UNIT-3

BUILDING BLOCKS FOR ANDROID APPLICATION DESIGN

Introduction to layouts:

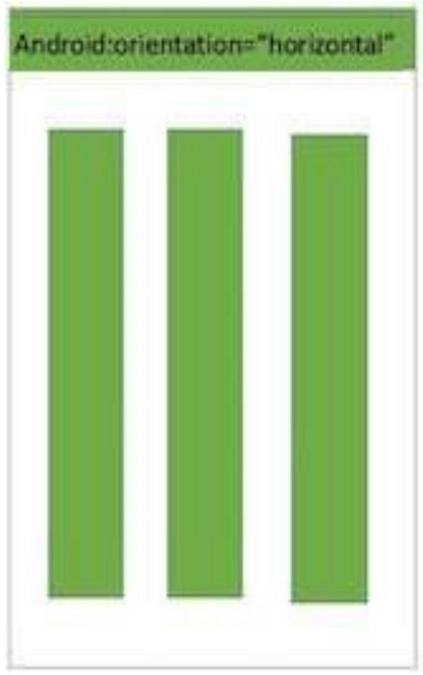
A container is a view used to contain other views. Android offers a collection of view classes that act as containers for views. These container classes are called layouts, and it decides the organization, size, and position of their children views. Every Layout is defined in an xml file which is located in **App > res > Layout** .

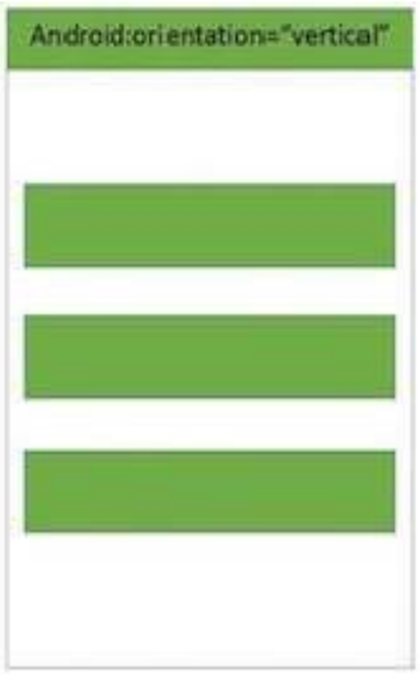
Types of Layouts:

1. **Linear layout**- Organizes its children either horizontally or vertically.
2. **Relative layout**- Organizes its children relative to one another or to the parent.
3. **Absolute layout**- Each child control is given a specific location within the bounds of the container.
4. **Frame layout**- Displays a single view, that is the next view replaces the previous view and hence is used to dynamically change the children in the layout.
5. **Table layout**- Organizes its children tabular form.
6. **Grid layout**- Organizes its children in grid format.

Linear layout:

The LinearLayout is the most basic layout, and it arranges its elements sequentially, either horizontally or vertically.





To arrange controls within a linear layout, the following attributes are used

- **android:orientation**—Used for arranging the controls in the container in horizontal or vertical order

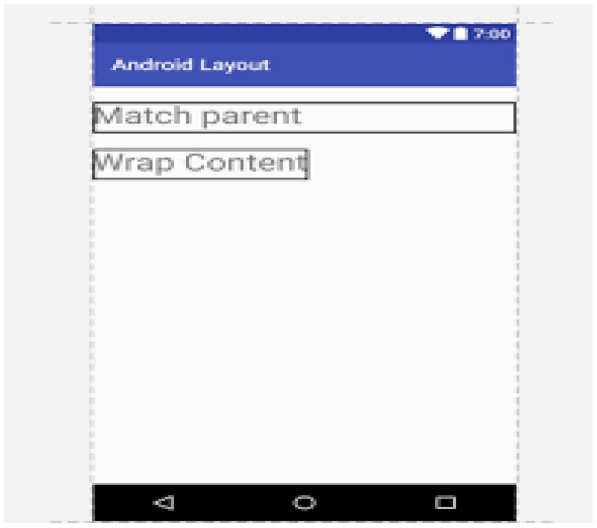
- **android:layout_width**—Used for defining the width of a control
- **android:layout_height**—Used for defining the height of a control
- **android:padding**—Used for increasing the whitespace between the boundaries of the control and its actual content
- **android:layout_weight**—Used for shrinking or expanding the size of the control to consume the extra space relative to the other controls in the container
- **android:gravity**—Used for aligning content within a control
- **android:layout_gravity**—Used for aligning the control within the container

Applying the orientation Attribute:

- This specifies the direction of arrangement and you will use android:orientation=”horizontal" for a row, android:orientation=”vertical" for a column.
- The default is horizontal.
- The orientation can be modified at runtime through the setOrientation() method. That is, by supplying the values HORIZONTAL or VERTICAL to the setOrientation() method.

Applying the height and width Attributes:

1. The default height and width of a control are decided on the basis of the text or content that is displayed through it.
2. To specify a certain height and width to the control, we use the android:layout_width and android:layout_height attributes.
    - By supplying specific dimension values for the control in terms of px (pixels), dip/dp (device independent pixels), sp (scaled pixels), pts (points), in (inches), and mm (millimeters).
    - By providing the value as wrap_content. When assigned to the control’s height or width, this attribute resizes the control to expand to fit its contents.
    - By providing the value as match_parent. When assigned to the control’s height or width, this attribute forces the size of the control to expand to fill up all the available space of the enclosing container.

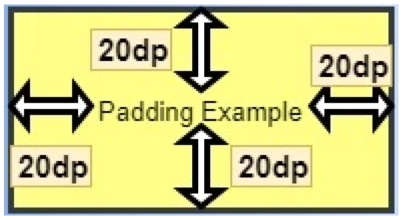


Applying the padding Attribute

The padding attribute is used to increase the whitespace between the boundaries of the control and its actual content.

Example:

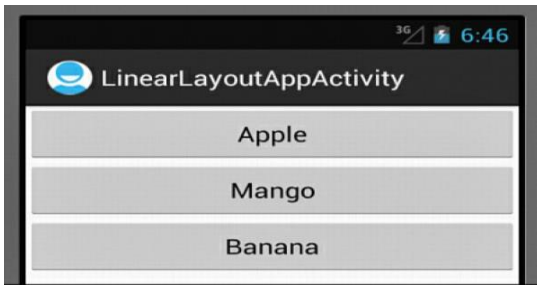
```
android:padingleft=”20dp”
android:paddingright=”20dp”
Aandroid:padingtop=”20dp”
android:padding=”20dp”
```



An Example Programs for linear layout by Adding Three Button Controls:

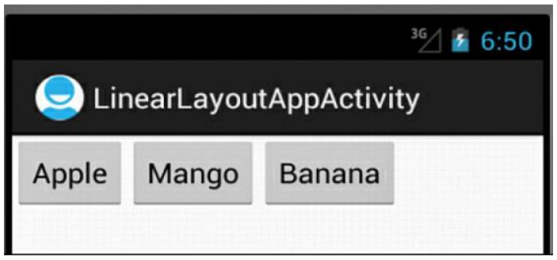
```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical" >
<Button
android:id="@+id/Apple"
android:text="Apple"
android:layout_width="match_parent"
android:layout_height="wrap_content" />
<Button
android:id="@+id/Mango"
android:text="Mango"
android:layout_width="match_parent"
android:layout_height="wrap_content" />
<Button
android:id="@+id/Banana"
android:text="Banana"
android:layout_width="match_parent"
android:layout_height="wrap_content" />
</LinearLayout>
```

Output:



The activity_linear_layout_app.xml File on Setting Horizontal.

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="horizontal" >
<Button
android:id="@+id/Apple"
android:text="Apple"
android:layout_width="wrap_content"
android:layout_height="wrap_content" />
<Button
android:id="@+id/Mango"
android:text="Mango"
android:layout_width="wrap_content"
android:layout_height="wrap_content" />
<Button
android:id="@+id/Banana"
android:text="Banana"
android:layout_width="wrap_content"
android:layout_height="wrap_content" />
</LinearLayout>
```



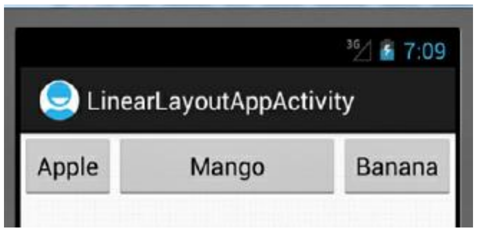
Applying weight Attribute

- The weight attribute affects the size of the control which is used to expand or shrink and consume extra space relative to the other controls in the container.
- The values of the weight attribute range from 0.0 to 1.0, where 1.0 is the highest value.

The activity_linear_layout_app.xml File on Applying the weight Attribute to the Button Controls:

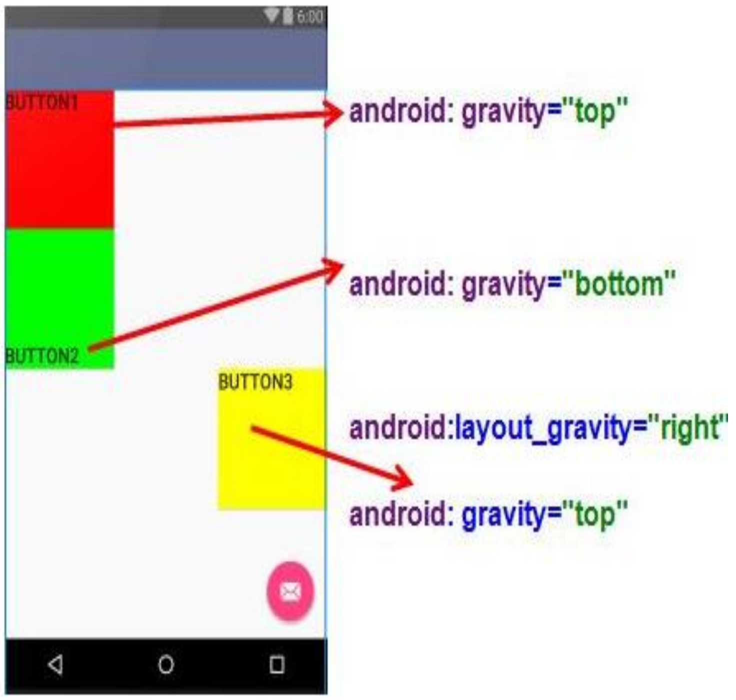
```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="horizontal"
android:layout_width="match_parent"
android:layout_height="match_parent">
<Button
android:id="@+id/Apple"
android:text="Apple"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
```

```
android:layout_weight="0.0" />
<Button
android:id="@+id/Mango"
android:text="Mango"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_weight="1.0" />
<Button
android:id="@+id/Banana"
android:text="Banana"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_weight="0.0" />
</LinearLayout>
```



Applying the Gravity Attribute:

- The Gravity attribute is for aligning the content within a control.
- To align the text of a control to the center, we set the value of its android:gravity attribute to center.
- The valid options for android:gravity include left, center, right, top, bottom, center_horizontal, center_vertical, fill_horizontal, and fill_vertical.
- To align the Button control itself in the LinearLayout (the container), we use the android:layout_gravity attribute.



Relative layout:

Android RelativeLayout enables you to specify how child views are positioned relative to each other. The position of each view can be specified as relative to sibling elements or relative to the parent.

Attributes of Relative layout:

The attributes used to set the location of the control relative to a container are
• **android:layout_alignParentTop**—The top of the control is set to align with the top of the container.
• **android:layout_alignParentBottom**—The bottom of the control is set to align with the bottom of the container.
• **android:layout_alignParentLeft**—The left side of the control is set to align with the left side of the container.
• **android:layout_alignParentRight**—The right side of the control is set to align with the right side of the container.
• **android:layout_centerHorizontal**—The control is placed horizontally at the center of the container.
• **android:layout_centerVertical**—The control is placed vertically at the center of the container.
• **android:layout_centerInParent**—The control is placed horizontally and vertically at the center of the container.
The attributes to control the position of a control in relation to other controls are
• **android:layout_above**—The control is placed above the referenced control.
• **android:layout_below**—The control is placed below the referenced control.
• **android:layout_toLeftOf**—The control is placed to the left of the referenced control.
• **android:layout_toRightOf**—The control is placed to the right of the referenced control.

The attributes that control the alignment of a control in relation to other controls are
• android:layout_alignTop—The top of the control is set to align with the top of the referenced control.
• **android:layout_alignBottom**—The bottom of the control is set to align with the bottom of the referenced control.
• **android:layout_alignLeft**—The left side of the control is set to align with the left side of the referenced control.
• **android:layout_alignRight**—The right side of the control is set to align with the right side of the referenced control.
• **android:layout_alignBaseline**—The baseline of the two controls will be aligned.

For spacing, Android defines two attributes: android:layout_margin and android:padding. The android:layout_margin attribute defines spacing for the container, while android:padding defines the spacing for the view. Let's begin with padding.
• **android:padding**—Defines the spacing of the content on all four sides of the control. To define padding for each side individually, use android:paddingLeft, android:paddingRight, android:paddingTop, and android:paddingBottom.

• **android:layout_margin**—Defines the spacing of the control in relation to the controls or the container on all four sides. To define spacing for each side individually, we use the android:layout_marginLeft, android:layout_marginRight, android:layout_marginTop, and android:layout_marginBottom options.

• **android:layout_marginTop**—Defines the spacing between the top of the control and the related control or container.

• **android:layout_marginBottom**—Defines the spacing between the bottom of the control and the related control or container.

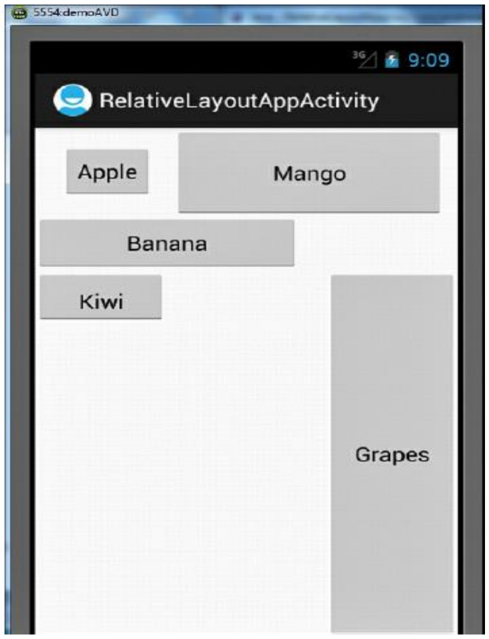
• **android:layout_marginRight**—Defines the spacing between the right side of the control and the related control or container.

• **android:layout_marginLeft**—Defines the spacing between the left side of the control and the related control or container.

The activity_relative_layout_app.xml File on Arranging the Button Controls in the RelativeLayout Container

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
android:layout_width="match_parent"
android:layout_height="match_parent">
<Button
android:id="@+id/Apple"
android:text="Apple"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_marginTop="15dip"
android:layout_marginLeft="20dip" />
<Button
android:id="@+id/Mango"
android:text="Mango"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:padding="28dip"
android:layout_toRightOf="@id/Apple"
android:layout_marginLeft="15dip"
android:layout_marginRight="10dip"
android:layout_alignParentTop="true" />
<Button
android:id="@+id/Banana"
android:text="Banana"
android:layout_width="200dip"
android:layout_height="50dip"
android:layout_marginTop="15dip"
android:layout_below="@id/Apple"
android:layout_alignParentLeft="true" />
<Button
android:id="@+id/Grapes"
android:text="Grapes"
```

```
android:layout_width="wrap_content"
android:layout_height="match_parent"
android:minWidth="100dp"
android:layout_alignParentRight="true"
android:layout_below="@id/Banana" />
<Button
android:id="@+id/Kiwi"
android:text="Kiwi"
android:layout_width="100dip"
android:layout_height="wrap_content"
android:layout_below="@id/Banana"
android:paddingTop="15dip"
android:paddingLeft="25dip"
android:paddingRight="25dip" />
</RelativeLayout>
```



<u>Absolute layout:</u>

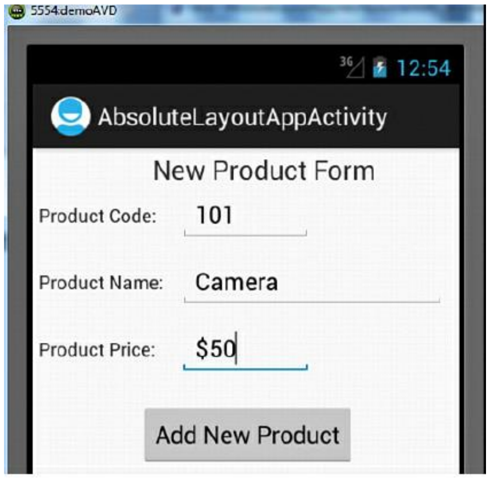
Each child in an AbsoluteLayout is given a specific location within the bounds of the container. The controls in AbsoluteLayout are laid out by specifying their exact *X* and *Y* positions.The coordinate 0,0 is the origin and is located at the top-left corner of the screen.

The Layout File activity_absolute_layout_app.xml on Arranging Controls in the AbsoluteLayout Container

```
<AbsoluteLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
android:layout_width="match_parent"
android:layout_height="match_parent">
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="New Product Form"
android:textSize="20sp"
```

```
android.textStyle="bold"
android:layout_x="90dip"
android:layout_y="2dip"/>
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Product Code:"
android:layout_x="5dip"
android:layout_y="40dip" />
<EditText
android:id="@+id/product_code"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:minWidth="100dip"
android:layout_x="110dip"
android:layout_y="30dip" />
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Product Name:"
android:layout_x="5dip"
android:layout_y="90dip"/>
<EditText
android:id="@+id/product_name"
android:layout_width="200dip"
android:layout_height="wrap_content"
android:minWidth="200dip"
android:layout_x="110dip"
android:layout_y="80dip"
android:scrollHorizontally="true" />
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Product Price:"
android:layout_x="5dip"
android:layout_y="140dip" />
<EditText
android:id="@+id/product_price"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:minWidth="100dip"
android:layout_x="110dip"
android:layout_y="130dip" />
<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:id="@+id/click_btn"
```

```
android:text="Add New Product"
android:layout_x="80dip"
android:layout_y="190dip" />
</AbsoluteLayout>
```



New Product Form

Product Code: 101

Product Name: Camera

Product Price: $50

Using Image view:

An ImageView control is used to display images in Android applications. An image can be displayed by assigning it to the ImageView control and including the android:src attribute in the XML definition of the control.

Attributes:

android:src—Used to assign the image from drawable resources.

Example:

android:src = "@drawable/bintupic"

You do not need to specify the image file extension. JPG and GIF files are supported, but the preferred image format is PNG.

android:scaleType—Used to scale an image to fit its container. The valid values for this attribute include fitXY, center, centerInside, and fitCenter.

fitXY - scales the image around the X and Y axes to match the size of container.

center- centers the image in the container without scaling it.

centerInside - scales the image uniformly, maintaining the aspect ratio so that the width and height of the image fit the size of its container.

fitCenter- scales the image while maintaining the aspect ratio, so that one of its X or Y axes fits the container.

android:adjustViewBounds—If set to true, the attribute adjusts the bounds of the ImageView control

android:resizeMode—The resizeMode attribute is used to make a control resizable so we can resize it horizontally, vertically, or around both axes.

FrameLayout:

- FrameLayout is used to display a single View.
- The View added to a FrameLayout is placed at the top left edge of the layout, that is views are stacked on top of each other.
- You can add many views or viewGroups to the frame layout
- To display images in Android applications, the image is first copied into the res/drawable folder and from there, it is referred to in the layout and other XML files.

The Layout File activity_frame_layout_app.xml on Arranging the ImageView in the FrameLayout Container

```
<FrameLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
android:layout_width="match_parent"
android:layout_height="match_parent">
<ImageView
android:id="@+id/imageview"
android:src = "@drawable/p2"
android:layout_width="wrap_parent"
android:layout_height="wrap_parent"
android:scaleType="fitXY" />
<ImageView
android:id="@+id/imageview2"
android:src = "@drawable/p3"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:scaleType="fitXY"
android:visibility=”gone”/>
</Frame Layout>
```

Code Written in the Java Activity File FrameLayoutAppActivity.java

```
package com.androidunleashed.framelayoutapp;
import android.app.Activity;
import android.os.Bundle;
import android.widget.ImageView;
import android.view.Menu;
import android.view.View;
public class FrameLayoutAppActivity extends Activity implements view.onClickListener {
ImageView i1,i2;
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_frame_layout_app);
i1=(ImageView) findviewbyId(R.id.imageview);
i2=(ImageView) findviewbyId(R.id.imageview2);
i1.setonClickListener(this);
i2.setonClickListener(this);
}
Public void onclick(view view)  {
      If(view.getId()==R.id.imageview)
          {
                  i1.setvisibility(View.gone);
                  i2.setvisibility(View.visible);
          }
   else
```

```
        {
                i2.setvisibility(View.gone);
               i1.setvisibility(View.visible);
        }
}
```

Table layout:

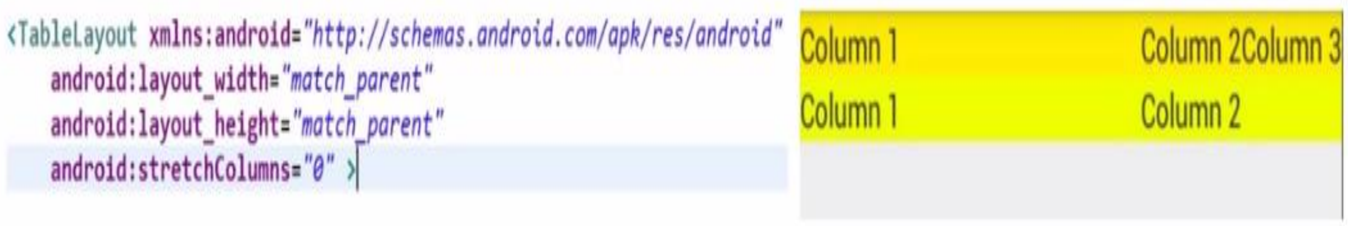
- The TableLayout is used for arranging the enclosed controls into rows and columns.
- Each new row in the TableLayout is defined through a TableRow object.
- A row can have zero or more controls, where each control is called a cell.
- The number of columns in a TableLayout is determined by the maximum number of cells in any row.

Operations applicable to Table layout:

1. Stretching column:

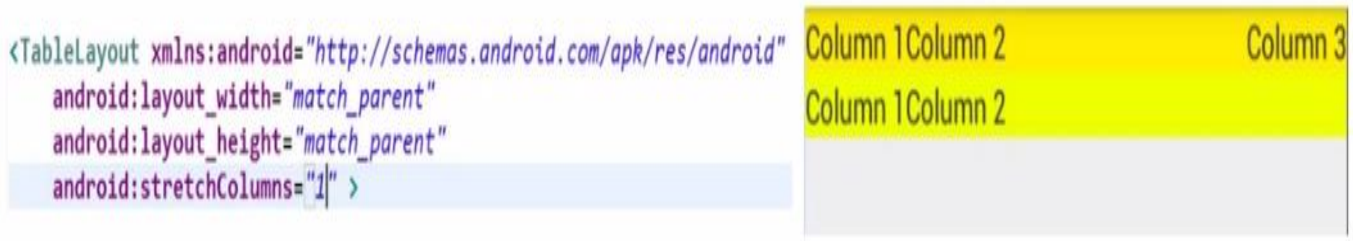
android:stretchcolumns: stretch either one column or a set of columns or all columns..(column indices start with 0).It can have the values like:

- android:stretchcolumns=”0”- The first column is stretched
- android:stretchColumns="1"-The second column is stretched .
- android:stretchColumns="0,1"-Both the first and second columns are stretched.
- android:stretchColumns="*"-All columns are stretched to take up the available space.



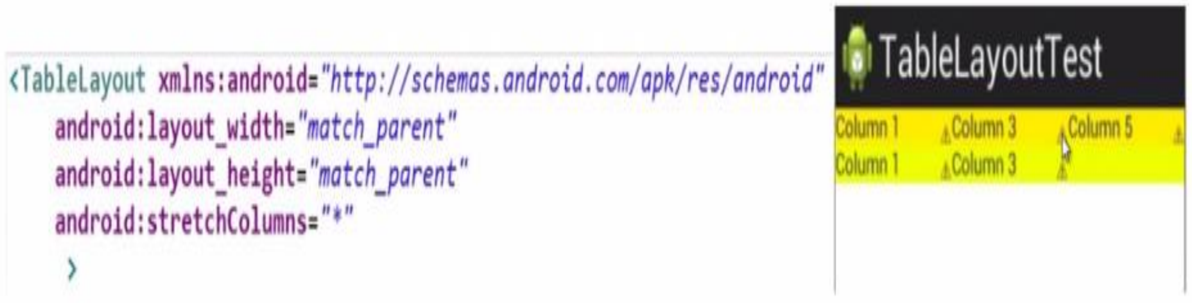
```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:stretchColumns="0" >
```

```
Column 1                 Column 2Column 3
Column 1                 Column 2
```



```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:stretchColumns="1" >
```

```
Column 1Column 2                  Column 3
Column 1Column 2
```



```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:stretchColumns="*"
    >
```

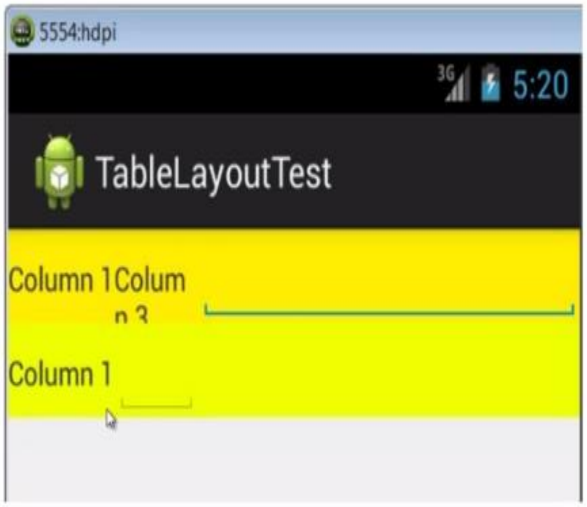
```
Column 1     Column 3     Column 5
Column 1     Column 3
```

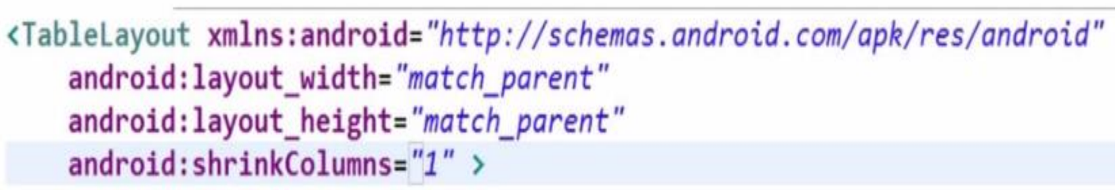
2. Shrinkingcolumns:

android:shrinkcolumns: Removes unnecessary extra space from a column and shrink it.

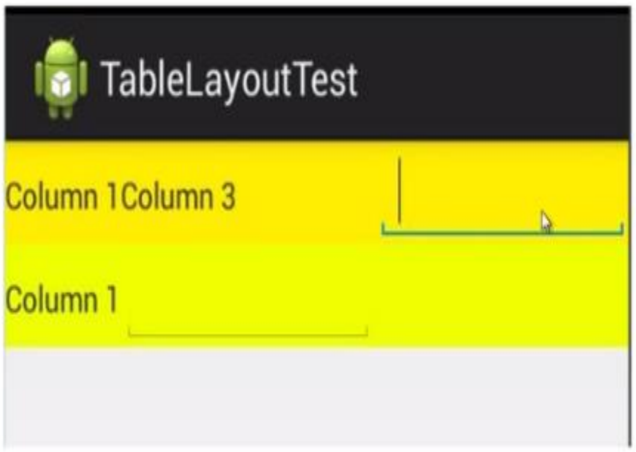
android:shrinkColumns="0"—The first column’s width shrinks or reduces by wordwrapping its content.

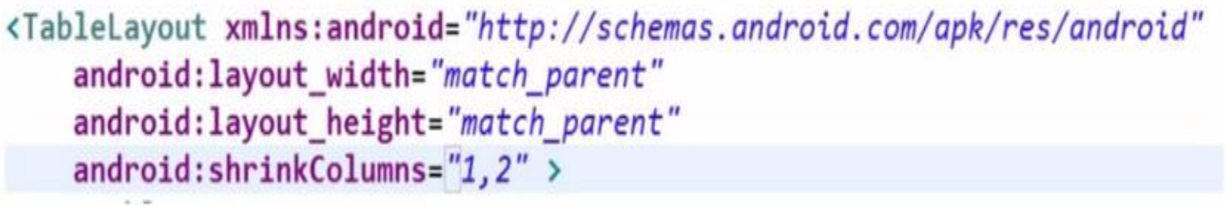
android:shrinkColumns="*"—The content of all columns is word-wrapped to shrink their widths.





```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:shrinkColumns="1" >
```



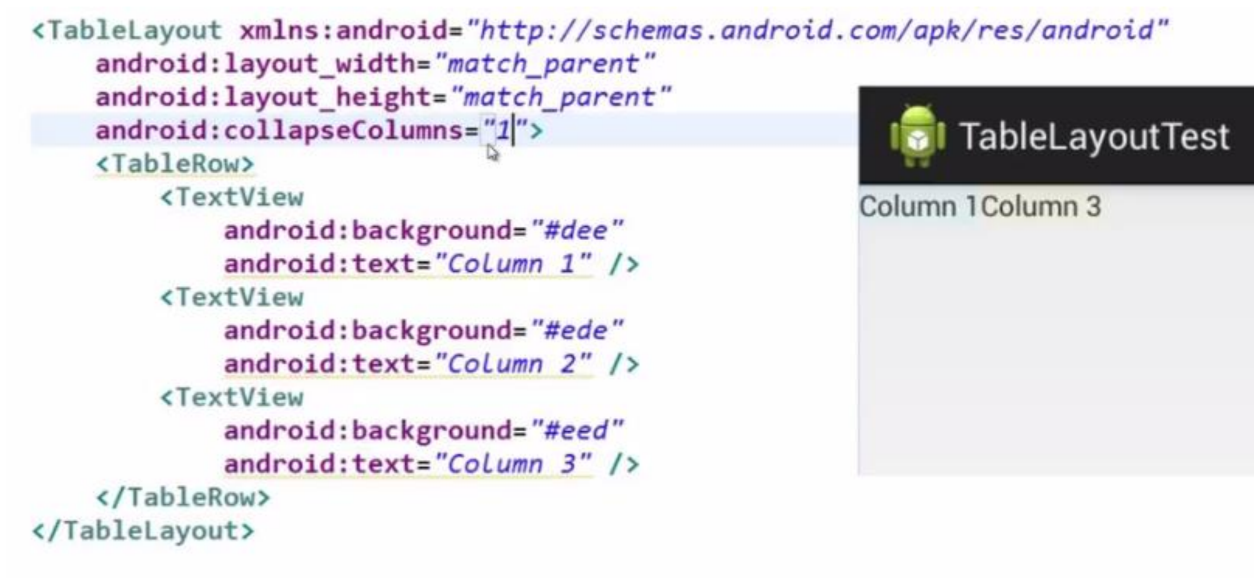


```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:shrinkColumns="1,2" >
```

3. Collapsing columns:

android:collapsecolumns- The idex of the column that you want to hide

android:collapseColumns="0"—The first column appears collapsed; that is, it is part of the table but is invisible. It can be made visible through coding by using the setColumnCollapsed() method.



```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:collapseColumns="1">
    <TableRow>
        <TextView
            android:background="#dee"
            android:text="Column 1" />
        <TextView
            android:background="#ede"
            android:text="Column 2" />
        <TextView
            android:background="#eed"
            android:text="Column 3" />
    </TableRow>
</TableLayout>
```

Grid layout:

GridLayout lays out views in a two-dimensional grid pattern, that is, in a series of rows and columns. The intersection of row and column is known as a grid cell, and it is the place where child views are placed. It is easier to use GridLayout when compared to TableLayout. Without specifying intermediate views. More than one view can be placed in a grid cell.

Specifying Row and Column Position

The two attributes that are used to specify the row and column position of the grid cell for inserting views are android:layout_row and android:layout_column.

Ex:

android:layout_row="0"

android:layout_column="0"

Spanning Rows and Columns

Views can span rows or columns if desired. The attributes used for doing so are android:layout_rowSpan and android:layout_columnSpan.

Ex:

android:layout_rowSpan="2"

android:layout_columnSpan="3"

Inserting Spaces in the GridLayout

For inserting spaces, a spacing view called Space is used. That is, to insert spaces, the Space view is inserted as a child view.

Ex:

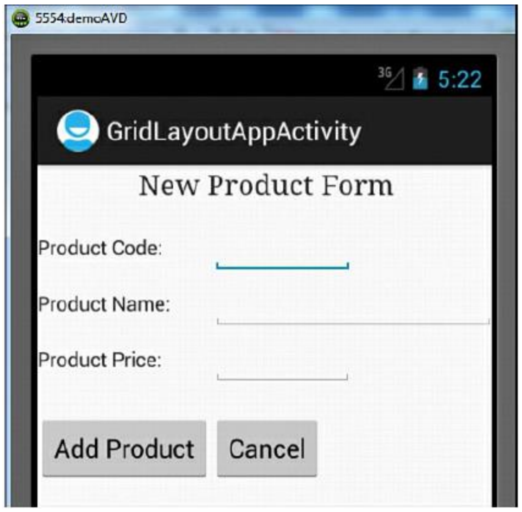
```
<Space
android:layout_row="1"
android:layout_column="0"
android:layout_width="50dp"
android:layout_height="10dp" />
```

The Layout File activity_grid_layout_app.xml on Arranging Controls in a GridLayout Container

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

```
android:layout_height="match_parent"
android:orientation="horizontal"
android:rowCount="7"
android:columnCount="2" >
<TextView
android:layout_row="0"
android:layout_column="0"
android:text="New Product Form"
android:typeface="serif"
android:layout_columnSpan="2"
android:layout_gravity="center_horizontal"
android:textSize="20dip" />
<Space
android:layout_row="1"
android:layout_column="0"
android:layout_width="50dp"
android:layout_height="10dp" />
<TextView
android:layout_row="2"
android:layout_column="0"
android:text="Product Code:" />
<EditText
android:id="@+id/prod_code"
android:layout_width="100dip" />
<TextView
android:text="Product Name:" />
<EditText
android:layout_row="3"
android:layout_column="1"
android:id="@+id/prod_name"
android:layout_width="200dip" />
<TextView
android:layout_row="4"
android:layout_column="0"
android:text="Product Price:" />
<EditText
android:layout_row="4"
android:layout_column="1"
android:id="@+id/prod_price"
android:layout_width="100dip" />
<Space
android:layout_row="5"
android:layout_column="0"
android:layout_width="50dp"
android:layout_height="20dp" />
<Button
android:layout_row="6"
```

```
android:layout_column="0"
android:id="@+id/add_button"
android:text="Add Product" />
<Button
android:id="@+id/cancel_button"
android:text="Cancel" />
</GridLayout>
```



Adapting to Screen Orientation:

Android supports two screen orientations: portrait and landscape. Portrait mode is longer in height and smaller in width, whereas landscape mode is wider but smaller in height.

There are two ways to handle changes in screen orientation:

• **Anchoring controls**—Set the controls to appear at the places relative to the four edges of the screen.

Example:

The Layout file activity_screen_orientation_app.xml on Laying Out Controls Relative to the Four Edges of the Screen

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
android:layout_width="match_parent"
android:layout_height="match_parent">
<Button
android:id="@+id/Apple"
android:text="Apple"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_marginTop="15dip"
android:layout_marginLeft="20dip" />
<Button
android:id="@+id/Mango"
android:text="Mango"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:padding="28dip"
android:layout_toRightOf="@id/Apple"
```

```
android:layout_marginLeft="15dip"
android:layout_marginRight="10dip"
android:layout_alignParentTop="true" />
<Button
android:id="@+id/Banana"
android:text="Banana"
android:layout_width="200dip"
android:layout_height="50dip"
android:layout_marginTop="15dip"
android:layout_below="@id/Apple"
android:layout_alignParentLeft="true" />
<Button
android:id="@+id/Grapes"
android:text="Grapes"
android:layout_width="wrap_content"
android:layout_height="match_parent"
android:minWidth="100dp"
android:layout_alignParentRight="true"
android:layout_below="@id/Banana" />
<Button
android:id="@+id/Kiwi"
android:text="Kiwi"
android:layout_width="100dip"
android:layout_height="wrap_content"
android:layout_below="@id/Banana"
android:paddingTop="15dip"
android:paddingLeft="25dip"
android:paddingRight="25dip" />
</RelativeLayout>
```

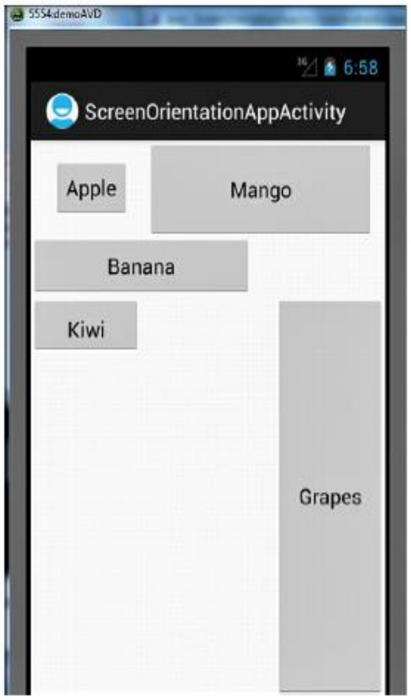
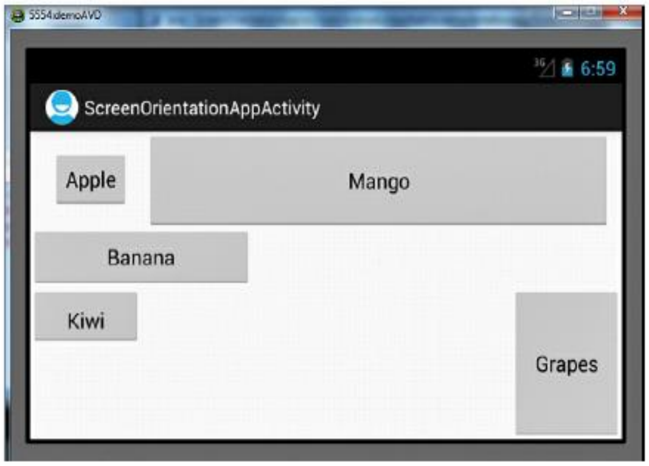


fig (left)



fig(right)

- When the application is run while in the default portrait mode, the controls appear as shown in above fig(left).
- When it is rotated to landscape mode the button controls are shown above fig(right).
- To switch between portrait mode and landscape mode on the device emulator, press the Ctrl+F11 keys.

Defining layout for each mode—A new layout file is defined for each of the two screen orientations,that is Potrait mode landscape.

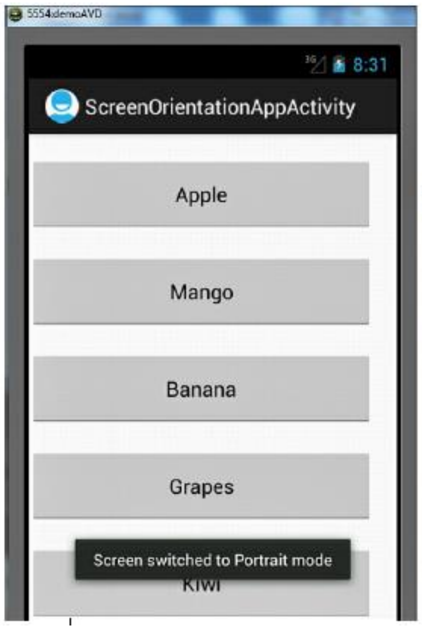
Example:

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
android:layout_width="match_parent"
android:layout_height="match_parent">
<Button
android:id="@+id/Apple"
android:text="Apple"
android:layout_width="250dp"
android:layout_height="wrap_content"
android:padding="20dip"
android:layout_marginTop="20dip" />
<Button
android:id="@+id/Mango"
android:text="Mango"
android:layout_width="250dp"
android:layout_height="wrap_content"
android:padding="20dip"
android:layout_marginTop="20dip"
android:layout_toRightOf="@id/Apple" />
<Button
android:id="@+id/Banana"
android:text="Banana"
android:layout_width="250dip"
android:layout_height="wrap_content"
android:padding="20dip"
android:layout_marginTop="20dip"
android:layout_below="@id/Apple" />
<Button
android:id="@+id/Grapes"
android:text="Grapes"
android:layout_width="250dip"
android:layout_height="wrap_content"
android:padding="20dip"
android:layout_marginTop="20dip"
android:layout_below="@id/Apple"
android:layout_toRightOf="@id/Banana" />
<Button
android:id="@+id/Kiwi"
android:text="Kiwi"
```

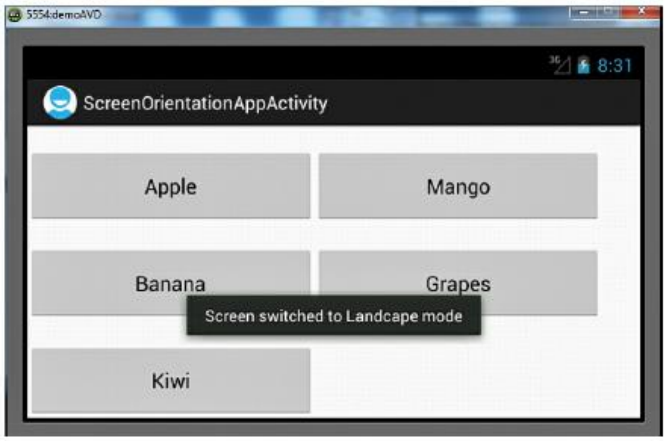
```
android:layout_width="250dip"
android:layout_height="wrap_content"
android:padding="20dip"
android:layout_marginTop="20dip"
android:layout_below="@id/Banana" />
</RelativeLayout>
```

Code Written in the Java Activity File:

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.Toast;
public class ScreenOrientationAppActivity extends Activity {
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_screen_orientation_app);
if(getResources().getDisplayMetrics().widthPixels>getResources().heightPixels)
{
Toast.makeText(this,"Screen switched to Landscape
mode",Toast.LENGTH_SHORT).
show();
}
else
{
Toast.makeText(this,"Screen switched to Portrait
mode",Toast.LENGTH_SHORT).
show();
}
}
}
```



Screen switched to Portrait mode

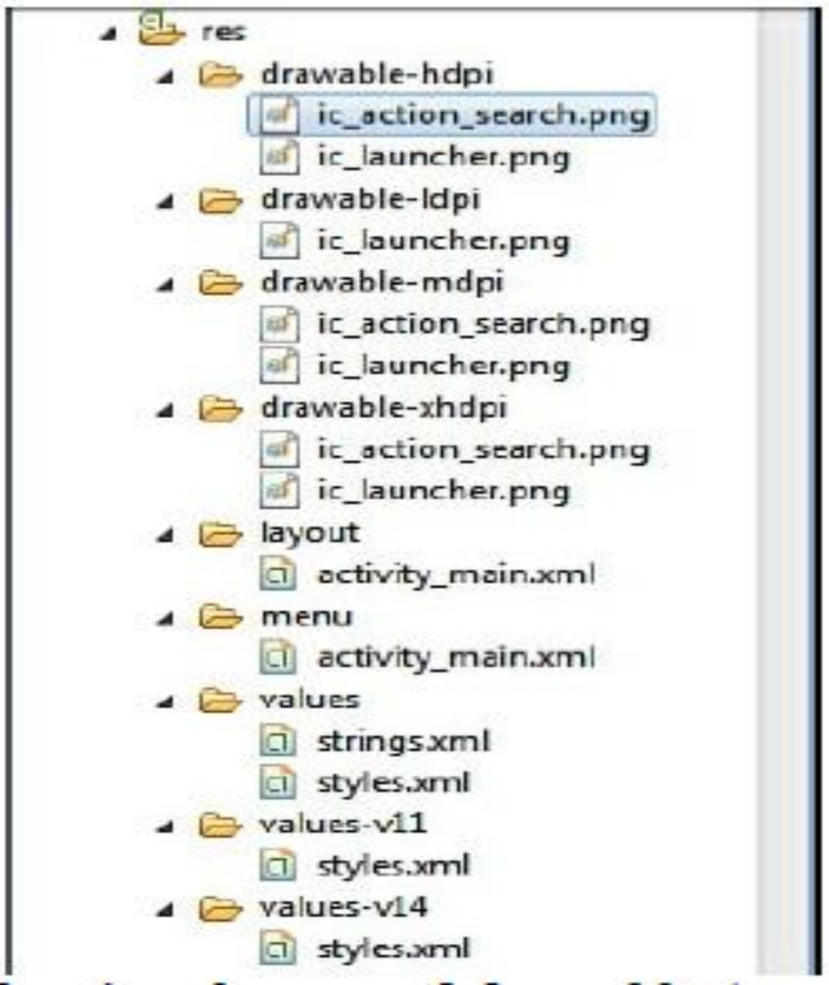


Screen switched to Landcape mode

Resources:

- Resources includes textdata, bitmaps, audio, video files and other items which is used by android application.
- It is also refers to the external files which holds the information such as strings, layouts, images.
- This Resources are kept separately in XML files and it is accessed in Java code through the id's assigned to them.
- Resources are broadly divided into three categories—values, drawable, and layout which is stored in the res folder of our project hierarchy.

The res folder showing the nested drawable (drawable-hdpi, drawableldpi, drawable-mdpi) layouts and values folders and their content



Types of Resources:

- **Drawable folder:**

  1. This Drawable folder is used to store the images in our application.
  2. Depending on the target platform chosen, our application can have either a single directory, drawable, or four directories such as:
  drawable-ldpi- low resolution images.lpi stands for low dots per inch
  drawable-mdpi- medium resolution images.mdpi stands for medium dots per inch.
  drawable-hdpi- high resolution images.hdpi stands for high dots per inch
  drawable-xhdpi-extrahigh rsolution images.xhdpi stands for extrahigh dots per inch.
  4. If our application has a single directory, drawable, then the images to be used in our application.

3. If our applications have four directories then the images with different screen resolutions are stored in respective folders.

- **Layout folder:**
  1. This folder contains automatic layout file created for us.
  2. The default name assigned to this file is activity_mai.xml, but we can assign any name.
- **Menu folder:**
  1. This folder contains XML file(s) that represent application menus.
  2. The default name assigned to this file is activity_mai.xml, but we can assign any name.
- **Values folder :**
  1. This folder by default contains a strings.xml file that we can use to define values resources and it also include strings, colors, dimensions, styles, and string or integer arrays.
  2. We can also create individual XML files for each of these resources instead.
  3. The following is a list of some XML files that we can create in the values folder:
     1.arrays.xml
     2.Colors.xml
     3.dimens.xml
     4.Strings.xml
     5. Styles.xml
- **values-v11-**This folder contains the styles.xml file that declares the holographic theme, which is used when the application runs on Android 3.0 (API Level 11) or higher.
- **values-v14-**This folder contains the styles.xml file that declares the DeviceDefault theme, which is used when the application runs on Android 4.0 (API Level 14) or higher.

Creating Values Resources

- The resources in the values directory include different types, such as strings, colors, dimensions, and string or integer arrays. All the values are stored in XML files in the res/values folder.
- It is preferred to have a separate XML file for each type of resource in the values directory.
- The filename can be anything and it should contain only lowercase letters, numbers, period (.), and underscore (_)
- We can have any number of XML files to represent resources.
- We should not save resource files directly inside the res/ folder because it will cause a compiler error.

String Resources:

Default Code in the strings.xml File:

```
<resources>
<string name="app_name">ValuesResourcesApp</string>
<string name="hello_world">Hello world!</string>
<string name="menu_settings">Settings</string>
<string
    name="title_activity_values_resources_app">ValuesResourcesAppActivity</
    </resources>
```

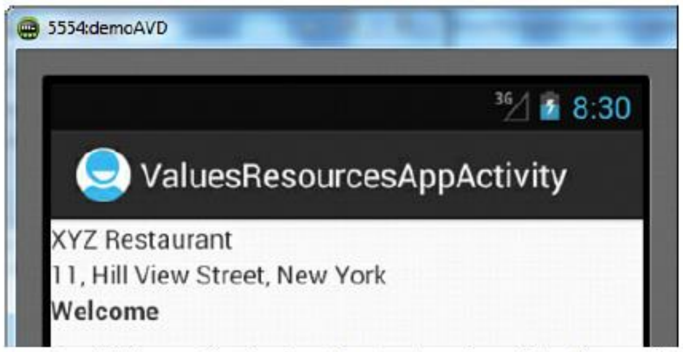
- <resources> is a root element followed by one or more child elements.
- Each <string> element represents one string resource that contains app_name, str_name, str_address, and str_message.

Code Written in the Layout File activity_values_resources_app.xml

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
android:layout_width="match_parent"
android:layout_height="match_parent" >
<TextView
android:id="@+id/name_view"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="@string/str_name" />
<TextView
android:id="@+id/address_view"
android:layout_width="match_parent"
android:layout_height="wrap_content" />
<TextView
android:id="@+id/message_view"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="@string/str_message" />
    </LinearLayout>
```

Code Written in the Java Activity File ValuesResourcesAppActivity.java

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
public class ValuesResourcesAppActivity extends Activity {
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_values_resources_app);
String strAddress=getString(R.string.str_address);
TextView addressView=
(TextView)findViewById(R.id.address_view);
addressView.setText(strAddress);
}
}
```



XYZ Restaurant
11, Hill View Street, New York
Welcome

. The three **TextViews** displaying the text assigned to them via String resource and Java code

Dimensions.xml:

Dimension resources are used for standardizing certain application measurements. These resources can be used to specify the sizes for fonts, layouts, and widgets. dimension resource is defined through the dimen element. The dimen element contains a numerical value followed by unit of measurement such as:

- **px**—Pixels
- **in**—Inches
- **mm**—Millimeters
- **pt**—Points
- **dp**—Density-independent pixels based on a 160-dpi (pixel density per inch) screen
- **sp**—Scale-independent pixels

To try out dimension resources, let’s open the application ValuesResourcesApp and add an XML file called dimens.xml to the values folder.

Example:

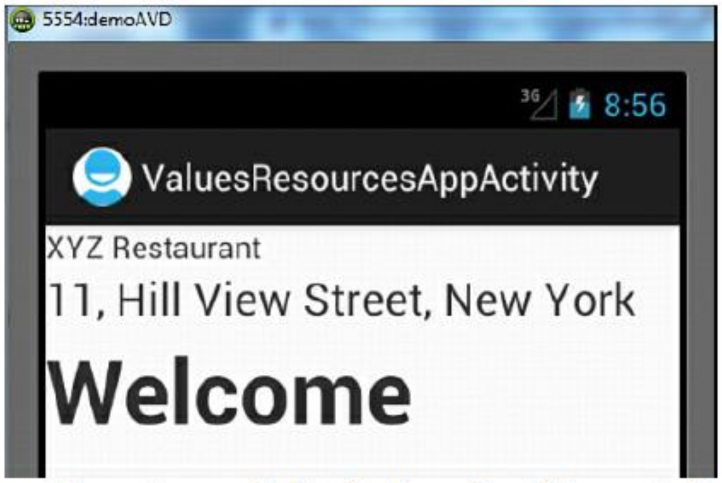
Code Written in the dimens.xml File

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<dimen name="small_size">15dp</dimen>
<dimen name="medium_size">15sp</dimen>
<dimen name="large_size">20pt</dimen>
</resources>
```

The Layout File activity_values_resources_app.xml on Applying Dimension Resources to TextView Controls

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
android:layout_width="match_parent"
android:layout_height="match_parent" >
<TextView
android:id="@+id/name_view"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="@string/str_name"
android:textSize="@dimen/small_size" />
<TextView
android:id="@+id/address_view"
android:layout_width="match_parent"
android:layout_height="wrap_content" />
<TextView
android:id="@+id/message_view"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="@string/str_message"
android:textSize="@dimen/large_size" />
</LinearLayout>
```

To apply the dimension resource medium_size to our TextView address_view, add these statements to the Java file ValuesResourcesAppActivity.java:
float addressSize=this.getResources().getDimension(R.dimen.medium_size);
addressView.setTextSize(addressSize);



XYZ Restaurant
11, Hill View Street, New York
Welcome

. **Different dimensions applied to the three `TextViews` via dimension resources**

Color Resources:

To define a color resource, we use the color element.
The color value is specified in the form of hexadecimal RGB values preceded by an optional Alpha channel.
The hex values formats are:
• **#RGB**—For example, #F00 for a Red color.
• **#RRGGBB**—For example, #FF0000 for a Red color
• **#ARGB**—For example, #5F00 for a Red color with an Alpha of 5.
• **#AARRGGBB**—For example, #50FF0000 for a Red color with an Alpha of 50.
To apply color resources to our application ValuesResourcesApp, let's add an XML file called colors.xml to the values folder.
Example:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<color name="red_color">#F00</color>
<color name="green_color">#00FF00</color>
<color name="blue_alpha_color">#500000FF</color>
</resources>
```

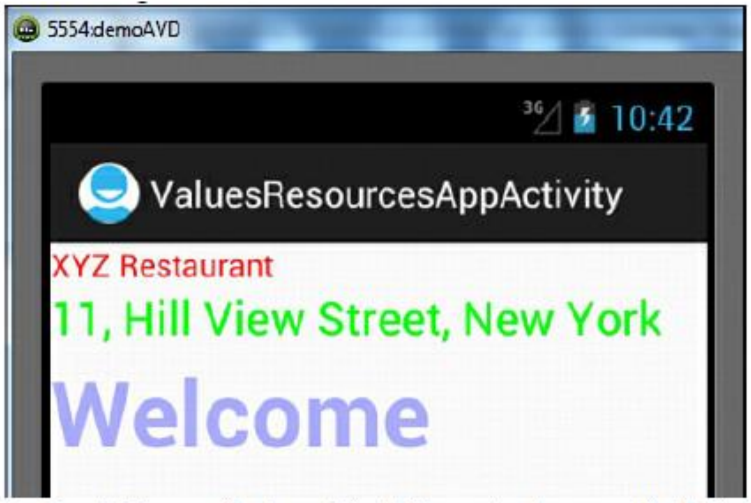
The Layout File activity_values_resources_app.xml on Applying Color Resources

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
android:layout_width="match_parent"
android:layout_height="match_parent" >
<TextView
android:id="@+id/name_view"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="@string/str_name"
android:textSize="@dimen/small_size"
android:textColor="@color/red_color"/>
```

```
<TextView
android:id="@+id/address_view"
android:layout_width="match_parent"
android:layout_height="wrap_content" />
<TextView
android:id="@+id/message_view"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="@string/str_message"
android:textSize="@dimen/large_size"
android:textColor="@color/blue_alpha_color"/>
</LinearLayout>
```

The Code Written in the Java Activity File ValuesResourcesAppActivity.java

```
package com.androidunleashed.valuesresourcesapp;
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
public class ValuesResourcesAppActivity extends Activity {
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_values_resources_app);
String strAddress=getString(R.string.str_address);
TextView addressView=
(TextView)findViewById(R.id.address_view);
addressView.setText(strAddress);
float addressSize=
this.getResources().getDimension(R.dimen.medium_size);
addressView.setTextSize(addressSize);
int
addressColor=this.getResources().getColor(R.color.green_color);
addressView.setTextColor(addressColor);
}
}
```



XYZ Restaurant
11, Hill View Street, New York
Welcome

The three **TextViews** displayed in different colors applied via the Color Resource. The **TextView** controls are distinguished through different sizes.

Styles and themes:

- A style is a collection of attributes such as color, font, margin, and padding that we can apply collectively to our Views, Activity, or an entire application.
- A style is created by using a style element with one or more item child elements.
- Each item element includes a name property to define the attributes such as color, font, and so on. An item element defines an attribute and its value

Example:

```
<resources>
<style name="resource_ID">
<item name="attribute_name">value </item>
</style>
</resources>
```

The Code Written in the styles.xml File

```
<resources>
<style name="AppTheme" parent="android:Theme.Light" />
<style name="style1">
<item name="android:textColor">#00FF00 </item>
<item name="android:typeface">serif</item>
<item name="android:textSize">30sp </item>
</style>
<style name="style2" parent="style1" >
<item name="android:textColor">#0000FF</item>
<item name="android:typeface">sans</item>
<item name="android:background">#FF0000</item>
<item name="android:padding">10dip</item>
</style>
<style name="style3" parent="style2" >
<item name="android:textColor">#00FF00</item>
<item name="android:background">#00000000</item>
<item name="android:typeface">monospace</item>
<item name="android:gravity">center</item>
</style>
</resources>
```

The Layout File activity_values_resources_app.xml on Applying Styles to TextView Controls

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
android:layout_width="match_parent"
android:layout_height="match_parent">
<TextView
android:id="@+id/name_view"
style="@style/style1"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="@string/str_name"/>
<TextView
```

```
android:id="@+id/address_view"
style="@style/style2"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="@string/str_address" />
<TextView
android:id="@+id/message_view"
style="@style/style3"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="@string/str_message" />
</LinearLayout>
```

Arrays

Arrays refer to a collection of values or elements As the name suggests, the string array provides an array of strings. Such a resource is popularly used with selection widgets such as ListView and Spinner that need to display a collection of selectable items to the user.

Code in the strings.xml File on Adding a String Array

```
<resources>
<string name="app_name">StringArrayApp</string>
<string name="menu_settings">Settings</string>
<string
name="title_activity_string_array_app">StringArrayAppActivity</string>
<string-array name="fruits">
<item>Apple</item>
<item>Mango</item>
<item>Orange</item>
<item>Grapes</item>
<item>Banana</item>
</string-array>
</resources>
```

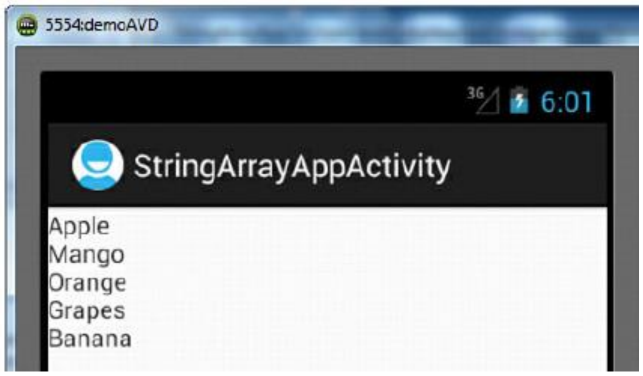
Code in the Layout File activity_string_array_app.xml on Adding the TextView Control

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:orientation="vertical"
android:layout_width="match_parent"
android:layout_height="match_parent">
<TextView
android:id="@+id/fruits_view"
android:layout_width="match_parent"
android:layout_height="wrap_content" />
</LinearLayout>
```

Code Written in the Java Activity File StringArrayAppActivity.java

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
public class StringArrayAppActivity extends Activity {
```

```
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_string_array_app);
TextView fruitsView =
(TextView)findViewById(R.id.fruits_view);
String[] fruitsArray =
getResources().getStringArray(R.array.fruits);
String str = "";
for(int i = 0; i < fruitsArray.length; i++){
str += fruitsArray[i] + "\n";
}
fruitsView.setText(str);
}
}
```



```
Apple
Mango
Orange
Grapes
Banana
```

Using Drawable resources:

- A drawable resource represents a graphic file or xml drawable file that can be drawn on screen. In Android, these files are stored in res/drawable folder.
- To prevent blurred images, drawable resources for different screen resolutions are provided in screen resolution specific drawable folders such as drawable-mdpi, drawable-hdpi, drawable-xhdpi and drawable-xxhdpi. Here are step by step how to use Drawable Resource XML in Android Studio.
- Firstly, Right Click on drawable which is present in the res folder
- Copy the images and paste it in the drawable folder.
- After we add images to the res/drawable folders, the gen folder is regenerated where the R.java file resides.
- The R.java file includes a reference to the newly added image and hence can be used in the layout file or other Java code. The syntax for referencing the image in the layout file :

@drawable/image_filename

In Java code, the image can be referenced using the following syntax:

R.drawable.image_filename

The Layout file activity_disp_image_app.xml

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:orientation="vertical"
android:layout_width="match_parent"
```

```
android:layout_height="match_parent">
<ImageView
android:id="@+id/image_toview"
android:layout_width="match_parent"
android:layout_height="match_parent"/>
</LinearLayout>
```

The Java Activity File DispImageAppActivity.java

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.ImageView;
public class DispImageAppActivity extends Activity {
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_disp_image_app);
ImageView image = (ImageView)
findViewById(R.id.image_toview);
image.setImageResource(R.drawable.bintupic);
}
}
```

Switching States with Toggle Buttons:

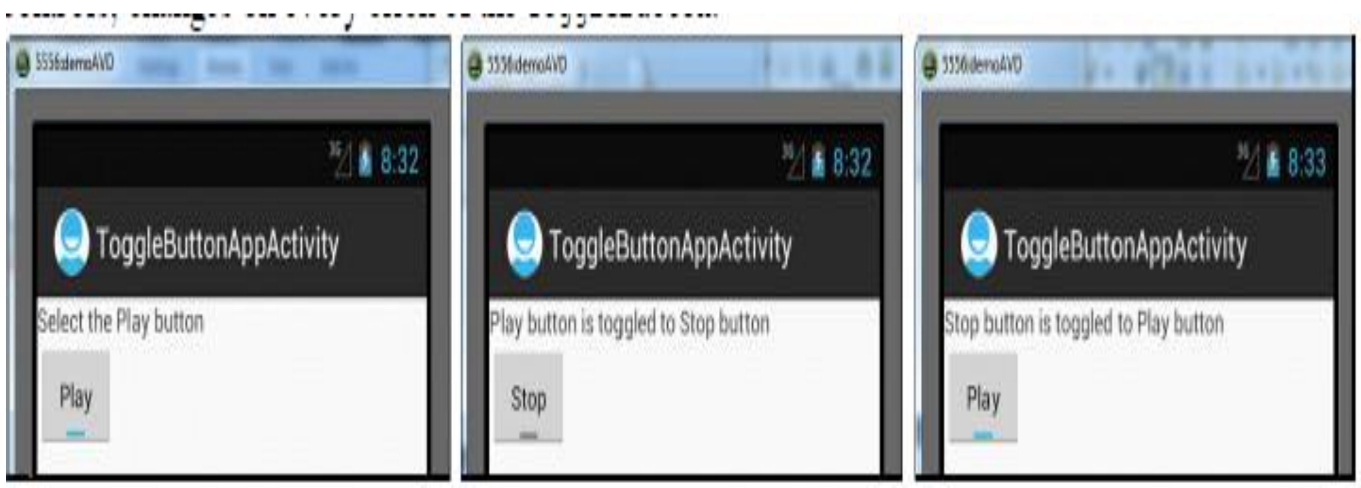
- In Android, ToggleButton is used to display checked and unchecked state of a button.
- ToggleButton allow the users to change the setting between two states like turn on/off your wifi, Bluetooth etc from your phone's setting menu.
- To display a ToggleButton, we use the <ToggleButton> tag in the layout file.
- To set the text for the button, the two attributes android:textOn and android:textOff are used.

Code in Layout File activity_toggle_button_app.xml on Adding the TextView and ToggleButton Controls

```
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:orientation="vertical"
android:layout_width="match_parent"
android:layout_height="match_parent">
<TextView
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="Select the Play button"
android:id="@+id/response"/>
<ToggleButton android:id="@+id/playstop_btn"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textOn="Play"
android:textOff="Stop"/>
</LinearLayout>
```

Code Written in the Java Activity File ToggleButtonAppActivity.java

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
import android.widget.ToggleButton;
import android.view.View.OnClickListener;
import android.view.View;
public class ToggleButtonAppActivity extends Activity {
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_toggle_button_app);
final TextView resp =
(TextView)this.findViewById(R.id.response);
final ToggleButton playStopButton = (ToggleButton)
findViewById(R.id.playstop_btn);
playStopButton.setChecked(true);
playStopButton.setOnClickListener(new OnClickListener()
{
public void onClick(View v) {
if (playStopButton.isChecked()) {
resp.setText("Stop button is toggled to Play
button");
}
else {
resp.setText("Play button is toggled to Stop
button");
}
}
});
}
```


Select the Play button

Play button is toggled to Stop button

Stop button is toggled to Play button

Creating image switcher application:

It is an element of transition widget which helps us to add transitions on the images. It is mainly useful to animate an image on screen. ImageSwitcher switches smoothly between two images and thus provides a way of transitioning from one Image to another through appropriate animations.

Code in the Layout File activity_disp_image_app.xml on Adding a ToggleButton and Assigning an Image to the ImageView Control

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:orientation="vertical"
android:layout_width="match_parent"
android:layout_height="wrap_content">
<ImageView
android:id="@+id/image_toview"
android:src="@drawable/bintupic"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:adjustViewBounds="true" />
<ToggleButton android:id="@+id/change_image"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textOn="Previous Image"
android:textOff="Next Image"
android:layout_gravity="center"
android:layout_marginTop="10dip" />
</LinearLayout>
```

Code Written in the Java Activity File DispImageAppActivity.java

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.ImageView;
import android.widget.ToggleButton;
import android.view.View;
import android.view.View.OnClickListener;
public class DispImageAppActivity extends Activity {
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_disp_image_app);
final ImageView image = (ImageView)
findViewById(R.id.image_toview);
final ToggleButton changeButton=
(ToggleButton)findViewById(R.id.change_image);
changeButton.setOnClickListener(new OnClickListener(){
public void onClick(View v){
```

```
if (changeButton.isChecked()) {
image.setImageResource(R.drawable.bintupic2);
}
else {
image.setImageResource(R.drawable.bintupic);
}
}
});
}
}
```

Scrolling through scrollview:

A ScrollView is a special type of control that sets up a vertical scrollbar in a View container. This control is used when we try to display Views that are too long to be accommodated in a single screen. ScrollView is a subclass of frame layout

```
<ScrollView
xmlns:android="http://schemas.android.com/apk/res/android"
android:id="@+id/scrollwid"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:fillViewport="true"
android:orientation="vertical" >
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical" >
<ImageView
android:id="@+id/image_toview"
android:src="@drawable/bm1"
android:layout_width="200dip"
android:layout_height="250dip"
android:layout_gravity="center" />
<ImageView
android:id="@+id/image_toview2"
android:src="@drawable/bm2"
android:layout_width="200dip"
android:layout_height="250dip"
android:layout_gravity="center"
android:layout_marginTop="10dip" />
<ImageView
android:id="@+id/image_toview3"
android:src="@drawable/bm3"
android:layout_width="200dip"
android:layout_height="250dip"
android:layout_gravity="center"
android:layout_marginTop="10dip" />
```

```
</LinearLayout>
</ScrollView>
```

Playing Audio:

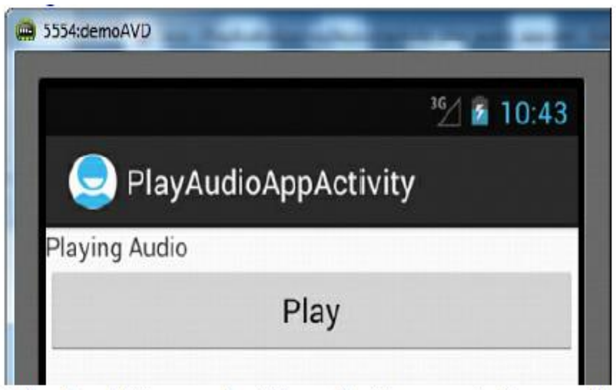
- The audio file that we want to play must be located in the res/raw folder of our application.
- To create raw folder Right-click the res folder in the Package Explorer window and select New, Folder.
- In the dialog box that opens, enter the name of the new folder as raw and click the Finish button.
- In the raw folder, let’s copy an audio file called song1.mp3.

The Layout File activity_play_audio_app.xml on Adding the TextView and Button Control

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:orientation="vertical"
android:layout_width="match_parent"
android:layout_height="match_parent">
<TextView
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="Playing Audio" />
<Button android:id="@+id/playbtn"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="Play" />
</LinearLayout>
```

Code Written in the Java Activity File PlayAudioAppActivity.java

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.Button;
import android.view.View;
import android.media.MediaPlayer;
public class PlayAudioAppActivity extends Activity {
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_play_audio_app);
Button playButton = (Button) findViewById(R.id.playbtn);
playButton.setOnClickListener(new
Button.OnClickListener() {
public void onClick(View v) {
MediaPlayer mp =
MediaPlayer.create(PlayAudioAppActivity.this,R.mp.start();
}
});
}
}
```



7. The TextView and a Play Button control on application startup

- We can control the volume of the audio with the volume switches on the side of the Android emulator.
- we can also display an image on the Button control by adding the following attributes to the <Button> element:

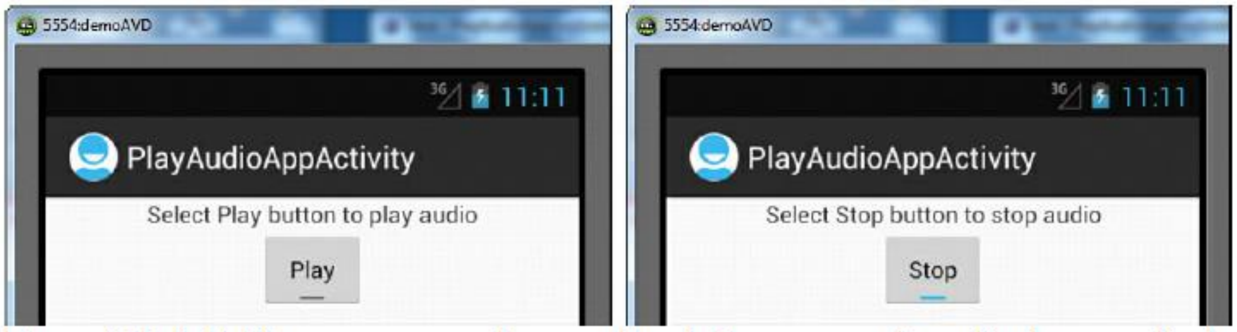
• **android:drawableTop**—The image is displayed above the button text.
Example:
android:drawableTop="@drawable/ic_launcher"
• **android:drawableBottom**—The image is displayed below the button text
• **android:drawableLeft**—The image is displayed to the left of the button text.
• **android:drawableRight**—The image is displayed to the right of the button text .
To switch the status of the audio from play to stop and vice versa, we replace the Button control with the ToggleButton control.

The Layout File activity_play_audio_app.xml on Replacing Button with the ToggleButton Control

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:orientation="vertical"
android:layout_width="match_parent"
android:layout_height="match_parent">
<TextView
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:gravity="center"
android:id="@+id/response"/>
<ToggleButton android:id="@+id/playstop_btn"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textOn="Stop"
android:textOff="Play"
android:layout_gravity="center" />
</LinearLayout>
```

Code Written in the Java Activity File PlayAudioAppActivity.java

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.ToggleButton;
import android.view.View;
import android.widget.TextView;
import android.media.MediaPlayer;
import android.view.View.OnClickListener;
public class PlayAudioAppActivity extends Activity {
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_play_audio_app);
final TextView response =
(TextView)this.findViewById(R.id.response);
response.setText("Select Play button to play audio");
final MediaPlayer mp =
MediaPlayer.create(PlayAudioAppActivity.this,R.raw.song1);
final ToggleButton playStopButton = (ToggleButton)
findViewById(R.id.playstop_btn);
playStopButton.setOnClickListener(new OnClickListener()
{
public void onClick(View v) {
if (playStopButton.isChecked()) {
response.setText("Select Stop button to
stop audio");
mp.start();
}
else {
response.setText("Select Play button to play
audio");
mp.pause();
}
}
});
}
}
```



Select Play button to play audio

Select Stop button to stop audio

Playing Video:

To play video in an application, Android provides a VideoView control, and it provides several buttons for controlling video play. The buttons are like pause, rewind, and fast-forward.
Like audio, the video doesn’t play when placed in the raw folder. Instead it needs to be placed in the SDCARD folder or we can play a video that is available on the Internet.

Loading Video onto an SD Card:

- An emulator must be running while loading video onto an SD Card. Switch on the emulator by selecting the Window, AVD Manager option. Select the demoAVD virtual device and select the Start button.
- We get a dialog box showing the Launch Options of the emulator. Select the Launch button to run the emulator.
- After you select the Launch button, the emulator starts.
- Close the AVD Manager dialog.

To load a video onto an SD card, follow these steps:

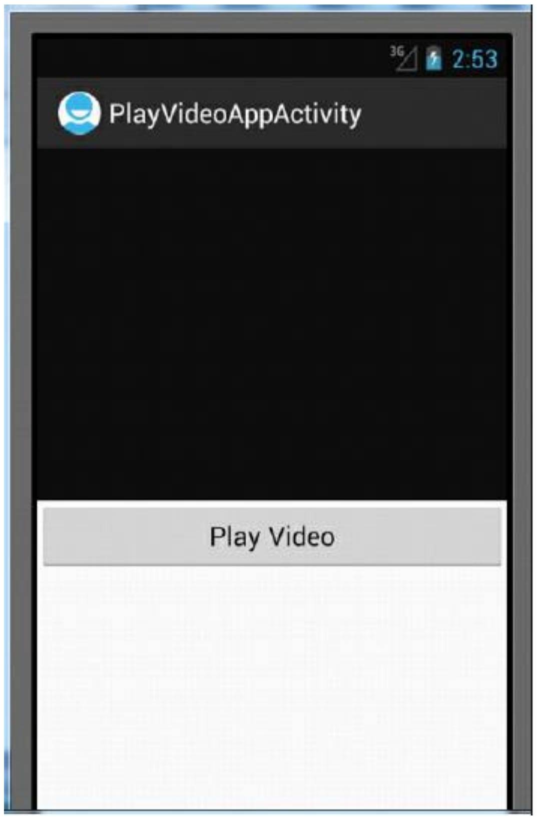
1. Open the DDMS perspective by selecting the Window, Open Perspective, DDMS option.
2. In the DDMS perspective, open the File Explorer by selecting Window, Show View, File Explorer.
3. If you can't see the running emulator anywhere, open the Devices view by selecting Window, Show View, Devices option. We are able to see all the running emulators, and we can select the one that we want to use for playing the video.
4. In the File Explorer view, we see different folders and files in the emulator.
5. Navigate the tree and expand the mnt node.
6. In the mnt node, select the sdcard node.
7. on the top right side of File Explorer, you see the messages Pull a file
from the device and Push a file onto the device.
8. Click the button Push a file onto the device.Then we will see a dialog box for choosing a video from thedisk drive.
9. After selecting a video, select the OK button to load the selected video onto the SD Card.
10. The video.pm4 which has been loaded onto the SD card is shown in File Explorer.
11. Then we can go ahead and add a VideoView control to our layout file to view the loaded video.

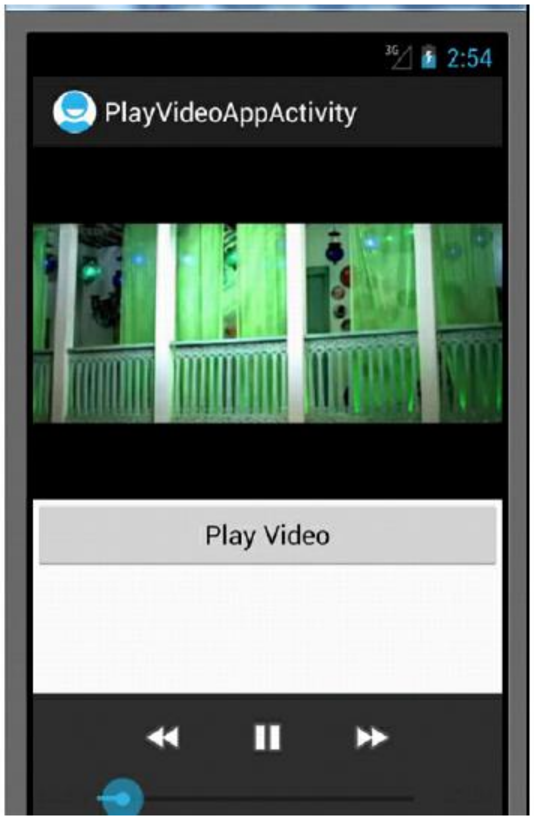
Code in the Layout File activity_play_video_app.xml on Adding VideoView and Button Controls

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:orientation="vertical"
android:layout_width="match_parent"
android:layout_height="match_parent" >
<VideoView android:id="@+id/video"
android:layout_width="320dip"
android:layout_height="240dip"/>
<Button android:id="@+id/playvideo"
android:text="Play Video"
android:layout_height="wrap_content"
android:layout_width="match_parent" />
</LinearLayout>
```

Code in the Java Activity File PlayAudioAppActivity.java

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.Button;
import android.view.View.OnClickListener;
import android.view.View;
import android.widget.MediaController;
import android.widget.VideoView;
public class PlayVideoAppActivity extends Activity {
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_play_video_app);
Button playVideoButton=
(Button)findViewById(R.id.playvideo);
playVideoButton.setOnClickListener(new OnClickListener()
{
public void onClick(View view){
VideoView videoView=
(VideoView)findViewById(R.id.video);
videoView.setMediaController(new
MediaController(PlayVideoAppActivity.this));
videoView.setVideoPath("sdcard/video.mp4");
videoView.requestFocus();
videoView.start();
}
});
}
}
```





Displaying Progress with ProgressBar:

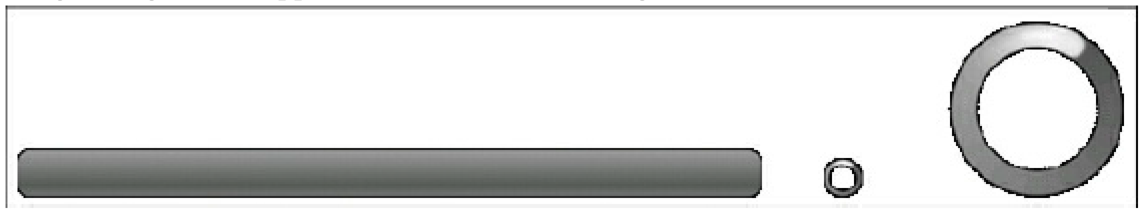
The ProgressBar is a control commonly used for displaying the progress of execution of tasks such as downloading a file, installing software, executing complex queries, and playing audio and video. To make the ProgressBar display in the form of a horizontal bar, set its style attribute to @android:style/Widget.ProgressBar.Horizontal,

```
<ProgressBar android:id="@+id/progressbar"
android:layout_width="match_parent"
android:layout_height="wrap_content"
style="@android:style/Widget.ProgressBar.Horizontal" />
```

The following styles can be applied to the ProgressBar:

• Widget.ProgressBar.Horizontal
• Widget.ProgressBar.Small
• Widget.ProgressBar.Large
• Widget.ProgressBar.Inverse
• Widget.ProgressBar.Small.Inverse
• Widget.ProgressBar.Large.Inverse

The ProgressBar for Widget.ProgressBar.Horizontal, Widget.ProgressBar.Small, and Widget.ProgressBar appears as shown in below figure



The inverse style is used when our application uses a light-colored theme, such as a white background. The minimum value of the ProgressBar is by default 0. The maximum value of the ProgressBar is set by the android:max attribute.

Write a program To play and stop the audio with the ToggleButton control and to display the progress of the audio in the ProgressBar control.

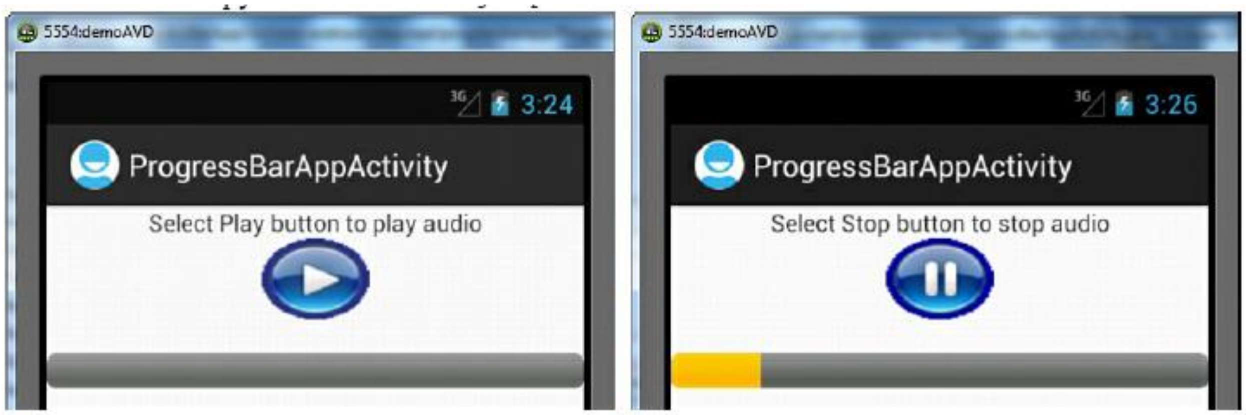
Code Written in Layout File activity_progress_bar_app.xml on Adding the ToggleButton and ProgressBar Controls.

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical" >
<TextView
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:gravity="center"
android:id="@+id/response"/>
<ToggleButton android:id="@+id/playstop_btn"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textOn=""
android:textOff=""
```

```
android:layout_gravity="center"
android:background="@drawable/play" />
<ProgressBar android:id="@+id/progressbar"
android:layout_width="match_parent"
android:layout_height="wrap_content"
style="@android:style/Widget.ProgressBar.Horizontal"
android:layout_marginTop="20dip" />
</LinearLayout>
```

Code Written in the Activity File ProgressBarAppActivity.java

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.ToggleButton;
import android.view.View;
import android.widget.TextView;
import android.media.MediaPlayer;
import android.view.View.OnClickListener;
import android.widget.ProgressBar;
import android.os.Handler;
public class ProgressBarAppActivity extends Activity {
MediaPlayer mp;
ProgressBar progressBar;
private final Handler handler = new Handler();
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_progress_bar_app);
final TextView response =
(TextView)this.findViewById(R.id.response);
response.setText("Select Play button to play audio");
progressBar=(ProgressBar)findViewById(R.id.progressbar);
mp =
MediaPlayer.create(ProgressBarAppActivity.this,R.raw.song1);
final ToggleButton playStopButton = (ToggleButton)
findViewById(R.id.playstop_btn);
progressBar.setProgress(0);
progressBar.setMax(mp.getDuration());
playStopButton.setOnClickListener(new OnClickListener()
{
public void onClick(View v) {
if (playStopButton.isChecked()) {
response.setText("Select Stop button to stop
audio");
playStopButton.setBackgroundDrawable(
getResources().getDrawable(R.drawable.stop));
mp.start();
updateProgressBar();
}
else {
response.setText("Select Play button to play
audio");
playStopButton.setBackgroundDrawable(
getResources().getDrawable(R.drawable.play));
mp.pause();
}
}
});
}
private void updateProgressBar() {
progressBar.setProgress(mp.getCurrentPosition());
if (mp.isPlaying()) {
Runnable notification = new Runnable() {
public void run() {
updateProgressBar();
}
};
handler.postDelayed(notification,1000);
}
}
}
```



- An event handler, OnClickListener, is added to the ToggleButton object playStopButton. Whenever the ToggleButton is clicked, its callback method, onClick(), is executed.
- In the onClick() method, we check the status of the ToggleButton. If the ToggleButton is set to the On state, we call the MediaPlayer’s start() method to play the audio and set the TextView to display the text Select Stop button to stop audio to tell the user that if the ToggleButton is pressed again, the audio will be stopped.
- The updateprogressbar() method is called to update the ProgressBar indicator.

Using Assets:

Assets provide a way to include arbitrary files like text, xml, fonts, music, and video in your application. If you try to include these files as "resources", Android will process them into its resource system and you will not be able to get the raw data. If you want to access data untouched, Assets are one way to do it.Assets added to your project will show up just like a file system that can read from by your application using AssetManager.

Code in Layout File activity_assets_app.xml on Adding the TextView Control

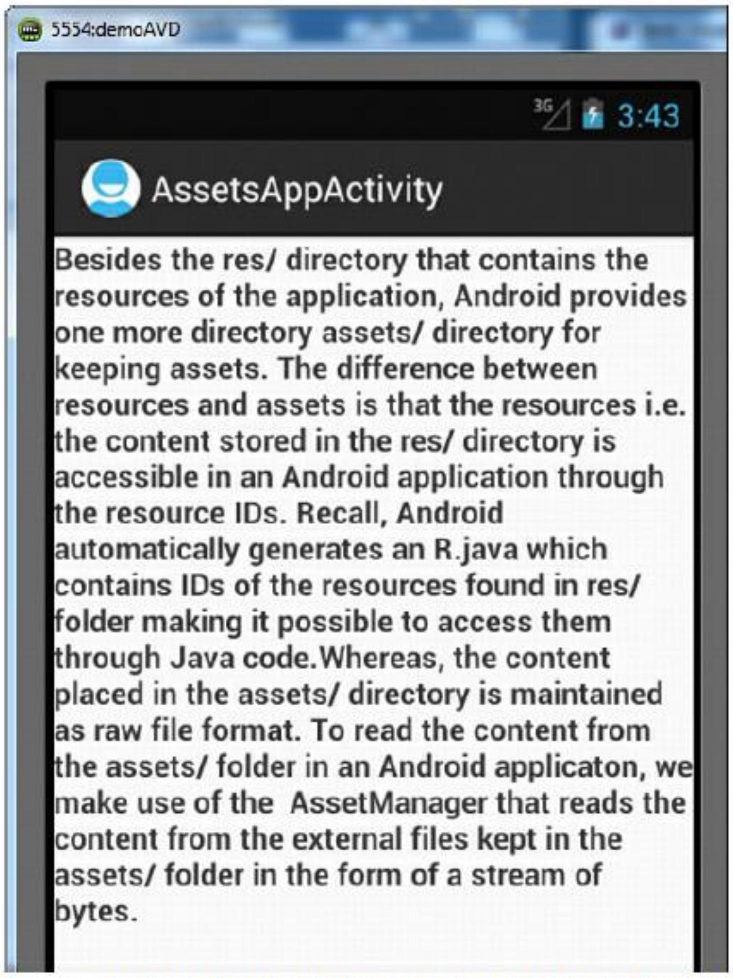
```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical" >
<TextView
android:id="@+id/file_view"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:textSize="15dip"
android:textStyle="bold" />
</LinearLayout>
```

Code in the Java Activity File AssetsAppActivity.java

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
import java.io.InputStream;
import android.content.res.AssetManager;
import java.io.IOException;
public class AssetsAppActivity extends Activity {
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_assets_app);
TextView fileView=
(TextView)findViewById(R.id.file_view);
InputStream input;
AssetManager assetManager = getAssets();
try {
input = assetManager.open("info.txt");
int fileSize = input.available();
byte[] buffer = new byte[fileSize];
input.read(buffer);
input.close();
String textForm = new String(buffer);
fileView.setText(textForm);
} catch (IOException e) {
```

```
fileView.setText("Some exception has occurred");
}
}
}
```

- The getAssets() method opens the Assets folder and returns a handle to this folder.
- To read content from a file, we need an InputStream object, so the open() method of the AssetManager class is called, passing the filename info.txt to it as the parameter.
- The open() method opens the file info.txt in the assets folder and returns it in the form of the InputStream, which is assigned to the InputStream instance input.
- The size of the file is computed by calling an available() method on the InputStream class.
- A buffer equal to the file size is defined, and the file content is read via the InputStream object into the buffer.
- The InputStream object is closed. The file content in the form of bytes stored in the buffer is converted into a string and is assigned to the TextView control for display.
- After the application is run, we find the content of the file info.txt is displayed through TextView, as shown in figure:



Besides the res/ directory that contains the resources of the application, Android provides one more directory assets/ directory for keeping assets. The difference between resources and assets is that the resources i.e. the content stored in the res/ directory is accessible in an Android application through the resource IDs. Recall, Android automatically generates an R.java which contains IDs of the resources found in res/ folder making it possible to access them through Java code.Whereas, the content placed in the assets/ directory is maintained as raw file format. To read the content from the assets/ folder in an Android applicaton, we make use of the AssetManager that reads the content from the external files kept in the assets/ folder in the form of a stream of bytes.

The content of the file in the `assets` folder displayed via `TextView`