Part 4

Android – User Interfaces Using XML Layouts

Victor Matos
Cleveland State University

Notes are based on:

The Busy Coder's Guide to Android Development
by Mark L. Murphy

ISBN: 978-0-9816780-0-9
&
Android Developers
http://developer.android.com/index.html

The View Class

- The **View class** represents the basic building block for user interface components.
- A **View** occupies a rectangular area on the screen and is responsible for *drawing* and *event handling*.
- View is the base class for ***widgets***, which are used to create interactive UI components (buttons, text fields, etc.).
- The **ViewGroup** subclass is the base class for ***layouts***, which are invisible containers that hold other Views (or other ViewGroups) and define their layout properties.

Using Views

All of the views in a window are arranged in a single tree.

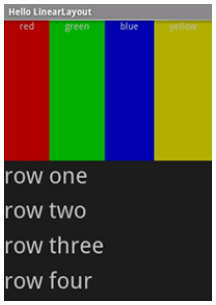
You can add views either from code or by specifying a tree of views in one or more XML layout files.

Once you have created a tree of views, there are typically a few types of common operations you may wish to perform:

1. **Set properties:** for example setting the text of a *TextView*. Properties that are known at build time can be set in the XML layout files.
2. **Set focus:** The framework will handled moving focus in response to user input. To force focus to a specific view, call *requestFocus().*
3. **Set up listeners:** Views allow clients to set listeners that will be notified when something interesting happens to the view. For example, a Button exposes a listener to notify clients when the button is clicked.
4. **Set visibility:** You can hide or show views using *setVisibility(int)*.

A brief sample of UI components

Layouts



Linear Layout
A LinearLayout is a GroupView that will lay child View elements vertically or horizontally.

Relative Layout
A RelativeLayout is a ViewGroup that allows you to layout child elements in positions relative to the parent or siblings elements.

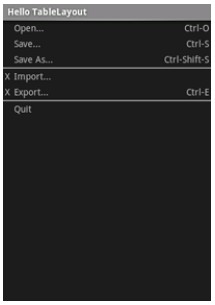
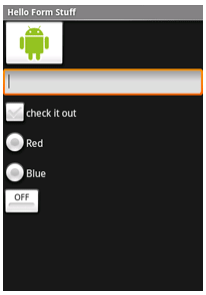


Table Layout
A TableLayout is a ViewGroup that will lay child View elements into rows and columns.

A brief sample of UI components

Widgets



GalleryView

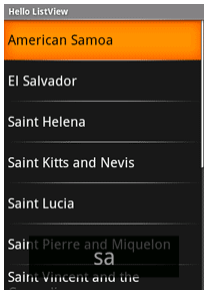
TabWidget

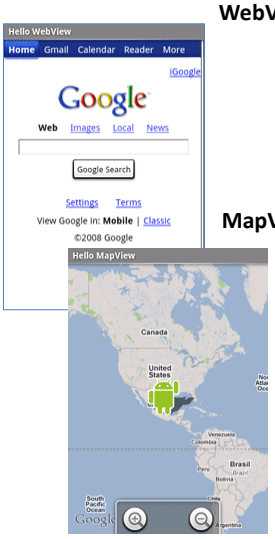
Spinner

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.

A brief sample of UI components





WebView

MapView

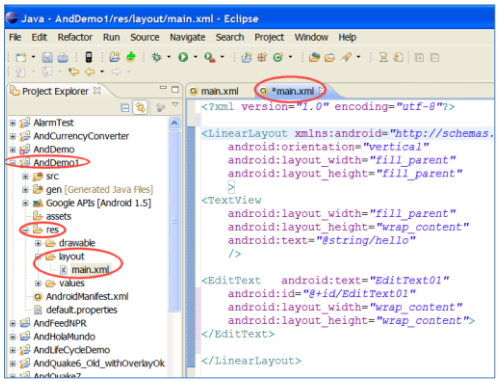
AutoCompleteTextView
It is a version of the *EditText* widget that will provide auto-complete suggestions as the user types. The suggestions are extracted from a collection of strings.

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

What is an XML Layout?

An **XML-based layout** is a specification of the various UI components (widgets) and the relationships to each other – and to their containers – all written in XML format.

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



What is an XML Layout?

Each **XML** file contains a ***tree of elements*** specifying a layout of widgets and containers that make up one View (shown later).

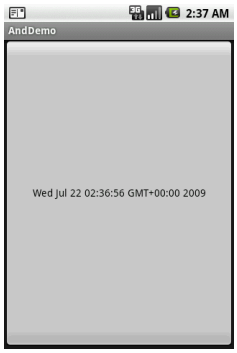
The attributes of the XML elements are *properties*, describing how a widget should look or how a container should behave.

Example:
If a *Button* element has an attribute value of

android:textStyle = "bold"

that means that the text appearing on the face of the button should be rendered in a boldface font style.

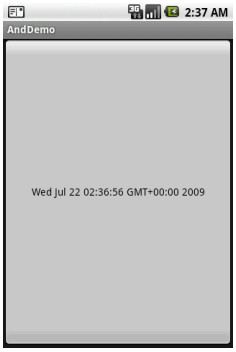
An example



The application places a button to occupy the screen. When clicked the button's text shows current time.

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

public class AndDemo extends Activity {
Button btn;

@Override
public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
   setContentView(R.layout.main);
   btn = (Button) findViewById(R.id.myButton);
   btn.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View v) {
         updateTime();
      }
   });
   }// onCreate
//
private void updateTime() {
   btn.setText(new Date().toString());
}

}
```


An example



This is the XML-Layout definition

```
<?xml version="1.0" encoding="utf-8"?>
<Button
xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/myButton"
   android:text=""
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
/>
```

The root element needs to declare the Android XML namespace:

xmlns:android="http://schemas.android.com/apk/res/android"

All other elements will be children of the root and will inherit that namespace declaration.

Because we want to reference this button from our Java code, we need to give it an identifier via the **android:id** attribute.

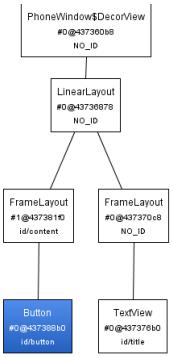
An example *cont.*

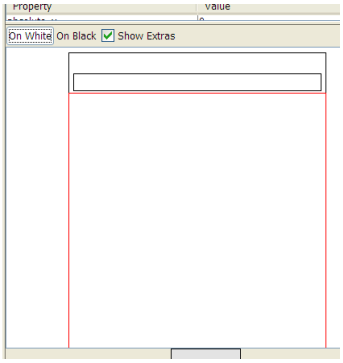
```
<?xml version="1.0" encoding="utf-8"?>
<Button
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/myButton"
   android:text=""
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
/>
```

The remaining attributes are properties of this Button instance:

- **android:text** indicates the initial text to be displayed on the button face (in this case, an empty string)
- **android:layout_width** and **android:layout_height** tell Android to have the button's width and height fill the "parent" container, in this case the entire screen.

UI Hierarchy

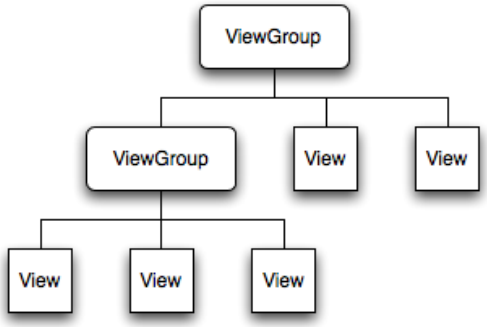




1. The *HierarchyViewer* displays all the UI elements on the screen.
2. It includes the top caption as *TextView* held in its own *FrameLayout*, as well as our Button in its FrameLayou.
3. Both are part of a higher *LinearLayout* which renders all elements of the *PhoneWindowView*.

Android Layouts

1. The most common way to define your layout and express the view hierarchy is with an XML layout file.
2. XML offers a human-readable structure for the layout, much like HTML.
3. Each element in XML is either a *View* or *ViewGroup* object



Android Layouts

Displaying the Application's View

The Android UI Framework paints the screen by walking the View tree by asking each component to draw itself in a *pre-order traversal* way.

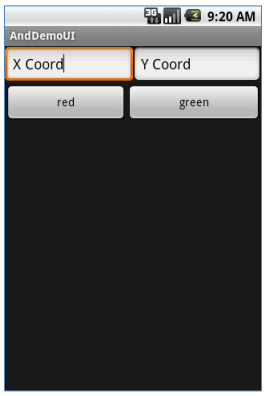
In other words, each component draws itself and then asks each of its children to do the same.

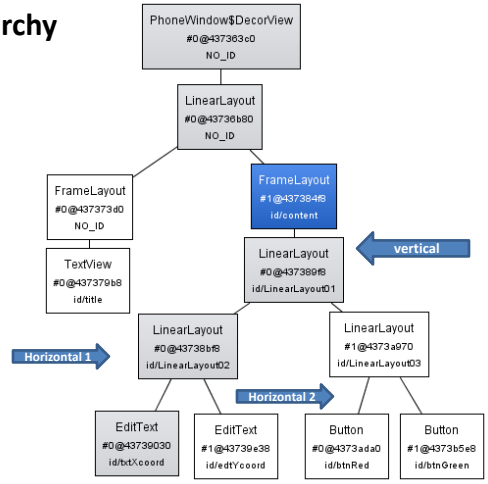
When the whole tree has been rendered, the smaller, nested components that are the leaves of the tree –and that were, therefore, painted later – appear to be drawn on top of the components that are nearer to the root and that were painted earlier.

See: Android – Application Development, by R. Rogers et al. O'Reilly Pub. 2009, ISBN 978-0-596-52147-0

Android Layouts

Example: Display UI Hierarchy



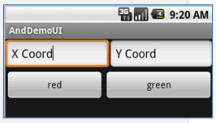


See: Android – Application Development, by R. Rogers et al. O'Reilly Pub. 2009, ISBN 978-0-596-52147-0

Android Layouts

Example: Display UI Hierarchy

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout android:id="@+id/LinearLayout01"
android:layout_width="fill_parent" android:layout_height="fill_parent"
android:orientation="vertical" xmlns:android="http://schemas.android.com/apk/res/android">
      <LinearLayout android:id="@+id/LinearLayout02"
      android:layout_width="fill_parent" android:layout_height="wrap_content">
            <EditText android:id="@+id/txtXcoord" android:layout_width="wrap_content"
                  android:layout_height="wrap_content" android:text="X Coord"
                  android:layout_weight="1">
            </EditText>
            <EditText android:id="@+id/edtYcoord" android:layout_width="wrap_content"
                  android:layout_height="wrap_content" android:text="Y Coord"
                  android:layout_weight="1">
            </EditText>
      </LinearLayout>
      <LinearLayout android:id="@+id/LinearLayout03"
      android:layout_width="fill_parent" android:layout_height="wrap_content">
            <Button android:id="@+id/btnRed" android:layout_width="wrap_content"
                  android:layout_height="wrap_content" android:text="red"
                  android:layout_weight="1">
            </Button>
            <Button android:id="@+id/btnGreen" android:layout_width="wrap_content"
                  android:layout_height="wrap_content" android:text="green"
                  android:layout_weight="1">
            </Button>
      </LinearLayout>
</LinearLayout>
```



Common Layouts

There are five basic types of Layouts:
Frame, Linear, Relative, Table, and **Absolute.**

1. FrameLayout

FrameLayout is the simplest type of layout object. It's basically a *blank space on your screen* that you can later fill with a single object — for example, a picture that you'll swap in and out.

All child elements of the FrameLayout are *pinned to the top left corner of the screen*; you cannot specify a different location for a child view. Subsequent child views will simply be drawn over previous ones, partially or totally obscuring them (unless the newer object is transparent).

Common Layouts

2. LinearLayout

LinearLayout aligns all children in a single direction — *vertically* or *horizontally* depending on the **android:orientation** attribute.

All children are stacked one after the other, so a

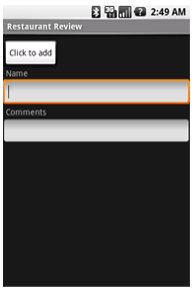
- *vertical* list will only have one child per row, no matter how wide they are, and a
- *horizontal* list will only be one row high (the height of the tallest child, plus padding).

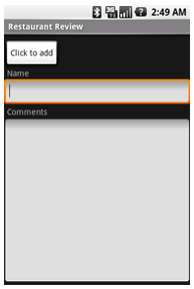
A LinearLayout respects *margins* between children and the *gravity* (right, center, or left alignment) of each child.

Common Layouts

2. LinearLayout

You may attribute a **weight** to children of a LinearLayout.
Weight gives an "importance" value to a view, and allows it to expand to fill any remaining space in the parent view.





Example:
The following two forms represent a LinearLayout with a set of elements: a button, some labels and text boxes. The text boxes have their width set to *fill_parent*; other elements are set to *wrap_content*. The gravity, by default, is left.

The difference between the two versions of the form is that the form on the left has weight values unset (**0** by default), while the form on the right has the comments text box weight set to **1**. If the Name textbox had also been set to 1, the Name and Comments text boxes would be the same height.

Common Layouts

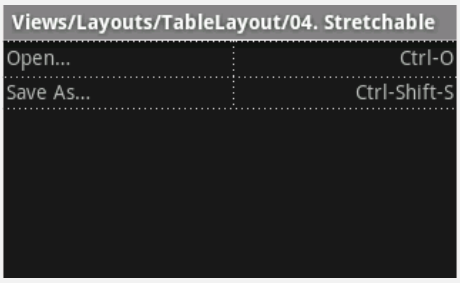
3. TableLayout

1. TableLayout positions its children into **rows** and **columns**.
2. TableLayout containers do not display border lines for their rows, columns, or cells.
3. The table will have *as many columns as the row with the most cells*.
4. A table can leave cells empty, but *cells cannot span columns*, as they can in HTML.
5. *TableRow* objects are the child views of a TableLayout (each TableRow defines a single row in the table).
6. Each row has zero or more cells, each of which is defined by any kind of other View. So, the cells of a row may be composed of a variety of View objects, like ImageView or TextView objects.
7. A cell may also be a ViewGroup object (for example, you can nest another TableLayout as a cell).

Common Layouts

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:stretchColumns="*">
      <TableRow>
            <TextView android:text="Open…"
                android:padding="3dip" />
            <TextView android:text="Ctrl-O"
                android:gravity="right"
                android:padding="3dip" />
      </TableRow>
      <TableRow>
            <TextView android:text="Save As…"
               android:padding="3dip" />
            <TextView  android:text="Ctrl-Shift-S"
                android:gravity="right"
                android:padding="3dip" />
      </TableRow>
</TableLayout>
```

TableLayout Example
The following sample layout has two rows and two cells in each. The accompanying screenshot shows the result, with cell borders displayed as dotted lines (*added for visual effect*).



Common Layouts

4. RelativeLayout

1. RelativeLayout lets child views specify their *position relative to the parent view or to each other* (specified by ID).

2. You can align two elements by *right border*, or make one *below* another, *centered* in the screen, *centered left*, and so on.

3. Elements are *rendered in the order given*, so if the first element is centered in the screen, other elements aligning themselves to that element will be aligned relative to screen center.

4. Also, because of this ordering, if using XML to specify this layout, the element that you will reference (in order to position other view objects) must be listed in the XML file before you refer to it from the other views via its reference ID.

Common Layouts

4. RelativeLayout

5. The defined RelativeLayout parameters are (android:layout_...) :

- width, height,
- below, above
- alignTop, alignParentTop,
- alignBottom, alignParentBottom
- toLeftOf, toRightOf
- padding [Bottom|Left|Right|Top], and
- margin [Bottom|Left|Right|Top].

For example, assigning the parameter

android:layout_toLeftOf="@+id/my_button"

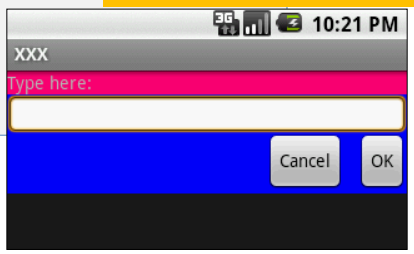
to a TextView would place the TextView to the left of the View with the ID *my_button*

Common Layouts

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="#ff0000ff"
        android:padding="10px" >

  <TextView android:id="@+id/label"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="#ffff0077"
        android:text="Type here:" />

  <EditText android:id="@+id/entry"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/label" />
```

RelativeLayout Example

The example below shows an XML file and the resulting screen in the UI. Note that the attributes that refer to relative elements (e.g., *layout_toLeft*) refer to the ID using the syntax of a relative resource (*@id/id*).



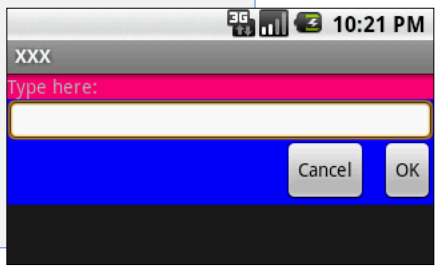
Continue next page

Common Layouts

```
<Button
    android:id="@+id/ok"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@id/entry"
    android:layout_alignParentRight="true"
    android:layout_marginLeft="10px"
    android:text="OK" />

<Button
    android:text="Cancel"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_toLeftOf="@id/ok"
    android:layout_alignTop="@id/ok“ />

</RelativeLayout>
```

RelativeLayout Example

Cont.

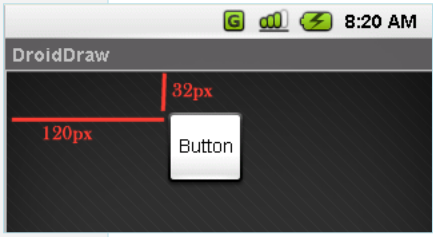


Common Layouts

5. AbsoluteLayout

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.

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout
    android:id="@+id/myAbsoluteLayout"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android" >

<Button
    android:id="@+id/myButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Button"
    android:layout_x="120px"
    android:layout_y="32px"
    >
</Button>
</AbsoluteLayout>
```



A Detailed List of Widgets

For a detailed list consult:

http://developer.android.com/reference/android/widget/package-summary.html

AbsListView
AbsListView.LayoutParams
AbsoluteLayout
AbsoluteLayout.LayoutParams
AbsSeekBar
AbsSpinner
AdapterView<T extends Adapter>
AdapterContextMenuInfo
AlphabetIndexer
AnalogClock
ArrayAdapter<T>
AutoCompleteTextView
BaseAdapter
BaseExpandableListAdapter
Button
CheckBox
CheckedTextView
Chronometer
CompoundButton
CursorAdapter
CursorTreeAdapter
DatePicker
DialerFilter
DigitalClock
EditText
ExpandableListView
ExpandableListContextMenuInfo
Filter
Filter.FilterResults
FrameLayout
FrameLayout.LayoutParams
Gallery
Gallery.LayoutParams
GridView
HeaderViewListAdapter
HorizontalScrollView
ImageButton
ImageSwitcher
ImageView
LinearLayout
LinearLayout.LayoutParams
ListView
ListView.FixedViewInfo
MediaController
MultiAutoCompleteTextView
CommaTokenizer
PopupWindow
ProgressBar
RadioButton
RadioGroup
RadioGroup.LayoutParams
RatingBar
RelativeLayout
RelativeLayout.LayoutParams
RemoteViews
ResourceCursorAdapter
ResourceCursorTreeAdapter
Scroller
ScrollView
SeekBar
SimpleAdapter
SimpleCursorAdapter
SimpleCursorTreeAdapter
SimpleExpandableListAdapter
SlidingDrawer
Spinner
TabHost
TabHost.TabSpec
TableLayout
TableLayout.LayoutParams
TableRow
TableRow.LayoutParams
TabWidget
TextSwitcher
TextView
TextView.SavedState
TimePicker
Toast
ToggleButton
TwoLineListItem
VideoView
ViewAnimator
ViewFlipper
ViewSwitcher
ZoomButton
ZoomControls

Why Use XML Layouts?

One 'good' reason:

XML material is human-readable and "intuitive" !

NOTE
It looks *reasonable* to keep the UI specs in a separated text file rather than mixing it with Java code.

What is sorely missed at this point is a good UI design tool (similar to Forms Designer in Visual Studio) to simplify and accelerate the design process.

XML as a GUI definition format is becoming more commonplace. Microsoft's *Extensible Application Markup Language* **XAML**, Adobe's Flex, and Mozilla's *User Interface Language* **XUL** take a similar approach to that of Android:

put layout details in an XML file and
put programming intelligence in source files.

Using @ in XML Layouts

Again, the button application introduced early in Example 1,

```
<?xml version="1.0" encoding="utf-8"?>
<Button
xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/myButton"
   android:text=""
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
/>
```

Anything you *do want to use* in your Java source needs an

android:id="…"

The convention is to use ***@+id/nnn*** as the id value, where the ***nnn*** represents your locally-unique name for the widget (eg. *@+id/myButton*).

Attaching Layouts to Java Code

Assume *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.

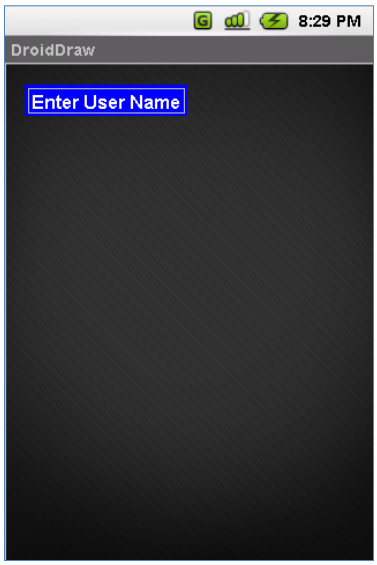
Attaching Layouts to Java Code

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());
}
```

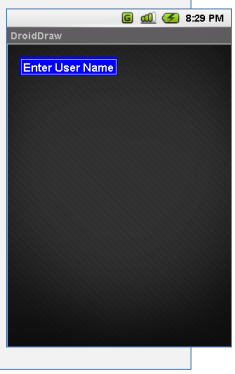

Basic Widgets: Labels



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

Basic Widgets: Labels

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout
    android:id="@+id/absLayout"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android"
>
<TextView
    android:id="@+id/myTextView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="#ff0000ff"
    android:padding="3px"
    android:text="Enter User Name"
    android:textSize="16sp"
    android:textStyle="bold"
    android:gravity="center"
    android:layout_x="20px"
    android:layout_y="22px" >
</TextView>

</AbsoluteLayout>
```



Basic Widgets: Labels/TextViews

http://developer.android.com/reference/android/widget/TextView.html

Attribute Name	*Related Method*	*Description*
android:autoLink	setAutoLinkMask(int)	Controls whether links such as urls and email addresses are automatically found and converted to clickable links.
android:autoText	setKeyListener(KeyListener)	If set, specifies that this TextView has a textual input method and automatically corrects some common spelling errors.
android:bufferType	setText(CharSequence,TextView.BufferType)	Determines the minimum type that getText() will return.
android:capitalize	setKeyListener(KeyListener)	If set, specifies that this TextView has a textual input method and should automatically capitalize what the user types.
android:cursorVisible	setCursorVisible(boolean)	Makes the cursor visible (the default) or invisible Must be a boolean value, either "true" or "false".
android:digits	setKeyListener(KeyListener)	If set, specifies that this TextView has a numeric input method and that these specific characters are the ones that it will accept.
android:drawableBottom	setCompoundDrawablesWithIntrinsicBounds(Drawable,Drawable,Drawable,Drawable)	The drawable to be drawn below the text.
android:drawableLeft	setCompoundDrawablesWithIntrinsicBounds(Drawable,Drawable,Drawable,Drawable)	The drawable to be drawn to the left of the text.
android:drawablePadding	setCompoundDrawablePadding(int)	The padding between the drawables and the text.
android:drawableRight	setCompoundDrawablesWithIntrinsicBounds(Drawable,Drawable,Drawable,Drawable)	The drawable to be drawn to the right of the text.
android:drawableTop	setCompoundDrawablesWithIntrinsicBounds(Drawable,Drawable,Drawable,Drawable)	The drawable to be drawn above the text.
android:editable		If set, specifies that this TextView has an input method.
android:editorExtras	setInputExtras(int)	Reference to an <input-extras> XML resource containing additional data to supply to an input method, which is private to the implementation of the input method.
android:ellipsize	setEllipsize(TextUtils.TruncateAt)	If set, causes words that are longer than the view is wide to be ellipsized instead of broken in the middle.
android:ems	setEms(int)	Makes the TextView be exactly this many ems wide Must be an integer value, such as "100".
android:freezesText	setFreezesText(boolean)	If set, the text view will include its current complete text inside of its frozen icicle in addition to meta-data such as the current cursor position.

Basic Widgets: Labels/TextViews *cont.*

http://developer.android.com/reference/android/widget/TextView.html

Attribute Name	Related Method	Description
android:gravity	setGravity(int)	Specifies how to align the text by the view's x and/or y axis when the text is smaller than the view.
android:height	setHeight(int)	Makes the TextView be exactly this many pixels tall.
android:hint	setHint(int)	Hint text to display when the text is empty.
android:imeActionId	setImeActionLabel(CharSequence,int)	Supply a value for EditorInfo.actionId used when an input method is connected to the text view.
android:imeActionLabel	setImeActionLabel(CharSequence,int)	Supply a value for EditorInfo.actionLabel used when an input method is connected to the text view.
android:imeOptions	setImeOptions(int)	Additional features you can enable in an IME associated with an editor, to improve the integration with your application.
android:includeFontPadding	setIncludeFontPadding(boolean)	Leave enough room for ascenders and descenders instead of using the font ascent and descent strictly.
android:inputMethod	setKeyListener(KeyListener)	If set, specifies that this TextView should use the specified input method (specified by fully-qualified class name).
android:inputType	setRawInputType(int)	The type of data being placed in a text field, used to help an input method decide how to let the user enter text.
android:lineSpacingExtra	setLineSpacing(float,float)	Extra spacing between lines of text.
android:lineSpacingMultiplier	setLineSpacing(float,float)	Extra spacing between lines of text, as a multiplier.
android:lines	setLines(int)	Makes the TextView be exactly this many lines tall Must be an integer value, such as "`100`".
android:linksClickable	setLinksClickable(boolean)	If set to false, keeps the movement method from being set to the link movement method even if autoLink causes links to be found.
android:marqueeRepeatLimit	setMarqueeRepeatLimit(int)	The number of times to repeat the marquee animation.
android:maxEms	setMaxEms(int)	Makes the TextView be at most this many ems wide Must be an integer value, such as "`100`".
android:maxHeight	setMaxHeight(int)	Makes the TextView be at most this many pixels tall Must be a dimension value, which is a floating point number appended with a unit such as "`14.5sp`".
android:maxLength	setFilters(InputFilter)	Set an input filter to constrain the text length to the specified number.
android:maxLines	setMaxLines(int)	Makes the TextView be at most this many lines tall Must be an integer value, such as "`100`".

Basic Widgets: Labels/TextViews *cont.*

http://developer.android.com/reference/android/widget/TextView.html

Attribute Name	Related Method	Description
android:maxWidth	setMaxWidth(int)	Makes the TextView be at most this many pixels wide Must be a dimension value, which is a floating point number appended with a unit such as "`14.5sp`".
android:minEms	setMinEms(int)	Makes the TextView be at least this many ems wide Must be an integer value, such as "`100`".
android:minHeight	setMinHeight(int)	Makes the TextView be at least this many pixels tall Must be a dimension value, which is a floating point number appended with a unit such as "`14.5sp`".
android:minLines	setMinLines(int)	Makes the TextView be at least this many lines tall Must be an integer value, such as "`100`".
android:minWidth	setMinWidth(int)	Makes the TextView be at least this many pixels wide Must be a dimension value, which is a floating point number appended with a unit such as "`14.5sp`".
android:numeric	setKeyListener(KeyListener)	If set, specifies that this TextView has a numeric input method.
android:password	setTransformationMethod(TransformationMethod)	Whether the characters of the field are displayed as password dots instead of themselves.
android:phoneNumber	setKeyListener(KeyListener)	If set, specifies that this TextView has a phone number input method.
android:privateImeOptions	setPrivateImeOptions(String)	An addition content type description to supply to the input method attached to the text view, which is private to the implementation of the input method.
android:scrollHorizontally	setHorizontallyScrolling(boolean)	Whether the text is allowed to be wider than the view (and therefore can be scrolled horizontally).
android:selectAllOnFocus	setSelectAllOnFocus(boolean)	If the text is selectable, select it all when the view takes focus instead of moving the cursor to the start or end.
android:shadowColor	setShadowLayer(float,float,float,int)	Place a shadow of the specified color behind the text.
android:shadowDx	setShadowLayer(float,float,float,int)	Horizontal offset of the shadow.
android:shadowDy	setShadowLayer(float,float,float,int)	Vertical offset of the shadow.
android:shadowRadius	setShadowLayer(float,float,float,int)	Radius of the shadow.

Basic Widgets: Labels/TextViews *cont.*

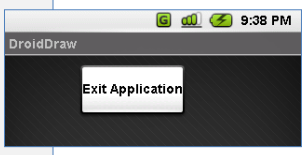
http://developer.android.com/reference/android/widget/TextView.html

Attribute Name	*Related Method*	*Description*
android:singleLine	setTransformationMethod(TransformationMethod)	Constrains the text to a single horizontally scrolling line instead of letting it wrap onto multiple lines, and advances focus instead of inserting a newline when you press the enter key.
android:text	setText(CharSequence)	Text to display.
android:textColor	setTextColor(ColorStateList)	Text color.
android:textColorHighlight	setHighlightColor(int)	Color of the text selection highlight.
android:textColorHint	setHintTextColor(int)	Color of the hint text.
android:textColorLink	setLinkTextColor(int)	Text color for links.
android:textScaleX	setTextScaleX(float)	Sets the horizontal scaling factor for the text Must be a floating point value, such as "`1.2`".
android:textSize	setTextSize(float)	Size of the text.
android:textStyle	setTypeface(Typeface)	Style (bold, italic, bolditalic) for the text.
android:typeface	setTypeface(Typeface)	Typeface (normal, sans, serif, monospace) for the text.
android:width	setWidth(int)	Makes the TextView be exactly this many pixels wide.

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/btnExitApp"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:padding="10px"
android:layout_marginLeft="5px"
android:text="Exit Application"
android:textSize="16sp"
android:textStyle="bold"
android:gravity="center"
android:layout_gravity="center_horizontal"
>
</Button>
```



Your turn!

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

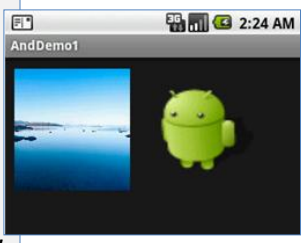
1. Currency calculator
2. Tip Calculator
3. Simple Flashlight

Basic Widgets: Images

- **ImageView** and **ImageButton** are two Android widgets that allow embedding of images in your applications.
- Both are *image-based widgets* analogue to *TextView* and *Button*, respectively.
- Each widget takes an **android:src** or **android:background** attribute (in an XML layout) to specify what picture to use.
- Pictures are usually reference a *drawable* resource.
- You can also set the image content based on a URI from a content provider via setImageURI().
- **ImageButton**, is a subclass of ImageView. It adds the standard *Button* behavior for responding to *click* events.

Basic Widgets: Images

```
. . .
<ImageButton
      android:id="@+id/myImageBtn1"
      android:background="@drawable/default_wallpaper"
      android:layout_width="125px"
      android:layout_height="131px"
>
</ImageButton>

<ImageView
      android:id="@+id/myImageView1"
      android:background="@drawable/ic_launcher_android"
      android:layout_width="108px"
      android:layout_height="90px"
>
</ImageView>
```



Basic Widgets: EditText

- The **EditText** (or *textBox*) widget is an extension of *TextView* that allows updates.
- The control configures itself to be *editable.*
- Important Java methods are: txtBox.setText("someValue") and txtBox.getText().toString()

Basic Widgets: EditText

In addition to the standard TextView properties EditText has many others features such as:

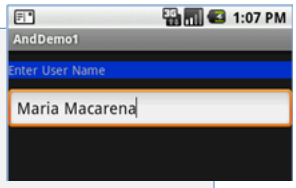
- **android:autoText**, (true/false) provides automatic spelling assistance
- **android:capitalize**, (*words/sentences*) automatic capitalization
- **android:digits**, to configure the field to accept only certain digits
- **android:singleLine**, is the field for single-line / multiple-line input
- **android:password**, (*true/false*) controls field's visibility
- **android:numeric**, (*integer, decimal, signed*) controls numeric format
- **android:phonenumber**, (true/false) Formatting phone numbers

Basic Widgets: EditViews

Example

```
...
<EditText
    android:id="@+id/txtUserName"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textSize="18sp"
    android:autoText="true"
    android:capitalize="words"
    android:hint="First Last Name"
>
</EditText>
...
```

Upper case words

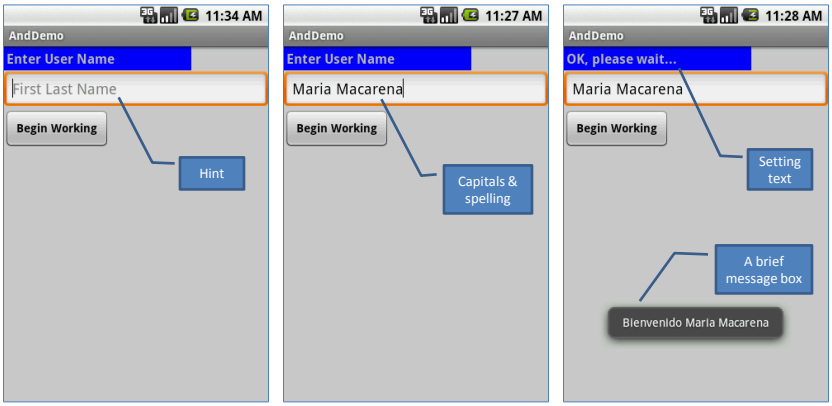


Enter "teh"
will be corrected to: "The"

Suggestion (grey)

Basic Widgets: Example 1

In this little example we will use an **AbsoluteLayout** holding a label(**TexView**), a textBox (**EditText**), and a **Button**.
We will use the view as a sort of simplified login screen.



Basic Widgets: Example 1

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

<TextView
android:id="@+id/labelUserName"
android:layout_width="227px"
android:layout_height="wrap_content"
android:background="#ff0000ff"
android:padding="3px"
android:text="Enter User Name"
android:textSize="16sp"
android:textStyle="bold"
>
</TextView>
```

```
<EditText
android:id="@+id/txtUserName"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:textSize="18sp"
android:autoText="true"
android:capitalize="words"
android:hint="First Last Name"
>
</EditText>

<Button
android:id="@+id/btnBegin"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text=" Begin Working "
android:textSize="14px"
android:textStyle="bold"
>
</Button>

</LinearLayout>
```

Basic Widgets: Example 1

Android's Application (1 of 2)

```
package cis493.gui;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;
//////////////////////////////////////////////////////////////////////////
// "LOGIN" - a gentle introduction to UI controls

public class AndDemo extends Activity {
    TextView labelUserName;
    EditText txtUserName;
    Button btnBegin;

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

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

4. Android – UI - User Interfaces

Basic Widgets: Example 1

Android's Application (2 of 2)

```
        //LISTENER: wiring the button widget to events-&-code
        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(getApplicationContext(),
                       "Bienvenido " + userName,
                       Toast.LENGTH_SHORT).show();
            }
            Toast.makeText(getApplicationContext(),
                   "Bienvenido " + userName,
                   Toast.LENGTH_SHORT).show();
        }

         });// onClick

    }//onCreate

}//class
```

Basic Widgets: Example 1

Note:

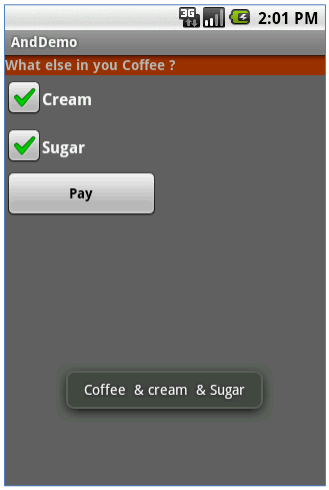
Another way of defining a Listener for multiple button widgets

```
package cis493.gui;

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

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

        //binding the UI's controls defined in "main.xml" to Java code
        btnBegin = (Button) findViewById(R.id.btnBegin);
        btnExit = (Button) findViewById(R.id.btnExit);

        //LISTENER: wiring the button widget to events-&-code
        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 specific type of two-states button that can be either *checked* or *unchecked*.

A example usage of a checkbox inside your activity would be the following:



Example 2: CheckBox

Complete code for the checkBox demo (1 of 3)

Layout: main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
android:id="@+id/linearLayout"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:background="#ff666666"
android:orientation="vertical"
xmlns:android="http://schemas.android.com/apk/res/android"
>

<TextView
android:id="@+id/labelCoffee"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:background="#ff993300"
android:text="What else in you Coffee ?"
android:textStyle="bold"
>
</TextView>
```

```
<CheckBox
android:id="@+id/chkCream"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Cream"
android:textStyle="bold"
>
</CheckBox>
<CheckBox
android:id="@+id/chkSugar"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Sugar"
android:textStyle="bold"
>
</CheckBox>
<Button
android:id="@+id/btnPay"
android:layout_width="153px"
android:layout_height="wrap_content"
android:text="Pay"
android:textStyle="bold"
>
</Button>
</LinearLayout>
```


Example 2: CheckBox

Complete code for the checkBox demo (2 of 3)

```
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.Toast;

public class AndDemo extends Activity {
    CheckBox chkCream;
    CheckBox chkSugar;
    Button btnPay;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.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 (1 of 2)

```
        //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.

Basic Widgets: RadioButtons

Example

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

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

<RadioGroup
android:id="@+id/radGroupCoffeeType"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:orientation="vertical"
>
<TextView
android:id="@+id/labelCoffeeType"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:background="#ff993300"
android:text="What type of coffee?"
android:textStyle="bold"
>
</TextView>
<RadioButton
android:id="@+id/radDecaf"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Decaf"
>
</RadioButton>
<RadioButton
android:id="@+id/radExpresso"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Expresso"
>
</RadioButton>
<RadioButton
android:id="@+id/radColombian"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Colombian"
>
</RadioButton>
</RadioGroup>

...

</LinearLayout>
```


Basic Widgets: RadioButtons

Android Activity (1 of 3)

```
package cis493.demoui;
// example using RadioButtons
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.Toast;

public class AndDemoUI extends Activity {
    CheckBox chkCream;
    CheckBox chkSugar;
    Button btnPay;
    RadioGroup radCoffeeType;
    RadioButton radDecaf;
    RadioButton radExpresso;
    RadioButton radColombian;
```

Basic Widgets: RadioButtons

Android Activity (2 of 3)

```
@Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.main);
       //binding XM1 controls to Java code
       chkCream = (CheckBox)findViewById(R.id.chkCream);
       chkSugar = (CheckBox)findViewById(R.id.chkSugar);
       btnPay = (Button) findViewById(R.id.btnPay);

       radCoffeeType = (RadioGroup)findViewById(R.id.radGroupCoffeeType);
       radDecaf = (RadioButton)findViewById(R.id.radDecaf);
       radExpresso = (RadioButton)findViewById(R.id.radExpresso);
       radColombian = (RadioButton)findViewById(R.id.radColombian);
```


Basic Widgets: 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 (radExpresso.isChecked())
           msg = "Expresso " + msg;

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

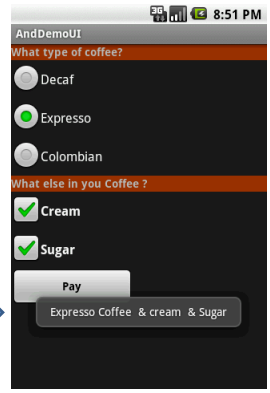
Basic Widgets: RadioButtons

Example

This UI uses *RadioButtons* and *CheckBoxes* to define choices

RadioGroup

Summary of choices



UI – Other Features

All *widgets* extend **View** therefore they acquire a number of useful View properties and methods including:

XML Controls the focus sequence:

- android:visibility
- Android:background

Java methods

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

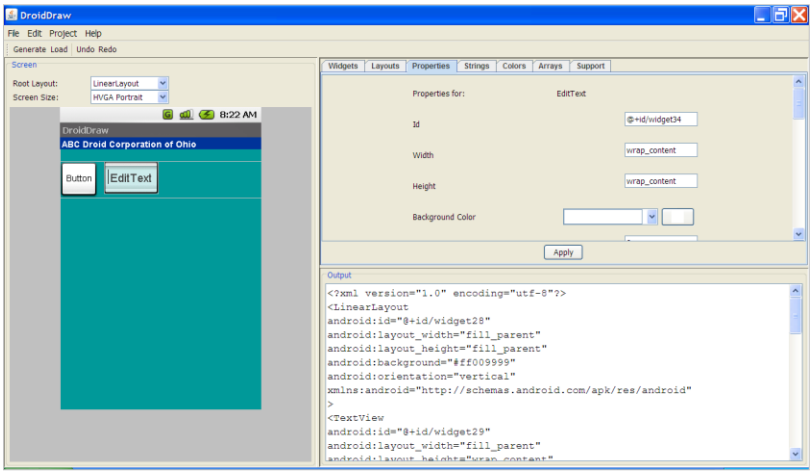
UI - User Interfaces

Questions ?

61

UI - User Interfaces

Resource: DroidDraw www.droidDraw.org



62

Android Asset Studio – Beta (3-October-2010)

AAS Link: http://code.google.com/p/android-ui-utils/
Icon Gen http://android-ui-utils.googlecode.com/hg/asset-studio/dist/index.html
Pencil 1.2 http://pencil.evolus.vn/en-US/Home.aspx
Video: http://www.youtube.com/watch?v=EaT7sYr_f0k&feature=player_embedded

WARNING: These utilities are currently in beta.

Utilities that help in the design and development of Android application user interfaces. This library currently consists of three individual tools for designers and developers:

1. UI Prototyping Stencils

A set of stencils for the Pencil GUI prototyping tool, which is available as an add-on for Firefox or as a standalone download.

2. Android Asset Studio

Try out the beta version: Android Asset Studio (shortlink: http://j.mp/androidassetstudio)

A web-based set of tools for generating graphics and other assets that would eventually be in an Android application's res/ directory.

Currently available asset generators area available for:

Launcher icons
Menu icons
Tab icons
Notification icons

Support for creation of XML resources and nine-patches is planned for a future release.

3. Android Icon Templates

A set of Photoshop icon templates that follow the icon design guidelines, complementing the official Android Icon Templates Pack.

63