UNIT II

CHAPTER 3

Delightful User Experience

Syllabus

Drawables, Themes and Styles, Material design, Providing resources for adaptive layouts

Syllabus Topic : Drawables

3.1 Drawables

- Drawable are compiled images that you can use in your app. Android provides classes and resources to help you include rich images in your application with a minimal impact to your app's performance.
- You also learn how to use styles and themes to provide a consistent appearance to all the elements in your app while reducing the amount of code.
- A drawable is a graphic that can be drawn to the screen. retrieve drawables using APIs such as getDrawable(int, apply a drawable to an XML resource using attributes such as android:drawable and android:icon.

1. Android includes several types of drawables like Image files
2. Nine-patch files
3. Layer lists
4. Shape drawables
5. State lists
6. Level lists
7. Transition drawables
8. Vector drawables

Using drawables

- To display a drawable, use the ImageView class to create a View. In the <ImageView> element in your XML file, define how the drawable is displayed and where the drawable file is located.
- Consider ImageView displays an image called "cake1.jpg": then in xml pass the given code

```
<ImageView
    android:id="@+id/tiles"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/cake1" />
```

About the <ImageView> attributes :

- The android:id attribute sets a shortcut name that you use to call the image later.
- The android:layout_width and android:layout_height attributes specify the size of the View.
- Here height and width are set to wrap_content, means the View is only big enough to enclose the image within it, plus padding.
- The android:src attribute gives the location where this image is stored.

Store image files in the res/drawable folder. Use them with the android:src attribute for an ImageViewand its descendants, or to create a BitmapDrawable class in Java code.

To represent a drawable in your app, use the Drawable class or one of its subclasses.

Consider this code retrieves the cake1.jpg image as a Drawable:

```
Resources res = getResources();
Drawable drawable = res.getDrawable(R.drawable.cake1);
```

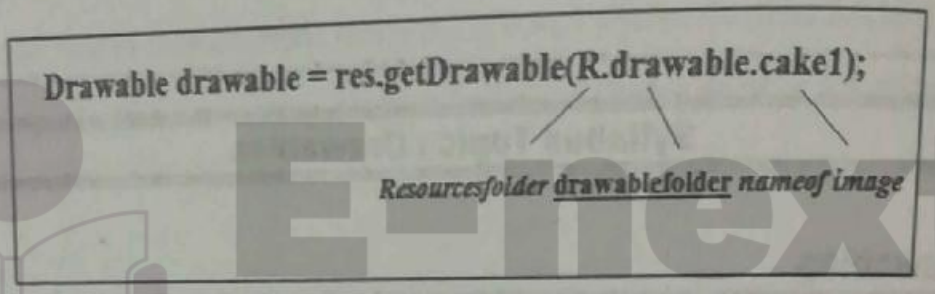


Fig. 3.1.1

- Expand folder res then select drawable folder as show above and copy the images that you want to add in your appplication in drawable folder as shown in Fig. 3.1.2.

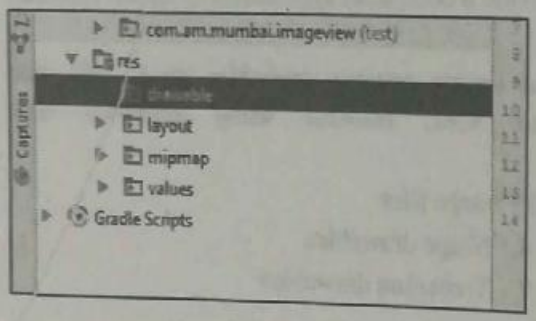


Fig. 3.1.2

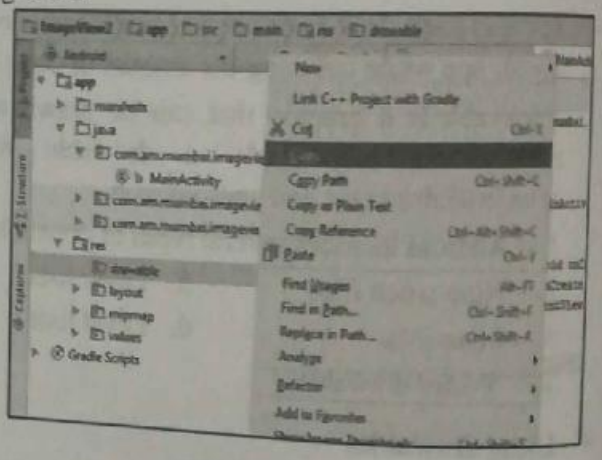


Fig. 3.1.3

- Here we add three images as cake1.jpg, cake2.jp and cake3.jpg by simply copy all this file in drawable folder when we copy images it will looks like as below for looking added images expand drawable fodder

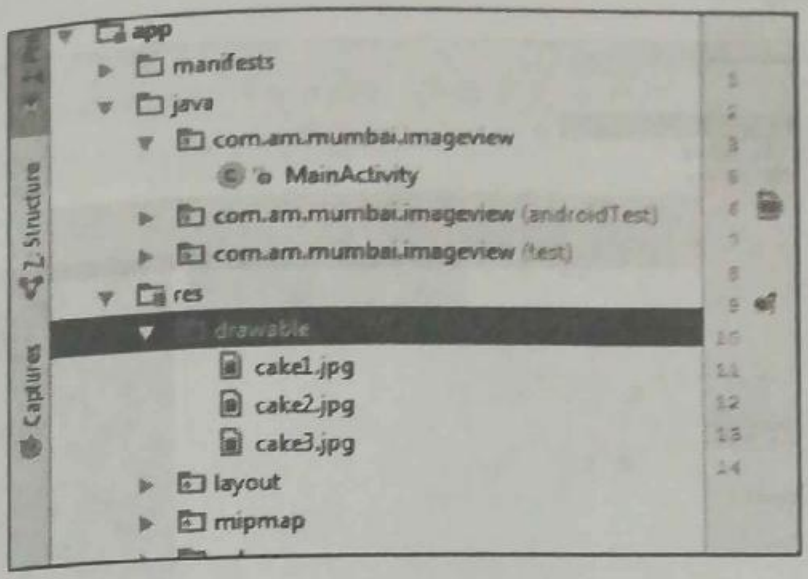


Fig. 3.1.4

- An image file is a generic bitmap file. Android supports image files in several formats: WebP (preferred), PNG (preferred), and JPG (acceptable). GIF and BMP formats are supported, but discouraged.
- Be aware that images look different on screens with different pixel densities and aspect ratios. For information on supporting different screen sizes, see Speeding up your app, below, and the screen sizes guide.
- To apppy images on perticular ImageButton or ImageView do the folllowing steps

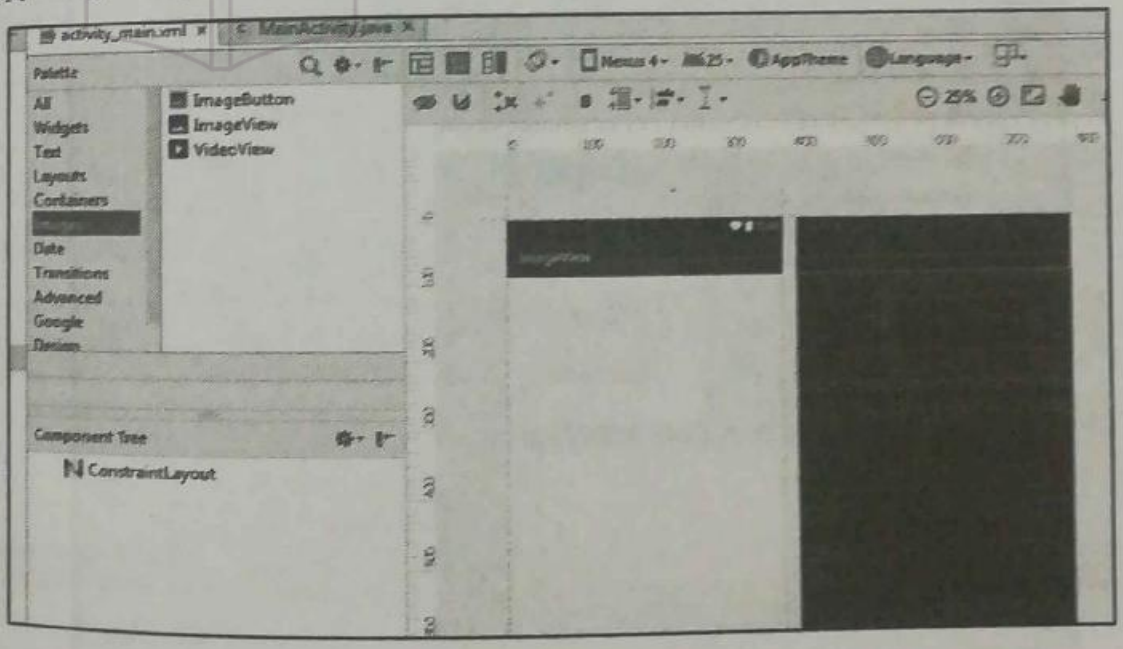


Fig. 3.1.5

- **Drag and drop ImageView**

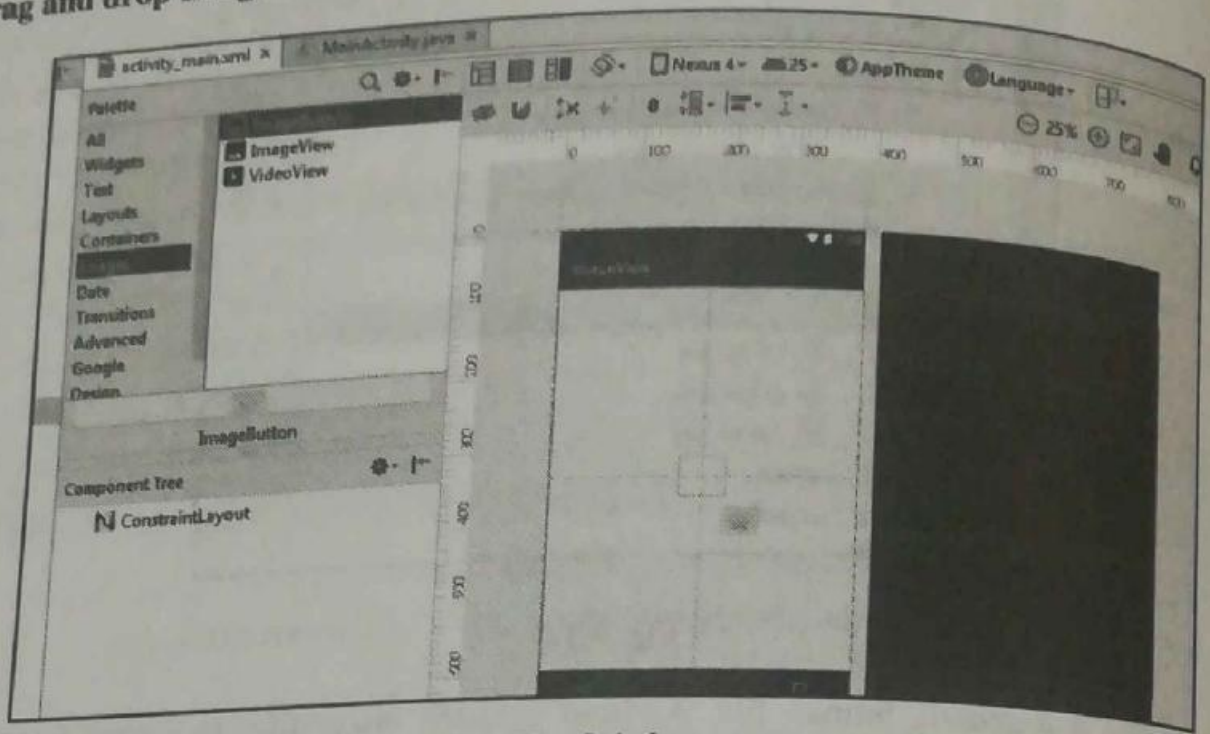


Fig. 3.1.6

– When you add ImageButton new window is appeared for add image for ImageButton and se perticular image for ImageButton by selecting any images which are already added.

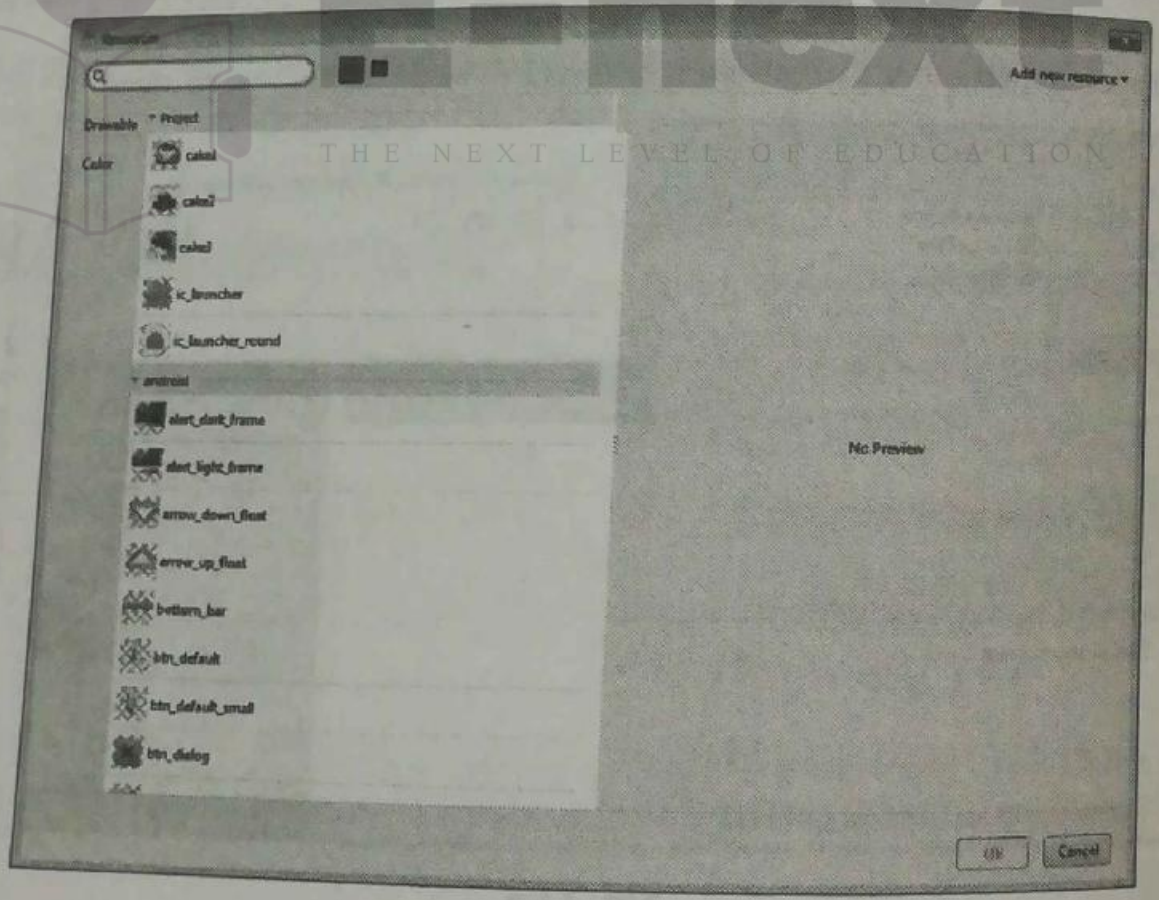


Fig. 3.1.7

Here we select cake1 image

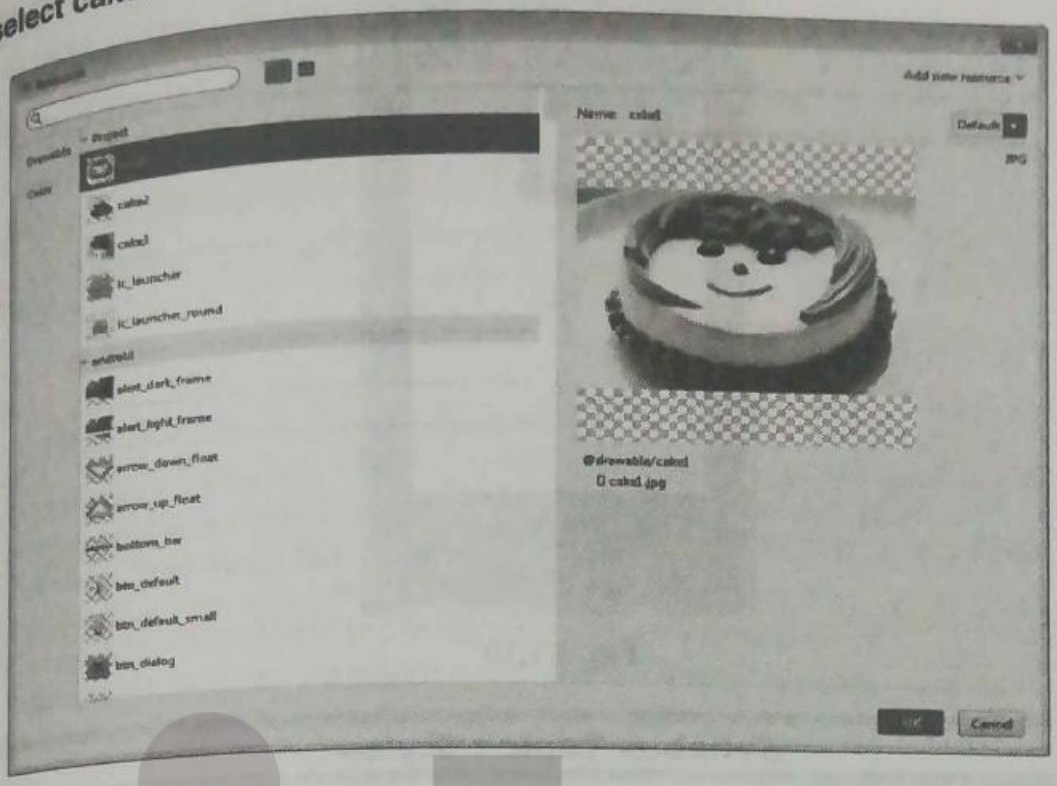


Fig. 3.1.8

When you add image for Button it looks like as

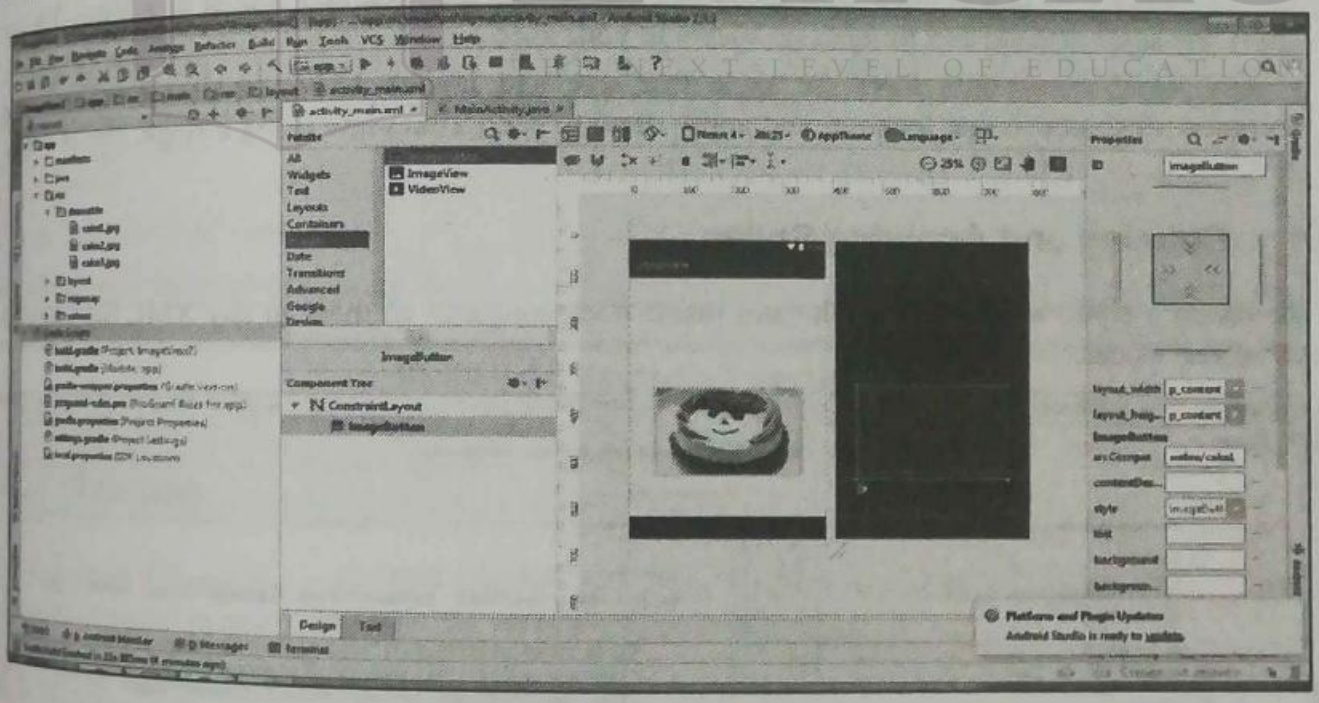


Fig. 3.1.9

In this way you add more images for ImageButton and ImageView when you run app it looks like as shown in Fig. 3.1.10.

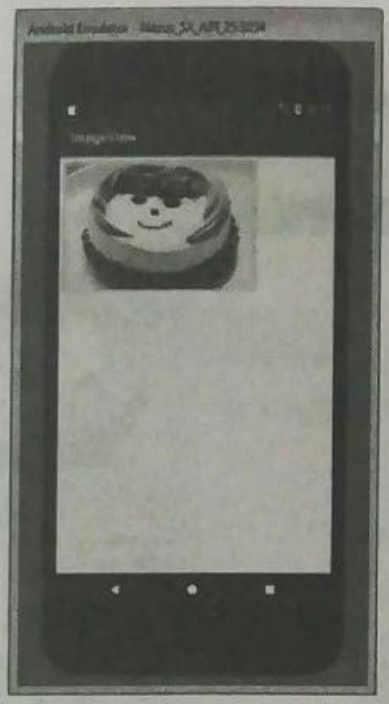


Fig. 3.1.10

Syllabus Topic : Styles

3.2 Styles

- In Android, a *style* is a collection of attributes that define the look and format of a View. You can apply the same style to any number of Views in your app;
- several TextViews might have the same text size and layout. Using styles allows keep these common attributes in one location and apply them to each TextView using a single line of code in XML.

3.2.1 Defining and Applying Styles

To create a style, add a <style> element inside a <resources> element in any XML file located in the res/values/ folder.

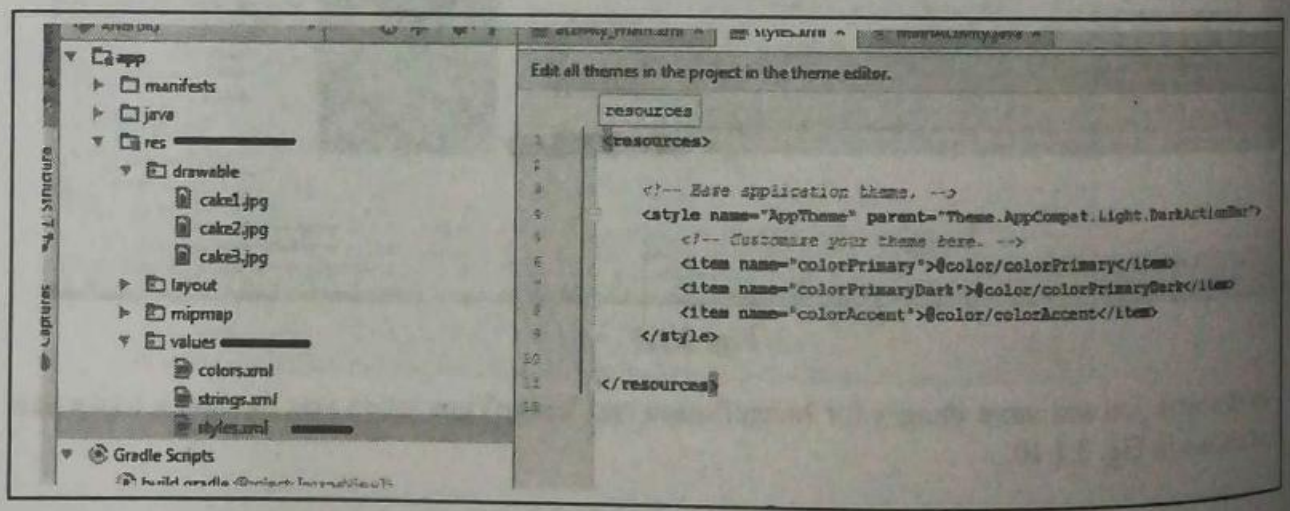


Fig. 3.2.1

- When you create a project in Android Studio, a res/values/styles.xml file is created for you.

A <style> element includes the following.

```
<resources> <style name="CodeFont">
    <item name="android:typeface">monospace</item>
    <item name="android:textColor">#D7D6D7</item>
  </style> </resources>
```

OR

Style.xml file

```
<resources>
  <!-- Base application theme. -->
  <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
    <!-- Customize your theme here. -->
    <item name="colorPrimary">@color/colorPrimary</item>
    <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
    <item name="colorAccent">@color/colorAccent</item>
  </style>
</resources>
```

- A name attribute. Use the style's name when you apply the style to a View.
- An optional parent attribute. You learn about using parent attributes in the Inheritance section below.
- Any number of <item> elements as child elements of <style>. Each <item> element includes one style attribute.

The following XML applies the new CodeFont style to a View:

```
<TextView
  style="@style/CodeFont"
  android:text="@string/code_string"/>
```

Syllabus Topic : Themes

3.3 Themes

You create a theme the same way you create a style, which is by adding a <style> element inside a <resources> element in any XML file located in the res/values/ folder.

Q. What's the difference between a style and a theme?

- A style applies to a View. In XML, you apply a style using the style attribute.
- A theme applies to an entire Activity or application, rather than to an individual View. In XML, you apply a theme using the android:theme attribute.

Any style can be used as a theme. For example, you could apply the CodeFont style as a theme for an Activity, and all the text inside the Activity would use gray monospace font.

3.3.1 Applying Themes

- To apply a theme to your app, declare it inside an <application> element inside the AndroidManifest.xml file. This example applies the AppTheme theme to the entire application:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.exampledomain.myapp">
  <application
    ...
    android:theme="@style/AppTheme">
  </application>
```

Or

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.am.mumbai.imageview">
  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>
</manifest>
```

- To apply a theme to an Activity, declare it inside an <activity> element in the AndroidManifest.xml file. In this example, the android:theme attribute applies the Theme_Dialog platform theme to the Activity:

```
<activity android:theme="@android:style/Theme.Dialog">
```

3.3.2 Default Theme

When you create a new project in Android Studio, a default theme is defined for you within the styles.xml file. For example, this code might be in your styles.xml file:

```
<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
  <!-- Customize your theme here. -->
  <item name="colorPrimary">@color/colorPrimary</item>
```

```
<item name="colorPrimaryDark">@color/colorPrimaryDark</item>
<item name="colorAccent">@color/colorAccent</item>
</style>
```

In this example, AppTheme inherits from Theme.AppCompat.Light.DarkActionBar, which is one of the many Android platform themes available to you

3.3.3 Platform Styles and Themes

The Android platform provides a collection of styles and themes that you can use in your app. To find a list of all of them, you need to look in two places:

- The R.style class lists most of the available platform styles and themes.
- The support.v7.appcompat.R.style class lists more of them. These styles and themes have "AppCompat" in their names, and they are supported by the v7 appcompat library.
- The style and theme names include underscores. To use them in your code, replace the underscores with periods. For example, here's how to apply the Theme_NoTitleBar theme to an activity:

```
<activity android:theme="@android:style/Theme.NoTitleBar"
```

And here's how to apply the AlertDialog_AppCompat style to a View:

```
<TextView
style="@style/AlertDialog.AppCompat"
android:text="@string/code_string" />
```

Syllabus Topic : Material Design

3.4 Material Design

- Google created Material Design in 2014 which is a visual design philosophy. The aim of Material Design is a unified user experience across platforms and device sizes.
- Material Design includes a set of guidelines for style, layout, motion, and other aspects of app design.
- Material Design is for desktop web applications as well as for mobile apps.
- In Material Design, elements in your Android app behave like real world materials: they cast shadows, occupy space, and interact with each other.

Bold, graphic, intentional

- Material Design involves deliberate color choices, edge-to-edge imagery, large-scale typography, and intentional white space that create a bold and graphic interface.
- Emphasize user actions in your app so that the user knows right away what to do, and how to do it. For example, highlight things that users can interact with, such as buttons, EditText fields, and switches.

Colors

- Material Design color palette
- Material Design principles include the use of bold color. The Material Design color palette contains colors to choose from, each with a primary color and shades labeled from 50 to 900:

- Choose a color labeled "500" as the primary color for your brand. Use that color and shades of that color in your app.
- Choose a contrasting color as your accent color and use it to create highlights in your app. Select any color that starts with "A."
- When you create an Android project in Android Studio, a sample Material Design color scheme is selected for you and applied to your theme. In values/colors.xml, three <color> lements are defined, colorPrimary, colorPrimaryDark, and colorAccent:

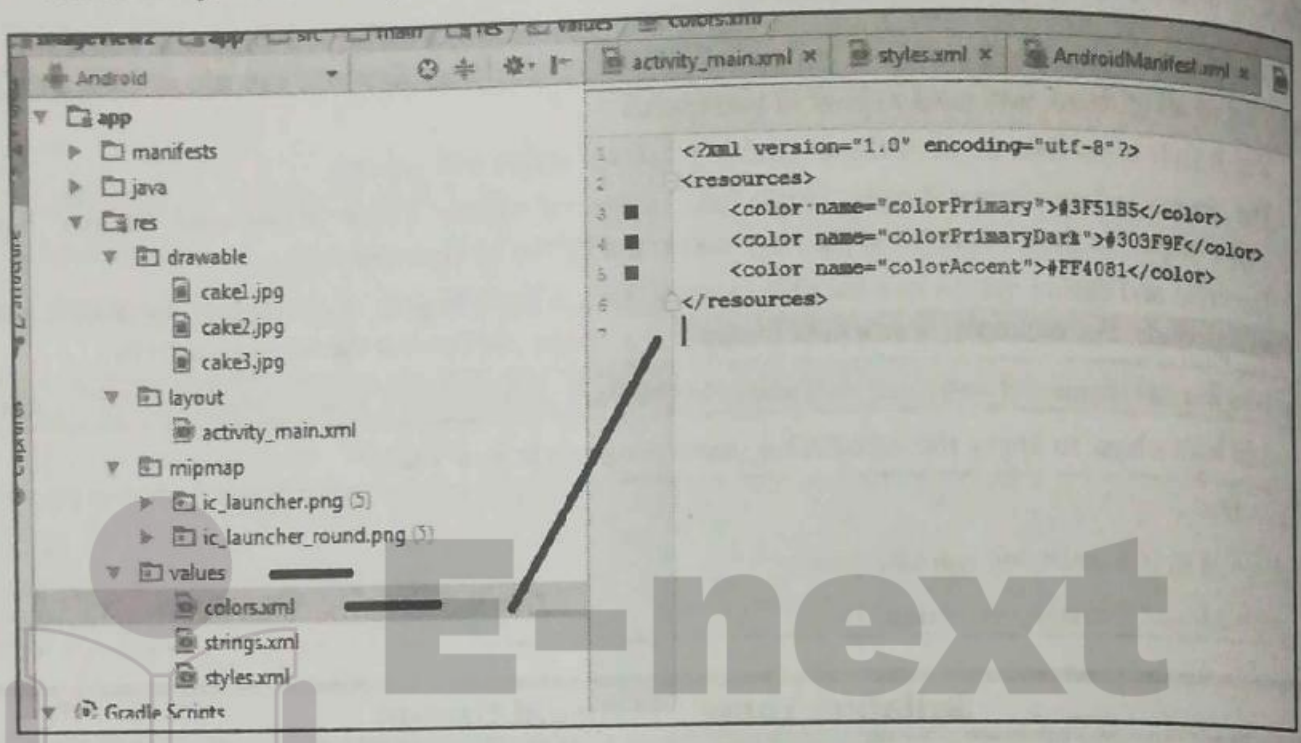


Fig. 3.4.1

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <color name="colorPrimary">#3F51B5</color>
   <color name="colorPrimaryDark">#303F9F</color>
   <color name="colorAccent">#FF4081</color>
</resources>
```

- In values/styles.xml, the three defined colors are applied to the default theme, which applies the colors to some app elements by default:
- colorPrimary is used by several Views by default. colorPrimary is used as the background color for the action bar. Change this value to the "500" color that you select as your brand's primary color.
- colorPrimaryDark is used in areas that need to slightly contrast with your primary color EX- status bar. Set this value to a slightly darker version of your primary color.

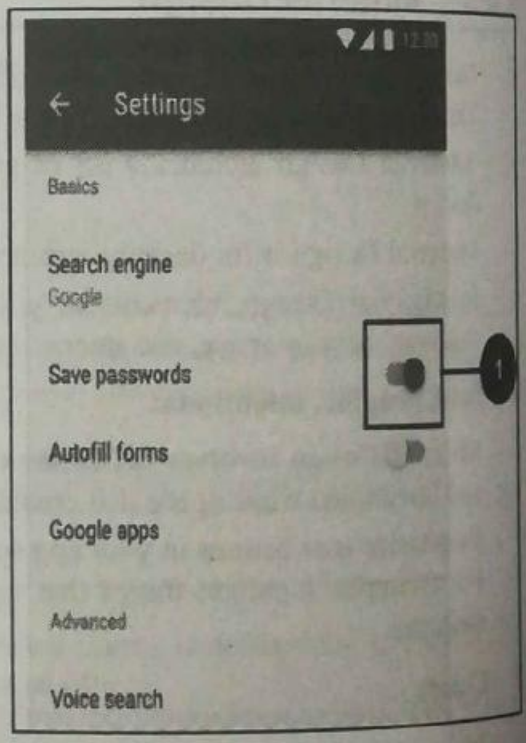


Fig. 3.4.2

- colorAccent is used as the highlight color for several Views and used for switches in the "on" position, floating action buttons, and more.
- In the screenshot below, the background of the action bar uses colorPrimary (indigo), the status bar uses colorPrimaryDark (a darker shade of indigo), and the switch in the "on" position uses colorAccent (a shade of pink).
- In this layout, the switch in the "on" position is highlighted with a pink accent color.

Q. How to use the Material Design color palette in your Android app?

1. Pick a primary color for your app from Material Design color palette and copy its hex value into the colorPrimary item in colors.xml.
2. Pick a darker shade of this color and copy its hex value into the colorPrimaryDark item.
3. Pick an accent color from the shades starting with an "A" and copy its hex value into the colorAccent item.
4. If you need more colors, create additional <color> elements in the colors.xml file. For example, you could pick a lighter version of indigo and create an additional <color> element named colorPrimaryLight. (The name is up to you.)

```
5. <color name="colorPrimaryLight">#9FA8DA</color>
6. <!-- A lighter shade of indigo. -->
```

- To use this color, reference it as @color/colorPrimaryLight.
- Changing the values in colors.xml automatically changes the colors of the Views in your app, because the colors are applied to the theme in styles.xml.

3.4.1 Font Styles

- The Android platform provides predefined font styles and sizes that you can use in your app. These styles and sizes were developed to balance content density and reading comfort under typical conditions.
- Type sizes are specified with sp (scaleable pixels) to enable large type modes for accessibility.
- Be careful not to use too many different type sizes and styles together in your layout.

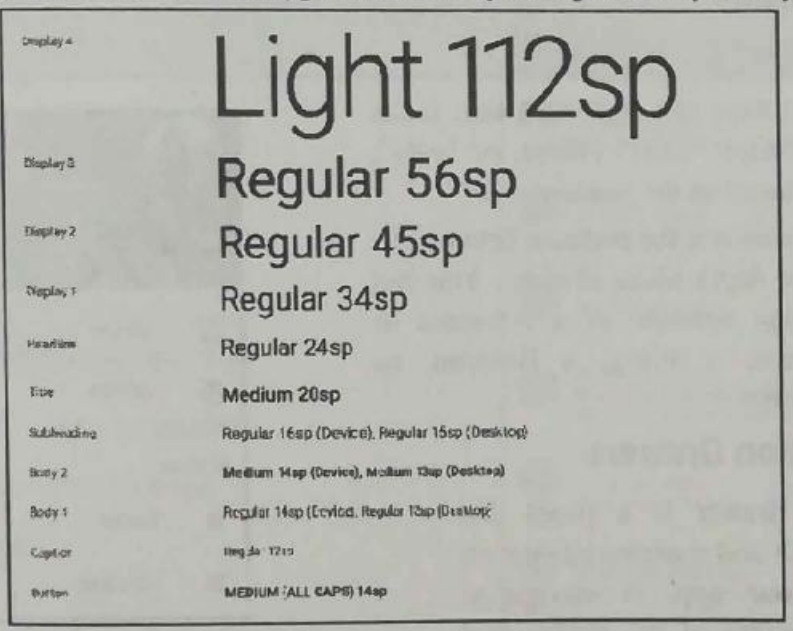
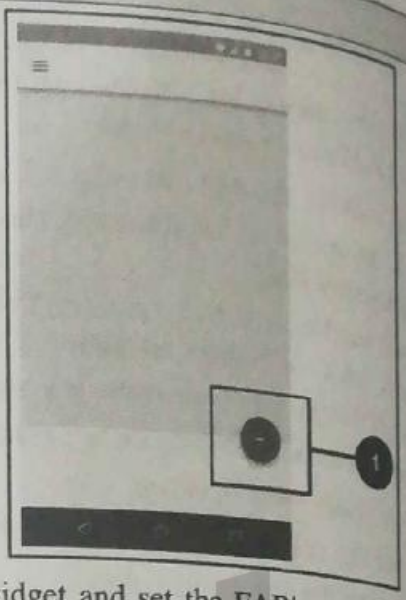


Fig. 3.4.3

- To use one of these predefined styles in a View, set the android:textAppearance attribute. This attribute defines the default appearance of the text: its color, typeface, size, and style.
- Use the backward-compatible TextAppearance.AppCompat style.
- For example, to make a TextView appear in the Display 3 style, add the following attribute to the TextView in XML:

```
android:textAppearance="@style/TextAppearance.AppCompat.Display3"
```

3.4.2 Examples of Font Styles



☞ **Floating action buttons (FABs)**

- Use a floating action button (FAB) for actions you want to encourage users to take. A FAB is a circled icon that floats "above" the UI.
- On focus it changes color slightly, and it appears to lift up when selected. When tapped, it can contain related actions.

☞ **A normal-sized FAB**

To implement a FAB, use the FloatingActionButton widget and set the FAB's attributes in your layout XML. For example:

```
<android.support.design.widget.FloatingActionButton
    android:id="@+id/addNewItemFAB"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/ic_plus_sign"
    app:fabSize="normal"
    app:elevation="10%" />
```

- The fabSize attribute sets the FAB's size. It can be "normal" (56dp), "mini" (40dp), or "auto", which changes based on the window size.
- The FAB's elevation is the distance between its surface and the depth of its shadow. You can set the elevation attribute as a reference to another resource, a string, a Boolean, or several other ways.

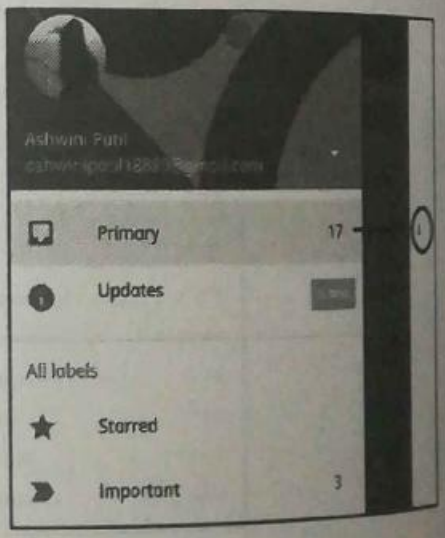


Fig. 3.4.4

3.4.3 Navigation Drawers

A navigation drawer is a panel that slides in from the left and contains navigation destinations for your app. A navigation drawer spans the height of the screen, and everything behind it is visible, but darkened.

An "open" navigation drawer

To implement a navigation drawer, use the DrawerLayout APIs available in the Support Library.

In your XML, use a DrawerLayout object as the root view of your layout. Inside it, add two views, one for your primary layout when the drawer is hidden, and one for the contents of the drawer.

For example, the following layout has two child views: a FrameLayout to contain the main content (populated by a Fragment at runtime), and a ListView for the navigation drawer.

```
<android.support.v4.widget.DrawerLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/drawer_layout"
  android:layout_width="match_parent"
  android:layout_height="match_parent">
  <!-- The main content view -->
  <FrameLayout
    android:id="@+id/content_frame"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
  <!-- The navigation drawer -->
  <ListView android:id="@+id/left_drawer"
    android:layout_width="240dp"
    android:layout_height="match_parent"
    android:layout_gravity="start"
    android:choiceMode="singleChoice"
    android:divider="@android:color/transparent"
    android:dividerHeight="0dp"
    android:background="#111"/>
</android.support.v4.widget.DrawerLayout>
```

3.4.4 Snackbars

A snackbar provides brief feedback about an operation through a message in a horizontal bar on the screen.

It contains a single line of text directly related to the operation performed. A snackbar can contain a text action, but no icons.

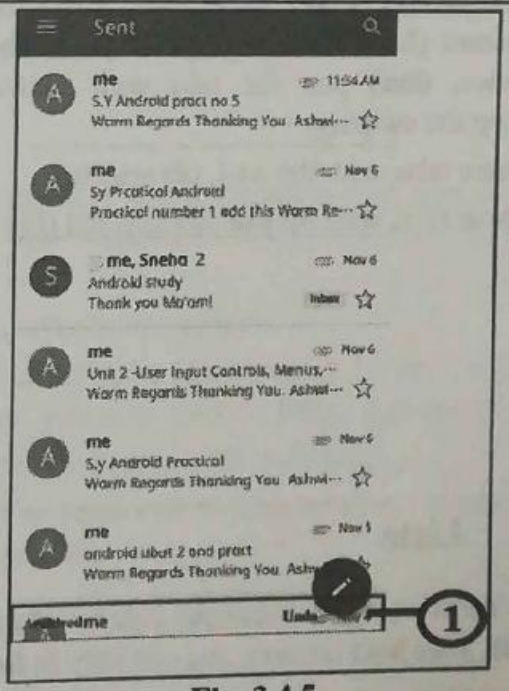


Fig. 3.4.5

- **Snackbar**

- Snackbars automatically disappear after a timeout or after a user interaction elsewhere on the screen. You can associate a snackbar with any kind of view (any object derived from the View class). However, if you associate the snackbar with a CoordinatorLayout, the snackbar gains additional features:
- The user can dismiss the snackbar by swiping it away.
- The layout moves some other UI elements when the snackbar appears. For example, if the layout has a FAB, the layout moves the FAB up when it shows the snackbar, instead of drawing the snackbar on top of the FAB.
- To create a Snackbar object, use the Snackbar.make() method. Specify the ID of the CoordinatorLayoutview to use for the snackbar, the message that the snackbar displays, and the length of time to show the message. For example, this Java statement creates the snackbar and calls show() to show the snackbar to the user:
- Snackbar.make(findViewById(R.id.myCoordinatorLayout),R.string.email_sent,
Snackbar.LENGTH_SHORT).show;
- For more information, see Building and Displaying a Pop-Up Message and the Snackbar reference. To make sure you're using snackbars as intended, see the snackbar usage information in the Material Design guide.
- Tip: A *toast* is similar to a snackbar, but toasts are usually used for system messaging, and users can't be swiped off the screen.

3.4.5 Tabs

- Use *tabs* to organize content at a high level. For example, the user might use tabs to switch between Views, data sets, or functional aspects of an app. Present tabs as a single row above their associated content. Make tab labels short and informative.
- You can you use tabs with *swipe views* in which users navigate between tabs with a horizontal finger gesture (horizontal paging). If your tabs use swipe views, don't pair the tabs with content that also supports swiping.
- Three tabs, with the ALL tab selected
- Three tabs, with the one tab selected (Fig. 3.4.6).

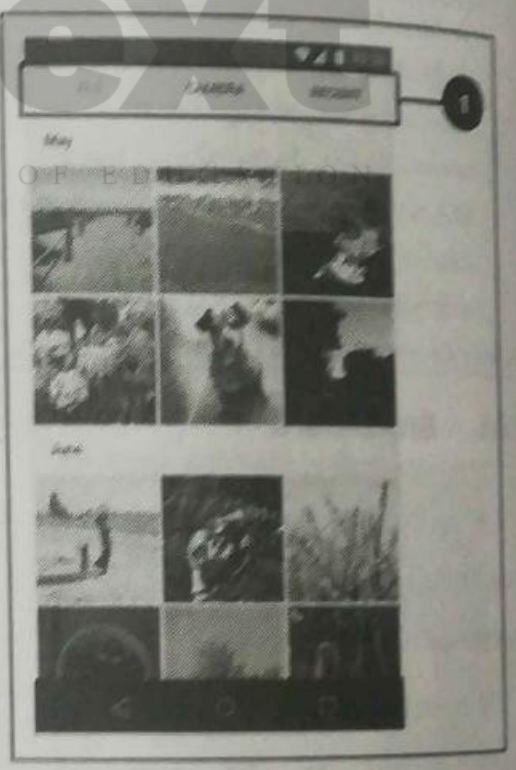


Fig. 3.4.6

3.4.6 Lists

- A *list* is a single continuous column of rows of equal width. Each row functions as a container for a tile. *Tiles* hold content, and can vary in height within a list.

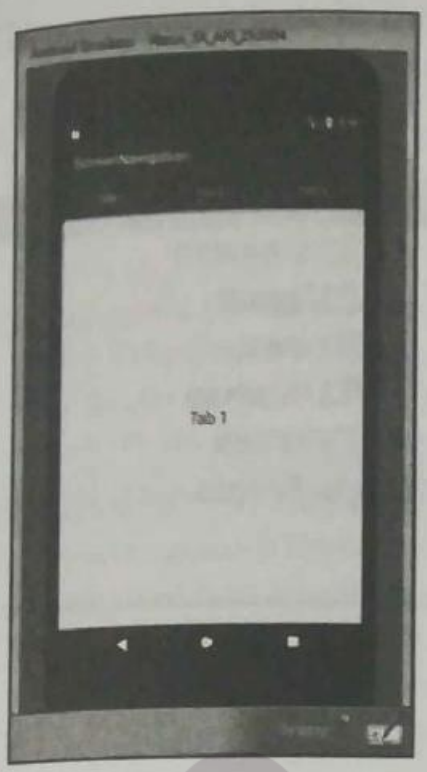


Fig. 3.4.7

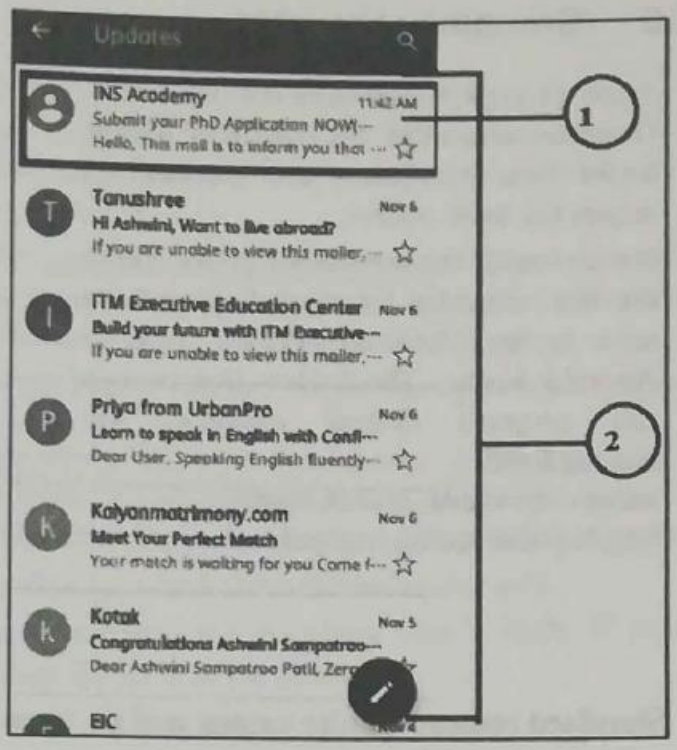


Fig. 3.4.8

1. A tile within the list
2. A list with rows of equal width, each containing a tile

Syllabus Topic : Providing Resources for Adaptive Layouts

3.5 Providing Resources for Adaptive Layouts

An adaptive *layout* works well for different screen sizes and orientations, different devices, different locales and languages, and different versions of Android.

3.5.1 Externalizing Resources

- When you *externalize* resources, you keep them separate from your application code. For example, instead of hard-coding a string into your code, you name the string and add it to the res/values/strings.xml file.
- Always externalize resources such as drawables, icons, layouts, and strings.
- Maintain externalized resources separately from your other code. If a resource is used in several places in your code and you need to change the resource, you only need to change it in one place.
- You can provide alternative resources that support specific device configurations,
- consider example as devices with different languages or screen sizes. This becomes increasingly important as more Android-powered devices become available.

3.5.2 Grouping resources

- Store all your resources in the res/ folder. Organize resources by type into folders under /res. You must use standardized names for these folders.
- For example, the screenshot below shows the file hierarchy for a small project, as seen in the "Android" Project view in Android Studio. The folders that contain this project's default resources use standardized
names: drawable, layout, menu,
mipmap (for icons), and values.

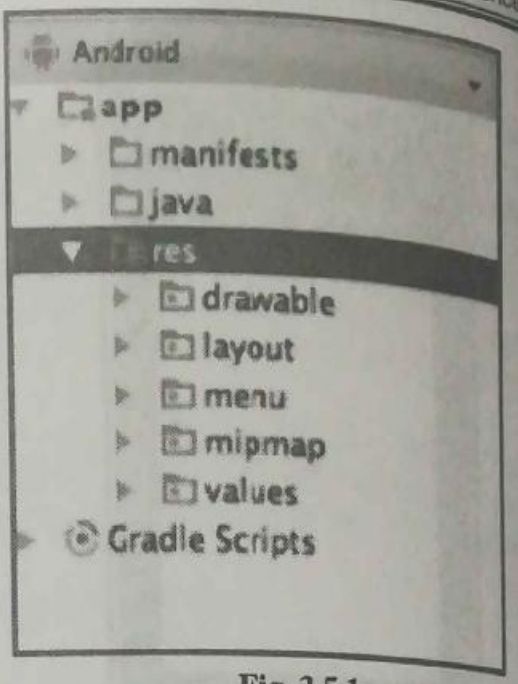


Fig. 3.5.1

☞ **Standard resource folder names. and its types**

Following are standard resource folder names. and its types

Name	Resource Type
animator/	It is XML files and define property animations.
anim/	It is XML files & define tween animations.
color/	It is XML files & define "state lists" of colors. (It is different from the colors.xml file under the values/ folder.)
drawable/	It is Bitmap files & contains -WebP, PNG, 9-patch, JPG, GIF and also XML files that are compiled into drawables
mipmap/	It is Drawable files for different launcher icon densities.
layout/	It is XML files & define user interface layouts.
menu/	It is XML files that define application menus.
raw/	It is Arbitrary files& it is saved in raw form. For open these resources with a raw InputStream,nedd to call Resources.openRawResource() with the resource ID, as syntax is R.raw.filename.
values/	It is XML files & contain simple values, like strings, integers, and colors. For clarity, place unique resource types in different files. For example, here are some filename conventions for resources you can create in this folder: arrays.xml for resource arrays (typed arrays) dimens.xml for dimension values strings.xml, colors.xml, styles.xml
xml/	It is Arbitrary XML files & it is read at runtime by calling Resources.getXml(). we have Various XML configuration files, such as a searchable configuration, must be saved here, along with preference settings.

3.5.3 Alternative Resources

- To support specific device configurations most apps provide alternative resources .
- Consider , your app should include alternative drawable resources for different screen densities, and alternative string resources for different languages, then as runtime, Android detects the current device configuration and loads the appropriate resources.
- If no resources are available for the device's specific configuration, Android uses the default resourcesthat you include in your app—the default drawables, which are in the res/drawable/ folder, the default text strings, which are in the res/values/strings.xml file, and so on.
- Like default resources, alternative resources are kept in folders inside res/. Alternative-resource folders use the following naming convention:

```
<resource_name>-<config_qualifier>
```

- <resource_name> is the folder name for this type of resource. For example, "drawable" or" values".
- <config_qualifier> specifies a device configuration for which these resources are used
- To add multiple qualifiers to one folder name, separate the qualifiers with a dash. If you use multiple qualifiers for a resource folder, you must list them in the order

Examples with one qualifier

- String resources localized would be in a res/values-ja/strings.xmlfile. Default string resources (resources to be used when no language-specific resources are found) would be in res/values/strings.xml.
- Style resources for API level 21 and higher would be in a res/values-v21/styles.xml file. Default style resources would be in res/values/styles.xml.

Example with multiple qualifiers

- Layout resources for a right-to-left layout running in "night" mode would be in a res/layout-ldrtl-night/ folder.
- In the "Android" view in Android Studio, the qualifier is not appended to the end of the folder. Instead, the qualifier is shown as a label on the right side of the file in parentheses. For example, in the "Android" view shown below, the res/values/dimens.xml/ folder shows two files:
  - The dimens.xml file, which includes default dimension resources.
  - The dimens.xml (w820dp) file, which includes dimension resources for devices that have a minimum available screen width of 820dp.

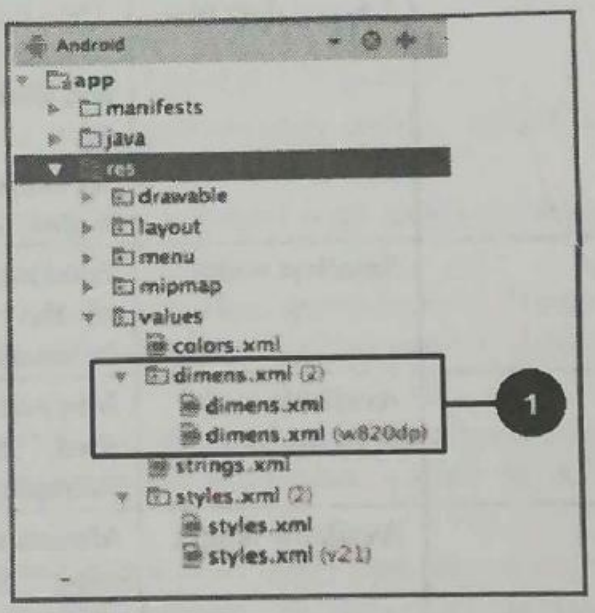


Fig. 3.5.2

1. In the "Android" view in Android Studio, default resources for dimensions are shown in the same folder as alternative resources for dimensions.

   In the "Project" view in Android Studio, the same information is presented differently, as shown in the below
2. In the "Project" view in Android Studio, default resources for dimensions are shown in the res/values folder.
3. Alternative resources for dimensions are shown in res/values-<qualifier> folders.

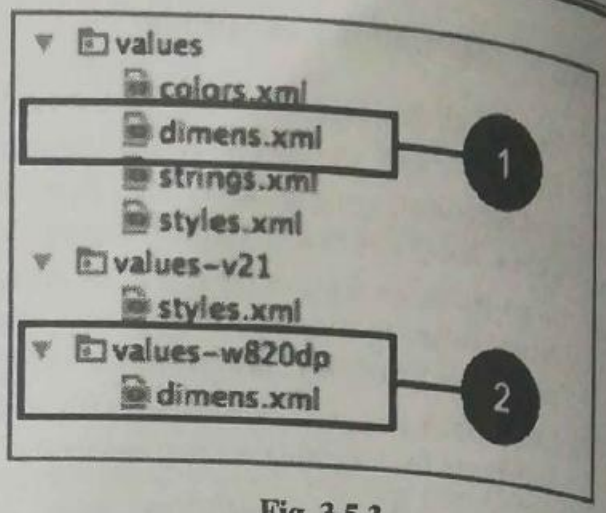


Fig. 3.5.3

They are listed in the order you must use when you combine multiple qualifiers in one folder name.

For example in res/layout-ldrtl-night/, the qualifier for layout direction is listed before the qualifier for night mode, because layout direction is listed before night mode in the Table 3.5.1

Table 3.5.1.: Configuration qualifiers that android supports

Precedence	Qualifier	Description
1.	MCC and MNC	The mobile country code (MCC), optionally followed by mobile network code (MNC) from the SIM card in the device. For example, mcc310 is U.S. on any carrier, mcc310-mnc004 is U.S. on Verizon, and mcc208-mnc00 is France on Orange.
2.	Localization	Language, or language and region. Examples: en, en-rUS, fr-rFR, fr-rCA. Described in Localization, below.
3.	Layout direction	The layout direction of your application. Possible values include ldltr (layout direction left-to-right, which is the default) and ldrtl(layout direction right-to-left). To enable right-to-left layout features, set supportsRtl to "true" and set targetSdkVersion to 17 or higher.
4	Smallest width	Fundamental screen size as indicated by the shortest dimension of the available screen area. Example: sw320dp. Described in Smallest width, below.
5	Available width	Minimum available screen width at which the resource should be used. Specified in dp units. The format is wdp, for example, w720dpand w1024dp.
6	Available height	Minimum available screen height at which the resource should be used. Specified in dp units. The format is hdp, for example, h720dpand h1024dp.
7	Screen size	**Possible values:** **small:** Screens such as QVGA low-density screens **normal:** Screens such as HVGA medium-density **large:** Screens such as VGA medium-density **xlarge:** Screens such as those on tablet-style devices

Precedence	Qualifier	Description
8	Screen aspect	Possible values include long (for screens such as WQVGA, WVGA, FWVGA) and notlong (for screens such as QVGA, HVGA, and VGA).
9	Round screen	Possible values include round (for screens such as those on round wearable devices) and notround (for rectangular screens such as phones).
10	Screenorientation	**Possible values** : port, land. Described in Screen orientation, below.
11	UI mode	**Possible values** **car:** Device is displaying in a car dock **desk:** Displaying in a desk dock **television:** Displaying on a large screen that the user is far away from, primarily oriented around D-pad or other non-pointer interaction **appliance:** Device is serving as an appliance, with no display **watch:** Device has a display and is worn on the wrist
12	Night mode	**Possible values** Night, notnight
13	Screen pixel density	**Possible values** **ldpi:** Low-density screens; approximately 120dpi. **mdpi:** Medium-density (on traditional HVGA) screens; approximately 160dpi. **hdpi:** High-density screens; approximately 240dpi. **xhdpi:** Approximately 320dpi. Added in API level 8. **xxhdpi:** Approximately 480dpi. Added in API level 16. **xxxhdpi:** Launcher icon only; approximately 640dpi. Added in API level 18. **nodpi:** For bitmap resources that you don't want scaled to match the device density. **tvdpi:** Screens between mdpi and hdpi; approximately 213dpi. Intended for televisions, and most apps shouldn't need it. Added in API level 13. **anydpi:** Matches all screen densities and takes precedence over other qualifiers. Useful for vector drawables. Added in API level 21. **Note:** Using a density qualifier doesn't imply that the resources are *only* for screens of that density. If you don't provide alternative resources with qualifiers that better match the current device configuration, the system may use whichever resources are the best match.
14	Touchscreen type	Possible values include notouch (device doesn't have a touchscreen) and finger (device has a touchscreen).

Precedence	Qualifier	Description
15	Keyboard availability	**Possible values:** **keysexposed:** Device has a keyboard available. **keyshidden:** Device has a hardware keyboard available but it's hidden, and the device does not have a software keyboard enabled. **keyssoft:** Device has a software keyboard enabled, whether it's visible or not.
16	Primary text input method	**Possible values:** **nokeys:** Device has no hardware keys for text input. **qwerty:** Device has a hardware qwerty keyboard, whether it's visible to the user or not. **12key:** Device has a hardware 12-key keyboard, whether it's visible to the user or not.
17	Navigation key availability	Possible values include navexposed (navigation keys are available to the user) and navhidden (navigation keys are not available, for example they're behind a closed lid).
18	Primary non-touch navigation method	**Possible values** **nonav:** Device has no navigation facility other than the touchscreen. **dpad:** Device has a directional-pad (D-pad). **trackball:** Device has a trackball. **wheel:** Device has a directional wheel for navigation (uncommon).
19	Platform version (API level)	The API level supported by the device. Described in Platform version, below.

3.5.4 Creating Alternative Resources

To create alternative resource folders most easily in Android Studio, use the "Android" view in the Project tool window.

1. Selecting the "Android" view in Android Studio. If you don't see these options, make sure the Project tool window is visible by selecting **View > Tool Windows > Project**.
2. To use Android Studio to create a new configuration-specific alternative resource folder in res/:
3. Be sure you are using the "Android" view, as shown above.

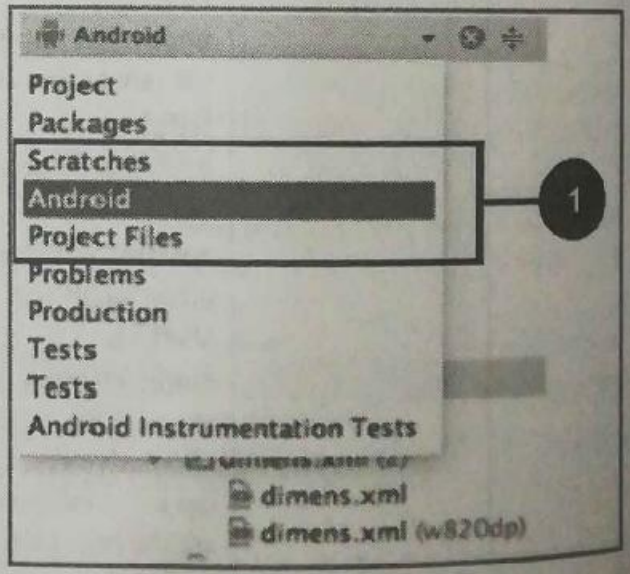


Fig. 3.5.4

1. Right-click on the res/ folder and select **New > Android resource directory**. The New Resource Directory dialog box appears.
2. Select the type of resource and the qualifiers that apply to this set of alternative resources.
3. Click OK.

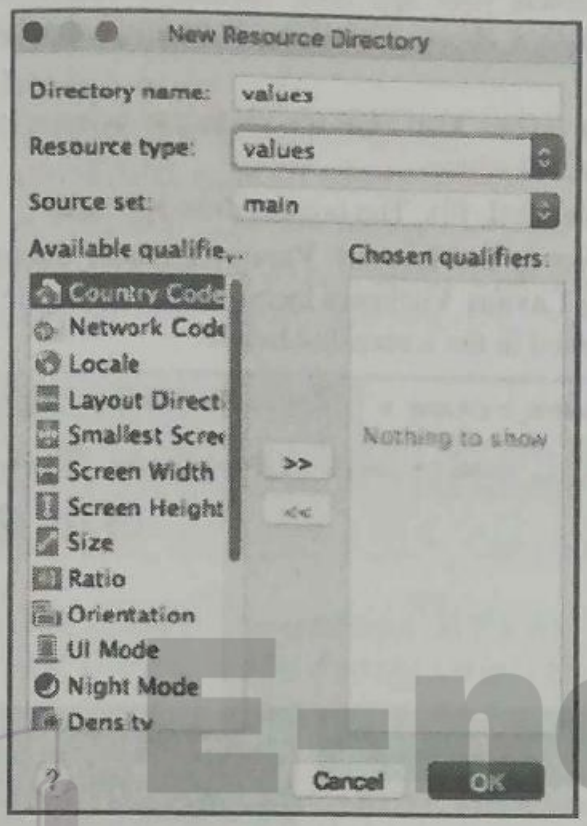


Fig. 3.5.5

If you can't see the new folder in the Project tool window in Android Studio, switch to the "Project" view, as shown in the screenshot below. If you don't see these options, make sure the

Project tool window is visible by selecting **View > Tool Windows > Project**.

Save alternative resources in the new folder. The alternative resource files must be named exactly the same as the default resource files, for example "styles.xml" or "dimens.xml".

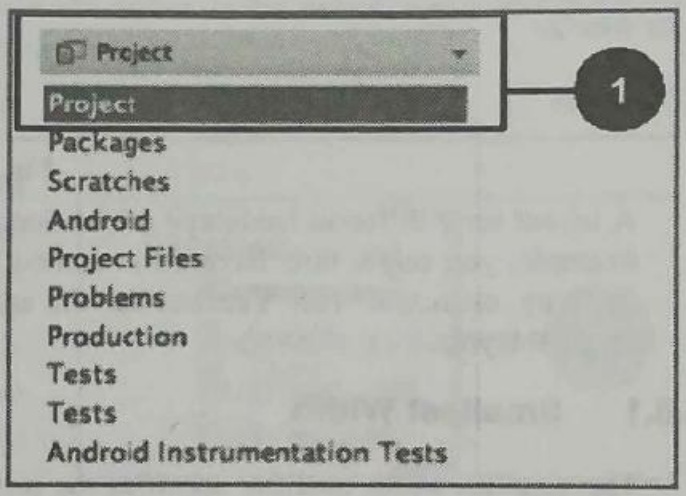


Fig. 3.5.6

3.6 Screen Orientation

The screen-orientation qualifier has two possible values:

Port: The device is in portrait mode (vertical). For example, res/layout-port/ would contain layout files to use when the device is in portrait mode.

- **Land:** The device is in landscape mode (horizontal). For example, res/layout-land/ would contain layout files to use when the device is in landscape mode.
- If the user rotates the screen while your app is running, and if alternative resources are available, Android automatically reloads your app with alternative resources that match the new device configuration. For information about controlling how your app behaves during a configuration change,
- To create variants of your layout XML file for landscape orientation and larger displays, use the layout editor. To use the layout editor:

1. In Android Studio, open the XML file. The layout editor appears.
2. From the drop-down menu in the **Layout Variants** menu, choose an option such as **Create Landscape Variant**. The **Layout Variants** menu, which is visible when an XML file is open in Android Studio, is highlighted in the screenshot below.

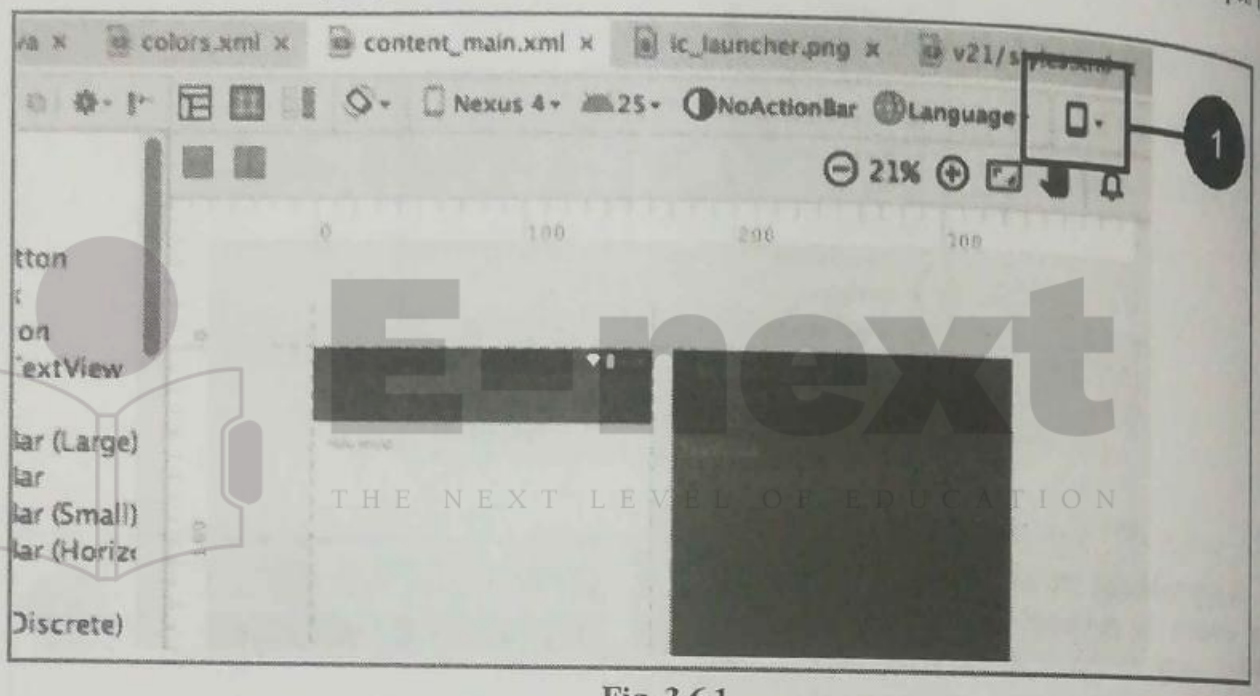


Fig. 3.6.1

- A layout for a different landscape orientation appears, and a new XML file is created for you. For example, you might now have a file named "activity_main.xml (land)" along with your original "activity_main.xml" file. You can use the editor to change the new layout without changing the original layout.

3.6.1 Smallest Width

- The smallest-width qualifier specifies the minimum width of the device. It is the shortest of the screen's available height and width, the "smallest possible width" for the screen. The smallest width is a fixed screen-size characteristic of the device, and it does not change when the screen's orientation changes.
- Specify smallest width in dp units, using the following format:

```
sw<N>dp
```

where <N> is the minimum width. For example, resources in a file named res/values-sw320dp/styles.xml are used if the device's screen is always at least 320dp wide.

- You can use this qualifier to ensure that a certain layout won't be used unless it has at least <N> dps of width available to it, regardless of the screen's current orientation.

Some values for common screen sizes

- 320, for devices with screen configurations such as
- 240 x 320 ldpi (QVGA handset)
- 320 x 480 mdpi (handset)
- 480x800 hdpi (high-density handset)
- 480, for screens such as 480x800 mdpi (tablet/handset)
- 600, for screens such as 600x1024 mdpi (7" tablet)
- 720, for screens such as 720x1280 mdpi (10" tablet)

- When your application provides multiple resource folders with different values for the smallest-width qualifier, the system uses the one closest to (without exceeding) the device's smallest width.

Example

res/values-sw600dp/dimens.xml contains dimensions for images. When the app runs on a device with a smallest width of 600dp or higher (such as a tablet), Android uses the images in this folder.

3.6.2 Platform Version

- The platform-version qualifier specifies the minimum API level supported by the device. For example, use v11 for API level 11 (devices with Android 3.0 or higher).
- Use the platform-version qualifier when you use resources for functionality that's unavailable in prior versions of Android.
- Put default versions of the images in a res/drawable folder. These images must use an image format that's supported for all API levels, for example JPEG.
- Put WebP versions of the images in a res/drawable-v17 folder. If the device uses API level 17 or greater, Android will select these resources at runtime.

3.6.3 Providing Default Resources

- *Default resources* specify the default design and content for your application. For example, when the app runs in a locale for which you have not provided locale-specific text, Android loads the default strings from res/values/strings.xml. If this default file is absent, or if it is missing even one string that your application needs, then your app doesn't run and shows an error.

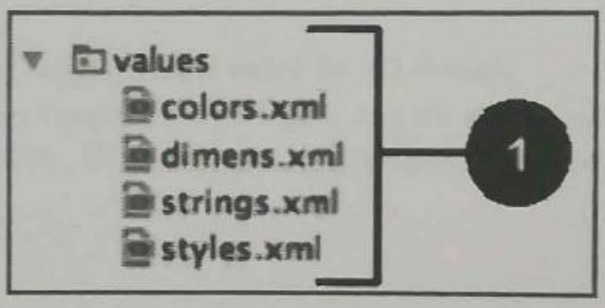


Fig. 3.6.2

- Default resources have standard resource folder names (values, for example) without any qualifiers in the folder name or in parentheses after the file names.

Default resources

> **Note:** Always provide default resources, because your app might run on a device configuration that you don't anticipate.

- Sometimes new versions of Android add configuration qualifiers that older versions don't support.

- If you use a new resource qualifier and maintain code compatibility with older versions of Android, then when an older version of Android runs your app, the app crashes unless default resources are available. This is because the older version of Android can't use the alternative resources that are named with the new qualifier.
- When an API level 4 device runs the app, the device can't access your drawable resources. The Android version doesn't know about night and notnight, because these qualifiers weren't added until API level 8. The app crashes, because it doesn't include any default resources to fall back on.

Review Questions

Q. 1 Explain how to define and apply style ? **(Refer section 3.2)**

Q. 2 Explain how to define and apply themes ? **(Refer section 3.3.1)**

Q. 3 Describe font styles. **(Refer section 3.4.1)**

Q. 4 Explain snackbar in detail. **(Refer section 3.4.4)**

Q. 5 Write short note on Externalizing resources. **(Refer section 3.5.1)**

Q. 6 How to create alternative resources in android ? **(Refer section 3.5.4)**

□□□