Common Attributes and Properties in Android Views

All UI elements are of type View and have some common properties which are useful to understand their behavior. Whenever you drag and drop a UI element from Designer view, it also automatically generate at least these 3 properties of the android view.

android:id

It is used to identify a particular View in Java Code. When you assign an ID to a View, it makes that android view uniquely identifiable through this ID in that Layout. In later chapters, we will see the use of ID in details.

Here is how it will look like in code android:id="@+id/textView1"

android:layout_width

layout_width property of android views defines the width of that view. You should define width in px or dp.

android:layout_height

layout_height property of android views defines the height of that view. You should define height in px or dp.

Here is how Height and Width will look into code

```
android:layout_width="match_parent"
android:layout_height="wrap_content"
```

Since all UI elements are inherited from android views, all have an onClick property. We will explore onClick for some of the UI elements.

Here are some common attributes or terminology which will be good to understand before watching UI elements in action

wrap_content

This is the value which you can assign to layout's width or height. This tells android to make the size of the UI element just equal to the size of content which the UI has.

For example, a text view with the text "HELLO" with width as wrap_content will only be 5 characters wide(as HELLO is 5 characters wide).

match_parent or fill_parent

Always use match_parent over fill_parent, even though both means same. This value is again to be assigned to width or height. This tells android to match the view's height or width to its parent's height or width. If the parent is taking full size on screen, setting this value on any UI element will also make its height or width equal to the screen size.

dp

Dp is density independent pixels. As Android devices comes with different density of the screen, this value can be used instead of PX (pixle). If you use dp, Android will automatically calculate px for you based on the screen of the android device.

For Example: On some device 1 dp might be equal to 1 px but on some device 1dp = 5px

Padding

Padding is the space inside the border, between the border and the actual view's content. Padding is always inside the view and goes completely around content. You can set padding on all 4 sides, or on an individual side.

For Example: If you set padding property in EditText, the cursor inside the EditText will move.

Here is how Padding looks in code:

For all sides

```
android:padding="2dp"
```

For Individual Sides

```
android:paddingLeft="2dp"
android:paddingRight="3dp"
android:paddingTop="5dp"
android:paddingBottom="8dp
```

Margin

Margins are the spaces outside the border, between the border and the other elements next to this view, IE margin is the outside view, it is a space between 2 Android views, whenever you want to give space between 2 views you can use Margin.

For Example: If you set margin in EditText, the EditText will move by that value from it's parent view.

Here is how Margin will Look at the code:

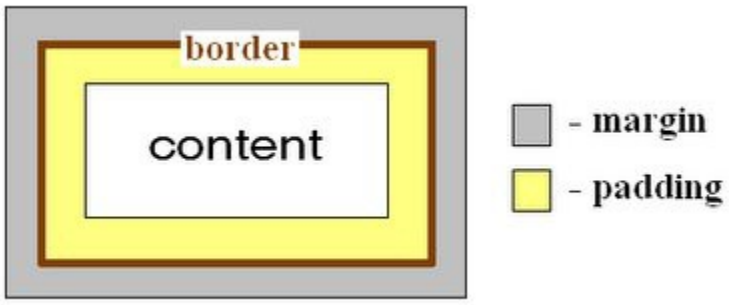
For all sides

```
android:layout_margin="2dp"
```

For Individual Sides

```
android:layout_marginLeft="2dp"
android:layout_marginRight="3dp"
android:layout_marginTop="5dp"
android:layout_marginBottom="8dp
```

You might wonder what is the difference between margin and padding. Let's see an Image to Understand better.



In the image, margin is the gray area outside the entire object. The border is the point where your view ends.

Margin and Padding are the same concept, as in CSS.

Gravity

Whenever you want to align content inside the view, use gravity. android:gravity sets the gravity of the content of the View its used on. Android Gravity, when applied on a view, specifies the direction of its content to be aligned.

For Example: In A Linear Layout when you set Gravity to Center, all its View will get Center aligned.

This is how you set the gravity of layout android:gravity="center"

Layout Gravity

Layout Gravity is the Outside gravity of the View. It sets the gravity of the View or Layout in its parent.

Here is how you set layout gravity android:layout_gravity="right"

HTML/CSS Equivalents:

android:layout_gravity = float in CSS

android:gravity = text-align in CSS

'android:' namespace

You might wonder why all properties start with android:`. If you notice the first line of any XML file there is a namespace definition :

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

Notice the attribute android here, which is referring to Android Schema. This Schema has all references to values like id, text other attributes. You can read more about XML Schema from this link

TextView

Textview is most simple UI element. It is used to display static text, which cannot be changed by the user.

In HTML world, TextView is nothing but a Label.

Lets see some of the useful properties of TextView

android:text

This property defines the text which will come inside the TextView

android:textColor

This property defines the color of text. This take HexaDecimal text color for example #ffff00

android:textSize

This property defines the size of text. It takes values in Sp. For example "15sp"

Example

This is how a simple textview looks in code.

```
<TextView
        android:id="@+id/textView1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Hello World"
        android:textColor="#ffff00"
        android:textSize="15sp"/>
```

A lot of time you require to set the text of TextView Dynamically via code. Let's see how we can change the text of textView via code.

First Step is to get reference of TextView.

```
TextView txtView = (TextView)findViewById(R.id.textView1);
```

Here you call findViewByID method by passing the android:id of the TextView.

Since all the UI elements are view, findViewbyId returns a view object which we need to typecase into correct subview, in this case TextView.

Now, to change the text just call setText method on the txtView object as :

```
txtView.setText("Set your Text Here");
```

With these 2 simple lines of code, you can change the text of textview dynamically

EditText

EditText is another most used UI element in Android. Whenever you want to take some input from the user, you can use EditText.

In HTML world EditText is nothing but an input tag.

Lets see some useful properties of EditText

android:text : this property defines the text which will come inside the EditText, or the text which user will enter. Typically in Java code, once you have the reference of editText object, this is the property which you call to get the value which user might have set.

android:hint: this property defines the hint which you want to give to user, this hint appear inside the edittext but gets disappear once user start typing anything inside the edit text

android:ems: this property defines how many characters long the text should be. Since there is no text inside edit text, making it wrap_content will not help. This property helps you in defining the exact width for edit text.

android:background: this property defines the background of the edit text. You can either choose a color or set an image as the background of the edit text.

android:inputType: this property defines what kind of content the edit text will contain.

For Example: **textPassword** will make the text automatically converted into * as user starts typing. **Number** will change the keyboard layout to numeric, hence helping the user to type the number faster.

This is a very powerful property of EditText and if you know what kind of content your EditText needs, make sure to define that using inputType.

Code

This is how a simple editText will look like

```
<EditText
        android:id="@+id/editText1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:ems="10"
        android:text="Pranay"
        android:hint="Enter User Name"
        android:background="#ff0000"
        android:inputType="text">
```

Most of the times, you would want to get the value of EditText or prepopulate the value dynamically. Let's see how we can do this in code.

First let get the reference of the EditText

```
EditText edtText = (EditText)findViewById(R.id.editText1);
```

As discussed earlier, we just call findViewById to get the reference of a view in code.

Now to get the content of the edit text, just call getText method, like this

```
edtText.getText()
```

Similar to TextView, to set the text of EditText, you can call

```
edtText.setText("Set your value")
```

Button

What

Button is the most used UI element in Android. For every action you want to do, you want the user to click on a button.

Properties

Lets see some of the useful properties of a Button

android:text : this property defines the text which will come on the top of a button. You can change the text dynamically through code

android:textColor: this property defines the color of text on the button. This take HexaDecimal text color for example #ffff00

android:background: this property defines the background of the button. You can either choose a color or set a image as background of the button

android:clickable : this property makes the button enabled or disabled. This takes boolean values (false/true)

Click Listener

Button is made to be clicked. Lets see how we can add a click listener to the button. Even though you can directly define a method from the UI which should be called when button is clicked, the best way to do it is using code.

Get the object of the button in code.

```
Button b = (Button) findViewById(R.id.button1);
```

if you notice the above code, you create a button object and then call findViewByID method by passing the android:id of the button.

Since all the UI elements are view, findViewbyId returns a view object which we need to typecast into correct sub view.

Add listener

```
b.setOnClickListener(new OnClickListener() {

    @Override
    public void onClick(View v) {
```

```
            // Do what you want after button click here
        }
    });
```

In the above code, you add a click listener to the button. Whenever the button is clicked , the onclick method of button is called.

Go to hands on tutorial on Click Listener

Code

This is how a button will look in XML code

```
<Button
android:id="@+id/button1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:background="#f23400"
android:clickable="false"
android:text="Button" />
```

Similar to editText and textView you can call **setText** method on the button object to change the text on the button dynamically.

ImageView

What

Any android app is incomplete without use of an image. Image View is the UI control which you can use to display image in your android app.

In html, image view is simply img tag.

Properties

Lets see some of the useful properties of Image View

android:src : The most important property of ImageView is src . This property helps you to define the image which will be displayed in the image view. You can pass android drawable or your own drawable. Let's see an example

```
android:src="@drawable/ic_launcher"
```

Here "@drawable" refers to your drawable folder which is present inside res folder. After that, you put the image name without the extension.

android:contentDescription : This is not a required property, but it is recommended to set this property. ContentDescription is a text description of the image which you have set. This property helps with accessibility. If this property is set, users facing difficulty in viewing the image can use a Screen Reader, which will read out the text from content description.

ImageView Example

This is how an ImageView will look in XML code

```
<ImageView
        android:id="@+id/imageView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:contentDescription="Image Description"
        android:src="@drawable/ic_launcher" />
```

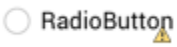
A lot of time you would want to change the image of ImageView dynamically from the code. Let us see how we can do that.

Let's get the reference of image view ImageView imView = (ImageView)findViewById(R.id.imageView1);

Once we have the image view object, we can call setImageResource method to change the image. You can pass the drawable which you want to set to the imageview .

```
imView.setImageResource(R.drawable.new_image);
```

RadioButton



What

If you want your users to pick one from a set of options, use Radio button.

In HTML the input type="radio" is radio button.

Properties

Let's see what are the properties of radio button

RadioGroup : Radio button does not reside on its own. A radio button will always be inside a Radio Group. Radio Group can have multiple radio buttons. All the radio buttons inside the radio group will have a Click event which will be invoked when user selects the radio button.

For example :

```
<RadioGroup
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">
```

The orientation of radio group helps you in aligning radio button in vertical or horizontal format.

android:text : Android text property of a radio button is used for putting text in radio button.

android:checked : Checked property of radio button lets you define preselected/default selected option in a radio group. For example: if you have 3 radio buttons you can use this property in any one button to make it selected by default.

```
android:checked="true"
```

Click listener

Radio buttons are created to be clicked. In a radio group, all the radio buttons will have a click listener. This click listener will be invoked when the user will click on any of the radio button. There are multiple way to add a click listener, but simple way is to do in code using android:onClick property of a radio button.

Lets see an example of an Android RadioButton :

```
<RadioButton
           android:id="@+id/radio_code"
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:onClick="onRadioButtonClicked"
           android:text="Code" />

<RadioButton
           android:id="@+id/radio_learn"
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:onClick="onRadioButtonClicked"
           android:text="Learn"
           android:checked="true"
           />
```

In the above example, notice the property onClick. Since these 2 radio buttons reside in the same radio group, both have a same click listener.

Here is how the click method looks

```
public void onRadioButtonClicked(View view) {
    // Is the button now checked?
    boolean checked = ((RadioButton) view).isChecked();

    // Check which radio button was clicked
    switch(view.getId()) {
        case R.id.radio_code:
            if (checked)
                // Code option selected
            break;
        case R.id.radio_learn:
            if (checked)
                // Learn Option Selected
            break;
    }
}
```

You can create a method in your code, in which you define onClick action of a radio button. In this method, you can get the id of the radio via **view.getId()** . Once you have the id, you can use a switch case statement to determine which radio button was clicked by checking **view.isChecked()**

With a simple code, you can know which radio button selection was made.

RadioButton Example

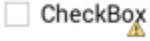
Here is how the complete Radio button declaration will look inside the XML

```
 <RadioGroup
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical" >

         <RadioButton
                        android:id="@+id/radio_code"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:onClick="onRadioButtonClicked"
                        android:text="Code" />

        <RadioButton
                        android:id="@+id/radio_learn"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:onClick="onRadioButtonClicked"
                        android:text="Learn"
                        android:checked="true"
                        />
</RadioGroup>
```

CheckBox



What

If you want to select one or more options from a set, use Check Box.

In HTML the input type="checkbox" is checkbox.

Properties

Let's see what are the properties of checkbox. Most of the properties of checkbox and radio button are similar.

android:text : this property of a checkbox is used for putting text in a radio button.

android:checked : Checked property of checbox lets you have the checkbox checked by default.

```
android:checked="true"
```

Click listener

All the checkbox will have click listener. You can have a common click listener or different click listeners. This click listener will be invoked when the user will select or unselect any checkbox. There are multiple way to add a click listener, but simple way is to do in code using android:onClick property of checkbox.

Let's see an example

```
<CheckBox android:id="@+id/feedback"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Feeback"
        android:onClick="onCheckboxClicked"/>

    <CheckBox android:id="@+id/collectdata"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Collect Data"
        android:onClick="onCheckboxClicked"/>
```

In the above example notice the property onClick. You can either have a common click listener for all the checkbox or can have their individual clicklisteners.

Here is how a common click method looks like :

```
public void onCheckboxClicked(View view) {
    // Is the view now checked?
```

```
        boolean checked = ((CheckBox) view).isChecked();

        // Check which checkbox was clicked
        switch(view.getId()) {
            case R.id.feedback:
                if (checked)
                    // send feedback
                else
                    // don't send feedback
                break;
            case R.id.collectdata:
                if (checked)
                    // Collect Data
                else
                    // Don't collect data
                break;
        }
    }
```

You can create a method in your code in which you define onClick event of a Checkbox. Similar to radio button, in this method, you can get the id of the checbox via **view.getId()**. Once you have the id, you can use a switch case statement to determine which checbox was clicked. Then you can use **view.isChecked()** to determine if the button was checked or unchecked.

With this simple code, you can find out the state of the check box.

ListView

What

Whenever you have a date which is repeated and comes as a collection or list, list view is best User Interface element to use. ListView helps you to display repeating data in the form of a scrollable list. Users can then select any list item by clicking on it.

The ListView is widely used in android applications. A simpler example of list view is your contact book, where you have a list of your contacts displayed in a list view.

Listview is a complex UI pattern, and we have a separate section about listview. You can read more about list view in ListView Section
Go to hands on tutorial on ListView

GridView

What

When you have data which you want to display as a grid of row and columns, you can use gridview. GridView displays item in 2 dimensional scrollable grid.

The most simple example of gridview is your Picture Gallery, In Picture Gallery all the pictures are arranged in a grid like format.

You can read more about gridview at this link :
http://developer.android.com/guide/topics/ui/layout/gridview.html

ScrollView

What

ScrollView helps you in making any Layout automatically scrollable. When you are not using listview and have content which is going beyond the screen size, you can use scroll view to make this content scrollable.

Properties

The most important property of Scroll View is it's children. ScrollView can only have 1 child. It means that you need to have all your layouts nested under 1 layout which acts as a children of your list view.

Let's see an Example:

Let's assume we have 2 edit Text in our layouts like this

```
 <EditText
                android:id="@+id/editText1"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:ems="10"
                android:text="Pranay"
                android:hint="Enter User Name"
                android:background="#ff0000"
                android:inputType="text">

<EditText
                android:id="@+id/editText2"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:ems="10"
                android:text="Airan"
                android:hint="Enter Password"
                android:background="#ff0000"
                android:inputType="text">
```

Now if we want to make our UI with these 2 Edit Text Scrollable, we need to put them into a parent layout. ScrollView cannot have 2 children, so we cannot put this edit text into a scroll view directly.

Let's add a Linear layout as a parent of this edit text.

```
<LinearLayout
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:orientation="vertical" >

        <EditText
                       android:id="@+id/editText1"
                       android:layout_width="wrap_content"
                       android:layout_height="wrap_content"
                       android:ems="10"
                       android:text="Pranay"
                       android:hint="Enter User Name"
                       android:background="#ff0000"
                       android:inputType="text">

       <EditText
                       android:id="@+id/editText2"
                       android:layout_width="wrap_content"
                       android:layout_height="wrap_content"
                       android:ems="10"
                       android:text="Airan"
                       android:hint="Enter Password"
                       android:background="#ff0000"
                       android:inputType="text">

</LinearLayout>
```

Now that we have both the edit Text into 1 layout, we can put this Linear layout as child of a ScrollView. You can then go on and add more complex elements inside this layout and all the elements will now be scrollable.

code

This is how our complete scroll view will look

```
<ScrollView
  android:layout_height="wrap_content"
  android:layout_width="wrap_content">

 <LinearLayout
               android:layout_width="wrap_content"
               android:layout_height="wrap_content"
               android:orientation="vertical" >

                <EditText
                              android:id="@+id/editText1"
                              android:layout_width="wrap_content"
                              android:layout_height="wrap_content"
                              android:ems="10"
                              android:text="Pranay"
                              android:hint="Enter User Name"
                              android:background="#ff0000"
                              android:inputType="text">
```

```
                <EditText
                        android:id="@+id/editText2"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:ems="10"
                        android:text="Airan"
                        android:hint="Enter Password"
                        android:background="#ff0000"
                        android:inputType="text">

        </LinearLayout>
</ScrollView>
```

Android has a lot of UI elements which are designed for specific purposes,