

Android - Layouts & UI

Adding an image to resource

The simplest way is by dragging an image into the res/drawable folder of your project (you may need to change the top tab to project instead of android in the project view window and then drag).

then go to your xml and input an "<ImageView>". from here you have to do the usual (width, height) with the added "src" tag which will lead to our resource inside the drawable file.

Note! You can also write instead of "src" you can write "background" which will put the image in the background of ImageView, by adding the image to the background it will fill the entire Imageview while the src attribute is affected by the scale_type attribute(this attribute scale the image in its view. possible values: fitXY-fill all view, center,top etc).

Adding the design through java - Dynamic Layout

Firstly remove from your java file you wish to create the design the
"setContentView(R.layout.activity);"

Then create an instance variable of the specific view.(for example: "TextView textview = new TextView(this)")

Note! when creating an instance of a view element it is required to receive an instance of Context class. Since the Activity inherits from Context we can pass "this" which mean that you are using the class you are currently in as your context or the "getApplicationContex()" which will receive the context of the whole application. In general the Context is the app resources given by the android operating system. As the name suggests, the context is current state of the application/object. It lets newly created objects understand what has been going on.

For each element we create programmatically we should set the layout_width and layout_height through the view's method setLayoutParams, we can pass the WRAP_CONENT or FILL_PARENT or fixed size only in pixels(we can also set the layoutParams when adding the view to his parent with the method addview that accept two arguments).

After creating all the elements(buttons, textview, imageview) we can create the Layout which inherits from ViewGroup and intended to hold the views with the function addView of

the ViewGroup. After that we use the **setContentView** and instead of giving it an id we give it the root layout.

Creating a new resource

We will use for this example creating a color resource.

To do so you must Right click on the values and create a new xml file (name it). After that you write the starting tag “<resources>” and then input the resources of your choice. For example file called colors.xml and in each <item> will be color name and its HEX value.

If we want to create a layout resource we will click on res/layout folder and choose new layout resource file.

LinearLayout - layout_weight

Is a relative way to fit multiple views in one layout.

This work by giving them a number (sort of a percentage)

for example:

Having four views each having 2 in the “layout_weight” = will give each view an equal amount on screen.

Having four views when two having 2 in the “layout_weight” and two other one have 1 = will give two large views that are equal and two other smaller view that are equal on screen but each one is half the size of the first two. meaning the first two will occupy 4/6 of the screen and the others will occupy 2/6 of the screen. Just connects all the weights together and divide the screen by how much you gave each one.

You can also add a “weight_sum” to the layout holding these view allowing you to give a number to the full screen (this allows you to create an empty space in the screen).

Note! to make this work you must define either the height (if the linear layout is vertical) or the width (if the linear layout is horizontal) to 0dp.

Table layout

Is a layout that is base one rows and columns (“<TableRow>”)

When adding a view to the TableRow element each view is one column in the layout. you can force the view to be in a specific column by using the layout_column attribute.

Note! You can give the table layout an id, get a reference to it in your java file and then manipulate the whole table layout, and collapse columns during runtime using the function **setColumnCollapse**.

view visibility

each view has a visibility attribute with the following values

VISIBLE = you can see the view

INVISIBLE = hides the view

GONE = removes the view's frame from the layout (not removing the instance just the space he is taking).

Relativelayout

Relative positioning to any view using the id or to the parent.

We can center the items (horizontal or vertical in the parent) or we can align them relatively to the parent or other items in the layout. This is the most commonly used layout.

Frame layout

Starts all tag at point zero (top left corner) and from there you position the rest.

Toast

is a UI element that appears and then vanishes after a certain amount of time (a kind of growler)

To use this:

```
Toast.makeText(context,"Your message",Toast.LENGTH_SHORT).show();
```

SeekBar

We can set the maximum and the current value. If we want to be notified by the system of the seekbar changes we use a special listener called `OnSeekBarChangeListener`.

Alpha

all drawables have an alpha property that sets the transparency of the image when 0 is fully transparent and 1 is fully visible(opaque). You can set the Drawable alpha's by using the `setAlpha(int)` and pass any value between 0 and 255.

EditText - InputType

All edit texts has input type property that sets the possible inputs for the edittext, the correct keyboard will open for each input type. You can combine different behaviors and input method styles with the bitwise xor. For example, here's how to create a text field that capitalizes the first word of a sentence and also auto-corrects misspellings: `"textCapSentences|textAutoCorrect"`

ScrollView

ViewGroup that allows you to scroll horizontal or vertical The scroll view can only have one direct child inside of it. So the best option to deal with this is to put all of your views inside a viewGroup of any kind (TableRow, LinearLayout...).

onLongClick

when using the on long click you will have to define either true or false at the end of the function. These refers to how the rest of the operations will deal with this click.

ture - will consume the click and not allow any other operations performed by the system, default operations. Meaning "I have dealt with the click don't do anything further")

false - will pass the click to the next operation as defined in the system.

Tag

tag allows you to save an object inside a view for the developer to transfer or manipulate in the code. the tag can be any object and is generally used for any purpose of passing information within the view. When you get only the view upon a click you can get extra information with the view's tag.

State-list

Using a state list allows you to pre define different look for state through your xml file. For example different pictures for each button state, or different colors.

To do this you must create a new xml file inside the drawable folder or the color folder. The main tag will be <selector> and add as many <item> as you want. each <item> will contain the drawable attribute stating the image or the color and the state attribute which can be true or false. The states are boolean for example state_pressed, state_checked and more (the full list is shown below). Where the boolean will define either this drawable item will be affected by this state or not.

After deciding all of this you can then add to the layout xml to the widget you want the state list file you wrote before in the src attribute.

Note to use the state-list using colors for example you will need to create a new directory called "color" and inside that directory add the state list xml file. Then, create a resource file inside the "values" directory with all of the colors you would like to use in the changer. After this you can define a different state for each color in the changer xml file by connecting to each color in the color resource file (' android:src="@color/black" ' for example).

Full state options:

<item

```
    android:drawable="@[package:]drawable/drawable_resource"
    android:state_pressed=["true" | "false"]
    android:state_focused=["true" | "false"]
    android:state_hovered=["true" | "false"]
    android:state_selected=["true" | "false"]
    android:state_checkable=["true" | "false"]
    android:state_checked=["true" | "false"]
    android:state_enabled=["true" | "false"]
```

```
android:state_activated=["true" | "false"]  
android:state_window_focused=["true" | "false"] />
```

instead of the android:drawable you can use android:color