Android - Layouts

The basic building block for user interface is a **View** object which is created from the View class and 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 like buttons, text fields, etc.

The View objects are usually called "widgets" and must contain exactly one root element, which must be a View or ViewGroup object. Once you've defined the root element, you can add additional layout objects or widgets as child elements to gradually build a View hierarchy that defines your layout.

The **ViewGroup** is a subclass of **View** and provides invisible container that hold other Views or other ViewGroups and define their layout properties.

The ViewGroup objects are usually called "layouts" can be one of many types that provide a different layout structure, such as LinearLayout or ConstraintLayout .

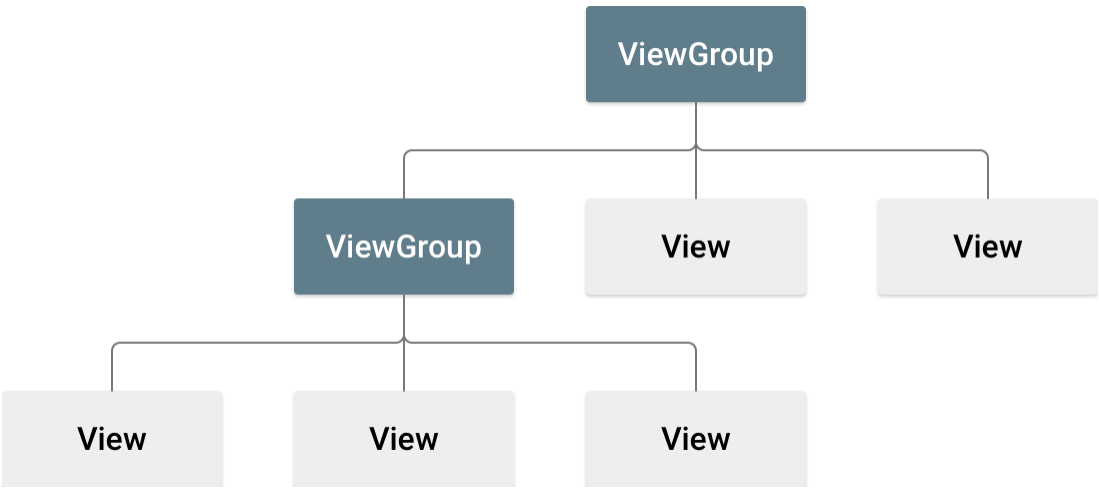
At third level we have different layouts which are subclasses of ViewGroup class and a typical layout defines the visual structure for an Android user interface and can be created either at run time using **View/ViewGroup** objects or you can declare your layout using simple XML file **main_layout.xml** which is located in the res/layout folder of your project.

You can declare a layout in two ways:

- **Declare UI elements in XML**. Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses, such as those for widgets and layouts.

  You can also use Android Studio's Layout Editor to build your XML layout using a drag-and-drop interface.

- **Instantiate layout elements at runtime**. Your app can create View and ViewGroup objects (and manipulate their properties) programmatically.



Layout Attributes

Each layout has a set of attributes which define the visual properties of that layout. There are few common attributes among all the layouts and their are other attributes which are specific to that layout. Following are common attributes and will be applied to all the layouts:

Attribute & Description
android:id This is the ID which uniquely identifies the view.
android:layout_width This is the width of the layout.
android:layout_height This is the height of the layout

android:layout_marginTop This is the extra space on the top side of the layout.
android:layout_marginBottom This is the extra space on the bottom side of the layout.
android:layout_marginLeft This is the extra space on the left side of the layout.
android:layout_marginRight This is the extra space on the right side of the layout.
android:layout_gravity This specifies how child Views are positioned.
android:layout_weight This specifies how much of the extra space in the layout should be allocated to the View.
android:layout_x This specifies the x-coordinate of the layout.
android:layout_y This specifies the y-coordinate of the layout.
android:layout_width This is the width of the layout.
android:layout_width This is the width of the layout.

android:paddingLeft This is the left padding filled for the layout.
android:paddingRight This is the right padding filled for the layout.
android:paddingTop This is the top padding filled for the layout.
android:paddingBottom This is the bottom padding filled for the layout.

Here width and height are the dimension of the layout/view which can be specified in terms of dp (Density-independent Pixels), sp (Scale-independent Pixels), pt (Points which is 1/72 of an inch), px(Pixels), mm (Millimeters) and finally in (inches).

You can specify width and height with exact measurements but more often, you will use one of these constants to set the width or height −

- **android:layout_width=wrap_content** tells your view to size itself to the dimensions required by its content.
- **android:layout_width=fill_parent** tells your view to become as big as its parent view.

Gravity attribute plays important role in positioning the view object and it can take one or more (separated by '|') of the following constant values.

Constant	Value	Description
top	0x30	Push object to the top of its container, not changing its size.

bottom	0x50	Push object to the bottom of its container, not changing its size.
left	0x03	Push object to the left of its container, not changing its size.
right	0x05	Push object to the right of its container, not changing its size.
center_vertical	0x10	Place object in the vertical center of its container, not changing its size.
fill_vertical	0x70	Grow the vertical size of the object if needed so it completely fills its container.
center_horizontal	0x01	Place object in the horizontal center of its container, not changing its size.
fill_horizontal	0x07	Grow the horizontal size of the object if needed so it completely fills its container.
center	0x11	Place the object in the center of its container in both the vertical and horizontal axis, not changing its size.
fill	0x77	Grow the horizontal and vertical size of the object if needed so it completely fills its container.
clip_vertical	0x80	Additional option that can be set to have the top and/or bottom edges of the child clipped to its container's bounds. The clip will be based on the vertical gravity: a top gravity will clip the bottom edge, a bottom gravity will clip the top edge, and neither will clip both edges.

clip_horizontal	0x08	Additional option that can be set to have the left and/or right edges of the child clipped to its container's bounds. The clip will be based on the horizontal gravity: a left gravity will clip the right edge, a right gravity will clip the left edge, and neither will clip both edges.
start	0x00800003	Push object to the beginning of its container, not changing its size.
end	0x00800005	Push object to the end of its container, not changing its size.

View Identification

A view object may have a unique ID assigned to it which will identify the View uniquely within the tree. The syntax for an ID, inside an XML tag is −

```
android:id="@+id/my_button"
```

Following is a brief description of @ and + signs −

- The at-symbol (@) at the beginning of the string indicates that the XML parser should parse and expand the rest of the ID string and identify it as an ID resource.
- The plus-symbol (+) means that this is a new resource name that must be created and added to our resources. To create an instance of the view object and capture it from the layout, use the following −

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

Android Layout Types

There are number of Layouts provided by Android which you will use in almost all the Android applications to provide different view:

Linear Layout

LinearLayout is a view group that aligns all children in a single direction, vertically or horizontally.

It creates a scrollbar if the length of the window exceeds the length of the screen.

Following are the important attributes specific to LinearLayout:

Attribute & Description
android:id This is the ID which uniquely identifies the layout.
android:baselineAligned This must be a boolean value, either "true" or "false" and prevents the layout from aligning its children's baselines.
android:baselineAlignedChildIndex When a linear layout is part of another layout that is baseline aligned, it can specify which of its children to baseline align.

android:divider This is drawable to use as a vertical divider between buttons. You use a color value, in the form of "#rgb", "#argb", "#rrggbb", or "#aarrggbb".
android:gravity This specifies how an object should position its content, on both the X and Y axes. Possible values are top, bottom, left, right, center, center_vertical, center_horizontal etc.
android:orientation This specifies the direction of arrangement and you will use "horizontal" for a row, "vertical" for a column. The default is horizontal.
android:weightSum Sum up of child weight

Relative Layout

RelativeLayout is a view group that displays child views in relative positions, and Enables you to specify the location of child objects relative to each other (child A to the left of child B) or to the parent (aligned to the top of the parent).

Following are the important attributes specific to RelativeLayout:

Attribute & Description
android:id This is the ID which uniquely identifies the layout.
android:gravity This specifies how an object should position its content, on both the X and Y axes. Possible values are top, bottom, left, right, center, center_vertical, center_horizontal etc.
android:ignoreGravity This indicates what view should not be affected by gravity.

Using RelativeLayout, you can align two elements by right border, or make one below another, centered in the screen, centered left, and so on.

By default, all child views are drawn at the top-left of the layout, so you must define the position of each view using the various layout properties available from **RelativeLayout.LayoutParams** and few of the important attributes are given below:

Attribute & Description
android:layout_above

Positions the bottom edge of this view above the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name"

android:layout_alignBottom

Makes the bottom edge of this view match the bottom edge of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".

android:layout_alignLeft

Makes the left edge of this view match the left edge of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".

android:layout_alignParentBottom

If true, makes the bottom edge of this view match the bottom edge of the parent. Must be a boolean value, either "true" or "false".

android:layout_alignParentEnd

If true, makes the end edge of this view match the end edge of the parent. Must be a boolean value, either "true" or "false".

android:layout_alignParentLeft

If true, makes the left edge of this view match the left edge of the parent. Must be a boolean value, either "true" or "false".

android:layout_alignParentRight

If true, makes the right edge of this view match the right edge of the parent. Must be a boolean value, either "true" or "false".

android:layout_alignParentStart

If true, makes the start edge of this view match the start edge of the parent. Must be a boolean value, either "true" or "false".

android:layout_alignParentTop

If true, makes the top edge of this view match the top edge of the parent. Must be a boolean value, either "true" or "false".

android:layout_alignRight

Makes the right edge of this view match the right edge of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".

android:layout_alignStart

Makes the start edge of this view match the start edge of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".

android:layout_alignTop

Makes the top edge of this view match the top edge of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".

android:layout_below

Positions the top edge of this view below the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".

android:layout_centerHorizontal

If true, centers this child horizontally within its parent. Must be a boolean value, either "true" or "false".

android:layout_centerInParent

If true, centers this child horizontally and vertically within its parent. Must be a boolean value, either "true" or "false".

android:layout_centerVertical

If true, centers this child vertically within its parent. Must be a boolean value, either "true" or "false".

android:layout_toEndOf

Positions the start edge of this view to the end of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".

android:layout_toLeftOf

Positions the right edge of this view to the left of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".

android:layout_toRightOf

Positions the left edge of this view to the right of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".

android:layout_toStartOf

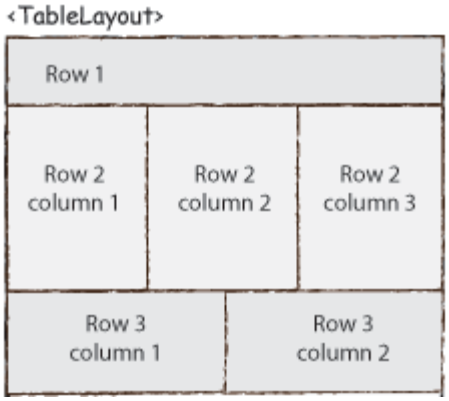
Positions the end edge of this view to the start of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".

Table Layout

TableLayout is a view that groups views into rows and columns.

Android TableLayout going to be arranged groups of views into rows and columns. You will use the <TableRow> element to build a row in the table. Each row has zero or more cells; each cell can hold one View object.

TableLayout containers do not display border lines for their rows, columns, or cells.



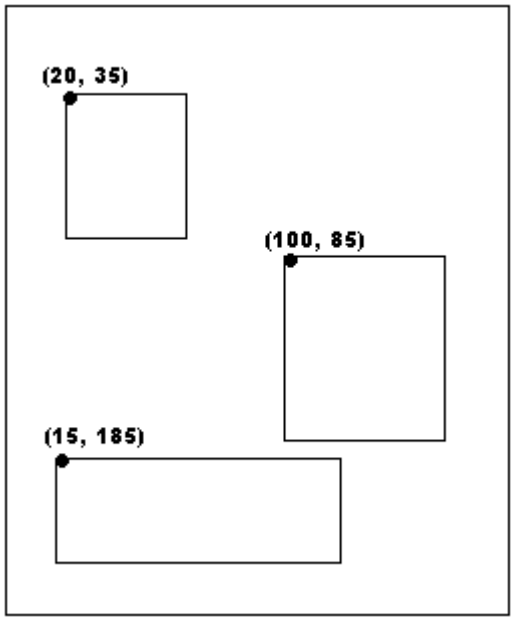
Following are the important attributes specific to TableLayout:

Attribute & Description
android:id This is the ID which uniquely identifies the layout.
android:collapseColumns This specifies the zero-based index of the columns to collapse. The column indices must be separated by a comma: 1, 2, 5.
android:shrinkColumns The zero-based index of the columns to shrink. The column indices must be separated by a comma: 1, 2, 5.
android:stretchColumns The zero-based index of the columns to stretch. The column indices must be separated by a comma: 1, 2, 5.

Absolute Layout

AbsoluteLayout enables you to specify the exact location of its children.

An Absolute Layout 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.



Following are the important attributes specific to AbsoluteLayout:

Attribute & Description
android:id This is the ID which uniquely identifies the layout.
android:layout_x This specifies the x-coordinate of the view.
android:layout_y This specifies the y-coordinate of the view.

Frame Layout

The FrameLayout is a placeholder on screen that you can use to display a single view.

Following are the important attributes specific to FrameLayout:

Attribute & Description
android:id This is the ID which uniquely identifies the layout.
android:foreground This defines the drawable to draw over the content and possible values may be a color value, in the form of "#rgb", "#argb", "#rrggbb", or "#aarrggbb".
android:foregroundGravity Defines the gravity to apply to the foreground drawable. The gravity defaults to fill. Possible values are top, bottom, left, right, center, center_vertical, center_horizontal etc.
android:measureAllChildren Determines whether to measure all children or just those in the VISIBLE or INVISIBLE state when measuring. Defaults to false.

List View

ListView is a view group that displays a list of scrollable items. (Displays a scrolling single column list)

Android ListView is a view which groups several items and display them in vertical scrollable list. The list items are automatically inserted to the list using an Adapter that pulls content from a source such as an array or database.

An adapter actually bridges between UI components and the data source that fill data into UI Component. Adapter holds the data and send the data to adapter view, the view can takes the data from adapter view and shows the data on different views like as spinner, list view, grid view etc.

The ListView and GridView are subclasses of AdapterView and they can be populated by binding them to an Adapter, which retrieves data from an external source and creates a View that represents each data entry.

Android provides several subclasses of Adapter that are useful for retrieving different kinds of data and building views for an AdapterView (i.e. ListView or GridView). The common adapters are ArrayAdapter,Base Adapter,CursorAdapter, SimpleCursorAdapter,SpinnerAdapter and

WrapperListAdapter. We will see separate examples for both the adapters.

Following are the important attributes specific to GridView:

Attribute & Description
android:id This is the ID which uniquely identifies the layout.
android:divider This is drawable or color to draw between list items.

android:dividerHeight This specifies height of the divider. This could be in px, dp, sp, in, or mm.
android:entries Specifies the reference to an array resource that will populate the ListView.
android:footerDividersEnabled When set to false, the ListView will not draw the divider before each footer view. The default value is true.
android:headerDividersEnabled When set to false, the ListView will not draw the divider after each header view. The default value is true.

ArrayAdapter

You can use this adapter when your data source is an array. By default, ArrayAdapter creates a view for each array item by calling toString() on each item and placing the contents in a TextView. Consider you have an array of strings you want to display in a ListView, initialize a new ArrayAdapter using a constructor to specify the layout for each string and the string array:

```
ArrayAdapter adapter = new ArrayAdapter<String>(this,R.layout.ListView,StringArray);
```

Here are arguments for this constructor:

- First argument **this** is the application context. Most of the case, keep it **this**.
- Second argument will be layout defined in XML file and having **TextView** for each string in the array.
- Final argument is an array of strings which will be populated in the text view.

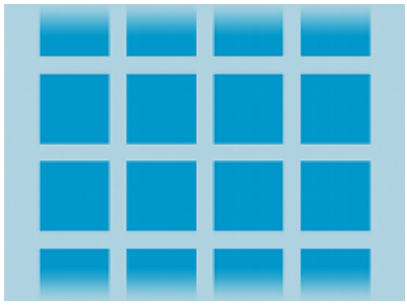
Once you have array adapter created, then simply call **setAdapter()** on your **ListView** object as follows –

```
ListView listView = (ListView) findViewById(R.id.listview);
listView.setAdapter(adapter);
```

You will define your list view under res/layout directory in an XML file. For our example we are going to using activity_main.xml file.

Grid View

GridView is a ViewGroup that displays items in a two-dimensional, scrollable grid.



Android **GridView** shows items in two-dimensional scrolling grid (rows & columns) and the grid items are not necessarily predetermined but they automatically inserted to the layout using a **ListAdapter**

An adapter actually bridges between UI components and the data source that fill data into UI Component. Adapter can be used to supply the data to like spinner, list view, grid view etc.

The **ListView** and **GridView** are subclasses of **AdapterView** and they can be populated by binding them to an **Adapter**, which retrieves data from an external source and creates a View that represents each data entry.

Following are the important attributes specific to GridView :

Attribute & Description

android:id

This is the ID which uniquely identifies the layout.

android:columnWidth

This specifies the fixed width for each column. This could be in px, dp, sp, in, or mm.

android:gravity

Specifies the gravity within each cell. Possible values are top, bottom, left, right, center, center_vertical, center_horizontal etc.

android:horizontalSpacing

Defines the default horizontal spacing between columns. This could be in px, dp, sp, in, or mm.

android:numColumns

Defines how many columns to show. May be an integer value, such as "100" or auto_fit which means display as many columns as possible to fill the available space.

android:stretchMode

Defines how columns should stretch to fill the available empty space, if any. This must be either of the values −

- none − Stretching is disabled.
- spacingWidth − The spacing between each column is stretched.
- columnWidth − Each column is stretched equally.
- spacingWidthUniform − The spacing between each column is uniformly stretched..

android:verticalSpacing

Defines the default vertical spacing between rows. This could be in px, dp, sp, in, or mm.