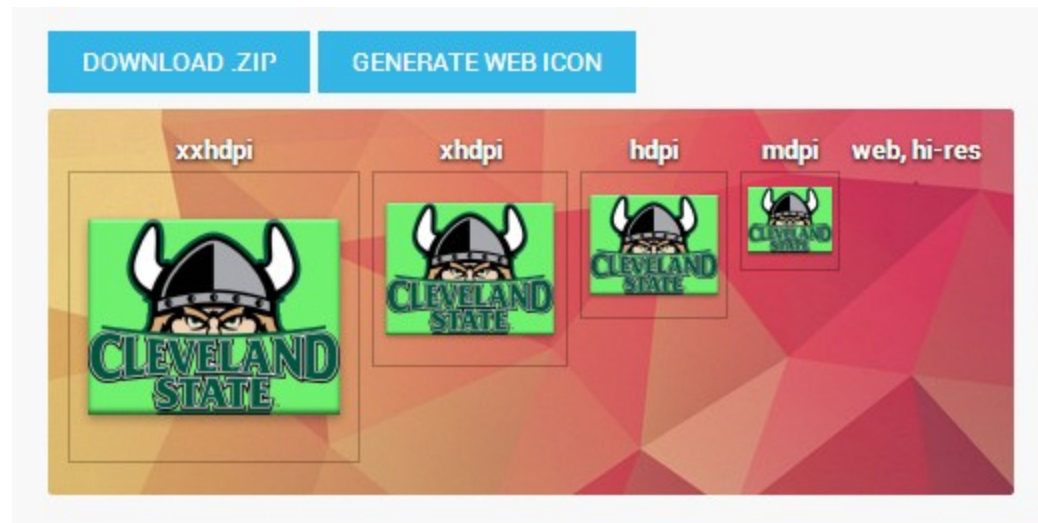
DroidDraw

A simple GUI generator - **LINK:** www.droidDraw.org



Appendix B:

Android Asset Studio

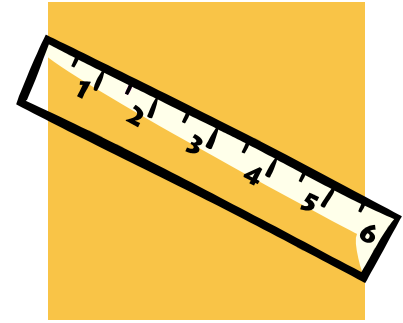


LINK: <http://android-ui-utils.googlecode.com/hg/asset-studio/dist/index.html>

This tool offers to the designers a number of options to craft high-quality apps, among those components it has

Icon Generators	Other Generators	Community Tools
Launcher icons Action bar and tab icons Notification icons Navigation drawer indicator Generic icons	Device frame generator Simple nine-patch gen.	Android Action Bar Style Generator Android Holo Colors Generator

Appendix C: Measuring Graphic Elements



Q. What is **dpi** (also known as **ppi**) ?

Stands for *dots per inch*. It suggests a measure of screen quality. You can compute it using the following formula:

$$dpi = \sqrt{widthPixels^2 + heightPixels^2} / diagonalInches$$

G1 (base device 320x480)	155.92 dpi	(3.7 in diagonally)
Nexus (480x800)	252.15 dpi	
HTC One (1080x1920)	468 dpi	(4.7 in)
Samsung S4 (1080x1920)	441 dpi	(5.5 in)

Q. What is the difference between **dp**, **dip** and **sp** units in Android?

dp (also known as **dip**) *Density-independent Pixels* – is an abstract unit based on the physical density of the screen. These units are relative to a 160 dpi screen, so one dp is one pixel (dp) on a 160 dpi screen. Use it for measuring anything but fonts – DO NOT USE dp, in, mm

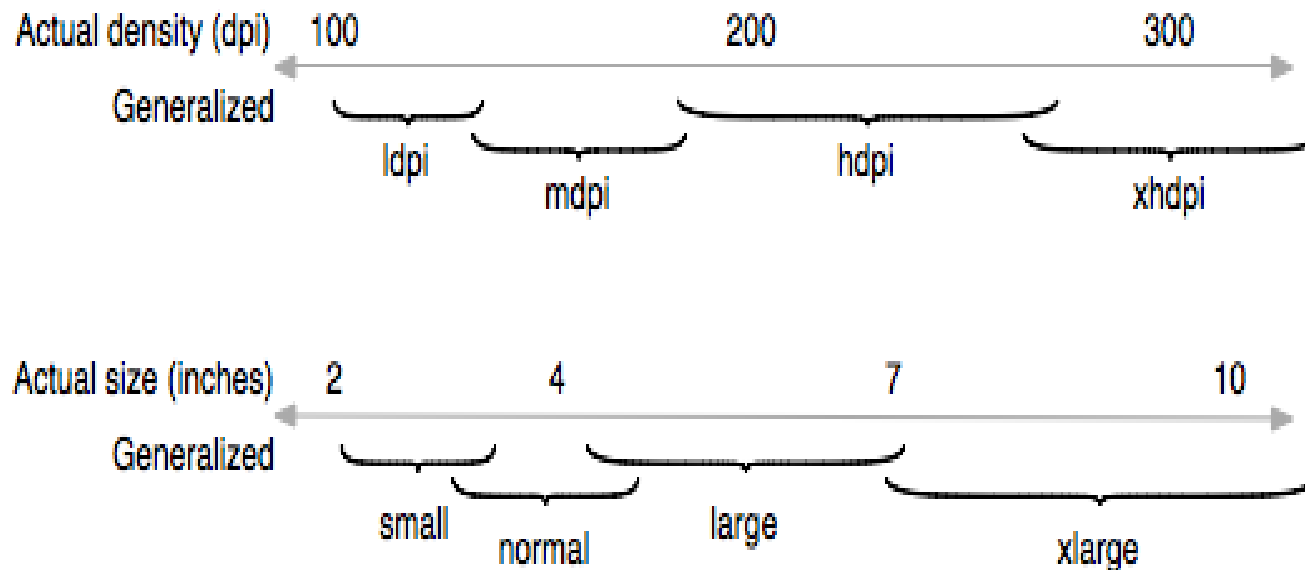
sp

Scale-independent Pixels – similar to the relative density dp unit, but used for font size preference.

Appendix C: Measuring Graphic Elements

Q. How Android deals with screen resolutions?

Illustration of how the Android platform maps actual screen densities and sizes to generalized density and size configurations.



Appendix C: Measuring Graphic Elements

Q. What do I gain by using screen densities?

More homogeneous results as shown below



Examples of density independence on WVGA high density (left), HVGA medium density (center), and QVGA low density (right).

Q. How to set different density/size screens in my application?

The following manifest fragments declares support for small, normal, large, and xlarge screens in any density.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    <supports-screens
        android:anyDensity="true"
        android:largeScreens="true"
        android:normalScreens="true"
        android:smallScreens="true"
        android:xlargeScreens="true" />
    ...
</manifest>
```

Appendix C: Measuring Graphic Elements

Q. Give me an example on how to use dp units.

Assume you design your interface for a G1 phone having 320x480 pixels (Abstracted LCD density is **160** – See your AVD entry the actual pixeling is a: $2 \times 160 \times 3 \times 160$)

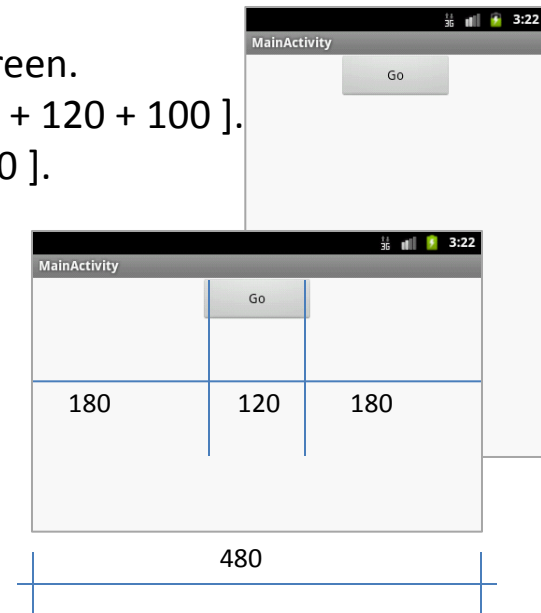
Assume you want a 120dp button to be placed in the middle of the screen.

On portrait mode you could allocate the 320 horizontal pixels as [100 + 120 + 100].

On Landscape mode you could allocate 480 pixels as [180 + 120 + 180].

The XML would be

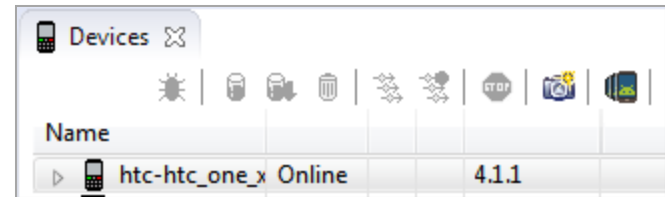
```
<Button
    android:id="@+id/button1"
    android:layout_height="wrap_content"
    android:layout_width="120dp"
    android:layout_gravity="center"
    android:text="@+id/go_caption" />
```



If the application is deployed on devices having a higher resolution the button is still mapped to the middle of the screen.

Appendix D: Hierarchy Viewer Tools

The HierarchyViewer Tool allows exploration of a displayed UI. Use **DDMS** > Click on Devices > Click on HierarchyViewer (next to camera)



The screenshot displays the Hierarchy Viewer tool. The left pane shows a mobile application interface for 'GuiDemo' with a coffee selection screen. The right pane shows the UI hierarchy tree, and the bottom pane shows the Node Detail for the selected LinearLayout.

UI Hierarchy Tree:

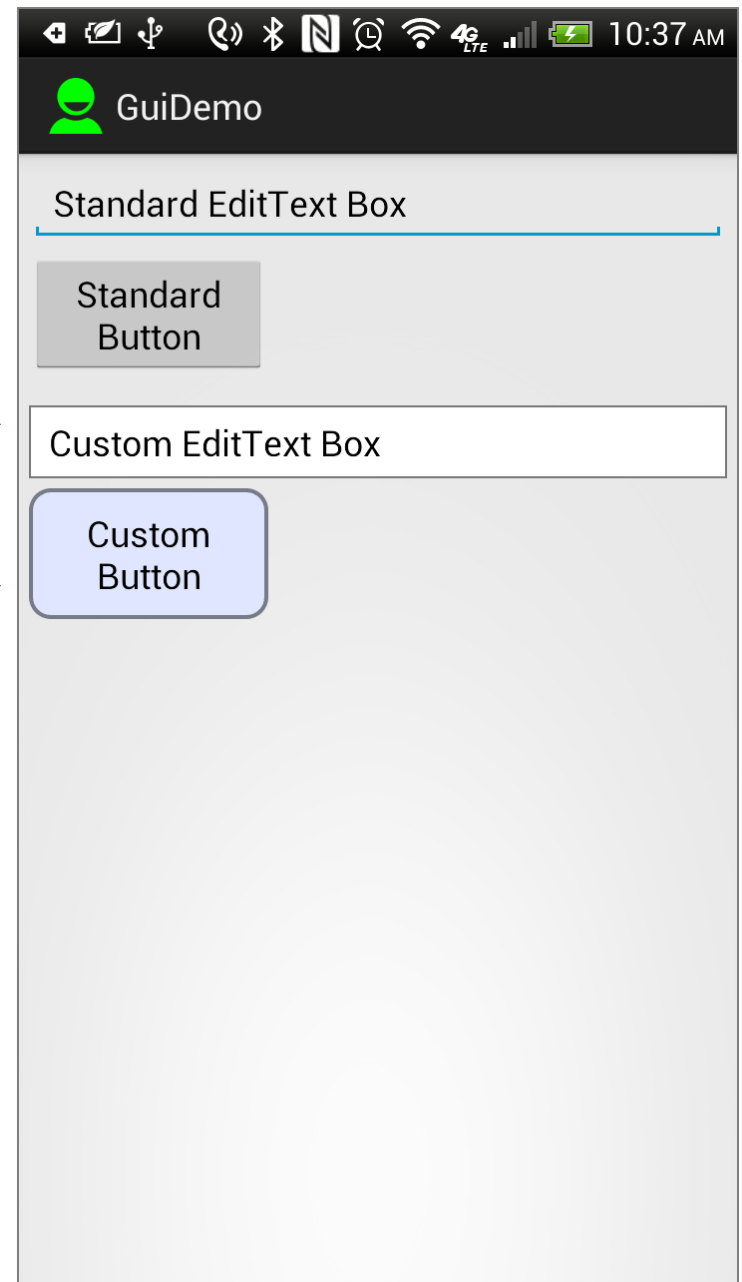
- (0) LinearLayout [97,73][244,122]
 - (0) TextView:GuiDemo [97,73][244,122]
- (1) FrameLayout [0,146][720,1280]
 - (0) LinearLayout [0,146][720,1280]
 - (0) TextView:What kind of Coffee? [10,156][710,194]
 - (1) RadioGroup [10,194][710,425]
 - (0) RadioButton:Decaf [10,194][170,271]
 - (1) RadioButton:Espresso [10,271][218,348]
 - (2) RadioButton:Colombian [10,348][240,425]
 - (2) TextView:What else do you like in your coffee? [10,425]
 - (3) CheckBox:Cream [10,463][179,540]
 - (4) CheckBox:Sugar [10,540][170,617]
 - (5) Button:Pay [10,617][316,713]

Node Detail:

index	0
text	
class	android.widget.LinearLayout
package	csu.matos.guidemo
content-desc	
checkable	false
checked	false
clickable	false
enabled	true
focusable	false
focused	false

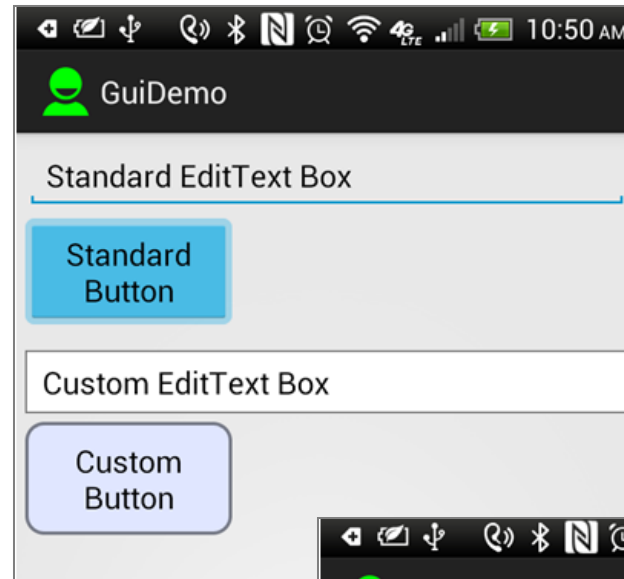
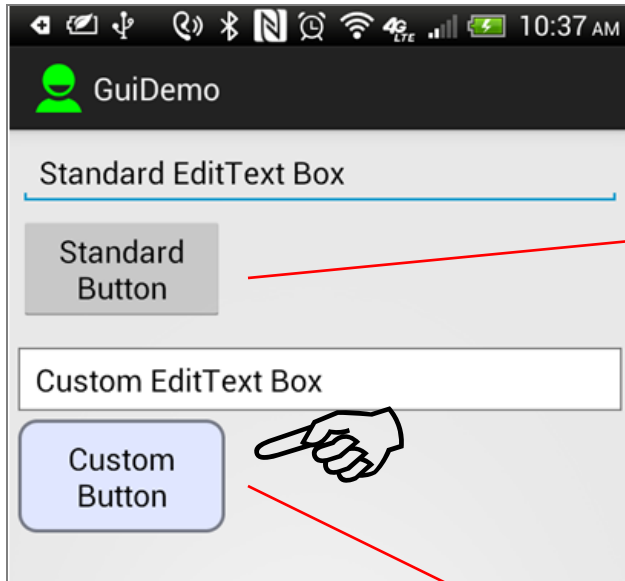
Appendix E: Customizing Widgets

1. The appearance of a widget can be adjusted by the user. For example a button widget could be modified by changing its shape, border, color, margins, etc.
2. Basic shapes include: rectangle, oval, line, and ring.
3. In addition to visual changes, the widget's reaction to user interaction could be adjusted for events such as: Focused, Clicked, etc.
4. The figure shows and EditText and Button widgets as *normally* displayed by a device running SDK4.3 (Ice Cream). The bottom two widgets are custom made versions of those two controls respectively.

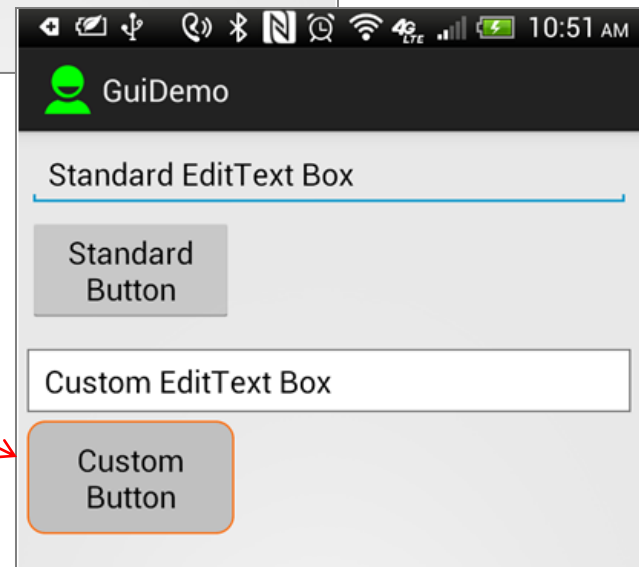


Appendix E: Customizing Widgets

The image shows visual feedback provided to the user during the clicking of a standard and a *custom* Button widget. Assume the device runs under SDK4.3



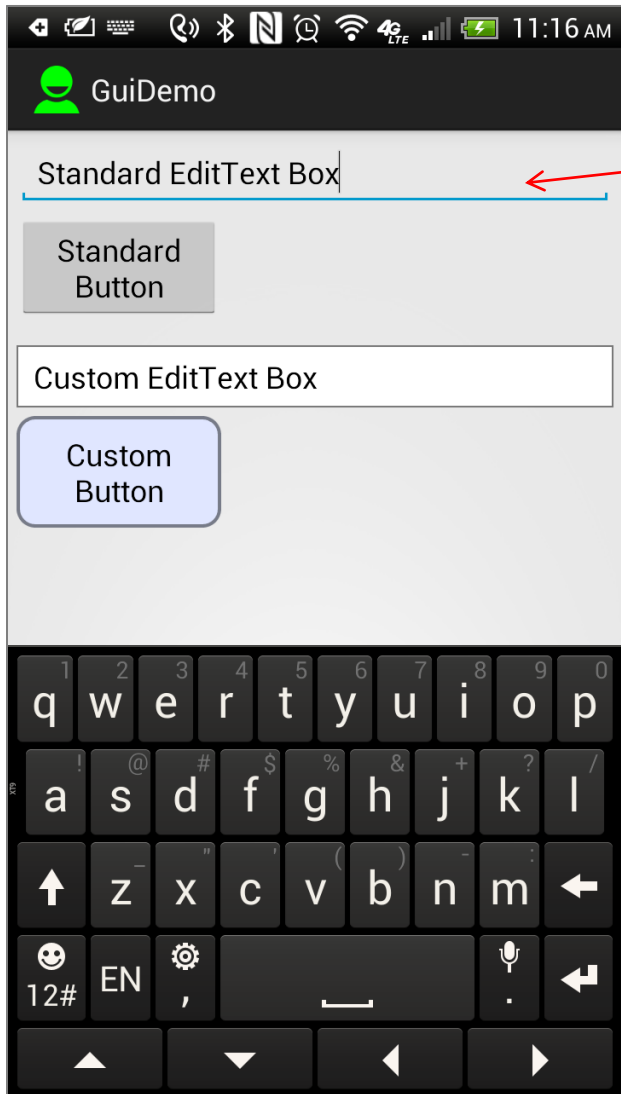
Standard behavior – buttons turns blue when it is pressed.



Custom behavior – buttons turns dark grey with an orange border when it is pressed.

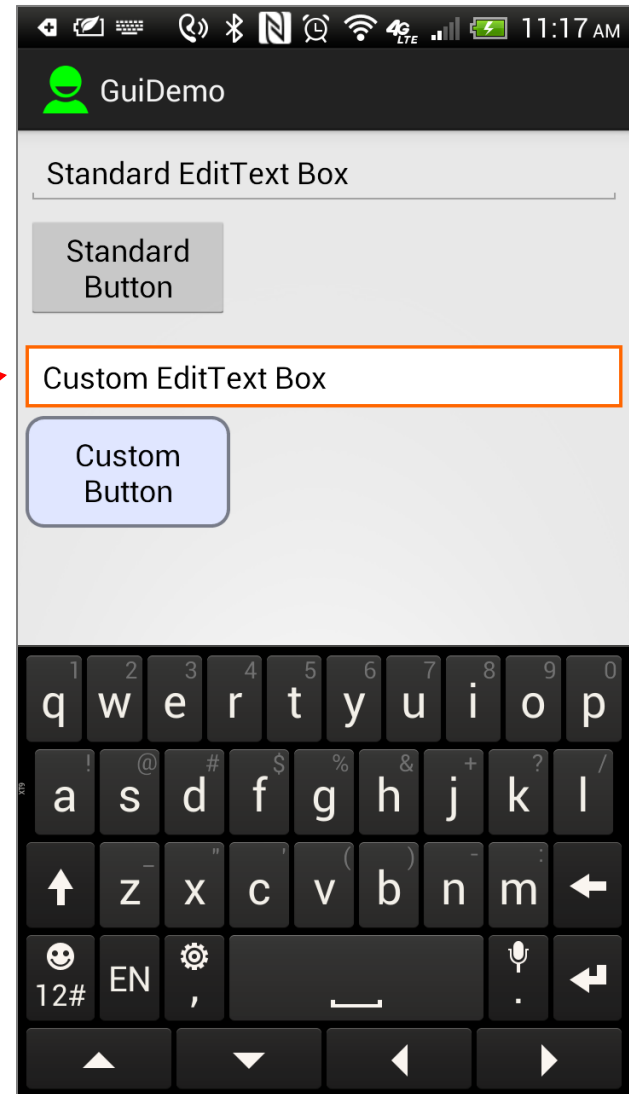
Appendix E: Customizing Widgets

Observe the transient response of the standard and custom made EditText boxes when the user touches the widgets provoking the 'Focused' event.



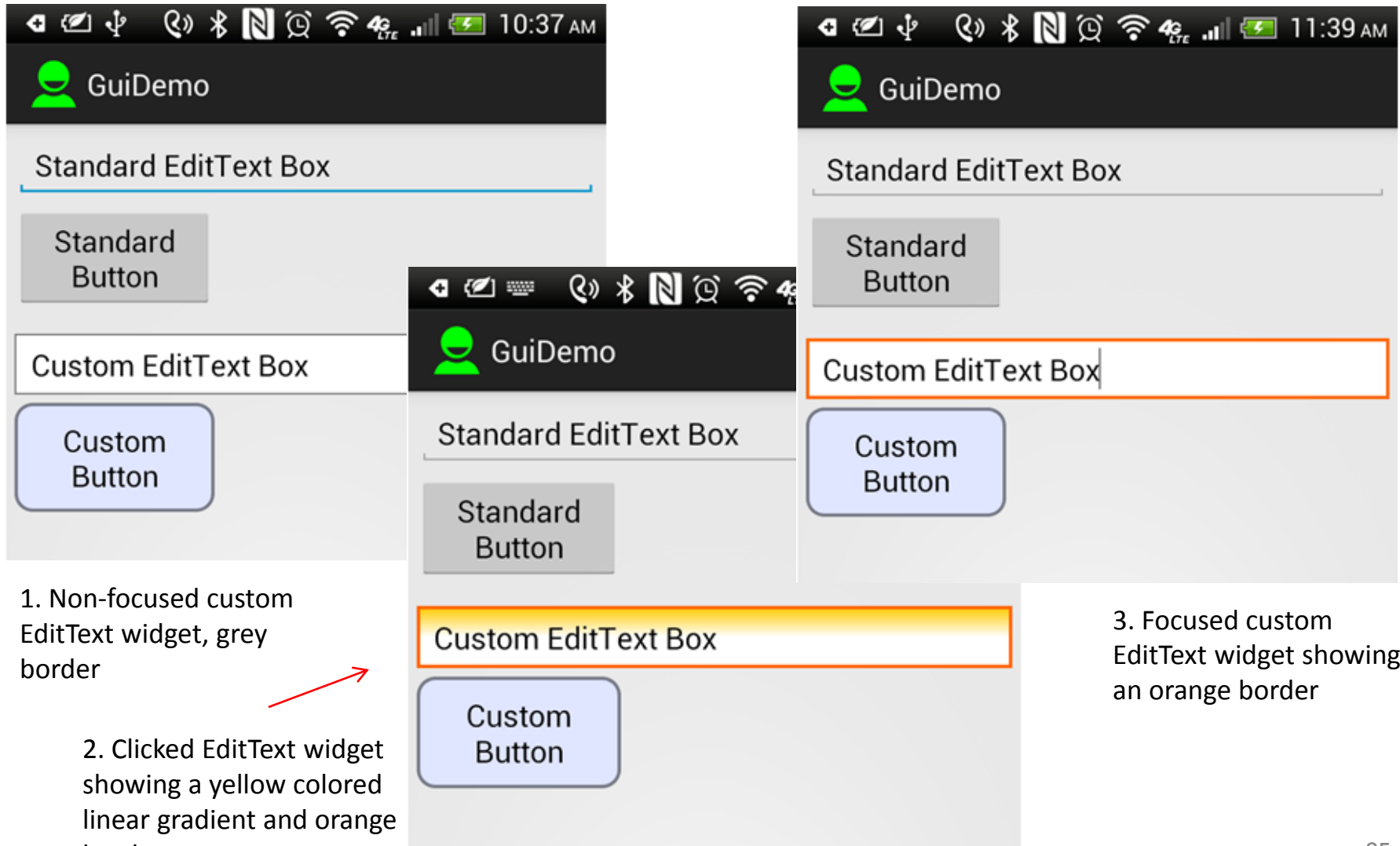
When focused
the standard box
shows a blue
bottom line

A focused
custom box
shows an orange
all-around frame



Appendix E: Customizing Widgets

When the user taps on the custom made EditText box a gradient is applied to the box to flash a visual feedback reassuring the user of her selection.

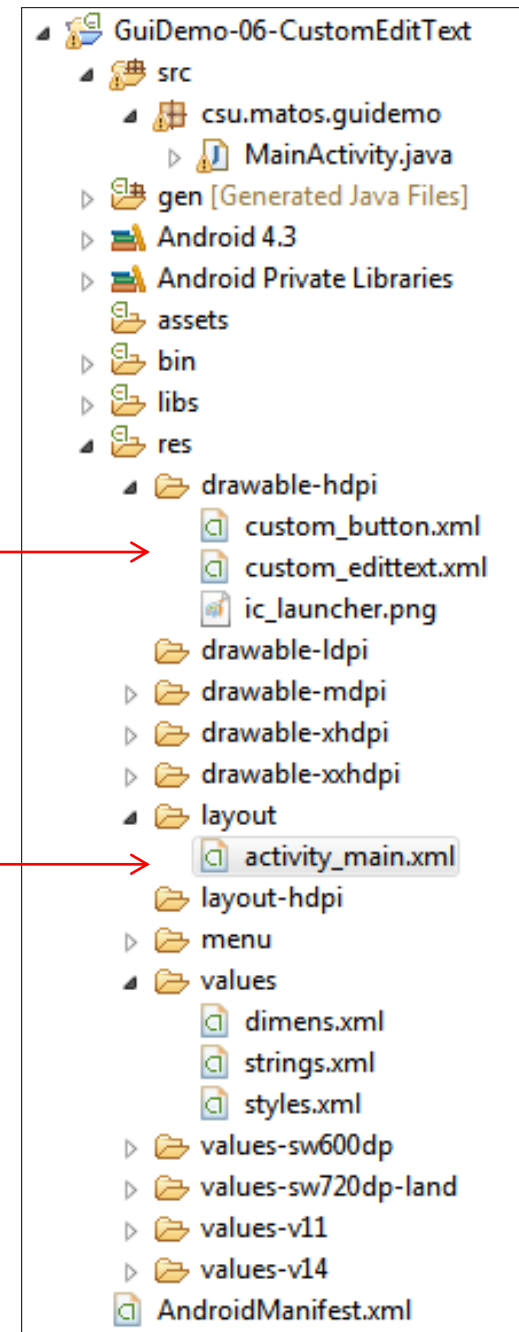


Appendix E: Customizing Widgets

Organizing the application

Definition of the custom templates for Button and EditText widgets

Layout referencing standard and custom made widgets



Appendix E: Customizing Widgets

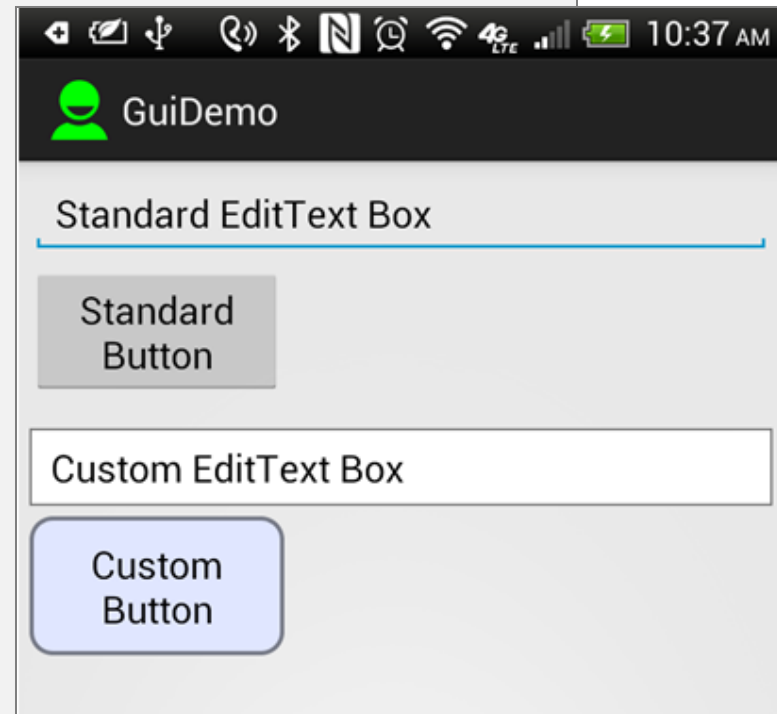
Activity Layout 1 of 2

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="5dp" >

    <EditText
        android:id="@+id/editText1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginBottom="5dp"
        android:ems="10"
        android:inputType="text"
        android:text="@string/standard_edittext" >

        <requestFocus />
    </EditText>

    <Button
        android:id="@+id/button1"
        android:layout_width="120dp"
        android:layout_height="wrap_content"
        android:layout_marginBottom="15dp"
        android:text="@string/standard_button" />
</LinearLayout>
```



Appendix E: Customizing Widgets

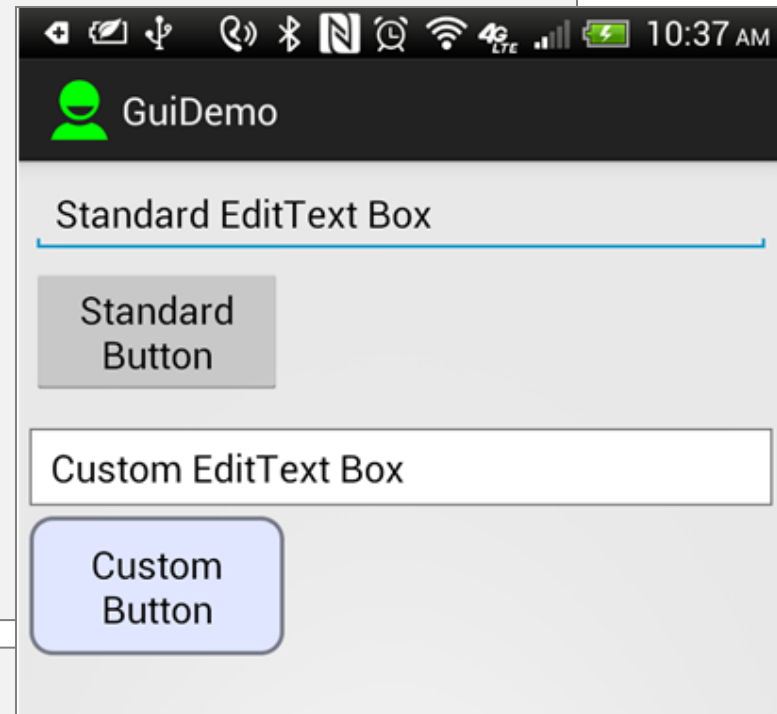
Activity Layout (2 of 2) and Resource: res/values/strings

```
<EditText
    android:id="@+id/editText2"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginBottom="5dp"
    android:background="@drawable/custom_edittext"
    android:ems="10"
    android:inputType="text"
    android:text="@string/custom_edittext" />

<Button
    android:id="@+id/button2"
    android:layout_width="120dp"
    android:layout_height="wrap_content"
    android:background="@drawable/custom_button"
    android:text="@string/custom_button" />

</LinearLayout>
```

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">GuiDemo</string>
    <string name="action_settings">Settings</string>
    <string name="standard_button">Standard Button</string>
    <string name="standard_edittext">Standard EditText Box</string>
    <string name="custom_button">Custom Button</string>
    <string name="custom_edittext">Custom EditText Box</string>
</resources>
```

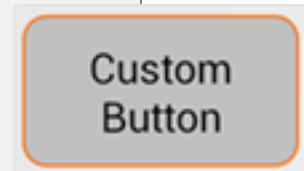


Appendix E: Customizing Widgets

Resource: res/drawable/custom_button.xml

The custom Button widget has two faces based on the event **state_pressed** (true, false). The Shape attribute specifies its solid color, padding, border (stroke) and corners (rounded corners have radius > 0)

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android" >
  <item android:state_pressed="true">
    <shape android:shape="rectangle">
      <corners android:radius="10dp"/>
      <solid android:color="#ffc0c0c0" />
      <padding android:left="10dp"
        android:top="10dp"
        android:right="10dp"
        android:bottom="10dp"/>
      <stroke android:width="1dp" android:color="#ffff6600"/>
    </shape>
  </item>
  <item android:state_pressed="false">
    <shape android:shape="rectangle">
      <corners android:radius="10dp"/>
      <solid android:color="#ffe0e6ff"/>
      <padding android:left="10dp"
        android:top="10dp"
        android:right="10dp"
        android:bottom="10dp"/>
      <stroke android:width="2dp" android:color="#ff777b88"/>
    </shape>
  </item>
</selector>
```



Appendix E: Customizing Widgets

Resource: res/drawable/custom_edittext.xml

The rendition of the custom made EditText widget is based on three states: normal, state_focused, state_pressed.

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:state_pressed="true">
    <shape android:shape="rectangle">
      <gradient
        android:angle="90"
        android:centerColor="#FFffffff"
        android:endColor="#FFffcc00"
        android:startColor="#FFffffff"
        android:type="linear" />
      <stroke android:width="2dp" android:color="#FFff6600" />
      <corners android:radius="0dp" />
      <padding android:left="10dp"
        android:top="6dp"
        android:right="10dp"
        android:bottom="6dp" />
    </shape>
  </item>
```



Custom EditText Box

Appendix E: Customizing Widgets

Resource: res/drawable/custom_edittext.xml

The rendition of the custom made EditText widget is based on three states: normal, state_focused, state_pressed.

```
<item android:state_focused="true">
  <shape>
    <solid android:color="#FFffffff" />
    <stroke android:width="2dp" android:color="#FFff6600" />
    <corners android:radius="0dp" />
    <padding android:left="10dp"
      android:top="6dp"
      android:right="10dp"
      android:bottom="6dp" />
  </shape>
</item>

<item>
  <!-- state: "normal" not-pressed & not-focused -->
  <shape>
    <stroke android:width="1dp" android:color="#ff777777" />
    <solid android:color="#ffffffff" />
    <corners android:radius="0dp" />
    <padding android:left="10dp"
      android:top="6dp"
      android:right="10dp"
      android:bottom="6dp" />
  </shape>
</item>
</selector>
```



Custom EditText Box



Custom EditText Box